Alma Mater Studiorum – Università di Bologna

Dottorato di Ricerca in
Computer Science and Engineering
XXXI Ciclo

Coinductive Equivalences and Metrics for Higher-order Languages with Algebraic Effects

Presentata da
Francesco Gavazzo

Supervisore
Prof. Ugo Dal Lago

Coordinatore Dottorato
prof. Paolo Ciaccia

Settore Concorsuale
01/B1 - Informatica

Settore Scientifico Disciplinare
INF-01 - Informatica

Esame finale anno 2019

To my wife, Marta

Abstract

This dissertation investigates notions of program equivalence and metric for higher-order sequential languages with algebraic effects.

Computational effects are those aspects of computation that involve forms of interaction with the environment. Due to such an interactive behaviour, reasoning about effectful programs is well-known to be hard. This is especially true for higher-order effectful languages, where programs can be passed as input to, and returned as output by other programs, as well as perform side-effects. Additionally, when dealing with effectful languages, program equivalence is oftentimes too coarse, not allowing, for instance, to quantify the observable differences between programs. A natural way to overcome this problem is to refine the notion of a program equivalence into the one of a program distance or program metric, this way allowing for a finer, quantitative analysis of program behaviour. A proper account of program distance, however, requires a more sophisticated theory than program equivalence, both conceptually and mathematically. This often makes the study of program distance way more difficult than the corresponding study of program equivalence.

Algebraic effects provide a powerful formalism to structure effectful higher-order (sequential) computations. Accordingly, effectful computations are produced by means of effect-triggering operations which act as sources of the side effects of interest. Such operations are algebraic, in the sense that their (operational) semantics is independent of their continuation, and thus effects act independently of the evaluation context in which they are executed. Algebraic effects can be used to model several computational effects, proving formal models for higher-order languages with nondeterministic, probabilistic, and imperative features, as well as combinations thereof. In fact, contrary to other theories of computational effects, algebraic effects naturally support operations to combine algebraic theories, and thus allow for the combination of effects with one another. These features make reasoning about program equivalence for languages with algebraic effects challenging, as the operational behaviour of a program may be determined by complex interactions between the program and the environment.

The first part of this dissertation studies bisimulation-based notions of equivalence and refinement for λ-calculi enriched with algebraic effects. In particular, notions of effectful applicative and normal form (bi)similarity are defined for both call-by-name and call-by-value λ-calculi, as well as a notion of monadic applicative (bi)similarity for call-by-name calculi only. For all these notions, congruence and precongruence theorems are proved, which directly lead to soundness results with respect to an extension of Morris' contextual equivalence to effectful calculi. In order to design the aforementioned notions of equivalence and refinement, an abstract relational framework is developed, which is based on the notions of a monad and of a relator, the latter being an abstract construction axiomatising relation lifting operations.

The second part of this dissertation is devoted to the study of program distances for languages with algebraic effects. Following Lawvere analysis of metric spaces as enriched categories, the abstract relational framework developed in the first part of the dissertation, is extended to relations taking values over quantales, the latter being algebraic structures whose elements represent 'abstract quantities'. Using such a framework, the notion of an effectful applicative bisimulation distance is defined, and its

main properties are studied. Due to the so-called distance trivialisation, however, the notion of an effectful applicative bisimulation distance is defined for a calculus supporting program sensitivity. The latter is the law describing how much operational differences in output are affected by operational differences in input. Relying on the notion of program sensitivity, an abstract (pre)congruence theorem for effectful applicative (bi)similarity distance is proved, which states that the latter behaves as a Lipschitz-continuous metric-like function, this way enabling compositional reasoning about program distance.

Acknowledgements

I am very grateful to my supervisor, Ugo Dal Lago. None of the results of this dissertation would have been obtained without his patient supervision and his careful guidance. Ugo has been always available to discuss my work and my private life, in both cases giving me constant support as well as suggestions of the highest quality. He always supported me and encouraged me to do some independent research, this way allowing me to grow as a researcher. From him I learnt how to do research. I simply cannot express my gratitude for that.

A special thank goes to Davide Sangiorgi, who has been a second supervisor. With him I enjoyed several discussions on the history and foundation of coinductive techniques. His approach to the subject deeply influenced my way of doing research.

I am also grateful to Lars Birkedal and Michele Pagani for having reviewed this dissertation in the short amount of time they have at their disposal.

The results presented in this dissertation have been discussed with (and improved by) many people in several occasions. Special thanks go to Paul Blain Levy, who introduced me to the mathematics I needed to develop my theories. I am also very grateful to Alex Simpson and Niels Voorneveld, with whom I discussed ideas on program metric during the (wonderful) four months I spent in Ljubljana.

A spacial thank goes to my family. First of all to my wife Marta: nothing of what I did would have been possible without her. My parents Chiara and Giuseppe have always supported me and encouraged me to pursue my goals. I would never thank them enough for that. I am also very grateful to my brother Federico and to my grandfather Gastone. The latter has always been a source of inspiration for me. Writing this dissertation, his reminder of the proverb *il meglio è nemico del bene* (perfect is the enemy of good) has been extremely valuable.

In the last three years I shared wonderful moments with colleagues and friends. Thank you all. Among those, special thanks go to Valeria, the two Vincenzos, Saverio, Adele, the two Stefanos, Angelo, Federico, the two Lucas, Flavio, Daniel, and Tong.

I am also grateful to all professors and researchers who contributed to my scientific and philosophical formation. Among those, special thanks go to Antonio Maria Nunziante for having taught me Leibniz's philosophy, to Gilberto Filè for having introduced me to program correctness and formal methods, to Alessandra Palmigiano and Giuseppe Greco for the passion they put in teaching me algebra and proof theory, and to Henk Barendregt for our 'cappuccino meetings' solving exercises from his λ-calculus book.

Contents

Chapter 1

Introduction

> This is why we call it a "calculus". We dare to use this word by analogy with Leibniz's differential calculus; the latter - incomparably greater - is based upon continuous mathematics, while the π-calculus is based upon algebra and logic; but the goal in each case is analysis, in one case of physical systems and in the other case of informatic systems.
>
> Speech by Robin Milner on receiving an Honorary Degree from the University of Bologna

Program equivalence is arguably one of the most important concepts in the theory of programming languages. Besides its prime importance in the general understanding of programming languages, program equivalence is also recognised as a fundamental tool in fields like program verification, compiler optimisation design, and security. Unfortunately, giving satisfactory definitions and methodologies for program equivalence is a challenging problem. In fact, in order to deem two programs as equivalent, several aspects of their behaviour should be taken into account. Among those, one should consider whether the two programs both terminate, what is their input-output behaviour, how the programs interact with the environment, and which kind of side-effects (if any) are produced during their evaluation.

As a consequence, reasoning about program equivalence is particularly hard when dealing with *higher-order effectful languages* i.e. languages in which programs can be passed as input to, and returned as output by other programs, as well as perform side-effects. A remarkable example is provided by programming languages based on algebraic effects (G. D. Plotkin & Power, 2002, 2003) — such as *Eff*[1] (Bauer & Pretnar, 2015) — as well as extensions of mainstream languages — such as OCaml, Scala, and C — with algebraic effects (Brachthäuser & Schuster, 2017; Dolan et al., 2017; Leijen, 2017).

Programs written in such languages usually employ specific operations to produce effectful computations, such as probabilistic choices or primitives to raise exceptions. As a consequence, the result of the evaluation of a program usually depends on the environment in which the program is evaluated, hence making reasoning about effectful programs notoriously hard.

A further difficulty is provided by the possibility of combining effects with one another. In fact, real

[1] https://www.eff-lang.org/.

world programs are often designed to perform combinations of different effects — using e.g. probabilistic, nondeterministic, and imperative primitives — and are thus complex entities. Reasoning about such programs without an appropriate formal apparatus is simply not possible. For instance, even answering a simple question like *what does this program do?* might require non-trivial forms of reasoning about the interaction between the program and the environment. This cannot be safely done without a toolkit of adequate formal techniques.

A less explored, yet increasingly important, concept than program equivalence, is the one of a *program metric* or *program distance*, i.e. of a metric-like function defining a distance between programs according to their behaviour. The notion of a program metric is emerging as a fundamental tool in the study of languages for probabilistic computation (Crubillé & Dal Lago, 2015, 2017), differential privacy (de Amorim, Gaboardi, Hsu, Katsumata, & Cherigui, 2017; Reed & Pierce, 2010; Xu, Chatzikokolakis, & Lin, 2014), and effectful languages, more generally. In all these scenarios, program equivalence is a too coarse notion for reasoning about program behaviour, as it does not allow to quantify the observable, operational differences between programs. For instance, in probabilistic languages a small perturbation in the probabilistic behaviour of programs may break program equivalence without proving any further information.

To solve these issues, programming language theorists are increasingly interested in studying notions of program distance, this way allowing for a quantitative (and thus finer) analysis of program behaviour. A proper account of program metrics, however, requires a more sophisticated theory than program equivalence, both conceptually and mathematically. This often makes the design and study of program distances way more difficult than the corresponding study of program equivalences.

In the first part of this dissertation we aim to answer a simple, yet non-trivial question, namely when two effectful programs should be deemed as (behaviourally) equivalent. In the second part, instead, we tackle a different problem, namely the one of quantifying behavioural differences between programs.

Before introducing the results obtained, we informally outline the main ideas behind the models we use, and the notions we investigate in this work.

1.1 Program Equivalence: a Conceptual Introduction

A programming language is an artificial formalism in which algorithms can be expressed. For all its artificiality, though, this formalism remains a *language*[2], and can thus be analysed using notions and results from linguistics. However, looking at algorithms as specific mathematical functions (notably, the computable ones), programming languages can also be seen as playing a role similar to algebra, as they aim to control and manipulate complexity throughout linguistic abstraction. In this introduction we will take advantage of both these views on programming languages, referring to them as the *linguistic* and *algebraic* point of view, respectively.

Looking at a programming language as a language, we see that a program is ultimately a phrase in such a language, and it is therefore natural to ask what its meaning is. The branch of programming language theory that studies how to answer this question is called *programming language semantics*. One of the most interesting question in program semantics is the one asking when two programs are equivalent. Formally, answers to such a question are provided by relations between program phrases that behave as notions of equality. According to the algebraic perspective, program equivalence is nothing more than equality; according to the linguistic perspective, program equivalence can be understood as a synonymy.

According to both these views, program equivalence is of paramount importance for the formal study of programming languages. Algebraically, a good notion of program equivalence allows to design algebraic identities through which we can manipulate programs *symbolically*. Linguistically, a

[2]Quoted from (Gabbrielli & Martini, 2010).

well-behaved program equivalence $\simeq$ allows to regard the meaning of a program phrase e as the $\simeq$-equivalence class of e (denoted by $[e]_{\simeq}$). This is nothing more than stating that the meaning of an expression is the meaning of its synonyms.

Let us now try to spell out which are the minimal features that qualify a relation between program phrases as a candidate notion of program equivalence. First of all, we notice that some of such features will actually depend on the specific language one is interested in. We will come back on that later. Nonetheless, there are some structural properties that any notion of program equivalence has to satisfy. Let $\simeq$ be a candidate relation.

1. In order to behave as an equality relation, $\simeq$ needs to be an equivalence relation, i.e. it must be reflexive, symmetric, and transitive. Algebraically, such properties allow to deduce the equivalence $e \simeq e'$ from a chain of equivalences $e \simeq e_1 \simeq \cdots \simeq e_n \simeq e'$. Linguistically, $\simeq$ being an equivalence relation we can state that the meaning of an expression e is the set $[e]_{\simeq}$.

2. Additionally, $\simeq$ has to be *compatible* with all language constructors. That is, $\simeq$ must be closed under all language (syntactical) constructors. For instance, if our programming language has an 'if-then-else' constructor, and $e_1 \simeq e_1'$, $e_2 \simeq e_2'$, $e_3 \simeq e_3'$, then we also need to have **if** e_1 **then** e_2 **else** $e_3 \simeq$ **if** e_1' **then** e_2' **else** e_3'. Algebraically, thinking to language constructors as operations, we see that compatibility makes $\simeq$ a *congruence* relation. This gives the powerful reasoning principle of *substitution of equals for equals* (also known as *Leibniz's identity law*). Accordingly, we can infer the equivalence $C[e] \simeq C[e']$ between the compound phrases $C[e]$ and $C[e']$ in virtue of the equivalence $e \simeq e'$. Linguistically, requiring $\simeq$ to be a congruence gives the so-called *principle of compositionality*[3]. Indeed, the meaning $[C[e]]_{\simeq}$ of a compound phrase $C[e]$ can be seen as a function of the meaning $[e]_{\simeq}$ of its sub-phrase e. It comes with no surprise that a large part of this dissertation will be devoted in proving congruence properties of program equivalences.

Having outlined some minimal desiderata any candidate notion of program equivalence has to satisfy, we now look at the specific kind of equivalences we aim to study in this dissertation.

1.1.1 Higher-order Languages

According to our previous discussion, it comes with no surprise that answering the question of whether two programs are equivalent is non-trivial. First of all, we should ask ourselves whether the latter is a well-posed question. In fact, there is no unique notion of a program: different programming languages might come with different notions of a program. It is thus of paramount importance for our investigation to fix a good model for the languages we aim to analyse.

In this dissertation we focus on *higher-order functional languages*, that is languages in which programs are modelled as functions, and the latter are allowed to be passed as argument to, and returned as result by other functions. A main feature of functional languages is that they have a solid formal foundation, which allows for a fine mathematical analysis of their properties. In fact, starting from the pioneering work by Landin (Landin, 1965a, 1965b) (see also (G. Plotkin, 1975)), the λ-calculus (Barendregt, 1984; Church, 1985) has been recognised as a foundational calculus for functional programming languages.

The λ-calculus

The λ-calculus (Barendregt, 1984; Church, 1985) is a formalism to study functions as *rules* (as opposed to the set-theoretic understanding of functions as *graphs*), i.e. as processes mapping arguments to values (Barendregt, 1984). The syntax of the λ-calculus is minimal: a term of the λ-calculus is either a variable

[3] Whereby the meaning of a complex expression is determined by its structure and the meanings of its constituents.

x, an application of terms fe, or a functional abstraction $\lambda x.f$. We call terms of the λ-calculus *λ-terms*. The intended meaning of a λ-abstraction $\lambda x.f$ is to represent a function in the variable x with body f. The latter describes how the variable x (which acts as a placeholder for an argument) is manipulated by the function. Dually, an application fe represents the application of the function f to the argument e. The only operation allowed to manipulate λ-terms is the so-called β-rule, which rewrites a term of the form $(\lambda x.f)e$ as $f[x := e]$. The β-rule simply states that once a function $\lambda x.f$ is applied to an argument e, the λ-term $(\lambda x.f)e$ thus obtained is rewritten as the λ-term obtained by replacing all occurrences of the variable x in f (the body of the function) with e.

From this sketched description of the λ-calculus, we immediately recognise that all λ-terms indeed represent and manipulate functions, so that the λ-calculus may serve as a linguistic abstraction for studying functions. As showed in (Turing, 1937), the class of functions expressible in the λ-calculus is exactly the class of computable functions, hence hinting the relationship between the λ-calculus and the idea of a programming language as a language to express and manipulate algorithms.

However, the link between the λ-calculus and programming languages still needs some refinements. In fact, since a λ-term may contain several occurrences of terms of the form $(\lambda x.f)e$, (called redexes), the β-rule can be applied to different redexes of the same λ-term, hence giving potentially different results. Such a nondeterminism can be eliminated by fixing a so-called reduction strategy (Barendregt, 1984). The latter is simply a rule that determines to which redex the β-rule should be applied in a λ-term. In this dissertation we are concerned with two different reduction strategies, the *call-by-name* and *call-by-value* reduction strategy (G. Plotkin, 1975).

Roughly speaking, in the call-by-name reduction strategy, when a function $\lambda x.f$ is applied to an argument e, the redex $(\lambda x.f)e$ is reduced as it is, thus giving $f[x := e]$. In the call-by-value reduction strategy, instead, a redex is rewritten only if the argument of the function is a *value*, i.e. a λ-term on which the β-rule cannot be applied any further.

Once a reduction strategy is fixed, the λ-calculus can be regarded as a foundational calculus for functional programming languages, as showed by the pioneering work by Landin (Landin, 1965a, 1965b) who established a correspondence between expressions of ALGOL 60[4] (Backus et al., 1960) and (a modified version of) the λ-calculus. Accordingly, λ-terms can be roughly identified with programs, and evaluating a program corresponds to apply the β-rule to the λ-term representing the program until a value is reached. Notice that since we have fixed a reduction strategy, program evaluation is *deterministic*.

Actually, even when equipped with a reduction strategy the λ-calculus is still lacking an important feature of (mainstream) functional programming languages. In fact, almost no functional programming language evaluates functions until they are called. Using the vocabulary of the λ-calculus, that means that the β-rule cannot be applied on expressions (redexes) under the scope of a λ-abstraction[5]. As a consequence, any λ-abstraction $\lambda x.e$ is regarded as a value. The λ-calculus obtained by regarding λ-abstractions as values (independently of the reduction strategy adopted) is called the *lazy* λ-calculus (Abramsky, 1990a). Our interest being in programming languages, from now, unless explicitly mentioned, we assume the λ-calculus to be the lazy one, and simply refer to it as λ-calculus.

The λ-calculus can be thus regarded as a foundational calculus for functional programming languages, and thus used as formal tool for their theoretical analysis. On one hand, the λ-calculus is expressive enough to allow for the description of interesting features of several languages, on the other hand, it abstracts from those concrete details of real languages that would make the analysis too dependent on the language chosen.

At this point it comes with no surprise that the theoretical model we use for our investigation is

[4]Notice, however, that ALGOL 60 is a sequential procedural language, rather than a functional one.

[5] Consider, for instance, the following Standard ML piece of code (Milner, Tofte, & Harper, 1990), to which we refer to as P: **fn** x=>((**fn** y=>y)x). Regarded as a λ-term, the latter can rewritten as $\lambda x.((\lambda y.y)x)$. According to our definition of reduction strategy (say the call-by-value one), an application of the β-rule gives $\lambda x.x$. However, the evaluation mechanism of Standard ML treats P as value, and does not evaluate (**fn** y=>y)x.

based on the λ-calculus.

1.1.2 Three Views on Program Equivalence

In light of previous discussion, in order to study program equivalence it is somehow sufficient to study equivalences between terms of the λ-calculus. That is, we can rephrase the question *when two programs are equivalent?* as *when two terms of the λ-calculus are equivalent?*. Answering such a question means nothing more than finding good notions of equivalence for the λ-calculus. Here we give a lightweight overview of the main ideas behind (some of) the equivalences we will study in this dissertation. In light of the correspondence between programs and λ-terms, we will sometimes use the latter expressions interchangeably.

Before continuing, we remark that although in this introduction we deal with program equivalence only, this dissertation focuses both on program equivalence and program refinement, the latter being nothing but the order-theoretic counterpart of program equivalence[6].

The kind of equivalences we will study in this work are the so-called *operationally-based* equivalences, that is equivalences that aim to identify programs exhibiting the same *operational behaviour*. The operational behaviour of terms in the λ-calculus is essentially given by the β-rule (properly adapted to the reduction strategy considered), and thus our notions of equivalence should be equate λ-terms according to how they behave when evaluated. For the sake of the argument, we fix the call-by-value reduction strategy.

Does the Best Equivalence Exist?

Let us recall that any good notion of program equivalence must be a compatible equivalence relation. Additionally, we remarked that such a relation should distinguish λ-terms for their operational behaviour only. As a first approximation, the latter requirement might be formalised by stating that if two terms e and e' are related, then e evaluates to a given value v if and only if e' does the same. The choice of the value we look at does not really matter.

Such a choice, however, does not work well, neither for the call-by-name nor for the call-by-value λ-calculus. For instance, the two λ-terms $\lambda x.I$ and $\lambda x.II$, where I is the λ-term $\lambda y.y$, should be deemed as operationally equivalent, intuitively. However, being values they trivially evaluate to different values, and thus we should regard them as operationally different. The standard solution to overcome this problem is relaxing the above requirement asking that e converges (i.e. the evaluation of e terminates) if and only if e' does. That is, instead of comparing λ-terms for converge to the same value, we compare them convergence only.

It is then natural to ask: *does a compatible equivalence relation that compare programs for convergence exist?*. The answer is obviously positive, as syntactic equality is such a relation. This is clearly an unsatisfactory answer. What we would like to come up with is not just *a* compatible equivalence relation that compare programs for convergence, but actually the *coarsest* one. That is, what we are looking for is a compatible equivalence relation that *discriminate* programs *only* for their convergence behaviour. Nothing more, nothing less. Mathematically, we are asking whether there exists the *largest* compatible equivalence relation that compares programs for their convergence behaviour.

In his doctoral dissertation (Morris, 1969) Morris gave a positive answer to such a question, introducing the so-called *contextual equivalence* (also known as *operational* or *observational* equivalence). Due to its universal property[7], contextual equivalence is considered the golden notion of equivalence

[6] That is, a program refinement is a relation between program phrases relating pair programs e, e' according to the rationale the behaviour of e is refined (or approximates) the behaviour of e'. Mathematically, moving from program equivalence to program refinement means moving from equivalences to preorders, and from congruence to precongruence relations.

[7] Contextual equivalence being the *largest* relation satisfying a desired property P (notably, being a compatible equivalence relation equating λ-terms for their convergence behaviour), it can be seen as a *canonical* relation satisfying P. Its universal

for the λ-calculus. Moreover, contextual equivalence can be explicitly characterised by saying that two λ-terms e, e' are contextually equivalent if and only if for all compound expressions $C[e]$, $C[e']$, the former converges if and only if the latter does, this way highlighting how compatibility is an essential part of its definition[8].

Unfortunately, contextual equivalence has a major drawback: in order to deem two λ-terms e, e' as contextually equivalent, one has to analyse their behaviour in *any* possible compound expression $C[e]$, $C[e']$, which is something oftentimes not doable in practice, due to the higher-order nature of the λ-calculus. For that reason, semanticists are continuously interested in designing better proof techniques for contextual equivalence. The proof techniques that are relevant for this dissertation take the form of relations between λ-terms that satisfy the following minimal desiderata:

1. They must be easier to use than contextual equivalence.
2. They must be *sound* for contextual equivalence, meaning that, as relations, they must be included in contextual equivalence. If the other inclusion holds as well, the relation is said to *fully abstract* (or *complete*) for contextual equivalence.

If, additionally, such relations are congruence relations, then they can also become interesting notions of program equivalence by themselves (i.e. independently of their relationship with contextual equivalence). Among such techniques we mention *logical relations* (G. Plotkin, 1973; Reynolds, 1983), CIU-like equivalences (Mason & Talcott, 1991), applicative bisimilarity (Abramsky, 1990a), and Böhm-tree like equivalences (Barendregt, 1984; Böhm, 1968).

Interactions, Tests, and Experiments

Taking inspiration from ideas developed in process calculi (e.g. (Milner, 1989; Sangiorgi & Walker, 2001)), it is possible to look at notions of equivalence for the λ-calculus from a concurrent perspective, as well as to design new equivalences with a more 'interactive' flavor. Accordingly, one thinks to program equivalence by means of *experiments* and *observations*, following the rationale that two λ-terms should be deemed as equivalent if there is no experiment (testing λ-terms for their operational behaviour) that can tell them apart.

This way, program equivalence is defined by means of an interactive process in which (pairs of) programs (i.e. λ-terms) interact with an external observer that tries to discriminate them using a given apparatus of tests, experiments, and observations. The largest such an apparatus, the stronger the discriminating power of the observer.

As we are interested in operationally-based equivalences, the main notion of observation we will use is simply convergence, meaning that the outcome of an experiment will be a 'yes/no' answer telling whether the tested program(s) converge or not. Moreover, dealing with higher-order languages (notably, the λ-calculus), the role of the *experiment* is oftentimes — but not always — played by specific families of programs, that represent the environment in which programs are evaluated. Such programs are called

property states exactly that: any relation satisfying P is included in contextual equivalence.

[8] Most textbooks take this characterisation of contextual equivalence as a definition, and then prove its universal property. Moreover, we should notice that Morris's notion of contextual equivalence is actually different from the one used nowadays for the call-by-name and call-by-value λ-calculus, as it is based on the idea of comparing terms for convergence to the same value (normal form, actually), and not just for convergence. Morris' notion of contextual equivalence, however, is formulated for the λ-calculus equipped with the so-called *head* reduction strategy (Barendregt, 1984). Without entering details, we simply remark that such a strategy allows us to reduce under λ-abstractions, and therefore does not regard $\lambda x.II$ as a value (notice that II is a redex). That allows us to prove that in such a λ-calculus, the notion of contextual equivalence obtained comparing terms for convergence and the notion of contextual equivalence obtained comparing terms for convergence to the same value coincide. The proof of such a result is, however, non-trivial as it requires the so-called *Separation Theorem* (Böhm, 1968) (a result on which we will come back later).

contexts, and are roughly formalised as programs with a hole $[-]$ to be filled in with the tested program. This is nothing but the structure we have used so far to build compound expressions.

The larger is the family of contexts we use to test programs, the more testing power we have at our disposal. In particular, if we test programs against any possible context, then we obtain Morris' contextual equivalence. We thus notice how contextual equivalence captures the idea of operational indistinguishability in a black-box scenario.

This perspective on contextual equivalence is essentially based on the 'testing' approach to behavioural equivalence (De Nicola & Hennessy, 1983), whereby the behaviour of a program is investigated by a series of tests. There is another fundamental notion of program equivalence which is deeply rooted in ideas coming from Concurrency Theory, namely *applicative bisimilarity* (Abramsky, 1990a). A major contribution of Concurrency Theory has been the introduction of the notion of a *bisimulation* and of bisimulation equality, called *bisimilarity* (Milner, 1989). Bisimilarity is a *coinductively*-defined (Park, 1981; Sangiorgi, 2011) notion of equivalence, which is arguably the most studied form of behavioural equality for processes.

The theory of bisimulation has found an important application in the λ-calculus starting with the work by Abramsky (Abramsky, 1990a), who introduced the notion of an *applicative bisimulation*. Abramsky built on the assumption that λ-terms ultimately represent (kind of) functions, notably λ-abstractions, and should be thus tested according to the *function extensionality principle*. The latter states that two functions $F, G : A \to B$ are equal if for any $a \in A$, $F(a) = G(a)$. The fundamental assumption behind this definition is that there is a notion of equality for elements belonging to the set B[9]. Applying extensionality to λ-terms, however, gives problems of well-foundedness: deeming two λ-terms f_1, f_2 to be equal if and only if for any value v, $f_1 v$ is equal to $f_2 v$, we see that we are using the relation we are trying to define as a part of its very definition.

Abramsky bypassed this problem imagining the following testing scenario. Given a λ-term e, e is first evaluated, hence testing it for convergence. If it passes the test, then obviously e converged to a value v. At this point we have the real interaction phase between the environment and v. The environment cannot inspect v, nor it can use it as desired. The only possible interaction it can have with v is by passing a value w to v, hence resuming the whole testing process on the term vw.

Two terms are applicative bisimilar if they cannot be discriminated by the above testing process. Obviously, testing λ-terms in this way may take an infinite amount of time, and it thus comes with no surprise that applicative bisimilarity is defined *coinductively*.

Comparing λ-terms following Abramsky's testing scenario, it is natural to ask how much testing power we are loosing compared to contextual equivalence. In fact, applicative bisimilarity tests programs for their applicative behaviour only, hence drastically reducing the testing power given by arbitrary contexts, at least apparently. This is actually (part of) the strength of applicative bisimilarity, which thus qualifies as a good candidate proof technique for contextual equivalence. Abramsky showed that not only applicative bisimilarity works fine as a proof technique for contextual equivalence (soundness), but it also provides an alternative characterisation of the latter (full abstraction).

Looking at the Past Throughout the Concurrency Glasses

Recalling that we are interesting in operational equivalences, and that the operational behaviour of a λ-term is essentially determined by the β-rule, there is candidate notion of equivalence that we have not investigated, which is actually the most straightforward one: two λ-terms are equivalent if and only if they evaluate to the same value.

This equivalence is rooted in Church's original idea that the meaning of a λ-term is the value it evaluates to, and is strongly related to the notion of β-equality (or β-convertibility) (Barendregt, 1984).

[9] This is indeed the case in Set Theory, where everything is a set (sets are essentially built starting from the empty set using the axioms of e.g. Zermelo and Frankel (Jech, 1997)) and set-theoretic equality is defined by means of the axiom of extensionality.

Broadly speaking, the latter is the notion of equality arising from the β-rule[10]. The major drawback of Church's proposal is that not all λ-terms converge[11], and thus all diverging terms should be regarded as equal (or, from a linguistic perspective, as meaningless). Unfortunately, the theory induced by such a notion of equality has been proved to be inconsistent (see (Barendregt, 1984) for a nice exposition of the subject). Historically, this problem has been solved by looking at a more informative notion of a value, which can be elegantly described via the notion of a *Böhm tree* of a λ-term, giving raise to the so-called *standard theory* (Barendregt, 1984).

Roughly speaking, the Böhm tree of a λ-term e is a kind of infinitary value[12] representing all the stable amount of information obtained by evaluating e. As a consequence, all λ-terms have a Böhm tree, and we can thus regard two λ-terms as equivalent if and only if they have the same Böhm trees. Actually, the Böhm tree of a λ-term e is obtained by evaluating e using neither the call-by-name nor the call-by-value reduction strategy, but the so-called head reduction strategy (Barendregt, 1984). We will be sloppy on that for the moment, and simply recall that Böhm tree-like structures can be defined for both the call-by-name (giving the notion of a *Lévy-Longo tree* (Lévy, 1975; Longo, 1983)) and the call-by-value (see e.g. (Carraro & Guerrieri, 2014; S. B. Lassen, 2005)) λ-calculus.

At this point of the story it is then natural to ask whether there is a link between the above 'tree-like' equivalences, and the interactive approach to program equivalence of previous section. This is indeed the case, as shown in (Sangiorgi, 1992, 1994). Moving from the theory of *open bisimulation* for the π-calculus (Sangiorgi, 1993) and from encodings of the λ-calculus into the π-calculus (Milner, 1992), Sangiorgi modified Abramsky's testing scenario as follows: instead of testing a value (which, for the sake of the argument, we assume to be a λ-abstraction) $\lambda x.e$ by passing it a value v as argument, the environment can now inspect the body of the function, i.e. the term e under the lambda[13].

Compared to Abramsky's idea of testing λ-terms extensionally, open bisimulation can be seen as an *intensional* notion of equivalence, whereby functions are not tested for their input-output behaviour, but for their intensional (syntactical, to some extent) structure. The tree-like structures associated to λ-terms can be now seen as unfolded representation of this new testing process, and their equality thus coincides with *open bisimilarity*, the latter being open bisimulation equality. This result had several implications, as it provided (to the best of the author's knowledge) the first *coinductive* account to tree-like equivalences, whose theory was essentially induction-based at the time (see e.g. (Barendregt, 1984)).

The results proved in (Sangiorgi, 1992, 1994) concerned the call-by-name λ-calculus and the associated notion of Lévy-Longo tree equality. Such results have been then extended to Böhm trees (S. B. Lassen, 1999) and to the call-by-value λ-calculus (S. B. Lassen, 2005), introducing the general notion of a *normal form bisimulation*[14].

Open bisimilarity provides a powerful candidate proof technique for contextual equivalence, as λ-terms are essentially tested in isolation (the environment can interact with them inspecting their intensional structure only, and does not have the power to influence computations[15]), meaning that in order to prove two λ-terms to be open bisimilar it is enough to reason about them *locally*. Open bisimilarity can indeed be proved to be a sound proof technique for contextual equivalence. Unfortunately, due to the limited testing power provided by open bisimulations, open bisimilarity turned out to strictly finer (i.e. strictly included) than contextual equivalence, in general[16].

[10] That is, the congruence relation (inductively) generated by the relation relating terms of the form $(\lambda x.e)v$ with $e[x := v]$.

[11] Think, for instance, to $\Omega \triangleq (\lambda x.xx)(\lambda x.xx)$.

[12] Normal form, actually. We will say more on that later.

[13] Defining a λ-term e to be closed if all its variables are bound by a λ (and open otherwise), we see that the environment now tests *open* terms, from which the name *open bisimulation*.

[14] The expressions *normal form bisimulation* and *open bisimulation* are used interchangeably, although the former is arguably the most used one. We will stick with this convention, although we remark that the name *open bisimulation* seems more appropriate given both the foundational link with the notion of an open bisimulation for the π-calculus, and the relevance of testing *open* terms (the latter being a central difference with applicative bisimulation).

[15] E.g. by passing values as arguments to the tested terms.

[16] That holds for both the call-by-name and the call-by-value λ-calculus, as we will see. However, due to the celebrated Böhm

1.1.3 A Bird's-eye View on Program Equivalence

In previous section we informally introduced some of the (operational) equivalences will study in this dissertation. There are, of course, many other ones, such as logical relations-based equivalences (Appel & McAllester, 2001; G. Plotkin, 1973; Reynolds, 1983; Sieber, 1992), just to mention a single example. Designing operational equivalences as well as (relationally-based) proof techniques for contextual equivalence, it is possible to recognise a limited number of different principles (or patterns) behind such techniques. These principles can be used to classify program equivalences according to the key ideas behind their definitions, and thus provide a convenient lens through which our analysis can be viewed. In the following, we summarise such principles, taking inspiration from (McLaughlin, McKinna, & Stark, 2018).

- *Observational*, whereby programs are tested against a restricted family of contexts. This idea finds its prime example in Milner's context lemma (Milner, 1977), and gives rise to program equivalences such as CIU-like equivalences (Mason & Talcott, 1991).
- *Applicative or extensional*, whereby programs are tested according to the function extensionality principle for their input-output behaviour. This principle gives rise to Abramsky's applicative bisimilarity (Abramsky, 1990a), and its extensions (e.g. environmental bisimilarity (Koutavas, Levy, & Sumii, 2011; Sangiorgi, Kobayashi, & Sumii, 2011)).
- *Intensional*, whereby programs are tested in isolation according to the syntactic structure of suitable infinitary normal forms. This is the principle behind Böhm tree-like equivalences (Barendregt, 1984; Böhm, 1968) and open/normal form equivalences (S. B. Lassen, 1999; Sangiorgi, 1994).
- *Logical*, whereby programs are tested against logically related inputs and are required to produce logically related outputs. This is the principle behind logical relations (Appel & McAllester, 2001; G. Plotkin, 1973; Reynolds, 1983; Sieber, 1992).

1.2 Computational Effects

The reader should now have some intuitions behind programs equivalence in general, and some concrete λ-calculus equivalence in particular. However, up to this point we have ignored one of the main issue of this dissertation, namely *computational effects*. Roughly speaking, (computational) effects are those aspects of computation that involve forms of interaction with the environment (sometimes referred to as the external world). Classical examples are probabilistic and nondeterministic computations (in both cases the environment provides the source of nondeterminism), imperative stateful computations (where the environment is given by the machine state), and computations performing input and output operations. It is clear that any serious analysis of program equivalence should be concerned with computational effects[17], to some extent (after all, even printing the result of a computation on the screen of a laptop is a computational effect).

Before looking at what a notion of an *effectful program equivalence* might be, we have to understand what an *effectful program* is. In fact, the correspondence between programs and λ-terms on which we built our previous analysis does not directly scale to effectful programming languages.

Theorem (also known as Separation Theorem) (Böhm, 1968), open bisimilarity has been proved to be fully abstract for contextual equivalence when the λ-calculus is equipped with the head reduction strategy.

[17] That of course does not mean that the analysis of effect-free programs is meaningless: several interesting notions can be actually studied in the setting of effect-free programs, where such notions find good approximations.

1.2.1 The Computational λ-calculus

According to our informal understanding of computational effects, a fast examination of the syntax of the λ-calculus reveals a major drawback: the grammar of the λ-calculus allows to write *pure* programs only, i.e. programs that can be somehow described as mathematical functions[18]. This is a major point of departure with ordinary programming languages, which are, on the contrary, deeply effectful.

The notion of a *pure program* is usually identified with the one of a *referential transparent program* (Mitchell, 2002). Accordingly, a program is referentially transparent if it is equivalent to the value (if any) it evaluates to. A more pragmatic, although less formal, definition of referential transparency states that a program is referentially transparent if evaluating it multiple times always gives the same result. This property is not enjoyed by effectful programs, in general, as their evaluation depends on the environment in which they are executed (the result of program might depend on a keyboard input given by the user, for instance). Terms of the λ-calculus, instead, can be deemed as referentially transparent, since the evaluation of a λ-term is just the sequential application of the β-rule. As a consequence, the λ-calculus as it is, does not provide an adequate foundational calculus for effectful programming languages.

This deficiency led semanticists to look at extensions of the λ-calculus capable of modelling effectful computations. In his seminal work on notions of computation (Moggi, 1989, 1991), Moggi gave a unified account of computational effects as monads (MacLane, 1971), and designed a core calculus, called *computational λ-calculus*, for effectful computations.

The key insight of Moggi's analysis can be easily explained by means of a simple example. First of all, let us recall that a pure program f can be roughly approximated by a function $F : I \to O$ mapping inputs to outputs. In particular, the function F (and thus the program f) associates to any input $i \in I$ its unique output value $F(i)$ (notice that, according to such a reading, pure programs always terminates ...). This is no longer the case if f behaves nondeterministically, so that to any input i, f may associate several outputs. That is, f cannot be modelled as a function from inputs to outputs anymore. Moggi observed that such a nondeterministic program f can actually be modelled as function F, but instead of having as codomain the set O of outputs, this time F has as codomain the powerset of O, denoted by $\mathcal{P}(O)$. That is, to any input $i \in I$, the function F associates a unique set $F(i)$ of outputs, representing the possible outputs f may (nondeterministically) produce. In this way, it is possible to model the impure, nondeterministic program f by means of a function $F : I \to \mathcal{P}(O)$, and thus by a pure program.

The very same analysis can be given for many other effects. For instance, programs printing numerical strings can be modelled using the set $\mathbb{N}^* \times O$. Abstracting, we can think to an effectful program f as a function $F : I \to T(O)$, where T is some construction modelling the kind of effects f may produce. We call such constructions *notions of computations* (Moggi, 1991).

It is then natural to ask what is the structure of T. To answer such a question, let us look at pure programs first, and thus at functions. For simplicity, let us consider endofunctions, i.e. functions of the form $F : A \to A$, for some set A. The natural algebraic structure associated with the space of such functions is the one of a monoid: function composition plays the role of monoid multiplication, and the identity function $1 : A \to A$ behaves as the unit of the monoid. This is the algebra of functions, and thus of (pure) functional programs. Generalising from endofunctions to functions, the algebra of functions gives the notion of a category (MacLane, 1971). In particular, given functions $F : A \to B$ and $G : B \to C$, we can compose them obtaining $G \circ F : A \to C$. Computationally, that means that we can pass the output of a program as input to another program.

A minimal desideratum T should satisfy, is to allow to lift the algebra of functions to functions of the form $F : A \to TB$. Let us briefly expand on that. Consider two functions $F : A \to TB$ and $G : B \to TC$ modelling effectful programs. We immediately notice that the output of F is different from the input of G, as the former comes together with the effects produced by the program associated to F.

[18] This is not entirely true, as, for the sake of the argument, we are overlooking issues related to non-termination and partiality.

Nonetheless, we would like to be able to pass the actual value computed by F as input to G, combining somehow the effects produced. What we need is thus an operation $\circ_T$ allowing to produce the function $G \circ_T F : A \to TC$. Additionally, we would like such an operation $\circ_T$ to behave as kind of function composition, this way allowing to restore the algebra of functions[19]. The constructions T that allows to perform such an algebraic constructions are called monad (MacLane, 1971), and are a well-known notion in Category Theory (MacLane, 1971).

1.2.2 Algebraic Effects

Moggi's account of effectful computations is mostly denotational, focusing more on the nature of the functions associated with λ-terms, rather than on the linguistic analysis of effectful programs. From an operational perspective, the computational λ-calculus has minimal differences with the ordinary λ-calculus, and thus it does not provide as an adequate calculus for an operational analysis of effectful languages. In particular, the computational λ-calculus lacks primitives to actually produce effects, with the consequent necessity of extending it with suitable effect-triggering operations, whenever one is interested in giving formal accounts of effectful languages.

Moving from these observations, Plotkin and Power have recently introduced the theory of *algebraic effects* (G. D. Plotkin & Power, 2001, 2002, 2003) aiming to provide a more satisfactory account of effectful computations, both from an operational and denotational perspective. Accordingly, the theory of algebraic effects focuses on the computational primitives producing effects, rather than on effects on their own. In languages supporting algebraic effects, programs are written using effect-triggering operations (such as probabilistic choices or primitives for raising exceptions) which act as sources of the side effects of interest.

Although introduced only in recent years, algebraic effects provide a standard formalism for the operational analysis of effectful languages. This is due to their wide applicability to model concrete effectful calculi, on one hand, and on their nice operational properties, on the other hand. In fact, algebraic operations provide, among others, models for probabilistic and nondeterministic computations (by means of probabilistic and nondeterministic choice operations), imperative computations (by means of primitives for reading and writing global stores), computations with output (by means of primitives for printing values), and computations with exceptions (by means of primitives for throwing exceptions), as well as combinations thereof. In fact, contrary to monads, algebraic effects naturally support operations to combine effects with one another (Hyland, Plotkin, & Power, 2006), hence providing facilities for the design and analysis of languages exhibiting different kinds of effects.

Additionally, the algebraic nature of operations allows to have a tight control on the way effects are produced and propagated in a computation. In fact, from an operational perspective, algebraicity of an operation states that the operational behaviour of such an operation is independent of its arguments, meaning that effects act independently of the evaluation contexts in which they are executed. Taking advantage of this property of algebraic operations, (G. D. Plotkin & Power, 2001) defines a λ-calculus with algebraic operations, which plays the same role of Moggi's computational λ-calculus. Contrary to the latter, however, the λ-calculus in (G. D. Plotkin & Power, 2001) has a well-behaved operational semantics, hence giving evidences that by enriching the λ-calculus with algebraic operations, one obtains an adequate foundational calculus for effectful languages.

Obviously, algebraic effects have their own limitations. For instance, neither continuations (Hyland, Levy, Plotkin, & Power, 2007) nor exception handlers (G. D. Plotkin & Pretnar, 2013) can be modelled as algebraic effects[20]. Nonetheless, as previously stressed, several computational effects can be described

[19] Of course we also need to have a special function $\eta : A \to T(A)$ playing the role of the unit of $\circ_T$.

[20] Moving from the analysis of exception handlers, the theory of algebraic effects has been extended to the so-called theory of *effects and handlers* (Bauer & Pretnar, 2015; G. D. Plotkin & Pretnar, 2013; Pretnar, 2015). The latter is obtained by extending programming languages not only with algebraic operations, but also with the so-called *effects handlers*, the latter being syntactical

by means of algebraic operations. Additionally, by enriching the λ-calculus with algebraic operations we obtain a powerful (and mathematically well-behaved) formalism for the operational analysis of effectful programming languages. Even if such a formalism does not allow for a comprehensive analysis of effectful computations, it certainly allows for the development of a rich theory of effectful program equivalence, as this dissertation aims to show.

1.2.3 Effectful Program Equivalence

In previous section we have identified a family of core calculi for studying effectful programming languages, namely extensions of the λ-calculus with algebraic operations. The next step in our investigation is to understand what a notion of program equivalence for such calculi might be.

Looking back at our general analysis of program equivalence, we notice that purity did not play any role there, so that we are still interesting in finding suitable congruence relations between program phrases. That is not the case for the concrete equivalences we have introduced in Subsection 1.1.2. In fact, all of them rely on the fundamental assumption that the operational behaviour of a λ-term is entirely determined by the β-rule. Dealing with effectful programs, such an assumption has to be dropped: the evaluation of a program cannot be identified with the β-rule anymore, as it also has to take into account algebraic operations.

This is a great source of complexity for the operational analysis of program behaviour. Contrary to the β-rule (which made program evaluation a local process), evaluating algebraic operations require a program to interact with the environment, hence making the evaluation process itself interactive. Looking at program equivalence in terms of tests and observations, this further source of complexity results in far richer notions of observation, whereby the external observer can detect (to some extent, at least) not only the result of the evaluation of a program, but also the effects produced during such an evaluation (this way requiring to introduce suitable techniques to compare effects with one another).

The first part of this dissertation studies the equivalences introduced in Subsection 1.1.2 for λ-calculi extended with algebraic operations, with special focus on coinductive equivalences.

1.3 From Equivalences to Distances

Program equivalence answers the question of whether two programs have the same behaviour, and gives meaning to programs via a notion of synonymy. However, when dealing with effectful programs one may be interested in answering a different, more informative question: *how much different two programs are?*. Our focus being on effectful languages, it is of prime importance for our investigation to provide a satisfactory answer to such a question.

Our approach towards such an achievement is to design suitable notions of program distance (or program metric). Quantifying differences between program behaviours is of paramount importance for effectful languages, where effects are, by their very nature, context dependent. This is not the case for pure programs, whose behaviour is independent of the environment in which they are evaluated. As a consequence, changes in the environment do not affect the observable behaviour programs exhibit. However, as we have already remarked, effectful programs interact with the environment, and are thus sensitive to context changes. For instance, small perturbations in probabilities may break equivalence of probabilistic programs. It is then useful to have information on how much changes in the environment affect changes in the operational behaviour of programs, thus allowing for a finer, quantitative analysis of program equivalence.

The design of well-behaved notions of program distance constitute a challenging problem in the context of higher-order computation. In fact, higher-order programs have the ability to copy and evaluate

constructors handling the behaviour of (algebraic) operations.

their input as they like, so to speak. That means that when a higher-order program receives inputs with similar (but not equal) behaviour, each time it forces their evaluation an observable difference between their behaviour is detected. Additionally, whenever such a difference is the detected, it is added to the ones previously observed. As a consequence, by copying and evaluating its inputs enough times, the program is able to fully discriminate between its inputs, even if their expected difference is, in principle, negligible.

This observation mines our approach to the behavioural analysis of programs at its very roots: by reasoning in terms of experiments and observations, an external observer can always fully discriminate (i.e. assign maximal distance) to any pair of non-equivalent programs. As we will see, fixing this problem will force us to change our notion of testing process, following the rationale that one should take into account not only *how* a program is tested, but also *how much* it is tested.

The second main contribution of this dissertation is the design of a well-behaved program distance capable of handling the above described phenomenon. As we will see, achieving such a goal is simply not possible without providing information on the environment in which programs are tested. After all, this does not come with a big surprise, since context dependency is an intrinsic feature of effectful programs.

We conclude this informal introduction spending few words on the foundational meaning of moving from program equivalence to program distance.

1.3.1 A Foundational Interlude

Following the linguistic perspective on programming languages, we may ask what is the linguistic counterpart of the shifting from program equivalence to program distance. From a mathematical perspective, moving from program equivalence to program distance means refining equivalence relations into *pseudometrics*[21] (Searcóid, 2006), with the rationale that given a pseudometric δ, the quantity $\delta(e, e')$ quantifies the (behavioural) differences between the program phrases e and e'. As a notion of program equivalence describes synonymy, this way inducing a notion of meaning for program phrases, we may now ask what is the linguistic interpretation of program distance.

A conceptual (philosophical, somehow) answer to such a question is provided by *structural linguistics* (de Saussure & Baskin, 2011). Accordingly, meaning is produced *differentially*, i.e. throughout differences. As a consequence, in order to understand the meaning of an expression it is necessary to study its differences with the other expressions of the language, hence giving to the role of synonymy a secondary role.

The kind of differences we will consider in this work are rather rudimentary, as they are essentially based on (abstract) notions of quantities. Nonetheless, our analysis of notions of program distance can be read as constituting a small contribution to a richer understanding of program semantics, whereby the notion of identity (formalised through the notion of program equivalence) is secondary to the notion of difference (here formalised through the notion of program distance[22])[23].

Now that the reader has some background notions, we can outline the main contribution of this dissertation in a more detailed way.

[21] Understanding how to refine compatibility is not straightforward, and we will come back on that later.

[22] It is a fascinating question how to generalise the notion of a program *distance* to the (richer) notion of a program *difference*.

[23] The idea of regarding difference as having an ontological privilege over identity is at the heart of the field of philosophy known as *Differential Ontology* (Donkel, 2001).

1.4 First Contribution: Effectful Program Equivalence and Refinement

The first main contribution of this dissertation is the development of operationally-based notions of program equivalence and refinement for λ-calculi enriched with algebraic effects. The equivalences and refinements designed are mostly of a coinductive nature (with some minor exceptions given by contextual-like equivalences) and are based on different forms of bisimulation equivalence (resp. refinement). Following the classification of program equivalences outlined in Subsection 1.1.3, we can list the equivalences and refinements designed as follows.

Observational. Concerning observational equivalences and refinements, we define the notions of *effectful contextual approximation* $\preceq^{\text{ctx}}$ (resp. equivalence $\simeq^{\text{ctx}}$) and *effectful CIU approximation* $\preceq^{\text{ciu}}$ (resp. equivalence $\simeq^{\text{ciu}}$). These are defined in Chapter 5.

Applicative. Concerning applicative equivalences and refinements, we define the notions of *effectful applicative similarity* $\preceq^{\text{A}}$ (resp. bisimilarity $\simeq^{\text{A}}$) and *monadic applicative similarity* $\preceq^{\text{M}}$ (resp. bisimilarity $\simeq^{\text{M}}$). These are all defined in Chapter 5 and Chapter 6, respectively.

Intensional. Concerning intensional equivalences and refinements, we define the notions of *effectful normal form (bi)similarity* (notably effectful eager normal for similarity $\preceq^{\text{E}}$ and bisimilarity $\simeq^{\text{E}}$, and effectful weak head normal form similarity $\preceq^{\text{W}}$ and bisimilarity $\simeq^{\text{W}}$). These are defined in Chapter 7.

We briefly expand on the contributions given.

1.4.1 Effectful Applicative Similarly and Bisimilarity

In Chapter 5 we define the notion of *effectful applicative similarity* and *bisimilarity* for a call-by-value λ-calculus with algebraic effects (Chapter 3). Applicative bisimilarity is a *coinductive* (Milner, 1989; Park, 1981) program equivalence for pure λ-calculi that tests *pure* functional programs *extensionally*, i.e. for their input-output behaviour. Due to its coinductive nature, applicative bisimilarity comes equipped with a powerful proof technique, called the *coinduction proof principle*, which makes applicative bisimilarity a powerful tool for proving equivalence between programs.

Since Abramsky's seminal work (Abramsky, 1990b), applicative bisimilarity has been extended to several effectful calculi, notable examples being its extension to nondeterministic (S. Lassen, 1998b; C. L. Ong, 1993) and probabilistic (Crubillé & Dal Lago, 2014; Dal Lago, Sangiorgi, & Alberti, 2014) λ-calculi. However, all these extensions are rather specific to the effects considered, and seem to rely on their specific properties (for instance, the proof of congruence of applicative bisimilarity in (Dal Lago et al., 2014) relies on the Max-flow Min-cut Theorem (Schrijver, 1986)).

Our notion of effectful applicative (bi)similarity, instead, is designed to be parametric over a large class of computational effects. Concretely, effectful applicative (bi)similarity is parametrised by a monad and a *relator* (Barr, 1970; Thijs, 1996), the latter being an abstract construction axiomatising the structural properties of relation lifting operations (Kurz & Velebil, 2016). This level of abstraction makes effectful applicative bisimilarity (resp. bisimilarity) a powerful notion of program equivalence (resp. refinement) for effectful higher-order languages. In fact, suitable choices of relators allow to recover pure, probabilistic, nondeterministic, and imperative notions of applicative (bi)similarity, as well as combinations thereof.

In order to develop the theory of effectful applicative (bi)similarity, in Chapter 5 we develop an abstract relational framework in which several notions of effectful program equivalence and refinement can be defined. We take advantage of such a framework to define and analyse, among others, effectful

notions of contextual equivalence and approximation, CIU equivalence and approximation, and normal form similarity and bisimilarity (Chapter 7). The main results proved in Chapter 5 are precongruence (Theorem 4) and congruence (Theorem 5) theorems for effectful applicative similarity and bisimilarity, respectively. These theorems state that for a suitable class of relators, whose members we call Σ-continuous relators, effectful applicative similarity is a precongruence relation, whereas effectful applicative bisimilarity is a congruence relation. Remarkably, all the relators studied in this dissertation (Section 4.3) are Σ-continuous. As an immediate corollary, we obtain soundness of effectful applicative similarity (resp. bisimilarity) for effectful contextual approximation (resp. equivalence).

Theorem 4 and Theorem 5 are proved using an abstraction of Howe's method, which take advantage of the structural properties of relators. Howe's method (Howe, 1996) is a powerful relational construction used to prove (pre)congruence properties of operationally defined program relations. Howe's technique has been originally designed for the pure λ-calculus, and has later been generalised to several effectful λ-calculi, prime examples being its generalisation to nondeterministic (S. Lassen, 1998b) and probabilistic (Dal Lago et al., 2014) λ-calculi. Contrary to the first extension, the latter one is highly non-trivial, and relies on results from probability theory and linear programming (see Chapter 2 for an extensive discussion on Howe's method). This, as well as other further difficulties, led to the belief that Howe's technique is fragile in presence of effectful extensions of the λ-calculus.

Besides providing a powerful construction to prove congruence properties of effectful notions of equivalence, our abstract Howe's method sheds some light on the very essence of Howe's construction itself. In fact, using relators we can isolate the complexity of (pre)congruence proofs of applicative (bi)similarity, showing how Howe's method can be understood as a general relational technique parametrised by the effects considered. Additionally, we show a further application of our abstract account of Howe's method by proving that effectful CIU approximation (resp. equivalence) is fully abstract for effectful contextual approximation (resp. equivalence) (Theorem 6).

1.4.2 Normal Form Similarly and Bisimilarity

In Chapter 7 we define the notions of *effectful normal form similarity* and *bisimilarity* for both call-by-name and call-by-value calculi with algebraic effects.

The notion of applicative (bi)similarity is rooted in the idea that programs should be compared *extensionally*, i.e. according to their input-output behaviour. However, it is also possible to test programs intensionally, i.e. by inspecting the syntactic structure of the (possibly infinitary) normal form produced by a term during (iterations of) the evaluation process. The notion of equivalence one obtains by following this route is called *open* or *normal form (bi)similarity*[24] (S. B. Lassen, 1999, 2005; Sangiorgi, 1994).

Starting from the pioneering work by Böhm on the pure λ-calculus, the notion of a *Böhm tree* (Barendregt, 1984), and the associated notion of *Böhm tree equality*, has been proved extremely useful in reasoning about program behaviour. Roughly speaking, the Böhm tree of a λ-term e is a possibly infinite tree representing the infinitary head-normal form of e. The celebrated Böhm Theorem, also known as Separation Theorem (Böhm, 1968), stipulates that two terms are contextually equivalent if and only if their respective (appropriately η-equated) Böhm trees are the same.

The notion of equivalence induced by Böhm trees can be characterised without any reference to trees, by means of a suitable bisimilarity relation (S. B. Lassen, 1999; Sangiorgi, 1992, 1994). Additionally, Böhm trees can also be defined when λ-terms are evaluated according to the call-by-name (S. B. Lassen, 1999; Sangiorgi, 1994) and the call-by-value (S. B. Lassen, 2005) reduction strategy. In both cases, the notion of program equivalence one obtains by comparing the syntactic structure of trees, admits an elegant coinductive characterisation as a suitable bisimilarity relation. The family of bisimilarity relations thus obtained goes under the name of *normal form bisimilarity*.

[24] See Subsection 1.1.2 for a short remark on terminology.

Compared to other standard operational techniques, normal form bisimilarity has the major advantage of being an *intensional* program equivalence, equating programs according to the syntactic structure of the (possibly infinitary) trees produced by their evaluation. As a consequence, in order to deem two programs as normal form bisimilar, it is sufficient to test them in isolation, i.e. independently of their interaction with the environment (the latter being allowed to observe, but not to influence, computations). This feature makes normal form bisimilarity a powerful technique for program equivalence. The drawback of such a restricted form of testing, is, however, that normal form bisimilarity is usually sound but not fully abstract for contextual equivalence (although full abstraction results are known to hold for calculi with a rich expressive power (Sangiorgi, 1994; Støvring & Lassen, 2007)).

In Chapter 7 we generalise normal form (bi)similarity to λ-calculi with algebraic effects. The framework employed for such a generalisation is the same one defined in Chapter 5 for effectful applicative (bi)similarity. Perhaps surprisingly, the very same axioms of relators that guarantee effectful applicative similarity to be a precongruence relation (and effectful applicative bisimilarity to be a congruence relations), guarantee our generalisation of normal form similarity, which we call effectful normal similarity, to be a precongruence relation too (and thus effectful normal bisimilarity to be a congruence relation). This is the content of Theorem 9, Theorem 10, and Theorem 11.

Additionally, effectful normal form (bi)similarity provides a sound proof technique not only for effectful contextual approximation (resp. equivalence), but for effectful applicative (bi)similarity as well (Proposition 22). Finally, we show that normal form (bi)similarity allows for enhancements of the bisimulation proof method (Pous & Sangiorgi, 2012) (Theorem 13 and Theorem 12), hence making normal form (bi)similarity an extremely powerful tool for reasoning about program equivalence.

1.4.3 Monadic Applicative Similarly and Bisimilarity

In Chapter 6 we propose a new notion of applicative bisimilarity for a call-by-name calculus with algebraic effects, which we call *monadic applicative (bi)similarity*.

Contrary to effectful applicative (bi)similarity, monadic applicative (bi)similarity is not defined as a relation between programs, but as a relation between their semantics. This paradigm shift originates by the observation that in call-by-name calculi, contextual equivalence is *de facto* a form of trace equivalence. In fact, contrary to call-by-value calculi, in a call-by-name λ-calculus terms can be tested in functional position only, meaning that testing a term in an arbitrary environment is morally equivalent to testing the term against a finite sequence of inputs.

Taking advantage of this observation, monadic applicative (bi)similarity is shown to be not only sound, but also fully abstract for effectful contextual approximation/equivalence, under mild conditions (Theorem 8, Corollary 4, and Corollary 5). That holds, however, only for call-by-name calculi.

We summarise the soundness and completeness results obtained for call-by-value calculi in Table 1.1, whereas the soundness and completeness results obtained for call-by-name calculi are summarised in Table 1.2.

$\precsim^{E}$	$\subsetneq$	$\precsim^{A}$	$\subsetneq$	$\precsim^{ciu}$	$=$	$\precsim^{ctx}$
$\simeq^{E}$	$\subsetneq$	$\simeq^{A}$	$\subsetneq$	$\simeq^{ciu}$	$=$	$\simeq^{ctx}$

Table 1.1: Call-by-value approximation/equivalence spectrum.

$\precsim^{W}$	$\subsetneq$	$\precsim^{A}$	$\subsetneq$	$\precsim^{M}$	$=$	$\precsim^{ciu}$	$=$	$\precsim^{ctx}$
$\simeq^{W}$	$\subsetneq$	$\simeq^{A}$	$\subsetneq$	$\simeq^{M}$	$=$	$\simeq^{ciu}$	$=$	$\simeq^{ctx}$

Table 1.2: Call-by-name equivalence/approximation spectrum.

1.5 Second Contribution: Effectful Program Distances

From Chapter 9 to Chapter 12 we tackle the problem of defining a well-behaved program distance for effectful languages. As already remarked, the question of quantifying observable differences between programs is particularly interesting (and challenging) for effectful higher-order languages, where ordinary qualitative (i.e. boolean-valued) equivalences and refinements are often too strong. This is witnessed by recent results on behavioural pseudometrics for probabilistic λ-calculi (Crubillé & Dal Lago, 2015, 2017) as well as results on semantics of higher-order languages for differential privacy (de Amorim et al., 2017; Reed & Pierce, 2010).

In Chapter 12 we define *effectful applicative bisimilarity distance*, the quantitative refinement of effectful applicative bisimilarity, for an extension of the calculus *Fuzz* (de Amorim et al., 2017; Reed & Pierce, 2010) with algebraic effects (Chapter 10). *Fuzz* is a foundational calculus developed in the context of differential privacy (Dwork, 2006) with a linear-like type system (Maraist, Odersky, Turner, & Wadler, 1999) capable of expressing *program sensitivity*. Roughly speaking, the latter is the law describing how much behavioural differences in output are affected by behavioural difference in input, and can thus be used to provide information on the environment in which programs are tested.

The necessity of moving from the mainstream formalisms studied in previous parts of this dissertation to calculi with program sensitivity is grounded in the so-called *distance trivialisation* (Crubillé & Dal Lago, 2015, 2017). Roughly speaking, a program metric trivialises when it collapses to a program equivalence. Distance trivialisation results are rooted in higher-order features of functional languages. Intuitively, when passing programs with distance ε as input to another program e, the latter can copy and evaluate its inputs *ad libitum*, each time detecting a difference ε between them. Additionally, each time such a difference is detected, it is also added to the one previously observed, so that in the end e will be able to fully discriminate between its inputs. This makes reasoning about program metrics *compositionally* simply not possible. Program sensitivity provides an elegant way to give information about the testing power of programs, and thus about the environment in which programs are evaluated. Relying on program sensitivity, it is possible to define a suitable notion of 'metric-compositionality' in the form of a Lipschitz-continuity condition.

In Chapter 10 and Chapter 11 we develop a general framework to study program metrics in the abstract, and instantiate such a framework to define *effectful applicative (bi)similarity distance* (Chapter 12), the quantitative refinement of effectful applicative (bi)similarity. Accordingly, the theory of effectful applicative (bi)similarity distance builds on three major improvements of the theory of effectful applicative (bi)similarity of Chapter 5.

1. The first improvement is to move from boolean-valued relations to relations taking values on quantitative domains (such as $[0, \infty]$ or $[0, 1]$) in such a way that restricting these domains to the boolean algebra $\{0, 1\}$ makes the theory collapse to the usual theory of effectful applicative (bi)similarity. For that, we rely on Lawvere's analysis (F. Lawvere, 1973) of generalised metric spaces and preordered sets as enriched categories, and work with relations taking values over arbitrary quantales (Rosenthal, 1990). We call such relations *quantale-valued relations*.

2. The second improvement is the generalisation of the notion of a relator to quantale-valued relators, i.e. relators acting on quantale-valued relations. Perhaps surprisingly, such a generalisation is at the heart of the filed of *monoidal topology* (Hofmann, Seal, & Tholen, 2014), a subfield of categorical topology aiming to unify ordered, metric, and topological spaces in categorical terms.

3. The third improvement is the development of a *compositional* theory of behavioural quantale-valued relations (and thus of behavioural distances). Due to distance trivialisation, ensuring compositionality in an higher-order setting is particularly challenging. In order to achieve compositionality, we take advantage of the notion of program sensitivity, and use the latter to define a

suitable notion of metric-like compositionality. As we will see in Chapter 9, such a paradigm shift is necessary in order to ensure good properties of program metrics.

The result obtained is an abstract theory of behavioural quantale-valued relations that allows to define notions of quantale-valued applicative similarity and bisimilarity — which we call effectful applicative similarity and bisimilarity distance, respectively — parametrised by a quantale-valued relator. The notions obtained generalise the existing notions of real-valued applicative (bi)similarity and can be instantiated to concrete calculi to provide new notions of applicative (bi)similarity distance. A remarkable example is provided by probabilistic λ-calculi, where to the best of the author's knowledge a (non-trivial) applicative distance for a universal (i.e. Turing complete) probabilistic λ-calculus is still lacking in the literature (but see Section 13.1).

Our first main result (Theorem 16) states that under suitable conditions on monads and quantale-valued relators, effectful applicative similarity distance is a compatible (i.e. compositional) reflexive and transitive quantale-valued relation. The second main result proved (Theorem 17), instead, states that under mild conditions Theorem 16 extends to effectful applicative bisimilarity distance, which is thus proved to be a compatible, reflexive, symmetric, and transitive quantale-valued relation (i.e. a compatible pseudometric).

Chapter 2

An Informal, Motivating Example

> In signs one observes an advantage in discovery which is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then indeed the labor of thought is wonderfully diminished
>
> Gottfried Wilhelm Leibniz, Briefwechsel von G. W. Leibniz mit Mathematikern

In order to introduce and justify the abstract framework developed in this thesis, we begin with a gentle, yet non-trivial, example of an effectful (functional) calculus, namely a call-by-value probabilistic λ-calculus. Such a calculus, which we call Λ_p, is a fine-grain (P. Levy, Power, & Thielecke, 2003) version of the probabilistic untyped λ calculus studied in e.g. (Dal Lago et al., 2014; Dal Lago & Zorzi, 2012). The latter is obtained by adding to the pure, untyped λ-calculus (Barendregt, 1984; Church, 1985) a binary fair probabilistic choice operator, hence making program evaluation inherently probabilistic.

The reason behind this choice is simple: Λ_p allows to identify most of the features one encounters when studying (algebraic) effectful extensions of λ-calculi, and thus provides a good starting point to build intuitions behind the abstract machinery we will introduce in next chapters.

Besides introducing the syntax and (operational) semantics of Λ_p, in this chapter we define the notions of *probabilistic contextual approximation* (resp. *equivalence*) and *probabilistic applicative similarity* (resp. *bisimilarity*), and prove that probabilistic applicative similarity is sound for probabilistic contextual approximation, meaning that the former is contained in the latter. Although this thesis studies several notions of program equivalence and refinement, such as applicative and normal form (bi)similarity, contextual equivalence and approximation, and CIU equivalence and approximation, here we will *de facto* focus on applicative (bi)similarity only. In fact, proving probabilistic applicative similarity to be a precongruence relation (from which its inclusion in contextual approximation directly follows) is highly non-trivial, and provides fundamental insights into the mathematical apparatus we will use in our abstract analysis of program equivalence and refinement.

Let us expand on the latter point. In his seminal work (Howe, 1996) Howe introduced a powerful relational technique to prove congruence of applicative bisimilarity for pure (i.e. effect free) λ-calculi: such a technique is nowadays known as *Howe's method* (Pitts, 2011). Howe's method has been extended to several calculi with specific effects, its extension to nondeterministic (S. Lassen, 1998b; C. L. Ong, 1993) and probabilistic (Crubillé & Dal Lago, 2014; Dal Lago et al., 2014) λ-calculi being prime examples.

In most of these extensions the logical complexity of Howe's method (which is essentially determined by the so-called *Key Lemma*) is, in a way, comparable to the one of Howe's original method for the pure λ-calculus.

This is not the case for probabilistic calculi, where the proof of precongruence for applicative similarity (and thus the proof of congruence for applicative bisimilarity) requires to combine non-trivial results from linear programming (Schrijver, 1986) with the standard relational technique of Howe's method. This is witnessed by the complicated proof of precongruence of probabilistic applicative similarity in (Dal Lago et al., 2014). As a consequence, there is an apparent mismatch between the complexity of Howe's method when applied to different effectful calculi, which led several researchers to consider Howe's technique fragile in presence of effects.

The abstract analysis of applicative (bi)similarity we will make in Chapter 5 allows to isolate the complexity of (pre)congruence proofs for applicative (bi)similarity, showing how Howe's method can be understood as a general relational technique parametric with respect to the effects considered. That allows to understand the deep reasons why different instances of Howe's method in the literature seem to have different complexities: those are, so to speak, properties of effects, rather than of Howe's method.

Organisation

After having introduced the syntax and operational semantics of Λ_p, we define probabilistic contextual approximation and probability applicative similarity. Compared to their symmetric counterparts, probabilistic applicative similarity and probabilistic contextual approximation are amenable to an easier mathematical treatment, and thus we focus on the latter for now.

We stress the definition of probabilistic applicative similarity in terms of a suitable *relation lifting* operation. This is major difference compared to previous works on probabilistic applicative (bi)similarity (Crubillé & Dal Lago, 2014; Dal Lago et al., 2014). Looking at (probabilistic) applicative similarity in terms of relation lifting operations allows to isolate the structural properties such operations should satisfy in order to guarantee (probabilistic) applicative similarity to be precongruence relation. We conclude the chapter discussing the proof of the precongruence theorem for probabilistic applicative similarity and its possible generalisation.

Finally, we remark that the theory presented in this chapter is nothing more than a mechanical instantiation of the abstract theory of *effectful applicative (bi)similarity* we will study in Chapter 5. Such a theory might then appear to the reader as a rather straightforward generalisation of the the theory of probabilistic applicative similarity presented here: if that is the case, then this introductory chapter made its purpose. Chronologically, however, Ugo Dal Lago, Paul Blain Levy, and the author first developed the theory of effectful applicative (bi)similarity, which then led to a better understanding of probabilistic applicative (bi)similarity.

2.1 A Probabilistic λ-calculus and Its Operational Semantics

We now introduce our running example calculus Λ_p. We will not give all formal details here (these are not needed for our purposes), which are given instead in Chapter 3 in full generality.

Terms of Λ_p are divided into two disjoint classes: *computations* (denoted by $e, f, \ldots$) and *values* (denoted by $v, w, \ldots$). These are defined in Figure 2.1, where x ranges over a fixed countably infinite set of variables.

The difference between values and computation is the following: a computation *produces* (when evaluated) a value, and in doing so can perform some (side) effects, whose nature is probabilistic. A value, on the contrary, *is* the result of the evaluation of a computation. The key computation constructors are these:

e, f	::= **return** v	(return)	v, w ::= x	(variable)
	$\mid vw$	(application)	$\mid \lambda x.e.$	(abstraction)
	$\mid$ **let** $x = e$ **in** f	(sequencing)		
	$\mid e$ **or** f.	(probabilistic choice)		

Figure 2.1: Syntax of Λ_Σ.

- **return** v is the *trivial computation*. It converts a value v into a computation **return** v that (trivially) produces v (without performing any side effect).
- **let** $x = e$ **in** f is the *sequenced computation* (also known as *sequencing*). Its operational meaning can be informally described as follows: "evaluate e (say it produces a value v), bind v to x in f (we use the notation $f[x := v]$ to denote the resulting computation), then evaluate $f[x := v]$". Clearly the effects produced during evaluation of e and f must be combined (we will come back on that later).
- e **or** f is the actual effectful computation. The operation symbol **or** acts as an effect-triggering operation, evaluating e with probability $\frac{1}{2}$, and f with probability $\frac{1}{2}$.

An example may help to clarify the intuitive (operational) semantics of Λ_p.

Example 1. Let $Pr(e \Downarrow v)$ denotes the probability that the program e evaluates to the value v. For instance, the term $e \triangleq$ (**return** v_1) **or** (**return** v_2) evaluates to v_1 with probability $\frac{1}{2}$ and to v_2 with probability $\frac{1}{2}$. Symbolically, $Pr(e \Downarrow v_i) = \frac{1}{2}$ (for $i \in \{1, 2\}$). Let now f be a term with a single free variable x such that $f[x := v_1]$ evaluates to u_1 with probability $\frac{1}{3}$ and $f[x := v_2]$ evaluates to u_2 with probability $\frac{2}{3}$. Then the probability that the program **let** $x = e$ **in** f evaluates to u_1 is obtained as the product of the probability of e evaluating to v_1 and the probability of $f[x := v_1]$ evaluating to u_1. This is the way effects are composed in Λ_p. More formally, the probability that the program **let** $x = e$ **in** f evaluates to a value w is obtained as:

$$Pr(\textbf{let } x = e \textbf{ in } f \Downarrow w) = \sum Pr(e \Downarrow v) \cdot Pr(f[x := v] \Downarrow w)$$

⊠

Before giving a formal treatment of the operational semantics of Λ_p, let us introduce some syntactical conventions. We adopt standard syntactical conventions as in (Barendregt, 1984). In particular, we write $e[x := v]$ (resp $w[v/x]$) for the capture-free substitution of the value v for all free occurrences of x in e (resp. w) and identify terms up to renaming of bound variables. A *program* is a closed computation, i.e. a computation without free variables. In light of the relational apparatus we will develop in next sections, it is useful to introduce the 'hygienic' convention of keeping track of free variables of computations and values. We do so by means of sequents of the form $\Gamma \vdash^\Lambda e$ and $\Gamma \vdash^\mathsf{v} v$. The letter Γ ranges over finite sets of variables, and the intended meaning of a sequent $\Gamma \vdash^\Lambda e$ is that e is a computation with free variables among Γ (the sequent $\Gamma \vdash^\mathsf{v} v$ has a similar reading). Rules for sequents are given in Figure 2.2, where we write Γ, x in place of $\Gamma \cup \{x\}$.

Clearly, provable sequents are closed under weakening, meaning that if $\Gamma \vdash^\Lambda e$ (resp. $\Gamma \vdash^\mathsf{v} v$) is provable, then so is $\Gamma, x \vdash^\Lambda e$ (resp. $\Gamma, x \vdash^\mathsf{v} v$). From now on when speaking about sequents we will tacitly mean provable sequents. Closed terms thus correspond to sequents with empty premises (which

$$\frac{}{\Gamma, x \vdash^{\mathsf{v}} x} \qquad \frac{\Gamma, x \vdash^{\Lambda} e}{\Gamma \vdash^{\mathsf{v}} \lambda x.e} \qquad \frac{\Gamma \vdash^{\mathsf{v}} v \quad \Gamma \vdash^{\mathsf{v}} w}{\Gamma \vdash^{\Lambda} vw} \qquad \frac{\Gamma \vdash^{\mathsf{v}} v}{\Gamma \vdash^{\Lambda} \mathbf{return}\ v} \qquad \frac{\Gamma \vdash^{\Lambda} e \quad \Gamma, x \vdash^{\Lambda} f}{\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f} \qquad \frac{\Gamma \vdash^{\Lambda} e \quad \Gamma \vdash^{\Lambda} f}{\Gamma \vdash^{\Lambda} e\ \mathbf{or}\ f}$$

Figure 2.2: Sequents for Λ_p.

we simply denote as $\vdash^{\Lambda} e$ and $\vdash^{\mathsf{v}} v$). We also introduce the following notation:

$$\Lambda \triangleq \{e \mid \exists \Gamma.\, \Gamma \vdash^{\Lambda} e\} \qquad \Lambda_\Gamma \triangleq \{e \mid \Gamma \vdash^{\Lambda} e\} \qquad \Lambda_\circ \triangleq \{e \mid \ \vdash^{\Lambda} e\}$$
$$\mathcal{V} \triangleq \{v \mid \exists \Gamma.\, \Gamma \vdash^{\mathsf{v}} v\} \qquad \mathcal{V}_\Gamma \triangleq \{v \mid \Gamma \vdash^{\mathsf{v}} v\} \qquad \mathcal{V}_\circ \triangleq \{v \mid \ \vdash^{\mathsf{v}} v\}$$

We can now have a closer look at the operational semantics of Λ_p. A convenient way of representing the result of the evaluation of a program e is by means of (discrete) subdistributions over values. A discrete subdistribution over a set X is a function $\mu : X \to [0,1]$ such that the support $\mathrm{supp}(\mu) \triangleq \{x \in X \mid \mu(x) > 0\}$ of μ is countable, and $\sum_{x \in \mathrm{supp}(\mu)} \mu(x) \leq 1$ (notice that because $\mathrm{supp}(\mu)$ is countable such a sum always exists). Clearly, if X is countable (as it is e.g. the set of closed values), every map $\mu : X \to [0,1]$ has countable support (and thus we simply write $\sum_{x \in X} \mu(x)$ in place of $\sum_{x \in \mathrm{supp}(\mu)} \mu(x)$). We denote by DX the collection of (discrete) subdistributions over X, and refer to its elements simply as subdistributions.

We define the result $[\![e]\!]$ of the evaluation of a program e as a subdistribution over (closed) values. The idea is that $[\![e]\!](v)$ gives the probability that e evaluates to v. That is, $[\![e]\!](v) = Pr(e \Downarrow v)$. This justifies our choice of using subdistributions rather than distributions: by assigning probability zero to values we can model divergence. For instance, the probability subdistribution $[\![\Omega]\!]$ associated with the purely divergent program $\Omega \triangleq (\lambda x.xx)(\lambda x.xx)$ will assign probability 0 to any value (since Ω diverges, it converges to a value v with probability 0).

When trying to formalise this idea we immediately face a first difficulty. In fact, contrary to pure calculi, we cannot give a standard, *inductive* operational semantics to Λ_p. In pure (as well as purely nondeterministic) λ-calculi, if a term converges to a value, then it does so in a finite number of steps. In a probabilistic setting a program may converge to a value with probability 1, but in infinitely many steps only.

To see that, it is useful to represent the subdistribution obtained evaluating a program syntactically. We do so by means of the so-called *computation trees* (G. D. Plotkin & Power, 2001). These are infinitary trees whose nodes are labelled with operation symbols (just the probabilistic choice symbol **or**, in our case) and whose leaves are either values or a bottom symbol $\bot$ denoting pure divergence. For instance, the computation tree and its associated probabilistic reading of the recursive program $e \triangleq (\mathbf{fix}(z,x).((\mathbf{return}\ I)\ \mathbf{or}\ zx))v$, where I is the identity combinator $\lambda x.\mathbf{return}\ x$ and **fix** is a call-by-value fixed point combinator[1] , are given in Figure 2.3. Looking at Figure 2.3, we easily see that e converges to I with probability $\lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{2^k} = 1$, but it does so taking infinitely many reduction steps.

To model such an infinitary behaviour we follow (Dal Lago & Zorzi, 2012) and notice that DX carries an ω-cppo structure. The order $\sqsubseteq$ on DX is defined pointwise: we say that $\mu \sqsubseteq \nu$ holds if and only $\mu(x) \leq \nu(x)$, for any $x \in X$. Clearly the always zero distribution $\bot$ acts as bottom element for $\sqsubseteq$. Moreover, any ω-chain $(\mu_n)_n = \mu_0 \sqsubseteq \mu_1 \sqsubseteq \cdots \sqsubseteq \mu_n \sqsubseteq \cdots$ has least upper bound given by $\sup_n \mu_n$.

[1] The call-by-value fixed point combinator **fix** is defined by the rule

$$\frac{\Gamma, z, x \vdash^{\Lambda} e}{\Gamma \vdash^{\mathsf{v}} \mathbf{fix}(z,x).e}$$

Its computational behaviour is given by a kind of generalised β-rule, which allows to rewrite a term of the form $(\mathbf{fix}(z,x).e)v$ as $e[z := \mathbf{fix}\ (z,x).e, x := v]$.

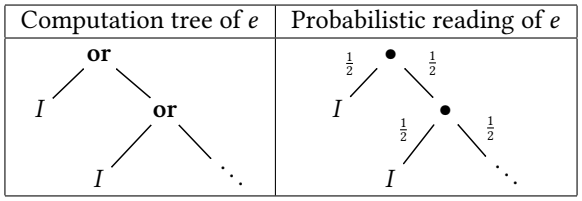


Figure 2.3: Computation tree of $e \triangleq (\mathbf{fix}(z,x).((\mathbf{return}\ I)\ \mathbf{or}\ zx))v$.

Relying on the ω-cppo of $D\mathcal{V}_\circ$, we define a $\mathbb{N}$-indexed family of evaluation functions $[\![-]\!]_n : \Lambda_\circ \to D\mathcal{V}_\circ$ such that, for any program e, $([\![e]\!]_n)_n$ forms an ω-chain in $D\mathcal{V}_\circ$. Intuitively, $[\![e]\!]_n$ is the n-th approximation of the evaluation $[\![e]\!]$ of e. Accordingly, we define $[\![e]\!]_0$ as $\bot$. Moreover, we would like the resulting evaluation function $[\![-]\!]$ to satisfy some minimal desiderata. First of all, in order to give semantics to programs of the form $\mathbf{return}\ v$ we need to regard values as subdistributions. In particular, we expect $[\![\mathbf{return}\ v]\!]$ to assign probability 1 to v (obviously $\mathbf{return}\ v$ converges to v with probability one), and 0 to all other values. This can be easily done since we can regard any element of a set X as a subdistribution by means of the map $\eta : X \to DX$ mapping $x \in X$ to its Dirac distribution $\eta(x)$ defined by:

$$\eta(x)(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{otherwise.} \end{cases}$$

Secondly, in order to treat the operation symbol $\mathbf{or}$ as a fair probabilistic choice, we expect

$$[\![e\ \mathbf{or}\ f]\!](v) = \frac{1}{2} \cdot [\![e]\!](v) + \frac{1}{2} \cdot [\![f]\!](v)$$

to hold for any value v. Lastly, we analyse sequencing. Given a program $\mathbf{let}\ x = e\ \mathbf{in}\ f$ we see that the (open) computation $x \vdash^\Lambda f$ induces a function $[\![f[x := -]]\!] : \mathcal{V}_\circ \to D\mathcal{V}_\circ$ defined by:

$$[\![f[x := -]]\!](v) \triangleq [\![f[x := v]]\!].$$

There is a natural way to lift a function $f : X \to DY$ (not to be confusion with the term f above) to a function $f^\dagger : DX \to DY$ which corresponds to the way we 'concatenated effects' in Example 1. Simply define:

$$f^\dagger(\mu)(y) \triangleq \sum_{x \in X} f(x)(y) \cdot \mu(x).$$

As a consequence, we would expect $[\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!] = [\![f[x := -]]\!]^\dagger [\![e]\!]$. We can thus give the following definition.

Definition 1. *The $\mathbb{N}$-indexed family of functions $[\![-]\!]_n : \Lambda_\circ \to T\mathcal{V}_\circ$ is inductively defined as follows:*

$$\begin{aligned} [\![e]\!]_0 &\triangleq \bot \\ [\![\mathbf{return}\ v]\!]_{n+1} &\triangleq \eta(v) \\ [\![(\lambda x.e)v]\!]_{n+1} &\triangleq [\![e[x := v]]\!]_n \\ [\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!]_{n+1} &\triangleq [\![f[x := -]]\!]_n^\dagger [\![e]\!]_n \\ [\![e\ \mathbf{or}\ f]\!]_{n+1} &\triangleq [\![e]\!]_n \oplus [\![f]\!]_n, \end{aligned}$$

where for $\mu, \nu \in DX$, $(\mu \oplus \nu)(x) \triangleq \frac{1}{2} \cdot \mu(x) + \frac{1}{2} \cdot \nu(x)$.

To see that Definition 1 indeed gives an ω-chain we notice that we have the following monotonicity and continuity result.

Lemma 1. *Recall that any function space of the form $X \to DY$ inherits an ω-cppo structure from DY pointwise[2]. The following monotonicity and continuity conditions hold, where n ranges over natural numbers.*

$$\begin{aligned} \mu \sqsubseteq \mu', \nu \sqsubseteq \nu' &\implies \mu \oplus \nu \sqsubseteq \mu' \oplus \nu' & \sup_n \mu_n \oplus \sup_n \nu_n &= \sup_n(\mu_n + \nu_n) \\ f \sqsubseteq g &\implies f^\dagger \sqsubseteq g^\dagger & (\sup_n f_n)^\dagger &= \sup_n f_n^\dagger \\ \mu \sqsubseteq \nu &\implies f^\dagger(\mu) \sqsubseteq f^\dagger(\nu) & f^\dagger(\sup_n \mu_n) &= \sup_n f^\dagger(\mu_n). \end{aligned}$$

The proof of Lemma 1 is straightforward (just notice that summations commute with suprema, since all numbers involved are positive). Using Lemma 1 (monotonicity) we see that the evaluation function of Definition 1 defines an ω-chain $(\llbracket e \rrbracket_n)_n$, for any closed term e. As a consequence, we can define $\llbracket e \rrbracket$ as the limit of such a chain:

$$\llbracket e \rrbracket \triangleq \sup_n \llbracket e \rrbracket_n.$$

Finally, we observe that another consequence of Lemma 1 (continuity) is that our evaluation semantics is continuous.

Proposition 1. *The function $\llbracket - \rrbracket : \Lambda_\circ \to D\mathcal{V}_\circ$ is the least function $\varphi : \Lambda_\circ \to D\mathcal{V}_\circ$ such that the following identities hold:*

$$\begin{aligned} \varphi(\mathbf{return}\ v) &= \eta(v) \\ \varphi((\lambda x.e)v) &= \varphi(e[x := v]) \\ \varphi(\mathbf{let}\ x = e\ \mathbf{in}\ f) &= (v \mapsto \varphi(f[x := v]))^\dagger \varphi(e) \\ \varphi(e\ \mathbf{or}\ f) &= \varphi(e) \oplus \varphi(f). \end{aligned}$$

At this point the reader might have noticed that so far we did not use anything really specific to subdistributions. Our evaluation semantics relied on the existence of an ω-cppo structure on sets of the form DX, as well as on the existence of (families of) functions of the form $\eta : X \to DX$ and $f^\dagger : DX \to DY$, for $f : X \to Y$. Such maps give D a monad structure (MacLane, 1971). We required these maps to properly interact with the ω-cppo structure of DX in the form of monotonicity and continuity properties. The only place where subdistributions played an explicit role is in the definition of the semantics of the operation symbol **or**. However, a more careful analysis reveals that the way we interpret such an operation symbol does not really matter, as far as it is interpreted as a continuous binary operation on DX. Moving from these observations, in the next chapter we will define an abstract, monadic semantics for effectful calculi. But before that, let us introduce the notion of probability applicative similarity.

[2] Overloading the notation, we define

$$\begin{aligned} f \sqsubseteq g &\overset{\triangle}{\iff} \forall x \in X.\, f(x) \sqsubseteq g(x) \\ \bot(x) &\triangleq \bot. \end{aligned}$$

.

2.2 Probabilistic Applicative Similarity

Following Abramsky's insights, we notice that the evaluation semantics of Definition 1 defines a probabilistic labelled transition systems over computations and values of Λ_p. Such a transition system is better described abstractly, as a structure resembling a *Markov chain*.

Recall that a discrete time Markov chain (Howard, 2007) (Markov chain, hereafter) is given by a set X (the state space of the chain) together with a map $c : X \to DX$. The function c describes probabilistic transitions over X, with the rationale that if $c(x)(y) = p$, then there is a transition from x to y with probability p. A natural notion of equivalence on the state space X is given by *Larsen-Skou probabilistic bisimilarity* (Larsen & Skou, 1989). Since most of the times we will be more interested in studying the more primitive notion of probabilistic similarity[3], we prefer to work directly with probabilistic similarity (we will discuss probabilistic bisimilarity in later chapters).

Recall that for a set X, a relation $\mathcal{R} \subseteq X \times X$, and a set $\mathfrak{X} \subseteq X$, the $\mathcal{R}$-image of $\mathfrak{X}$ is defined as $\mathcal{R}[\mathfrak{X}] \triangleq \{y \in Y \mid \exists x \in \mathfrak{X}.\, x\, \mathcal{R}\, y\}$. Moreover, for $\mu \in DX$ and $\mathfrak{X} \subseteq X$ we write $\mu(\mathfrak{X})$ for $\sum_{x \in \mathfrak{X}} \mu(x)$.

Definition 2. *Given a Markov chain $c : X \to DX$, we say that a relation $\mathcal{R} \subseteq X \times X$ is a probabilistic simulation if*

$$x\, \mathcal{R}\, y \implies \forall \mathfrak{X} \subseteq X.\, c(x)(\mathfrak{X}) \leq c(y)(\mathcal{R}[\mathfrak{X}]).$$

Probabilistic simulation is defined by *lifting* a relation $\mathcal{R}$ on the state space X to the set of subdistributions over X. Such a lifting can be defined more abstractly as follows. Given $\mu \in DX$, $\nu \in DY$, and a relation $\mathcal{R} \subseteq X \times Y$, we define the relation $\hat{D}\mathcal{R} \subseteq DX \times DY$ by:

$$\mu\, \hat{D}\mathcal{R}\, \nu \overset{\triangle}{\iff} \forall \mathfrak{X} \subseteq X.\, \mu(\mathfrak{X}) \leq \nu(\mathcal{R}[\mathfrak{X}]).$$

With a further abstraction, we can regard $\hat{D}$ as a set-indexed family of functions from $\mathsf{Rel}(X, Y)$ to $\mathsf{Rel}(DX, DY)$, where for sets X, Y, $\mathsf{Rel}(X, Y)$ denotes the complete lattice of binary relations (ordered by set-theoretic inclusion) between X and Y. We can thus phrase Definition 2 as follows.

Lemma 2. *Given a Markov chain $c : X \to DX$, define the functional $\Phi : \mathsf{Rel}(X, X) \to \mathsf{Rel}(DX, DY)$ by:*

$$\Phi\mathcal{R} \triangleq \{(x, y) \mid c(x)\, \hat{D}\mathcal{R}\, c(y)\}.$$

Then a relation $\mathcal{R} \subseteq X \times X$ is a probabilistic simulation if and only if $\mathcal{R} \subseteq \Phi\mathcal{R}$. That is, probabilistic simulations are exactly post-fixed points of Φ.

Since $\hat{D}$ is monotone (i.e. $\mathcal{R} \subseteq \mathcal{S} \implies \hat{D}\mathcal{R} \subseteq \hat{D}\mathcal{S}$), we see that Φ is a monotone endofunction on $\mathsf{Rel}(X, X)$. By the Knaster-Tarski Theorem (Davey & Priestley, 1990; Tarski, 1955), Φ has a greatest fixed point, which we call *probabilistic similarity* and denote by $\preceq$. Probabilistic similarity being defined coinductively (i.e. as the greatest fixed point of a Φ), it comes with an associated *coinduction proof principle* (Milner & Tofte, 1991; Park, 1981): in order to prove that two states of a Markov chain are similar, it is sufficient to exhibit a simulation relating them. Formally:

$$\frac{\exists\, \mathcal{R}\,.\, \mathcal{R} \subseteq \Phi\mathcal{R} \quad x\, \mathcal{R}\, y}{x \preceq y}\ (\preceq\text{-coind.})$$

or, in point-free notation,

$$\frac{\mathcal{R} \subseteq \Phi\mathcal{R}}{\mathcal{R} \subseteq \preceq}\ (\preceq\text{-coind.})$$

The notation employed to denote probabilistic similarity suggests the latter to be a preorder relation (which is clearly a desired property). This is indeed the case. In order to see that, we first observe that the map $\hat{D}$ satisfies the following properties.

[3] In general, a bisimulation is a relation $\mathcal{R}$ such that both $\mathcal{R}$ and its converse $\mathcal{R}^\circ$ are simulation.

Lemma 3. *For all relations $\mathcal{R} \subseteq X \times Y$, $\mathcal{S} \subseteq Y \times Z$ the following hold:*

$$=_{DX} \subseteq \hat{D}(=_X)$$
$$\hat{D}\,\mathcal{S} \cdot \hat{D}\mathcal{R} \subseteq \hat{D}(\mathcal{S} \cdot \mathcal{R}),$$

where $\mathcal{S} \cdot \mathcal{R} \triangleq \{(x, z) \mid \exists y \in Y.\, x\,\mathcal{R}\,y \wedge y\,\mathcal{S}\,z\}$ denotes relation composition.

We do not give a proof of Lemma 3 here, as proving properties of $\hat{D}$ is in general non-trivial. A guide is given in the case of *full* distributions by *Strassen's Theorem* (Strassen, 1965), an important result that characterises $\hat{D}$ (whose definition is based on an universal quantification) existentially.

Theorem 1 (Strassen's Theorem)**.** *Let μ and ν be full distribution over X and Y, respectively. A* coupling *of μ, ν is distribution ω over $X \times Y$ such that:*

$$\sum_{y \in Y} \omega(x, y) = \mu(x)$$
$$\sum_{x \in X} \omega(x, y) = \nu(y).$$

We denote by $\Omega(\mu, \nu)$ the set of couplings of μ and ν. The following holds for any relation $\mathcal{R} \subseteq X \times Y$:

$$\mu\,\hat{D}\mathcal{R}\,\nu \iff [\exists \omega \in \Omega(\mu, \nu).\, \omega(x, y) > 0 \implies x\,\mathcal{R}\,y].$$

We will give intuitions behind the meaning of the notion of a coupling and Strassen's Theorem in Example 30. For now, it is sufficient to observe that using Strassen Theorem it is possible to easily prove several properties of $\hat{D}$, provided we restrict our treatment to full distributions.

Using Lemma Lemma 3 and the coinduction proof principle, we can prove that $\preceq$ is a preorder. For instance, by showing that the identity relation is a probabilistic simulation, we can conclude $\preceq$ to be reflexive. Let us briefly expand on that. To see that for any Markov chain $c : X \to DX$, the identity relation $=_X$ is a probabilistic simulation, we simply observe that if $x =_X y$, then $c(x) =_{DX} c(y)$, and thus $c(x)\,\hat{D}(=_X)\,c(y)$ by Lemma 3. In a similar fashion we can show that $\preceq \cdot \preceq$ is a simulation, hence concluding $\preceq$ to be transitive.

We now look back at Λ_p. The evaluation function $[\![-]\!] : \Lambda_\circ \to D\mathcal{V}_\circ$ does not exactly define a Markov chain but, since there is an injection mapping values to computations, we have a 'kind of' Markov chain. In particular, given a relation $\mathcal{R}_\mathcal{V} \subseteq \mathcal{V}_\circ \times \mathcal{V}_\circ$ over closed values, we can modify Definition 2 by saying that a relation $\mathcal{R}_\Lambda \subseteq \Lambda_\circ \times \Lambda_\circ$ over closed computations is a probabilistic simulation with respect to $\mathcal{R}_\mathcal{V}$ if:

$$e\,\mathcal{R}_\Lambda\,f \implies [\![e]\!]\,\hat{D}\mathcal{R}_\mathcal{V}\,[\![f]\!].$$

At this point we just need to provide conditions on $\mathcal{R}_\mathcal{V}$ making the above implication meaningful. As for the pure λ-calculus, we test the behaviour of closed values (i.e. λ-abstraction) applicatively, hence requiring:

$$v\,\mathcal{R}_\mathcal{V}\,w \implies \forall u \in \mathcal{V}_\circ.\, vu\,\mathcal{R}_\Lambda\,wu.$$

Summing up, we give the following definition of a *probabilistic applicative simulation* relation.

Definition 3. *A pair of relations $\mathcal{R} = (\mathcal{R}_\Lambda \subseteq \Lambda_\circ \times \Lambda_\circ, \mathcal{R}_\mathcal{V} \subseteq \mathcal{V}_\circ \times \mathcal{V}_\circ)$ is a* probabilistic applicative simulation *if for all closed computations e, f, and all closed values v, w, we have:*

$$e\,\mathcal{R}_\Lambda\,f \implies [\![e]\!]\,\hat{D}\mathcal{R}_\mathcal{V}\,[\![f]\!] \tag{app 1}$$
$$v\,\mathcal{R}_\mathcal{V}\,w \implies \forall u \in \mathcal{V}_\circ.\, vu\,\mathcal{R}_\Lambda\,wu. \tag{app 2}$$

Definition 3 induces an endofunction $\mathcal{R} \mapsto [\mathcal{R}]$ on $\mathsf{Rel}(\Lambda_\circ, \Lambda_\circ) \times \mathsf{Rel}(\mathcal{V}_\circ, \mathcal{V}_\circ)$ mapping a pair of relations $(\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ to the pair of relations $([\mathcal{R}]_\Lambda, [\mathcal{R}]_\mathcal{V})$ defined by:

$$e\ [\mathcal{R}]_\Lambda\ f \overset{\triangle}{\iff} [\![e]\!]\ \hat{D}\mathcal{R}_\mathcal{V}\ [\![f]\!]$$
$$v\ [\mathcal{R}]_\mathcal{V}\ w \overset{\triangle}{\iff} \forall u \in \mathcal{V}_\circ.\ vu\ \mathcal{R}_\Lambda\ wu.$$

In particular, we see that a pair of relations $\mathcal{R}$ as above is a probabilistic applicative simulation if and only if $\mathcal{R} \subseteq [\mathcal{R}]$. Moreover, the set $\mathsf{Rel}(\Lambda_\circ, \Lambda_\circ) \times \mathsf{Rel}(\mathcal{V}_\circ, \mathcal{V}_\circ)$ inherits a complete lattice structure from $\mathsf{Rel}(\Lambda_\circ, \Lambda_\circ)$ and $\mathsf{Rel}(\mathcal{V}_\circ, \mathcal{V}_\circ)$ pointwise. Monotonicity of $\hat{D}$ makes $\mathcal{R} \mapsto [\mathcal{R}]$ monotone too, so that the latter has a greatest fixed point, which we call *probabilistic applicative similarity* and denote by $\preceq\ = (\preceq_\Lambda, \preceq_\mathcal{V})$. That $\preceq$ is a good candidate program refinement is witnessed by the following result, which, as before, can be proved by coinduction (relying on Lemma 3).

Proposition 2. *Probabilistic applicative similarity is a* preadequate *preorder relation, where a pair of relations $(\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ is preadequate if $e\ \mathcal{R}_\Lambda\ f \implies [\![e]\!](\mathcal{V}_\circ) \leq [\![f]\!](\mathcal{V}_\circ)$.*

Preadequacy of a relation (over closed computations) means that whenever two computations e, f are related, then the probability of convergence of e (i.e. $[\![e]\!](\mathcal{V}_\circ)$) is bounded by the probability of convergence of f (i.e. $[\![f]\!](\mathcal{V}_\circ)$). Preadequacy is nothing but the non-symmetric counterpart of the notion of *adequacy*, which simply states that related computations have the same probability of convergence (which is indeed a desired property of any notion of probabilistic equivalence). We will expand on the notions of adequacy and preadequacy in Chapter 5.

Proposition 2 gives good hints that probabilistic applicative similarity is an interesting candidate program refinement for Λ_p. Preadequacy tells us that probabilistic applicative similarity is consistent with ground observations on the operational behaviour of programs, whereas reflexivity and transitivity tell that we can use inequational reasoning to study program behaviour. In particular, transitivity allows us to deduce the refinement $e \preceq e'$ from the chain of intermediate refinements $e \preceq e_1 \preceq \cdots e_n \preceq e'$. However, in order to indeed qualify as a good program refinement, probabilistic applicative similarity must support *compositional* reasoning about program behaviour.

Compositionality is a fundamental notion in programming language theory, as well as in many other fields. In our context, we can roughly state that a program relation is compositional if it is preserved by all language (syntactical) constructors. That is, compositionality provides a suitable variation of Leibniz's identity law, whereby a refinement between compound expressions $C[e] \preceq C[f]$ follows from a refinement $e \preceq f$ between their sub-expressions. This way, we can take advantage of powerful algebraic laws for inqueational reasoning[4].

Formally, we say that a program equivalence is compositional if it is a *congruence* relation, and that a program refinement is compositional if it is a *precongruence* relation. As a consequence, what we need to show is that probabilistic applicative similarity is a precongruence relation. Before that, however, we introduce the notion of *probabilistic contextual approximation*.

2.3 Probabilistic Contextual Approximation

In previous sections we identified some desiderata that any (good) notion of program refinement should satisfy. These can be summarised as follows:

[4] An analogous argument holds for probabilistic applicative bisimilarity (and more generally for notions of program equivalence) and equational reasoning. Moreover, the notion of compositionality is oftentimes defined as the principle stating that the meaning of an expression is a function of the meaning of its parts (i.e. its sub-expressions). If we define the meaning of a program as its equivalence class modulo a given program equivalence, then our notion compositionality states exactly that the meaning of a program is determined by the meaning of its sub-expressions.

1. The candidate notion must be a preorder.

2. The candidate notion must be preadequate.

3. The candidate notion must be compositional, i.e. a precongruence relation.

It is therefore natural to ask whether there exists a canonical program refinement meeting all the above desiderata. Obviously, the identity relation provides one such a refinement. In fact, it comes with no surprise that syntactic equality behaves both as a notion program equivalence and as a notion of program refinement. Moreover, syntactic equality is the least relation satisfying conditions 1,2, and 3 above, meaning that syntactic equality is included in any refinement satisfying such conditions. However, it comes by itself that comparing programs according to their syntactic structure only does not tell anything useful about their operational behaviour.

There is another canonical notion of program refinement satisfying the above condition, namely Morris' style *contextual approximation* (also known as *operational* or *observational* approximation) (Morris, 1969). The latter is the *largest*, i.e. the less discriminating (or coarsest), relation satisfying conditions 1, 2, and 3. This universal property gives contextual approximation a prime position among program refinements, making it the universally accepted notion of operational refinement for sequential, higher-order languages.

From a more concrete perspective, contextual approximation is a syntax directed notion of refinement that orders programs according to a prefixed notion of observation (such as probability of convergence, in our case, or pure convergence, in the pure λ-calculus). Accordingly, a program f contextually refines a program e if there is no context of the language (the latter being a kind of program with a hole to be filled in with the tested program) capable of detecting operational behaviours of e that cannot be simulated by f. Such operational behaviours are specified by means of a suitable notion of observation, which describes the observable outcomes of the execution of a program.

For our calculus Λ_p, (probabilistic) contextual approximation $\preceq^{\mathrm{ctx}}$ can be roughly defined as follows:

$$e \preceq^{\mathrm{ctx}} f \iff \forall C.\, [\![C[e]]\!](\mathcal{V}_\circ) \leq [\![C[f]]\!](\mathcal{V}_\circ).$$

Here C ranges over *contexts*, i.e. terms with a single hole that can be replaced by a given program e, obtaining a new *program* $C[e]$. Defining and working with contexts, however, can be rather heavy and error-prone, as contexts cannot be defined modulo renaming of bound variables (see Remark 10 for details). We thus define probabilistic contextual approximation through its universal property. In order to do so, we first have to introduce some basic notions of program relational algebra in the spirit of (S. Lassen, 1998b).

The expression *program relational algebra* is used to denote a a symbolic apparatus allowing for algebraic manipulations and calculations with program relations. Such relations, which we call λ-term relations (S. Lassen, 1998b; Pitts, 2011), relate (possibly open) computations with (possibly open) computations, and (possibly open) values with (possibly open) values.

Definition 4. *1. A* closed *λ-term relation is a pair $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ of relations $\mathcal{R}_\Lambda$ and $\mathcal{R}_\mathcal{V}$ on $\Lambda_\circ$ and $\mathcal{V}_\circ$, respectively. We refer to $\mathcal{R}_\Lambda$ as the computation component of $\mathcal{R}$, and to $\mathcal{R}_\mathcal{V}$ as the value component of $\mathcal{R}$.*

2. An open *λ-term relation $\mathcal{R}$ associates to each finite set of variables Γ a relation $\Gamma \vdash^\Lambda - \mathcal{R} -$ on Λ_Γ, and a relation $\Gamma \vdash^\mathsf{v} - \mathcal{R} -$ on $\mathcal{V}_\Gamma$. We require open λ-term relations to be closed under weakening:*

$$\frac{\Gamma \vdash^\Lambda e\ \mathcal{R}\ f}{\Gamma, x \vdash^\Lambda e\ \mathcal{R}\ f} \qquad \frac{\Gamma \vdash^\mathsf{v} e\ \mathcal{R}\ w}{\Gamma, x \vdash^\mathsf{v} v\ \mathcal{R}\ w}$$

Although it might appear cumbersome at first, the notation we use for open λ-term relations has several advantages and highly improves readability (especially when dealing with typed calculi) of the relational techniques we will develop. As a convention, when referring to λ-term relations what we mean is *open* λ-term relations.

Example 2. Both the discrete/identity relation I and the indiscrete relation 0 defined by the rules below are open λ-term relations. The empty relation is an open λ-term relation too.

$$\frac{e \in \Lambda_\Gamma}{\Gamma \vdash^\Lambda e \mathrel{\mathsf{I}} e} \qquad \frac{v \in \mathcal{V}_\Gamma}{\Gamma \vdash^\mathcal{V} v \mathrel{\mathsf{I}} v} \qquad \frac{e, f \in \Lambda_\Gamma}{\Gamma \vdash^\Lambda e \mathrel{0} f} \qquad \frac{v, w \in \mathcal{V}_\Gamma}{\Gamma \vdash^\mathcal{V} v \mathrel{0} w}$$

Since $e \in \Lambda_\Gamma$ (resp. $v \in \mathcal{V}_\Gamma$) implies $e \in \Lambda_{\Gamma,\Delta}$ (resp. $v \in \mathcal{V}_{\Gamma,\Delta}$), for any finite set of variables Δ, both 0 and 1 are closed under weakening. ⊠

We denote by Rel and Rel^c the collections of open and closed λ-term relations, respectively. We can lift most of the usual (abstract) relational algebra to λ-term relations. For instance, by noticing that for any finite set of variables Γ both $\mathsf{Rel}(\Lambda_\Gamma, \Lambda_\Gamma)$ and $\mathsf{Rel}(\mathcal{V}_\Gamma, \mathcal{V}_\Gamma)$ form a complete lattice, we see that Rel carries a complete lattice structure too, its ordering being defined pointiwise. In particular, we write $\mathcal{R} \subseteq \mathcal{S}$ if:

$$\forall \Gamma, e, f.\, \Gamma \vdash^\Lambda e \mathrel{\mathcal{R}} f \implies \Gamma \vdash^\Lambda e \mathrel{\mathcal{S}} f$$
$$\forall \Gamma, v, w.\, \Gamma \vdash^\mathcal{V} v \mathrel{\mathcal{R}} w \implies \Gamma \vdash^\mathcal{V} v \mathrel{\mathcal{S}} w.$$

We will take advantage of the complete lattice structure of Rel by defining λ-term relations both inductively and coinductively.

Additionally, given λ-term relations $\mathcal{R}$ and $\mathcal{S}$ we can define the composition of $\mathcal{S}$ with $\mathcal{R}$, denoted by $\mathcal{S} \cdot \mathcal{R}$, as follows:

$$\frac{\Gamma \vdash^\Lambda e \mathrel{\mathcal{R}} g \quad \Gamma \vdash^\Lambda g \mathrel{\mathcal{S}} f}{\Gamma \vdash^\Lambda e \mathrel{(\mathcal{S} \cdot \mathcal{R})} f} \qquad \frac{\Gamma \vdash^\mathcal{V} v \mathrel{\mathcal{R}} u \quad \Gamma \vdash^\mathcal{V} u \mathrel{\mathcal{S}} w}{\Gamma \vdash^\mathcal{V} v \mathrel{(\mathcal{S} \cdot \mathcal{R})} w}$$

Since both $\mathcal{R}$ and $\mathcal{S}$ are closed under weakening, then so is $\mathcal{S} \cdot \mathcal{R}$. Moreover, we see that the unit of composition is given by the discrete λ-term relation I.

As a consequence, we obtain the notion of a *preorder* λ-term relation. In fact, we say that λ-term relation $\mathcal{R}$ is reflexive if $\mathsf{I} \subseteq \mathcal{R}$ and transitive if $\mathcal{R} \cdot \mathcal{R} \subseteq \mathcal{R}$. Moreover, we can define the converse λ-term relation $\mathcal{R}^\circ$ of $\mathcal{R}$ by the rules below, hence obtaining the notion of symmetric λ-term relation (i.e. a λ-term relation such that $\mathcal{R}^\circ \subseteq \mathcal{R}$) and thus the notion of an *equivalence* λ-term relation too.

$$\frac{\Gamma \vdash^\Lambda f \mathrel{\mathcal{R}} e}{\Gamma \vdash^\Lambda e \mathrel{\mathcal{R}^\circ} f} \qquad \frac{\Gamma \vdash^\mathcal{V} w \mathrel{\mathcal{R}} v}{\Gamma \vdash^\mathcal{V} v \mathrel{\mathcal{R}^\circ} w}$$

Finally, we observe that there are maps $-^c : \mathsf{Rel} \to \mathsf{Rel}^c$ and $-^o : \mathsf{Rel}^c \to \mathsf{Rel}$ restring open λ-term relations to closed ones, and extending closed λ-term relations to open ones. Formally, we define such maps as follows. Given $\mathcal{R} \in \mathsf{Rel}$, define the *closed restriction*[5] $\mathcal{R}^c = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ of $\mathcal{R}$ by:

$$\frac{\vdash^\Lambda e \mathrel{\mathcal{R}} f}{e \mathrel{\mathcal{R}_\Lambda} f} \qquad \frac{\vdash^\mathcal{V} v \mathrel{\mathcal{R}} w}{v \mathrel{\mathcal{R}_\mathcal{V}} w}$$

Dually, given a closed λ-term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ we define $\mathcal{R}^o$ as its *open extension*. The latter is defined as follows, where $\Gamma \triangleq \vec{x} \triangleq x_1, \ldots, x_n$ and $\vec{u} \triangleq u_1, \ldots, u_n$:

$$\frac{\forall \vec{u} \in \mathcal{V}_\circ.\, e[\vec{x} := \vec{u}] \mathrel{\mathcal{R}_\Lambda} f[\vec{x} := \vec{u}]}{\Gamma \vdash^\Lambda e \mathrel{\mathcal{R}^o} f} \qquad \frac{\forall \vec{u} \in \mathcal{V}_\circ.\, v[\vec{u}/\vec{x}] \mathrel{\mathcal{R}_\mathcal{V}} u[\vec{u}/\vec{x}]}{\Gamma \vdash^\mathcal{V} v \mathrel{\mathcal{R}^o} w}$$

[5] Notice that the notation $\mathcal{R}_\Lambda$ (resp. $\mathcal{R}_\mathcal{V}$) is not defined for open λ-term relations, so that we can safely use that to denote the computation (resp. value) component of $\mathcal{R}^c$.

Taking advantage of these definitions, for a *closed* λ-term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_V)$ we will often write $\vdash^\Lambda e \,\mathcal{R}\, f$ in place of $e \,\mathcal{R}_\Lambda\, f$, and $\Gamma \vdash^\Lambda e \,\mathcal{R}\, f$ in place of $\Gamma \vdash^\Lambda e \,\mathcal{R}^o\, f$ (and similarity for values). Dually, for an open λ-term relation $\mathcal{S}$ we will use the notations $\vdash^\Lambda e \,\mathcal{S}\, f$ and $e \,\mathcal{S}_\Lambda\, f$ interchangeably (and similarity for values).

Next we introduce the notion of a *substitutive* λ-term relation, which will be fundamental for proving probabilistic applicative similarity to be a precongruence.

Definition 5. *1. A λ-term relation $\mathcal{R}$ is* value-substitutive *if the following hold, where u ranges over closed values:*

$$\frac{\Gamma, x \vdash^\Lambda e \,\mathcal{R}\, f}{\Gamma \vdash^\Lambda e[x := u] \,\mathcal{R}\, f[x := u]} \qquad \frac{\Gamma, x \vdash^v v \,\mathcal{R}\, w}{\Gamma \vdash^v v[u/x] \,\mathcal{R}\, w[u/x]}$$

2. A λ-term relation $\mathcal{R}$ is substitutive *if the following hold:*

$$\frac{\Gamma, x \vdash^\Lambda e \,\mathcal{R}\, f \quad \vdash^v v \,\mathcal{R}\, w}{\Gamma \vdash^\Lambda e[x := v] \,\mathcal{R}\, f[x := w]} \qquad \frac{\Gamma, x \vdash^v v \,\mathcal{R}\, w \quad \vdash^v u \,\mathcal{R}\, u'}{\Gamma \vdash^v v[u/x] \,\mathcal{R}\, w[u'/x]}$$

A closed λ-relation is (value) substitutive if its open extension is. Moreover, we notice that the open extension of a closed λ-term relation is trivially value-substitutive.

We now have formal notions that allow us to talk about substitutive preorder (and equivalence) λ-term relations. However, what we are actually interested in are precongruence λ-term relations. In order to define the notion of a precongruence λ-term relation, we introduce the notion of *compatibility*. Roughly speaking, a λ-term relation is compatible if it is preserved by all Λ_p syntactic constructors. Formally, we follow (Gordon, 1994; S. Lassen, 1998b) and define compatibility via the notion of *compatible refinement*.

Definition 6. *The* compatible refinement $\widehat{\mathcal{R}}$ *of an open λ-term relation $\mathcal{R}$ is defined by the rules in Figure 2.4. We say $\mathcal{R}$ is compatible if $\widehat{\mathcal{R}} \subseteq \mathcal{R}$, and that a closed λ-term relation is compatible if its open extension is.*

$$\frac{}{\Gamma, x \vdash^v x \,\widehat{\mathcal{R}}\, x}\ (\text{comp-var})$$

$$\frac{\Gamma, x \vdash^\Lambda e \,\mathcal{R}\, f}{\Gamma \vdash^v \lambda x.e \,\widehat{\mathcal{R}}\, \lambda x.f}\ (\text{comp-abs}) \qquad \frac{\Gamma \vdash^v v \,\mathcal{R}\, v' \quad \Gamma \vdash^v w \,\mathcal{R}\, w'}{\Gamma \vdash^\Lambda vw \,\widehat{\mathcal{R}}\, v'w'}\ (\text{comp-app})$$

$$\frac{\Gamma \vdash^v v \,\mathcal{R}\, w}{\Gamma \vdash^\Lambda \mathbf{return}\ v \,\widehat{\mathcal{R}}\, \mathbf{return}\ w}\ (\text{comp-ret}) \qquad \frac{\Gamma \vdash^\Lambda e \,\mathcal{R}\, e' \quad \Gamma.x \vdash^\Lambda f \,\mathcal{R}\, f'}{\Gamma \vdash^\Lambda \mathbf{let}\ x = e\ \mathbf{in}\ f \,\widehat{\mathcal{R}}\, \mathbf{let}\ x = e'\ \mathbf{in}\ f'}\ (\text{comp-let})$$

$$\frac{\Gamma \vdash^\Lambda e \,\mathcal{R}\, e' \quad \Gamma \vdash^\Lambda f \,\mathcal{R}\, f'}{\Gamma \vdash^\Lambda e\ \mathbf{or}\ f \,\widehat{\mathcal{R}}\, e'\ \mathbf{or}\ f'}\ (\text{comp-op})$$

Figure 2.4: Compatible refinement of $\mathcal{R}$.

Notice that $\widehat{\mathcal{R}}$ is indeed a λ-term relation (notably, $\widehat{\mathcal{R}}$ is closed under weakening). Moreover, Definition 6 induces a map $\mathcal{R} \mapsto \widehat{\mathcal{R}}$ on the collection of open λ-term relations which is monotone and satisfies the identity $\widehat{(\mathcal{S} \cdot \mathcal{R})} = \widehat{\mathcal{S}} \cdot \widehat{\mathcal{R}}$. In particular, a λ-term relation is compatible if and only if it is a pre-fixed

point of $\mathcal{R} \mapsto \widehat{\mathcal{R}}$. It is not hard to see that the discrete open λ-term relation I of Example 2 is a prefixed point of $\mathcal{R} \mapsto \widehat{\mathcal{R}}$, and actually the least such. As a consequence, any compatible relation is reflexive.

Standard calculations show that compatibility is preserved by relation composition, and that the arbitrary intersection of compatible λ-term relations is compatible (the empty intersection being the indiscrete λ-term relation 0). This is not the case for set-theoretic union, as the union of compatible λ-term relations is in general not compatible. Nonetheless, we can define the join a family ρ of compatible λ-term relations as the smallest compatible λ-term relation extending $\bigcup \rho$, i.e. as:

$$\bigcap \{\mathcal{S} \mid \widehat{\mathcal{S}} \subseteq \mathcal{S}, \bigcup \rho \subseteq \mathcal{S}\}.$$

This way, we can give to the collection of compatible λ-term relations a complete lattice structure (see chapter 5 for details), and thus define compatible λ-term relations both inductively and coinductively.

We now have all the necessary ingredients to define probabilistic contextual approximation.

Definition 7. *We say that a λ-term relation $\mathcal{R}$ is* preadequate *if $\mathcal{R} \in \mathsf{Adeq}$, where:*

$$\mathcal{R} \in \mathsf{Adeq} \stackrel{\triangle}{\iff} \forall e, f \in \Lambda_\circ.\ [\![e]\!](\mathcal{V}_\circ) \leq [\![e]\!](\mathcal{V}_\circ).$$

Define probabilistic contextual equivalence $\preceq^{\mathrm{ctx}}$ *as the* largest preadequate compatible *λ-term relation.*

It is not immediately clear whether Definition 7 indeed defines a λ-term relation. In fact, we notice that preadequacy is not a monotone property, meaning that $\mathcal{R} \subseteq \mathcal{S}$ in general does not implies that if $\mathcal{R} \in \mathsf{Adeq}$, then $\mathcal{S} \in \mathsf{Adeq}$ too. As a consequence, we cannot extract (at least in a natural way) a monotone functional from Definition 7. However, we can show that the λ-term relation

$$\bigcap \{\mathcal{R} \in \mathsf{Adeq} \mid \widehat{\mathcal{R}} \subseteq \mathcal{R}\}$$

is itself a preadequate compatible λ-term relation, and thus identity probabilistic contextual approximation with the latter. We will not give a proof of such a fact here, as the latter is a special case of the more result proved in Lemma 12.

The reader might also have noticed that the definition of the property Adeq involves only the behaviour of λ-term relations on computations, and does not say anything about their behaviour on values. This is rather obvious, as adequacy is an operational notion based on the evaluation semantics of a program. Formally, this does *not* mean that on values $\preceq^{\mathrm{ctx}}$ coincides with the indiscrete relation 0. In fact, compatibility of $\preceq^{\mathrm{ctx}}$ forces to relate only those values that behaves appropriately when used inside computations. For instance, if $\vdash^{\mathsf{v}} v \preceq^{\mathrm{ctx}} v'$, then by compatibility (and reflexivity) of $\preceq^{\mathrm{ctx}}$ we have $\vdash^{\Lambda} vw \preceq^{\mathrm{ctx}} v'w$ too, which gives $[\![vw]\!](\mathcal{V}_\circ) \leq [\![v'w]\!](\mathcal{V}_\circ)$. Obviously, this cannot be the case for all pairs of values v, v'.

Definition 7 comes with an associated proof technique resembling a coinduction proof principle. In order to prove that a program e contextually approximates a program f, it is sufficient to exhibiting a compatible preadequate λ-term relation $\mathcal{R}$ relating e and f. Symbolically,

$$\frac{\widehat{\mathcal{R}} \subseteq \mathcal{R} \quad \mathcal{R} \in \mathsf{Adeq}}{\mathcal{R} \subseteq \preceq^{\mathrm{ctx}}}\ (\preceq^{\mathrm{ctx}}\text{-UMP})$$

We can use this proof technique to prove that $\preceq^{\mathrm{ctx}}$ is a preorder, and thus a precongruence λ-term relation. For instance, in order to show that $\preceq^{\mathrm{ctx}}$ is transitive, it sufficient to prove that $\widehat{(\preceq^{\mathrm{ctx}} \cdot \preceq^{\mathrm{ctx}})} \subseteq (\preceq^{\mathrm{ctx}} \cdot \preceq^{\mathrm{ctx}})$ and $(\preceq^{\mathrm{ctx}} \cdot \preceq^{\mathrm{ctx}}) \in \mathsf{Adeq}$. By the proof principle induced by Definition 7 we can then infer $\preceq^{\mathrm{ctx}} \cdot \preceq^{\mathrm{ctx}} \subseteq \preceq^{\mathrm{ctx}}$, i.e. transitivity of $\preceq^{\mathrm{ctx}}$.

Although $\preceq^{\mathrm{ctx}}$ has several interesting mathematical properties, it has a major drawback: proving that a program f contextually approximates a program e is oftentimes simply not doable in practice.

The reason is evident if we think to $\preceq^{ctx}$ as defined with explicit contexts. In order to check whether $\vdash^{\Lambda} e \preceq^{ctx} f$ holds, one has to study the behaviour of e and f inside any possible context of the language. In particular, it is necessary to test the behaviour of e and f when passed as input to arbitrary programs, of which we have no information about.

It is then useful to define suitable *proof techniques* for (probabilistic) contextual approximation. For us, a proof technique has the form of a λ-term relation $\mathcal{R}$ that has nicer operational properties than (probabilistic) contextual equivalence. The technique is said to be *sound* if $\mathcal{R} \subseteq \preceq^{ctx}$ and *fully abstract* if $\mathcal{R} = \preceq^{ctx}$. Besides investigating probabilistic applicative similarity as program refinement *per se*, we can think to it as a proof technique for probabilistic contextual approximation. In particular, probabilistic applicative similarity testing values for their applicative behaviour only (i.e. in functional, as opposed to argument, position) and coming with an associated coinduction proof principle, it definitely qualifies as a powerful proof technique for $\preceq^{ctx}$ (provided, of course, that it is included in $\preceq^{ctx}$).

Before giving a proof of precongruence of probabilistic applicative similarity, we remark that in this section, even more than in previous ones, we have almost never used specific feature of Λ_p. The only place where we used its semantics is in the definition of the notion of preadequacy. It comes with no surprise that we can abstract from such a notion and work with more general notions of preadequacy.

2.4 Howe's Method

In light of previous discussion, it is of paramount importance for our aims to prove (the open extension of) probabilistic applicative bisimilarity to be compatible. A direct attempt to prove compatibility of probabilistic applicative similarity requires to show $\preceq$ to be substitutive, which turns out to be problematic (see (Pitts, 2011)). Howe bypassed the problem by constructing a λ-term relation $\preceq^H$ extending $\preceq$ that is compatible by construction. The so-called *Key Lemma* states that $\preceq^H$ is indeed a simulation, and thus coincide with $\preceq$, by coinduction.

Definition 8. *Given a closed λ-term relation $\mathcal{R}$, the* Howe extension $\mathcal{R}^H$ *of $\mathcal{R}$ is inductively defined by the following rules:*

$$\frac{\Gamma \vdash^{\Lambda} e \,\widehat{\mathcal{R}^H}\, g \quad \Gamma \vdash^{\Lambda} g \,\mathcal{R}^o\, f}{\Gamma \vdash^{\Lambda} e \,\mathcal{R}^H\, f}\ \text{(H-1)} \qquad \frac{\Gamma \vdash^{v} v \,\widehat{\mathcal{R}^H}\, u \quad \Gamma \vdash^{v} u \,\mathcal{R}^o\, w}{\Gamma \vdash^{v} v \,\mathcal{R}^H\, w}\ \text{(H-2)}.$$

Unfolding Definition 8 we see that we can give an alternative characterisation of $\mathcal{R}^H$ using rules in Figure 2.5. Additionally, probabilistic applicative similarity being reflexive and transitive, we see that $\preceq^H$ satisfies nice several properties.

Lemma 4. *The following hold:*

1. $\preceq^o \subseteq \preceq^H$.
2. $\preceq^o \cdot \preceq^H \subseteq \preceq^H$ *(we refer to this property as pseudo-transitivity).*
3. $\preceq^H$ *is reflexive, compatible, and substitutive.*

We do not prove Lemma 4 here, as we will prove its generalisation in Chapter 5.

By Lemma 4, we know that $\preceq^H$ is compatible and extends $\preceq^o$, so that the restriction of $\preceq^H$ to closed terms extends $\preceq$. As a consequence, in order to prove probabilistic applicative similarity to be compatible, it is sufficient to show that the restriction of $\preceq^H$ to closed terms is included in $\preceq$. That is:

$$\frac{\vdash^{\Lambda} e \preceq^H f}{\vdash^{\Lambda} e \preceq f} \qquad \frac{\vdash^{v} v \preceq^H w}{\vdash^{v} v \preceq w}$$

$$\frac{\Gamma, x \vdash^{\mathsf{v}} x \mathcal{R} v}{\Gamma, x \vdash^{\mathsf{v}} x \mathcal{R}^H v} \text{ (H-var)}$$

$$\frac{\Gamma, x \vdash^{\Lambda} e \mathcal{R}^H f \quad \Gamma \vdash^{\mathsf{v}} \lambda x.f \mathcal{R} v}{\Gamma \vdash^{\mathsf{v}} \lambda x.e \,\widehat{\mathcal{R}}\, v} \text{ (H-abs)} \qquad \frac{\Gamma \vdash^{\mathsf{v}} v \mathcal{R}^H v' \quad \Gamma \vdash^{\mathsf{v}} w \mathcal{R}^H w' \quad \Gamma \vdash^{\Lambda} v'w' \mathcal{R} e}{\Gamma \vdash^{\Lambda} vw \mathcal{R}^H e} \text{ (H-app)}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} v \mathcal{R}^H w \quad \Gamma \vdash^{\Lambda} \mathbf{return}\ w \mathcal{R} e}{\Gamma \vdash^{\Lambda} \mathbf{return}\ v \mathcal{R}^H e} \text{ (H-val)}$$

$$\frac{\Gamma \vdash^{\Lambda} e \mathcal{R}^H e' \quad \Gamma, x \vdash^{\Lambda} f \mathcal{R}^H f' \quad \Gamma \vdash^{\Lambda} \mathbf{let}\ x = e'\ \mathbf{in}\ f' \mathcal{R} g}{\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f \mathcal{R}^H g} \text{ (H-let)}$$

$$\frac{\Gamma \vdash^{\Lambda} e \mathcal{R}^H e' \quad \Gamma \vdash^{\Lambda} f \mathcal{R}^H f' \quad \Gamma \vdash^{\Lambda} e'\ \mathbf{or}\ f' \mathcal{R} g}{\Gamma \vdash^{\Lambda} e\ \mathbf{or}\ f \mathcal{R}^H g} \text{ (H-op)}$$

Figure 2.5: Howe extension of $\mathcal{R}$.

By coinduction, it is thus sufficient to prove that the restriction of $\preceq^H$ to closed terms is a probabilistic applicative simulation. This is the content of the so-called *Key Lemma*.

Lemma 5 (Key Lemma)**.** *The restriction of $\preceq^H$ to closed terms is a probabilistic applicative simulation.*

Concretely, the Key Lemma requires to prove the following implications, for all closed computations e, f, and all closed values v, w:

$$\vdash^{\Lambda} e \preceq^H f \implies [\![e]\!]\, \hat{D}(\preceq^H_{\mathcal{V}})\, [\![f]\!]$$
$$\vdash^{\mathsf{v}} v \preceq^H w \implies \forall u \in \mathcal{V}_\circ.\ \vdash^{\Lambda} vu \preceq^H wu.$$

Proving the second implication is straightforward, whereas proving the first one, i.e. condition (app 1) in Definition 3, is non-trivial. First of all, we observe that attempting a proof of (app 1) by induction on e, we get stuck in the case for sequencing, i.e. for $e = (\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2)$. In fact, treating such a case requires to deal with the open computation $x \vdash^{\Lambda} e_2$, and reasoning on instances of the form $e_2[x := v]$ does allow us to use the induction hypothesis.

Standard proofs of the Key Lemma for the pure λ-calculus proceeds by induction on the big-step semantics of the language, which is something not possible for Λ_p due to its infinitary operational semantics. To solve this problem we notice that $\hat{D}$ satisfies the following induction-like principle. For any ω-chain $(\mu_n)_n$ in DX, relation $\mathcal{R}$ on X, and $\nu \in DX$, we have:

$$(\forall n \geq 0.\ \mu_n\, \hat{D}\mathcal{R}\, \nu) \implies \sup_n \mu_n\, \hat{D}\mathcal{R}\, \nu.$$

As a consequence, we can attempt to prove (app 1) showing

$$\forall n \geq 0.\ [\![e]\!]_n\, \hat{D}(\preceq^H_{\mathcal{V}})\, [\![f]\!],$$

and proceeding by induction on n. The case for $n = 0$ trivially holds, since for any $\nu \in DX$ and relation $\mathcal{R}$ on X we have $\bot\, \hat{D}\mathcal{R}\, \nu$.

Let us now look at the $(n+1)$-case. We proceed by case analysis according to the definition of $[\![-]\!]_n$, and relying on the definition of $\preceq^H$ given by rules in Figure 2.5, which are syntax-directed. Let us examine the most interesting cases.

- Suppose $e = \mathbf{return}\ v$. We assume $\vdash^{\Lambda} \mathbf{return}\ v \preceq^{H} f$ and show $[\![\mathbf{return}\ v]\!]_{n+1}\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![f]\!]$. By rules in Figure 2.5, the former must be the conclusion of a derivation of the form:

$$\frac{\vdash^{\mathsf{v}} v \preceq^{H} w \quad \vdash^{\Lambda} \mathbf{return}\ w \preceq f}{\vdash^{\Lambda} \mathbf{return}\ v \preceq^{H} f}\ (\text{H-val}).$$

  Since $\preceq$ is a simulation, $\mathbf{return}\ w \preceq g$ implies $\eta(w)\ \hat{D}(\preceq)\ [\![f]\!]$. If it is the case that $\vdash^{\mathsf{v}} v \preceq^{H} w$ implies $\eta(v)\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ \eta(w)$, the we could conclude the thesis using Lemma 4 (pseudo-transitivity). That is indeed the case. In fact, the operation $\hat{D}$ satisfies the following general law. For all sets X, Y, and relation $\mathcal{R} \subseteq X \times Y$ we have:

$$\forall x, x' \in X.\ x\ \mathcal{R}\ x' \implies \eta(x)\ \hat{D}\mathcal{R}\ \eta(x').$$

- Suppose $e = (\mathbf{let}\ x = g\ \mathbf{in}\ g')$ and assume $\vdash^{\Lambda} \mathbf{let}\ x = g\ \mathbf{in}\ g' \preceq^{H} f$. By very definition of $[\![-]\!]_n$, we have to show:

$$[\![g'[x := -]]\!]^{\dagger}_{n} [\![g]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![f]\!].$$

  By rules in Figure 2.5, $\vdash^{\Lambda} \mathbf{let}\ x = g\ \mathbf{in}\ g' \preceq^{H} f$ must be the conclusion of a derivation of the form:

$$\frac{\vdash^{\Lambda} g \preceq^{H} h \quad x \vdash^{\Lambda} g' \preceq^{H} h' \quad \vdash^{\Lambda} \mathbf{let}\ x = h\ \mathbf{in}\ h' \preceq f}{\vdash^{\Lambda} \mathbf{let}\ x = g\ \mathbf{in}\ g' \preceq^{H} f}\ (\text{H-let}).$$

  We can apply the induction hypothesis on $\vdash^{\Lambda} g \preceq^{H} h$, obtaining $[\![g]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![h]\!]$. However, we cannot do the same on $x \vdash^{\Lambda} g' \preceq^{H} h'$. Nonetheless, we notice that since $\preceq^{H}$ is substitutive, we have:

$$\forall v, w \in \mathcal{V}_{\circ}.\ \vdash^{\mathsf{v}} v \preceq^{H} w \implies \vdash^{\Lambda} g'[x := v] \preceq^{H} h'[x := w].$$

  At this point the induction hypothesis can be applied, obtaining $[\![g'[x := v]]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![h'[x := w]]\!]$. Stated otherwise, the functions $[\![g'[x := -]]\!]_n, [\![h'[x := -]]\!] : \mathcal{V}_{\circ} \to D\Lambda_{\circ}$ satisfy the following generalised monotonicity condition:

$$\vdash^{\mathsf{v}} v \preceq^{H} w \implies [\![g'[x := v]]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![h'[x := w]]\!].$$

  What we need to prove in order to conclude the thesis — again by Lemma 4 (pseudo-transitivity), noticing that $\vdash^{\Lambda} \mathbf{let}\ x = h\ \mathbf{in}\ h' \preceq f$ implies $[\![\mathbf{let}\ x = h\ \mathbf{in}\ h']\!]\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![f]\!]$ — is that generalised monotonicity is preserved by the operation $-^{\dagger}$. That is, we wish the following general law to hold. For all programs e, e' and computations $x \vdash^{\Lambda} f, f'$:

$$\frac{\forall v, v' \in \mathcal{V}_{\circ}.\ \vdash^{\mathsf{v}} v \preceq^{H} v' \implies [\![f[x := v]]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![f'[x := v']]\!]}{[\![e]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![e']\!] \implies [\![f[x := -]]\!]^{\dagger}_{n} [\![e]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![f'[x := -]]\!]^{\dagger} [\![e']\!]}$$

  Notice that the above law can rewritten as:

$$\frac{\forall v, v' \in \mathcal{V}_{\circ}.\ \vdash^{\mathsf{v}} v \preceq^{H} v' \implies [\![f[x := v]]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![f'[x := v']]\!]}{[\![e]\!]_n\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![e']\!] \implies [\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!]_{n+1}\ \hat{D}(\preceq^{H}_{\mathcal{V}})\ [\![\mathbf{let}\ x = e'\ \mathbf{in}\ f']\!]}$$

  The above condition is nothing but a specific instance of a more general (algebraic) law describing the general behaviour of $\hat{D}$. For all sets X, Y, subdistributions $\mu \in DX$, $\nu \in DY$, functions $f, g : X \to DY$, and relations $\mathcal{R}, \mathcal{S}$ over X and Y, respectively, we have:

$$\frac{\forall x, x' \in X.\ x\ \mathcal{R}\ x' \implies f(x)\ \hat{D}\mathcal{S}\ g(x')}{\mu\ \hat{D}\mathcal{R}\ \nu \implies f^{\dagger}(\mu)\ \hat{D}\mathcal{S}\ g^{\dagger}(\nu)}$$

  Providing that $\hat{D}$ satisfies the above property, is non-trivial, and essentially requires a modification of Strassen's Theorem to subdistribution. We will prove that in a more abstract way in Chapter 4.

- Finally, we consider the case for $e = g \ \mathbf{or}\ g'$. This case can be proved as previous ones, simply noticing that the following general law holds, for all sets X, subdistributions $\mu, \mu', \nu, \nu' \in DX$, and relation $\mathcal{R}$ on X:

$$\frac{\mu \ \hat{D}\mathcal{R} \ \mu' \quad \nu \ \hat{D}\mathcal{R} \ \nu'}{\mu \oplus \nu \ \hat{D}\mathcal{R} \ \mu' \oplus \nu'}$$

Summing up, the proof of the Key Lemma is rather straightforward once we know that the map $\hat{D}$ satisfies some general properties, which are totally independent of Λ_p. The reader may have noticed that such properties can, in principle, be defined on any relation lifting operation. This is indeed the case, as we are going to discuss.

2.5 Final Discussion

Looking back at what we have done in this chapter, we see that our definitions and results do not really depend neither on subdistributions (and thus on the subdistribution functor D) nor on the map $\hat{D}$ in an essential way. Rather, they follow from suitable structural properties of D and $\hat{D}$. Such structural properties can be summarised as follows:

1. In order to give operational semantics to Λ_p, we noticed that the set DX comes with an ω-cppo structure, as well as with maps $\eta : X \to DX$ and $f^\dagger : DX \to DY$, for any map $f : X \to DY$. Such maps are required to be monotone and continuous. Abstractly, this means that the subdistribution functor D is part of a monad carrying a suitable domain structure, and that the latter properly interacts with the monad structure of D.

2. In order to define probabilistic applicative similarity, we introduced a suitable map $\hat{D}$ lifting relations over sets X to relations over DX. The map is required to be monotone and to quasi-preserve the identity relation as well as relation composition. That guaranteed probabilistic applicative similarity to be a preorder.

3. Finally, knowing that $\hat{D}$ satisfies a suitable induction-like principle, and some suitable conditions with respect to the maps η and $f^\dagger$ (meaning that $\hat{D}$ properly interacts with the monad structure of D), we were able to use Howe's method to prove compatibility of $\preceq$, hence concluding the latter to be a precongruence.

There are many other functors other than D that satisfy these properties, the powerset functor being an example of such a functor.

However, to model effectful calculi we also need suitable effect-triggering operations, which act as sources of side effects. This is done in Λ_p using the probabilistic choice operation **or**. We observed that the choice of operations does not really matter, as far as they can be interpreted as operations on D, and behave in a suitable way. Again, this is not specific to the functor D (think, for instance, to a nondeterministic choice operation interpreted as set-theoretic union on the powerset functor).

2.5.1 What's Next?

In next chapters we will take advantage of the above observations, defining an abstract calculus parametric over collections of operation symbols and monads. We will then define an abstract notion of applicative similarity, which we call *effectful applicative similarity*, and prove it to be a precongruence using a generalisation of Howe's method.

The resulting theory is rich, allowing to describe a large family of effectful calculi, including probabilistic and nondeterministic calculi, as well as calculi with input/output, global states, exceptions, and

combinations thereof. Additionally, the mathematical framework we will develop to define effectful applicative similarity provides a powerful formal tool to study several notions of program equivalence and refinement. Among them, we will define and analyse generalisation of contextual approximation (resp. equivalence), CIU approximation (resp. equivalence), and normal form similarity (resp. bisimilarity) to effectful calculi.

Chapter 3

A Computational Calculus for Algebraic Effects

> [. . .] what I will call a *symbolic calculus*; which, with certain symbols and certain laws of combination, is *symbolic algebra*: an art, not a science; and an apparently useless art, except as it may afterwards furnish the grammar of a science.
>
> Augustus de Morgan, Trigonometry and Double Algebra

In this chapter we introduce the main calculus we will use in this dissertation, namely a call-by-value λ-calculus (G. Plotkin, 1975) with *algebraic operations* in the style of Plotkin and Power (G. D. Plotkin & Power, 2001, 2002, 2003). We call such a calculus Λ_Σ. Λ_Σ is a calculus with *algebraic effects* (G. D. Plotkin & Power, 2002, 2003), meaning that effects are produced (and can only be produced) by suitable operations. In his seminal work on notions of computation (Moggi, 1989, 1991), Moggi gave a unified account of computational effects as strong monads. An example is provided by the calculus Λ_p of Chapter 2, where we used the subdistribution monad to model (the result of) probabilistic computations. However, notions of computation (and thus monads) describe the effects produced by computations only, and do not prescribe the existence of computational primitives to produce such effects. To do so, one needs to have operations in the language to actually make effects happen. This in indeed the case in Λ_p, where probabilistic computations are produced by the probabilistic choice operation **or**, and then propagated using sequencing. The calculus Λ_Σ builds upon this observation. Therefore, Λ_Σ relies on monads to model computational effects, and on (algebraic) operations to produce such effects.

These features qualify Λ_Σ as a minimal computational λ-calculus with algebraic effects. On a syntactic level, Λ_Σ is parametrised by a signature Σ of (uninterpreted) operation symbols. Examples are operation symbols meant to model probabilistic and nondeterministic choices, input and output primitives, as well as primitives to read and write from a global store, or rise exceptions. On a semantic level, we use monads (on Set) to define an abstract operational semantics for Λ_Σ, which we called *monadic operational semantics*. Following Plotkin and Power (G. D. Plotkin & Power, 2001, 2002), we interpret operation symbols as *algebraic operations* on monads. From an operational perspective, algebraicity of an operation means that the (operational) behaviour of the operation is independent of its arguments, or,

equivalently, of the environment in which it is evaluated. Such a restriction allows to structure effectful computations and gives a tight control on the way effects are produced. All the operation symbols previously mentioned can be interpreted as algebraic operations on suitable monads. An example of a non-algebraic, yet monadic effect (Benton, Hughes, & Moggi, 2000) is provided by continuations[1](Hyland et al., 2007), whereas exception handlers[2] are an example of non-algebraic, yet natural operations.

This chapter is organised as follows: we first recall some necessary background notions on monads and give examples of relevant notions of computation (i.e. monads modelling interesting computational effects). We then introduce algebraic operations abstractly, giving examples of both algebraic and non-algebraic operations. Lastly, we introduce the calculus Λ_Σ and define its monadic operational semantics. Such a semantics is defined in the same spirit of the evaluational semantics of Λ_p. This means that in order to address infinitary behaviours, we need to require monads to carry a suitable domain structure and operation symbols to be monotone and continuous.

3.1 Monads and Algebraic Operations

We assume the reader to be familiar with basic category theory (MacLane, 1971) and basic domain theory (Abramsky & Jung, 1994). In particular, we assume the reader to know the notions of a category and of a functor, as well as the notions of an ω-pointed complete partial order (ω-cppo, for short), and of a monotone and continuous function. Throughout this dissertation, we will work with the categories Set (of sets and functions) and Rel (of sets and relations), only[3]. As a consequence, the amount of category theory we will use is minimal: we may say that we use the *vocabulary* of category theory, rather than category theory itself.

Concerning notation, we try to use the same notation used in (MacLane, 1971) for category theory, and in (Abramsky & Jung, 1994) for domains. In order to improve readability, we use the notation $(x_n)_n$ to denote an ω-chain $x_0 \sqsubseteq \cdots \sqsubseteq x_n \sqsubseteq \cdots$ in a domain $(X, \sqsubseteq, \bot)$, hence assuming n to range over elements of $\mathbb{N}$. As a minor difference with standard notation, we denote the composition of a morphism g with a morphism f (with appropriate source and target) in a category $\mathbb{C}$ as $g \cdot f$, rather than $g \circ f$. We denote by $1_X : X \to X$ the identity morphism, omitting subscripts when unambiguous.

We begin our technical exposition recalling the notion of a monoidal category (MacLane, 1971).

Definition 9. *A* monoidal category $\langle \mathbb{C}, \otimes, I, \alpha, \lambda, \rho \rangle$ *consists of a category* $\mathbb{C}$ *equipped with a bifunctor* $\otimes : \mathbb{C} \otimes \mathbb{C} \to \mathbb{C}$*, called the* tensor product *of* $\mathbb{C}$*, a distinguished object* I*, called the unit of* $\mathbb{C}$*, and three natural isomorphisms:*

$$\begin{aligned} \alpha_{X,Y,Z} &: X \otimes (Y \otimes Z) \cong (X \otimes Y) \otimes Z \\ \lambda_X &: I \otimes X \cong X \\ \rho_X &: X \otimes X \cong X. \end{aligned}$$

The natural isomorphisms are required to satisfy the following coherence conditions:

$$\begin{aligned} 1_X \otimes \alpha_{Y,Z,W} \cdot \alpha_{X,Y\otimes Z,W} \cdot (\alpha_{X,Y,Z} \otimes 1_W) &= \alpha_{X,Y,Z\otimes W} \cdot \alpha_{X\otimes Y,Z,W} \\ (1_X \otimes \lambda_Y) \cdot \alpha_{X,I,Y} &= \rho_X \otimes 1_Y. \end{aligned}$$

[1] According to (Hyland et al., 2007) continuations involve *logical*, as opposed to *algebraic*, operations.

[2] Non-algebraicity of exception handlers led to the development of more general theories of algebraic effects, where algebraic operations are combined with specific algebraic morphisms manipulating the control flow of programs. Such morphisms are called *handlers*, and the resulting theory goes under the name of *theory of effects and handlers* (Bauer & Pretnar, 2015; G. D. Plotkin & Pretnar, 2013; Pretnar, 2015).

[3] In Chapter 10 and Chapter 11 we will study a refinement of Rel, where relations take values over arbitrary quantales (Rosenthal, 1990).

A symmetric monoidal category is a monoidal category $\mathbb{C}$ equipped with the additional natural isomorphism $\gamma_{X,Y} : X \otimes Y \cong Y \otimes X$ subject to the coherence condition:

$$\gamma_{Y,X} = \gamma_{X,Y} = 1_{X \otimes Y}.$$

Our primary example of a symmetric monoidal category is the category Set, with tensor product given by cartesian product and unit object given by the terminal object. We now recall the notion of a strong monad (Kock, 1972; Moggi, 1991). For our purpose, it is more convenient to work with the equivalent notion of a *(strong) Kleisli triple*.

Definition 10. *Given a monoidal category $\langle \mathbb{C}, \otimes, I, \alpha, \lambda, \rho \rangle$, a* strong Kleisli triple *on $\mathbb{C}$ is a triple $\mathbb{T} = \langle T, \eta, -^* \rangle$ consisting of:*

1. *A map T over objects of $\mathbb{C}$, called the* carrier *of $\mathbb{T}$.*
2. *A $\mathbb{C}$-object-indexed family η of morphisms $\eta_X : X \to TX$, called the* unit *of $\mathbb{T}$ on X.*
3. *An operation $-^*$, called* strong Kleisli extension*, mapping a $\mathbb{C}$-morphism $f : Z \otimes X \to TY$ to $f^* : Z \otimes TX \to TY$. The morphism f^* is called the* strong Kleisli extension *of f.*

These data must satisfy the following identities, where $f : W \otimes X \to TY$ and $g : W' \otimes Y \to TZ$:

$$\begin{aligned} f^* \cdot (1_V \otimes \eta_X) &= f \\ (\eta_X \cdot \lambda_X)^* &= \lambda_{TX} \\ (g^* \cdot (1_U \otimes f) \cdot \alpha_{U,V,X})^* &= g^* \cdot (1_U \otimes f^*) \cdot \alpha_{U,V,TX}. \end{aligned}$$

If we relax Definition 10 replacing $-^*$ with an operation $-^\dagger$, called *Kleisli extension*, mapping a $\mathbb{C}$-morphism $f : X \to TY$ to $f^\dagger : TX \to TY$ subject to the identities (where $f : X \to TY$ and $g : Y \to TZ$)

$$\begin{aligned} \eta_X^\dagger &= 1_{TX} \\ f^\dagger \cdot \eta_X &= f \\ g^\dagger \cdot f^\dagger &= (g^\dagger \cdot f)^\dagger, \end{aligned}$$

then we obtain the notion of a *Kleisli triple*. With the exception of Chapter 9 to Chapter 12, throughout this dissertation we will work with Kleisli triples ony.

We immediately notice that Kleisli triples can be defined on arbitrary categories (and not only on monoidal ones), and that $f^\dagger$ can be defined in terms of $-^*$ and the unit object. Moreover, (strong) Kleisli triples and (strong) monads (MacLane, 1971) are in a bijective correspondence. Each Kleisli triple $\langle T, \eta, -^\dagger \rangle$ yields a monad $\langle T, \eta, \mu \rangle$ by defining:

$$\begin{aligned} Tf &\triangleq (\eta_Y \cdot f)^\dagger \\ \mu_X &\triangleq (1_{TX})^\dagger. \end{aligned}$$

Vice versa, for any monad (T, η, μ) we define

$$f^\dagger : TX \to TY \triangleq \mu_Y \cdot Tf.$$

Finally, if $\langle T, \eta, \mu \rangle$ has strength $t_{X,Y} : X \otimes TY \to T(X \otimes Y)$, then we can define $f^* : V \otimes TX \to TY$ as $\mu_Y \cdot Tf \cdot t_{V,X}$. Vice versa, we can define $t_{X,Y}$ as $\eta^*_{X \otimes Y}$.

In light of these equivalences, we will abuse terminology and simply refer to (strong) monads to denote both (strong) monads and (strong) Kleisli triples. We also recall that every monad on Set is

strong. Finally, unless explicitly mentioned, all functors and monads will be on Set. Accordingly, when speaking of functors and monads we tacitly assume to refer to Set endofunctors and monads on Set, respectively.

When working with an arbitrary monad, we will use the traditional notation $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ to denote it. Such a notation, however, is not very handy when dealing with either specific monads or with multiple monads at the same time. To fix this issue, we sometimes use the following notation and convention. Given a monad $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$, we always use capital Latin letter to denote its carrier, whereas the name of the monad is given by the same letter written in 'blackboard style'. This might in principle create confusion with the notation used for sets, i.e. Latin letters $X, Y, W, Z, X', Y', \ldots$, but the context will resolve any ambiguity (either implicitly or explicitly). The unit of $\mathbb{T}$ will be denoted by T — i.e. with same letter used to denote its carrier, but written in small capital style — whereas we use the notation $-^{\mathbb{T}}$ in place of $-^\dagger$. For instance, when working with two monads $\mathbb{T}, \mathbb{U}$ we denote them as $\mathbb{T} = \langle T, \mathrm{T}, -^{\mathbb{T}} \rangle$ and $\mathbb{U} = \langle U, \mathrm{U}, -^{\mathbb{U}} \rangle$, respectively. Finally, when unambiguous we omit subscripts. For instance, for a fixed set X and a monad $\mathbb{T}$ we write $\eta(x)$ (reps. $\mathrm{T}(x)$) in place of $\eta_X(x)$ (resp. $\mathrm{T}_X(x)$).

3.1.1 Relevant Examples

Before introducing algebraic operations, we give examples of monads modelling relevant computational effects.

Example 3 (Partiality monad)**.** We model partial computations using the partiality (also called maybe) monad $\mathbb{M} = \langle M, \mathrm{M}, -^{\mathbb{M}} \rangle$. The carrier MX of $\mathbb{M}$ is defined as $\{just\ x \mid x \in X\} \cup \{\bot\}$, where $\bot$ is a special symbol denoting divergent computations. The unit and Kleisli extension of $\mathbb{M}$ are defined as follows:

$$\mathrm{M}(x) \triangleq just\ x$$
$$f^{\mathbb{M}}(\mathfrak{x}) \triangleq \begin{cases} f(x) & \text{if } \mathfrak{x} = just\ x, \\ \bot & \text{if } \mathfrak{x} = \bot. \end{cases}$$

We notice that we can combine the maybe monad with any other monad $\mathbb{T}$, as an instance of the so-called *sum* of effects (Hyland et al., 2006), an operation on Lawvere theories (Hyland & Power, 2007; W. F. Lawvere, 2004) introduced to combine algebraic theories. Since our focus is on monads, a general treatment of such an operation is outside the scope of this dissertation. Nevertheless, for our purposes the following result is enough.

Proposition 3. *Given a monad $\mathbb{T} = \langle T, \mathrm{T}, -^{\mathbb{T}} \rangle$, define the* sum *$\mathbb{T}\mathbb{M}$ of $\mathbb{T}$ and $\mathbb{M}$ as the triple $\langle TM, \mathrm{TM}, -^{\mathbb{T}\mathbb{M}} \rangle$ defined as follows:*

$$TMX \triangleq T(MX)$$
$$\mathrm{TM}_X \triangleq \mathrm{T}_{MX} \cdot \mathrm{M}_X$$
$$f^{\mathbb{T}\mathbb{M}} \triangleq (f_M)^{\mathbb{T}},$$

where, for a function $f : X \to TMY$ we define $f_M : MX \to TMY$ by:

$$f_M(\mathfrak{x}) \triangleq \begin{cases} \mathrm{T}_{MX}(\bot) & \text{if } \mathfrak{x} = \bot \\ f(x) & \text{if } \mathfrak{x} = just\ x. \end{cases}$$

Then $\mathbb{T}\mathbb{M}$ is a monad.

Proving Proposition 3 is a straightforward exercise (the reader can also consult (Hyland et al., 2006)). ⊠

Example 4 (Exception monad)**.** Let $\mathcal{E}$ be a set of exception symbols. The exception monad $\mathbb{E} = \langle E, \mathrm{E}, -^{\mathbb{E}} \rangle$, over $\mathcal{E}$, has carrier $E(X) \triangleq X + \mathcal{E}$ (for simplicity we assume $\mathcal{E}$ to be disjoint from X), whereas E and $-^{\mathbb{E}}$ are defined as follows:

$$\mathrm{E}(x) \triangleq x$$
$$f^{\mathbb{E}}(\mathfrak{x}) \triangleq \begin{cases} f(x) & \text{if } \mathfrak{x} = x, \\ \mathsf{e} & \text{if } \mathfrak{x} = \mathsf{e} \in \mathcal{E}. \end{cases}$$

The exception monad is used to model total computations with exceptions, an element $\mathsf{e} \in \mathcal{E}$ denoting the exception raised by a computation. It is easy to see that $\mathbb{E}$ has the same structure of $\mathbb{M}$, so that we can replace $\mathbb{M}$ with $\mathbb{E}$ in Proposition 3. In particular, we can model partial computations with exceptions using the sum $\mathbb{ME}$.

⊠

Example 5 (Output monad)**.** Let $\mathcal{A}$ be a given alphabet and let $\mathcal{A}^*$ be the set of finite strings over $\mathcal{A}$. We denote by ε the empty string, and write uw for the concatenation of the strings u and w The output monad $\mathbb{O} = \langle O, \mathrm{o}, -^{\mathbb{O}} \rangle$ has carrier $O(X) \triangleq \mathcal{A}^* \times X$, whereas o and $-^{\mathbb{O}}$ are defined as follows:

$$\mathrm{o}(x) \triangleq (\varepsilon, x)$$
$$f^{\mathbb{O}}(u, x) \triangleq (uw, y),$$

where $(w, y) = f(x)$. The output monad models total computations with outputs. An element (u, x) represents the result of a computation outputting the string u and returning the element x.

We can extend $\mathbb{O}$ to take into account divergent computations. To do so, we have to take into account divergent computations outputting infinitely many symbols (such as the recursively defined program $e = \mathbf{print}_{\mathrm{c}}.e$). Let $\mathcal{A}^\infty$ be the set of finite and infinite strings over $\mathcal{A}$. We extend the definition of string concatenation to infinite strings by defining uw to be u if u is infinite. The partial output monad $\mathbb{O}^\infty = \langle O^\infty, \mathrm{o}^\infty, -^{\mathbb{O}^\infty} \rangle$ has carrier $O^\infty(X) \triangleq \mathcal{A}^\infty \times MX$, whereas o^∞ and $-^{\mathbb{O}^\infty}$ are defined as follows:

$$\mathrm{o}^\infty(x) \triangleq (\varepsilon, \mathit{just}\ x)$$
$$f^{\mathbb{O}^\infty}(u, \mathfrak{x}) \triangleq \begin{cases} (u, \bot) & \text{if } \mathfrak{x} = \bot \\ (uw, \mathfrak{y}) & \text{if } \mathfrak{x} = \mathit{just}\ x \text{ and } f(x) = (w, \mathfrak{y}). \end{cases}$$

⊠

Example 6 (Powerset monad)**.** The non-empty powerset monad $\mathbb{F} = \langle F, \mathrm{F}, -^{\mathbb{F}} \rangle$ has carrier $FX \triangleq \{\mathfrak{x} \subseteq X \mid \mathfrak{x} \neq \emptyset\}$, whereas F and $-^{\mathbb{F}}$ are defined by:

$$\mathrm{F}(x) \triangleq \{x\}$$
$$f^{\mathbb{F}}(\mathfrak{X}) \triangleq \bigcup_{x \in \mathfrak{X}} f(x).$$

The non-empty powerset monad is used to model total nondeterministic computations, with the rationale that a set $\mathfrak{X} \in FX$ denotes the set of possible (nondeterministic) outcomes of a computation. In order to model partial computations one can use the full powerset monad $\mathbb{P} = \langle \mathcal{P}, \mathrm{P}, -^{\mathbb{P}} \rangle$, where P and $-^{\mathbb{P}}$ are defined as for $\mathbb{F}$, or use $\mathbb{FM}$, the sum $\mathbb{F}$ and $\mathbb{M}$. In the former case we use the empty set to model the result of a purely divergent computation, whereas in the latter case we explicitly keep track of divergence using the symbol $\bot$. We prefer the latter alternative, as it provides a finer analysis of pure nondeterminism[4].

[4]For instance, consider the programs $e \triangleq (\mathbf{return}\ v)\ \mathbf{or}\ \Omega$ and $f \triangleq \mathbf{return}\ v$, where $\mathbf{or}$ is a nondeterministic choice operation. Modelling nondeterministic computations using the full powerset monad $\mathbb{P}$, we see that both both e and f produce the set $\{v\}$ as outcome. However, if we instead model nondeterministic computations using the monad $\mathbb{FM}$, then we see that e produces the set $\{\mathit{just}\ v, \bot\}$ as outcome, whereas f produces $\{v\}$. Both e and f may converge, but, contrary to e, f must also converge.

⊠

Example 7 (Distribution monad)**.** The (discrete) distribution monad $\mathbb{D} = \langle D, \mathrm{D}, -^{\mathbb{D}} \rangle$ has carrier $\mathbb{D}X \triangleq \{\mu : X \to [0,1] \mid \sum_{x \in \mathrm{supp}(\mu)} \mu(x) = 1\}$, where $\mathrm{supp}(\mu) \triangleq \{x \in X \mid \mu(x) > 0\}$ is the support of the distribution μ). The maps D and $-^{\mathbb{D}}$ are defined as follows:

$$\mathrm{D}(x)(y) \triangleq \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$$
$$f^{\mathbb{D}}(\mu)(y) \triangleq \sum_{x \in X} \mu(x) \cdot f(x)(y).$$

Sometimes it is useful to write a distribution μ as a weighted formal sum. That is, we write μ as the sum[5] $\sum_{i \in I} p_i \cdot x_i$ such that $\mu(x) = \sum_{x_i = x} p_i$. Accordingly, we can write $\mathrm{D}(x)$ as $1 \cdot x$. To improve readability[6], for a distribution $\mu \in DX$, we will often write $\sum_{x \in X} \mu(x)$ in place of $\sum_{x \in \mathrm{supp}(\mu)} \mu(x)$.

We have already seen that $\mathbb{D}$ does not take into account partiality. Probabilistic partial computations are modelled using the monad $\mathbb{D}\mathbb{M}$, the sum of $\mathbb{D}$ and $\mathbb{M}$. Another possibility is to work with the subdistribution monad $\mathbb{D}^{\leq 1}$, which is defined as $\mathbb{D}$ but requiring the weaker condition $\sum_{x \in X} \mu(x) \leq 1$. Although we will work with $\mathbb{D}\mathbb{M}$, we notice that $\mathbb{D}^{\leq 1}$ is isomorphic to $\mathbb{D}\mathbb{M}$, the bijection $\varphi : \mathcal{D}^{\leq 1}X \to DMX$ being defined by:

$$\varphi(\mu)(\bot) \triangleq 1 - \mu(X)$$
$$\varphi(\mu)(\mathit{just}\ x) \triangleq \mu(x)$$
$$\varphi^{-1}(\mu)(x) \triangleq \mu(\mathit{just}\ x).$$

⊠

Example 8 (Global state monad)**.** L Let $\mathcal{L}$ be a set of public location names. For simplicity we assume locations to store bits (later, we will allow locations to store more general values). A store (or state) is a function $\sigma : \mathcal{L} \to \{0,1\}$. We write S for the set of stores $\{0,1\}^{\mathcal{L}}$. The global state monad $\mathbb{G} = \langle G, \mathrm{G}, -^{\mathbb{G}} \rangle$ has carrier $GX \triangleq (X \times S)^S$, whereas G and $-^{\mathbb{G}}$ are defined by:

$$\mathrm{G}(x)(\sigma) \triangleq (x, \sigma)$$
$$f^{\mathbb{G}}(\alpha)(\sigma) \triangleq f(x')(\sigma'),$$

where $\alpha(\sigma) = (x', \sigma')$. An element of GX is thus a function α mapping a state σ to a pair (x', σ') according to the following rationale: α represents an imperative computation that when executed in the initial state σ ends in the final state σ', returning the result x. We can combine the global state monad with the partiality monad, obtaining the monad $\mathbb{M} \otimes \mathbb{G}$ whose carrier is $(M \otimes G)X \triangleq M(X \times S)^S$. Actually, the global state monad can be combined with any monad $\mathbb{T}$ using the so-called *tensor* of effects (Hyland et al., 2006). As for the sum of effects, the formal definition of the tensor operation is defined in terms of Lawvere theories, and thus it is outside the scope of this dissertation. Nonetheless, for our purposes the following result is enough.

Proposition 4. *Given a monad $\mathbb{T} = \langle T, \mathrm{T}, -^{\mathbb{T}} \rangle$ the tensor $\mathbb{T} \otimes \mathbb{G}$ of $\mathbb{T}$ and $\mathbb{G}$ is the triple $\langle T \otimes G, \mathrm{T} \otimes \mathrm{G}, -^{\mathbb{T} \otimes \mathbb{G}} \rangle$ defined as follows:*

$$(T \otimes G)X \triangleq T(S \times X)^S$$
$$(\mathrm{T} \otimes \mathrm{G})_X = \mathsf{curry}\ \mathrm{T}_{S \times X}$$
$$f^{\mathbb{T} \otimes \mathbb{G}}(\alpha)(\sigma) = (\mathsf{uncurry}\ f)^{\mathbb{T}}(\alpha)(\sigma),$$

[5] For simplicity, we write only those p_is such that $p_i > 0$.

[6] Additionally, we will mostly work with countable sets (such as the set of values), so that $\sum_{x \in X} \mu(x)$ will be always defined.

where for $f : X \to Y^Z$ *and* $g : Z \times X \to Y$ *we have* uncurry $f : Z \times X \to Y$ *and* curry $g : X \to Y^Z$*. Then* $\mathbb{T} \otimes \mathbb{G}$ *is a monad.*

Tedious calculations show that $\mathbb{T} \otimes \mathbb{G}$ is indeed a monad, but we refer to (Hyland et al., 2006) for a proper treatment. We notice that tensoring $\mathbb{G}$ with $\mathbb{DM}$ we obtain a monad for probabilistic imperative computations, whereas tensoring $\mathbb{G}$ with $\mathbb{FM}$ we obtain a monad for nondeterministic imperative computations. ⊠

Example 9 (Cost monad)**.** The (total) cost (also known as ticking or improvement (Sands, 1998)) monad $\mathbb{C}_0 = \langle C_0, \mathrm{c}_0, -^{\mathbb{C}_0} \rangle$ has carrier $C_0 X \triangleq \mathbb{N} \times X$, whereas c_0 and $-^{\mathbb{C}_0}$ are defined as follows:

$$\mathrm{c}_0(x) \triangleq (0, x)$$
$$f^{\mathbb{C}_0}(n, x) \triangleq (n + m, y),$$

where $f(x) = (m, y)$. The (total) cost monad is used to model the cost of (total) computations. An element (n, x) models the result of a computation outputting the value x with cost n (the latter being an abstract notion that can be instantiated to e.g. the number of reduction steps performed). In order to model the cost of partial computations, we consider the monad $\mathbb{C} = \langle C, \mathrm{c}, -^{\mathbb{C}} \rangle$ defined as:

$$CX \triangleq M(\mathbb{N} \times X)$$
$$\mathrm{c}(x) \triangleq \mathit{just}\ (0, x),$$
$$f^{\mathbb{C}}(\mathfrak{x}) \triangleq \begin{cases} \bot & \text{if } \mathfrak{x} = \bot \text{ or } \mathfrak{x} = \mathit{just}\ (n, x) \text{ and } f(x) = \bot \\ \mathit{just}\ (n + m, y) & \text{if } \mathfrak{x} = \mathit{just}\ (n, x) \text{ and } f(x) = \mathit{just}\ (m, y). \end{cases}$$

Our design choice can be motivated thus: as the element $\bot$ models divergent computations, we can assume the latter to have all the same cost, so that such information need not be explicitly written (for instance, measuring the number of reduction steps performed, we would have that divergent computations all have cost ∞). ⊠

3.1.2 Algebraic Operations

Monads provide an elegant way to structure effectful computations. However, they do not offer any actual effect constructor. Following Plotkin and Power (G. D. Plotkin & Power, 2001, 2002, 2003) we use *algebraic operations* as effect producers. As already remarked, from an operational perspective algebraic operations are those operations whose behaviour is independent of their continuations, or, equivalently, of the environment in which they are evaluated. This is reflected by important operational equivalences and refinements, such as the following one that the reader can easily verify to hold in Λ_p:

$$\mathbf{let}\ x = (e\ \mathbf{or}\ f)\ \mathbf{in}\ g \preceq^{\mathrm{ctx}} (\mathbf{let}\ x = e\ \mathbf{in}\ g)\ \mathbf{or}\ (\mathbf{let}\ x = f\ \mathbf{in}\ g).$$

We begin giving a formal definition of a *finitary* algebraic operation on a monad. Let us recall that a signature Σ consists of a collection of operation symbols **op** together with their arity, the latter being a function associating to each operation symbol a natural number representing its number of arguments (or operands). We sometimes use the notation $\mathbf{op} : n$ to indicate that the operation symbol **op** has arity n. In the following, given a function $f : X \to Y$ we write $\prod_n f : X^n \to Y^n$ for $\underbrace{f \times \cdots \times f}_{n}$.

Definition 11. *Let* Σ *be a signature. We say that a monad* $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ *is* Σ*-algebraic if associated with any n-ary operation symbol* **op** *in* Σ *is a set-indexed family of functions* $[\![\mathbf{op}]\!]^{\mathbb{T}}_X : (TX)^n \to TX$ *such that*

$$[\![\mathbf{op}]\!]^{\mathbb{T}}_Y \cdot \prod_n f^\dagger = f^\dagger \cdot [\![\mathbf{op}]\!]^{\mathbb{T}}_X \tag{alg op}$$

holds, for any n-ary operation symbol $\mathbf{op} \in \Sigma$ *and function* $f : X \to TY$*.*

Most of the times we will simply write $[\![\mathbf{op}]\!]$ in place of $[\![\mathbf{op}]\!]^{\mathbb{T}}$. Moreover, when unambiguous we will omit subscripts, hence writing $[\![\mathbf{op}]\!]$ in place of $[\![\mathbf{op}]\!]_X$. Notice that Definition 11 is essentially requiring TX to carry a Σ-algebra structure such that $f^\dagger$ is a Σ-algebra homomorphism, for any set X and function $f : X \to TY$. In fact, when written in pointiwise notation, (alg op) gives:

$$f^\dagger([\![\mathbf{op}]\!]_X(\mathfrak{x}_1, \ldots, \mathfrak{x}_n)) = [\![\mathbf{op}]\!]_Y(f^\dagger(\mathfrak{x}_1), \ldots, f^\dagger(\mathfrak{x}_n)).$$

Remark 1. If the monad is strong we straightforwardly generalise Definition 11 modifying (alg op) as

$$f^*(z, [\![\mathbf{op}]\!]_X(\mathfrak{x}_1, \ldots, \mathfrak{x}_n)) = [\![\mathbf{op}]\!]_Y(f^*(v, \mathfrak{x}_1), \ldots, f^*(v, \mathfrak{x}_n)), \qquad \text{(strong alg op)}$$

where now $f : Z \times X \to TY$.

Example 10 (Partiality monad)**.** The partiality monad $\mathbb{M}$ usually comes with no operation, as the possibility of divergence is an implicit feature of any Turing complete language. However, it is sometimes useful to add an explicit divergence operation (for instance, in strongly normalising calculi). For that, we consider the signature $\Sigma_{\mathbb{M}} \triangleq \{\uparrow\colon 0\}$. Having arity zero, the operation $\uparrow$ acts as a constant, and has semantics $[\![\uparrow]\!]^{\mathbb{M}} = \bot$. Since $f^\dagger(\bot) = \bot$, $\mathbb{M}$ is $\Sigma_{\mathbb{M}}$-algebraic. We thus observe that although divergence is often not considered an effect, from a mathematical perspective it certainly behaves as such. ⊠

Example 11 (Exception monad)**.** For the exception monad $\mathbb{E}$ (with set of exception symbols $\mathcal{E}$), we define the signature $\Sigma_{\mathbb{E}} \triangleq \{\mathbf{raise}_{\mathsf{e}} : 0 \mid \mathsf{e} \in \mathcal{E}\}$. Having arity zero, an operation of the form $\mathbf{raise}_{\mathsf{e}}$ acts as a constant, and has semantics defined as $[\![\mathbf{raise}_{\mathsf{e}}]\!]^{\mathbb{E}} \triangleq \mathsf{e}$, for any set X. Obviously $\mathbb{E}$ is $\Sigma_{\mathbb{E}}$-algebraic. We now generalise Proposition 3 to Σ-algebraic monad. We do so for the exception monad (as the maybe monad might be regard as a trivial case with no operations).

Proposition 5. *Let $\mathbb{T} = \langle T, \mathsf{T}, -^{\mathbb{T}}\rangle$ be $\Sigma_{\mathbb{T}}$-algebraic monad. Then the the monad $\mathbb{T}\mathbb{E}$ is $\Sigma_{\mathbb{T}\mathbb{E}}$-algebraic, where $\Sigma_{\mathbb{T}\mathbb{E}} \triangleq \Sigma_{\mathbb{T}} \cup \Sigma_{\mathbb{E}}$.*

As for Proposition 3, the proof is straightforward (again, the reader can consult (Hyland et al., 2006) for details). In fact, we can interpret operation symbols in $\Sigma_{\mathbb{T}\mathbb{E}}$ as follows:

$$[\![\mathbf{op}]\!]^{\mathbb{T}\mathbb{E}}_X \triangleq [\![\mathbf{op}]\!]^{\mathbb{T}}_{EX}$$
$$[\![\mathbf{raise}_{\mathsf{e}}]\!]^{\mathbb{T}\mathbb{E}}_X \triangleq \mathsf{T}_{EX}([\![\mathbf{raise}_{\mathsf{e}}]\!]^{\mathbb{E}}_X),$$

where $\mathbf{op}$ is an n-ary operation symbol in $\Sigma_{\mathbb{T}}$. Easy calculations show that (alg op) holds. For instance, we have:

$$\begin{aligned}
[\![\mathbf{op}]\!]^{\mathbb{T}\mathbb{E}}_Y \cdot \prod_n f^{\mathbb{T}\mathbb{E}} &= [\![\mathbf{op}]\!]^{\mathbb{T}\mathbb{E}}_Y \cdot \prod_n (f_E)^{\mathbb{T}} \\
&\quad \text{[By definitions in Proposition 3]} \\
&= (f_E)^{\mathbb{T}} \cdot [\![\mathbf{op}]\!]^{\mathbb{T}\mathbb{E}}_X \\
&\quad \text{[By (alg op) for } \mathbb{T}\text{]} \\
&= (f_E)^{\mathbb{T}} \cdot [\![\mathbf{op}]\!]^{\mathbb{T}\mathbb{E}}_X, \\
&\quad \text{[By definitions in Proposition 3]}
\end{aligned}$$

where f_E is the obvious modification of f_M in Proposition 3 to the exception monad.

⊠

Example 12 (Output monad)**.** Both for the total (i.e. $\mathbb{O}$) and the partial (i.e. $\mathbb{O}^\infty$) output monad we define the signature $\Sigma_{\mathbb{O}} \triangleq \{\mathbf{print}_{\mathsf{c}} : 1 \mid \mathsf{c} \in \mathcal{A}\}$. The intended semantics of a program $\mathbf{print}_{\mathsf{c}}(e)$ is to print the character c and then continue as e. Formally, we interpret $\mathbf{print}_{\mathsf{c}}$ as $[\![\mathbf{print}_{\mathsf{c}}]\!]^{\mathbb{O}}(w, \mathfrak{x}) \triangleq (c \cdot w, \mathfrak{x})$, where $(w, \mathfrak{x}) \in O^\infty(X)$ and $\mathsf{c} \cdot w$ denotes the string obtained by appending c to w. Straightforward calculations show that both for $\mathbb{O}$ and $\mathbb{O}^\infty$ are $\Sigma_{\mathbb{O}}$-algebraic. ⊠

Example 13 (Powerset monad)**.** For the non-empty powerset monad $\mathbb{F}$ we define the signature $\Sigma_{\mathbb{F}} \triangleq \{\mathbf{or} : 2\}$. The semantics of **or** is defined as $[\![\mathbf{or}]\!]^{\mathbb{F}}(\mathfrak{X}, \mathfrak{Y}) \triangleq \mathfrak{X} \cup \mathfrak{Y}$. Clearly $\mathbb{F}$ is $\Sigma_{\mathbb{FM}}$-algebraic. Moreover, by Proposition 5 $\mathbb{FM}$ is $\Sigma_{\mathbb{FM}}$-algebraic. ⊠

Example 14 (Distribution monad)**.** For the distribution monad $\mathbb{D}$ we define the signature $\Sigma_{\mathbb{D}} \triangleq \{\mathbf{or}_p : 2 \mid p \in \mathbb{Q} \cap [0,1]\}$. The intended semantics of a program $e\, \mathbf{or}_p\, f$ is to evaluate to e with probability p, and to f with probability $1-p$. The interpretation of $\mathbf{or}_p$ is defined by $[\![\mathbf{or}_p]\!]^{\mathbb{D}}(\mu, \nu)(x) \triangleq p \cdot \mu(x) + (1-p) \cdot \nu(x)$. We will actually work with the single operation $\mathbf{or}_{\frac{1}{2}}$ (abbreviated as **or**). We see that $\mathbb{D}$ is $\Sigma_{\mathbb{D}}$-algebraic, and that by by Proposition 5 $\mathbb{DM}$ is $\Sigma_{\mathbb{DM}}$-algebraic. ⊠

Example 15 (Cost)**.** For the partial cost monad $\mathbb{C}$ we define the signature $\Sigma_{\mathbb{C}} \triangleq \{\mathbf{tick} : 1\}$. The intended semantics of **tick** is to add a unit to the cost counter:

$$[\![\mathbf{tick}]\!]^{\mathbb{C}}(\mathfrak{x}) \triangleq \begin{cases} \bot & \text{if } \mathfrak{x} = \bot \\ \mathit{just}\ (n+1, x) & \text{if } \mathfrak{x} = \mathit{just}\ (n, x). \end{cases}$$

Clearly $\mathbb{C}$ is $\Sigma_{\mathbb{C}}$-algebraic. ⊠

Example 16 (Global State)**.** For the global state monad $\mathbb{G}$ we define a signature containing operations for reading the content of a location, and from storing bits in a location. Formally, define the signature $\Sigma_{\mathbb{G}} \triangleq \{\mathbf{set}_{\ell:=b} : 1, \mathbf{get}_\ell : 2 \mid \ell \in \mathcal{L}, b \in \{0,1\}\}$ whose interpretation is defined as follows:

$$[\![\mathbf{set}_{\ell:=b}]\!]^{\mathbb{G}}(\alpha)(\sigma) \triangleq \alpha(\sigma[\ell := b])$$

$$[\![\mathbf{get}_\ell]\!]^{\mathbb{G}}(\alpha, \beta)(\sigma) \triangleq \begin{cases} \alpha(\sigma) & \text{if } \sigma(\ell) = 0 \\ \beta(\sigma) & \text{if } \sigma(\ell) = 1, \end{cases}$$

where $\sigma[\ell := b]$ denotes the function that behaves as σ on locations $\ell' \neq \ell$ and that returns b on ℓ. We can extend Proposition 4 to Σ-algebraic monads.

Proposition 6. *Let $\mathbb{T} = \langle T, \mathrm{T}, -^{\mathrm{T}} \rangle$ be $\Sigma_{\mathbb{T}}$-algebraic monad. Then the the monad $\mathbb{T} \otimes \mathbb{G}$ is $\Sigma_{\mathbb{T}\otimes\mathbb{G}}$-algebraic, where $\Sigma_{\mathbb{T}\otimes\mathbb{G}} \triangleq \Sigma_{\mathbb{T}} \cup \Sigma_{\mathbb{G}}$.*

As for Proposition 4, proving Proposition 6 is straightforward (the reader can consult (Hyland et al., 2006) for details). In fact, we can interpret operation symbols in $\Sigma_{\mathbb{T}\otimes\mathbb{G}}$ as follows:

$$[\![\mathbf{op}]\!]^{\mathbb{T}\otimes\mathbb{G}}_X(\alpha_1, \ldots, \alpha_n)(\sigma) \triangleq [\![\mathbf{op}]\!]^{\mathbb{T}}_{X\times S}(\alpha_1(\sigma), \ldots, \alpha_n(\sigma))$$

$$[\![\mathbf{set}_{\ell:=b}]\!]^{\mathbb{T}\otimes\mathbb{G}}_X(\alpha)(\sigma) \triangleq \alpha(\sigma[\ell := b])$$

$$[\![\mathbf{get}_\ell]\!]^{\mathbb{T}\otimes\mathbb{G}}_X(\alpha, \beta)(\sigma) \triangleq \begin{cases} \alpha(\sigma) & \text{if } \sigma(\ell) = 1 \\ \beta(\sigma) & \text{if } \sigma(\ell) = 0, \end{cases}$$

where **op** is an n-ary operation symbol in $\Sigma_{\mathbb{T}}$. Easy calculations show that indeed (alg op) holds. For instance, we have:

$$\begin{aligned} f^{\mathbb{T}\otimes\mathbb{G}}([\![\mathbf{op}]\!]^{\mathbb{T}\otimes\mathbb{G}}_X(\alpha_1, \ldots, \alpha_n))(\sigma) &= (\mathsf{uncurry}\ f)^{\mathrm{T}}([\![\mathbf{op}]\!]^{\mathbb{T}}_{X\times S}(\alpha_1(\sigma), \ldots, \alpha_n(\sigma))) \\ &\quad \text{[By definitions in Proposition 6]} \\ &= [\![\mathbf{op}]\!]^{\mathbb{T}}_{X\times S}((\mathsf{uncurry}\ f)^{\mathrm{T}}(\alpha_1)(\sigma), \ldots, (\mathsf{uncurry}\ f)^{\mathrm{T}}(\alpha_n)(\sigma)) \\ &\quad \text{[By (alg op) for } \mathbb{T}] \\ &= [\![\mathbf{op}]\!]^{\mathbb{T}\otimes\mathbb{G}}_X(f^{\mathbb{T}\otimes\mathbb{G}}(\alpha_1), \ldots, f^{\mathbb{T}\otimes\mathbb{G}}(\alpha_n))(\sigma). \\ &\quad \text{[By definitions in Proposition 6]} \end{aligned}$$

⊠

Working with global states we see a limit of our framework, namely that operation symbols must have a finitary arity. If we allow locations to store arbitrary values (such as representation of natural numbers), then it is natural to introduce an operation $\mathbf{get}_\ell$ whose computational behaviour is as follows: $\mathbf{get}_\ell(x.e)$ reads the content of the location ℓ, which is a value v, and continues as $e[x := v]$. In order to model this kind of operation we introduce *generalised operations* (G. D. Plotkin & Power, 2003).

Definition 12. *A* generalised operation *on a set X is a function $\omega : P \times X^I \to X$. The set P is called the* parameter set *of the operation, whereas the (index) set I is called the* arity *of the operation.*

A generalised operation $\omega : P \times X^I \to X$ thus takes as arguments a parameter p and a map $\kappa : I \to X$ giving for each index $i \in I$ the argument $\kappa(i)$ to pass to ω. The notion of an algebraic signature is modified accordingly.

Definition 13. 1. *A* generalised signature Σ *consists of a set of operation symbols* $\mathbf{op}$ *together with their* generalised arity. *The latter is given by a a parameter set P and an arity/index set I. We write $\mathbf{op} : P \rightsquigarrow I$ to denote that the operation symbol $\mathbf{op}$ has parameter set P and arity set I.*

2. *Given a generalised signature Σ, we say that a monad $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ is Σ-algebraic if associated with any generalised operation symbol $\mathbf{op} : P \rightsquigarrow I$ in Σ is a set-indexed family of functions $[\![\mathbf{op}]\!]^{\mathbb{T}}_X : P \times (TX)^I \to TX$. such that the following identity holds for any map $f : X \to TY$, parameter $p \in P$, and map $\kappa : I \to TX$:*

$$f^\dagger([\![\mathbf{op}]\!]^{\mathbb{T}}_X(p, \kappa)) = [\![\mathbf{op}]\!]^{\mathbb{T}}_Y(p, f^\dagger \cdot \kappa). \qquad \text{(gen alg op)}$$

It is easy to see by taking the one-element set as parameter set and a finite set as arity set, generalised operations subsume finitary operations. We apply the same notational convention introduced for finitary operations to generalised operations.

Example 17 (Global states, revised)**.** We consider a variation of the global state monad (which, overloading the notation, we still denote by $\mathbb{G}$) in which locations store arbitrary structures which we assume to be encoded as values (via some standard encoding). The main example the reader may keep in mind is given by natural numbers and Church numerals. We denote by $\mathcal{V}$ the collection of such values, and assume $\mathcal{V}$ to be countable. Formally, we let S be the set $\mathcal{V}^{\mathcal{L}}$, and consider a signature consisting of generalised operations $\mathbf{set}_\ell : \mathcal{V} \rightsquigarrow 1$ and $\mathbf{get}_\ell : 1 \rightsquigarrow \mathcal{V}$. From a computational perspective such operations are used to build programs of the form $\mathbf{set}_\ell(v, e)$ (oftentimes abbreviated as $\ell := v; e$) and $\mathbf{get}_\ell(x.e)$. The former stores the value v in the location ℓ and continues as e, whereas the latter reads the content of the location ℓ, say it is v, and continue as $e[x := v]$. Here e is used as the description of a function κ_e from values to terms defined by $\kappa_e(v) \triangleq e[x := v]$. The interpretation of the new operations on $\mathbb{G}$ is standard:

$$[\![\mathbf{set}_\ell]\!]^{\mathbb{G}}(v, \alpha)(\sigma) = \alpha(\sigma[\ell := v])$$
$$[\![\mathbf{get}_\ell]\!]^{\mathbb{G}}(\kappa)(\sigma) = \kappa(\sigma(\ell))(\sigma).$$

⊠

For simplicity, we work with generalised operation symbols with generalised arity given by sets of values only. As a convention, we simply refer to operation symbols and signatures to denote generalised operation symbols and generalised signatures, respectively. Moreover, even if standard finitary operation symbols can be modelled as generalised operation symbols, it is useful to allow signatures to distinguish between different kinds of operation symbols. Given a signature Σ and $\mathbb{T}$, we allow Σ to contain operation symbols of the following kinds:

1. Finitary operation symbols. We write $\mathbf{op} : n$ to denote that the operation symbol $\mathbf{op}$ has arity n. Its interpretation on $\mathbb{T}$ is given by a set-indexed family of functions $[\![\mathbf{op}]\!]_X : (TX)^n \to TX$.

2. Finitary parametric operation symbols. We write $\mathbf{op} : P \rightsquigarrow n$ to denote that the operation symbol $\mathbf{op}$ has arity n and parameter set P. Its interpretation on $\mathbb{T}$ is given by a set-indexed family of functions $[\![\mathbf{op}]\!]_X : P \times (TX)^n \to TX$.

3. Generalised (non-parametric) operation symbols. We write $\mathbf{op} : \mathcal{V}$ to denote that the operation symbol $\mathbf{op}$ has arity set $\mathcal{V}$. Its interpretation on $\mathbb{T}$ is given by a set-indexed family of functions $[\![\mathbf{op}]\!]_X : (TX)^{\mathcal{V}} \to TX$.

4. Generalised operation symbols. We write $\mathbf{op} : P \rightsquigarrow \mathcal{V}$ to denote that the operation symbol $\mathbf{op}$ has parameter set P and arity set $\mathcal{V}$. Its interpretation on $\mathbb{T}$ is given by a set-indexed family of functions $[\![\mathbf{op}]\!]_X : P \times (TX)^{\mathcal{V}} \to TX$.

Remark 2 (Generalised operations, abstractly.)**.** Our presentation of generalised operations is rather concrete. Their original formulation in (G. D. Plotkin & Power, 2003) (where they are called *algebraic operations*) is given in terms of enriched category theory (Kelly, 2005). Roughly speaking, an algebraic operation is defined as a family of morphisms $\alpha_x : (Tx)^v \to (Tx)^w$ on a symmetric monoidal $\mathbb{V}$-category $\mathbb{C}$ parametrically natural in the Kleisli $\mathbb{V}$-category $\mathbb{C}_{\mathbb{T}}$. Here V is a complete and cocomplete symmetric monoidal closed category[7], $\mathbb{T}$ is a strong $\mathbb{V}$-monad on $\mathbb{C}$, and $(-)^v$ denotes cotensor with an object v of V. It is easy to see that taking $\mathsf{V} = \mathsf{Set}$ we recover our, concrete, notion of generalised operation, the set v being the arity set and w being the parameter set. Requiring α_x to be parametrically natural in $\mathbb{C}_{\mathbb{T}}$ is essentially requiring (strong) Kleisli extensions of functions to behave as algebra homomorphisms, and in the more abstract setting of (G. D. Plotkin & Power, 2003) ensures algebraic operations to commute with evaluation contexts. The main advantage of the enriched approach is that it allows a uniform treatment of complex arities as objects of V. Interesting examples are for V instantiated as Set, ω-Cpo (the category of ω-cppos), the category of posets, and functor categories (working with categories of the form $[W, \mathsf{Set}]$, where W is a suitable category of worlds, it is possible to model local states as algebraic effects).

Following the line of our working example Λ_p, we see that Σ-algebraic monads do not have enough structure to give operational semantics to effectful calculi. In fact, in Chapter 2 we relied on the ω-cppo structure of D to model the infinitary operational behaviour of Λ_p programs. As a consequence, in order to give a general monadic operational semantics to effectful calculi, we need to impose some domain theoretic properties on monads, whereby we account for infinitary computational behaviours. This can be elegantly done using the abstract notion of (ω-cppo)-enrichment (Kelly, 2005). However, such a level of abstraction is not necessary for our purposes, and thus we prefer the following more concrete definition.

Definition 14. *Given a Σ-algebraic monad $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ we say that $\mathbb{T}$ is* Σ-continuous[8] *if to any set X is associated an order $\sqsubseteq_X$ and an element $\bot_X \in TX$ such that $\langle TX, \sqsubseteq_X, \bot_X \rangle$ is an ω-cppo and the following conditions hold, for any operation symbol $\mathbf{op} : P \rightsquigarrow \mathcal{V}$ in Σ.*

Monotonicity. *For all functions $f, g : X \to TY$, $\kappa, \nu : \mathcal{V} \to TX$, and elements $\mathfrak{x}, \mathfrak{y} \in TX$, we have:*

$$\begin{aligned} \kappa \sqsubseteq \nu &\implies \mathbf{op}(p, \kappa) \sqsubseteq \mathbf{op}(p, \nu) \\ f \sqsubseteq g &\implies f^\dagger \sqsubseteq g^\dagger \\ \mathfrak{x} \sqsubseteq \mathfrak{y} &\implies f^\dagger(\mathfrak{x}) \sqsubseteq f^\dagger(\mathfrak{y}). \end{aligned}$$

[7] A concept we will meet again when studying abstract generalised distances.
[8] Cf. (Goguen, Thatcher, Wagner, & Wright, 1977).

Continuity. *For all ω-chains $(\kappa_n)_n$, $(f_n)_n$, and $(\mathfrak{x}_n)_n$ in $(TX)^{\mathcal{V}}$, $(TY)^X$, and TX, respectively, we have:*

$$[\![\mathbf{op}]\!]_X(p, \bigsqcup_n \kappa_n) = \bigsqcup_n [\![\mathbf{op}]\!]_X(p, \kappa_n)$$
$$(\bigsqcup_n f_n)^\dagger = \bigsqcup_n f_n^\dagger$$
$$f^\dagger(\bigsqcup_n \mathfrak{x}_n) = \bigsqcup_n f^\dagger(\mathfrak{x}_n).$$

Strictness. *For any $f : X \to TY$ we have:*

$$f^\dagger(\bot) = \bot.$$

When clear from the context, we will write $\bot$ and $\sqsubseteq$ in place of $\bot_X$ and $\sqsubseteq_X$, respectively.

Example 18. The monads $\mathbb{M}$, $\mathbb{ME}$, $\mathbb{FM}$, $\mathbb{DM}$, $\mathbb{O}^\infty$, $\mathbb{GM}$, and $\mathbb{C}$ are Σ-continuous. The order on MX, MEX, and $\mathbb{C}$ is the flat ordering $\sqsubseteq$ defined by:

$$\mathfrak{x} \sqsubseteq \mathfrak{y} \overset{\triangle}{\iff} \mathfrak{x} = \bot \text{ or } \mathfrak{x} = \mathfrak{y}.$$

Indeed $\bot$ behaves as the bottom element. The order on FM is defined by:

$$\mathfrak{x} \sqsubseteq \mathfrak{y} \overset{\triangle}{\iff} \forall x \in X.\ just\ x \in \mathfrak{x} \implies just\ x \in \mathfrak{y}.$$

Notice that we have $\{\bot\} \sqsubseteq \mathfrak{y}$ for any set $\mathfrak{y}$. Similarly, the order on DMX is defined by:

$$\mu \sqsubseteq \nu \overset{\triangle}{\iff} \forall x \in X.\ \mu(just\ x) \leq \nu(just\ x).$$

Notice that we have $\eta(\bot) \sqsubseteq \mu$, for any $\mu \in DMX$. The order on GMX is defined pointiwise from the flat ordering on $M(X \times S)$, whereas we order $O^\infty X$ as follows:

$$\langle u, \mathfrak{x}\rangle \sqsubseteq \langle w, \mathfrak{y}\rangle \overset{\triangle}{\iff} (\mathfrak{x} = \bot \wedge u \subseteq w) \vee (\mathfrak{x} = just\ x \wedge \mathfrak{y} = just\ x \wedge u = w),$$

where $\subseteq$ denotes the prefix order on $\mathcal{A}^\infty$. Checking the strictness, monotonicity, and continuity properties of Definition 14 is a routine exercise. ⊠

With Definition 14 we have introduced all the abstract counterparts of the structural properties of the subdistribution monad we used in Chapter 2 to give operational semantics to Λ_p. Therefore, it is time to introduce our vehicle calculus Λ_Σ and its monadic operational semantics.

3.2 A Computational Calculus with (Algebraic) Operations

In this section we define the syntax and semantics of Λ_Σ, our computational calculus with algebraic operations. Λ_Σ is an untyped call-by-value λ-calculus extending Levy's *fine-grain call-by-value* (P. Levy et al., 2003) with algebraic operations. The syntax of Λ_Σ is parametrised by a (generalised) signature of operation symbols Σ, and it is given in Figure 3.1, where x ranges over a fixed (countably infinite) set of variables, $\mathbf{op} : P \rightsquigarrow I$ (I is countable) ranges over elements of Σ, and p ranges over elements of P.

The rationale behind Λ_Σ is the same one behind Λ_p. We have two disjoint syntactical classes, namely the class of *computations* (denoted by $e, f, \ldots$) and the class of *values* (denoted by $v, w, \ldots$). We generically refer to syntactical expressions of Λ_Σ (i.e. computations or values) as *terms*. As for Λ_p, it is useful to introduce the 'hygienic' convention of keeping track of free variables of computations and values.

$$
\begin{array}{rlr}
e, f & ::= \mathbf{return}\ v & \text{(return)} \\
& \mid v w & \text{(application)} \\
& \mid \mathbf{let}\ x = e\ \mathbf{in}\ f & \text{(sequencing)} \\
& \mid \mathbf{op}(p, x.e) & \text{(operation).}
\end{array}
\qquad
\begin{array}{rlr}
v, w & ::= x & \text{(variable)} \\
& \mid \lambda x.e & \text{(abstraction).}
\end{array}
$$

Figure 3.1: Syntax of Λ_Σ.

$$
\frac{}{\Gamma, x \vdash^{\mathrm{v}} x}
\qquad
\frac{\Gamma, x \vdash^{\Lambda} e}{\Gamma \vdash^{\mathrm{v}} \lambda x.e}
\qquad
\frac{\Gamma \vdash^{\mathrm{v}} v \quad \Gamma \vdash^{\mathrm{v}} w}{\Gamma \vdash^{\Lambda} v w}
\qquad
\frac{\Gamma \vdash^{\mathrm{v}} v}{\Gamma \vdash^{\Lambda} \mathbf{return}\ v}
\qquad
\frac{\Gamma \vdash^{\Lambda} e \quad \Gamma, x \vdash^{\Lambda} f}{\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f}
\qquad
\frac{\Gamma, x \vdash^{\Lambda} e \quad p \in P}{\Gamma \vdash^{\Lambda} \mathbf{op}(p, x.e)}
$$

Figure 3.2: Sequents for Λ_Σ.

We do that by means of sequents of the form $\Gamma \vdash^{\Lambda} e$ and $\Gamma \vdash^{\mathrm{v}} v$. The letter Γ ranges over finite sets of variables, and the intended meaning of a sequent $\Gamma \vdash^{\Lambda} e$ is that e is a computation with free variables among Γ (the sequent $\Gamma \vdash^{\mathrm{v}} v$ has a similar reading). Rules for sequents are given in Figure 3.2, where we write Γ, x in place of $\Gamma \cup \{x\}$.

Clearly, provable sequents are closed under weakening, meaning that e.g. if $\Gamma \vdash^{\Lambda} e$ is provable, then so is $\Gamma, x \vdash^{\Lambda} e$. From now on when speaking about sequents we will tacitly mean provable sequents. Closed terms thus correspond to sequents with empty premises (which we simply denote as e.g. $\vdash^{\Lambda} e$). We also introduce the following notation:

$$
\begin{array}{lll}
\Lambda \triangleq \{e \mid \exists \Gamma.\ \Gamma \vdash^{\Lambda} e\} & \Lambda_\Gamma \triangleq \{e \mid \Gamma \vdash^{\Lambda} e\} & \Lambda_\circ \triangleq \{e \mid\ \vdash^{\Lambda} e\} \\
\mathcal{V} \triangleq \{v \mid \exists \Gamma.\ \Gamma \vdash^{\mathrm{v}} v\} & \mathcal{V}_\Gamma \triangleq \{v \mid \Gamma \vdash^{\mathrm{v}} v\} & \mathcal{V}_\circ \triangleq \{v \mid\ \vdash^{\mathrm{v}} v\}.
\end{array}
$$

The intuition behind constructors of Λ_Σ is the same one given for Λ_p. Concerning formal details, we adopt standard syntactical conventions as in (Barendregt, 1984) (notably the so-called variable convention). The notion of a free (resp. bound) variable is defined as usual (notice that the variable x is bound in $\mathbf{op}(p, x.e)$). In particular, the collection $FV(e)$ (resp. $FV(v)$) of free variables of a computation e (resp. value v) is the smallest set Γ such that $\Gamma \vdash^{\Lambda} e$ (resp. $\Gamma \vdash^{\mathrm{v}} v$) is provable. 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 (i.e. if $\vdash^{\Lambda} e$ is provable, and similarity for values).

Oftentimes we refer to closed computations as *programs*. Moreover, for finite lists of syntactic expressions (such as variables, computations, or values) we employ the 'bar notation', thus writing $\vec{\varphi}$ for a finite list $\varphi_1, \ldots, \varphi_n$ of φs. For instance, we write $\vec{x}$ and $\vec{v}$ for a finite list of variables and values, respectively.

Finally, we write $e[x := v]$ (resp. $w[v/x]$) for the capture-free substitution of the value v for all free occurrences of x in e (resp. w). Formally, we define $e[x := v]$ and $w[v/x]$ by mutual recursion on e and

w as follows:

$$\begin{aligned} x[v/x] &\triangleq v \\ y[v/x] &\triangleq x \\ (\lambda y.e)[v/x] &\triangleq \lambda y.e[x := v] \end{aligned}$$

$$\begin{aligned} (\mathbf{return}\ w)[x := v] &\triangleq \mathbf{return}\ w[v/x] \\ (wu)[x := v] &\triangleq w[v/x]u[v/x] \\ (\mathbf{let}\ y = e\ \mathbf{in}\ f)[x := v] &\triangleq \mathbf{let}\ y = e[x := v]\ \mathbf{in}\ f[x := v] \\ \mathbf{op}(p, y.e)[x := v] &\triangleq \mathbf{op}(p, y.e[x := v]). \end{aligned}$$

Obviously, the very definition of substitution gives the following inference rules (recall that we assume the variable convention):

$$\frac{\Gamma, x \vdash^{\Lambda} e \quad \Delta \vdash^{\mathrm{v}} v}{\Gamma, \Delta \vdash^{\Lambda} e[x := v]} \qquad \frac{\Gamma, x \vdash^{\mathrm{v}} w \quad \Delta \vdash^{\mathrm{v}} v}{\Gamma, \Delta \vdash^{\mathrm{v}} w[v/x]}$$

The fine-grain style has several advantages when studying meta-theoretical properties of calculi: there is a neat distinction between computations and values (which gives, for instance, a smooth definition of the notion of an applicative (bi)simulation relation), and proofs are more streamlined. However, calculi in fine-grain style has the drawback of being cumbersome for writing examples. For instance, the identity combinator $\lambda x.x$ in fine-grain style is written as $\mathbf{return}\ (\lambda x.(\mathbf{return}\ x))$. For this reason, our examples will be mostly given using the following coarse-grain λ-calculus syntax:

$$e, f ::= x \mid \lambda x.e \mid f e \mid \mathbf{op}(p, x.e).$$

It is not hard to convince ourselves that the two presentations are equivalent, as summarised by Table 3.1.

λ-calculus syntax	fine-grain syntax
x	$\mathbf{return}\ x$
$\lambda x.e$	$\mathbf{return}\ \lambda x.e$
$f e$	$\mathbf{let}\ x = f\ \mathbf{in}\ (e\ \mathrm{to}\ y.xy)$
$\mathbf{op}(p, x.e)$	$\mathbf{op}(p, x.e)$

Table 3.1: Correspondence between fine-grain and coarse-grain calculi

We now define a *call-by-value monadic operational semantics* for Λ_Σ. In the following, we assume a Σ-continuous monad $\mathbb{T}$ to be fixed.

Remark 3. Since we give semantics to programs, i.e. closed computations, it makes sense to consider generalised operation symbols whose arity set I is encoded by some subset of $\mathcal{V}_\circ$. For simplicity, we ignore the specific subset of (closed) values used to encode elements of I, and simply write $\mathbf{op} : P \rightsquigarrow \mathcal{V}_\circ$ for operation symbols in Σ. As already remarked, the standard example the reader may keep in mind is given by $I = \mathbb{N}$ and the encoding of natural numbers as Church numerals.

Operational semantics is defined by means of an evaluation function $[\![-]\!] : \Lambda_\circ \to T\mathcal{V}_\circ$ mapping a program to a *monadic value*, i.e. an element of $T\mathcal{V}_\circ$.

Definition 15. *The $\mathbb{N}$-indexed family of functions $[\![-]\!]_n : \Lambda_\circ \to T\mathcal{V}_\circ$ is inductively defined as follows:*

$$\begin{aligned}
[\![e]\!]_0 &\triangleq \bot \\
[\![\mathbf{return}\ v]\!]_{n+1} &\triangleq \eta(v) \\
[\![(\lambda x.e)v]\!]_{n+1} &\triangleq [\![e[x := v]]\!]_n \\
[\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!]_{n+1} &\triangleq [\![f[x := -]]\!]_n^\dagger [\![e]\!]_n \\
[\![\mathbf{op}(p, x.e)]\!]_{n+1} &\triangleq [\![\mathbf{op}]\!](p, v \mapsto [\![e[x := v]]\!]_n).
\end{aligned}$$

Notice that the index n in $[\![-]\!]_n$ does not actually refer to the number of β-reductions performed. Moreover, since $\mathbb{T}$ is Σ-continuous, $([\![e]\!]_n)_n$ forms an ω-chain in $T\mathcal{V}_\circ$, and thus has least upper bound $\bigsqcup_n [\![e]\!]_n$.

Definition 16. *The evaluation function $[\![-]\!] : \Lambda_\circ \to T\mathcal{V}_\circ$ is defined as $[\![e]\!] \triangleq \bigsqcup_{n<} [\![e]\!]_n$.*

Finally, we notice that Σ-continuity gives the following universal property of the evaluation function.

Lemma 6. *The evaluation function $[\![-]\!]$ is the least function $\varphi : \Lambda_\circ \to T\mathcal{V}_\circ$ satisfying the following identities:*

$$\begin{aligned}
\varphi(\mathbf{return}\ v) &= \eta(v) \\
\varphi((\lambda x.e)v) &= \varphi(e[x := v]) \\
\varphi(\mathbf{let}\ x = e\ \mathbf{in}\ f) &= (v \mapsto \varphi(f[x := v]))^\dagger \varphi(e) \\
\varphi(\mathbf{op}(p, x.e)) &= [\![\mathbf{op}]\!](p, v \mapsto \varphi(e[x := v])).
\end{aligned}$$

Before giving examples of concrete instances of Λ_Σ we briefly expand on the operational meaning of algebraic operations. At the very begging of this chapter, we informally defined algebraic operations as operations whose semantics commutes with evaluation contexts. We can now make such definition a formal result. First of all, we give Λ_Σ an operational semantics based on Felleisen's notion of an *evaluation context* (Felleisen & Hieb, 1992). We will extensively use these kinds of semantics in Chapter 6 and Chapter 7. Λ_Σ evaluation contexts are defined by the following grammar, where e is a computation with $FV(e) \subseteq \{x\}$:

$$E ::= [-] \mid \mathbf{let}\ x = E\ \mathbf{in}\ e.$$

The hole $[-]$ in an evaluation context acts as a placeholder for a *computation*, and we write $E[e]$ for the computation obtained by replacing the hole $[-]$ in E with the computation e. We immediately notice that any computation admits the following unique decomposition (see also Lemma 25).

Lemma 7. *Any program e can be uniquely decomposed in one of the following (mutually exclusive) forms:*

1. $\mathbf{return}\ v$;
2. $E[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ e]$;
3. $E[vw]$;
4. $E[\mathbf{op}(p, x.e)]$.

Using Lemma 7 we can give Λ_Σ an alternative operational semantics, and prove its equivalence with the one defined in Definition 15.

Definition 17. *Define the $\mathbb{N}$-indexed family of evaluation function $(\!|-|\!)_n : \Lambda_\circ \to T\Lambda_\circ$ as follows:*

$$\begin{aligned}
(\!|e|\!)_0 &= \bot \\
(\!|\mathbf{return}\ v|\!)_{n+1} &= \eta(v) \\
(\!|E[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ e]|\!)_{n+1} &= (\!|E[e[x := v]]|\!)_n \\
(\!|E[(\lambda x.e)v]|\!)_{n+1} &= (\!|E[e[x := v]]|\!)_n \\
(\!|E[\mathbf{op}(p, x.e)]|\!)_{n+1} &= [\![\mathbf{op}]\!](p, v \mapsto (\!|E[e[x := v]]|\!)_n).
\end{aligned}$$

As for Definition 15, $((\!|e|\!)_n)_n$ forms an ω-chain in $T\mathcal{V}_\circ$, so that we can define $(\!|e|\!)$ as $\bigsqcup_n (\!|e|\!)_n$. Since for any $\mathbf{op} \in \Sigma$, $[\![\mathbf{op}]\!]$ is continuous, we see that $(\!|-|\!)$ enjoys the following universal property.

Lemma 8. *The evaluation function $(\!|-|\!)$ is the least function $\varphi : \Lambda_\circ \to T\mathcal{V}_\circ$ satisfying the following identities:*

$$\begin{aligned}
\varphi(\mathbf{return}\ v) &= \eta(v) \\
\varphi(E[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ e]) &= \varphi(E[e[x := v]]) \\
\varphi(E[(\lambda x.e)v]) &= \varphi(E[e[x := v]]) \\
\varphi(E[\mathbf{op}(p, x.e)]) &= [\![\mathbf{op}]\!](p, v \mapsto \varphi(E[e[x := v]])).
\end{aligned}$$

Finally, we can rely on Lemma 6 and Lemma 8 to prove that the two operational semantics given to Λ_Σ are equivalent.

Lemma 9. *For any closed computation e, $(\!|e|\!) \sqsubseteq [\![e]\!]$.*

Proof. We first prove that $[\![-]\!]$ satisfies the identities in Lemma 8, for which we will infer $(\!|e|\!) \sqsubseteq [\![e]\!]$.

1. The first identity obviously holds.

2. For the second identity, we proceed by induction on E. Suppose $E = [-]$. We reason as follows:

$$\begin{aligned}
[\![\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ e]\!] &= [\![e[x := -]]\!]^\dagger(\eta(v)) \\
&\quad \text{[By Lemma 6]} \\
&= [\![e[x := v]]\!]. \\
&\quad \text{[Since } \forall f : X \to TY.\ f^\dagger \cdot \eta_X = f\text{]}
\end{aligned}$$

   Suppose E to be of the form $\mathbf{let}\ y = E'[-]\ \mathbf{in}\ f$, for some evaluation context E'. By induction hypothesis we have $[\![E'[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ e]]\!] = [\![E'[e[x := v]]]\!]$, so that we can calculate:

$$\begin{aligned}
[\![\mathbf{let}\ y = E'[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ e]\ \mathbf{in}\ f]\!] &= [\![f[y := -]]\!]^\dagger([\![E'[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ e]]\!]) \\
&\quad \text{[By Lemma 6]} \\
&= [\![f[y := -]]\!]^\dagger([\![E'[e[x := v]]]\!]) \\
&\quad \text{[By induction hypothesis]} \\
&= [\![\mathbf{let}\ y = E'[e[x := v]]\ \mathbf{in}\ f]\!]. \\
&\quad \text{[By Lemma 6]}
\end{aligned}$$

3. For the third identity we proceed exactly as for the second one.

4. We prove the fourth identity by induction on E. The base case is trivial. Suppose now E to be of the form $\mathbf{let}\ y = E'\ \mathbf{in}\ f$, and write φ for the function $[\![f[y := -]]\!]$. We have:

$$\begin{aligned}
[\![\mathbf{let}\ y = (E'[\mathbf{op}(p, x.e)])\ \mathbf{in}\ f]\!] &= \varphi^\dagger([\![E'[\mathbf{op}(p, x.e)]]\!]) \\
&\quad \text{[By Lemma 6]} \\
&= \varphi^\dagger([\![\mathbf{op}]\!](p, v \mapsto [\![(E'[e])[y := v]]\!])) \\
&\quad \text{[By induction hypothesis]} \\
&= [\![\mathbf{op}]\!](p, v \mapsto \varphi^\dagger([\![(E'[e])[x := v]]\!])) \\
&\quad \text{[By (gen alg op)]} \\
&= [\![\mathbf{op}]\!](p, v \mapsto [\![\mathbf{let}\ y = (E'[e])[x := v]\ \mathbf{in}\ f]\!]). \\
&\quad \text{[By Lemma 6]}
\end{aligned}$$

Finally, since x is bound in e we can assume x to fresh in E', so to have:

$$\begin{aligned}
[\![E[\mathbf{op}(p, x.e)]]\!] &= [\![\mathbf{op}]\!](p, v \mapsto [\![\mathbf{let}\ y = (E'[e])[x := v]\ \mathbf{in}\ f]\!]) \\
&= [\![\mathbf{op}]\!](p, v \mapsto [\![\mathbf{let}\ y = E'[e[x := v]]\ \mathbf{in}\ f]\!]) \\
&= [\![\mathbf{op}]\!](p, v \mapsto [\![E[e[x := v]]]\!]) \\
&= [\![\mathbf{op}(p, x.E[e])]\!].
\end{aligned}$$

□

We now have to prove that for any closed computation e, $[\![e]\!] \sqsubseteq (\!|e|\!)$. That is a direct consequence of the following result.

Lemma 10. *For any closed computation e and evaluation context E we have:*

$$(\!|E[e]|\!) = (v \mapsto (\!|E[\mathbf{return}\ v]|\!))^\dagger (\!|e|\!).$$

Proof. Let us write φ for the function $v \mapsto (\!|E[\mathbf{return}\ v]|\!)$. By induction on n with a case analysis on the shape of $E[e]$ according to Lemma 7, we show that for any $n \geq 0$ we have

$$(\!|E[e]|\!)_n \sqsubseteq (v \mapsto (\!|E[\mathbf{return}\ v]|\!))^\dagger (\!|e|\!),$$

from which follows $(\!|E[e]|\!) \sqsubseteq (v \mapsto (\!|E[\mathbf{return}\ v]|\!))^\dagger (\!|e|\!)$. The case for $n = 0$ is trivial. We proceed with the inductive step.

- If $E[e] = \mathbf{return}\ v$, then we have

  $$(\!|\mathbf{return}\ v|\!)_{n+1} = \eta(v) = [\![\mathbf{return}\ v]\!]^\dagger.$$

  Suppose now $E[e]$ to be of the form $F[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ f]$. Then we must have $E = E[E'[-]]$ and $e = E'[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ f]$, for some evaluation context E'. We have:

  $$\begin{aligned}
  (\!|E[e]|\!)_{n+1} &= (\!|E[E'[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ f]]|\!)_{n+1} \\
  &= (\!|E[E'[f[x := v]]]|\!)_n \\
  &\quad \text{[By Definition 17]} \\
  &\sqsubseteq \varphi^\dagger (\!|E'[f[x := v]]|\!) \\
  &\quad \text{[By induction hypothesis]} \\
  &= \varphi^\dagger (\!|E'[\mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ f]|\!) \\
  &\quad \text{[By Lemma 8]} \\
  &= \varphi^\dagger (\!|e|\!).
  \end{aligned}$$

- If $E[e] = F[(\lambda x.f)v]$, then we must have $F = E[E'[-]]$ and $e = E'[(\lambda x.f)v]$, for some evaluation context E'. We proceed exactly as in previous case.
- If $E[e] = F[\mathbf{op}(p, x.f)]$, then we have $F = E[E'[-]]$ and $e = E'[\mathbf{op}(p, x.f)]$, for some evaluation context E'. Let us write κ for the map $v \mapsto (\!|f[x := v]|\!)$. Relying on the induction hypothesis, we have:

$$\begin{aligned}
(\!|E[e]|\!)_{n+1} &= (\!|E[E'[\mathbf{op}(p, x.f)]]|\!)_{n+1} \\
&= [\![\mathbf{op}]\!](p, v \mapsto (\!|E[E'[f[x := v]]]|\!)_n) \\
&\quad \text{[By Definition 17]} \\
&\sqsubseteq [\![\mathbf{op}]\!](p, v \mapsto (\varphi^\dagger (\!|E'[f[x := v]]|\!))) \\
&\quad \text{[By induction hypothesis and Definition 14]} \\
&= [\![\mathbf{op}]\!](p, \varphi^\dagger \cdot \kappa) \\
&= \varphi^\dagger([\![\mathbf{op}]\!](p, \kappa)) \\
&\quad \text{[By (gen alg op)]} \\
&= \varphi^\dagger (\!|e|\!).
\end{aligned}$$

In a similar fashion it is possible to show that for any $n \geq 0$ we have $(v \mapsto (\!|E[\mathbf{return}\ v]|\!))^\dagger (\!|e|\!)_n \sqsubseteq (\!|E[e]|\!)$, from which we infer $(v \mapsto (\!|E[\mathbf{return}\ v]|\!))^\dagger (\!|e|\!) \sqsubseteq (\!|E[e]|\!)$, and thus the wished thesis. □

Corollary 1. *For any closed computation e we have $[\![e]\!] = (\!|e|\!)$.*

In particular, from Corollary 1 we infer the equality $[\![E[\mathbf{op}(p, x.e)]]\!] = [\![\mathbf{op}(p, x.E[e])]\!]$ which states that the operational semantics of **op** commutes with evaluation contexts. This is nothing but the operational behaviour we had in mind for algebraic operations.

We now give examples of concrete instances of Λ_Σ and show how our monadic operational semantics can be instantiated to standard, concrete semantics. In doing so, we also discuss an example of a non-algebraic – yet natural – operation, namely exception handling. We omit the example of the probabilistic instance of Λ_Σ, as the latter obviously coincides[9] with Λ_p.

3.2.1 Relevant Examples

Example 19 (Pure λ-calculus)**.** If we consider the empty signature and the partiality monad $\mathbb{M}$ we recover Levy's fine-grain call-by-value (P. Levy et al., 2003). We refer to such a calculus as $\Lambda_\mathbb{M}$ (we prefer the latter notation to $\Lambda_{\Sigma_\mathbb{M}}$). Concretely, the syntax of $\Lambda_\mathbb{M}$ is defined by the following grammars:

$$\begin{aligned}
v, w &::= x \mid \lambda x.e \\
e, f &::= \mathbf{return}\ v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f.
\end{aligned}$$

We give $\Lambda_\mathbb{M}$ a standard big-step operational semantics by means of binary convergence relation $\Downarrow$ between programs and closed values inductively defined as follows:

$$\frac{}{\mathbf{return}\ v \Downarrow v} \qquad \frac{e[x := v] \Downarrow w}{(\lambda x.e)v \Downarrow w} \qquad \frac{e \Downarrow v \quad f[x := v] \Downarrow w}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Downarrow w}$$

[9] Notice, however, that Λ_p models probabilistic computations using the subdistribution monad and thus differs from the instance of Λ_Σ based on the partial full distribution monad $\mathbb{DM}$ (and the signature $\Sigma_{\mathbb{DM}}$), which we denote as $\Lambda_{\mathbb{DM}}$. Nevertheless, since these two monads are equivalent, we can safely regard Λ_p and $\Lambda_{\mathbb{DM}}$ as equivalent too.

Moreover, we can define a coinductive diverge predicate $\Uparrow$ on programs as follows[10]:

$$\frac{e[x := v] \Uparrow}{(\lambda x.e)v \Uparrow} \qquad \frac{e \Uparrow}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Uparrow} \qquad \frac{e \Downarrow v \quad f[x := v] \Uparrow}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Uparrow}.$$

It is easy to show that $[\![e]\!] = \mathit{just}\ v$ if and only if $e \Downarrow v$, and conversely that $[\![e]\!] = \bot$ if and only if $e \Uparrow$. ⊠

Example 20 (Nondeterministic λ-calculus)**.** Instantiating Λ_Σ with the the monad $\mathbb{FM}$ and its associated signature $\Sigma_{\mathbb{FM}}$ we obtain the nondeterministic call-by-value λ-calculus $\Lambda_{\mathbb{FM}}$. Concretely, $\Lambda_{\mathbb{FM}}$ is defined by the follows syntax:

$$\begin{aligned} v, w &::= x \mid \lambda x.e \\ e, f &::= \mathbf{return}\ v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \mid e\ \mathbf{or}\ f. \end{aligned}$$

We inductively define the may convergence relation $\Downarrow$ between programs and closed values as follows:

$$\frac{}{\mathbf{return}\ v \Downarrow v} \qquad \frac{e[x := v] \Downarrow w}{(\lambda x.e)v \Downarrow w} \qquad \frac{e \Downarrow v \quad f[x := v] \Downarrow w}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Downarrow w} \qquad \frac{e \Downarrow v}{e\ \mathbf{or}\ f \Downarrow v} \qquad \frac{f \Downarrow w}{e\ \mathbf{or}\ f \Downarrow w}$$

Dually, we coinductively define a may divergence predicate $\Uparrow$ as follows:

$$\frac{e[x := v] \Uparrow}{(\lambda x.e)v \Uparrow} \qquad \frac{e \Uparrow}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Uparrow} \qquad \frac{e \Downarrow v \quad f[x := v] \Uparrow}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Uparrow} \qquad \frac{e \Uparrow}{e\ \mathbf{or}\ f \Uparrow} \qquad \frac{f \Uparrow}{e\ \mathbf{or}\ f \Uparrow}$$

It is straightforward to prove that $e \Downarrow v$ if and only if $\mathit{just}\ v \in [\![e]\!]$, and dually $e \Uparrow$ if and only if $\bot \in [\![e]\!]$. Moreover, we can recover a must converge semantics by noticing that e must converge if and only if $\bot \notin [\![e]\!]$. It follows that $[\![e]\!] = \bot$ if and only if e must diverge. ⊠

Example 21 (λ-calculus with exceptions)**.** We now instantiate Λ_Σ with the monad $\mathbb{ME}$ and its associated signature $\Sigma_{\mathbb{ME}}$ plus a new operation for handling exceptions. The concrete syntax of the calculus is thus defined:

$$\begin{aligned} v, w &::= x \mid \lambda x.e \\ e, f &::= \mathbf{return}\ v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \mid \mathbf{raise}_{\mathsf{e}} \mid \mathbf{handle}_{\mathsf{e}}(e, f). \end{aligned}$$

Instead on giving semantics using the monad $\mathbb{ME}$ (that is not the purpose of this example) we inductively define a converge relation $\Downarrow$ between programs and closed value and an inductive predicate $\lightning \mathsf{e}$ with the intended meaning that the exception e has been raised (for simplicity we assume only one exception e to be available). These relations are defined as follows:

$$\frac{}{\mathbf{return}\ v \Downarrow v} \quad \frac{e[x := v] \Downarrow w}{(\lambda x.e)v \Downarrow w} \quad \frac{e \Downarrow v \quad f[x := v] \Downarrow w}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Downarrow w} \quad \frac{e \Downarrow v}{\mathbf{handle}_{\mathsf{e}}(e, f) \Downarrow v} \quad \frac{e \lightning \mathsf{e} \quad f \Downarrow v}{\mathbf{handle}_{\mathsf{e}}(e, f) \Downarrow v}$$

$$\frac{}{\mathbf{raise}_{\mathsf{e}} \lightning \mathsf{e}} \quad \frac{e[x := v] \lightning \mathsf{e}}{(\lambda x.e)v \lightning \mathsf{e}} \quad \frac{e \lightning \mathsf{e}}{\mathbf{let}\ x = e\ \mathbf{in}\ f \lightning \mathsf{e}} \quad \frac{e \Downarrow v \quad f[x := v] \lightning \mathsf{e}}{\mathbf{let}\ x = e\ \mathbf{in}\ f \lightning \mathsf{e}} \quad \frac{e \lightning \mathsf{e} \quad f \lightning \mathsf{e}}{\mathbf{handle}_{e}(e, f) \lightning \mathsf{e}}$$

The evaluation of a computation of the form $\mathbf{handle}_{\mathsf{e}}(e, f)$ depends on the evaluation of e. That is, exception handling cannot be considered as an algebraic operation, as it directly depends on one of its arguments. To see that, it is sufficient to observe that no reasonable notion of equivalence will equate

[10] Throughout this dissertation we use double-lines inference rules for coinductive definitions.

the two terms e and f defined as follows (where 0 and 1 denote the the booelan values false and true, respectively):

$$E \triangleq \mathbf{let}\ x = [-]\ \mathbf{in}\ (\mathbf{if}\ x\ \mathbf{then}\ \mathbf{raise}_{\mathrm{e}}\ \mathbf{else}\ x)$$
$$e \triangleq E[\mathbf{handle}_{\mathrm{e}}(\mathbf{return}\ 1, \mathbf{return}\ 0)]$$
$$f \triangleq \mathbf{handle}_{\mathrm{e}}(E[\mathbf{return}\ 1], E[\mathbf{return}\ 0]).$$

In fact, we have $e \not\Downarrow \mathrm{e}$, whereas $f \Downarrow 0$.

⊠

Example 22 (Imperative λ-calculus)**.** Instantiating Λ_Σ with the monad $\mathbb{M}\otimes\mathbb{G}$ and its associated signature$\Sigma_{\mathbb{T}\otimes\mathbb{G}}$ we obtain the calculus $\Lambda_{\mathbb{M}\otimes\mathbb{G}}$. The concrete syntax of the calculus (recall Remark 3) is as follows, where we use the lighter notation $\ell := v; e$ in place of $\mathbf{set}_\ell(v, e)$:

$$\begin{aligned} v, w &::= x \mid \lambda x.e \\ e, f &::= \mathbf{return}\ v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \mid \ell := v; e \mid \mathbf{get}_\ell(x.e). \end{aligned}$$

We define a convergence relation $\Downarrow$ by means of judgments of the form $\langle e, \sigma\rangle \Downarrow \langle v, \sigma'\rangle$ (where e is a program and v is a closed value) inductively as follows:

$$\frac{}{\langle \mathbf{return}\ v, \sigma\rangle \Downarrow \langle v, \sigma\rangle} \qquad \frac{\langle e[x := v], \sigma\rangle \Downarrow \langle w, \sigma'\rangle}{\langle (\lambda x.e)v, \sigma\rangle \Downarrow \langle w, \sigma'\rangle} \qquad \frac{\langle e, \sigma\rangle \Downarrow \langle w, \sigma''\rangle \quad \langle f[x := w], \sigma''\rangle \Downarrow \langle v, \sigma'\rangle}{\langle \mathbf{let}\ x = e\ \mathbf{in}\ f, \sigma\rangle \Downarrow \langle v, \sigma'\rangle}$$

$$\frac{\langle e[x := w], \sigma\rangle \Downarrow \langle v, \sigma'\rangle \quad \sigma(\ell) = w}{\langle \mathbf{get}_\ell(x.e), \sigma\rangle \Downarrow \langle v, \sigma'\rangle} \qquad \frac{\langle e, \sigma[\ell := w]\rangle \Downarrow \langle v, \sigma'\rangle}{\langle \ell := w; e, \sigma\rangle \Downarrow \langle v, \sigma'\rangle}$$

Dually, we define a divergence predicate $\Uparrow$ coinductively as follows:

$$\frac{\langle e[x := v], \sigma\rangle \Uparrow}{\langle (\lambda x.e)v, \sigma\rangle \Uparrow} \qquad \frac{\langle e, \sigma\rangle \Uparrow}{\langle \mathbf{let}\ x = e\ \mathbf{in}\ f, \sigma\rangle \Uparrow} \qquad \frac{\langle e, \sigma\rangle \Downarrow \langle w, \sigma'\rangle \quad \langle f[x := w], \sigma'\rangle \Uparrow}{\langle \mathbf{let}\ x = e\ \mathbf{in}\ f, \sigma\rangle \Uparrow}$$

$$\frac{\langle e[x := v], \sigma\rangle \Uparrow \quad \sigma(\ell) = v}{\langle \mathbf{get}_\ell(x.e), \sigma\rangle \Uparrow} \qquad \frac{\langle e, \sigma[\ell := v]\rangle \Uparrow}{\langle \ell := v; e, \sigma\rangle \Uparrow}.$$

As usual we can show that $\langle e, \sigma\rangle \Downarrow \langle v, \sigma'\rangle$ if and only if $[\![e]\!](\sigma) = \mathit{just}\ (\sigma', v)$, and, dually, $\langle e, \sigma\rangle \Uparrow$ if and only if $[\![e]\!](\sigma) = \bot$. Moreover, we see that the semantics given to operation symbols behaves as desired. For instance, assuming $\sigma(\ell) = v_0$, we have:

$$[\![\mathbf{get}_\ell(x.e)]\!](\sigma) = [\![\mathbf{get}_\ell]\!](v \mapsto [\![e[x := v]]\!])(\sigma) = [\![e[x := v_0]]\!](\sigma).$$

⊠

Example 23 (λ-calculus with output)**.** Instantiating Λ_Σ with the partial output monad and the associated signature $\Sigma_{\mathbb{O}^\infty}$, we obtain the calculus $\Lambda_{\mathbb{O}^\infty}$. The concrete syntax of $\Lambda_{\mathbb{O}^\infty}$ is defined by the following grammar:

$$\begin{aligned} v, w &::= x \mid \lambda x.e \\ e, f &::= \mathbf{return}\ v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \mid \mathbf{print}_{\mathrm{c}}.e. \end{aligned}$$

We inductively define a convergence relation $\Downarrow$ between programs and pairs of *finite* strings over $\mathcal{A}$ (here denoted by letters $a, b, \ldots$ to avoid confusion) and values as follows:

$$\frac{}{\mathbf{return}\ v \Downarrow \langle \varepsilon, v\rangle} \qquad \frac{e[x := v] \Downarrow \langle a, w\rangle}{(\lambda x.e)v \Downarrow \langle a, w\rangle} \qquad \frac{e \Downarrow \langle a, v\rangle \quad f[x := v] \Downarrow \langle b, w\rangle}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Downarrow \langle ab, w\rangle} \qquad \frac{e \Downarrow \langle a, v\rangle}{\mathbf{print}_{\mathrm{c}}.e \Downarrow \langle \mathrm{c} \cdot a, v\rangle}$$

Dually, we coinductively define a divergence relation $\Uparrow$ between programs and possibly infinite strings over $\mathcal{A}$ (again denoted by letter $a, b, \ldots$) as follows:

$$\frac{e[x := v] \Uparrow a}{(\lambda x.e)v \Uparrow a} \qquad \frac{e \Uparrow a}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Uparrow a} \qquad \frac{e \Downarrow (a, v) \quad f[x := v] \Uparrow b}{\mathbf{let}\ x = e\ \mathbf{in}\ f \Uparrow ab} \qquad \frac{e \Uparrow a}{\mathbf{print}_{\mathtt{c}}.e \Uparrow \mathtt{c} \cdot a}\ .$$

As usual, we can show $e \Downarrow \langle a, v\rangle$ if and only if $[\![e]\!] = (a, \mathit{just}\ v)$, and $e \Uparrow a$ if and only if $[\![e]\!] = (a, \bot)$. ⊠

Example 24 (λ-calculus with cost)**.** Instantiating Λ_Σ with the cost monad $\mathbb{C}$ and its associated signature $\Sigma_{\mathbb{C}}$, we obtain the calculus $\Lambda_{\mathbb{C}}$. The concrete syntax of the calculus is as follows:

$$\begin{aligned} v, w &::= x \mid \lambda x.e \\ e, f &::= \mathbf{return}\ v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \mid \mathbf{tick}(e). \end{aligned}$$

The calculus by itself is not very interesting. However, we can consider its fragment defined by the following grammar:

$$\begin{aligned} v, w &::= x \mid \lambda x.\mathbf{tick}(e) \\ e, f &::= v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f. \end{aligned}$$

By putting a ticking operation after all λ-abstractions we can count the number of reduction steps performed during the evaluation of a program. In fact, defining the encoding $\overline{e}$ of a computation of $\Lambda_{\mathbb{M}}$ as follows

$$\begin{aligned} \overline{x} &\triangleq x \\ \overline{\lambda x.e} &\triangleq \lambda x.\mathbf{tick}(\overline{e}) \\ \overline{\mathbf{return}\ v} &\triangleq \mathbf{return}\ \overline{v} \\ \overline{vw} &\triangleq \overline{v}\,\overline{w} \\ \overline{\mathbf{let}\ x = e\ \mathbf{in}\ f} &\triangleq \mathbf{let}\ x = \overline{e}\ \mathbf{in}\ \overline{f}, \end{aligned}$$

we see that $[\![\overline{e}]\!] = (n, \mathit{just}\ v)$ if and only if $e \Downarrow v$ performing n (β-)reduction steps, i.e. steps of the form $(\lambda x.e)v \mapsto e[x := v]$. ⊠

So far we have only given examples of calculi with just one type of effect (provided we do not consider partiality as an effect). We can also instantiate Λ_Σ to obtain richer effectful calculi, as witnessed by the following example.

Example 25 (Imperative probabilistic λ-calculus)**.** We instantiate Λ_Σ with the monad $\mathbb{DM} \otimes \mathbb{G}$ and its associated signature $\Sigma_{\mathbb{DM}\otimes\mathbb{G}}$. The concrete syntax of the calculus is given by:

$$\begin{aligned} v, w &::= x \mid \lambda x.e \\ e, f &::= v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \mid e\ \mathbf{or}\ f \mid \ell := v; e \mid \mathbf{get}_\ell(x.e). \end{aligned}$$

A computation e evaluates to a function $[\![e]\!]$ mapping an initial state σ to (something equivalent to) a subdistribution over pairs consisting of a value and a final state. The rationale is that $[\![e]\!](\sigma)(v, \sigma')$ gives the probability that the computation e when evaluated in the initial state σ converges to a value v with final state σ'. ⊠

Example 26 (Imperative λ-calculus with cost)**.** We instantiate Λ_Σ with the monad $\mathbb{C} \otimes \mathbb{G}$ and its associated signature $\Sigma_{\mathbb{C}\otimes\mathbb{G}}$. We call the resulting calculus $\Lambda_{\mathbb{C}\otimes\mathbb{G}}$. As for $\Lambda_{\mathbb{C}}$, it is more interesting to work

with a subset of the calculus rather than with full $\Lambda_{\mathbb{C}\otimes\mathbb{G}}$. In particular, we can consider the fragment defined by following grammar, where ℓ is fixed location and k ranges over locations different from ℓ:

$$\begin{aligned} v, w &::= x \mid \lambda x.e \\ e, f &::= \mathbf{return}\ v \mid vw \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \mid k := v; e \mid \mathbf{tick}(\ell := v; e) \mid \mathbf{get}_\ell(x.e). \end{aligned}$$

The ticking operation is used to count the number of times we store a value in the location ℓ (obviously slightly modifications of the above grammar allow to work with different subsets of $\mathcal{L}$) during the evaluation of a program. To see that, let us modify the big-step semantics of $\Lambda_{\mathbb{M}\otimes\mathbb{G}}$ using judgments of the form $\langle e, \sigma\rangle \Downarrow \langle v, \sigma', n\rangle$, where n is a natural number giving the number of times the location ℓ is used to store a value:

$$\frac{}{\langle \mathbf{return}\ v, \sigma\rangle \Downarrow \langle v, \sigma, 0\rangle} \qquad \frac{\langle e[x := v], \sigma\rangle \Downarrow \langle w, \sigma', n\rangle}{\langle (\lambda x.e)v, \sigma\rangle \Downarrow \langle w, \sigma', n\rangle} \qquad \frac{\langle e, \sigma\rangle \Downarrow \langle w, \sigma'', n\rangle \quad \langle f[x := w], \sigma''\rangle \Downarrow \langle v, \sigma', m\rangle}{\langle \mathbf{let}\ x = e\ \mathbf{in}\ f, \sigma\rangle \Downarrow \langle v, \sigma', n + m\rangle}$$

$$\frac{\langle e[x := w], \sigma\rangle \Downarrow \langle v, \sigma', n\rangle \quad \sigma(\ell) = w}{\langle \mathbf{get}_\ell(x.e), \sigma\rangle \Downarrow \langle v, \sigma', n\rangle} \qquad \frac{\langle e, \sigma[\ell := w]\rangle \Downarrow \langle v, \sigma', n\rangle}{\langle \ell := w; e, \sigma\rangle \Downarrow \langle v, \sigma', n + 1\rangle} \qquad \frac{\langle e, \sigma[k := w]\rangle \Downarrow \langle v, \sigma', n\rangle}{\langle k := w; e, \sigma\rangle \Downarrow \langle v, \sigma', n\rangle}$$

Let us the consider the translation from terms in $\Lambda_{\mathbb{M}\otimes\mathbb{G}}$ to the above fragment of $\Lambda_{\mathbb{C}\otimes\mathbb{G}}$ induced by the rules:

$$\overline{\ell := v; e} \triangleq \mathbf{tick}(\ell := v; \overline{e}) \qquad\qquad \overline{k := v; e} \triangleq k := v; \overline{e}.$$

Then we see that $[\![\overline{e}]\!](\sigma) = \mathit{just}\ (\sigma', n, v)$ if and only if $\langle e, \sigma\rangle \Downarrow \langle v, \sigma', n\rangle$. ⊠

Having defined both the syntax and semantics of Λ_Σ, it is time to study notions of program equivalence and refinement for it. As for Λ_p, we aim to develop a notion of applicative (bi)similarity for Λ_Σ, which we call *effectful applicative (bi)similarity*. As we will see, such a notion can be instantiated to the calculi of previous examples and allows to recover standard notions of applicative (bi)similarity for them. However, in order to develop the theory of effectful applicative (bi)similarity we need a further mathematical tool, namely the notion of a *relator* (Thijs, 1996) or *lax extension* (Barr, 1970; Hofmann et al., 2014). Roughly speaking, the latter is an abstraction of the map $\hat{D}$ we used to define probabilistic applicative similarity for Λ_p axiomatising its relevant structural properties.

Chapter 4

Relators and Relation Lifting

> The third branch of science may be called semiotics, or the doctrine of signs. Because these are mostly words, this part of science is aptly enough termed also logic. The business of this is to study the nature of the signs that the mind makes use of for understanding things and for conveying its knowledge to others. None of the things the mind contemplates is present to the understanding (except itself); so it must have present to it something that functions as a sign or representation of the thing it is thinking about; and this is an idea.
>
> John Locke, An Essay Concerning Human Understanding

In this chapter we introduce what we might say to be *the* central tool we use in this dissertation, namely the notion of a *relator* (Thijs, 1996) or *lax extension* (Barr, 1970). Broadly speaking, a relator Γ for a functor T (on Set) describes a possible way to lift a relation $\mathcal{R}$ between two sets X and Y to a relation $\Gamma\mathcal{R}$ between TX and TY. Clearly, there are many ways to perform such a lifting. Relators provide an axiomatisation of those lifting operations that preserve some natural structural properties.

The notion of a lax extension of a monad has been introduced by Barr (Barr, 1970) in the context of (categorical) topology. Building on previous works by Manes (E. Manes, 1969), Barr characterised topological spaces as lax algebras for the ultrafilter monad in Rel, the category of sets and relations. That is, any topological space can be described as set X together with a (convergence) *relation* $\alpha \subseteq UX \times X$ (where U denotes the carrier of the ultrafilter monad) satisfying suitable *inequalities*. The shift from functions to relations, and from equalities to inequalities, required Barr to solve several difficulties, the biggest one being to 'extend' the ultrafilter monad from Set to Rel in a suitable way.

Barr defined an abstract construction that allows to 'extend' not only the ultrafilter monad from Set to Rel, but also any other monad (resp. functor) satisfying some minimal conditions (notably, preservation of weak pullbacks). Such a construction is nowadays known as *Barr extension* and (the outcome of) its application to a monad (resp. functor) is called the lax extension of the monad (resp. functor), as the

latter is extended from Set to Rel, laxly. Barr extension provides a canonical way to extend monads (resp. functors) from Set to Rel, laxly, but there are many other possible such extensions, which are arguably well behaved. The criteria that qualify such extensions as well behaved can be nicely axiomatised, the resulting notion being the one of a *lax extension* of a monad (resp. functor) (Hoffman, 2015; Hofmann et al., 2014).

Years later, Thijs (Thijs, 1996) introduced the notion of a *relator* (also called relational extension) to study notions of (bi)simulation relation in the abstract setting of universal coalgebra (J. J. M. M. Rutten, 2000). Roughly speaking, given a relator Γ as above and a T-coalgebra (on Set) $\gamma : X \to TX$, one defines a relation $\mathcal{R}$ on X to be a simulation if $x\ \mathcal{R}\ y \implies \gamma(x)\ \Gamma\mathcal{R}\ \gamma(y)$. The defining properties of relators ensure that there exists a largest simulation, and that the latter is a reflexive and transitive relation. Additionally, it is easy to extract a notion of bisimilarity from the one of similarity and to prove that the latter is an equivalence relation. Since then, the 'relator approach' (as well as related 'relation lifting' approaches (Hermida & Jacobs, 1998; Jacobs, 2016; Kurz & Velebil, 2016)) has been extremely influent in the field of (universal) coalgebra, as witnessed by the numerous works on the subject (see (Jacobs, 2016; Kurz & Velebil, 2016) for further references).

In this dissertation we use relators to define several notions of simulation and bisimulation (notably applicative, monadic, and normal form (bi)simulation) for higher-order effectful languages, as well as other notions of program equivalence and refinement (notably contextual and CIU approximation and equivalence). Our main results state that the structural properties of relators ensure precongruence and congruence properties of the obtained notions of program refinement and equivalence, respectively. Therefore, the general theory of relators qualifies as a formal toolkit that deserves a spot in the semanticist's arsenal.

4.1 Preliminaries

Before defining relators formally, it is useful to recall some background notions of relational calculus. That will allow us to reduce the complexity of some proofs by means of simple, pointfree calculations. Contrary to standard, set-based presentations of relations, we use an 'algebraic' notation for relations and their algebra. This choice has the advantage of highlighting the connection between relations and abstract notions of distance (the central theme of the second part of this work).

Throughout the rest of this dissertation, let $\mathbb{2}$ denote the complete lattice $\langle 2, \leq \rangle$ of boolean values[1] with $2 \triangleq \{\mathsf{true}, \mathsf{false}\}$. Moreover, we tacitly assume all functors and monads to be functors and monads on Set, unless differently specified.

Definition 18. *The category* Rel *has sets as objects and relations as morphisms. A relation $\mathcal{R}$ between two sets X and Y, written as $\mathcal{R} : X \nrightarrow Y$, is a map $\mathcal{R} : X \times Y \to 2$. Given elements $x \in X$, $y \in Y$ we say that x is $\mathcal{R}$-related to y, and write $x\ \mathcal{R}\ y$, if $\mathcal{R}(x, y) = \mathsf{true}$. For any set X the identity relation $\mathsf{I}_X : X \nrightarrow X$ is defined as equality on X. Moreover, for relations $\mathcal{R} : X \nrightarrow Y$ and $\mathcal{S} : Y \nrightarrow Z$, we define the composition $\mathcal{S} \cdot \mathcal{R} : X \nrightarrow Z$ of $\mathcal{S}$ with $\mathcal{R}$ as:*

$$(\mathcal{S} \cdot \mathcal{R})(x, z) \triangleq \bigvee_{y \in Y} \mathcal{R}(x, y) \wedge \mathcal{S}(y, z).$$

Composition of relations is associative, and I *is the unit of composition, so that* Rel *is indeed a category.*

Remark 4. Notice that relations are not defined as objects of Set (namely as subsets of cartesian products). We keep explicit the distinction between a relation, which is a morphism in Rel, and its graph[2], which is an object in Set.

[1] The notation for joins and meets is as usual.

[2] For $\mathcal{R} : X \to Y$ define the graph $\mathcal{GR}$ of $\mathcal{R}$ as $\mathcal{GR} \triangleq \{(x, y) \mid \mathcal{R}(x, y) = \mathsf{true}\}$.

Rel is a monoidal category with unit given by the one-element set and tensor product given by cartesian product of sets, with $\mathcal{R} \times \mathcal{S} : X \times Y \nrightarrow X' \times Y'$ defined pointwise via binary meet. Moreover, for all sets X, Y, the hom-set $\mathsf{Rel}(X, Y)$ inherits a complete lattice structure from $\mathbb{2}$ pointwise. As it is customary, we denote the resulting order on $\mathsf{Rel}(X, Y)$ by $\subseteq$, whereas meets and joins are denoted by $\bigcap$ and $\bigcup$, respectively. The complete lattice structure of hom-sets nicely interacts with the monoid structure of relation composition, meaning that Rel forms a *quantaloid* (Hofmann et al., 2014). In particular, for all relations $\mathcal{R} : X \nrightarrow Y$, $\mathcal{S}_i : Y \nrightarrow Z$ $(i \in I)$, and $\mathcal{Q} : Z \nrightarrow W$ the following distributivity laws hold:

$$\mathcal{Q} \cdot (\bigcup_{i \in I} \mathcal{S}_i) = \bigcup_{i \in I} (\mathcal{Q} \cdot \mathcal{S}_i),$$
$$(\bigcup_{i \in I} \mathcal{S}_i) \cdot \mathcal{R} = \bigcup_{i \in I} (\mathcal{S}_i \cdot \mathcal{R}).$$

Distributivity implies monotonicity of relation composition in both arguments. In fact, assuming $\mathcal{R}_1 \subseteq \mathcal{R}_2$, i.e. $\mathcal{R}_2 = \bigcup_{i \in \{1,2\}} \mathcal{R}_i$ we have:

$$\mathcal{S} \cdot \mathcal{R}_2 = \mathcal{S} \cdot \bigcup_{i \in \{1,2\}} \mathcal{R}_i = \bigcup_{i \in \{1,2\}} (\mathcal{S} \cdot \mathcal{R}_i) \supseteq \mathcal{S} \cdot \mathcal{R}_1.$$

In a similar fashion we can prove that $\mathcal{S}_1 \subseteq \mathcal{S}_2$ implies $\mathcal{S}_1 \cdot \mathcal{R} \subseteq \mathcal{S}_2 \cdot \mathcal{R}$, and thus monotonicity of composition in both arguments.

We also notice that Rel is self-dual, since for all sets X, Y there is a bijection $-^\circ$ between $\mathsf{Rel}(X, Y)$ and $\mathsf{Rel}(Y, X)$ mapping each relation $\mathcal{R} : X \nrightarrow Y$ to its dual (or opposite) relation $\mathcal{R}^\circ : Y \nrightarrow X$ defined by $\mathcal{R}^\circ(y, x) \triangleq \mathcal{R}(x, y)$. It is a routine exercise to verify that $-^\circ$ is monotone and satisfies the following identities (the third one stating that $-^\circ$ is an involution):

$$(\mathcal{S} \cdot \mathcal{R})^\circ = \mathcal{R}^\circ \cdot \mathcal{S}^\circ$$
$$\mathsf{I}^\circ = \mathsf{I}$$
$$(\mathcal{R}^\circ)^\circ = \mathcal{R}.$$

Additionally, there is a map $-_\circ$ from Set to Rel that interprets the graph of a function $f : X \to Y$ (the latter being a morphism in Set) as the relation $f_\circ : X \nrightarrow Y$ defined by

$$f_\circ(x, y) = \begin{cases} \mathsf{true} & \text{if } f(x) = y \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

The map $-_\circ$ defines a functor from Set to Rel, so that $(g \cdot f)_\circ = g_\circ \cdot f_\circ$ and $1_\circ = \mathsf{I}$. Moreover, we see that $-_\circ$ is faithful. As a consequence, we write $f : X \to Y$ in place of $f_\circ : X \nrightarrow Y$, unless we work in Set and Rel at the same time (that will only happen in Subsection 4.3.1). The notation used for the graph functor works as reminder for the following inequalities, for $f : X \to Y$:

$$\mathsf{I}_X \subseteq f^\circ \cdot f_\circ$$
$$f_\circ \cdot f^\circ \subseteq \mathsf{I}_Y.$$

It is useful to keep in mind the pointwise reading of relations of the form $g^\circ \cdot \mathcal{S} \cdot f$, for a relation $\mathcal{S} : Z \nrightarrow W$ and functions $f : X \to Z$, $g : Y \to W$:

$$(g^\circ \cdot \mathcal{S} \cdot f)(x, y) = \mathcal{S}(f(x), g(y)).$$

Given $\mathcal{R} : X \nrightarrow Y$ we can thus express a generalised monotonicity condition in pointfree fashion as:

$$\mathcal{R} \subseteq g^\circ \cdot \mathcal{S} \cdot f.$$

Indeed, taking $f = g$, we obtain standard monotonicity of f. We will make extensively use of the following *adjunction rules* (Hofmann et al., 2014), for $f : X \to Y$, $g : Y \to Z$, $\mathcal{R} : X \nrightarrow Y$, $\mathcal{S} : Y \nrightarrow Z$, and $\mathcal{Q} : X \nrightarrow Z$:

$$g \cdot \mathcal{R} \subseteq \mathcal{Q} \iff \mathcal{R} \subseteq g^\circ \cdot \mathcal{Q} \tag{adj 1}$$
$$\mathcal{Q} \cdot f^\circ \subseteq \mathcal{S} \iff \mathcal{Q} \subseteq \mathcal{S} \cdot f. \tag{adj 2}$$

Using (adj 1) and (adj 2) we see that generalised monotonicity $\mathcal{R} \subseteq g^\circ \cdot \mathcal{S} \cdot f$ can be equivalently expressed via the following lax commutative diagram:

$$\begin{array}{ccc} X & \xrightarrow{f} & Z \\ {\scriptstyle \mathcal{R}}\downarrow & \subseteq & \downarrow{\scriptstyle \mathcal{S}} \\ Y & \xrightarrow[g]{} & W \end{array}$$

The diagram acts as a graphical representation of the expression $g \cdot \mathcal{R} \subseteq \mathcal{S} \cdot f$, which, by (adj 1), is equivalent to $\mathcal{R} \subseteq g^\circ \cdot \mathcal{R} \cdot f$.

Finally, since we are interested in preorder and equivalence relations, we recall that we can define the notions of a reflexive, a transitive, and a symmetric relation pointfree. In fact, a relation $\mathcal{R} : X \nrightarrow X$ is reflexive if $\mathsf{I}_X \subseteq \mathcal{R}$, transitive if $\mathcal{R} \cdot \mathcal{R} \subseteq \mathcal{R}$, and symmetric if $\mathcal{R} \subseteq \mathcal{R}^\circ$.

4.2 Relators

The notion of a relator (Barr, 1970; Thijs, 1996) Γ for a functor T is an abstraction meant to capture the possible ways a relation $\mathcal{R} : X \nrightarrow Y$ can be lifted to a relation $\Gamma\mathcal{R} : TX \nrightarrow TY$.

Definition 19. *A* relator *for a functor T is a set-indexed family of maps $(\mathcal{R} : X \nrightarrow Y) \mapsto (\Gamma\mathcal{R} : TX \nrightarrow TY)$ satisfying conditions* (rel 1)-(rel 4)*. We say that Γ is* conversive *if it additionally satisfies condition* (rel 5)*.*

$$\mathsf{I}_{TX} \subseteq \Gamma(\mathsf{I}_X) \tag{rel 1}$$
$$\Gamma\mathcal{S} \cdot \Gamma\mathcal{R} \subseteq \Gamma(\mathcal{S} \cdot \mathcal{R}) \tag{rel 2}$$
$$Tf \subseteq \Gamma f, \quad (Tf)^\circ \subseteq \Gamma f^\circ \tag{rel 3}$$
$$\mathcal{R} \subseteq \mathcal{S} \implies \Gamma\mathcal{R} \subseteq \Gamma\mathcal{S} \tag{rel 4}$$
$$\Gamma(\mathcal{R}^\circ) = (\Gamma\mathcal{R})^\circ. \tag{rel 5}$$

Conditions (rel 1), (rel 2), and (rel 4) are rather standard[3]. As we will see, condition (rel 4) makes the defining endofunction of (bi)simulation relations monotone, whereas conditions (rel 1) and (rel 2) make notions of (bi)similarity reflexive and transitive. Similarly, condition (rel 5) makes notions of bisimilarity symmetric. Condition (rel 3), which actually consists of two conditions, states that relators behave as expected when acting on (graphs of) functions. In (P. Levy, 2011) a kernel preservation condition is required in place of (rel 3). Such condition is also known as *stability* in (Hughes & Jacobs, 2004). Stability requires the equality

$$\Gamma(g^\circ \cdot \mathcal{R} \cdot f) = (Tg)^\circ \cdot \Gamma\mathcal{R} \cdot Tf \tag{stability}$$

to hold. It is easy to see that a relator always satisfies stability.

Proposition 7. *Conditions* (rel 1)-(rel 4) *implies* (stability)*.*

[3]Notice that since $\mathsf{I} = (1)_\circ$ we can derive condition (rel 1) from condition (rel 3).

Proof. Proving that $(Tg)^\circ \cdot \Gamma\mathcal{R} \cdot Tf \subseteq \Gamma(g^\circ \cdot \mathcal{R} \cdot f)$ is straightforward. Indeed we have:

$$\begin{aligned}(Tg)^\circ \cdot \Gamma\mathcal{R} \cdot Tf &\subseteq \Gamma g^\circ \cdot \Gamma\mathcal{R} \cdot \Gamma f \\ &\quad \text{[By (rel 3)]} \\ &\subseteq \Gamma(g^\circ \cdot \mathcal{R} \cdot f). \\ &\quad \text{[By (rel 2)]}\end{aligned}$$

To prove $\Gamma(g^\circ \cdot \mathcal{R} \cdot f) \subseteq (Tg)^\circ \cdot \Gamma\mathcal{R} \cdot Tf$ we reason as follows:

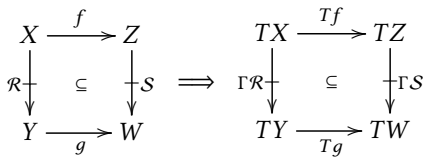
$$\begin{aligned}\Gamma(g^\circ \cdot \mathcal{R} \cdot f) &\subseteq (Tg)^\circ \cdot Tg \cdot \Gamma(g^\circ \cdot \mathcal{R} \cdot f) \cdot (Tf)^\circ \cdot Tf \\ &\quad \text{[Since } \forall h.\ \mathsf{I} \subseteq h^\circ \cdot h\text{]} \\ &\subseteq (Tg)^\circ \cdot \Gamma g \cdot \Gamma(g^\circ \cdot \mathcal{R} \cdot f) \cdot \Gamma f^\circ \cdot Tf \\ &\quad \text{[By (rel 3)]} \\ &\subseteq (Tg)^\circ \cdot \Gamma(g \cdot g^\circ \cdot \mathcal{R} \cdot f \cdot f^\circ) \cdot Tf \\ &\quad \text{[By (rel 2)]} \\ &\subseteq (Tg)^\circ \cdot \Gamma\mathcal{R} \cdot Tf. \\ &\quad \text{[Since } \forall h.\ h \cdot h^\circ \subseteq \mathsf{I}\text{]}\end{aligned}$$

□

We also notice that, since relators are monotone, stability gives the following implication

$$\mathcal{R} \subseteq g^\circ \cdot \mathcal{S} \cdot f \implies \Gamma\mathcal{R} \subseteq (Tg)^\circ \cdot \Gamma\mathcal{S} \cdot Tf,$$

which can be diagrammatically expressed as:



Finally, we observe that any relator Γ for T induces an endomap T_Γ on Rel that acts as T on sets and as Γ on relations. It is easy to check that conditions in Definition 19 makes T_Γ a *lax endofunctor*. When T_Γ is a functor, we say that Γ is *functorial*. Concretely, this is the case if conditions (rel 1) and (rel 2) are equalities, i.e. if the following identities hold:

$$\mathsf{I}_{TX} = \Gamma(\mathsf{I}_X) \qquad \text{(rel funct 1)}$$

$$\Gamma\mathcal{S} \cdot \Gamma\mathcal{R} = \Gamma(\mathcal{S} \cdot \mathcal{R}). \qquad \text{(rel funct 2)}$$

Before giving examples of relators it is useful to observe that the collection of relators is closed under specific operations (see (P. Levy, 2011) for a proof of the following proposition).

Proposition 8. *Let T, U be functors, and let UT denote their composition. Moreover, let Γ, Δ be relators for T and U, respectively, and $\{\Gamma_i\}_{i\in I}$ be a family of relators for T. Then:*

1. *The map $\Delta\Gamma$ defined by $\Delta\Gamma\mathcal{R} \triangleq \Delta(\Gamma\mathcal{R})$ is a relator for UT.*
2. *The map $\bigwedge_{i\in I} \Gamma_i$ defined by $(\bigwedge_{i\in I} \Gamma_i)\mathcal{R} \triangleq \bigcap_{i\in I} \Gamma_i\mathcal{R}$ is a relator for T.*
3. *The map Γ° defined by $\Gamma^\circ\mathcal{R} \triangleq (\Gamma\mathcal{R}^\circ)^\circ$ is a relator for T.*

4. *The map* $\Gamma \wedge \Gamma^\circ$ *is the greatest conversive relator smaller than* Γ*, according to the pointwise order.*

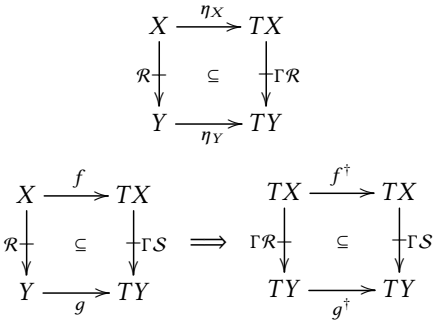
Relators provide a powerful abstraction of notions of 'relation lifting', as witnessed by the numerous examples of relators we will discuss in Section 4.3. Before discussing such examples, we introduce the notion of a *relator for a monad* or *lax extension of a monad*. In fact, according to Definition 19 relators extend functors from Set to Rel, laxly. However, we defined notions of computation (and thus modelled computational effects) as monads. Therefore, it seems natural to require relators to extend *monads* (and not just functors) from Set to Rel, laxly. Requiring such a condition is exactly what is needed to make applicative (as well as normal form) similarity a precongruence relation. As the reader will notice, we already met the defining conditions of a lax extension of a monad in our analysis of the precongruence theorem for probabilistic applicative similarity (Chapter 2). In fact, such conditions provided exactly those structural properties of $\hat{D}$ that made the semantics of return and sequencing well behaved with respect to probabilistic applicative similarity.

Definition 20. *Let* $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ *be a monad, and* Γ *be a relator for* T*. We say that* Γ *is a relator for* $\mathbb{T}$ *if it satisfies the following conditions:*

$$\mathcal{R} \subseteq \eta_Y^\circ \cdot \Gamma\mathcal{R} \cdot \eta_X, \tag{rel unit}$$

$$\mathcal{R} \subseteq g^\circ \cdot \Gamma\mathcal{S} \cdot f \implies \Gamma\mathcal{R} \subseteq (g^\dagger)^\circ \cdot \Gamma\mathcal{S} \cdot f^\dagger. \tag{rel bind}$$

We can express conditions (rel unit) and (rel bind) diagrammatically as follows:

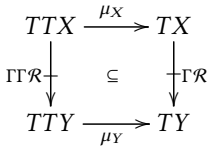


It is an easy exercise in algebra to verify that a relator for a monad $\mathbb{T}$ laxly extends $\mathbb{T}$ to Rel, in the sense that the natural transformations $\eta_X : X \to T_\Gamma X$ and $\mu_X : T_\Gamma T_\Gamma X \to T_\Gamma X$ are oplax (see e.g. (Hofmann et al., 2014)).

Proposition 9. *Let* $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ *be a monad, and* Γ *a relator for* T*. Then property* (rel bind) *is equivalent to the validity of the following set-indexed family of inequalities*

$$\mu_Y \cdot \Gamma\Gamma\mathcal{R} \subseteq \Gamma\mathcal{R} \cdot \mu_X, \tag{rel } \mu)$$

which express the following (lax) commutative condition:



Proof. We show that (rel bind) implies (rel μ)[4].

$$\begin{aligned}
\mu_Y \cdot \Gamma\Gamma\mathcal{R} \subseteq \Gamma\mathcal{R} \cdot \mu_X \iff & \Gamma\Gamma\mathcal{R} \subseteq \mu_Y^\circ \cdot \Gamma\mathcal{R} \cdot \mu_X \\
& \text{[By (adj 1)]} \\
\iff & \Gamma\Gamma\mathcal{R} \subseteq (1_{TY}^\dagger)^\circ \cdot \Gamma\mathcal{R} \cdot (1_{TX}^\dagger) \\
& \text{[Since } \mu_Z = 1_{TZ}^\dagger\text{]} \\
\iff & \Gamma\mathcal{R} \subseteq 1_{TY}^\circ \cdot \Gamma\mathcal{R} \cdot 1_{TX} \\
& \text{[By (rel bind)]} \\
\iff & \Gamma\mathcal{R} \subseteq 1_{TY}^\circ \cdot \Gamma\mathcal{R} \cdot 1_{TX} \\
& \text{[Since } 1^\circ = 1\text{]}
\end{aligned}$$

Vice versa, we prove (rel μ) implies (rel bind). Assume $\mathcal{R} \subseteq g^\circ \cdot \Gamma\mathcal{S} \cdot f$. We have:

$$\begin{aligned}
\Gamma\mathcal{R} \subseteq (g^\dagger)^\circ \cdot \Gamma\mathcal{S} \cdot f^\dagger \iff & \Gamma\mathcal{R} \subseteq (\mu \cdot Tg)^\circ \cdot \Gamma\mathcal{S} \cdot (\mu \cdot Tf) \\
& \text{[Since } h^\dagger = \mu \cdot Th\text{]} \\
\iff & \Gamma\mathcal{R} \subseteq Tg^\circ \cdot \mu^\circ \cdot \Gamma\mathcal{S} \cdot \mu \cdot Tf \\
& \text{[Since } (k \cdot h)^\circ = h^\circ \cdot k^\circ\text{]} \\
\iff & \Gamma\mathcal{R} \subseteq Tg^\circ \cdot \Gamma\Gamma\mathcal{S} \cdot Tf \\
& \text{[By (rel } \mu\text{) and (adj 1)]} \\
\iff & \Gamma\mathcal{R} \subseteq \Gamma(g^\circ \cdot \Gamma\mathcal{S} \cdot f) \\
& \text{[By (stability)]} \\
\iff & \mathcal{R} \subseteq g^\circ \cdot \Gamma\mathcal{S} \cdot f. \\
& \text{[By (rel 4)]}
\end{aligned}$$

□

Finally we remark that Proposition 8 does not generalise to relators for monads, as the composition of the carriers of two monads does not necessarily carry a monad structure. Nonetheless, we can still prove that the collection of relators for a given monad is closed under specific operations.

Proposition 10. *Given a monad* $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ *and relators* Γ, Δ *for it, both* $\Gamma \wedge \Delta$ *and* Γ° *are relators for* $\mathbb{T}$*. In particular,* $\Gamma \wedge \Gamma^\circ$ *is the largest conversive relator for* $\mathbb{T}$ *contained in* Γ*.*

Proof. The proof is straightforward. As an example, we show that $\Lambda \triangleq \Gamma \wedge \Delta$ is a relator for $\mathbb{T}$. We use Proposition 9 and show that

$$\mu_y \cdot \Lambda\Lambda\mathcal{R} \subseteq \Lambda\mathcal{R} \cdot \mu_X.$$

[4] Recall that $1_\circ = \mathsf{I}$, so that we write e.g. $1 \cdot \mathcal{R}$ in place of $1_\circ \cdot \mathcal{R} = \mathsf{I} \cdot \mathcal{R}$.

We have:

$$\begin{aligned}
\mu_y \cdot \Lambda\Lambda\mathcal{R} \subseteq \Lambda\mathcal{R} \cdot \mu_X \iff & \Lambda\Lambda\mathcal{R} \subseteq \mu_Y^\circ \cdot \Lambda\mathcal{R} \cdot \mu_X \\
& \text{[By (adj 1)]} \\
\iff & \Gamma\Gamma\mathcal{R} \cap \Gamma\Delta\mathcal{R} \cap \Delta\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R} \subseteq \mu_Y^\circ \cdot (\Gamma\mathcal{R} \cap \Delta\mathcal{R}) \cdot \mu_X \\
& \text{[Since } \Lambda = \Gamma \wedge \Delta\text{]} \\
\impliedby & \Gamma\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R} \subseteq \mu_Y^\circ \cdot (\Gamma\mathcal{R} \cap \Delta\mathcal{R}) \cdot \mu_X \\
& \text{[Since } \Gamma\Gamma\mathcal{R} \cap \Gamma\Delta\mathcal{R} \cap \Delta\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R} \subseteq \Gamma\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R}\text{]} \\
\iff & \mu_Y \cdot (\Gamma\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R}) \cdot \mu_X^\circ \subseteq \Gamma\mathcal{R} \cap \Delta\mathcal{R}. \\
& \text{[By (adj 1) and (adj 2)]}
\end{aligned}$$

To prove the latter inequality, it is sufficient to prove

$$\begin{aligned}
\mu_Y \cdot (\Gamma\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R}) \cdot \mu_X^\circ \subseteq \Gamma\mathcal{R} \\
\mu_Y \cdot (\Gamma\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R}) \cdot \mu_X^\circ \subseteq \Delta\mathcal{R}.
\end{aligned}$$

We prove the first one (the second one is proved in the same fashion). We reason as follows:

$$\begin{aligned}
\mu_Y \cdot (\Gamma\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R}) \cdot \mu_X^\circ \subseteq \Gamma\mathcal{R} \iff & \Gamma\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R} \subseteq \mu_Y^\circ \cdot \Gamma\mathcal{R} \cdot \mu_X \\
& \text{[By (adj 1) and (adj 2)]} \\
\impliedby & \Gamma\Gamma\mathcal{R} \subseteq \mu_Y^\circ \cdot \Gamma\mathcal{R} \cdot \mu_X \\
& \text{[Since } \Gamma\Gamma\mathcal{R} \cap \Delta\Delta\mathcal{R} \subseteq \Gamma\Gamma\mathcal{R}\text{]} \\
\iff & \text{(rel } \mu\text{).} \\
& \text{[By (adj 1)]}
\end{aligned}$$

□

4.3 Relevant Examples

We now examine examples of relators for the monads studied in Chapter 3.

Example 27 (Partiality monad)**.** For the partiality monad $\mathbb{M}$ we define the set-indexed families of maps $\hat{\mathbb{M}}, \check{\mathbb{M}} : \mathsf{Rel}(X, Y) \to \mathsf{Rel}(MX, MY)$ as follows:

$$\begin{aligned}
\mathfrak{x}\ \hat{\mathbb{M}}\mathcal{R}\ \mathfrak{y} &\overset{\triangle}{\iff} (\mathfrak{x} = \bot) \vee (\exists x \in X.\ \exists y \in Y.\ \mathfrak{x} = \mathit{just}\ x \wedge \mathfrak{y} = \mathit{just}\ y \wedge x\ \mathcal{R}\ y). \\
\mathfrak{x}\ \check{\mathbb{M}}\mathcal{R}\ \mathfrak{y} &\overset{\triangle}{\iff} (\mathfrak{y} = \bot) \vee (\exists x \in X.\ \exists y \in Y.\ \mathfrak{x} = \mathit{just}\ x \wedge \mathfrak{y} = \mathit{just}\ y \wedge x\ \mathcal{R}\ y).
\end{aligned}$$

The mapping $\hat{\mathbb{M}}$ describes the structure of the usual *simulation* clause for partial computations, whereas $\check{\mathbb{M}}$ describes the corresponding *co-simulation* clause. It is easy to see that $\hat{\mathbb{M}}$ is a relator for $\mathbb{M}$ and that $\check{\mathbb{M}} = \hat{\mathbb{M}}^\circ$. By Proposition 10, the map $\hat{\mathbb{M}} \wedge \check{\mathbb{M}}$ is a conversive relator for $\mathbb{M}$. It is immediate to see that the latter relator describes the structure of the usual *bisimulation* clause for partial computations. ⊠

Example 28 (Exception monad)**.** Proceeding as in Example 27, we define a relator for the exception monad $\mathbb{E}$ as the mapping $\hat{\mathbb{E}} : \mathsf{Rel}(X, Y) \to \mathsf{Rel}(EX, EY)$ defined by:

$$\mathfrak{x}\ \hat{\mathbb{E}}\mathcal{R}\ \mathfrak{y} \overset{\triangle}{\iff} (\mathfrak{x} = \mathsf{e} \implies \mathfrak{y} = \mathsf{e}) \wedge \forall x \in X.\ (\mathfrak{x} = x \implies \exists y \in Y.\ (\mathfrak{y} = y \wedge x\ \mathcal{R}\ y)).$$

Easy calculations show that indeed $\hat{\mathbb{E}}$ is a relator for the monad $\mathbb{E}$. As before, $\hat{\mathbb{E}}$ induces a canonical conversive relator, namely $\hat{\mathbb{E}} \wedge \hat{\mathbb{E}}^\circ$. We now generalise Proposition 3 and Proposition 5 to take into account relators.

Proposition 11. *Given a monad $\mathbb{T} = \langle T, \mathtt{T}, -^{\mathtt{T}} \rangle$ and a relator $\hat{\mathbb{T}}$ for $\mathbb{T}$, define the sum $\mathbb{T}\hat{\mathbb{E}}$ of $\hat{\mathbb{T}}$ and $\hat{\mathbb{E}}$ as $\mathbb{T}\hat{\mathbb{E}} \triangleq \hat{\mathbb{T}}\hat{\mathbb{E}}$. Then $\mathbb{T}\hat{\mathbb{E}}$ is a relator for $\mathbb{T}\mathbb{E}$.*

The proof of Proposition 11 goes as follows. By Proposition 8, we know that $\hat{\mathbb{T}}\hat{\mathbb{E}}$ is a relator for the functor TE. Therefore, it remains to show that $\hat{\mathbb{T}}\hat{\mathbb{E}}$ is also a relator for $\mathbb{T}\mathbb{E}$. Proving $\hat{\mathbb{T}}\hat{\mathbb{E}}$ to satisfy condition (rel unit) is straightforward. We show that it satisfies condition (rel bind) too. To see that, we observe that $\hat{\mathbb{T}}\hat{\mathbb{E}}$ satisfies the following 'intermediate' condition:

$$\mathcal{R} \subseteq g^\circ \cdot \hat{\mathbb{T}}\hat{\mathbb{M}}\mathcal{S} \cdot f \implies \hat{\mathbb{E}}\mathcal{R} \subseteq g_E^\circ \cdot \hat{\mathbb{T}}\hat{\mathbb{E}}\mathcal{S} \cdot f_E, \tag{4.1}$$

or, diagrammatically:

$$\begin{array}{ccc} X & \xrightarrow{f} & TEX \\ {\scriptstyle\mathcal{R}}\downarrow & \subseteq & \downarrow{\scriptstyle\hat{\mathbb{T}}\hat{\mathbb{E}}\mathcal{S}} \\ Y & \xrightarrow[g]{} & TEY \end{array} \implies \begin{array}{ccc} EX & \xrightarrow{f_E} & TEX \\ {\scriptstyle\hat{\mathbb{E}}\mathcal{R}}\downarrow & \subseteq & \downarrow{\scriptstyle\hat{\mathbb{T}}\hat{\mathbb{E}}\mathcal{S}} \\ EY & \xrightarrow[g_E]{} & TEY \end{array}$$

In fact, if $\mathfrak{x}\ \hat{\mathbb{E}}\mathcal{R}\ \mathfrak{y}$, then either $\mathfrak{x} = \mathsf{e}$ or $\mathfrak{x} = x$, for some $x \in X$. In the first case we must have $\mathfrak{y} = \mathsf{e}$ too, and thus $f_E(\mathfrak{x}) = \mathtt{T}_{X+E}(\mathsf{e})$ and $g_E(\mathfrak{y}) = \mathtt{T}_{Y+E}(\mathsf{e})$. We can conclude the thesis since $\hat{\mathbb{T}}$ satisfies condition (rel unit) (and obviously $\mathsf{e}\ \hat{\mathbb{E}}\mathcal{R}\ \mathsf{e}$ holds). In the second case we have $\mathfrak{x} = x$, for some $x \in X$. As a consequence, since $\mathfrak{x}\ \hat{\mathbb{E}}\mathcal{R}\ \mathfrak{y}$, we also have $\mathfrak{y} = y$, for some $y \in Y$ such that $x\,\mathcal{R}\,y$. Therefore, $f_E(\mathfrak{x}) = f(x)$ and $g_E(\mathfrak{y}) = g(x)$, so that the thesis follows by the main hypothesis. Using (4.1) we conclude the main thesis as follows:

$$\begin{aligned} \mathcal{R} \subseteq g^\circ \cdot \hat{\mathbb{T}}\hat{\mathbb{E}}\mathcal{S} \cdot f &\implies \hat{\mathbb{E}}\mathcal{R} \subseteq g_E^\circ \cdot \hat{\mathbb{T}}\hat{\mathbb{E}}\mathcal{S} \cdot f_E \\ &\quad \text{[By (4.1)]} \\ &\implies \hat{\mathbb{T}}\hat{\mathbb{E}}\mathcal{R} \subseteq (g_E^{\mathtt{T}})^\circ \cdot \hat{\mathbb{T}}\hat{\mathbb{E}}\mathcal{S} \cdot f_E^{\mathtt{T}} \\ &\quad \text{[Since } \hat{\mathbb{T}} \text{ satisfies (rel bind)]} \\ &\iff \mathbb{T}\hat{\mathbb{E}}\mathcal{R} \subseteq (g^{\mathtt{T}\mathbb{E}})^\circ \cdot \mathbb{T}\hat{\mathbb{E}}\mathcal{S} \cdot f^{\mathtt{T}\mathbb{E}}. \\ &\quad \text{[Since } h^{\mathtt{T}\mathbb{E}} = h_E^{\mathtt{T}}\text{]} \end{aligned}$$

⊠

Example 29 (Nondeterministic monad)**.** For the monad $\mathbb{F}$ define maps $\hat{\mathbb{F}}, \check{\mathbb{F}} : \mathsf{Rel}(X, Y) \to \mathsf{Rel}(FX, FY)$ as:

$$\mathfrak{X}\ \hat{\mathbb{F}}\mathcal{R}\ \mathfrak{Y} \overset{\triangle}{\iff} \forall x \in \mathfrak{X}.\ \exists y \in \mathfrak{Y}.\ x\,\mathcal{R}\,y.$$
$$\mathfrak{X}\ \check{\mathbb{F}}\mathcal{R}\ \mathfrak{Y} \overset{\triangle}{\iff} \forall y \in \mathfrak{Y}.\ \exists x \in \mathfrak{X}.\ x\,\mathcal{R}\,y.$$

It is not hard to see that the relator $\hat{\mathbb{F}}$ describes Milner's simulation clause for (unlabelled) transition systems. We immediately see that $\check{\mathbb{F}} = \hat{\mathbb{F}}^\circ$, and that $\hat{\mathbb{F}} \wedge \hat{\mathbb{F}}^\circ$ describes the bisimulation clause for transition systems. Finally, easy calculations show that $\mathbb{F}$ is indeed a relator for $\mathbb{F}$.

Since $\mathbb{M}$ has essentially the same structure of $\mathbb{E}$, we can sum $\hat{\mathbb{F}}$ and $\hat{\mathbb{M}}$ obtaining a relator $\mathbb{F}\hat{\mathbb{M}} \triangleq \hat{\mathbb{F}}\mathbb{M}$ for $\mathbb{F}\mathbb{M}$. Spelling out the definition we obtain:

$$\mathfrak{X}\ \mathbb{F}\hat{\mathbb{M}}\mathcal{R}\ \mathfrak{Y} \iff \forall \mathfrak{x} \in \mathfrak{X}.\ (\mathfrak{x} = \mathit{just}\ x \implies \exists \mathfrak{y} \in \mathfrak{Y}.\ \mathfrak{y} = \mathit{just}\ y \wedge x\,\mathcal{R}\,y).$$

Notice that $\{\bot\}\,\mathbb{F}\hat{\mathbb{M}}\mathcal{R}\,\mathfrak{Y}$ always holds. The relator $\mathbb{F}\hat{\mathbb{M}}$ can be used to test nondeterministic computations for *may convergence*, the resulting similarity notion being known as *lower similarity* (S. Lassen, 1998b;

C. L. Ong, 1993). Additionally, working with the conversive relator $\hat{\mathbb{F}\mathbb{M}} \wedge \hat{\mathbb{F}\mathbb{M}}^{\circ}$ one recovers the notion of *lower bisimilarity*. Another interesting relator for $\mathbb{F}\mathbb{M}$ is the given by the map $\overline{\mathbb{F}\mathbb{M}}$ defined as:

$$\mathfrak{X}\ \overline{\mathbb{F}\mathbb{M}}\mathcal{R}\ \mathfrak{Y} \overset{\triangle}{\iff} (\bot \notin \mathfrak{X} \implies \bot \notin \mathfrak{Y}) \wedge (\forall y \in Y.\ just\ y \in \mathfrak{Y} \implies \exists x \in X.\ just\ x \in \mathfrak{X} \wedge x\ \mathcal{R}\ y).$$

It is not hard to see that $\overline{\mathbb{F}\mathbb{M}}$ is indeed a relator for $\mathbb{F}\mathbb{M}$ (the reader can also consult (P. Levy, 2011) for further details on $\overline{\mathbb{F}\mathbb{M}}$ as well as other examples of relators for *FM*). The relator $\overline{\mathbb{F}\mathbb{M}}$ tests nondeterministic computations for *must convergence*, the associated notion of similarity being *upper similarity* (S. Lassen, 1998b; C. L. Ong, 1993). As before, working with the conversive relator $\overline{\mathbb{F}\mathbb{M}} \wedge \overline{\mathbb{F}\mathbb{M}}^{\circ}$ we recover the notion of *upper bisimilarity*.

⊠

Example 30 (Distribution monad)**.** Following our working example Λ_p, we define a relator for the monad $\mathbb{D}$ relying on the notion of a *coupling* and results from optimal transport (Villani, 2008).

Definition 21. *Given two distributions $\mu \in D(X)$, $\nu \in D(Y)$, a* coupling *for μ and ν is a joint distribution $\omega \in D(X \times Y)$ such that:*

$$\mu = \sum_{y \in Y} \omega(-, y)$$
$$\nu = \sum_{x \in X} \omega(x, -).$$

We denote the set of couplings of μ and ν by $\Omega(\mu, \nu)$ and immediately observe that $\Omega(\mu, \nu)$ is always non-empty. In fact, the joint distribution ω defined by $\omega(x, y) \triangleq \mu(x) \cdot \nu(x)$ is a coupling of μ and ν.

Remark 5 (Couplings as transportation plans)**.** Given distributions $\mu \in DX$, $\nu \in DY$, the notion of a coupling $\omega \in \Omega(\mu, \nu)$ formalises the informal notion of a transportation plan. Thinking to a distribution μ as assigning weight to points in X, then a transportation plan from μ to ν is a mapping specifying how to move weights from X to Y in such a way that μ is 'transformed' into ν. Formally, a transportation plan from μ to ν is a family of maps $\pi_x : Y \to [0, 1]$ specifying for each $y \in Y$ how much weight of the weight $\mu(x)$ of x has to be moved to y. Accordingly to such a reading, we have to impose π_x some constraints. First of all, π_x does not move more weight than the available one: actually, since we want to move the whole weight mass μ, we require π to transport the whole μ. Formally, we require $\sum_y \pi_x(y) = \mu(x)$, for all $x \in X$.

Applying π to μ we obtain a new distribution (a new mass, so to speak) on Y, denoted by $T_\pi(\mu)$, and defined as:

$$T_\pi(\mu)(y) \triangleq \sum_{x \in X} \pi_x(y).$$

Obviously, since π has to transform μ into ν, we require $T_\pi(\mu) = \nu$. At this point it is straightforward to see that any transportation plan π from μ to ν induces a coupling $\underline{\pi} \in \Omega(\mu, \nu)$ defined by $\underline{\pi}(x, y) \triangleq \pi_x(y)$, and that any coupling $\omega \in \Omega(\mu, \nu)$ induces a transportation plan $\overline{\omega}$ defined by $\overline{\omega}_x(y) \triangleq \omega(x, y)$, so that the notions of a transportation plan and of a coupling are equivalent.

Following Remark 5, we can define a relator for $\mathbb{D}$ simply requiring transportation plans to move weight only between related points (that is, if $\pi_x(y) > 0$, then x and y must be related). Formally, we define the (set-indexed) map $\hat{\mathbb{D}} : \mathsf{Rel}(X, Y) \to \mathsf{Rel}(DX, DY)$ as follows:

$$\mu\ \hat{\mathbb{D}}\mathcal{R}\ \nu \overset{\triangle}{\iff} [\exists \omega \in \Omega(\mu, \nu).\ \forall x, y.\ \omega(x, y) > 0 \implies x\ \mathcal{R}\ y].$$

We will prove that $\hat{\mathbb{D}}$ is a relator for $\mathbb{D}$ in a more abstract setting in Subsection 4.3.1. We conclude our analysis of couplings stating the already mentioned *Strassen's Theorem*[5] (Strassen, 1965) which shows that $\hat{\mathbb{D}}$ can be defined both universally (i.e. using an universal quantification) and existentially (i.e. using an existential quantification).

Theorem 2 (Strassen's Theorem)**.** *Let $\mu \in DX$, $\nu \in DY$ be distributions, and $\mathcal{R} : X \nrightarrow Y$ be a relation. Then:*

$$\mu \,\hat{\mathbb{D}}\mathcal{R}\, \nu \iff \forall \mathfrak{X} \subseteq X.\, \mu(\mathfrak{X}) \leq \nu(\mathcal{R}[\mathfrak{X}]).$$

Even if we will give an abstract proof that $\hat{\mathbb{D}}$ is a relator for $\mathbb{D}$, using Theorem 2 it is possible to prove such a fact directly. Additionally, as a corollary of Theorem 2, we see that $\hat{\mathbb{D}}$ describes the defining clause of Larsen-Skou bisimulation for Markov chains (based on full distributions) (Larsen & Skou, 1989). Recall that given a Markov chain $c : X \to DX$, we say that an *equivalence* relation $\mathcal{R}$ on X is a bisimulation if $x\,\mathcal{R}\,y$ implies $c(x)(\mathfrak{B}) = c(y)(\mathfrak{B})$, for any $\mathcal{R}$-equivalence class $\mathfrak{B}$. Then, it is easy to see that we have:

$$\forall \mathfrak{X} \subseteq X.\, \mu(\mathfrak{X}) \leq \nu(\mathcal{R}[\mathfrak{X}]) \iff \forall \mathfrak{B} \in X/\,\mathcal{R}\,.\, \mu(\mathfrak{B}) = \nu(\mathfrak{B}).$$

Finally, we observe that we can sum $\hat{\mathbb{D}}$ with $\hat{\mathbb{M}}$ obtaining a relator $\mathbb{D}\hat{\mathbb{M}} \triangleq \hat{\mathbb{D}}\hat{\mathbb{M}}$ for the monad $\mathbb{D}\mathbb{M}$ (and thus for the subdistribution monad). ⊠

Example 31 (Output monad)**.** For the (partial) output monad $\mathbb{O}^\infty$ we define the map $\hat{\mathbb{O}^\infty} : \mathsf{Rel}(X, Y) \to \mathsf{Rel}(O^\infty X, O^\infty Y)$ as follows:

$$\langle u, \mathfrak{x}\rangle \,\hat{\mathbb{O}^\infty}\mathcal{R}\, \langle w, \mathfrak{y}\rangle \overset{\triangle}{\iff} (\mathfrak{x} = \bot \wedge u \subseteq w) \vee (\mathfrak{x} = \mathit{just}\ x \wedge \mathfrak{y} = \mathit{just}\ y \wedge u = w \wedge x\,\mathcal{R}\,y).$$

Tedious calculations show that $\hat{\mathbb{O}^\infty}$ is a relator for $\mathbb{O}^\infty$. ⊠

Example 32 (Cost monad)**.** For the total cost monad $\mathbb{C}_0$ define the map $\hat{\mathbb{C}_0} : \mathsf{Rel}(X, Y) \to \mathsf{Rel}(C_0X, C_0Y)$ as

$$\hat{\mathbb{C}_0}\mathcal{R} \triangleq (\geq \times \mathcal{R}),$$

where we recall that for $\mathcal{R} : X \nrightarrow Y$, $\mathcal{S} : X' \nrightarrow Y'$, we define $\mathcal{R} \times \mathcal{S} : X \times X' \nrightarrow Y \times Y'$ as

$$(\mathcal{R} \times \mathcal{S})((x, x'), (y, y')) \triangleq \mathcal{R}(x, y) \wedge \mathcal{S}(x', y'),$$

and $\geq$ denotes the opposite of the natural ordering on $\mathbb{N}$. It is straightforward to see that $\hat{\mathbb{C}_0}$ is indeed a relator for $\mathbb{C}_0$. The use of the opposite of the natural order in the definition of $\hat{\mathbb{C}_0}$ captures the idea that we use $\mathbb{C}_0$ to measure complexity. If we think to (n, x) as the result of a computation (the element x) together with the cost of the computation (the number n), then $(n, x)\,\hat{\mathbb{C}_0}\mathcal{R}\,(m, y)$ means that the results x and y are $\mathcal{R}$-related, and that $n \geq m$, meaning that right hand computation is more efficient than the left hand one. By taking the sum of $\hat{\mathbb{C}_0}$ and $\hat{\mathbb{M}}$ we obtain a relator for $\mathbb{C}$ that describes Sands' simulation clause for program improvement (Sands, 1998). ⊠

Example 33 (Global states)**.** For the global state monad $\mathbb{G}$ we define the map $\hat{\mathbb{G}} : \mathsf{Rel}(X, Y) \to \mathsf{Rel}(GX, GY)$ as follows:

$$\alpha \,\hat{\mathbb{G}}\mathcal{R}\, \beta \overset{\triangle}{\iff} \forall \sigma \in S.\, \alpha(\sigma)\ (\mathsf{I}_S \times \mathcal{R})\ \beta(\sigma).$$

It is straightforward to see that $\hat{\mathbb{G}}$ is a relator for $\mathbb{G}$. Moreover, we can generalise Proposition 4 and Proposition 6 to take into account relators.

[5] The original formulation of Strassen's Theorem is actually more general than Theorem 2 (notably, it does not require distributions to be discrete). This is reflected by the possibility of giving a proof of Theorem 2 based on the Max-flow Min-cut Theorem (Schrijver, 1986).

Proposition 12. *Given a monad* $\mathbb{T} = \langle T, \mathrm{T}, -^{\mathrm{T}} \rangle$ *and relator* $\hat{\mathbb{T}}$ *for it, define* $\widehat{\mathbb{T} \otimes \mathbb{G}}$ *as:*

$$\alpha \; (\widehat{\mathbb{T} \otimes \mathbb{G}})\mathcal{R} \; \beta \overset{\triangle}{\iff} \forall \sigma.\; \alpha(\sigma) \; \hat{\mathbb{T}}(\mathsf{I}_S \times \mathcal{R}) \; \beta(\sigma).$$

Then $(\widehat{\mathbb{T} \otimes \mathbb{G}})$ *is a relator for* $\mathbb{T} \otimes \mathbb{G}$.

Straightforward calculations show that $(\widehat{\mathbb{T} \otimes \mathbb{G}})$ is a relator for $T \otimes G$. The most interesting part of the proof of Proposition 12 is proving that $(\widehat{\mathbb{T} \otimes \mathbb{G}})$ satisfies condition (rel bind). We assume $\mathcal{R} \subseteq g^\circ \cdot (\mathbb{T} \hat{\otimes} \mathbb{G})\mathcal{S} \cdot f$, for relations and functions of appropriate source and target, and prove that $\alpha (\widehat{\mathbb{T} \otimes \mathbb{G}})\mathcal{R}\beta$ implies $f^{\mathbb{T}\otimes\mathbb{G}}(\alpha) \; (\widehat{\mathbb{T} \otimes \mathbb{G}})\mathcal{S} \; g^{\mathbb{T}\otimes\mathbb{G}}(\beta)$. First of all, we notice that since $\mathbb{T}$ satisfies condition (rel bind), we have:

$$\begin{aligned} \mathcal{R} \subseteq g^\circ \cdot (\widehat{\mathbb{T} \otimes \mathbb{G}})\mathcal{S} \cdot f \;&\implies \mathsf{I}_S \times \mathcal{R} \subseteq (\mathsf{uncurry}\; g)^\circ \cdot \hat{\mathbb{T}}(\mathsf{I}_S \times \mathcal{S}) \cdot \mathsf{uncurry}\; f \\ &\implies \hat{\mathbb{T}}(\mathsf{I}_S \times \mathcal{R}) \subseteq ((\mathsf{uncurry}\; g)^{\mathrm{T}})^\circ \cdot \hat{\mathbb{T}}(\mathsf{I}_S \times \mathcal{S}) \cdot (\mathsf{uncurry}\; f)^{\mathrm{T}}. \end{aligned}$$

In particular, for any store σ, we have

$$\begin{aligned} \alpha(\sigma) \; \hat{\mathbb{T}}(\mathsf{I}_S \times \mathcal{R}) \; \beta(\sigma) \;&\implies (\mathsf{uncurry}\; f)^{\mathrm{T}}\alpha(\sigma) \; \hat{\mathbb{T}}(\mathsf{I}_S \times \mathcal{S}) \; (\mathsf{uncurry}\; g)^{\mathrm{T}}\beta(\sigma) \\ &\iff f^{\mathbb{T}\otimes\mathbb{G}}(\alpha) \; (\widehat{\mathbb{T} \otimes \mathbb{G}})\mathcal{S} \; g^{\mathbb{T}\otimes\mathbb{G}}(\beta). \end{aligned}$$

As a consequence, it is sufficient to prove $\alpha(\sigma) \; \hat{\mathbb{T}}(\mathsf{I}_S \times \mathcal{R}) \; \beta(\sigma)$. But the latter is a direct consequence of $\alpha \; (\widehat{\mathbb{T} \otimes \mathbb{G}})\mathcal{R} \; \beta$, and thus we are done.

In particular, we can tensor $\hat{\mathbb{G}}$ with $\hat{\mathbb{M}}$, $\hat{\mathbb{FM}}$, and $\hat{\mathbb{DM}}$ obtaining relators for partial, nondeterministic, and probabilistic imperative computations. Additionally, tensoring $\hat{\mathbb{G}}$ with $\hat{\mathbb{C}}$ we obtain a relator for imperative computations with cost. ⊠

Before continuing our analysis of relators, we make a small digression on the so-called *Barr's construction* (the content of such a digression is not needed to follow the rest of this work).

4.3.1 Digression: Barr's Construction

Most of the conversive relators in previous examples are instances of a general construction known as *Barr construction* (see e.g. (Hofmann et al., 2014; Kurz & Velebil, 2016)). Such a construction builds for any functor T (on Set), a candidate relator $\overline{T}$ for T, called the *Barr extension* of T. If T preserves weak pullbacks, then $\overline{T}$ indeed defines a *functorial* relator for T, which is also canonical (see Theorem 3).

At the heart of the Barr construction relies the double nature of relations as morphisms in Rel and objects in Set. To avoid confusion, in this section we will adopt the following notational convention: for a relation $\mathcal{R} : X \nrightarrow Y$ we denote by $G_{\mathcal{R}} \subseteq X \times Y$ the corresponding object in Set. Any relation $G_{\mathcal{R}} \subseteq X \times Y$ can be represented as a span (see Appendix B) ($X \xleftarrow{\pi_1} G_{\mathcal{R}} \xrightarrow{\pi_2} Y$), where $\pi_1 : G_{\mathcal{R}} \to X$ and $\pi_2 : G_{\mathcal{R}} \to Y$ are projection maps. In particular, we have the identity $\mathcal{R} = \pi_2 \cdot \pi_1^\circ$ in Rel. We can now define the Barr extension of T.

Definition 22. *The* Barr extension *of a functor* T *(on* Set*) is the set-indexed family of maps* $\overline{T} : \mathsf{Rel}(X, Y) \to \mathsf{Rel}(TX, TY)$ *defined for* $\mathcal{R} = \pi_2 \cdot \pi_1^\circ : X \nrightarrow Y$ *by:*

$$\overline{T}\mathcal{R} \triangleq T\pi_2 \cdot (T\pi_1)^\circ,$$

where $\mathcal{R} = \pi_2 \cdot \pi_1^\circ$. *Pointwise,* $\overline{T}$ *can be characterised as follows:*

$$\mathfrak{x} \; \overline{T} \; \mathcal{R} \; \mathfrak{y} \iff \exists \mathfrak{w} \in TG_{\mathcal{R}}.\; T\pi_1(\mathfrak{w}) = \mathfrak{x} \wedge T\pi_2(\mathfrak{w}) = \mathfrak{y},$$

where $\mathfrak{x} \in TX$ *and* $\mathfrak{y} \in TY$

In general, $\overline{T}$ is not a relator for T, as it might fail to satisfy property (rel 2). However, $\overline{T}$ has been proved to be a a functorial relator exactly when T preserves weak pullbacks (see Appendix B for insights on the role of pullbacks in Barr's construction).

Theorem 3. *Let T be a functor. The following hold:*

1. *T preserves weak pullbacks or, equivalently, transforms pullbacks into weak pullbacks, if and only if $\overline{T}$ is a* functorial *relator for T.*
2. *$\overline{T}$ is canonical, in the sense that if Γ is a functorial relator for T, then $\Gamma = \overline{T}$ (and thus T preserves weak pullbacks).*
3. *If additionally T is the carrier of a monad $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ and T preserves weak pullbacks, then $\overline{T}$ is a relator for $\mathbb{T}$.*

The proof of Theorem 3 is not difficult, but is rather long. The reader can consult (Hofmann et al., 2014; Kurz & Velebil, 2016) for a detailed exposition.

Example 34 (The relator $\hat{\mathbb{D}}$)**.** We immediately see that $\hat{\mathbb{D}} = \overline{D}$. Since D preserves weak pullbacks, Theorem 3 implies that $\hat{\mathbb{D}}$ is indeed a relator for $\mathbb{D}$, and thus $\mathbb{D}\hat{\mathbb{M}}$ is a relator for $\mathbb{D}\mathbb{M}$.

In Example 30 we stated that it is possible to give a direct proof that $\hat{\mathbb{D}}$ is a relator $\mathbb{D}$ relying on Theorem 2 — the discrete version of Strassen's Theorem — which in turn relies on Max-flow Min-cut Theorem. Theorem 3 gives an indirect proof of such a result, and the reader may wondering whether the complexity of Theorem 2 is still present in this indirect proof. This is indeed the case. In fact, in (de Vink & Rutten, 1997) it is shown that the distribution functor preserves weak pullbacks (hence inferring 'good properties' of probabilistic bisimilarity) by presenting an argument relying on the the Max-flow Min-cut Theorem. ⊠

In light of the numerous examples of relators obtained as instances of Barr's construction, the Barr extension of a functor can be used to define a somehow canonical definition of a notion of bisimulation. However, begin conversive, the Barr extension of a functor does not provide an adequate tool to define canonical notions of similarity (and, as we will see, such notions of similarity are of paramount importance in this work). It is natural to ask whether there exists a canonical way to extract a relator defining notions of simulation from a functor. Unfortunately, the answer appears to be negative.

First of all, we observe that given a functor T carrying an order $\leq$ (that is, associating to any set X an order $\leq_X$ on TX) we can define the candidate relator

$$\mathcal{R} \mapsto \leq_Y \cdot \overline{T} \cdot \leq_X$$

for T. For instance, in this way we can recover the relator $\mathbb{F}\hat{\mathbb{M}}$ for FM using the order defined in Example 18. This observation led to the proposal in (Hughes & Jacobs, 2004) of considering notions of simulation defined in terms of *stable orders* on functors. Intuitively, a stable order on a functor T (on Set) is a functor F from Set to Preord (the category of preordered sets and monotone functions) mapping a set X to a preordered set $(TX, \leq_X)$ and sending weak pullbacks to preordered weak pullbacks (see (P. Levy, 2011)).

(Hughes & Jacobs, 2004) showed that given a stable order F on a functor T, the map $\mathcal{R} \mapsto \leq_Y \cdot \overline{T} \cdot \leq_X$ indeed defines a relator for T. However, in (P. Levy, 2011) it is shown that there is a bijection[6]

[6] Intuitively, given a relator Γ satisfying (rel funct 2), one defines the stable order F_Γ on T as follows:

$$F_\Gamma X \triangleq (TX, \Gamma(1_X))$$
$$F_\Gamma f \triangleq Tf.$$

between relators for T satisfying (rel funct 2) and stable orders on T, meaning that finding stable orders is essentially equivalent to finding specific relators.

This essentially concludes our exposition of the general theory of relators. However, there is still one point missing, namely the connection between relators and the domain structure we imposed on monads.

4.4 Σ-continuous relators

In order to accommodate infinitary computational behaviours, we required monads to come with a suitable domain structure. It is then natural to require relators to properly interact with such a structure. We have already seen an example of such an interaction in Chapter 2, where we proved precongruence of probabilistic applicative similarity relying on a suitable induction principle for $\hat{D}$. Relators allowing such forms of inductive reasoning are called Σ-*continuous* relators.

Definition 23. *Let $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ be a Σ-continuous monad and Γ be relator for $\mathbb{T}$. We say that Γ is* Σ-continuous *if it satisfies conditions* (ind 1), (ind 2), *and* (ind 3) — *called the* inductive conditions — *for any ω-chain $(\mathfrak{x}_n)_n$ in TX, element $\mathfrak{y} \in TY$, elements $\mathfrak{x}, \mathfrak{x}' \in TX$, and relation $\mathcal{R} : X \nrightarrow Y$.*

$$\bot \; \Gamma\mathcal{R} \; \mathfrak{y} \tag{ind 1}$$

$$\mathfrak{x} \sqsubseteq \mathfrak{x}', \mathfrak{x}' \; \Gamma\mathcal{R} \; \mathfrak{y} \implies \mathfrak{x} \; \Gamma\mathcal{R} \; \mathfrak{y} \tag{ind 2}$$

$$\forall n.\, \mathfrak{x}_n \; \Gamma\mathcal{R} \; \mathfrak{y} \implies \bigsqcup_{n \geq 0} \mathfrak{x}_n \; \Gamma\mathcal{R} \; \mathfrak{y}. \tag{ind 3}$$

Condition (ind 2) is actually not needed to prove congruence and precongruence properties of applicative bisimilarity and similarity, respectively. However, we will need it to prove the same properties for normal form bisimilarity and similarity, and thus we take it as a defining condition of the notion of a Σ-continuous relator. As a convention, when we want to stress that a property holds in virtue of one (or more) of the properties (ind 1)-(ind 3), we will simply say that the property follows since the relator is inductive.

Example 35. The relators $\hat{\mathbb{M}}$, $\hat{\mathbb{ME}}$, $\hat{\mathbb{FM}}$, $\hat{\mathbb{DM}}$, $\hat{\mathbb{O}^\infty}$, $\hat{\mathbb{C}}$, $\widehat{\mathbb{M} \otimes \mathbb{G}}$, $\widehat{\mathbb{FM} \otimes \mathbb{G}}$, $\widehat{\mathbb{DM} \otimes \mathbb{G}}$, $\widehat{\mathbb{C} \otimes \mathbb{G}}$ are Σ-continuous. ⊠

Remark 6 (Σ-algebraic relators)**.** The reader might have noticed that up to this point we have not required specific interactions between relators and algebraic operations on monads. That might appeared wired, since having required relators to properly interact both with the monad and the domain structure of a Σ-continuous monad $\mathbb{T}$, it seems natural to require relators to properly with its Σ-algebra structure too.

Formally, we can require the desired interaction by demanding a relators $\mathcal{R}$ to satisfy condition (Σ comp) below, for all operation symbol $(\mathbf{op} : P \rightsquigarrow I) \in \Sigma$, maps $\kappa, \nu : I \to TX$, parameter $p \in P$, and relation $\mathcal{R}$.

$$\forall i \in I.\, \kappa(i) \; \Gamma\mathcal{R} \; \nu(i) \implies [\![\mathbf{op}]\!](p, \kappa) \; \Gamma\mathcal{R} \; [\![\mathbf{op}]\!](p, \nu). \tag{Σ comp}$$

It is a remarkable result that if $\mathbb{T}$ is Σ-algebraic, then any relator Γ for $\mathbb{T}$ satisfies (Σ comp).

Proposition 13. *Let $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ be a Σ-algebraic monad, and let Γ be a relator for $\mathbb{T}$. Then Γ satisfies condition* (Σ comp).

Proof. For simplicity we give the proof for the case of operation symbols without parameters. This is not a real restriction, as we can assume without loss of generality all operations to be such. In fact,

we simply replace an operation symbol $\mathbf{op} : P \rightsquigarrow I$ with a P-indexed family of operation symbols $\mathbf{op}_p : I$. The proof proceeds using the correspondence between algebraic operations and *generic effects* (G. D. Plotkin & Power, 2003). Given an arity set I, an I-ary generic effect is an element of TI. Moreover, any I-ary generic effect g gives an I-ary algebraic operation $\mathbf{op}_g$ defined by $[\![\mathbf{op}_g]\!](\kappa) = \kappa^\dagger(g)$. Vice versa, any I-ary algebraic operation defines a generic effect $g_{\mathbf{op}} \in TI$ as $[\![\mathbf{op}]\!]_I(\eta_I)$. Straightforward calculations show that these transformations give a correspondence between algebraic operations and generic effects. Let us now move to (Σ comp). We wish to prove

$$\forall i \in I.\ \kappa(i)\ \Gamma\mathcal{R}\ \nu(i) \implies [\![\mathbf{op}]\!](\kappa)\ \Gamma\mathcal{R}\ [\![\mathbf{op}]\!](\nu).$$

Let g be the generic effect associated with $\mathbf{op}$, and assume $\kappa(i)\ \Gamma\mathcal{R}\ \nu(i)$ to hold for all $i \in I$. The latter hypothesis can be rewritten as $\mathsf{I}_I \subseteq \nu^\circ \cdot \Gamma\mathcal{R} \cdot \kappa$, from which we infer $\Gamma\mathsf{I}_I \subseteq (\nu^\dagger)^\circ \cdot \Gamma\mathcal{R} \cdot \kappa^\dagger$, by (rel bind). Therefore, we can conclude $\kappa^\dagger(g)\ \Gamma\mathcal{R}\ \nu^\dagger(g)$ (i.e. our main thesis), provided we prove $g\ \Gamma\mathsf{I}_I\ g$. The latter indeed holds, since by condition (rel 1) we have $\mathsf{I}_{TI} \subseteq \Gamma\mathsf{I}_I$. □

Proposition 13 concludes our exposition of the general theory of relators. Having at our disposal this powerful machinery, we are now going to apply it to define and study effectful equivalences and refinements for Λ_Σ.

Chapter 5

Effectful Applicative Similarity and Bisimilarity

> Equality gives rise to challenging questions which are not altogether easy to answer . . .
>
> Gottlob Frege, On Sense and Reference

In this chapter we define three notions of equivalence and three notions of refinement for the calculus Λ_Σ of Chapter 3, namely:

- Effectful contextual approximation $\preceq^{\text{ctx}}$ and equivalence $\simeq^{\text{ctx}}$ (Section 5.2).
- Effectful applicative similarity $\preceq^{\text{A}}$ and bisimilarity $\simeq^{\text{A}}$ (Section 5.3).
- Effectful CIU approximation $\preceq^{\text{ciu}}$ and equivalence $\simeq^{\text{ciu}}$ (Section 5.5).

All these notions are parametrised by a relator Γ, which specifies the observable properties of computations.

Our main results state that if Γ is Σ-continuous, then effectful applicative similarity is *sound* for effectful contextual approximation (Theorem 4), and effectful applicative bisimilarity is sound for effectful contextual equivalence (Theorem 5). None of them, however, is fully abstract. That is because effectful applicative similarity (resp. bisimilarity) subsumes lower applicative similarity (resp. bisimilarity) (S. Lassen, 1998b; C. L. Ong, 1993) which is known to be strictly finer than lower contextual approximation (resp. equivalence). We will say more about that later. Our soundness results are proved with a generalisation of Howe's method (Howe, 1996), which takes advantage of the structural properties of Σ-continuous relators.

A variation of Howe's method also allows to prove that effectful CIU approximation (resp. equivalence) is *fully abstract* with respect to effectful contextual approximation (resp. equivalence) (Theorem 6). These results are summarised in Table 5.1.

This chapter is structured as follows. In Section 5.1 we introduce λ-term relations and define a simple relational calculus (as we did in Chapter 2) that we will use throughout the rest of this dissertation. We will instantiate such a calculus to define the notions of effectful contextual approximation and equivalence (Section 5.2) as well as the notions of effectful applicative similarity and bisimilarity (Section 5.3).

$\preceq^{A}$	$\subsetneq$	$\preceq^{ciu}$	$=$	$\preceq^{ctx}$
$\simeq^{A}$	$\subsetneq$	$\simeq^{ciu}$	$=$	$\simeq^{ctx}$

Table 5.1: Equivalences and refinements for Λ_Σ, part 1.

Section 5.4 is entirely devoted to Howe's method and the proof of the precongruence theorem for effectful applicative similarity (Theorem 4). Congruence of effectful applicative bisimilarity (Theorem 5) is proved in Subsection 5.4.1. Finally, in Section 5.5 we define the notions of effectful CIU approximation and equivalence, and prove them to be fully abstract (Theorem 6) with respect to effectful contextual approximation and equivalence, respectively.

5.1 Relational Reasoning

In this section we extend the relational calculus developed in Chapter 2 for Λ_p to Λ_Σ. Most of the definitions and results presented in this section are slightly variations of the corresponding ones in Chapter 2. Therefore, the reader already familiar with such notions can safely jump to the next section.

Definition 24. *1. A* closed *λ-term relation is a pair $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ of relations $\mathcal{R}_\Lambda$ and $\mathcal{R}_\mathcal{V}$ on $\Lambda_\circ$ and $\mathcal{V}_\circ$, respectively. We refer to $\mathcal{R}_\Lambda$ as the computation component of $\mathcal{R}$, and to $\mathcal{R}_\mathcal{V}$ as the value component of $\mathcal{R}$.*

2. An open *λ-term relation $\mathcal{R}$ associates to each finite set of variables Γ a relation $\Gamma \vdash^\Lambda - \mathcal{R} - $ on Λ_Γ, and a relation $\Gamma \vdash^\mathcal{V} - \mathcal{R} - $ on $\mathcal{V}_\Gamma$. We require open λ-term relations to be closed under weakening:*

$$\frac{\Gamma \vdash^\Lambda e \,\mathcal{R}\, f}{\Gamma, x \vdash^\Lambda e \,\mathcal{R}\, f} \qquad \frac{\Gamma \vdash^\mathcal{V} e \,\mathcal{R}\, w}{\Gamma, x \vdash^\mathcal{V} v \,\mathcal{R}\, w}$$

Example 36. Both the discrete/identity relation I and the indiscrete relation $\mathsf{0}$ defined by the rules below are open λ-term relations. The empty relation is an open λ-term relation too.

$$\frac{e \in \Lambda_\Gamma}{\Gamma \vdash^\Lambda e \,\mathsf{I}\, e} \qquad \frac{v \in \mathcal{V}_\Gamma}{\Gamma \vdash^\mathcal{V} v \,\mathsf{I}\, v} \qquad \frac{e, f \in \Lambda_\Gamma}{\Gamma \vdash^\Lambda e \,\mathsf{0}\, f} \qquad \frac{v, w \in \mathcal{V}_\Gamma}{\Gamma \vdash^\mathcal{V} v \,\mathsf{0}\, w}$$

Since $e \in \Lambda_\Gamma$ (resp. $v \in \mathcal{V}_\Gamma$) implies $e \in \Lambda_{\Gamma,\Delta}$ (resp. $v \in \mathcal{V}_{\Gamma,\Delta}$), for any finite set of variables Δ, both $\mathsf{0}$ and $\mathsf{1}$ are closed under weakening. ⊠

We denote by Rel and Rel^c the collections of open and closed λ-term relations, respectively. Formally, we define Rel as $\prod_\Gamma \mathsf{Rel}(\Lambda_\Gamma, \Lambda_\Gamma) \times \mathsf{Rel}(\mathcal{V}_\Gamma, \mathcal{V}_\Gamma)$. Rel inherits a rich structure from $\mathsf{Rel}(\Lambda_\Gamma, \Lambda_\Gamma)$ and $\mathsf{Rel}(\mathcal{V}_\Gamma, \mathcal{V}_\Gamma)$, as witnessed by the following result.

Proposition 14. Rel *is a* quantale *(see Definition 61) with an involution. In particular, it is a complete lattice.*

Proof. We notice that Rel inherits a complete lattice structure from $\mathsf{Rel}(\Lambda_\Gamma, \Lambda_\Gamma)$ and $\mathsf{Rel}(\mathcal{V}_\Gamma, \mathcal{V}_\Gamma)$ pointwise. The bottom element is the empty relation, whereas the top element is the indiscrete relation $\mathsf{0}$. Monoid multiplication is defined as λ-term relation composition: given λ-term relations $\mathcal{R}$ and $\mathcal{S}$, we define the composition of $\mathcal{S}$ with $\mathcal{R}$, denoted by $\mathcal{S} \cdot \mathcal{R}$, as:

$$\frac{\Gamma \vdash^\Lambda e \,\mathcal{R}\, g \quad \Gamma \vdash^\Lambda g \,\mathcal{S}\, f}{\Gamma \vdash^\Lambda e \,(\mathcal{S} \cdot \mathcal{R})\, f} \qquad \frac{\Gamma \vdash^\mathcal{V} v \,\mathcal{R}\, u \quad \Gamma \vdash^\mathcal{V} u \,\mathcal{S}\, w}{\Gamma \vdash^\mathcal{V} v \,(\mathcal{S} \cdot \mathcal{R})\, w}$$

Since both $\mathcal{R}$ and $\mathcal{S}$ are closed under weakening, then so is $\mathcal{S} \cdot \mathcal{R}$. Moreover, we see that the unit of composition is given by the discrete λ-term relation I, and that Rel is indeed a monoid. It is straightforward to see that composition is monotone in both arguments. This makes Rel an ordered monoid. Finally, we observe that Rel also has an involution given by the mapping a λ-term relation $\mathcal{R}$ to its converse $\mathcal{R}^\circ$. The latter is defined as follow:

$$\frac{\Gamma \vdash^\Lambda f \,\mathcal{R}\, e}{\Gamma \vdash^\Lambda e \,\mathcal{R}^\circ\, f} \qquad \frac{\Gamma \vdash^\mathsf{v} w \,\mathcal{R}\, v}{\Gamma \vdash^\mathsf{v} v \,\mathcal{R}^\circ\, w}$$

□

By Proposition 14 we can define λ-term relations both inductively and coinductively. Concerning notation, we write $\mathcal{R} \subseteq \mathcal{S}$ if:

$$\forall \Gamma, e, f.\ \Gamma \vdash^\Lambda e \,\mathcal{R}\, f \implies \Gamma \vdash^\Lambda e \,\mathcal{S}\, f$$
$$\forall \Gamma, v, w.\ \Gamma \vdash^\mathsf{v} v \,\mathcal{R}\, w \implies \Gamma \vdash^\mathsf{v} v \,\mathcal{S}\, w.$$

Moreover, relying on the (complete) lattice structure of Rel we can define the notions of a *preorder* and *equivalence* λ-term relation. In fact, we say that λ-term relation $\mathcal{R}$ is reflexive if $\mathsf{I} \subseteq \mathcal{R}$, transitive if $\mathcal{R} \cdot \mathcal{R} \subseteq \mathcal{R}$, and symmetric if $\mathcal{R}^\circ \subseteq \mathcal{R}$.

There are maps $-^c : \mathsf{Rel} \to \mathsf{Rel}^c$ and $-^o : \mathsf{Rel}^c \to \mathsf{Rel}$ restring open λ-term relations to closed ones, and extending closed λ-term relations to open ones. Formally, we define such maps as follows.

Definition 25. *Given a λ-term relation $\mathcal{R} \in \mathsf{Rel}$, define the* closed restriction[1] *$\mathcal{R}^c = (\mathcal{R}_\Lambda, \mathcal{R}_\mathsf{V})$ of $\mathcal{R}$ by*

$$\frac{\vdash^\Lambda e \,\mathcal{R}\, f}{e \;\mathcal{R}_\Lambda\; f} \qquad \frac{\vdash^\mathsf{v} v \,\mathcal{R}\, w}{v \;\mathcal{R}_\mathsf{V}\; w}$$

Dually, given a closed λ-term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathsf{V})$ we define $\mathcal{R}^o$ as its open extension. *The latter is defined as follows, where $\Gamma \triangleq \vec{x} \triangleq x_1, \ldots, x_n$ and $\vec{u} \triangleq u_1, \ldots, u_n$:*

$$\frac{\forall \vec{u} \in \mathcal{V}_\circ.\ e[\vec{x} := \vec{u}] \,\mathcal{R}_\Lambda\, f[\vec{x} := \vec{u}]}{\Gamma \vdash^\Lambda e \,\mathcal{R}^o\, f} \qquad \frac{\forall \vec{u} \in \mathcal{V}_\circ.\ v[\vec{u}/\vec{x}] \,\mathcal{R}_\mathsf{V}\, u[\vec{u}/\vec{x}]}{\Gamma \vdash^\mathsf{v} v \,\mathcal{R}^o\, w}$$

Taking advantage of Definition 25, for a *closed* λ-term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathsf{V})$ we will often write $\vdash^\Lambda e \,\mathcal{R}\, f$ in place of $e \,\mathcal{R}_\Lambda\, f$, and $\Gamma \vdash^\Lambda e \,\mathcal{R}\, f$ in place of $\Gamma \vdash^\Lambda e \,\mathcal{R}^o\, f$ (and similarity for values). Dually, for an open λ-term relation $\mathcal{S}$ we will use the notations $\vdash^\Lambda e \,\mathcal{S}\, f$ and $e \,\mathcal{S}_\Lambda\, f$ interchangeably (and similarity for values). Next we introduce the notion of a *substitutive* λ-term relation.

Definition 26. 1. *A λ-term relation $\mathcal{R}$ is* value-substitutive *if the following hold, where u ranges over closed values:*

$$\frac{\Gamma, x \vdash^\Lambda e \,\mathcal{R}\, f}{\Gamma \vdash^\Lambda e[x := u] \,\mathcal{R}\, f[x := u]} \qquad \frac{\Gamma, x \vdash^\mathsf{v} v \,\mathcal{R}\, w}{\Gamma \vdash^\mathsf{v} v[u/x] \,\mathcal{R}\, w[u/x]}$$

2. *A λ-term relation $\mathcal{R}$ is* substitutive *if the following hold:*

$$\frac{\Gamma, x \vdash^\Lambda e \,\mathcal{R}\, f \quad \vdash^\mathsf{v} v \,\mathcal{R}\, w}{\Gamma \vdash^\Lambda e[x := v] \,\mathcal{R}\, f[x := w]} \qquad \frac{\Gamma, x \vdash^\mathsf{v} v \,\mathcal{R}\, w \quad \vdash^\mathsf{v} u \,\mathcal{R}\, u'}{\Gamma \vdash^\mathsf{v} v[u/x] \,\mathcal{R}\, w[u'/x]}$$

A closed λ-relation is (value) substitutive if its open extension is. Moreover, we notice that the open extension of a closed λ-term relation is trivially value-substitutive.

[1] Notice that the notation $\mathcal{R}_\Lambda$ (resp. $\mathcal{R}_\mathsf{V}$) is not defined for open λ-term relations, so that we can safely use that to denote the computation (resp. value) component of $\mathcal{R}^c$.

In order to define the notion of a precongruence λ-term relation, we introduce the notion of *compatibility*. Roughly speaking, a λ-term relation is compatible if it is preserved by all Λ_Σ syntactic constructors. Formally, we follow (Gordon, 1994; S. Lassen, 1998b) and define compatibility via the notion of a *compatible refinement* of a λ-term relation.

Definition 27. *The* compatible refinement $\widehat{\mathcal{R}}$ *of an open λ-term relation $\mathcal{R}$ is defined by the rules in Figure 5.1. We say $\mathcal{R}$ is compatible if $\widehat{\mathcal{R}} \subseteq \mathcal{R}$, and that a closed λ-term relation is compatible if its open extension is.*

$$\frac{}{\Gamma, x \vdash^{\mathsf{v}} x \;\widehat{\mathcal{R}}\; x}\ (\text{comp-var})$$

$$\frac{\Gamma, x \vdash^{\Lambda} e \;\mathcal{R}\; f}{\Gamma \vdash^{\mathsf{v}} \lambda x.e \;\widehat{\mathcal{R}}\; \lambda x.f}\ (\text{comp-abs}) \qquad \frac{\Gamma \vdash^{\mathsf{v}} v \;\mathcal{R}\; v' \quad \Gamma \vdash^{\mathsf{v}} w \;\mathcal{R}\; w'}{\Gamma \vdash^{\Lambda} vw \;\widehat{\mathcal{R}}\; v'w'}\ (\text{comp-app})$$

$$\frac{\Gamma \vdash^{\mathsf{v}} v \;\mathcal{R}\; w}{\Gamma \vdash^{\Lambda} \mathbf{return}\ v \;\widehat{\mathcal{R}}\; \mathbf{return}\ w}\ (\text{comp-ret}) \qquad \frac{\Gamma \vdash^{\Lambda} e \;\mathcal{R}\; e' \quad \Gamma, x \vdash^{\Lambda} f \;\mathcal{R}\; f'}{\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f \;\widehat{\mathcal{R}}\; \mathbf{let}\ x = e'\ \mathbf{in}\ f'}\ (\text{comp-let})$$

$$\frac{\Gamma, x \vdash^{\Lambda} e \;\mathcal{R}\; f}{\Gamma \vdash^{\Lambda} \mathbf{op}(p, x.e) \;\widehat{\mathcal{R}}\; \mathbf{op}(p, x.f)}\ (\text{comp-op})$$

Figure 5.1: Compatible refinement Λ_Σ.

Notice that $\widehat{\mathcal{R}}$ is indeed a λ-term relation (notably, $\widehat{\mathcal{R}}$ is closed under weakening). Definition 27 induces a map $\mathcal{R} \mapsto \widehat{\mathcal{R}}$ on the collection of open λ-term relations which is monotone and satisfies the following identities (see Subsection 5.4.1):

$$\widehat{\mathcal{S} \cdot \mathcal{R}} = \widehat{\mathcal{S}} \cdot \widehat{\mathcal{R}}$$
$$\widehat{\mathcal{R}^{\circ}} = (\widehat{\mathcal{R}})^{\circ}.$$

In particular, a λ-term relation is compatible if and only if it is a pre-fixed point of $\mathcal{R} \mapsto \widehat{\mathcal{R}}$. It is not hard to prove that the discrete open λ-term relation I of Example 36 is a pre-fixed point of $\mathcal{R} \mapsto \widehat{\mathcal{R}}$. Moreover, it is the least such. As a consequence, any compatible relation is reflexive. Dually, the indiscrete open λ-term relation 0 is the greatest fixed point of $\mathcal{R} \mapsto \widehat{\mathcal{R}}$.

Remark 7. If Σ consists of finitary operations only (as in most of the examples studied), then Definition 27 can be slightly simplified by replacing rule (comp-op) with the following one (for any n-ary operation symbol $\mathbf{op}$ in Σ):

$$\frac{\Gamma \vdash^{\Lambda} e_1 \;\mathcal{R}\; f_1 \quad \cdots \quad \Gamma \vdash^{\Lambda} e_n \;\mathcal{R}\; f_n}{\Gamma \vdash^{\Lambda} \mathbf{op}(e_1, \ldots, e_n) \;\widehat{\mathcal{R}}\; \mathbf{op}(f_1, \ldots, f_n)}$$

Easy calculations show that the arbitrary intersection of compatible λ-term relations is compatible (the empty intersection being 0), whereas their union need not be so. Nonetheless, we can still give to the collection of compatible λ-term relations a complete lattice structure.

Lemma 11. *The collection of compatible λ-term relations forms a complete lattice ordered by $\subseteq$.*

Proof. Given a set ρ of compatible relations we define the meet of ρ as $\bigcap \rho$. We cannot define the join of ρ as $\bigcup \rho$, since the union of compatible λ-relations is not necessarily compatible. We get round the

problem by defining the join of ρ as

$$\bigcap \{\mathcal{S} \mid \widehat{\mathcal{S}} \subseteq \mathcal{S}, \bigcup \rho \subseteq \mathcal{S}\}.$$

It is easy to see that the latter indeed satisfies the universal property of the join. Finally, we observe that the bottom element is given by the discrete λ-term relation I, whereas the top element is the indiscrete λ-term relation 0. □

Lemma 11 allows us to define compatible λ-term relations both inductively and coinductively. Moreover, we can rely on Definition 27 to define a closure operator mapping a λ-term relation to the least compatible λ-term relation extending it.

Definition 28. *The compatible closure $\mathcal{R}^{\mathrm{cc}}$ of an open λ-term relation is inductively defined by the following rules.*

$$\frac{\Gamma \vdash^{\Lambda} e \,\mathcal{R}\, f}{\Gamma \vdash^{\Lambda} e \,\mathcal{R}^{\mathrm{cc}}\, f} \qquad \frac{\Gamma \vdash^{\mathrm{v}} v \,\mathcal{R}\, w}{\Gamma \vdash^{\mathrm{v}} v \,\mathcal{R}^{\mathrm{cc}}\, w} \qquad \frac{\Gamma \vdash^{\Lambda} e \,\widehat{\mathcal{R}^{\mathrm{cc}}}\, f}{\Gamma \vdash^{\Lambda} e \,\mathcal{R}^{\mathrm{cc}}\, f} \qquad \frac{\Gamma \vdash^{\mathrm{v}} v \,\widehat{\mathcal{R}^{\mathrm{cc}}}\, w}{\Gamma \vdash^{\mathrm{v}} v \,\mathcal{R}^{\mathrm{cc}}\, w}$$

It is easy to see that $-^{\mathrm{cc}}$ is a closure operator, i.e. it is a monotone and idempotent map extending the identity function (meaning that $\mathcal{R} \subseteq \mathcal{R}^{\mathrm{cc}}$, for any λ-term relation $\mathcal{R}$). Moreover, $\mathcal{R}^{\mathrm{cc}}$ is the least compatible λ-term relation containing $\mathcal{R}$.

We now have all the background notions needed to define and study interesting notions of program equivalence and refinement for Λ_{Σ}. Although our focus is on effectful applicative (bi)similarity, we first introduce effectful contextual approximation and equivalence. Most of the definitions given in the rest of this chapter are parametrised by a Σ-continuous monad $\mathbb{T} = \langle T, \eta, -^{\dagger} \rangle$ and a relator Γ for $\mathbb{T}$. As a consequence, we assume both $\mathbb{T}$ and Γ to be fixed, thus oftentimes omitting to mention them explicitly.

5.2 Effectful Contextual Approximation and Equivalence

The universally accepted notion of operational equivalence (resp. refinement) for sequential, higher-order language is Morris' style *contextual equivalence* (reps. approximation) (Morris, 1969). The latter is a syntax directed notion of equivalence (resp. refinement) equating (resp. ordering) programs according to a prefixed notion of observation (mostly based on notions of convergence). Accordingly, two programs are deemed as contextually equivalent if there is no context of the language (the latter being a kind of program with a hole to be filled in with the tested program) capable of detecting differences in the operational behaviour of the two programs, according to the notion of observation given. That is, thinking to contexts as specific environments or as testing scenarios, contextual equivalence deems two programs as equivalent, if no environment (resp. testing scenario) is capable of distinguish them.

We have already discussed in Chapter 2 the rationale behind the notions of contextual approximation and contextual equivalence. Here we simply recall that in most languages, contextual equivalence (resp. approximation) is characterised by a *universal property*: contextual equivalence (resp. approximation) is the *largest* (i.e. the less discriminating) *adequate* (resp. *preadequate*) *congruence* (resp. *precongruence*) λ-term relation. A λ-term relation $\mathcal{R}$ is adequate (reps. preadequate) if $\mathcal{R}$ relates programs exhibiting the same operational behaviour (resp. the operational behaviour of the second refines the operational behaviour of the first).

In this setting, (pre)adequacy predicates can be used in place of notions of observation. That is, instead of dealing with an explicit notion of observation, we fix a predicate $\mathsf{Adeq} \subseteq \mathsf{Rel}$ on λ-term relations according to the rationale that if a λ-term relation $\mathcal{R}$ belongs to Adeq, then it does not relate programs which are observationally distinguishable. For instance, in the pure, untyped λ-calculus one is usually interested in observing convergence of a term (according to a given reduction strategy (G. Plotkin, 1975)). As a consequence, a relation is said to be adequate if whenever it relates two programs, then one

of the programs converges if and only so does the other. We will follow a different route in Chapter 6, where we will work with an explicit notion of observation.

The calculus Λ_Σ being untyped, we follow the convention of not observing values, but only effects. As a consequence, our notion of adequacy will be defined with respect to a relator Γ only.

Definition 29. *A λ-term relation $\mathcal{R}$ is* Γ-adequate *(just adequate, when Γ is clear from the context) if $\vdash^\Lambda e\ \mathcal{R}\ f$ implies $[\![e]\!]\ \Gamma 0_\mathcal{V}\ [\![f]\!]$, where $0^c = (0_\Lambda, 0_\mathcal{V})$ is the closed indiscrete λ-term relation.*

Example 37. The following examples give a taste of the rationale behind Definition 29.

- Instantiating Definition 29 with the relator $\hat{\mathbb{M}}$, we see that a λ-term relation $\mathcal{R}$ over terms in $\Lambda_\mathbb{M}$ is adequate if whenever $\vdash^\Lambda e\ \mathcal{R}\ f$ holds, then if e converges, then so does f. Formally:
$$\vdash^\Lambda e\ \mathcal{R}\ f \implies ([\![e]\!] \neq \bot \implies [\![f]\!] \neq \bot). \tag{5.1}$$
Replacing $\hat{\mathbb{M}}$ with $\hat{\mathbb{M}} \wedge \hat{\mathbb{M}}^\circ$, we see that the rightmost implication of (5.1) becomes an equivalence.
- Instantiating Definition 29 with the relator $\mathbb{F}\hat{\mathbb{M}}$, we see that a λ-term relation $\mathcal{R}$ over terms in $\Lambda_{\mathbb{F}\mathbb{M}}$ is adequate if whenever $\vdash^\Lambda e\ \mathcal{R}\ f$ holds, then if e may converge, then so does f. Formally:
$$\vdash^\Lambda e\ \mathcal{R}\ f \implies ([\![e]\!] \neq \{\bot\} \implies [\![f]\!] \neq \{\bot\}). \tag{5.2}$$
Replacing $\mathbb{F}\hat{\mathbb{M}}$ with $\mathbb{F}\hat{\mathbb{M}} \wedge \mathbb{F}\hat{\mathbb{M}}^\circ$, we see that the rightmost implication of (5.2) becomes an equivalence.
- Instantiating Definition 29 with the relator $\mathbb{D}\hat{\mathbb{M}}$, we see that a λ-term relation $\mathcal{R}$ over terms in $\Lambda_{\mathbb{D}\mathbb{M}}$ is adequate if whenever $\vdash^\Lambda e\ \mathcal{R}\ f$ holds, then the probability of convergence of e is less or equal than the probability of convergence of f. Formally:
$$\vdash^\Lambda e\ \mathcal{R}\ f \implies [\![e]\!](\mathcal{V}_\circ) \leq [\![f]\!](\mathcal{V}_\circ). \tag{5.3}$$
Replacing $\mathbb{D}\hat{\mathbb{M}}$ with $\mathbb{D}\hat{\mathbb{M}} \wedge \mathbb{D}\hat{\mathbb{M}}^\circ$, we see that the inequality in (5.3) becomes an equality.

⊠

Remark 8 (Adequacy vs Preadequacy)**.** Definition 29 makes redundant the terminological distinction between adequacy and preadequacy. Accordingly, λ-term relations are just adequate with respect to a relator Γ. Example 37 suggests that for for the examples studied in Section 4.3, preadequate relations are captured by $\hat{\mathbb{T}}$-adequate relations. Conversely, adequate relations are captured by $(\hat{\mathbb{T}} \wedge \hat{\mathbb{T}}^\circ)$-adequate relations.

In light of Definition 29 we wish to define effectful contextual approximation (reps. equivalence) as the largest adequate precongruence (resp. congruence). We have already seen that the collection of compatible λ-term relations forms a complete lattice. However, we soon realise that adequacy is not a 'monotone property', and thus we cannot appeal to the Knaster-Tarski Theorem to infer the existence of the largest adequate precongruence. Following (S. Lassen, 1998b) we prove the existence of the desired relation explicitly.

Lemma 12. *Let $\alpha \subseteq \mathsf{Rel}$ be a predicate on open λ-term relations closed under non-empty union and composition. If $\mathsf{I} \in \alpha$, then there exists a largest compatible λ-term relation in α.*

Proof. Define the λ-term relation $\mathcal{S}$ as
$$\mathcal{S} \triangleq \bigcup \{\mathcal{R} \in \alpha \mid \widehat{\mathcal{R}} \subseteq \mathcal{R}\}.$$
By hypothesis $\mathsf{I} \in \alpha$, so that $\{\mathcal{R} \in \alpha \mid \widehat{\mathcal{R}} \subseteq \mathcal{R}\}$ is non-empty. As a consequence, since α is closed under non-empty union, $\mathcal{S} \in \alpha$. In order to conclude the thesis it remains to prove $\widehat{\mathcal{S}} \subseteq \mathcal{S}$. Formally, we prove by simultaneous induction the following statements:

1. If $\Gamma \vdash^{\Lambda} e \,\widehat{\mathcal{S}}\, f$, then $\Gamma \vdash^{\Lambda} e \,\mathcal{S}\, f$.
2. If $\Gamma \vdash^{\mathsf{v}} v \,\widehat{\mathcal{S}}\, w$, then $\Gamma \vdash^{\mathsf{v}} v \,\mathcal{S}\, w$.

We prove a couple cases as illustrative examples.

- Suppose $\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f \,\widehat{\mathcal{S}}\, \mathbf{let}\ x = e'\ \mathbf{in}\ f'$, so that we have $\Gamma \vdash^{\Lambda} e \,\mathcal{S}\, e'$ and $\Gamma, x \vdash^{\Lambda} f \,\mathcal{S}\, f'$. By very definition of $\mathcal{S}$ there exist compatible λ-term relations $\mathcal{R}_1, \mathcal{R}_2 \in \alpha$ such that $\Gamma \vdash^{\Lambda} e \,\mathcal{R}_1\, e'$ and $\Gamma, x \vdash^{\Lambda} f \,\mathcal{R}_2\, f'$. Since both $\mathcal{R}_1$ and $\mathcal{R}_2$ are compatible, then so is $\mathcal{R}_2 \cdot \mathcal{R}_1$. Moreover, since α is closed under composition, then $\mathcal{R}_2 \cdot \mathcal{R}_1 \in \alpha$. We also notice that, since compatibility implies reflexivity, both $\Gamma \vdash^{\Lambda} e \,(\mathcal{R}_2 \cdot \mathcal{R}_1)\, e'$ and $\Gamma, x \vdash^{\Lambda} f \,(\mathcal{R}_2 \cdot \mathcal{R}_1)\, f'$ hold. In fact, $\mathcal{R}_1 = \mathsf{I} \cdot \mathcal{R}_1 \subseteq \mathcal{R}_2 \cdot \mathcal{R}_1$. Therefore
$$\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f \,\widehat{(\mathcal{R}_2 \cdot \mathcal{R}_1)}\, \mathbf{let}\ x = e'\ \mathbf{in}\ f',$$
and thus (by compatibility of $\mathcal{R}_2 \cdot \mathcal{R}_1$)
$$\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f \,(\mathcal{R}_2 \cdot \mathcal{R}_1)\, \mathbf{let}\ x = e'\ \mathbf{in}\ f'.$$
We conclude the wished thesis by very definition of union of λ-term relations.
- Suppose $\Gamma \vdash^{\Lambda} \mathbf{op}(p, x.e) \,\widehat{\mathcal{S}}\, \mathbf{op}(p, x.f)$, so that $\Gamma, x \vdash^{\Lambda} e \,\mathcal{S}\, f$. As a consequence, there exists a compatible λ-term relation $\mathcal{R} \in \alpha$ such that $\Gamma, x \vdash^{\Lambda} e \,\mathcal{R}\, f$. Therefore, we have $\Gamma \vdash^{\Lambda} \mathbf{op}(p, x.e) \,\widehat{\mathcal{R}}\, \mathbf{op}(p, x.f)$ and thus $\Gamma \vdash^{\Lambda} \mathbf{op}(p, x.e) \,\mathcal{R}\, \mathbf{op}(p, x.f)$, by compatibility of $\mathcal{R}$. We can conclude $\Gamma \vdash^{\Lambda} \mathbf{op}(p, x.e) \,\mathcal{S}\, \mathbf{op}(p, x.f)$.

□

Lemma 13. *The adequacy property* $\mathsf{Adeq}_\Gamma \triangleq \{\mathcal{R} \in \mathsf{Rel} \mid \vdash^{\Lambda} e \,\mathcal{R}\, f \implies [\![e]\!]\,\Gamma 0_{\mathcal{V}}\,[\![f]\!]\}$ *contains the discrete open λ-term relation, and it is closed under non-empty union and composition.*

Proof. We show that the open λ-term relation I is in Adeq_Γ. We first notice that $\vdash^{\Lambda} e \mathsf{I} f$ implies $e \,\mathsf{I}_{\Lambda}\, f$, and thus $e = f$. By property (rel 1) we have that $[\![e]\!]\,\Gamma \mathsf{I}_{\mathcal{V}}\,[\![e]\!]$, and thus $[\![e]\!]\,\Gamma 0_{\mathcal{V}}\,[\![e]\!]$, since Γ is monotone. Let us now prove closure under relation composition. Given two open λ-term relations $\mathcal{R}, \mathcal{S} \in \mathsf{Adeq}_\Gamma$, assume $\vdash^{\Lambda} e \,(\mathcal{S} \cdot \mathcal{R})\, f$. We show $[\![e]\!]\,\Gamma 0_{\mathcal{V}}\,[\![f]\!]$. Since both $\mathcal{R}$ and $\mathcal{S}$ belong to Adeq_Γ, we obtain $[\![e]\!]\,(\Gamma 0_{\mathcal{V}} \cdot \Gamma 0_{\mathcal{V}})\,[\![f]\!]$. We conclude the thesis by property (rel 2). Finally, given a non-empty set $\rho \subseteq \mathsf{Adeq}_\Gamma$ of λ-term relations, we see that if $\vdash^{\Lambda} e \,(\bigcup \rho)\, f$, there there exists $\mathcal{R} \in \rho$ (and thus $\mathcal{R} \in \mathsf{Adeq}_\Gamma$) such that $\vdash^{\Lambda} e \mathcal{R} f$. We conclude $[\![f]\!]\,\Gamma 0_{\mathcal{V}}\,[\![f]\!]$. □

Definition 30. *Define the λ-term relation $\preceq^{\mathrm{ctx}}$, called* effectful contextual approximation with respect to Γ *(effectful contextual approximation or even contextual approximation, for short) as the largest compatible and Γ-adequate relation.*

Remark 9. The notion $\preceq^{\mathrm{ctx}}$ used to denote contextual approximation does not give any information about the relator defining the relevant notion of adequacy. For that reason, a better notation for contextual approximation would be $\preceq^{\mathrm{ctx}}_{\Gamma}$. Since most of the times Γ will be clear from the context, we will write $\preceq^{\mathrm{ctx}}$ in place of $\preceq^{\mathrm{ctx}}_{\Gamma}$ where possible. We also apply this convention to the rest of the equivalences and refinements we will study in this dissertation.

The reader might have noticed that the definition of the property Adeq_Γ involves only the behaviour of λ-term relations on computations, and does not say anything about their behaviour on values. This is rather obvious, as adequacy is an operational notion based on the evaluation semantics of a program. Formally, this does *not* mean that on values $\preceq^{\mathrm{ctx}}$ coincides with the indiscrete relation. In fact, compatibility forces $\preceq^{\mathrm{ctx}}$ to relate only those values that behave appropriately when used inside computations.

For instance, if $v \preceq^{\mathrm{ctx}}_{\mathcal{V}} v'$, then by compatibility (and reflexivity) of $\preceq^{\mathrm{ctx}}$ we have $vw \preceq^{\mathrm{ctx}}_{\Lambda} v'w$ too, which gives $[\![vw]\!]\ \Gamma 0_{\mathcal{V}}\ [\![v'w]\!]$. Obviously, this cannot be the case for all pairs of values v, v'.

Definition 30 comes with an associated proof technique resembling a coinduction proof principle. In order to prove that a program e contextually approximates a program f, it is sufficient to exhibit a compatible and adequate λ-relation relating e and f. Symbolically:

$$\frac{\widehat{\mathcal{R}} \subseteq \mathcal{R} \quad \mathcal{R} \in \mathsf{Adeq}_{\Gamma}}{\mathcal{R} \subseteq \preceq^{\mathrm{ctx}}} \ (\preceq^{\mathrm{ctx}}\text{-UMP})$$

We can use this proof technique to prove that $\preceq^{\mathrm{ctx}}$ is actually a precongruence relation.

Corollary 2. *The relation $\preceq^{\mathrm{ctx}}$ is a precongruence relation.*

Proof. Since the discrete λ-term relation I is the least fixed point of $\mathcal{R} \mapsto \widehat{\mathcal{R}}$ and belongs to Adeq_{Γ}, it is sufficient to show that $\preceq^{\mathrm{ctx}}$ is transitive. We use the proof technique associated with the definition of $\preceq^{\mathrm{ctx}}$ showing that $\widehat{(\preceq^{\mathrm{ctx}} \cdot \preceq^{\mathrm{ctx}})} \subseteq (\preceq^{\mathrm{ctx}} \cdot \preceq^{\mathrm{ctx}})$ and $(\preceq^{\mathrm{ctx}} \cdot \preceq^{\mathrm{ctx}}) \in \mathsf{Adeq}_{\Gamma}$. The former obviously holds, since the composition of compatible λ-term relations is compatible. For the latter, we simply observe that in Lemma 13 we show Adeq_{Γ} to be closed under composition. □

Finally, we notice that we can extend Corollary 2 showing that if Γ is conversive, then $\preceq^{\mathrm{ctx}}$ is symmetric. In light of this observation we give the following definition of effectful contextual equivalence.

Definition 31. *The λ-term relation $\simeq^{\mathrm{ctx}}$, called* effectful contextual equivalence with respect to Γ *(effectful contextual equivalence or even contextual equivalence, for short) is defined as $\preceq^{\mathrm{ctx}}_{\Gamma \wedge \Gamma^{\circ}}$. In particular, $\simeq^{\mathrm{ctx}}$ is the largest compatible relation in $\mathsf{Adeq}_{\Gamma \wedge \Gamma^{\circ}}$.*

Notice that we are applying the convention of Remark 9 in the notation employed for effectful contextual equivalence. It is straightforward to see that $\simeq^{\mathrm{ctx}}$ is a congruence relation. Additionally, we can characterise $\simeq^{\mathrm{ctx}}$ as follows.

Proposition 15. $\simeq^{\mathrm{ctx}} = \preceq^{\mathrm{ctx}} \cap (\preceq^{\mathrm{ctx}})^{\circ}$.

Proof. The proof uses the proof techniques associated with $\preceq^{\mathrm{ctx}}$ and $\simeq^{\mathrm{ctx}}$, and is rather straightforward. The only relevant points to notice are the following: $(\preceq^{\mathrm{ctx}}_{\Gamma})^{\circ} = \preceq^{\mathrm{ctx}}_{\Gamma^{\circ}}$ and $\mathsf{Adeq}_{\Gamma \wedge \Gamma^{\circ}} = \mathsf{Adeq}_{\Gamma} \cap \mathsf{Adeq}_{\Gamma^{\circ}}$. □

5.3 Effectful Applicative Similarity and Bisimilarity

We now introduce effectful applicative similarity and bisimilarity with respect to a relator Γ. As usual, we assume a Σ-continuous monad $\mathbb{T}$ and relator Γ for it to be fixed. We begin our analysis defining the notion of an *effectful applicative simulation* relation with respect to a relator Γ (effectful applicative similarity, for short). Contrary to effectful contextual approximation, effectful applicative similarity is defined as a *closed* λ-term relation, and then extended to an open λ-term relation by means of its open extension.

Definition 32. *Given a closed λ-term relation $\mathcal{R} = (\mathcal{R}_{\Lambda}, \mathcal{R}_{\mathcal{V}})$ we define the closed λ-term relation $[\mathcal{R}] = ([\mathcal{R}]_{\Lambda}, [\mathcal{R}]_{\mathcal{V}})$ as follows:*

$$\vdash^{\Lambda} e\ [\mathcal{R}]\ f \overset{\triangle}{\iff} [\![e]\!]\ \Gamma \mathcal{R}_{\mathcal{V}}\ [\![f]\!]$$
$$\vdash^{\mathcal{V}} v\ [\mathcal{R}]\ w \overset{\triangle}{\iff} \forall u \in \mathcal{V}_{\circ}.\ \vdash^{\Lambda} vu\ \mathcal{R}_{\Lambda}\ wu.$$

A λ-term relation $\mathcal{R}$ is an effectful applicative simulation with respect to Γ *(effectful applicative simulation or even applicative simulation, for short) if $\mathcal{R} \subseteq [\mathcal{R}]$.*

Definition 32 can be equivalently stated by saying that a closed λ-term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ is an applicative simulation is the following conditions hold:

$$\vdash^\Lambda e \,\mathcal{R}\, f \implies [\![e]\!]\,\Gamma\mathcal{R}_\mathcal{V}\,[\![f]\!] \qquad \text{(app comp)}$$
$$\vdash^\mathsf{v} v \,\mathcal{R}\, w \implies \forall u \in \mathcal{V}_\circ.\ \vdash^\Lambda vu \,\mathcal{R}\, wu. \qquad \text{(app val)}$$

Definition 32 induces a map $\mathcal{R} \mapsto [\mathcal{R}]$ on the complete lattice of closed λ-term relations. Since Γ is monotone, then so is $\mathcal{R} \mapsto [\mathcal{R}]$.

Lemma 14. *The map $\mathcal{R} \mapsto [\mathcal{R}]$ is monotone.*

Proof. Suppose $\mathcal{R} \subseteq \mathcal{S}$. We first prove that $\vdash^\Lambda e\,[\mathcal{R}]\,f$ implies $\vdash^\Lambda e\,[\mathcal{S}]\,f$, i.e. $[\![e]\!]\,\Gamma\mathcal{S}_\mathcal{V}\,[\![f]\!]$. By hypothesis we have $[\![e]\!]\,\Gamma\mathcal{R}_\mathcal{V}\,[\![f]\!]$. Moreover, since $\mathcal{R} \subseteq \mathcal{S}$ and Γ is monotone, we can conclude $\Gamma\mathcal{R} \subseteq \Gamma\mathcal{S}$, and thus the wished implication. We now prove that $\vdash^\mathsf{v} v\,[\mathcal{R}]\,w$ implies $\vdash^\Lambda v\,[\mathcal{S}]\,w$, i.e. $\vdash^\Lambda vu\,\mathcal{S}\,wu$, for any closed value u. Since $\vdash^\mathsf{v} v\,[\mathcal{R}]\,w$, we have $\vdash^\Lambda vu\,\mathcal{R}\,wu$, and thus $\vdash^\Lambda vu\,\mathcal{S}\,wu$, because $\mathcal{R} \subseteq \mathcal{S}$. □

By Lemma 14 we can define effectful applicative similarity with respect to Γ, denoted by $\preceq^\mathsf{A}$, as the greatest fixed point of $\mathcal{R} \mapsto [\mathcal{R}]$. Obviously, $\preceq^\mathsf{A}$ is the largest closed λ-term relations satisfying conditions (app comp) and (app val).

Definition 33. *Define the closed λ-term relation $\preceq^\mathsf{A}$, called* effectful applicative similarity with respect to Γ *(effectful applicative similarity or even applicative similarity, for short), as the greatest fixed point of $\mathcal{R} \mapsto [\mathcal{R}]$ (which exists by the Knaster-Tarski Theorem).*

Again, notice that we are tacitly applying the notational convention of Remark 9. Additionally, since in this chapter we will deal with applicative similarity only, oftentimes we will use the lighter notation $\preceq$ in place of $\preceq^\mathsf{A}$ (the latter will be useful in later chapters, where other notions of similarity will be studied).

Applicative similarity comes with an associated coinduction proof principle: in order to prove that a program f applicatively refines a program e, it is sufficient to exhibit an applicative simulation relating them. Symbolically:

$$\frac{\mathcal{R} \subseteq [\mathcal{R}]}{\mathcal{R} \subseteq \preceq^\mathsf{A}}\ (\preceq^\mathsf{A}\text{-coind.})$$

Example 38. Instantiating Definition 33 with relators of the form $\hat{\mathbb{T}}$ as defined in Section 4.3, we obtain well known notions of applicative similarity. Notably, taking Γ to be $\hat{\mathbb{M}}$, $\hat{\mathbb{FM}}$, $\hat{\mathbb{DM}}$, and $\hat{\mathbb{C}}$ we recover (call-by-value) Abramsky's applicative similarity (Abramsky, 1990a), (lower) nondeterministic applicative similarity (S. Lassen, 1998b; C. L. Ong, 1993), probabilistic applicative similarity (Crubillé & Dal Lago, 2014; Dal Lago et al., 2014), and Sands' improvement similarity (Sands, 1998), respectively. ⊠

We now give an example of how we can use the coinduction proof principle to prove behavioural refinements between programs. Our example is given in Λ_p.

Example 39. Instantiating Definition 33 with the partial distribution monad $\mathbb{DM}$ and its associated relator $\hat{\mathbb{DM}}$ we recover probabilistic applicative similarity for Λ_p. Let us consider computations $e \triangleq (\mathbf{return}\ I)\ \mathbf{or}\ \Omega$, and $f \triangleq \mathbf{return}\ I$, where Ω is the purely divergent program $(\lambda x.xx)(\lambda x.xx)$ and $I \triangleq \lambda x.\mathbf{return}\ x$. We show that $e \preceq_\Lambda f$. Let us consider the λ-term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ whose graph is

$$G_{\mathcal{R}_\Lambda} \triangleq \{((\mathbf{return}\ I)\ \mathbf{or}\ \Omega, \mathbf{return}\ I)\}$$
$$G_{\mathcal{R}_\mathcal{V}} \triangleq \{(I, I)\}.$$

In order to prove that $\mathcal{R}$ is an applicative simulation, we have to show $[\![e]\!]\ \mathbb{D}\hat{\mathbb{M}}\mathcal{R}_{\mathcal{V}}\ [\![f]\!]$, that is:

$$\left(\frac{1}{2}\cdot \mathit{just}\ I + \frac{1}{2}\cdot \bot\right)\ \mathbb{D}\hat{\mathbb{M}}\mathcal{R}_{\mathcal{V}}\ \left(1\cdot I\right).$$

Define the distribution ω over $\mathbb{M}\mathcal{V}_\circ \times \mathbb{M}\mathcal{V}_\circ$ as follows:

$$\omega(\mathit{just}\ I, \mathit{just}\ I) \triangleq \frac{1}{2} \qquad\qquad \omega(\bot, \mathit{just}\ I) \triangleq \frac{1}{2}$$

and zero on all other elements. Clearly $\omega \in \Omega([\![e]\!], [\![f]\!])$. We have to prove that for all $\mathfrak{v}, \mathfrak{w} \in \mathbb{M}\mathcal{V}_\circ$, $\omega(\mathfrak{v}, \mathfrak{w}) > 0$, implies $\mathfrak{v}\ \hat{\mathbb{M}}\mathcal{R}_{\mathcal{V}}\ \mathfrak{w}$. The only cases to consider are for $\mathfrak{v} = \mathit{just}\ v$ and $\mathfrak{v} = \bot$. In the latter case we are trivially done by very definition of $\hat{\mathbb{M}}$. In the former case we have $\mathfrak{w} = \mathit{just}\ I$ too, and thus we are done since $I\ \mathcal{R}_{\mathcal{V}}\ I$. ⊠

We now analyse the metatheoretical properties of applicative similarity. First of all, we see that the coinduction proof principle allows for an handy proof of reflexivity and transitivity of $\preceq$.

Proposition 16. *Applicative similarity $\preceq$ is a reflexive and transitive (closed) λ-term relation.*

Proof. The proof is by coinduction. In order to prove reflexivity we show that the closed λ-term relation $\mathsf{I}^c = (\mathsf{I}_\Lambda, \mathsf{I}_{\mathcal{V}})$ is an applicative simulation. Clause (app val) is obviously satisfied, so that it remains to prove clause (app comp). That amounts to show $[\![e]\!]\ \Gamma\mathsf{I}_{\mathcal{V}}\ [\![e]\!]$. This is indeed the case, since by condition (rel 1) we have

$$\mathsf{I}_{T\mathcal{V}_\circ} \subseteq \Gamma(\mathsf{I}_{\mathcal{V}_\circ}) = \Gamma\mathsf{I}_{\mathcal{V}}$$

and obviously $[\![e]\!]\ \mathsf{I}_{T\mathcal{V}_\circ}\ [\![e]\!]$ holds. We prove transitivity of $\preceq$ by showing that $\preceq\ \cdot\ \preceq$ is an applicative simulation. Again, the clause (app val) is obviously satisfied. Let us move to (app comp). We have to show that $\vdash^\Lambda e\ (\preceq\ \cdot\ \preceq)\ f$ implies $[\![e]\!]\ \Gamma(\preceq_{\mathcal{V}}\ \cdot\ \preceq_{\mathcal{V}})\ [\![f]\!]$. By very definition of transitivity we have:

$$\begin{aligned}
\vdash^\Lambda e\ (\preceq\ \cdot\ \preceq)\ f &\implies [\![e]\!]\ (\Gamma\preceq_{\mathcal{V}}\ \cdot\ \Gamma\preceq_{\mathcal{V}})\ [\![f]\!] \\
&\quad \text{[By (app comp) and Definition 33]} \\
&\implies [\![e]\!]\ \Gamma(\preceq_{\mathcal{V}}\ \cdot\ \preceq_{\mathcal{V}})\ [\![f]\!]. \\
&\quad \text{[By (rel 2)]}
\end{aligned}$$

□

Next, we notice that since Γ is monotone, $\preceq\ \in \mathsf{Adeq}_\Gamma$. For:

$$\begin{aligned}
\vdash^\Lambda e \preceq f &\implies [\![e]\!]\ \Gamma\preceq_{\mathcal{V}}\ [\![f]\!] \\
&\quad \text{[By (app comp)]} \\
&\implies [\![e]\!]\ \Gamma 0_{\mathcal{V}}\ [\![f]\!]. \\
&\quad \text{[By (rel 4) since } \preceq_{\mathcal{V}} \subseteq 0_{\mathcal{V}}\text{]}
\end{aligned}$$

The last (but certainly not least) property we wish to prove about applicative similarity is compatibility, meaning that effectful applicative similarity is a (pre)adequate precongruence relation, and thus a sound proof technique for effectful contextual approximation. However, as we saw in Chapter 2, proving $\preceq$ to be compatible from first principles is bound to be hard, due to substitutivity. We prove a compatibility theorem for applicative similarity using a generalisation of Howe's method (Howe, 1996). Before that, let us introduce *effectful applicative bisimilarity*.

Definition 34. *Define the closed λ-term relation $\simeq^\Lambda$, called* effectful applicative bisimilarity with respect to Γ *(effectful applicative bisimilarity or even applicative bisimilarity, for short), as effectful applicative similarity with respect to $\Gamma \wedge \Gamma^\circ$.*

We apply the same notational and terminological conventions introduced for effectful applicative similarity to effectful applicative bisimilarity. In particular, throughout this chapter, we oftentimes write $\simeq$ in place of $\simeq^{\mathsf{A}}$.

A first consequence of Definition 34 is that $\simeq$ is the largest applicative simulation with respect to $\Gamma \wedge \Gamma^{\circ}$. Additionally, a straightforward proof by coinduction shows that $\simeq$ is indeed symmetric (since $\Gamma \wedge \Gamma^{\circ}$ is conversive) and contained in $\preceq$ (since $\Gamma \wedge \Gamma^{\circ} \leq \Gamma$). Finally, we give the following alternative characterisation of $\simeq$, which will be useful when proving applicative bisimilarity to be a congruence relation.

Proposition 17. *Effectful applicative bisimilarity is the largest symmetric applicative simulation with respect to* Γ.

Proof. Define the closed λ-term relation $\mathcal{S}$ as

$$\mathcal{S} \triangleq \bigcup \{\mathcal{R} \mid \mathcal{R} = \mathcal{R}^{\circ}, \mathcal{R} \subseteq [\mathcal{R}]\}.$$

Obviously $\mathcal{S}$ is a symmetric applicative simulation with respect to Γ, and actually the largest such. We prove $\simeq \subseteq \mathcal{S}$ using the coinduction proof principle associated with $\mathcal{S}$. To do so, it is sufficient to show that $\simeq$ is a symmetric applicative simulation with respect to Γ, which is indeed the case. Conversely, we prove $\mathcal{S} \subseteq \simeq$ using the coinduction proof principle associated with $\simeq$. Doing so amounts to show that $\vdash^{\wedge} e \mathcal{S} f$ implies $[\![e]\!] \Gamma \mathcal{S}_{\mathcal{V}} [\![f]\!]$ and $[\![f]\!] \Gamma^{\circ}(\mathcal{S}_{\mathcal{V}}) [\![e]\!]$. Since $\mathcal{S}$ is symmetric, the latter is equivalent to $[\![f]\!] \Gamma \mathcal{S}_{\mathcal{V}} [\![e]\!]$, and thus we can conclude the thesis since $\mathcal{S}$ is a symmetric applicative simulation with respect to Γ. □

It is now time to introduce the main results of this chapter, namely a precongruence theorem for effectful applicative similarity, and a congruence theorem for effectful applicative bisimilarity.

5.4 Howe's Method

Having observed the difficulties one encounters when trying to prove $\preceq$ to be compatible from first principles, in this section we develop a generalisation of Howe's method (Howe, 1996; Pitts, 2011) and use it to prove a precongruence theorem for $\preceq$, and a congruence theorem for $\simeq$.

At the heart of Howe's method is a relational construction (called precongruence candidate in (Howe, 1996)) extending $\preceq$ to a substitutive and compatible relation $\preceq^{H}$. The key ingredient to make such a method work is the so-called *Key Lemma*. The latter essentially states that $\preceq^{H}$ is an applicative simulation, and thus coincide with $\preceq$. Proving the Key Lemma is notoriously hard in presence of specific effects, the case of probabilistic calculi being a prime example (Dal Lago et al., 2014). Perhaps surprisingly, our proof of the Key Lemma is rather easy, and relies on structural properties of Σ-continuous relators only. This allows to separate the core of Howe's method, viewed as a syntactical construction, from its soundness, which instead is based on semantical and operational properties of the effects considered. Let us now enter the details of Howe's method.

Definition 35 (Howe extension, 1)**.** *Given a closed λ-term relation $\mathcal{R}$, define the open λ-term relation $\mathcal{R}^{H}$, called the* Howe extension *of $\mathcal{R}$, as the least fixed point of the map $\mathcal{X} \mapsto \mathcal{R}^{o} \cdot \widehat{\mathcal{X}}$.*

Since both $\mathcal{X} \mapsto \widehat{\mathcal{X}}$ and $\mathcal{X} \mapsto \mathcal{X}^{o}$ are monotone, then so is $\mathcal{X} \mapsto \mathcal{R}^{o} \cdot \widehat{\mathcal{X}}$, meaning that, by the Knaster-Tarski Theorem, Definition 35 indeed defines λ-term relation. In particular, $\mathcal{R}^{H}$ is the least λ-term relation satisfying $\mathcal{R}^{o} \cdot \mathcal{R}^{H} \subseteq \mathcal{R}^{H}$ (see (S. Lassen, 1998b; P. Levy, 2006) and Lemma 15 below). An equivalent, more concrete characterisation of $\mathcal{R}^{H}$ is given by the following definition.

Definition 36 (Howe extension, 2). *Given a* closed *λ-term relation $\mathcal{R}$, the* Howe extension *$\mathcal{R}^H$ of $\mathcal{R}$ is the* open *λ-term relation inductively defined as follows:*

$$\frac{\Gamma \vdash^{\Lambda} e \; \widehat{\mathcal{R}^H} \; g \quad \Gamma \vdash^{\Lambda} g \; \mathcal{R} \; f}{\Gamma \vdash^{\Lambda} e \; \mathcal{R}^H \; f} \qquad \frac{\Gamma \vdash^{\mathrm{v}} v \; \widehat{\mathcal{R}^H} \; u \quad \Gamma \vdash^{\mathrm{v}} u \; \mathcal{R} \; w}{\Gamma \vdash^{\mathrm{v}} v \; \mathcal{R}^H \; w}$$

Additionally, unfolding the definition of $\widehat{\mathcal{R}}$, we see that $\mathcal{R}^H$ is inductively defined by the rules in Figure 5.2.

$$\frac{\Gamma, x \vdash^{\mathrm{v}} x \; \mathcal{R} \; v}{\Gamma, x \vdash^{\mathrm{v}} x \; \mathcal{R}^H \; v} \text{ (H-var)}$$

$$\frac{\Gamma, x \vdash^{\Lambda} e \; \mathcal{R}^H \; f \quad \Gamma \vdash^{\mathrm{v}} \lambda x.f \; \mathcal{R} \; v}{\Gamma \vdash^{\mathrm{v}} \lambda x.e \; \widehat{\mathcal{R}} \; v} \text{ (H-abs)} \qquad \frac{\Gamma \vdash^{\mathrm{v}} v \; \mathcal{R}^H \; v' \quad \Gamma \vdash^{\mathrm{v}} w \; \mathcal{R}^H \; w' \quad \Gamma \vdash^{\Lambda} v'w' \; \mathcal{R} \; e}{\Gamma \vdash^{\Lambda} vw \; \mathcal{R}^H \; e} \text{ (H-app)}$$

$$\frac{\Gamma \vdash^{\mathrm{v}} v \; \mathcal{R}^H \; w \quad \Gamma \vdash^{\Lambda} \mathbf{return}\; w \; \mathcal{R} \; e}{\Gamma \vdash^{\Lambda} \mathbf{return}\; v \; \mathcal{R}^H \; e} \text{ (H-val)}$$

$$\frac{\Gamma \vdash^{\Lambda} e \; \mathcal{R}^H \; e' \quad \Gamma, x \vdash^{\Lambda} f \; \mathcal{R}^H \; f' \quad \Gamma \vdash^{\Lambda} \mathbf{let}\; x = e' \;\mathbf{in}\; f' \; \mathcal{R} \; g}{\Gamma \vdash^{\Lambda} \mathbf{let}\; x = e \;\mathbf{in}\; f \; \mathcal{R}^H \; g} \text{ (H-let)}$$

$$\frac{\Gamma, x \vdash^{\Lambda} e \; \mathcal{R}^H \; f \quad \Gamma \vdash^{\Lambda} \mathbf{op}(p, x.f) \; \mathcal{R} \; g}{\Gamma \vdash^{\Lambda} \mathbf{op}(p, x.e) \; \mathcal{R}^H \; g} \text{ (H-op)}$$

Figure 5.2: Howe extension of $\mathcal{R}$.

The Howe extension of preorder λ-term relation satisfies many nice properties. These are summarised by the following result.

Lemma 15. *Let $\mathcal{R}$ be reflexive and transitive closed λ-term relation. Then the following hold:*

1. $\mathcal{R}^o \subseteq \mathcal{R}^H$.
2. $\mathcal{R}^o \cdot \mathcal{R}^H \subseteq \mathcal{R}^H$
3. *$\mathcal{R}^H$ is compatible, and thus reflexive.*
4. *$\mathcal{R}^H$ is substitutive.*

Proof. The proof of Lemma 15 is standard (this comes with no surprise as the Lemma 15 focuses on syntactical, rather than semantical, properties of Howe's construction). The reader is referred to (S. Lassen, 1998b; Pitts, 2011) for detailed proofs. Here we sketch a proof of substitutivity of $\mathcal{R}^H$. We prove by simultaneous induction on $\Gamma, z \vdash^{\Lambda} e \; \mathcal{R}^H \; f$ and $\Gamma, z \vdash^{\mathrm{v}} v \; \mathcal{R}^H \; w$ the admissibility of the following rules:

$$\frac{\Gamma, z \vdash^{\Lambda} e \; \mathcal{R} \; f \quad \vdash^{\mathrm{v}} v \; \mathcal{R} \; w}{\Gamma \vdash^{\Lambda} e[x := v] \; \mathcal{R} \; f[x := w]} \qquad \frac{\Gamma, z \vdash^{\mathrm{v}} v \; \mathcal{R} \; w \quad \vdash^{\mathrm{v}} u \; \mathcal{R} \; u'}{\Gamma \vdash^{\mathrm{v}} v[u/x] \; \mathcal{R} \; w[u'/x]}$$

The key point is to observe that the definition of $\mathcal{R}^H$ involves the open extension of $\mathcal{R}$, which is value-substitutive by construction. As an illustrative example, we show how to handle the case corresponding

to rule (H-let). Concretely, we prove we prove $\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e[z := v]\ \mathbf{in}\ f[z := v]\ \mathcal{R}^H\ g[z := w]$ given the derivation

$$\frac{\Gamma, z \vdash^{\Lambda} e\ \mathcal{R}^H\ e' \quad \Gamma, z, x \vdash^{\Lambda} f\ \mathcal{R}^H\ f' \quad \Gamma, z \vdash^{\Lambda} \mathbf{let}\ x = e'\ \mathbf{in}\ f'\ \mathcal{R}\ g}{\Gamma, z \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f\ \mathcal{R}^H\ g}\ \text{(H-let)}$$

(notice that in the third premise, we are actually referring to the open extension of $\mathcal{R}$, which is trivially value substitutive). We apply the induction hypothesis on the first and second premise, obtaining:

$$\Gamma \vdash^{\Lambda} e[z := v]\ \mathcal{R}^H\ e'[z := w] \qquad\qquad \Gamma, x \vdash^{\Lambda} f[z := v]\ \mathcal{R}^H\ f'[z := w].$$

Since the open extension of $\mathcal{R}$ is value-substitutive, from the third premise we obtain

$$\Gamma \vdash^{\Lambda} \mathbf{let}\ x = e'[z := w]\ \mathbf{in}\ f'[z := w]\ \mathcal{R}\ g[z := w],$$

and thus an application of rule (H-let) gives the desired thesis. □

Since $\preceq$ is reflexive and transitive we can apply Lemma 15 on $\preceq^H$. We obtain a compatible and substitutive relation extending $\preceq^H$. In particular, by property (rel 2), Lemma 15 gives the following inequality, which we refer to as pseudo-transitivity of $\preceq^H$:

$$\Gamma\preceq \cdot \Gamma\preceq^H \subseteq \Gamma\preceq. \qquad \text{(pseudo-trans)}$$

We now prove the Key Lemma. Before entering the proof, the reader is invited to observe that so far none of our proofs and definitions rely on the relator Γ being inductive nor on Γ being a relator for $\mathbb{T}$ (we only relied on Γ being a relator for T only). This will not be the case for the Key Lemma, as its proof requires Γ to be Σ-continuous.

Lemma 16 (Key Lemma)**.** *Let Γ be Σ-continuous. Then the $(\preceq^H)^c$ is an applicative simulation.*

Proof. We first notice that $\vdash^{\mathsf{v}} v \preceq^H w$ implies $\vdash^{\Lambda} vu \preceq^H vu$, for any closed value u. In fact, $\preceq^H$ is reflexive (meaning that $\vdash^{\mathsf{v}} u \preceq^H u$ holds) and compatible. This shows that $(\preceq^H)^c$ satisfies condition (app val). Next, we show $(\preceq^H)^c$ satisfies condition (app comp) too. That amounts to prove that $\vdash^{\Lambda} e \preceq^H f$ implies $[\![e]\!]\Gamma\preceq^H_{\mathcal{V}} [\![f]\!]$. Since $[\![e]\!] = \bigsqcup_n [\![e]\!]_n$ and Γ is inductive, we can appeal to property (ind 3). As a consequence, it is sufficient to prove the following statement:

$$\vdash^{\Lambda} e \preceq^H f \implies \forall n \geq 0.\ [\![e]\!]_n\ \Gamma\preceq^H_{\mathcal{V}} [\![f]\!].$$

We assume $\vdash^{\Lambda} e \preceq^H f$ and proceed by induction on n. The case for $n = 0$ requires to prove $\bot\ \Gamma\preceq^H_{\mathcal{V}} [\![f]\!]$, which indeed holds since Γ is inductive. Let us not look at the $(n+1)$-case. We proceed by case analysis according to the definition of $[\![-]\!]_n$.

- Suppose $e = \mathbf{return}\ v$, so that $[\![e]\!]_{n+1} = \eta(v)$. We have to show:

  $$\vdash^{\Lambda} \mathbf{return}\ v \preceq^H f \implies \eta(v)\ \Gamma\preceq^H_{\mathcal{V}} [\![f]\!].$$

  Assume the antecedent of the above implication. By very definition of Howe extension, the latter must be the conclusion of a derivation of the form:

  $$\frac{\vdash^{\mathsf{v}} v \preceq^H w \quad \vdash^{\Lambda} \mathbf{return}\ w \preceq f}{\vdash^{\Lambda} \mathbf{return}\ v \preceq^H f}\ \text{(H-val)}$$

  By (app comp), we see that $\vdash^{\Lambda} \mathbf{return}\ w \preceq f$ implies $\eta(w)\ \Gamma\preceq_{\mathcal{V}} [\![f]\!]$, whereas $\vdash^{\mathsf{v}} v \preceq^H w$ implies $\eta(v)\ \Gamma\preceq^H_{\mathcal{V}} \eta(w)$, by (rel unit). As a consequence, we have $\eta(v)\ (\Gamma\preceq_{\mathcal{V}} \cdot \Gamma(\preceq^H)_{\mathcal{V}})\ [\![f]\!]$ and thus we conclude the thesis by (pseudo-trans).

- Suppose $e = (\lambda x.g)v$, so that $[\![e]\!]_{n+1} = [\![g[x := v]]\!]_n$. We have to show:

$$\vdash^\Lambda (\lambda x.g)v \preceq^H f \implies [\![g[x := v]]\!]_n \, \Gamma \preceq^H_\mathcal{V} .$$

Assume the antecedent of the above implication. By very definition of Howe extension, the latter must be the conclusion of a derivation of the form:

$$\dfrac{\dfrac{x \vdash^\Lambda g \preceq^H h \quad \vdash^\mathsf{v} \lambda x.h \preceq u}{\vdash^\mathsf{v} \lambda x.g \preceq^H u}\ \text{(H-abs)} \qquad \vdash^\mathsf{v} v \preceq^H w \quad \vdash^\Lambda uw \preceq f}{\vdash^\Lambda (\lambda x.g)v \preceq^H f}\ \text{(H-app)}$$

We have:

$$\begin{aligned} x \vdash^\Lambda g \preceq^H h,\ \vdash^\mathsf{v} v \preceq^H w &\implies \vdash^\Lambda g[x := v] \preceq^H h[x := w] \\ &\quad \text{[By Lemma 15 (substitutivity)]} \\ &\implies [\![g[x := v]]\!]_n \, \Gamma \preceq^H_\mathcal{V} [\![h[x := w]]\!]. \\ &\quad \text{[By induction hypothesis]} \end{aligned}$$

Additionally:

$$\begin{aligned} \vdash^\mathsf{v} \lambda x.h \preceq u &\implies \vdash^\mathsf{v} (\lambda x.h)w \preceq uw \\ &\quad \text{[By (app val)]} \\ &\implies [\![h[x := w]]\!] \, \Gamma \preceq_\mathcal{V} [\![uw]\!]. \\ &\quad \text{[By (app comp)]} \end{aligned}$$

As a consequence, we conclude $[\![g[x := v]]\!]_n \, \Gamma \preceq^H_\mathcal{V} [\![uw]\!]$, by (pseudo-trans). Finally, since $\vdash^\Lambda uw \preceq f$ implies $[\![uw]\!] \, \Gamma \preceq_\mathcal{V} [\![f]\!]$, by (app comp), we conclude the wished thesis, by (pseudo-trans).

- Suppose $e = (\mathbf{let}\ x = g\ \mathbf{in}\ h)$, so that $[\![e]\!]_{n+1} = [\![h[x := -]]\!]^\dagger_n [\![g]\!]_n$. We have to show:

$$\vdash^\Lambda \mathbf{let}\ x = g\ \mathbf{in}\ h \preceq^H f \implies [\![h[x := -]]\!]^\dagger_n [\![g]\!]_n \, \Gamma \preceq^H_\mathcal{V} [\![f]\!].$$

Assume the antecedent of the above implication. By very definition of Howe extension, the latter must be the conclusion of a derivation of the form:

$$\dfrac{\vdash^\Lambda g \preceq^H g' \quad x \vdash^\Lambda h \preceq^H h' \quad \vdash^\Lambda \mathbf{let}\ x = g'\ \mathbf{in}\ h' \preceq f}{\vdash^\Lambda \mathbf{let}\ x = g\ \mathbf{in}\ h \preceq^H f}\ \text{(H-let)}$$

First of all we notice that in order to prove the thesis, it is sufficient to prove:

$$[\![h[x := -]]\!]^\dagger_n [\![g]\!]_n \, \Gamma \preceq^H_\mathcal{V} [\![h'[x := -]]\!]^\dagger [\![g']\!]. \tag{5.4}$$

For, we have:

$$\begin{aligned} \vdash^\Lambda \mathbf{let}\ x = g'\ \mathbf{in}\ h' \preceq f &\implies [\![h'[x := -]]\!]^\dagger [\![g']\!] \, \Gamma \preceq_\mathcal{V} [\![f]\!] \\ &\quad \text{[By (app comp)]} \\ &\implies [\![h[x := -]]\!]^\dagger_n [\![g]\!]_n \, \Gamma \preceq^H_\mathcal{V} [\![f]\!]. \\ &\quad \text{[By (5.4) and (pseudo-trans)]} \end{aligned}$$

Let us prove (5.4). Applying the induction hypothesis on $\vdash^\Lambda g \preceq^H g'$ we obtain $[\![g]\!]_n \, \Gamma \preceq^H_\mathcal{V} [\![g']\!]$. As a consequence, by condition (rel bind), in order to prove (5.4) it is sufficient to prove:

$$\vdash^\mathsf{v} v \preceq^H w \implies [\![h[x := v]]\!]_n \, \Gamma \preceq^H_\mathcal{V} [\![h'[x := w]]\!].$$

The latter is a direct consequence of substitutivity of $\preceq^H$ and of the induction hypothesis. More precisely:

$$\begin{aligned}\vdash^{\mathsf{v}} v \preceq^H w &\implies \vdash^{\Lambda} h[x := v] \preceq^H h'[x := w] \\ &\quad \text{[By Lemma 15 (substitutivity) and } x \vdash^{\Lambda} h \preceq^H h'] \\ &\implies [\![h[x := v]]\!]_n \, \Gamma \preceq^H_{\mathcal{V}} [\![h'[x := w]]\!]. \\ &\quad \text{[By induction hypothesis]}\end{aligned}$$

- Suppose $e = \mathbf{op}(p, x.g)$, so that $[\![e]\!]_{n+1} = [\![\mathbf{op}]\!](p, v \mapsto [\![g[x := v]]\!]_n)$. We have to to show:

$$\vdash^{\Lambda} \mathbf{op}(p, x.g) \preceq^H f \implies [\![\mathbf{op}]\!](p, v \mapsto [\![g[x := v]]\!]_n) \, \Gamma \preceq^H_{\mathcal{V}} [\![f]\!].$$

Assume the antecedent of the above implication. By very definition of Howe extension, the latter must be the conclusion of a derivation of the form:

$$\frac{x \vdash^{\Lambda} g \preceq^H h \quad \vdash^{\Lambda} \mathbf{op}(p, x.h) \preceq f}{\vdash^{\Lambda} \mathbf{op}(p, x.g) \preceq^H f} \text{ (H-op)}$$

As for previous cases, by (pseudo-trans) in order to prove the thesis it is sufficient to show:

$$[\![\mathbf{op}]\!](p, v \mapsto [\![g[x := v]]\!]_n) \, \Gamma \preceq^H_{\mathcal{V}} [\![\mathbf{op}]\!](p, v \mapsto [\![h[x := v]]\!]). \tag{5.5}$$

To see that the latter holds we appeal to condition (Σ comp):

$$\begin{aligned}(5.5) &\impliedby \forall v \in \mathcal{V}_\circ. \, [\![g[x := v]]\!]_n \, \Gamma \preceq^H_{\mathcal{V}} [\![h[x := v]]\!] \\ &\quad \text{[By (Σ comp)]} \\ &\impliedby \vdash^{\Lambda} g[x := v] \preceq^H h[x := v] \\ &\quad \text{[By induction hypothesis]} \\ &\impliedby x \vdash^{\Lambda} g \preceq^H h. \\ &\quad \text{[By Lemma 15 (substitutivity and reflexivity)]}\end{aligned}$$

□

Theorem 4. *If Γ is Σ-continuous, then the open extension of effectful applicative similarity is a precongruence relation.*

Proof. We already know that $\preceq$ is a preorder, and thus $\preceq^o$ is a preorder too. To prove it is compatible (and thus a precongruence relation), it is sufficient to prove that $\preceq^H = \preceq^o$. By Lemma 15 we already know that the open extension of $\preceq$ is included in $\preceq^H$, so that it is sufficient to prove:

$$\Gamma \vdash^{\Lambda} e \preceq^H f \implies \Gamma \vdash^{\Lambda} e \preceq f.$$

For that, it is sufficient to show that $(\preceq^H)^c \subseteq \preceq$. In fact, if that is the case, then we have (where $\Gamma = \vec{x}$):

$$\begin{aligned}\Gamma \vdash^{\Lambda} e \preceq^H f &\implies \forall \vec{v} \in \mathcal{V}_\circ. \, \vdash^{\Lambda} e[\vec{x} := \vec{v}] \preceq^H f[\vec{x} := \vec{v}] \\ &\quad \text{[Since } \preceq^H \text{ is value-substitutive]} \\ &\implies \forall \vec{v} \in \mathcal{V}_\circ. \, \vdash^{\Lambda} e[\vec{x} := \vec{v}] \preceq f[\vec{x} := \vec{v}] \\ &\quad \text{[Since } (\preceq^H)^c \subseteq \preceq \text{]} \\ &\implies \forall \vec{v} \in \mathcal{V}_\circ. \, \vdash^{\Lambda} e \preceq f. \\ &\quad \text{[By definition of open extension]}\end{aligned}$$

Finally, to see that $(\preceq^H)^c$ is contained in $\preceq$ we proceed by coinduction, showing that $(\preceq^H)^c$ is an applicative simulation. The latter is exactly the content of the Key Lemma. □

An immediate corollary of Theorem 4 is that the open extension of applicative similarity is contained in contextual approximation (recall that $\preceq$ is adequate), meaning that it qualifies as a sound proof technique for the latter. The reverse inclusion, which would give *full abstraction* of $\preceq^o$ does not hold in general. A counterexample is given by $\Lambda_{\mathbb{FM}}$, where applicative similarity is well known to be strictly finer than contextual approximation (see Example 6.4.4 in (S. Lassen, 1998b)).

The next result we wish to prove is a congruence theorem for applicative bisimilarity. Since we already know applicative bisimilarity to be an equivalence relation, what we have to do is to prove that it is also compatible. It is not hard to realise that compatibility of $\simeq$ would easily follow if $\simeq$ coincides with $\preceq \cap \preceq^\circ$. This is the case in e.g. $\Lambda_{\mathbb{M}}$ and $\Lambda_{\mathbb{DM}}$, but it is not the case in $\Lambda_{\mathbb{FM}}$. Abstractly, the failure of such a property is reflected by the non-validity of the identity $\Gamma(\mathcal{R} \cap \mathcal{S}) = \Gamma\mathcal{R} \cap \Gamma\mathcal{S}$.

Applying Howe's construction to $\simeq$ also raises some problems. In fact, the Howe extension of a relation is built from relational refinement and postcomposition with the original relation, thus making the construction intrinsically asymmetric. As a consequence, $\mathcal{R}^H$ is in general not symmetric, meaning that there is little hope to prove the coincidence of $\simeq^H$ with $\simeq$. Howe (Howe, 1996) fixed this problem by observing that the transitive closure of $\simeq^H$ is indeed symmetric. The adaption of Howe's construction based on this observation goes under the name of *transitive closure trick* (S. Lassen, 1998b; Pitts, 2011).

5.4.1 The Transitive Closure Trick

In this section we adapt Howe's original formulation of the so-called transitive closure trick to prove a congruence theorem for applicative bisimilarity. Following (Howe, 1996), we begin by noticing that the transitive closure of the Howe extension of an equivalence λ-term relation is a compatible symmetric λ-term relation. Recall that the transitive closure $\mathcal{R}^T$ of a relation $\mathcal{R}$ is defined as follows:

$$\begin{aligned} \mathcal{R}^{(0)} &\triangleq \mathsf{I} \\ \mathcal{R}^{(n+1)} &\triangleq \mathcal{R}^{(n)} \cdot \mathcal{R} \\ \mathcal{R}^{T} &\triangleq \bigcup_n \mathcal{R}^{(n)}. \end{aligned}$$

Equivalently, we inductively define $\mathcal{R}^T$ as follows:

$$\frac{\Gamma \vdash^\Lambda e \,\mathcal{R}\, f}{\Gamma \vdash^\Lambda e \,\mathcal{R}^T\, f} \qquad \frac{\Gamma \vdash^\Lambda e \,\mathcal{R}\, g \quad \Gamma \vdash^\Lambda g \,\mathcal{R}^T\, f}{\Gamma \vdash^\Lambda e \,\mathcal{R}^T\, f} \qquad \frac{\Gamma \vdash^\mathsf{v} v \,\mathcal{R}\, w}{\Gamma \vdash^\mathsf{v} v \,\mathcal{R}^T\, w} \qquad \frac{\Gamma \vdash^\mathsf{v} v \,\mathcal{R}\, u \quad \Gamma \vdash^\mathsf{v} u \,\mathcal{R}^T\, w}{\Gamma \vdash^\mathsf{v} v \,\mathcal{R}^T\, w}$$

Lemma 17. *Let $\mathcal{R}$ be a reflexive and transitive λ-term relation. Then $(\mathcal{R}^H)^T$ is compatible. If additionally $\mathcal{R}$ is symmetric, then $(\mathcal{R}^H)^T$ is symmetric as well.*

Proof. We begin by showing that $(\mathcal{R}^H)^T$ is compatible, i.e. $\widehat{(\mathcal{R}^H)^T} \subseteq (\mathcal{R}^H)^T$. Since the composition of compatible relations is compatible, obviously $(\mathcal{R}^H)^T$ is compatible. However, since we did not give an explicit proof of the above fact, we sketch a proof of compatibility of $(\mathcal{R}^H)^T$ from first principles. The proof is by induction on the definition of $\widehat{(\mathcal{R}^H)^T}$. We show how to handle the case for sequencing (the remaining cases are proved in a similar, but easier way). Suppose to have:

$$\frac{\vdash^\Lambda e \,(\mathcal{R}^H)^T\, e' \qquad x \vdash^\Lambda f \,(\mathcal{R}^H)^T\, f'}{\vdash^\Lambda \mathbf{let}\ x = e\ \mathbf{in}\ f \;\widehat{(\mathcal{R}^H)^T}\; \mathbf{let}\ x = e'\ \mathbf{in}\ f'} \ (\text{comp-let})$$

We show $\vdash^\Lambda \mathbf{let}\ x = e\ \mathbf{in}\ f\ (\mathcal{R}^H)^T\ \mathbf{let}\ x = e'\ \mathbf{in}\ f'$. By very definition of transitive closure, there exist natural numbers n, m such that $\vdash^\Lambda e\ (\mathcal{R}^H)^{(n)}\ e'$ and $x \vdash^\Lambda f\ (\mathcal{R}^H)^{(m)}\ f'$ hold. Since $\mathcal{R}^H$ is reflexive and

transitive, without loss of generality we can assume $n = m$. In fact, if e.g. $n = m + l$, then we can 'complete' $(\mathcal{R}^H)^{(m)}$ as follows:

$$(\mathcal{R}^H)^{(m)} = (\mathcal{R}^H)^{(m)} \underbrace{\cdot \mathsf{I} \cdot \cdots \cdot \mathsf{I}}_{l\text{-times}} \subseteq (\mathcal{R}^H)^{(m)} \underbrace{\cdot \mathcal{R}^H \cdot \cdots \cdot \mathcal{R}^H}_{l\text{-times}} = (\mathcal{R}^H)^{(n)}.$$

We now prove the thesis by induction on n. The base case is trivial. Let us now prove the case for $n+1$. By very definition of transitive closure, we obtain the existence of computations e'' and f'' such that the following hold:

$$\vdash^\Lambda e \, \mathcal{R}^H \, e'' \qquad\qquad x \vdash^\Lambda f \, \mathcal{R}^H \, f''$$
$$\vdash^\Lambda e'' \, (\mathcal{R}^H)^{(n)} \, e' \qquad\qquad x \vdash^\Lambda f'' \, (\mathcal{R}^H)^{(n)} \, f'.$$

We apply the induction hypothesis on $\vdash^\Lambda e'' \, (\mathcal{R}^H)^{(n)} \, e'$ and $x \vdash^\Lambda f'' \, (\mathcal{R}^H)^{(n)} \, f'$, obtaining $\vdash^\Lambda \mathbf{let}\ x = e''\ \mathbf{in}\ f'' \, (\mathcal{R}^H)^T \, \mathbf{let}\ x = e'\ \mathbf{in}\ f'$. Moreover, since $\mathcal{R}^H$ is compatible, from $\vdash^\Lambda e \, \mathcal{R}^H \, e''$ and $x \vdash^\Lambda f \, \mathcal{R}^H \, f''$ we infer $\vdash^\Lambda \mathbf{let}\ x = e\ \mathbf{in}\ f \, \mathcal{R}^H \, \mathbf{let}\ x = e''\ \mathbf{in}\ f''$. We thus have

$$\vdash^\Lambda \mathbf{let}\ x = e\ \mathbf{in}\ f \, ((\mathcal{R}^H)^T \cdot \mathcal{R}^H) \, \mathbf{let}\ x = e'\ \mathbf{in}\ f',$$

from which the thesis follows, since $(\mathcal{R}^H)^T \cdot \mathcal{R}^H = (\mathcal{R}^H)^T$.

We now show that if $\mathcal{R}$ is an equivalence λ-term relation, then $(\mathcal{R}^H)^T$ is symmetric. That amounts to show $(\mathcal{R}^H)^T \subseteq ((\mathcal{R}^H)^T)^\circ$, which in turn is a direct consequence of $\mathcal{R}^H \subseteq ((\mathcal{R}^H)^T)^\circ$. Concretely, we simultaneously prove by induction on their premise the following implications:

$$\Gamma \vdash^\Lambda e \, \mathcal{R}^H \, f \implies \Gamma \vdash^\Lambda f \, (\mathcal{R}^H)^T \, e$$
$$\Gamma \vdash^\mathsf{v} v \, \mathcal{R}^H \, w \implies \Gamma \vdash^\mathsf{v} w \, (\mathcal{R}^H)^T \, v.$$

The proof is straightforward, and thus we show how to handle a couple of cases as illustrative examples only.

- Suppose $\Gamma \vdash^\mathsf{v} v \mathcal{R}^H w$ is the conclusion of an instance of rule (H-var), so that we have the following derivation:

$$\frac{\Gamma', x \vdash^\mathsf{v} x \, \mathcal{R} \, w}{\Gamma', x \vdash^\mathsf{v} x \, \mathcal{R}^H \, w} \ \text{(H-var)}$$

  Since $\mathcal{R}$ is symmetric (and thus so is its open extension), $\Gamma', x \vdash^\mathsf{v} x \, \mathcal{R} \, w$ implies $\Gamma', x \vdash^\mathsf{v} w \, \mathcal{R} \, x$. By Lemma 15 we know that the open extension of $\mathcal{R}$ is contained in $\mathcal{R}^H$, and thus we conclude $\Gamma', x \vdash^\mathsf{v} w \, \mathcal{R}^H \, x$.

- Suppose $\Gamma \vdash^\mathsf{v} e \mathcal{R}^H f$ is the conclusion of an instance of rule (H-let), so that we have the following derivation:

$$\frac{\Gamma \vdash^\Lambda e' \, \mathcal{R}^H \, e'' \quad \Gamma, x \vdash^\Lambda f' \, \mathcal{R}^H \, f'' \quad \Gamma \vdash^\Lambda \mathbf{let}\ x = e''\ \mathbf{in}\ f'' \, \mathcal{R} \, f}{\Gamma \vdash^\Lambda \mathbf{let}\ x = e'\ \mathbf{in}\ f' \, \mathcal{R}^H \, f} \ \text{(H-let)}$$

  Since $\mathcal{R}$ is symmetric (and thus so is its open extension), $\Gamma \vdash^\Lambda \mathbf{let}\ x = e''\ \mathbf{in}\ f'' \, \mathcal{R} \, f$ implies $\Gamma \vdash^\Lambda f \, \mathcal{R} \, \mathbf{let}\ x = e''\ \mathbf{in}\ f''$, and thus $\Gamma \vdash^\Lambda f \, \mathcal{R}^H \, \mathbf{let}\ x = e''\ \mathbf{in}\ f''$, by Lemma 15. Moreover, we can apply the induction hypothesis on $\Gamma \vdash^\Lambda e' \, \mathcal{R}^H \, e''$ and $\Gamma, x \vdash^\Lambda f' \, \mathcal{R}^H \, f''$, obtaining $\Gamma \vdash^\Lambda e'' \, (\mathcal{R}^H)^T \, e'$ and $\Gamma, x \vdash^\Lambda f'' \, (\mathcal{R}^H)^T \, f'$. By compatibility of $(\mathcal{R}^H)^T$ we obtain

$$\Gamma \vdash^\Lambda \mathbf{let}\ x = e''\ \mathbf{in}\ e'' \, (\mathcal{R}^H)^T \, \mathbf{let}\ x = e'\ \mathbf{in}\ f''.$$

  We conclude $\Gamma \vdash^\Lambda f \, ((\mathcal{R}^H)^T \cdot \mathcal{R}^H) \, \mathbf{let}\ x = e'\ \mathbf{in}\ f'$, and thus the desired thesis.

□

We can now state and prove our congruence theorem for applicative bisimilarity. Before doing so, let us remark that our proof of the Key Lemma does not depend on applicative similarity being the *largest* applicative simulation. Rather, it makes use only of $\preceq$ being reflexive, transitive, and an applicative simulation. As a consequence, we can generalise the Key Lemma over all reflexive and transitive applicative simulation.

Theorem 5. *If Γ is Σ-continuous, then the open extension of effectful applicative bisimilarity is a congruence relation.*

Proof. We already know $\simeq$ to be an equivalence relation (and thus $\simeq^o$ is). We prove that $\simeq^o$ is compatible by showing that $\simeq$ coincides with the restriction of $(\simeq^H)^T$ on closed terms. As for Theorem 4, from that follows $\preceq^o = (\simeq^H)^T$, noticing that if a λ-term relation $\mathcal{R}$ is value substitutive, then so is $\mathcal{R}^T$.
Let us show $\simeq \; = \; ((\simeq^H)^T)^c$. Since $\simeq^o \subseteq \simeq^H \subseteq (\simeq^H)^T$, it is sufficient to show that the restriction of $(\simeq^H)^T$ on closed terms is contained in $\simeq$. We prove that by coinduction, relying on the characterisation of $\simeq$ as the largest symmetric applicative simulation. As a consequence, we have to show that the restriction of $(\simeq^H)^T$ on closed terms is a symmetric applicative simulation (with respect to Γ). Symmetry follows by Lemma 17. Moreover, since $\simeq$ is a reflexive and transitive applicative simulation, by Key Lemma it follows that $\simeq^H$ is an applicative simulation too. Notice that the characterisation of $\simeq$ in terms of symmetric applicative simulations is central here. In fact, working with $\simeq$ defined as applicative similarity with respect to $\Gamma \wedge \Gamma^\circ$ does not allow us to apply the Key Lemma, since $\Gamma \wedge \Gamma^\circ$ is in general not inductive. Finally, since the composition of applicative simulations is an applicative simulation, we see that $(\simeq^H)^T$ is an applicative simulation too, and thus conclude that $\simeq$ coincides with the restriction of $(\simeq^H)^T$ on closed terms. Since by Lemma 17 the latter is compatible, $\simeq$ is compatible too, and thus a congruence relation. □

It is straightforward to see that applicative bisimilarity is adequate, with adequacy being defined by the predicate $\mathrm{Adeq}_{\Gamma \wedge \Gamma^\circ}$. Therefore, Theorem 5 implies that $\simeq$ is included in effectful contextual equivalence, and thus that it provides a sound proof technique for the latter. As for applicative similarity, the opposite inclusion does not hold in general (again, see (S. Lassen, 1998b)), meaning that effectful applicative bisimilarity is not fully abstract for effectful contextual equivalence.

5.5 CIU Approximation and Equivalence

In previous section we showed that effectful applicative similarity (resp. bisimilarity) is sound for effectful contextual approximation (resp. equivalence), but not fully abstract. Following (Pitts, 2011), here we show how to apply Howe's method as developed in Section 12.2 to prove a generalisation of Mason-Talcott *CIU Theorem* (Mason & Talcott, 1991), thus providing a handier characterisation of contextual approximation (resp. equivalence).

Roughly speaking, CIU-like theorems are a family of context lemmas (Milner, 1977) stating that two programs are contextually equivalent if no *evaluation* context can distinguish them. The acronymous *CIU* stands for *Closed Instantiation of Use*, since the original theorem in (Mason & Talcott, 1991) states that two terms are contextually equivalent if any closed instantiation of a use of the first is indistinguishable from the closed instantiation of a use of the second. Therefore, a *use* is simply an evaluation context, i.e. an environment that uses its input program. We will say more about ideas behind CIU approximation and equivalence in the next chapter, where such notions will be central themes.

Recall from Section 3.2 that Λ_Σ evaluation contexts are defined by the grammar:

$$E \; ::= \; [-] \mid \mathbf{let}\; x = E \;\mathbf{in}\; e.$$

The first requirement we have to make in order to define a meaningful notion of CIU approximation (and thus equivalence) is that whenever a term e CIU-approximates a term f, then no use, i.e. no evaluation context, can produce an observable behaviour using e that it is not able to produce using f. As for contextual approximation, we model approximation under (ground) observations (such us convergence, may convergence, or probability of convergence), as the relation $\Gamma 0_{\mathcal{V}}$. Accordingly, we said that a λ-term relation $\mathcal{R}$ is adequate with respect to Γ if $\mathcal{R} \in \mathsf{Adeq}_{\Gamma}$. Therefore, we require CIU approximation $\preceq^{\mathrm{ciu}}$ to satisfy the following property:

$$e \preceq^{\mathrm{ciu}}_{\Lambda} f \iff \forall E.\ [\![E[e]]\!]\ \Gamma 0_{\mathcal{V}}\ [\![E[f]]\!].$$

That, however, does not take into account values, as evaluation contexts take computations as input and produce new computations. As a consequence, we give the following definition inspired by (P. B. Levy, 2007).

Definition 37. *Let Γ be a relator. Define* effectful CIU approximation $\preceq^{\mathrm{ciu}}$ *with respect to Γ as the largest closed λ-relation $\mathcal{R}$ such that:*

$$\vdash^{\Lambda} e\ \mathcal{R}\ f \implies [\![e]\!]\ \Gamma 0_{\mathcal{V}}\ [\![e']\!] \tag{ciu 1}$$

$$\vdash^{\Lambda} e\ \mathcal{R}\ e' \implies \forall f \in \Lambda_x.\ \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f\ \mathcal{R}\ \mathbf{let}\ x = e'\ \mathbf{in}\ f. \tag{ciu 2}$$

$$\vdash^{\mathsf{v}} v\ \mathcal{R}\ v' \implies \forall e \in \Lambda_x.\ \vdash^{\Lambda} e[x := v]\ \mathcal{R}\ e[x := v']. \tag{ciu 3}$$

Definition 37 indeed defines a λ-term relation, since $\bigcup\{\mathcal{R} \mid \mathcal{R}$ satisfies (ciu 1), (ciu 2), (ciu 3)$\}$ itself satisfies (ciu 1), (ciu 2), and (ciu 3). Finally, we define effectful CIU equivalence $\simeq^{\mathrm{ciu}}$ with respect to Γ as $\preceq^{\mathrm{ciu}} \cap (\preceq^{\mathrm{ciu}})^{\circ}$.

As usual, we apply the notation and terminological conventions of Remark 9. Accordingly, we refer to effectful CIU approximation (resp. equivalence) with respect to Γ simply as CIU approximation (resp. equivalence). It is a straightforward exercise to verify that $\preceq^{\mathrm{ciu}}$ is a preorder (and $\simeq^{\mathrm{ciu}}$ an equivalence) relation.

Lemma 18. *CIU approximation $\preceq^{\mathrm{ciu}}$ is a reflexive and transitive λ-term relation, whereas CIU equivalence $\simeq^{\mathrm{ciu}}$ is a reflexive, symmetric, and transitive λ-term relation.*

We also notice that (the closed restriction of) $\preceq^{\mathrm{ctx}}$ satisfies conditions (ciu 1), (ciu 2), and (ciu 3), meaning that $(\preceq^{\mathrm{ctx}})^{c} \subseteq \preceq^{\mathrm{ciu}}$ (similarity, one easily sees that $(\simeq^{\mathrm{ctx}})^{c}$ is included in $\simeq^{\mathrm{ciu}}$). Actually, proving that $(\preceq^{\mathrm{ctx}})^{c}$ satisfies condition (ciu 3) requires to show substitutivity of $\preceq^{\mathrm{ctx}}$. Although we did not give any proof of that, the reader should notice that we can modify the theory of Section 5.2 working with the complete lattice of compatible and *substitutive* relations, rather than with the complete lattice of compatible relations only. In particular, an analogue of Lemma 12 can be easily proved for the space of compatible and substitutive relations, meaning that there exists a largest compatible and substitutive adequate λ-term relation. This change does not affect our soundness results, as effectful applicative (bi)similarity, as well as the other notions of equivalence and refinement studied in the next chapters, are substitutive.

Another, more direct proof of substitutivity of $\preceq^{\mathrm{ctx}}$ is obtained by showing that the (generalised) β-equivalence relation $=_{\beta}$, defined by:

$$\frac{\Gamma, x \vdash^{\Lambda} e \quad \Gamma \vdash^{\mathsf{v}} v}{\Gamma \vdash^{\Lambda} (\lambda x.e)v =_{\beta} e[x := v]}$$

is valid for $\preceq^{\mathrm{ctx}}$. An elegant way to prove such a result is by showing that the open extension of *effectful Kleene approximation*, defined by

$$\vdash^{\Lambda} e \preceq^{\mathrm{kle}} f \overset{\triangle}{\iff} [\![e]\!]\ \Gamma 0_{\mathcal{V}}\ [\![f]\!],$$

is contained in $\preceq^{\text{ctx}}$. We do not prove such a result here (although that can be easily done). The reader can consult e.g. Section 4.3 and Section 6.3 in (S. Lassen, 1998b) for details (although instantiated for $\Lambda_{\mathbb{M}}$ and $\Lambda_{\mathbb{FM}}$ only).

Lemma 19. *Effectful contextual approximation is contained in the open extension of CIU approximation.*

Proof. We have to show

$$\Gamma \vdash^{\Lambda} e \preceq^{\text{ctx}} f \implies \Gamma \vdash^{\Lambda} e \preceq^{\text{ciu}} f.$$

Let $\Gamma = \vec{x}$. Then we have:

$$\begin{aligned}
\Gamma \vdash^{\Lambda} e \preceq^{\text{ctx}} f &\implies \forall \vec{v} \in \mathcal{V}_{\circ}.\ \vdash^{\Lambda} e[\vec{x} := \vec{v}] \preceq^{\text{ctx}} f[\vec{x} := \vec{v}] \\
&\quad \text{[Since } \preceq^{\text{ctx}} \text{ is value-substitutive]} \\
&\implies \forall \vec{v} \in \mathcal{V}_{\circ}.\ \vdash^{\Lambda} e[\vec{x} := \vec{v}] \preceq^{\text{ciu}} f[\vec{x} := \vec{v}] \\
&\quad \text{[Since } (\preceq^{\text{ctx}})^{c} \subseteq \preceq^{\text{ciu}} \text{]} \\
&\implies \forall \vec{v} \in \mathcal{V}_{\circ}.\ \vdash^{\Lambda} e \preceq^{\text{ciu}} f. \\
&\quad \text{[By definition of open extension]}
\end{aligned}$$

□

Next, we wish to prove that actually $(\preceq^{\text{ciu}})^{o}$ provides an alternative characterisation of $\preceq^{\text{ctx}}$, and thus that it is fully abstract for it. For that it is sufficient to show compatibility of $(\preceq^{\text{ciu}})^{o}$. We prove such a compatibility result using Howe's method. In fact, by Lemma 18 $\preceq^{\text{ciu}}$ is a preorder, and thus by Lemma 15 we see that $(\preceq^{\text{ciu}})^{H}$ is reflexive, compatible, and substitutive open λ-term relation. Therefore, we wish to show that the closed restriction of $(\preceq^{\text{ciu}})^{H}$ is contained in $\preceq^{\text{ciu}}$. Mimicking the proof of Lemma 19, we see that we can then conclude $(\preceq^{\text{ciu}})^{o} = (\preceq^{\text{ciu}})^{H}$.

Let us show $((\preceq^{\text{ciu}})^{H})^{c} \subseteq \preceq^{\text{ciu}}$. Since $(\preceq^{\text{ciu}})^{H}$ is compatible and substitutive, $((\preceq^{\text{ciu}})^{H})^{c}$ obviously satisfies conditions (ciu 2) and (ciu 3). The next lemma, which plays the same role played by Lemma 16 for applicative similarity, shows that $(\preceq^{\text{ciu}})^{H}$ is adequate.

Lemma 20. *Let Γ be a Σ-continuous relator. If $\vdash^{\Lambda} e\ (\preceq^{\text{ciu}})^{H}\ f$, then $[\![e]\!]\ \Gamma 0_{\mathcal{V}}\ [\![f]\!]$.*

Proof. The proof goes as for Lemma 16. Since Γ is inductive, it is sufficient to prove that for any $n \geq 0$ we have:

$$\vdash^{\Lambda} e \preceq^{\text{ciu}} f \implies [\![e]\!]_n\ \Gamma 0_{\mathcal{V}}\ [\![f]\!].$$

The case for $n = 0$ is trivial, as Γ is inductive. For the inductive step, we proceed by case analysis on e.

- If e is of the form **return** v, then $\vdash^{\Lambda}$ **return** $v\ (\preceq^{\text{ciu}})^{H}\ f$ must be the conclusion of a derivation of the form:
$$\frac{\vdash^{\mathsf{v}} v\ (\preceq^{\text{ciu}})^{H}\ v' \quad \vdash^{\Lambda} \textbf{return}\ v' \preceq^{\text{ciu}} f}{\vdash^{\Lambda} \textbf{return}\ v\ (\preceq^{\text{ciu}})^{H}\ f}\ (\mathsf{H}\text{-val})$$
In particular, by condition (rel unit), $\vdash^{\mathsf{v}} v\ (\preceq^{\text{ciu}})^{H}\ v'$ implies $\eta(v)\ \Gamma(\preceq^{\text{ciu}}_{\mathcal{V}})^{H}\ \eta(v')$, and thus $\eta(v)\ \Gamma 0_{\mathcal{V}}\ \eta(v')$. Since $\vdash^{\Lambda}$ **return** $v' \preceq^{\text{ciu}} f$, we have
$$[\![\textbf{return}\ v]\!] = \eta(v)\ \Gamma 0_{\mathcal{V}}\ \eta(v') = [\![\textbf{return}\ v']\!]\ \Gamma 0_{\mathcal{V}}\ [\![f]\!],$$
from which we conclude the desired thesis.

- If $e = (\lambda x.e')v$, then $\vdash^\Lambda (\lambda x.e')v\ (\preceq^{\mathrm{ciu}})^H\ f$ must be the conclusion of a derivation of the form:

$$\dfrac{\dfrac{x \vdash^\Lambda e'\ (\preceq^{\mathrm{ciu}})^H\ g \quad \vdash^{\mathsf{v}} \lambda x.g \preceq^{\mathrm{ciu}} u}{\vdash^{\mathsf{v}} \lambda x.e'\ (\preceq^{\mathrm{ciu}})^H\ u}\ \text{(H-abs)} \quad \vdash^{\mathsf{v}} v\ (\preceq^{\mathrm{ciu}})^H\ v' \quad \vdash^\Lambda uv' \preceq^{\mathrm{ciu}} f}{\vdash^\Lambda (\lambda x.e')v\ (\preceq^{\mathrm{ciu}})^H\ f}\ \text{(H-app)}$$

From $x \vdash^\Lambda e'\ (\preceq^{\mathrm{ciu}})^H\ g$ and $\vdash^{\mathsf{v}} v\ (\preceq^{\mathrm{ciu}})^H\ v'$, by substitutivity, we infer $\vdash^\Lambda e'[x := v]\ (\preceq^{\mathrm{ciu}})^H\ g[x := v']$, and thus $[\![e'[x := v]]\!]_n\ \Gamma 0_{\mathcal{V}}\ [\![g[x := v']]\!]$. We now reason as follows. First of all we observe that we have the following closure properties for $\preceq^{\mathrm{ciu}}$.

$$\forall v, v'.\ \vdash^{\mathsf{v}} v \preceq^{\mathrm{ciu}} v' \implies \vdash^\Lambda \mathbf{return}\ v \preceq^{\mathrm{ciu}} \mathbf{return}\ v'$$
$$\forall v, v'.\ \vdash^{\mathsf{v}} v \preceq^{\mathrm{ciu}} v' \implies \forall w \in \mathcal{V}_\circ.\ \vdash^\Lambda \mathbf{let}\ x = (\mathbf{return}\ v)\ \mathbf{in}\ (xw) \preceq^{\mathrm{ciu}} \mathbf{let}\ x = (\mathbf{return}\ v')\ \mathbf{in}\ (xw).$$

In fact, the first implication follows by (ciu 3) taking the open computation $x \vdash^\Lambda \mathbf{return}\ x$, whereas the second implication follows from the first one and property (ciu 3) taking the open computation $y \vdash^\Lambda \mathbf{let}\ x = y\ \mathbf{in}\ xw$. As a consequence, we have:

$$\begin{aligned}
\vdash^{\mathsf{v}} \lambda x.g \preceq^{\mathrm{ciu}} u &\implies \vdash^\Lambda \mathbf{let}\ y = (\mathbf{return}\ \lambda x.g)\ \mathbf{in}\ (yv') \preceq^{\mathrm{ciu}} \mathbf{let}\ y = (\mathbf{return}\ u)\ \mathbf{in}\ (yv') \\
&\implies \vdash^\Lambda [\![\mathbf{let}\ y = (\mathbf{return}\ \lambda x.g)\ \mathbf{in}\ (yv')]\!]\ \Gamma 0_{\mathcal{V}}\ [\![\mathbf{let}\ y = (\mathbf{return}\ u)\ \mathbf{in}\ (yv')]\!] \\
&\quad \text{[By (ciu 1)]} \\
&\implies [\![g[x := v']]\!]\ \Gamma 0_{\mathcal{V}}\ [\![uv']\!] \\
&\implies [\![e'[x := v]]\!]_n\ \Gamma 0_{\mathcal{V}}\ [\![uv']\!] \\
&\quad \text{[Since } [\![e'[x := v]]\!]_n\ \Gamma 0_{\mathcal{V}}\ [\![g[x := v']]\!]\text{]} \\
&\implies [\![e'[x := v]]\!]_n\ \Gamma 0_{\mathcal{V}}\ [\![f]\!] \\
&\quad \text{[Since } \vdash^\Lambda uv' \preceq^{\mathrm{ciu}} f\text{]}.
\end{aligned}$$

We are done since $[\![(\lambda x.e')v)]\!]_{n+1} = [\![e'[x := v]]\!]_n$.

- If $e = (\mathbf{let}\ x = g\ \mathbf{in}\ g')$, then the thesis directly follows from condition (rel bind) as in Lemma 16. In fact $\vdash^\Lambda \mathbf{let}\ x = g\ \mathbf{in}\ g'\ (\preceq^{\mathrm{ciu}})^H\ f$ must be the conclusion of a derivation of the form:

$$\dfrac{\vdash^\Lambda g\ (\preceq^{\mathrm{ciu}})^H\ h \quad x \vdash^\Lambda g'\ (\preceq^{\mathrm{ciu}})^H\ h' \quad \vdash^\Lambda \mathbf{let}\ x = h'\ \mathbf{in}\ h' \preceq^{\mathrm{ciu}} f}{\vdash^\Lambda \mathbf{let}\ x = g\ \mathbf{in}\ g'\ (\preceq^{\mathrm{ciu}})^H\ f}\ \text{(H-let)}$$

By induction hypothesis, from $\vdash^\Lambda g\ (\preceq^{\mathrm{ciu}})^H\ h$ we infer $[\![g]\!]_n\ \Gamma 0_{\mathcal{V}}\ [\![h]\!]$. Moreover, using substitutivity of $(\preceq^{\mathrm{ciu}})^H$ and the induction hypothesis, we see that $x \vdash^\Lambda g'\ (\preceq^{\mathrm{ciu}})^H\ h'$ implies:

$$\forall v, v' \in \mathcal{V}_\circ.\ \vdash^{\mathsf{v}} v\ (\preceq^{\mathrm{ciu}})^H\ v' \implies [\![g'[x := v]]\!]_n\ \Gamma 0_{\mathcal{V}}\ [\![h'[x := v']]\!].$$

By condition (rel bind) we thus conclude

$$[\![\mathbf{let}\ x = g\ \mathbf{in}\ g']\!]_{n+1} = [\![g'[x := -]]\!]^\dagger_n [\![g]\!]_n\ \Gamma 0_{\mathcal{V}}\ [\![h'[x := -]]\!]^\dagger [\![h]\!] = [\![\mathbf{let}\ x = h\ \mathbf{in}\ h']\!]$$

and thus the wished thesis, since $\mathbf{let}\ x = h'\ \mathbf{in}\ h' \preceq^{\mathrm{ciu}} f$.

- If $e = \mathbf{op}(p, x.g)$, then $\vdash^\Lambda \mathbf{op}(p, g)\ (\preceq^{\mathrm{ciu}})^H\ f$ must be the conclusion of a derivation of the form:

$$\dfrac{x \vdash^\Lambda g\ (\preceq^{\mathrm{ciu}})^H\ g' \quad \vdash^\Lambda \mathbf{op}(p, x.g') \preceq^{\mathrm{ciu}} f}{\vdash^\Lambda \mathbf{op}(p, x.g)\ (\preceq^{\mathrm{ciu}})^H\ f}\ \text{(H-op)}$$

We have:

$$
\begin{aligned}
x \vdash^{\Lambda} g\ (\preceq^{\text{ciu}})^{H}\ g' &\implies \forall v \in \mathcal{V}_{\circ}.\ \vdash^{\Lambda} g[x := v]\ (\preceq^{\text{ciu}})^{H}\ g'[x := v] \\
&\quad \text{[By substitutivity and reflexivity of } (\preceq^{\text{ciu}})^{H}\text{]} \\
&\implies \forall v \in \mathcal{V}_{\circ}.\ [\![g[x := v]]\!]_{n}\ \Gamma 0_{\mathcal{V}}\ [\![g'[x := v]]\!] \\
&\quad \text{[By induction hypothesis]} \\
&\implies [\![\mathbf{op}]\!](p, v \mapsto [\![g[x := v]]\!]_{n})\ \Gamma 0_{\mathcal{V}}\ [\![\mathbf{op}]\!](p, v \mapsto [\![g'[x := v]]\!]) \\
&\quad \text{[By } (\Sigma \text{ comp})\text{]} \\
&\implies [\![\mathbf{op}(p, x.g)]\!]_{n+1}\ \Gamma 0_{\mathcal{V}}\ [\![\mathbf{op}(p, x.g')]\!] \\
&\implies [\![\mathbf{op}(p, x.g)]\!]_{n+1}\ \Gamma 0_{\mathcal{V}}\ [\![f]\!] \\
&\quad \text{[Since } \vdash^{\Lambda} \mathbf{op}(p, x.g') \preceq^{\text{ciu}} f\text{]}
\end{aligned}
$$

□

Theorem 6. *(The open extension of) effectful CIU approximation and effectful CIU equivalence coincide with effectful contextual approximation and effectful contextual equivalence, respectively.*

Proof. By Lemma 20 (and previous discussion) $(\preceq^{\text{ciu}})^{H}$ restricted to closed values is contained in $\preceq^{\text{ciu}}$, and thus $(\preceq^{\text{ciu}})^{H}$ coincides with $(\preceq^{\text{ciu}})^{o}$. Therefore $(\preceq^{\text{ciu}})^{o}$ is compatible, substitutive, and preadequate, and thus contained in $\preceq^{\text{ctx}}$. By Lemma 19 $\preceq^{\text{ctx}}$ is included in $(\preceq^{\text{ciu}})^{o}$, and thus we have $(\preceq^{\text{ciu}})^{o} = \preceq^{\text{ctx}}$. Finally, by Proposition 15 we have $(\simeq^{\text{ciu}})^{o} = \simeq^{\text{ctx}}$ too. □

Chapter 6

Monadic Applicative Similarity and Bisimilarity

> Language is the house of the truth of Being.
>
> Martin Heidegger, Letter on Humanism

In Chapter 5 we have defined effectful applicative similarity and bisimilarity for Λ_Σ, a *call-by-value* calculus with algebraic effects. It is not hard to see that the theory developed there can be rephrased in a call-by-name setting with minimal efforts. Theorem 4 and Theorem 5 show that effectful applicative similarity and bisimilarity are sound proof techniques for effectful contextual approximation and equivalence, respectively, although they are not fully abstract. For instance, when instantiated with the nondeterministic monad $\mathbb{FM}$, effectful applicative similarity can be shown to be strictly finer than contextual approximation (S. Lassen, 1998b).

In a call-by-name setting similar results hold. However, due to the simpler nature of call-by-name calculi, whose programs can only test their inputs in functional position, it is possible to characterise effectful contextual equivalence and approximation coinductively using a notion of effectful applicative-like (bi)similarity. The definition and analysis of such a notion of (bi)similarity, which we call *monadic applicative (bi)similarity*, constitute the main subject of this chapter.

Monadic applicative (bi)similarity builds on the idea of defining program equivalences not (directly) on programs, but on their semantics (i.e. on monadic values), which is something simply unsound in a call-by-value setting, as we will see. This way, the interaction between a program and the environment can be modeled by an ordinary, deterministic, transition system and, with minimal side conditions, the resulting notion of (bi)similarity can be proved to be *sound* and *fully abstract* with respect to contextual approximation and equivalence.

Before entering technicalities, we outline the main ideas behind this chapter by means of a semi-formal discussion. Before that, however, it is convenient to introduce the calculus used in this chapter in full generality.

6.1 A Computational Call-by-name Calculus

First of all we notice that in a call-by-name setting, the distinction between values and computations has a minimal importance. For that reason we will work with the *coarse-grain* version of Λ_Σ, denoted

by $\Lambda_\Sigma^{(n)}$, whose syntax is defined by the following grammar:

$$\begin{aligned} v, w &::= x \mid \lambda x.e \\ e, f &::= v \mid ef \mid \mathbf{op}(p, x.e) \end{aligned}$$

All syntactical and notational conventions are the same given for Λ_Σ[1], with the exception that in $\Lambda_\Sigma^{(n)}$ we allow the substitution of a (generic) term inside another (generic) term: we write $f[x := e]$ for the term obtained by simultaneous substitution of all free occurrences of x in f with e. Since in this chapter we will work with $\Lambda_\Sigma^{(n)}$ only, for readability we will omit superscripts, and simply refer to it as Λ_Σ.

Operational semantics of Λ_Σ is defined with respect to a Σ-continuous monad $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ which, from now on, we assume to be fixed. Additionally, it is convenient to define the operational semantics of Λ_Σ relying on Felleisen's notion of a call-by-name evaluation context (Felleisen & Hieb, 1992). Call-by-name evaluation contexts are terms with a single hole $[-]$ defined by the following grammar, where e is a closed term:

$$E, F ::= [-] \mid Ee.$$

We write $E[e]$ for the term obtained by substituting the term e for the hole $[-]$ in E. Notice that a call-by-name evaluation context (evaluation context, hereafter) is just an expression of the form $[-]e_1 \cdots e_n$.

Redexes are expressions of the form $(\lambda x.f)e$ or $\mathbf{op}(p, x.e)$, the former producing a computation step, the latter producing the effect described by the operation **op**. We notice that any term is either a value or an expression of the form $E[r]$, for a redex r. More explicitly, any term e is either a value v or can be uniquely decomposed as either $E[(\lambda x.f)e]$ or $E[\mathbf{op}(p, x.e)]$ (see also Lemma 25 and Lemma 29).

As usual, operational semantics is defined by means of an evaluation function $[\![-]\!] : \Lambda_\circ \to T\mathcal{V}_\circ$ mapping each closed term $e \in \Lambda_\circ$ to a monadic values $[\![e]\!] \in T\mathcal{V}_\circ$.

Definition 38. *Define the $\mathbb{N}$-indexed family of maps $[\![-]\!]_n : \Lambda_\circ \to T\mathcal{V}_\circ$ as follows:*

$$\begin{aligned} [\![e]\!]_0 &\triangleq \bot \\ [\![v]\!]_{n+1} &\triangleq \eta(v) \\ [\![E[(\lambda x.f)e]]\!]_{n+1} &\triangleq [\![E[f[x := e]]]\!]_n \\ [\![E[\mathbf{op}(p, x.e)]]\!]_{n+1} &\triangleq [\![\mathbf{op}]\!](p, v \mapsto [\![E[e[x := v]]]\!]_n) \end{aligned}$$

The sequence $([\![e]\!]_n)_n$ forms an ω-chain in $T\mathcal{V}_\circ$, so that we can define $[\![e]\!]$ as $\bigsqcup_n [\![e]\!]_n$.

As usual, monotonicity and continuity of operations $[\![\mathbf{op}]\!]$ and Kleisli extension ensure the evaluation function of Definition 38 to be the least map $\varphi : \Lambda_\circ \to T\mathcal{V}_\circ$ such that the following identities:

$$\begin{aligned} \varphi(v) &= \eta(v) \\ \varphi(E[(\lambda x.e)f]) &= \varphi(E[e[f := x]]) \\ \varphi(E[\mathbf{op}(p, x.e)]) &\triangleq [\![\mathbf{op}]\!](p, v \mapsto \varphi(E[e[x := v]])). \end{aligned}$$

Finally, we observe that we can rephrase the notion of an effectful applicative (bi)simulation to the call-by-name setting. Even if we do not syntactically distinguish between values and computations, it is still useful to work with (closed) λ-term relations.

[1] In particular, Λ denotes the collection of all terms, whereas $\mathcal{V} \subseteq \Lambda$ denotes the collection of all values. Closed terms and closed values are denoted by $\Lambda_\circ$ and $\mathcal{V}_\circ$, respectively.

Definition 39. *A closed λ-term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ is an effectful applicative simulation with respect to a relator Γ for $\mathbb{T}$ (applicative simulation, for short) if:*

$$e\,\mathcal{R}_\Lambda\,f \implies [\![e]\!]\,\Gamma\mathcal{R}_\mathcal{V}\,[\![f]\!]$$
$$v\,\mathcal{R}_\mathcal{V}\,w \implies \forall e \in \Lambda_\circ.\ ve\,\mathcal{R}_\Lambda\,we.$$

Notice that we do not require $\mathcal{R}_\mathcal{V} \subseteq \mathcal{R}_\Lambda$. However, by condition (rel unit) we can assume that to be the case without loss of generality. In fact, any applicative simulation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ can be extended to $(\mathcal{R}_\Lambda \cup \mathcal{R}_\mathcal{V}, \mathcal{R}_\mathcal{V})$, and the latter is still an applicative simulation (notice that $v\mathcal{R}_\mathcal{V}\,v'$ implies $\eta(v)\,\Gamma\mathcal{R}_\mathcal{V}\,\eta(v')$, by condition (rel unit)).

The notions of effectful applicative bisimulation, effectful applicative similarity, and effectful applicative bisimilarity are defined building on Definition 39 as in Chapter 5. In particular, we denote effectful applicative similarity and effectful applicative bisimilarity by $\preceq^{\mathsf{A}}$ and $\simeq^{\mathsf{A}}$, respectively.

6.2 Lift Transition Systems, Not Relations

To illustrate the rationale behind *monadic applicative (bi)similarity*, we first of all give a more explicit characterisation of effectful contextual equivalence and approximation. In fact, although in Chapter 5 we gave an elegant definition of effectful contextual approximation and equivalence (with respect to a relator Γ), here it is convenient to be a bit sloppy. We notice that, intuitively, e contextually approximates f if and only for any context C, $[\![C[e]]\!]\Gamma 0_\mathcal{V}[\![C[f]]\!]$. Here a context is a syntax tree with a unique hole defined by the following grammar:

$$C ::= [-] \mid \lambda x.C \mid eC \mid Ce \mid \mathbf{op}(p, x.C).$$

As usual, we write $C[e]$ for the term obtained by the replacing the hole $[-]$ with e in C. We say that a context C *closes* e, if $C[e]$ is a closed term. We denote by $\mathsf{Cl}(e)$ the collection of context that closes e.

Remark 10. Contrary to terms, contexts cannot be identified modulo renaming of bound variables. In fact, given a context C, free variables of a term e may become bound in $C[e]$ (e.g. consider the context $\lambda x.[-]$ and the term x). Equating contexts that differ only for the name of their bound variables can lead to undesired behaviours. For instance, the two contexts $\lambda x.[-]$ and $\lambda y.[-]$ are equal modulo renaming of bound variables, but give different terms when filled in with e.g. the term x.

As a consequence, working with contexts usually involves to deal with a lot of 'syntactic bureaucracy', which is the reason why in Chapter 5 we gave a 'context-free' definition of contextual equivalence and approximation. The reader can refer to (S. Lassen, 1998b; Pitts, 2011) for further details.

We can then rephrase the notion of effectful contextual approximation $\preceq^{\mathsf{ctx}}$ with respect to a relator Γ as follows:

$$e \preceq^{\mathsf{ctx}} f \overset{\triangle}{\iff} \forall C \in \mathsf{Cl}(e) \cap \mathsf{Cl}(f).\ [\![C[e]]\!]\,\Gamma 0_\mathcal{V}\,[\![C[f]]\!].$$

It is straightforward to see that by slightly modifying proofs of Chapter 5 we can prove soundness of (the open extension of) $\preceq^{\mathsf{A}}$ and $\simeq^{\mathsf{A}}$ with respect to $\preceq^{\mathsf{ctx}}$ and $\simeq^{\mathsf{ctx}}$ (the latter being defined as $\preceq^{\mathsf{ctx}} \cap (\preceq^{\mathsf{ctx}})^\circ$), respectively. That is, we have the inclusions:

$$(\preceq^{\mathsf{A}})^o \subseteq \preceq^{\mathsf{ctx}}$$
$$(\simeq^{\mathsf{A}})^o \subseteq \simeq^{\mathsf{ctx}}.$$

Moreover, the inclusion is strict. To see that let us consider the calculus $\Lambda_{\mathbb{FM}}$. It is not hard to convince ourselves that all terms of the form $e \triangleq \lambda x.(f\ \mathbf{or}\ g)$ and $e' \triangleq (\lambda x.f)\mathbf{or}(\lambda x.g)$ are contextually equivalent.

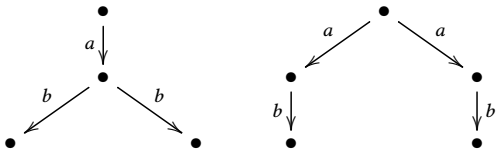
An informal argument for their equivalence is the following (a formal proof of their equivalence will be given later in full generality). Since in call-by-name calculi terms can only be tested in functional position, any context C will either not test e and e' at all, so that trivially no operational difference between $C[e]$ and $C[e']$ will be detected, or it will test them in functional position. That means that the evaluation of $C[e]$ and $C[e']$ will reach a state in which the terms eg and $e'g'$ have to be evaluated. Here g and g' are terms produced during the evaluation of $C[e]$ and $C[e']$ that may 'contain' other occurrences of e, e', respectively. In particular, we see that the only way for C to test e and e' is to test them against arbitrary input arguments. We thus reach the following conclusion: testing a program e in a context C is equivalent to testing e in functional position against finite sequences of inputs. This is nothing but the call-by-name version of the CIU Theorem in Section 5.5, which we can state as follows.

Theorem 7 (CIU)**.** *The following hold for all closed terms e, f:*

$$e \preceq^{\text{ctx}} f \iff \forall g_1, \dots, g_n.\ [\![eg_1 \cdots g_n]\!]\, \Gamma 0_{\mathcal{V}}\, [\![fg_1 \cdots g_n]\!].$$

Notice that Theorem 7 states that $\preceq^{\text{ctx}}$ (and thus and $\simeq^{\text{ctx}}$) is a form of *trace approximation*, and thus it is intrinsically *deterministic*. Using Theorem 7 we can now easily show that the terms e ans e' are indeed contextually equivalent. This comes with no surprise: contextual equivalence being a form of trace equivalence it is not sensitive to 'branching behaviours', and thus by no means it can tell e and e' apart.

This is not the case for applicative bisimilarity which, instead, is sensitive to forms of branching. From a bisimulation perspective, the difference between e and e' is clear: e postpones the nondeterministic choice to the moment it receives an input, whereas e' first makes the choice, and then waits to receive an input. Accordingly, it is sufficient to instantiate[2] e as $x\Omega I$ and e' as $xI\Omega$ to observe that e and e' cannot be applicatively bisimilar. In fact, the reader might have recognised that e and e' are nothing more than the encoding on $\Lambda_{\mathbb{FM}}$ of the labelled transition systems below, which are standard examples in concurrency theory to show that bisimilarity is strictly finer than trace equivalence.



Summing up, we observed that in a nondeterministic setting, applicative bisimilarity is sensitive to branching, whereas contextual approximation is not. This is because, in full generality, contextual approximation is a form of trace approximation, and thus intrinsically *deterministic*. This last observation suggests an easy way to make applicative similarity fully abstract with respect to contextual approximation, namely to *determinise* our notion of effectful applicative similarity.

Recall that in Chapter 2 we argued the the operational semantics of Λ_p induces a 'kind of' Markov chain. With a similar analogy, we can claim that the operational semantics of $\Lambda_{\mathbb{FM}}$ induces a 'kind of' transition system with divergence, where a labelled transition system with divergence consists of a set X (the state space) together with a transition function $c : X \to (FMX)^A$ (the set A being the set of labels or actions).

There are at least two natural notions of approximation for such systems. The first one is given by 'Milner style' similarity (Milner, 1989; Park, 1981), and can be defined using a suitable variation of the relator $\hat{\mathbb{FM}}$. The second one is given by trace approximation, and can be elegantly defined using the

[2] As usual, I denotes the identity combinator $\lambda x.x$ whereas Ω denotes the purely divergent term $(\lambda x.xx)(\lambda x.xx)$.

so-called powerset construction from automata theory[3] (Rabin & Scott, 1959). Roughly speaking, the powerset construction lifts a transition system $c : X \to (FMX)^A$ to a *deterministic* transition system $c' : FMX \to (FMX)^A$, whose state space is now (morally) given by sets of elements of the original state space. Writing Y for FMX, we see that $c' : Y \to Y^A$ is a *deterministic* labelled transition system, on which a notion of similarity can be easily defined. Remarkably, similarity in $c' : Y \to Y^A$ coincides with trace approximation in $c : X \to (FMX)^A$.

Taking inspiration from the powerset construction we aim to define notions of applicative similarity and bisimilarity on monadic values, rather than on terms, so to obtain a coinductive characterisation of contextual approximation and equivalence.

Remark 11 (The labelled transition system of λ-terms)**.** We already claim twice that the operational semantics of the calculi considered so far induce 'kind of' transition systems. We now make such intuition a formal observation, defining the labelled transition system (lts, hereafter) of λ-terms. Although we will not work with the lts of λ-term explicitly, the reader might find such a notion useful for intuitions and informal arguments.

The lts of λ-terms is a generalisation of Abramsky's applicative transition systems (Abramsky, 1990a), and it is defined as the pair $\langle \mathsf{eval} : \Lambda_\circ \to T\mathcal{V}_\circ, \mathsf{apply} : \mathcal{V}_\circ \to (\Lambda_\circ)^{\Lambda_\circ} \rangle$, where the evaluation function eval and the application function apply are defined as:

$$\mathsf{eval}(e) \triangleq [\![e]\!]$$
$$\mathsf{apply}(v, e) \triangleq ve.$$

Looking at the lts of λ-terms we see how the key ingredient to define the notion of an effectful applicative (bi)simulation is a notion of lifting (a relator) that allows to lift relations over values to relations over monadic values. In fact, using the lts of λ-terms we see that the proof obligations of Definition 39 can be expressed by the following lax commutative diagrams (the right hand side diagrams is actually a family of diagrams, as e ranges over arbitrary closed terms):

$$\begin{array}{ccc} \Lambda_\circ & \xrightarrow{\mathsf{eval}} & T\mathcal{V}_\circ \\ {\scriptstyle \mathcal{R}_\Lambda}\downarrow & \subseteq & \downarrow{\scriptstyle \Gamma\mathcal{R}_\mathcal{V}} \\ \Lambda_\circ & \xrightarrow[\mathsf{eval}]{} & T\mathcal{V}_\circ \end{array} \qquad \begin{array}{ccc} \mathcal{V}_\circ & \xrightarrow{\mathsf{apply}(-,e)} & \Lambda_\circ \\ {\scriptstyle \mathcal{R}_\mathcal{V}}\downarrow & \subseteq & \downarrow{\scriptstyle \mathcal{R}_\Lambda} \\ \mathcal{V}_\circ & \xrightarrow[\mathsf{apply}(-,e)]{} & \Lambda_\circ \end{array}$$

However, the lts of λ-terms also suggests another route to define notions of program equivalence and approximation. Instead of focusing on lifting relations, we can use the strong monad structure of T to lift the whole lts to $\langle \mathsf{eval}' : T\Lambda_\circ \to T\mathcal{V}_\circ, \mathsf{apply}' : T\mathcal{V}_\circ \to (T\Lambda_\circ)^{\Lambda_\circ} \rangle$ where

$$\mathsf{eval}' \triangleq \mathsf{eval}^\dagger$$
$$\mathsf{apply}' \triangleq \mathsf{curry}(\eta_{\Lambda_\circ} \cdot \mathsf{uncurry}(\mathsf{apply}))^*.$$

Using the new lts we can define *monadic applicative simulation* as a pair of relations $(\mathcal{R} : T\Lambda_\circ \nrightarrow T\Lambda_\circ, \mathcal{S} : T\mathcal{V}_\circ \nrightarrow T\mathcal{V}_\circ)$ such that:

$$\mathscr{E}\ \mathcal{R}\ \mathscr{F} \implies \mathsf{eval}'(\mathscr{E})\ \Gamma 0_\mathcal{V}\ \mathsf{eval}'(\mathscr{F})$$
$$\mathscr{V}\ \mathcal{S}\ \mathscr{W} \implies \forall e \in \Lambda_\circ.\ \mathsf{apply}'(\mathscr{V}, e)\ \mathcal{R}\ \mathsf{apply}'(\mathscr{W}, e).$$

As usual, we define *monadic applicative similarity* as the largest applicative simulation. We will explain the rationale behind monadic applicative similarity in the next section. For now, it is sufficient to observe that although coinductively defined, this new notion of applicative similarity is a trace-like approximation.

[3] Such a construction has been extended to a large class of coalgebras in e.g. (Hasuo, Jacobs, & Sokolova, 2007; Jacobs, Silva, & Sokolova, 2012).

6.3 Monadic Applicative Similarity and Bisimilarity

In this section we define the core notions of this chapter, namely the notions of a monadic applicative simulation and bisimulation.

We start by noticing that in previous section, oftentimes we compared programs using the relation $\Gamma 0_{\mathcal{V}}$. In Chapter 5 we proved such a relation to induce an interesting adequacy predicate, as it captures the idea of ground observation indistinguishability. For instance, we see that $[\![e]\!]\ (\hat{\mathbb{D}\mathbb{M}} \wedge \hat{\mathbb{D}\mathbb{M}}^{\circ})0_{\mathcal{V}}\ [\![f]\!]$ holds if and only if e and f have the same probability of convergence. As a consequence, if we let our ground observation to be the probability of convergence of a program, then $[\![e]\!]\,(\hat{\mathbb{D}\mathbb{M}} \wedge \hat{\mathbb{D}\mathbb{M}}^{\circ})0_{\mathcal{V}}\,[\![f]\!]$ states exactly that no ground observation can distinguish e and f.

The drawback of this approach is that the notion of ground observation remains implicit. Even if it is possible to develop the theory of monadic applicative (bi)similarity using the relation $\Gamma 0_{\mathcal{V}}$ only, it is useful and instructive to introduce an explicit notion of ground observation, and to work with the latter.

Intuitively, a ground observation is a function that takes a monadic value (representing the result of an effectful computation) and returns what is observable of it. Following standard practice, we assume that the observable part of a computation consists of the side effects happened during such a computation only. As a consequence, the observable part of a computation is uniquely determined by the semantics of the (algebraic) operations involved.

Definition 40. *Let $\mathbb{1} = \{*\}$ be the one element set and $!_X : X \to \mathbb{1}$ be the unique function mapping each element in X to $*$. Define the observation function $obs : T\mathcal{V}_\circ \to T\mathbb{1}$ as $T(!_{\mathcal{V}_\circ})$. Observations trivially extends to terms by defining $obs(e) \triangleq obs([\![e]\!])$.*

We can immediately prove basic algebraic properties of the observation function.

Lemma 21. *The following identities hold for any closed value v and ω-chain $(\mathfrak{v}_n)_n$ in $T\mathcal{V}_\circ$.*

$$\begin{aligned} obs(\bot) &= \bot_{\mathbb{1}} \\ obs(\eta_{\mathcal{V}_\circ}(v)) &= \eta_{\mathbb{1}}(*) \\ obs([\![\mathbf{op}]\!]_{\mathcal{V}_\circ}(p,\kappa)) &= [\![\mathbf{op}]\!]_{\mathbb{1}}(p, obs \cdot \kappa) \\ obs(\bigsqcup_{n<\omega} \mathfrak{v}_n) &= \bigsqcup_{n<\omega} obs(\mathfrak{v}_n). \end{aligned}$$

Proof. We first observe that by very definition of monad we have $obs = (\eta_{\mathbb{1}} \cdot !)^{\dagger}$. As a consequence, we can rely both on strictness and continuity of $-^{\dagger}$ to prove the first and last identities above. The second identity can be easily proved using standard monad equalities, whereas the third identity one is a direct consequence of algebraicity of operation symbols. □

Remark 12. Notice that the function *obs* is uniquely determined by the signature Σ. This is because of our choice of not distinguishing between values. Nevertheless, it is easy to see that our definition of *obs* can be extended to any function $\gamma : \mathcal{V}_\circ \to O$, where O represents a set of observables and γ encodes a ground observation on values. In fact, we would just define *obs* as $\eta \cdot \gamma$.

The following result links the map *obs* with the relation $\Gamma 0_{\mathcal{V}}$. Intuitively, it states that depending on whether we are dealing with relators for simulation or bisimulation, if $[\![e]\!]$ and $[\![f]\!]$ are related by $\Gamma 0_{\mathcal{V}}$, i.e. $[\![e]\!]\ \Gamma 0_{\mathcal{V}}\ [\![f]\!]$ holds, then either the ground observation of f refines the ground observation of e, or e and f have the same ground observation.

Lemma 22. *We say that a relator Γ for a functor T is flat if Γ satisfies (rel funct 1)[4], and that a relator is quasi-flat if $\Gamma \mathsf{I}_X = \sqsubseteq_X$ (recall that $\sqsubseteq_X$ is an ω-pointed complete partial order on TX). The following hold:*

[4] I.e. $\Gamma \mathsf{I}_X = \mathsf{I}_{TX}$

1. *If* Γ *is flat, then* $\Gamma 0_{\mathcal{V}} = obs^{\circ} \cdot \mathsf{I}_{T\mathbb{1}} \cdot obs$.
2. *If* Γ *is quasi-flat, then* $\Gamma 0_{\mathcal{V}} = obs^{\circ} \cdot \sqsubseteq_{\mathbb{1}} \cdot obs$.

Proof. Suppose Γ to be flat. Then calculate:

$$\begin{aligned} obs^{\circ} \cdot \sqsubseteq \cdot obs &= (T!_{\mathcal{V}_\circ})^{\circ} \cdot \Gamma \mathsf{I}_{\mathbb{1}} \cdot T!_{\mathcal{V}_\circ} \\ &\quad \text{[By (rel funct 1)]} \\ &= \Gamma(!^{\circ}_{\mathcal{V}_\circ} \cdot \mathsf{I}_{\mathbb{1}} \cdot !_{\mathcal{V}_\circ}) \\ &\quad \text{[By (stability)]} \\ &= \Gamma\, 0_{\mathcal{V}}\,. \end{aligned}$$

Similarly, we see that if Γ is quasi-flat, then $\Gamma 0_{\mathcal{V}} = obs^{\circ} \cdot \sqsubseteq_{\mathbb{1}} \cdot obs$. □

Finally, we notice that most of the relators of the form $\hat{\mathbb{T}} \wedge \hat{\mathbb{T}}^{\circ}$ in Section 4.3 are flat, and that most of the relators of the form $\hat{\mathbb{T}}$ in Section 4.3 are quasi-flat. In fact, it is easy to see that since $\sqsubseteq_X \cap \sqsubseteq_X^{\circ} = \mathsf{I}_{TX}$, if Γ is quasi-flat, then $\Gamma \wedge \Gamma^{\circ}$ is flat.

From now on we assume relators to be quasi-flat. This is a convenient assumption, as otherwise the following notions of contextual approximation and equivalence would be different than the one defined in terms of relators. In fact, we rephrase the notions of contextual equivalence and approximation in terms of *obs*, as well as the notion of *effectful CIU equivalence* and *approximation*, as follows.

Definition 41. *Contextual approximation* $\preceq^{\mathrm{ctx}} \colon \Lambda \nrightarrow \Lambda$ *is defined as follows:*

$$e \preceq^{\mathrm{ctx}} f \overset{\triangle}{\iff} \forall C \in \mathsf{Cl}(e) \cap \mathsf{Cl}(f).\ obs(C[e]) \sqsubseteq obs(C[f]).$$

CIU approximation $\preceq^{\mathrm{ciu}} \colon \Lambda_\circ \nrightarrow \Lambda_\circ$ *is defined as follows, where* e, f *are* closed *terms:*

$$e \preceq^{\mathrm{ciu}} f \overset{\triangle}{\iff} \forall E.\ obs(E[e]) \sqsubseteq obs(E[f]).$$

As usual, we define contextual $\simeq^{\mathrm{ctx}}$ *and CIU equivalence* $\simeq^{\mathrm{ciu}}$ *as* $\preceq^{\mathrm{ctx}} \cap (\preceq^{\mathrm{ctx}})^{\circ}$ *and* $\preceq^{\mathrm{ciu}} \cap (\preceq^{\mathrm{ciu}})^{\circ}$*, respectively.*

Finally, we define *monadic applicative simulation* and *monadic applicative bisimulation*. These are relations over monadic values that test the latter applicatively. In order for that to make sense, we have to lift the notion of application from terms to monadic values.

Definition 42. *For* $\mathfrak{v} \in T\mathcal{V}_\circ$ *and* $e \in \Lambda_\circ$ *define the* monadic application $\mathfrak{v} * e \in T\mathcal{V}_\circ$ *of* e *to* $\mathfrak{v}$ *as*

$$\mathfrak{v} * e \triangleq (v \mapsto [\![vf]\!])^{\dagger}(\mathfrak{v}).$$

We immediately notice that $[\![ef]\!] = [\![e]\!] * f$ and that straightforward calculations give the following identities:

$$\begin{aligned} \bot * e &= \bot; \\ \eta(\lambda x.f) * e &= [\![f[e := x]]\!]; \\ [\![\mathbf{op}]\!](p, \kappa) * e &= [\![\mathbf{op}]\!](p, i \mapsto \kappa(i) * e). \end{aligned}$$

Definition 43. *A relation* $\mathcal{R} : T\mathcal{V}_\circ \nrightarrow T\mathcal{V}_\circ$ *is a* monadic applicative simulation *(monadic simulation, hereafter) if*

$$\mathfrak{v}\ \mathcal{R}\ \mathfrak{w} \implies obs(\mathfrak{v}) \sqsubseteq obs(\mathfrak{w}) \wedge \forall e \in \Lambda_\circ.\ \mathfrak{v} * e\ \mathcal{R}\ \mathfrak{w} * e.$$

Monadic applicative similarity (monadic similarity, hereafter) $\preceq^{\mathsf{M}}$ *is defined as the* largest *monadic simulation. We say that* f *is monadic similar to* e*, and write* $e \preceq^{\mathsf{M}} f$*, if* $[\![e]\!] \preceq^{\mathsf{M}} [\![f]\!]$.

In particular, Definition 43 induced a monotone endofunction $\mathcal{R} \mapsto [\mathcal{R}]$ (monotonicity being a consequence of monotonicity of *obs*) on the complete lattice $\mathsf{Rel}(T\mathcal{V}_\circ, T\mathcal{V}_\circ)$ such that $\mathcal{R}$ is a monadic simulation if and only if $\mathcal{R} \subseteq [\mathcal{R}]$. As a consequence, we can define monadic applicative similarity as the greatest fixed point of $\mathcal{R} \mapsto [\mathcal{R}]$. As usual, $\precsim^{\mathsf{M}}$ comes with as associated *coinduction proof principle*:

$$\frac{\mathcal{R} \subseteq \precsim^{\mathsf{M}}}{\mathcal{R} \subseteq [\mathcal{R}]} \ (\precsim^{\mathsf{M}}\text{-coind.})$$

Using the coinduction proof principle of monadic similarity we can prove $\precsim^{\mathsf{M}}$ to be preorder.

Proposition 18. *Monadic similarity is reflexive and transitive.*

Proof. The proof is by coinduction. First we notice that $\mathsf{I}_{T\mathcal{V}_\circ}$ is a monadic simulation. Secondly, we observe that if $\mathcal{R}$ and $\mathcal{S}$ are applicative simulation, then so is $\mathcal{S} \cdot \mathcal{R}$ (from which follows $\precsim^{\mathsf{M}} \cdot \precsim^{\mathsf{M}} \subseteq \precsim^{\mathsf{M}}$). In fact, if both $\mathfrak{v}\ \mathcal{R}\ \mathfrak{w}$ and $\mathfrak{w}\ \mathcal{S}\ \mathfrak{u}$ hold, then we have $obs(\mathfrak{v}) \sqsubseteq obs(\mathfrak{u})$, as $\sqsubseteq$ is transitive. Moreover, since $\mathcal{R}$ and $\mathcal{S}$ are simulation, for any closed term e we have $\mathfrak{v} * e\ \mathcal{R}\ \mathfrak{w} * e$ and $\mathfrak{w} * e\ \mathcal{S}\ \mathfrak{u} * e$, and thus $\mathfrak{v} * e\ (\mathcal{S} \cdot \mathcal{R})\ \mathfrak{u} * e$. □

We also notice that monadic similarity is deterministic, so that we can define *monadic applicative bisimilarity* (monadic bisimilarity, hereafter) as $\precsim^{\mathsf{M}} \cap (\precsim^{\mathsf{M}})^\circ$. Since we have $\simeq^{\mathsf{ciu}} = \precsim^{\mathsf{ciu}} \cap (\precsim^{\mathsf{ciu}})^\circ$ and $\simeq^{\mathsf{ctx}} = \precsim^{\mathsf{ctx}} \cap (\precsim^{\mathsf{ctx}})^\circ$, our metatheoretical results on program refinement will straightforwardly generalises to the associated notions of program equivalence. Additionally, as a further consequence of determinism, we see that monadic similarity is a trace-like approximation, and that it actually coincides with CIU approximation.

Proposition 19. *Monadic similarity $\precsim^{\mathsf{M}}$ coincides with CIU approximation $\precsim^{\mathsf{ciu}}$.*

Proof. We first prove $\precsim^{\mathsf{M}} \subseteq \precsim^{\mathsf{ciu}}$. We notice that by very definition of monadic simulation, $e \precsim^{\mathsf{M}} f$ implies $eg \precsim^{\mathsf{M}} fg$, for any term g. As a consequence, for any evaluation context E, $e \precsim^{\mathsf{M}} f$ implies $E[e] \precsim^{\mathsf{M}} E[f]$, and thus $obs(E[e]) \sqsubseteq obs(E[f])$. To prove $\precsim^{\mathsf{ciu}} \subseteq \precsim^{\mathsf{M}}$ we proceed by coinduction showing that

$$\mathcal{R} \triangleq \{(\llbracket E[e] \rrbracket, \llbracket E[f] \rrbracket) \mid e \precsim^{\mathsf{ciu}} f\}$$

is a monadic simulation. Clearly $\mathcal{R}$ is closed under the lifting of application, since $\llbracket E[e] \rrbracket * f = \llbracket E[e] f \rrbracket = \llbracket E'[e] \rrbracket$, for $E' \triangleq Ef$. To conclude the thesis, we have to show $obs(E[e]) \sqsubseteq obs(E[f])$. This is obviously the case since $e \precsim^{\mathsf{ciu}} f$. □

Corollary 3. *Monadic bisimilarity $\simeq^{\mathsf{M}}$ coincides with CIU equivalence $\simeq^{\mathsf{ciu}}$.*

Example 40. We show that if $\llbracket \mathbf{op} \rrbracket_{\mathbb{1}}(p, i \mapsto \eta(*)) = \eta(*)$, then λs distribute over operations. I.e.

$$\mathbf{op}(p, x.\lambda y.e) \simeq^{\mathsf{M}} \lambda y.\mathbf{op}(p, x.e).$$

By Corollary 3, it is sufficient to show $op(p, x.\lambda y.e) \simeq^{\mathsf{ciu}} \lambda y.\mathbf{op}(p, x.e)$. Let $E \triangleq [-] f_1 \cdots e_n$. We show by induction on n

$$obs(E[op(p, x.\lambda y.e)]) = obs(E[\lambda y.\mathbf{op}(p, x.e)]).$$

The case for $n \geq 1$ is trivial. Suppose $n = 0$, so that we have to show $obs(\mathbf{op}(p, x.\lambda y.e)) = obs(\lambda y.\mathbf{op}(p, x.e))$. We calculate:

$$\begin{aligned}
obs(\mathbf{op}(p, x.\lambda y.e)) &= obs(\llbracket \mathbf{op} \rrbracket_{\mathcal{V}_\circ}(p, v \mapsto \llbracket \lambda y.e[x := v] \rrbracket)) \\
&= \llbracket \mathbf{op} \rrbracket_{\mathbb{1}}(p, v \mapsto obs(\eta(\lambda y.e[x := v]))) \\
&= \llbracket \mathbf{op} \rrbracket_{\mathbb{1}}(p, v \mapsto \eta(*)) \\
&= \eta(*) \\
&= obs(\lambda y.\mathbf{op}(p, x.e)).
\end{aligned}$$

The condition $[\![\mathbf{op}]\!]_{\mathbb{1}}(p, i \mapsto \eta(*)) = \eta(*)$ holds both for the partial nondeterministic monad $\mathbb{FM}$ (with signature $\Sigma_{\mathbb{FM}}$) and the partial distribution monad $\mathbb{DM}$ (with signature $\Sigma_{\mathbb{DM}}$), but fails, for instance, for the output monad $\mathbb{O}^\infty$ (with signature $\Sigma_{\mathbb{O}^\infty}$). ⊠

6.4 Full Abstraction

In this section we prove a CIU theorem (cf. Theorem 8) stating that $\preceq^{\text{ciu}}$ coincides with $\preceq^{\text{ctx}}$. In virtue of Proposition 19, we will also obtain full abstraction of $\preceq^{\text{M}}$ with respect to $\preceq^{\text{ctx}}$. Our proof of Theorem 8 follows (Pitts, 2011) and employs a variation of Howe's technique as defined in Chapter 5. First of all, we adapt the notion of Howe extension of a λ-term relation defined in Chapter 5 to $\Lambda_\Sigma^{(\mathsf{n})}$. In order to deal with evaluation contexts, we also define a suitable extension $\mathcal{R}^{\text{E}}$ of a λ-term relation $\mathcal{R}$ to *closed* evaluation contexts (i.e. evaluation contexts built using closed terms only).

Remark 13. The calculus $\Lambda_\Sigma^{(\mathsf{n})}$ being coarse-grain, we work with λ-term relations defined on the whole set of terms. As a consequence, for a λ-term relation $\mathcal{R}$ we simply write $\Gamma \vdash e\ \mathcal{R}\ f$ to say the the open terms e, f with free variables in Γ are related by $\mathcal{R}$. As usual, we use the notion $\vdash e\ \mathcal{R}\ f$ and $e\ \mathcal{R}\ f$, interchangeably. All the basic relational constructions (such as open extension and closed restriction) of Chapter 5 are modified accordingly.

Definition 44. *Given a closed λ-term relation $\mathcal{R}$, the Howe extension $\mathcal{R}^H$ of $\mathcal{R}$ is the open λ-term relation inductively defined by rules* (H-var)-(H-op) *in Figure 6.1. The evaluation context extension $\mathcal{R}^{\text{E}}$ of $\mathcal{R}$ is the relation over closed evaluation contexts defined by rules* (E-nil) *and* (E-nil) *in Figure 6.1.*

$$\frac{\Gamma \vdash x\ \mathcal{R}\ e}{\Gamma \vdash x\ \mathcal{R}^H\ e}\ (\text{H-var})$$

$$\frac{\Gamma, x \vdash e\ \mathcal{R}^H\ g \quad \Gamma \vdash \lambda x.g\ \mathcal{R}\ f}{\Gamma \vdash \lambda x.e\ \mathcal{R}^H\ f}\ (\text{H-abs}) \qquad \frac{\Gamma \vdash e\ \mathcal{R}^H\ e' \quad \Gamma \vdash h\ \mathcal{R}^H\ g' \quad \Gamma \vdash e'g'\ \mathcal{R}\ f}{\Gamma \vdash eg\ \mathcal{R}^H\ f}\ (\text{H-app})$$

$$\frac{\Gamma, x \vdash e\ \mathcal{R}^H\ g \quad \Gamma \vdash \mathbf{op}(p, x.g)\ \mathcal{R}\ f}{\Gamma \vdash \mathbf{op}(p, x.e)\ \mathcal{R}^H\ f}\ (\text{H-op})$$

$$\frac{}{[-]\ \mathcal{R}^{\text{E}}\ [-]}\ (\text{E-nil}) \qquad \frac{\vdash e\ \mathcal{R}^H\ e' \quad E\ \mathcal{R}^{\text{E}}\ E'}{Ee\ \mathcal{R}^{\text{E}}\ E'e'}\ (\text{E-app})$$

Figure 6.1: Howe and evaluation context extension of $\mathcal{R}$ for $\Lambda_\Sigma^{(\mathsf{n})}$.

We observe that if $\mathcal{R}$ is a reflexive λ-term relation, then $\mathcal{R}^{\text{E}}$ nicely interacts with $\mathcal{R}^H$, in the sense that the following holds (the proof is a straightforward induction on the derivation of $E\ \mathcal{R}^{\text{E}}\ E'$):

$$\frac{E\ \mathcal{R}^{\text{E}}\ E' \quad \Gamma \vdash e\ \mathcal{R}^H\ e'}{\Gamma \vdash E[e]\ \mathcal{R}^H\ E'[E']}$$

Moreover, it is a straightforward exercise to show that the analogous of Lemma 15 holds in the context of $\Lambda_\Sigma^{(\mathsf{n})}$. In particular, $\preceq^{\text{ciu}}$ being a preorder we see that $(\preceq^{\text{ciu}})^H$ is compatible, substitutive (obviously $(\preceq^{\text{ciu}})^o$ is value substitutive), and contains $(\preceq^{\text{ciu}})^o$.

The main technical part of our variation of the Howe's method is Lemma 24, which is nothing but a suitable variation of Lemma 20 of Chapter 5. In order to prove it, it is useful to first prove the following auxiliary result.

Lemma 23. *If* $\vdash E[e]\ (\preceq^{\text{ciu}})^H\ g$, *then there exist an evaluation context* F *and a term* f *such that* $E\ (\preceq^{\text{ciu}})^{\mathsf{E}}\ F$, $\vdash e\ (\preceq^{\text{ciu}})^H\ f$, *and* $\vdash F[f] \preceq^{\text{ciu}} g$.

Proof. We proceed by induction on E. If $E = [-]$, then we take $F \triangleq [-]$ and $f \triangleq g$. The thesis follows since $\preceq^{\text{ciu}}$ is reflexive. If E is of the form $E'e'$, then $\vdash E'[e]e'\ (\preceq^{\text{ciu}})^H\ g$ must be the conclusion of an instance of rule (H-app), i.e. of a derivation of the form

$$\frac{\vdash E'[e]\ (\preceq^{\text{ciu}})^H\ c \quad \vdash e'\ (\preceq^{\text{ciu}})^H\ f' \quad \vdash cf' \preceq^{\text{ciu}} g}{\vdash E'[e]e'\ (\preceq^{\text{ciu}})^H\ g}\ \text{(H-app)}$$

for some terms c, f'. By induction hypothesis, from $\vdash E'[e]\ (\preceq^{\text{ciu}})^H\ c$ we infer the existence of an evaluation context F' and a term f such that $E'\ (\preceq^{\text{ciu}})^{\mathsf{E}}\ F'$, $\vdash e\ (\preceq^{\text{ciu}})^H\ f$, and $\vdash F'[f] \preceq^{\text{ciu}} c$. From the latter we infer $\vdash F'[f]f' \preceq^{\text{ciu}} cf'$, and thus $\vdash F'[f]f' \preceq^{\text{ciu}} g$, since $\vdash cf' \preceq^{\text{ciu}} g$ and $\preceq^{\text{ciu}}$ is transitive. Finally, by rule (E-app) we see that $\vdash e'\ (\preceq^{\text{ciu}})^H\ f'$ and $E'\ (\preceq^{\text{ciu}})^{\mathsf{E}}\ F'$ implies $\vdash E'e'\ (\preceq^{\text{ciu}})^{\mathsf{E}}\ F'f'$, and thus we are done taking $F \triangleq F'f'$. □

Lemma 24. *For all closed evaluation contexts* E, F *and closed terms* e, f, *if* $E\ (\preceq^{\text{ciu}})^{\mathsf{E}}\ F$ *and* $\vdash e\ (\preceq^{\text{ciu}})^H\ f$, *then* $obs(E[e]) \sqsubseteq obs(F[f])$.

Proof. Since obs is continuous to prove the thesis it is sufficient to prove:

$$\forall n \geq 0.\ obs([\![E[e]]\!]_n) \sqsubseteq obs([\![F[f]]\!]).$$

We proceed by induction on n. The case for $n = 0$ is trivial. Let $n > 0$ and suppose the thesis holds for all $m < n$. We proceed by case analysis on the form of e.

- Suppose e to be a value $\lambda x.e'$. Then $\vdash e\ (\preceq^{\text{ciu}})^H\ f$ must be the conclusion of an derivation of the form

  $$\frac{x \vdash e\ (\preceq^{\text{ciu}})^H\ c \quad \vdash \lambda x.c \preceq^{\text{ciu}} f}{\vdash \lambda x.e'\ (\preceq^{\text{ciu}})^H\ f}\ \text{(H-abs)}$$

  We now make a further case analysis on the shape of E. If $E = [-]$, then, since $E\ (\preceq^{\text{ciu}})^{\mathsf{E}}\ F$, we must have $F = [-]$ too. Therefore, since $\vdash \lambda x.c \preceq^{\text{ciu}} f$ (meaning that $obs([\![\lambda x.c]\!]) \sqsubseteq obs([\![f]\!])$), we have:

  $$\begin{aligned} obs([\![e]\!]_n) &= obs(\eta(\lambda x.e')) \\ &= \eta_{\mathbb{1}}(*) \\ &= obs([\![\lambda x.c]\!]) \\ &\sqsubseteq obs([\![f]\!]). \end{aligned}$$

  Suppose now E to be of the form $[-]g\vec{g}$, for some terms $g, \vec{g}$. Since $E\ (\preceq^{\text{ciu}})^{\mathsf{E}}\ F$, there exists terms $h, \vec{h}$ such that $F = [-]h\vec{h}$, $\vdash g\ (\preceq^{\text{ciu}})^H\ h$, and $\vdash \vec{g}\ (\preceq^{\text{ciu}})^H\ \vec{h}$. Moreover, we have:

  $$[\![E[e]]\!]_n = [\![(\lambda x.e)g\vec{g}]\!]_n = [\![e'[x := g]\vec{g}]\!]_{n-1}.$$

  By substitutivity of $(\preceq^{\text{ciu}})^H$, $x \vdash e'\ (\preceq^{\text{ciu}})^H\ c$ and $\vdash g\ (\preceq^{\text{ciu}})^H\ h$ imply $\vdash e'[x := g]\ (\preceq^{\text{ciu}})^H\ c[x := h]$, and thus $\vdash e'[x := g]\vec{g}\ (\preceq^{\text{ciu}})^H\ c[x := h]\vec{h}$, since $(\preceq^{\text{ciu}})^H$ is compatible and $\vdash \vec{g}\ (\preceq^{\text{ciu}})^H\ \vec{h}$. We obtain

the wished thesis as follows:

$$\begin{aligned} obs(\llbracket E[e] \rrbracket_n) &= obs(\llbracket e'[x := g]\vec{g} \rrbracket_{n-1}) \\ &\sqsubseteq obs(\llbracket c[x := h]\vec{h} \rrbracket) \\ &\quad \text{[By induction hypothesis]} \\ &= obs(\llbracket F[\lambda x.c] \rrbracket) \\ &\sqsubseteq obs(\llbracket F[f] \rrbracket). \\ &\quad \text{[Since } \lambda x.c \preceq^{\text{ciu}} f\text{]} \end{aligned}$$

- Suppose e to be of the form $E'[(\lambda x.e')c]$, so that we have $\vdash E'[(\lambda x.e')c]\ (\preceq^{\text{ciu}})^H\ f$. By Lemma 23 there exists an evaluation context F' and a term f' such that $E'\ (\preceq^{\text{ciu}})^{\text{E}}\ F'$, $\vdash (\lambda x.e')c\ (\preceq^{\text{ciu}})^H\ f$, and $\vdash F'[f']\ \preceq^{\text{ciu}}\ f$. In particular, $\vdash (\lambda x.e')c\ (\preceq^{\text{ciu}})^H\ f$ must be the conclusion of a derivation of the form:

$$\dfrac{\dfrac{x \vdash e'\ (\preceq^{\text{ciu}})^H\ h \quad \vdash \lambda x.h \preceq^{\text{ciu}} g}{\vdash \lambda x.e'\ (\preceq^{\text{ciu}})^H\ g}\ \text{(H-abs)} \qquad \vdash c\ (\preceq^{\text{ciu}})^H\ g' \quad \vdash gg' \preceq^{\text{ciu}} f'}{\vdash (\lambda x.e')c\ (\preceq^{\text{ciu}})^H\ f}\ \text{(H-app)}$$

In particular, we now have the following CIU-approximations:

$$\vdash \lambda x.h \preceq^{\text{ciu}} g \tag{6.1}$$

$$\vdash gg' \preceq^{\text{ciu}} f' \tag{6.2}$$

$$\vdash F'[f'] \preceq^{\text{ciu}} f. \tag{6.3}$$

From $x \vdash e'\ (\preceq^{\text{ciu}})^H\ h$ and $\vdash c\ (\preceq^{\text{ciu}})^H\ g'$, by substitutivity, we obtain $\vdash e'[x := c]\ (\preceq^{\text{ciu}})^H\ h[x := g']$, whereas from $E\ (\preceq^{\text{ciu}})^{\text{E}}\ F$ and $E'\ (\preceq^{\text{ciu}})^{\text{E}}\ F'$ we infer $E[E'[-]]\ (\preceq^{\text{ciu}})^{\text{E}}\ F[F'[-]]$. Denoting by G and G' the evaluation contexts $E[E'[-]]$ and $F[F'[-]]$, respectively, we see that we thus have:

$$\vdash G[e'[x := c]]\ (\preceq^{\text{ciu}})^H\ G'[h[x := g']]. \tag{6.4}$$

We obtain the wished thesis as follows:

$$\begin{aligned} obs(\llbracket E[e] \rrbracket_n) &= obs(\llbracket G[(\lambda x.e')c] \rrbracket_n) \\ &= obs(\llbracket G[e'[x := c]] \rrbracket_{n-1}) \\ &\sqsubseteq obs(\llbracket G'[h[x := g']] \rrbracket) \\ &\quad \text{[By (6.4) and the induction hypothesis]} \\ &= obs(\llbracket G'[(\lambda x.h)g'] \rrbracket) \\ &\sqsubseteq obs(\llbracket G'[gg'] \rrbracket) \\ &\quad \text{[By (6.1)]} \\ &\sqsubseteq obs(\llbracket G'[f'] \rrbracket) \\ &\quad \text{[By (6.2)]} \\ &= obs(\llbracket F[F'[f']] \rrbracket) \\ &\sqsubseteq obs(\llbracket F[f] \rrbracket) \\ &\quad \text{[By (6.3)]} \end{aligned}$$

- Suppose e to be of the form $E'[\mathbf{op}(p, x.e')]$, so that we have $\vdash E'[\mathbf{op}(p, x.e')]\ (\preceq^{\text{ciu}})^H\ f$. We proceed exactly as in previous case. By Lemma 23 there exists an evaluation context F' and a term f' such

that $E'\ (\preceq^{\text{ciu}})^{\text{E}}\ F'$, $\vdash \mathbf{op}(p, x.e')\ (\preceq^{\text{ciu}})^{H}\ f'$, and $\vdash F'[f'] \preceq^{\text{ciu}} f$. In particular, $\vdash \mathbf{op}(p, x.e')\ (\preceq^{\text{ciu}})^{H}\ f'$ must be the conclusion of a derivation of the form:

$$\frac{x \vdash e'\ (\preceq^{\text{ciu}})^{H}\ c \quad \vdash \mathbf{op}(p, x.c) \preceq^{\text{ciu}} f'}{\vdash \mathbf{op}(p, x.e')\ (\preceq^{\text{ciu}})^{H}\ f'}\ \text{(H-op)}$$

We thus have the following CIU-approximations:

$$\vdash \mathbf{op}(p, x.c) \preceq^{\text{ciu}} f' \tag{6.5}$$

$$\vdash F'[f'] \preceq^{\text{ciu}} f. \tag{6.6}$$

From $x \vdash e'\ (\preceq^{\text{ciu}})^{H}\ c$, by substitutivity, we obtain $\vdash e'[x := v]\ (\preceq^{\text{ciu}})^{H}\ c[x := v]$, for any closed value v, whereas from $E\ (\preceq^{\text{ciu}})^{\text{E}}\ F$ and $E'\ (\preceq^{\text{ciu}})^{\text{E}}\ F'$ we infer $E[E'[-]]\ (\preceq^{\text{ciu}})^{\text{E}}\ F[F'[-]]$. Denoting by G and G' the evaluation contexts $E[E'[-]]$ and $F[F'[-]]$, respectively, we see that we have:

$$\vdash G[e'[x := v]]\ (\preceq^{\text{ciu}})^{H}\ G'[c[x := v]]. \tag{6.7}$$

We obtain the wished thesis as follows:

$$\begin{aligned}
obs([\![E[e]]\!]_n) &= obs([\![G[\mathbf{op}(p, x.e')]]\!]_n) \\
&= obs([\![\mathbf{op}]\!]_{\mathcal{V}_\circ}(p, v \mapsto [\![G[e'[x := v]]]\!]_{n-1}) \\
&= [\![\mathbf{op}]\!]_{\mathbb{1}}(p, v \mapsto obs([\![G[e'[x := v]]]\!]_{n-1})) \\
&\quad \text{[By Lemma 21]} \\
&\sqsubseteq [\![\mathbf{op}]\!]_{\mathbb{1}}(p, v \mapsto obs([\![G'[c[x := v]]]\!])) \\
&\quad \text{[By (6.7) and induction hypothesis]} \\
&= obs([\![\mathbf{op}]\!]_{\mathcal{V}_\circ}(p, v \mapsto [\![G'[c[x := v]]]\!])) \\
&\quad \text{[By Lemma 21]} \\
&= obs([\![G'[\mathbf{op}(p, x.c)]]\!]) \\
&\sqsubseteq obs([\![G'[f']]\!]) \\
&\quad \text{[By (6.5)]} \\
&= obs([\![F[F'[f']]]\!]) \\
&\sqsubseteq obs([\![F[f]]\!]). \\
&\quad \text{[By (6.6)]}
\end{aligned}$$

□

An immediate consequence of Lemma 24 is compatibility of $\preceq^{\text{ciu}}$ and thus its equivalence with $\preceq^{\text{ctx}}$.

Theorem 8 (CIU equivalence)**.** *CIU approximation $\preceq^{\text{ciu}}$ is compatible and thus coincide with $\preceq^{\text{ctx}}$.*

Proof. We know by very definition of Howe extension that $(\preceq^{\text{ciu}})^{H}$ is compatible. Therefore, to prove compatibility of $\preceq^{\text{ciu}}$ it is sufficient to show that the open extension of $\preceq^{\text{ciu}}$ coincides with $(\preceq^{\text{ciu}})^{H}$. As usual, that follows if the closed restriction of $(\preceq^{\text{ciu}})^{H}$ is included in $\preceq^{\text{ciu}}$. That is, if for all closed terms e, f:

$$\vdash e\ (\preceq^{\text{ciu}})^{H}\ f \implies \forall E.\ obs(E[e]) \sqsubseteq obs(E[f]).$$

The latter is direct consequence of Lemma 24, since $E\ (\preceq^{\text{ciu}})^{\text{E}}\ E$ always holds. Finally, we observe that, as in Lemma 19, we can show that $\preceq^{\text{ctx}}$ is included in the open extension $\preceq^{\text{ciu}}$, and thus conclude $(\preceq^{\text{ciu}})^{o} = \preceq^{\text{ctx}}$. □

In virtue of Corollary 3 we obtain full abstraction of (the open extension of) $\preceq^{\mathsf{M}}$ with respect to $\preceq^{\mathrm{ctx}}$.

Corollary 4 (Full abstraction 1)**.** *The open extension of monadic applicative similarity $\preceq^{\mathsf{M}}$ is fully abstract with respect to $\preceq^{\mathrm{ctx}}$.*

Finally, since $\simeq^{\mathsf{M}} = \preceq^{\mathsf{M}} \cap (\preceq^{\mathsf{M}})^{\circ}$, $\simeq^{\mathrm{ciu}} = \preceq^{\mathrm{ciu}} \cap (\preceq^{\mathrm{ciu}})^{\circ}$, and $\simeq^{\mathrm{ctx}} = \preceq^{\mathrm{ctx}} \cap (\preceq^{\mathrm{ctx}})^{\circ}$, we see that Theorem 8 and Corollary 4 give the following full abstraction result.

Corollary 5 (Full abstraction 2)**.** *The open extension of monadic applicative bisimilarity $\simeq^{\mathsf{M}}$ coincides with the open extension of CIU equivalence $\simeq^{\mathrm{ciu}}$ which coincides with contextual equivalence $\simeq^{\mathrm{ctx}}$.*

We conclude this chapter stressing that at the hearth of full abstraction of $\preceq^{\mathsf{M}}$ and $\simeq^{\mathsf{M}}$ is the simpler nature of call-by-name calculi compared to call-by-value calculi. In the latter, a context can test its input both in functional and argument position, thus obtaining a stronger testing power than call-by-name contexts, where, instead, terms can only be tested in functional position. For instance, the call-by-value context $(\lambda x.C)[-]$ forces the evaluation of its input, and then pass the result obtained to C.

This is indeed the reason why monadic applicative (bi)similarity works for call-by-name calculi only. In fact, extending monadic applicative bisimilarity to the call-by-value case (which is straightforward) would be simply unsound. For instance, the call-by-value probabilistic terms $\lambda x.(x \mathbf{or} \Omega)$, $(\lambda x.x) \mathbf{or} (\lambda x.\Omega)$ are not contextually equivalent, as witnessed by the context $(\lambda x.x(x(\lambda y.y)))[-]$. However, we see that $obs(\lambda x.(x \mathbf{\ or\ } \Omega)) = obs((\lambda x.x) \mathbf{\ or\ } (\lambda x.\Omega))$ and that

$$\begin{aligned}
[\![\lambda x.(x \mathbf{\ or\ } \Omega)]\!] * v &= (1 \cdot \mathit{just}\ (\lambda x.(x \mathbf{\ or\ } \Omega))) * v \\
&= \frac{1}{2} \cdot \mathit{just}\ v + \frac{1}{2} \cdot \bot \\
[\![(\lambda x.x) \mathbf{\ or\ } (\lambda x.\Omega)]\!] * v &= \left(\frac{1}{2} \cdot \mathit{just}\ (\lambda x.x) + \frac{1}{2} \cdot \mathit{just}\ (\lambda x.\Omega)\right) * v \\
&= \frac{1}{2} \cdot \mathit{just}\ v + \frac{1}{2} \cdot \bot,
\end{aligned}$$

meaning that $\lambda x.(x \mathbf{\ or\ } \Omega) \simeq^{\mathsf{M}} (\lambda x.x) \mathbf{\ or\ } (\lambda x.\Omega)$.

Having studied *applicative* notions of program approximation and equivalence, in the next chapter we apply our framework to design more intensional notions of refinement and equivalence, which we call *effectful normal form similarity* and *effectful normal form bisimilarity*.

Chapter 7

Normal Form Similarity and Bisimilarity

> The world is my world: this is manifest in the fact that the limits of language (of that language which alone I understand) mean the limits of my world
>
> Ludwig Wittgenstein, Tractatus Logico-Philosophicus

The notion of an applicative (bi)simulation is rooted in the idea that programs should be compared *extensionally*, i.e. according to their input-output behaviour. This is reflected in the (bi)simulation clause for values: if $\mathcal{R}$ is an applicative (bi)simulation and $\vdash^{\vee} v\ \mathcal{R}\ w$ holds, then so does $\vdash^{\wedge} vu\ \mathcal{R}\ wu$, for any closed value u. Since v, w are *closed* values, they must be λ-abstractions, i.e. terms of the form $\lambda x.e, \lambda x.f$, respectively. Instead of testing v and w extensionally, one could test them *intensionally*, looking at the open subterms e and f. The notion of equivalence and refinement obtained in this way, are called *normal form (bi)similarity* (S. B. Lassen, 1999, 2005; Sangiorgi, 1994).

Starting from the pioneering work by Böhm, the notion of a *Böhm tree* (Barendregt, 1984), and the associated notion of *Böhm tree equality*, has been proved extremely useful in reasoning about program behaviour. Roughly speaking, the Böhm tree $BT(e)$ of a λ-term e is a possibly infinite tree representing the infinitary head-normal form of e. The celebrated Böhm Theorem, also known as Separation Theorem (Böhm, 1968), stipulates that two terms are contextually equivalent if and only if their respective (appropriately η-equated) Böhm trees are the same.

Böhm Trees can also be defined when terms are *not* evaluated to their *head* normal form, like in the classic theory of λ-calculus, but to their *weak-head* normal form (S. B. Lassen, 1999; Sangiorgi, 1994), or to their *eager* normal form (S. B. Lassen, 2005). In both cases, satisfactory notions of program equivalence can be given, although tree equivalence turns out *not* to coincide with context equivalence (full abstraction can be recovered if the discriminating power of contexts is somehow increased, e.g. through concurrency features (Sangiorgi, 1994), or when the whole calculus is more expressive, e.g. by adding control and states (Støvring & Lassen, 2007)). Due to the infinitary nature of these 'normal form trees', tree-like equivalences can be expressed in a pure coinductive way, without any reference to trees, as first shown in (Sangiorgi, 1992, 1994). For that reason people often refer to them *normal-form bisimilarity* relations.

Contrary to applicative bisimilarity, contextual equivalence, and logical relations, normal form bisimilarity is completely independent on how the environment behaves, and focus on the syntactical structure of the terms at hand. That is, when checking whether two terms e and f are normal form bisimilar, only the syntactic structure of e and f is inspected. This is radically different from what happens for e.g. applicative bisimilarity where in order to see whether e and f are applicatively bisimilar we have to test them against arbitrary 'external' input values.

In this chapter we define *effectful normal form similarity* and *bisimilarity* for (variations of) Λ_Σ, and study their main metatheoretical properties. As for applicative (bi)similarity, we will prove congruence and precongruence theorems for normal form bisimilarity and similarity, respectively. Additionally, normal form bisimilarity allows for the of so-called up-to enhancements of the bisimulation method (Pous & Sangiorgi, 2012).

Remarkably, the very same notions of Σ-continuous monad and Σ-continuous relator that we used in Chapter 5 to prove (pre)congruence of effectful applicative (bi)similarity, guarantee similar results for normal form (bi)similarity. As usual, we begin our analysis with some informal considerations on the nature of the notions we aim to investigate.

7.1 From Applicative to Normal Form Bisimulation

In this section we informally introduce normal form bisimulation by studying some examples of (pairs of) programs whose equivalence cannot be readily established using the techniques developed in previous chapters, but that can be easily proved to be normal form bisimilar (and thus contextually equivalent). This gives evidence on the flexibility and strength of the proposed technique. We will focus on examples drawn from fixed point theory, simply because these, being infinitary in nature, are quite hard to be dealt with with "finitary" techniques like contextual equivalence. For simplicity, our examples are given in coarse-grain style.

Example 41. Our first example comes from the ordinary theory of pure, untyped λ-calculus. Let us consider Curry and Turing call-by-value fixed point combinators Y and Z:

$$Y \triangleq \lambda y.\Delta\Delta \qquad Z \triangleq \Theta\Theta$$
$$\Delta \triangleq \lambda x.y(\lambda z.xxz) \qquad \Theta \triangleq \lambda x.\lambda y.y(\lambda z.xxyz).$$

It is well known that Y and Z are contextually equivalent, and thus applicatively bisimilar (recall that in the pure untyped λ-calculus applicative bisimilarity and contextual equivalence coincide). However, proving Y and Z to be applicatively bisimilar is almost as hard as proving them to be contextually equivalent from first principles. In fact, as already remarked in previous chapters, applicative bisimilarity is an *extensional* notion of program equivalence, as it tests programs for their input-output behaviour. This is reflected by its proof obligation, which is based on a *universal quantification* on function arguments. A major drawback of extensionality is that proving $Y \simeq^{\mathrm{A}} Z$ turns out to be extremely difficult.

To have a taste of that, let us try to construct an applicative bisimulation relating Y and Z. Let $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{V})$ be our candidate relation. Clearly we need to have $(Y, Z) \in \mathcal{R}_\Lambda$. Since $[\![Y]\!] = \mathit{just}\ (\lambda y.\Delta\Delta)$ and $[\![Z]\!] = \mathit{just}\ (\lambda y.y(\lambda z.\Theta\Theta yz))$, in order for $\mathcal{R}$ to be an applicative bisimulation, we need

$$\Delta[y := v]\Delta[y := v]\ \mathcal{R}_\Lambda\ v(\lambda z.\Theta\Theta vz)$$

to hold for any value v. If that is the case, then we would need

$$[\![\Delta[y := v]\Delta[y := v]]\!] = [\![v(\lambda z.\Delta[y := v]\Delta[y := v]z)]\!]\ \hat{\mathbb{M}}\mathcal{R}_\mathcal{V}\ [\![v(\lambda z.\Theta\Theta vz)]\!]$$

too. Assuming $v = \lambda x.e$, we can rewrite the above requirement as:

$$[\![e[x := \lambda z.\Delta[y := v]\Delta[y := v]z]]\!]\ \hat{\mathbb{M}}\mathcal{R}_\mathcal{V}\ [\![e[x := \lambda z.\Theta\Theta vz]]\!].$$

At this point we are stuck, as no information is available about the term e. That is, we have no information on how e tests its input program. In particular, given any context $C[-]$, we can consider the value $\lambda x.C[x]$ (so that $e \equiv C[x]$), meaning that satisfying the above proof obligation (which is required in order for $\mathcal{R}$ to be an applicative bisimulation) is morally equivalent to prove Y and Z to be contextually equivalent directly. ⊠

Example 42. Our next example is a refinement of Example 41 to a probabilistic setting, as proposed in (Sangiorgi & Vignudelli, 2016) (although there it is formulated in its, slightly simpler, call-by-name version). We consider a variation of Turing's call-by-value fixed point combinator which at any iteration can probabilistically decide whether to start another iteration (following the pattern of the standard Turing fixed point combinator) or to turn for good into Y, where Y and Δ are defined as in Example 41:

$$Z \triangleq \Theta\Theta$$
$$\Theta \triangleq \lambda x.\lambda y.(y(\lambda z.\Delta\Delta z) \textbf{ or } y(\lambda z.xxyz)).$$

It is natural to ask whether these new version of Y and Z are still equivalent. However, following insights from previous example, it is not hard to see the equivalence between Y and Z is an example of an equivalence that cannot be readily proved by means of standard operational methods such as (probabilistic) contextual equivalence, (probabilistic) CIU equivalence, and (probabilistic) applicative bisimilarity. All the aforementioned techniques require to test programs in a given environment (such as a whole context or an input argument), and are thus ineffective in handling fixed point combinators such as Y and Z.

We will give an elementary proof of the equivalence between Y and Z in Example 45, and a more elegant proof relying on a suitable *up-to context* technique in Subsection 7.4.2. In (Sangiorgi & Vignudelli, 2016) the call-by-name counterparts of Y and Z are proved to be equivalent using probabilistic environmental bisimilarity (Sangiorgi & Vignudelli, 2016). The notion of an environmental bisimulation (Sangiorgi et al., 2011) involves both an environment storing pairs of terms played during the bisimulation game, and a clause universally quantifying over pairs of terms in the evaluation context closure of such an environment[1], thus making environmental bisimilarity a rather heavy technique to use. Our proof of the equivalence of Y and Z is simpler: in fact, our notion of effectful normal form bisimulation does not involve any universal quantification over all possible closed function arguments (like applicative bisimilarity), or their evaluation context closure (like environmental bisimilarity), or 'closed instantiations' or 'uses' (like CIU equivalence). ⊠

Example 43. Our third example concerns call-by-name calculi and shows how our notion of normal form bisimilarity can handle even intricate recursion schemes. We consider the following argument-switching probabilistic fixed point combinators:

$$P \triangleq AA \qquad\qquad Q \triangleq BB$$
$$A \triangleq \lambda x.\lambda y.\lambda z.(y(xxyz) \textbf{ or } z(xxzy)) \qquad\qquad B \triangleq \lambda x.\lambda y.\lambda z.(y(xxzy) \textbf{ or } z(xxyz)).$$

We easily see that P and Q satisfy the following (informal) program equations:

$$Pef = e(Pef) \textbf{ or } f(Pfe) \qquad\qquad Qef = e(Qfe) \textbf{ or } f(Qef).$$

Again, proving the equivalence between P and Q using applicative bisimilarity is problematic. In fact, testing the applicative behaviour of P and Q requires to reason about the behaviour of e.g. $e(Pef)$, which in turn requires to reason about the (arbitrary) term e, of which no information is provided. The

[1] Meaning that two terms e, f are tested for their applicative behaviour against all terms of the form $E[g], E[h]$, for any pair of terms (g, h) stored in the environment.

(essentially infinitary) normal forms of P and Q, however, can be proved to be essentially the same by reasoning about the syntactical structure of P and Q, as we will do in Section 7.4. Moreover, as we will see, our *up-to context* technique (Definition 55) enables an elegant and concise proof of the equivalence between P and Q. ⊠

Example 44. Our last example discuss the use of the cost monad as an *instrument* to facilitate a more intensional analysis of programs. In fact, as already remarked in Example 24, we can use the ticking operation **tick** to perform cost analysis. For instance, we can consider the following variation of Curry and Turing fixed point combinator of Example 41 obtained by adding the operation symbol **tick** after every λ-abstraction.

$$Y \triangleq \lambda y.\mathbf{tick}(\Delta\Delta) \qquad Z \triangleq \Theta\Theta$$
$$\Delta \triangleq \lambda x.\mathbf{tick}(y(\lambda z.\mathbf{tick}(xxz))) \qquad \Theta \triangleq \lambda x.\mathbf{tick}(\lambda y.\mathbf{tick}(y(\lambda z.\mathbf{tick}(xxyz)))).$$

Following Example 24, we see that every time a β-redex $(\lambda x.\mathbf{tick}(e))v$ is reduced, the ticking operation **tick** increases an imaginary cost counter of a unit. Using ticking, we can provide a more intensional analysis of the relationship between Y and Z, along the lines of Sands' improvement theory (Sands, 1998). ⊠

7.2 Computational Calculi Revisited

Contrary to Chapter 3 and Chapter 5, here we will work with the coarse-grain version of Λ_Σ, which for simplicity we keep denoting as Λ_Σ. As already remarked, fine-grain calculi are in general better suited for metatheoretical purposes, but are a bit cumbersome for writing and studying examples. Given the emphasis we gave to examples and the link we will make with Böhm tree-like equalities, we prefer to work with a pure untyped λ-calculus as in e.g. (Barendregt, 1984) enriched with algebraic operations.

The syntax of the calculus is standard, and it is defined in Figure 7.1, where x ranges over a fixed (countably infinite) set of variables, $\mathbf{op} : P \rightsquigarrow \mathcal{V}$ ranges over elements[2] of Σ, and p ranges over elements of P. Notational and syntactical conventions of Chapter 3 are left unchanged. In particular, we denote by Λ and $\mathcal{V} \subseteq \Lambda$ the collections of (open and closed) terms and values of Λ_Σ, respectively.

$e, f ::= v$	(value)	$v, w ::= x$	(variable)
$\mid f\,e$	(application)	$\mid \lambda x.e.$	(abstraction)
$\mid \mathbf{op}(p, x.e).$	(operation)		

Figure 7.1: Syntax of coarse-grain Λ_Σ.

The operational semantics of Λ_Σ is defined as in Chapter 3, with some minor differences. Before defining such a semantics, we a make couple of useful remarks.

Remark 14. 1. Testing terms according to the their (possibly infinitary) normal forms obviously requires to work with open terms. Indeed, in order to inspect the *intensional* behaviour of a value $\lambda x.e$, one has to inspect the intensional behaviour of e, which is an open term. As a consequence, contrary to all previous chapters, here we give operational semantics to both open and closed terms. Actually, the very distinction between open and closed terms is not that meaningful in this context, and thus we simply refer to terms. As a consequence, also the relational apparatus developed in Chapter 5 collapses to simple relations over terms.

[2] We tacitly apply Remark 3, here properly modified to *open* values.

2. According to point 1, we notice that *values* constitute a syntactic notion defined independently of the operational semantics of the calculus: values are just variables and λ-abstractions. However, giving operational semantics to arbitrary terms (and not just closed ones) we are interested in richer collections of irreducible expressions, i.e. expressions that cannot be simplified any further. Such collections, whose members we call *eager normal forms* in a call-by-value setting, and *weak head normal forms* in a call-by-name setting, will be different accordingly to the operational semantics adopted. For instance, in a call-by-name setting it is natural to regard the term $x((\lambda x.x)v)$ as a terminal expression, whereas in a call-by-value setting $x((\lambda x.x)v)$ can be further simplified to xv, which in turn should be regarded as a terminal expression.

7.2.1 Call-by-Value Operational Semantics

We begin by defining a monadic operational semantics for Λ_Σ relying on Felleisen's notion of a call-by-value evaluation context (Felleisen & Hieb, 1992) (recall Lemma 7 and Corollary 1). Call-by-value evaluation contexts are terms with a single hole $[-]$ defined by the following grammar:

$$E ::= [-] \mid Ee \mid vE.$$

Notice that, contrary to Chapter 3 and Chapter 6 here evaluation contexts are built using *open* terms. We write $E[e]$ for the term obtained by substituting the term e for the hole $[-]$ in E. Redexes are expressions of the form $(\lambda x.e)v$ or $\mathbf{op}(p, x.e)$, the former producing a computation step, the latter producing an effect. Following (S. B. Lassen, 2005) we define a *stuck term* as a term of the form $E[xv]$. Finally, we define the collection $\mathcal{E}$ of *eager normal forms* (enfs hereafter) as the collection of values and stuck terms. We let letters $s, t, \ldots$ to range over elements in $\mathcal{E}$. In particular, enfs are generated by the following grammar:

$$s ::= x \mid \lambda x.e \mid E[xv].$$

We notice that any term is either a value, a stuck term, or an expression of the form $E[r]$, for a redex r (cf. Lemma 7).

Lemma 25. *Any term e can be uniquely decomposed in one of the following (mutually exclusive) forms:*

1. v;
2. $E[vw]$;
3. $E[\mathbf{op}(p, x.f)]$.

Proof. By induction on e we show that e can be decomposed in one of the above forms. If e is a value, then we are trivially done. If e is a term of the form fg, then by induction hypothesis we have two cases to consider.

1. Suppose f is a value v. We apply the induction hypothesis on g. If the latter is a value w, then $fg = vw$ and we are done (consider the empty evaluation context). Otherwise g is a term of the form $E[r]$, for a redex r. We are done since $fg = vE[r]$ and vE is an evaluation context.
2. Suppose f is a term of the form $E[r]$, for a redex r. Then $fg = E[r]g$. We are done since Eg is an evaluation context.

We now prove uniqueness of such composition. This is indeed the case if e is a value. Suppose now e to be of the form $E[vw]$ and to admit another decomposition as above. Such a decomposition cannot be a value. Say it is of the form $E'[v'w']$. We show that then it must be the case that $v = v'$, $w = w'$, and $E = E'$. We proceed by induction on E.

If E is the empty context, we proceed by contradiction and assume $E \neq E'$. Then either $E' = Ff$ or $E' = uF$, for an evaluation context F. In the former case we would have $v = F[v'w']$, which is impossible ($F[v'w']$ cannot be a value). Similarly, in the latter case we would have $w = F[v'w']$ which, again, is not possible. We are done. Suppose now E to be of the form Ff. We immediately notice that $E' \neq [-]$: for we would have $F[vw] = v'$, which is impossible. Similarly we see that E' cannot be of the form uE'. We conclude $E' = F'f'$. The thesis now follows by induction hypothesis. The case for E of the form uF is handled similarity.

In a similar fashion we can prove that if $e = E[\mathbf{op}(p, x.f)]$ and $e = E'[r]$, then $r = \mathbf{op}(p, x.f)$ and $E = E'$. □

Operational semantics of Λ_Σ is defined with respect to a Σ-continuous monad $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ relying on Lemma 25. More precisely, we define a *call-by-value* evaluation function $[\![-]\!]$ mapping each term to an element in $T\mathcal{E}$.

Definition 45. *Define the $\mathbb{N}$-indexed family of maps $[\![-]\!]_n : \Lambda \to T\mathcal{E}$ as follows:*

$$\begin{aligned}
[\![e]\!]_0 &\triangleq \bot \\
[\![v]\!]_{n+1} &\triangleq \eta(v) \\
[\![E[xv]]\!]_{n+1} &\triangleq \eta(E[xv]) \\
[\![E[(\lambda x.e)v]]\!]_{n+1} &\triangleq [\![E[e[x := v]]]\!]_n \\
[\![E[\mathbf{op}(p, x.e)]]\!]_{n+1} &\triangleq [\![\mathbf{op}]\!]_{\mathcal{E}}(p, v \mapsto [\![E[e[x := v]]]\!]_n).
\end{aligned}$$

The monad $\mathbb{T}$ being Σ-continuous, we see that the sequence $([\![e]\!]_n)_n$ forms an ω-chain in $T\mathcal{E}$, so that we can define $[\![e]\!]$ as $\bigsqcup_n [\![e]\!]_n$.

As usual, Σ-continuity of $\mathbb{T}$ ensures continuity of the evaluation function.

Lemma 26. *The map $[\![-]\!]$ is the least function $\varphi : \Lambda \to T\mathcal{E}$ satisfying the following identities:*

$$\begin{aligned}
\varphi(v) &= \eta(v) \\
\varphi(E[xv]) &= \eta(E[xv]) \\
\varphi(E[(\lambda x.e)v]) &= \varphi(E[e[x := v]]) \\
\varphi(E[\mathbf{op}(p, x.e)) &= [\![\mathbf{op}]\!]_{\mathcal{E}}(p, v \mapsto \varphi(E[e[x := v]])).
\end{aligned}$$

In order to improve readability, in the rest of this chapter, for a monad $\mathbb{T}$, an element $\mathfrak{x} \in TX$ and a function $f : X \to TY$, we will sometimes write $\mathfrak{x} \gg\!= f$ in place of $f^\dagger(\mathfrak{x})$. In general, the bind operator $\gg\!= : TX \times (X \to TY) \to TY$ is not equivalent to $-^\dagger$. However, if $\mathbb{T}$ is strong, then we can define $\gg\!=$ as the strong Kleisli lifting of the evaluation map $\mathrm{ev} : (X \to TY) \times X \to TY$ (modulo isomorphisms of the form $X \times Y \cong Y \times X$). Since every monad on Set is strong, we can always assume monads to come with a bind operator $\gg\!=$. Moreover, we will occasionally use the notation $\Lambda_\Sigma^{(\mathrm{v})}$ when we want to emphasise that we are working with Λ_Σ equipped with the call-by-value operational semantics.

Before giving Λ_Σ a call-by-name operational semantics, let us spell out some useful syntactical properties of $\Lambda_\Sigma^{(\mathrm{v})}$. Working with call-by-value effectful languages we are interested in substituting *values* for variables in terms, rather than terms themselves. In order to improve readability, we will sometimes denote a substitution $-[x := v]$ as a map σ and write e^σ in place of $e[x := v]$. We also notice that for any value w, w^σ is a value. The notion of substitution is extended to evaluation contexts in the obvious way (defining $[-]^\sigma \triangleq [-]$). Notice that for any evaluation context E, E^σ is an evaluation context as well.

Finally, we notice that we can always decompose the evaluation of a term of the form $e[x := v]$ as follows: we first evaluate e obtaining an element $[\![e]\!] \in T\mathcal{E}$, and then use monadic binding to apply the substitution $-[x := v]$ to enfs in $[\![e]\!]$ and evaluate the results obtained (this way composing the effects produced during the evaluation process).

Lemma 27 (Substitution Lemma for $\Lambda_\Sigma^{(v)}$). *Let σ be the substitution $-[y := w]$, and let us write $[\![\sigma]\!]$ for $[\![-[y := w]]\!]$. The following hold:*

$$[\![e^\sigma]\!]_n \sqsubseteq [\![e]\!]_n \gg\!= [\![\sigma]\!]_n \tag{7.1}$$

$$[\![e]\!]_n \gg\!= [\![\sigma]\!] \sqsubseteq [\![e^\sigma]\!] \tag{7.2}$$

$$[\![e^\sigma]\!] = ([\![e]\!] \gg\!= [\![\sigma]\!]). \tag{7.3}$$

Proof. We immediately notice that (7.3) is a corollary of (7.1) and (7.2). Both (7.1) and (7.2) are proved by induction on n proceeding by case analysis according to Lemma 25. We prove (7.1) by induction on n. The case for $n = 0$ is trivial. Assume $n > 0$ and proceed by case analysis according to Lemma 25.

- Suppose $e = v$. We have:

$$\begin{aligned}
[\![v^\sigma]\!]_n &= \eta(v^\sigma) \\
&\quad \text{[By Definition 45 since } v^\sigma \text{ is a value]} \\
&= \eta(v) \gg\!= [\![\sigma]\!]_n. \\
&\quad \text{[By Definition 45 since } \forall x \in X. \forall f : X \to TY.\ \eta(x) \gg\!= f = f(x)]
\end{aligned}$$

- Suppose $e = E[xv]$. We have:

$$\begin{aligned}
[\![e]\!]_n \gg\!= [\![\sigma]\!]_n &= \eta(E[xv]) \gg\!= [\![\sigma]\!]_n \\
&\quad \text{[By Definition 45]} \\
&= [\![(E[xv])^\sigma]\!]_n \\
&\quad \text{[Since } \forall x \in X. \forall f : X \to TY.\ \eta(x) \gg\!= f = f(x)] \\
&= [\![e^\sigma]\!]_n.
\end{aligned}$$

- Suppose $e = E[(\lambda x.f)v]$. First of all notice that for any term e, and all values v, w such that $x \neq y$ and $x \notin FV(w)$, we have (cf. Substitution Lemma in (Barendregt, 1984)):

$$e[x := v][y := w] = e[y := w][x := v[y := w]]. \tag{pure subst}$$

  Therefore, we can reason as follows:

$$\begin{aligned}
[\![(E[(\lambda x.f)v])^\sigma]\!]_n &= [\![E^\sigma[(\lambda x.f^\sigma)v^\sigma]]\!]_n \\
&\quad \text{[By definition of substitution]} \\
&= [\![E^\sigma[f^\sigma[x := v^\sigma]]]\!]_{n-1} \\
&\quad \text{[By Definition 45]} \\
&= [\![E^\sigma[(f[x := v])^\sigma]]\!]_{n-1} \\
&\quad \text{[By (pure subst). See also below.]} \\
&= [\![(E[f[x := v]])^\sigma]\!]_{n-1} \\
&\quad \text{[By definition of substitution]} \\
&\sqsubseteq [\![E[f[x := v]]]\!]_{n-1} \gg\!= [\![\sigma]\!]_{n-1} \\
&\quad \text{[By induction hypothesis]} \\
&\sqsubseteq [\![E[f[x := v]]]\!]_{n-1} \gg\!= [\![\sigma]\!]_n \\
&\quad \text{[By monotonicity of } \gg\!=] \\
&= [\![E[(\lambda x.f)v]]\!]_n \gg\!= [\![\sigma]\!]_n. \\
&\quad \text{[By Definition 45]}
\end{aligned}$$

In particular, we notice that the equality $f^\sigma[x := v^\sigma] = (f[x := v])^\sigma$ follows by (pure subst), since $f^\sigma[x := v^\sigma] = f[y := w][x := v[y := w]]$ and we can assume $x \notin FV(w)$ as the former is bound by a λ-abstraction.

- Suppose $e = E[\mathbf{op}(p, x.f)]$. We have

$$\begin{aligned}
[\![(E[\mathbf{op}(p, x.f)])^\sigma]\!]_n &= [\![E^\sigma[\mathbf{op}(p, x.f^\sigma)]]\!]_n \\
&\quad \text{[By definition of substitution]} \\
&= [\![\mathbf{op}]\!](p, v \mapsto [\![E^\sigma[f^\sigma[x := v]]]\!]_{n-1}). \\
&\quad \text{[By Definition 45]}
\end{aligned}$$

We now observe that since x is bound in $x.f$, without loss of generality we can assume $x \notin FV(w)$, and thus $w[x := v] = w$. As a consequence, we have:

$$\begin{aligned}
f^\sigma[x := v] &= f[y := w][x := v] \\
&= f[x := v][y := w[v/x]] \\
&\quad \text{[By (pure subst)]} \\
&= f[x := v][y := w] \\
&\quad \text{[Since } w[v/x] = w\text{]} \\
&= (f[x := v])^\sigma.
\end{aligned}$$

We can thus conclude our argument as follows:

$$\begin{aligned}
[\![(E[\mathbf{op}(p, x.f)])^\sigma]\!]_n &= [\![\mathbf{op}]\!](p, v \mapsto [\![E^\sigma[f^\sigma[x := v]]]\!].n - 1) \\
&\quad \text{[By previous argument]} \\
&= [\![\mathbf{op}]\!](p, v \mapsto [\![(E[f[x := v]])^\sigma]\!]_{n-1}) \\
&\quad \text{[By definition of substitution, since } f^\sigma[x := v] = (f[x := v])^\sigma\text{]} \\
&\sqsubseteq [\![\mathbf{op}]\!](p, v \mapsto ([\![E[f[x := v]]]\!]_{n-1} \gg\!= [\![\sigma]\!]_{n-1})) \\
&\quad \text{[By induction hypothesis]} \\
&\sqsubseteq [\![\mathbf{op}]\!](p, v \mapsto [\![E[f[x := v]]]\!]_{n-1}) \gg\!= [\![\sigma]\!]_{n-1} \\
&\quad \text{[By (gen alg op)]} \\
&\sqsubseteq [\![\mathbf{op}]\!](p, v \mapsto [\![E[f[x := v]]]\!]_{n-1}) \gg\!= [\![\sigma]\!]_n \\
&\quad \text{[By monotonicity of } \gg\!=\text{]} \\
&= [\![E[\mathbf{op}(p, x.f)]]\!]_n \gg\!= [\![\sigma]\!]_n. \\
&\quad \text{[By Definition 45]}
\end{aligned}$$

□

Before giving Λ_Σ a call-by-name operational semantics, we make the following easy but useful observation.

Lemma 28. *If $e[x := v]$ is a value, then e is a value. Similarly, if $e[x := v]$ is a enf, then e is a enf. Moreover, if $e[x := v]$ is a λ-free enf (i.e. not an abstraction), then so is e.*

Proof. We immediately notice that if e is a variable, then we trivially have the thesis. Suppose e is not a variable. We proceed by induction on $e^\sigma = e[x := v]$. If $e^\sigma = \lambda y.f$, then since e is not a variable we must have $e = \lambda y.g$ with $g^\sigma = f$, and we are trivially done. If $e^\sigma = E[yw]$, then we proceed by case analysis on E.

- Suppose $E = [-]$. Then e must be of the form $e_1 e_2$ with $e_1^\sigma = y$ and $e_2^\sigma = w$. By induction hypothesis we have that e_2 is a value. Moreover, $e_1^\sigma = y$ implies that e_1 must be a variable. We are done.
- Suppose $E = Ff$. Then e is of the form $e_1 e_2$ with $e_1^\sigma = F[yw]$ and $e_2^\sigma = f$. By induction hypothesis e_1 is a λ-free enf. Moreover, e_1 cannot be a variable, and thus it must be of the form $F'[zu]$. We conclude $e_1 e_2 = F'[zu]e_2$, and thus we are done.
- Suppose $E \equiv wF$. As above.

□

7.2.2 Call-by-Name Operational Semantics

We now give Λ_Σ call-by-name monadic operational semantics. In order to do so, we first need to change our notion of evaluation context. As in Chapter 6, call-by-name evaluation contexts are defined by the following grammar, but contrary to Chapter 6 here we do not require e to be closed:

$$E ::= [-] \mid Ee.$$

As usual, we write $E[e]$ for the term obtained by substituting the term e for the hole $[-]$ in E. Redexes are expressions of the form $(\lambda x.e)f$ or $\mathbf{op}(p, x.e)$, whereas a *stuck term* is a term of the form $E[xe]$. In particular, stuck terms are expressions of the form $xe_0 \cdots e_k$. Following (S. B. Lassen, 1999; Sangiorgi, 1994) we define the collection $\mathcal{W}$ of *weak head normal forms* (whnfs hereafter) as the collection of values and stuck terms, and let letters $s, t, \ldots$ to range over whnfs. Notice that whnfs are generated by the following grammar:

$$s ::= x \mid \lambda x.e \mid xe_1 \cdots e_n.$$

As for the call-by-value case, we can easily prove the following unique decomposition lemma.

Lemma 29. *Any term e can be uniquely decomposed in one of the following (mutually exclusive) forms:*

1. v;
2. $E[ve]$;
3. $E[\mathbf{op}(p, x.f)]$;

Call-by-name operational semantics is defined with respect to a Σ-continuous monad $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ relying on Lemma 29. More precisely, we define a *call-by-name* evaluation function $[\![-]\!]$ mapping each term to an element in $T\mathcal{W}$.

Definition 46. *Define the $\mathbb{N}$-indexed family of maps $[\![-]\!]_n : \Lambda \to T\mathcal{W}$ as follows:*

$$\begin{aligned}
[\![e]\!]_0 &\triangleq \bot \\
[\![v]\!]_{n+1} &\triangleq \eta(v) \\
[\![E[xe]]\!]_{n+1} &\triangleq \eta(E[xe]) \\
[\![E[(\lambda x.f)e]]\!]_{n+1} &\triangleq [\![E[f[x := e]]]\!]_n \\
[\![E[\mathbf{op}(p, x.e)]]\!]_{n+1} &\triangleq [\![\mathbf{op}]\!]_{\mathcal{W}}(p, v \mapsto [\![e[x := v]]\!]_n).
\end{aligned}$$

The monad $\mathbb{T}$ being Σ-continuous, we see that the sequence $([\![e]\!]_n)_n$ forms an ω-chain in $T\mathcal{W}$, so that we can define $[\![e]\!]$ as $\bigsqcup_n [\![e]\!]_n$.

As usual, Σ-continuity of $\mathbb{T}$ ensures continuity of the evaluation function.

Lemma 30. *The map $[\![-]\!]$ is the least function $\varphi : \Lambda \to T\mathcal{W}$ satisfying the following identities:*

$$\begin{aligned}
\varphi(v) &= \eta(v) \\
\varphi(E[xe]) &= \eta(E[xe]) \\
\varphi(E[(\lambda x.f)e]) &= \varphi(E[f[x := e]]) \\
\varphi(E[\mathbf{op}(p, x.e)) &= [\![\mathbf{op}]\!]_{\mathcal{W}}(p, v \mapsto \varphi(E[e[x := v]])).
\end{aligned}$$

Following the convention introduced in previous section, we occasionally use the notation $\Lambda_\Sigma^{(\mathsf{n})}$ when we want to emphasise that we are working with Λ_Σ equipped with the call-by-name operational semantics. It is a routine exercise to show that the analogue of Lemma 27 holds for $\Lambda_\Sigma^{(\mathsf{n})}$.

Lemma 31 (Substitution Lemma for $\Lambda_\Sigma^{(\mathsf{n})}$)**.** *Let σ be the substitution $-[y := e']$, and let us write $[\![\sigma]\!]$ for $[\![-[y := e']]\!]$. The following hold:*

$$\begin{aligned}
[\![e^\sigma]\!]_n &\sqsubseteq [\![e]\!]_n \gg\!= [\![\sigma]\!]_n \\
[\![e]\!]_n \gg\!= [\![\sigma]\!] &\sqsubseteq [\![e^\sigma]\!] \\
[\![e^\sigma]\!] &= ([\![e]\!] \gg\!= [\![\sigma]\!]).
\end{aligned}$$

The proof of Lemma 31 closely follows the one of Lemma 27, but it is easier due to simpler form of call-by-name evaluation contexts, and thus it is omitted.

In order to compare the behaviour of terms of Λ_Σ we now introduce *effectful normal form similarity* and *bisimilarity*. The expression normal form (bi)similarity actually denotes two distinct notions, namely *eager normal form (bi)similarity* (S. B. Lassen, 2005) and *weak head normal form (bi)similarity* (also known as open (bi)similarity) (S. B. Lassen, 1999; Sangiorgi, 1994). As the names suggest, eager normal form (bi)similarity is nothing but the instantiation of the (informal) notion of normal form (bi)similarity to *call-by-value* calculi, whereas weak head normal form (bi)similarity is the instantiation of the (informal) notion of normal form (bi)similarity to *call-by-name* calculi.

As for monadic and effectful applicative (bi)similarity, our notion of effectful normal form (bi)similarity is parametrised by a relator, whereby computational effects are taken into account.

7.3 Effecful Normal Form Similarity and Bisimilarity

In the rest of this section, let a Σ-continuous monad $\mathbb{T} = \langle T, \eta, -^\dagger \rangle$ and a Σ-continuous relator Γ for it be fixed. As usual, Σ-continuity of Γ is not required for defining effectful normal form (bi)simulation, but it is central to prove that the induced notion of similarity and bisimilarity are precongruence and congruence relations, respectively. We begin with eager normal form (bi)similarity.

7.3.1 Eager Normal Form Similarity and Bisimilarity

In the rest of this subsection, we tacitly assume to work with the calculus $\Lambda_\Sigma^{(\mathsf{v})}$. Although we are working with a coarse-grain calculus, in an effectful setting it is important to distinguish between relations over *terms* and relation over *eager normal forms*. For that reason we we will work with a variation of λ-term relations.

Definition 47. *A λ-term relation (term relation, for short) $\mathcal{R}$ is a pair $(\mathcal{R}_\Lambda : \Lambda \twoheadrightarrow \Lambda, \mathcal{R}_\mathcal{E} : \mathcal{E} \twoheadrightarrow \mathcal{E})$.*

The collection of λ-term relations is the set $\mathsf{Rel}(\Lambda,\Lambda)\times\mathsf{Rel}(\mathcal{E},\mathcal{E})$ which inherits a complete lattice structure from $\mathsf{Rel}(\Lambda,\Lambda)$ and $\mathsf{Rel}(\mathcal{E},\mathcal{E})$ pointwise, hence allowing λ-term relations to be defined both inductively and coinductively. We use these properties to define our notion of effectful eager normal form similarity.

Definition 48. *A term relation $\mathcal{R} = (\mathcal{R}_\Lambda : \Lambda \nrightarrow \Lambda, \mathcal{R}_\mathcal{E} : \mathcal{E} \nrightarrow \mathcal{E})$ is an* effectful eager normal form simulation *with respect to Γ (hereafter enf-simulation, as Γ will be clear from the context) if the following hold.*

$$e\,\mathcal{R}_\Lambda\,f \implies [\![e]\!]\,\Gamma\mathcal{R}_\mathcal{E}\,[\![f]\!] \qquad \text{(enf comp)}$$

$$x\,\mathcal{R}_\mathcal{E}\,s \implies s = x \qquad \text{(enf var)}$$

$$\lambda x.e\,\mathcal{R}_\mathcal{E}\,s \implies \exists f.\, s = \lambda x.f \wedge e\,\mathcal{R}_\Lambda\,f \qquad \text{(enf abs)}$$

$$E[xv]\,\mathcal{R}_\mathcal{E}\,s \implies \exists E', v'.\, s = E'[xv'] \wedge v\,\mathcal{R}_\mathcal{E}\,v' \wedge \exists z \notin FV(E)\cup FV(E').\, E[z]\,\mathcal{R}_\Lambda\,E'[z]. \qquad \text{(enf stuck)}$$

We say that relation $\mathcal{R}$ respects enfs *if it satisfies conditions* (enf var), (enf abs), *and* (enf stuck).

Definition 48 is quite standard. Clause (enf comp) is essentially as (app comp). Clauses (enf var)-(enf stuck) state that whenever two enfs are related by $\mathcal{R}_\mathcal{E}$ then they must have the same syntactic structure. For instance, if $\lambda x.e\,\mathcal{R}_\mathcal{E}\,s$ holds, then s must the a λ-abstraction, i.e. an expression of the form $\lambda x.f$. Additionally, e and f must be related by $\mathcal{R}_\Lambda$.

Clause (enf stuck) is the most interesting one. It states that whenever $E[xv]\,\mathcal{R}_\mathcal{E}\,s$, then s must be a stuck term $E'[xv']$, for some evaluation context E' and value v'. Notice that $E[xv]$ and s must have the same 'stuck variable' x. Additionally, v and v' must be related by $\mathcal{R}_\mathcal{E}$, and E and E' must be properly related too. The idea is that to see whether E and E' are related, we replace the stuck expressions xv, xv' with a fresh variable z, and test $E[z]$ and $E'[z]$ (thus resuming the evaluation process). We require $E[z]\,\mathcal{R}_\mathcal{E}\,E'[z]$ to hold, for *some* fresh variable z. The choice of the variable does not really matter, provided it is fresh. In fact, as we will see, enf-similarity $\preceq^{\mathsf{E}}$ is substitutive and reflexive. In particular, if $E[z]\preceq^{\mathsf{E}}_\mathcal{E} E'[z]$ holds, then $E[y]\preceq^{\mathsf{E}}_\mathcal{E} E'[y]$ holds as well, for any fresh variable $y \notin FV(E)\cup FV(E')$.

Notice that Definition 48 does not involve any universal quantification. In particular, enfs are tested by inspecting their syntactic structure, thus making the definition of an enf-simulation somehow 'local': terms are tested in isolation and not via their interaction with the environment[3].

Remark 15. Definition 48 does not require for an enf-simulation $\mathcal{R}$ to have $\mathcal{R}_\mathcal{E}\subseteq\mathcal{R}_\Lambda$. Nonetheless, thanks to condition (rel unit) we can assume the latter to be case, without loss of generality. In fact, we can always *extend* $\mathcal{R}_\Lambda$ to a relation $\mathcal{S}_\Lambda$ such that $(\mathcal{S}_\Lambda,\mathcal{R}_\mathcal{E})$ is an enf-simulation. For that, it is sufficient to define $\mathcal{S}_\Lambda \triangleq \mathcal{R}_\Lambda \cup \{(s,s') \mid s\,\mathcal{R}_\mathcal{E}\,s'\}$. To see that we indeed obtain an enf-simulation it is sufficient to prove (enf comp), which amounts to show $\eta(s)\,\Gamma\mathcal{R}_\mathcal{E}\,\eta(s')$. That is indeed the case by condition (rel unit), since $s\,\mathcal{R}_\mathcal{E}\,s'$.

Definition 48 induces an endofunction $\mathcal{R}\mapsto[\mathcal{R}]$ on the complete lattice $\mathsf{Rel}(\Lambda,\Lambda)\times\mathsf{Rel}(\mathcal{E},\mathcal{E})$, where $[\mathcal{R}] = ([\mathcal{R}]_\Lambda,[\mathcal{R}]_\mathcal{E})$ is defined as follows (here $\mathsf{I}_\mathcal{X}$ denotes the identity relation on variables, i.e. the set of pairs of the form (x,x)):

$$[\mathcal{R}]_\Lambda \triangleq \{(e,f) \mid [\![e]\!]\,\Gamma\mathcal{R}_\mathcal{E}\,[\![f]\!]\}$$

$$[\mathcal{R}]_\mathcal{E} \triangleq \mathsf{I}_\mathcal{X} \cup \{(\lambda x.e, \lambda x.f) \mid e\,\mathcal{R}_\Lambda\,f\} \cup \{(E[xv], E'[xv']) \mid v\,\mathcal{R}_\mathcal{E}\,v' \wedge \exists z\notin FV(E)\cup FV(E').\, E[z]\,\mathcal{R}_\Lambda\,E'[z]\}.$$

It is easy to see that a term relation $\mathcal{R}$ is an enf-simulation if and only if $\mathcal{R}\subseteq[\mathcal{R}]$. Notice also that although $[\mathcal{R}]_\mathcal{E}$ always contains the identity relation on variables, $\mathcal{R}_\mathcal{E}$ does not have to: the empty relation $(\emptyset,\emptyset)$ is an enf-simulation.

[3] In the case of applicative simulation such interaction is expressed by clause (app val) in Definition 32: the environment interacts with tested values by passing them arbitrary closed values as arguments.

Lemma 32. *The mapping $\mathcal{R} \mapsto [\mathcal{R}]$ is monotone.*

Proof. Suppose $\mathcal{R} \subseteq \mathcal{S}$. If $e\ [\mathcal{R}]_\Lambda\ e'$, then we have $[\![e]\!]\ \Gamma\mathcal{R}_\mathcal{E}\ [\![e']\!]$. Since $\mathcal{R} \subseteq \mathcal{S}$ and Γ is monotone, we conclude $[\![e]\!]\ \Gamma\mathcal{S}_\mathcal{E}\ [\![e']\!]$. If $s\ [\mathcal{R}]_\mathcal{E}\ s'$, then we have three possible cases to consider.

- If $(s, s') = (x, x)$, for some variable x, then we trivially have $s\ [\mathcal{R}]_\mathcal{E}\ s'$.
- If $(s, s') = (\lambda x.e, \lambda x.e')$ with $e\ \mathcal{R}_\Lambda\ e'$, then since $\mathcal{R} \subseteq \mathcal{S}$ we infer $e\ \mathcal{S}_\Lambda\ e'$, and thus $\lambda x.e\ [\mathcal{S}]_\mathcal{E}\ \lambda x.e'$.
- If $(s, s') = (E[xv'], E'[xv'])$ with $v\ \mathcal{R}_\mathcal{E}\ v'$ and $E[z]\ \mathcal{R}_\Lambda\ E[z]$, for some fresh variable z, then $\mathcal{R} \subseteq \mathcal{S}$ implies $v\ \mathcal{S}_\mathcal{E}\ v'$ and $E[z]\ \mathcal{S}_\Lambda\ E'[z']$. We conclude $E[xv]\ [\mathcal{S}]_\mathcal{E}\ E'[xv']$.

□

By Knaster-Tarski Theorem, $\mathcal{R} \mapsto [\mathcal{R}]$ has a greatest fixed point which we call *effectful eager normal form similarity* with respect to Γ (hereafter enf-similarity) and denote by $\preceq^{\mathsf{E}} = (\preceq^{\mathsf{E}}_\Lambda, \preceq^{\mathsf{E}}_\mathcal{E})$. Notice that, as usual, we are applying the notational convention of Remark 9.

Enf-similarity is thus the largest enf-simulation with respect to Γ. Moreover, $\preceq^{\mathsf{E}}$ being defined coinductively, it comes with an associated coinduction proof principle:

$$\frac{\mathcal{R} \subseteq [\mathcal{R}]}{\mathcal{R} \subseteq \preceq^{\mathsf{E}}}\ (\preceq^{\mathsf{E}}\text{-coind.})$$

Example 45. Recall the probabilistic call-by-value fixed point combinators of Example 42:

$$Y \triangleq \lambda y.\Delta\Delta \qquad Z \triangleq \Theta\Theta$$
$$\Delta \triangleq \lambda x.y(\lambda z.xxz) \qquad \Theta \triangleq \lambda x.\lambda y.(y(\lambda z.\Delta\Delta z)\ \mathbf{or}\ y(\lambda z.xxyz)).$$

Let us consider the relator $\hat{\mathbb{D}\mathbb{M}}$ for probabilistic partial computations. We show $Y \preceq^{\mathsf{E}} Z$ by coinduction, proving that the term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{E})$ whose graph is defined as follows[4] is an enf-simulation (where I_Λ and $\mathsf{I}_\mathcal{E}$ denote the identity relations on terms and enfs, respectively):

$$\begin{aligned}
\mathcal{R}_\Lambda &\triangleq \{(Y, Z), (\Delta\Delta z, Zyz), (\Delta\Delta, y(\lambda z.\Delta\Delta z)\ \mathbf{or}\ y(\lambda z.Zyz))\} \cup \mathsf{I}_\Lambda \\
\mathcal{R}_\mathcal{E} &\triangleq \{(y(\lambda z.\Delta\Delta z), y(\lambda z.Zyz)), (\lambda z.\Delta\Delta z, \lambda z.Zyz), \\
&\qquad (\lambda y.\Delta\Delta, \lambda y.(y(\lambda z.\Delta\Delta z)\ \mathbf{or}\ y(\lambda z.Zyz))), (y(\lambda z.\Delta\Delta z)z, y(\lambda z.Zyz)z)\} \\
&\qquad \cup \mathsf{I}_\mathcal{E}
\end{aligned}$$

The term relation $\mathcal{R}$ is obtained by starting with the relation $\{(Y, Z)\}$ and progressively adding terms and eager normal forms going through clauses (enf comp)-(enf stuck) in Definition 48. Checking that $\mathcal{R}$ is an enf-simulation is straightforward. We show a couple of cases as illustrative examples.

1. We prove that $\Delta\Delta z\ \mathcal{R}_\Lambda\ Zyz$ implies $[\![\Delta\Delta z]\!]\ \hat{\mathbb{D}\mathbb{M}}(\mathcal{R}_\mathcal{E})\ [\![Zyz]\!]$. The latter amounts to show:

$$\left(1 \cdot \mathit{just}\ y(\lambda z.\Delta\Delta z)z\right)\ \hat{\mathbb{D}\mathbb{M}}(\mathcal{R}_\mathcal{E})\ \left(\frac{1}{2} \cdot \mathit{just}\ y(\lambda z.\Delta\Delta z)z + \frac{1}{2} \cdot \mathit{just}\ y(\lambda z.Zyz)z\right), \tag{7.4}$$

   where, as usual, we write distributions as weighted formal sums. To prove (7.4), is is sufficient to find a suitable coupling of $[\![\Delta\Delta z]\!]$ and $[\![Zyz]\!]$. Define the distribution $\omega \in D(M\mathcal{E} \times M\mathcal{E})$ as follows:

$$\omega(\mathit{just}\ y(\lambda z.\Delta\Delta z)z, \mathit{just}\ y(\lambda z.\Delta\Delta z)z) = \frac{1}{2}$$
$$\omega(\mathit{just}\ y(\lambda z.\Delta\Delta z)z, \mathit{just}\ y(\lambda z.Zyz)z) = \frac{1}{2},$$

[4] For readability we write $\mathcal{R}$ in place of $G_\mathcal{R}$.

and assigning zero to all other pairs in $M\mathcal{E} \times M\mathcal{E}$. Obviously ω is a coupling of $[\![\Delta\Delta z]\!]$ and $[\![Zyz]\!]$. Additionally, we see that $\omega(\mathfrak{x}, \mathfrak{y})$ implies $\mathfrak{x}\ \hat{\mathbb{M}}\mathcal{R}_{\mathcal{E}}\ \mathfrak{y}$, since

$$y(\lambda z.\Delta\Delta z)z\ \mathcal{R}_{\mathcal{E}}\ y(\lambda z.\Delta\Delta z)z$$
$$y(\lambda z.\Delta\Delta z)z\ \mathcal{R}_{\mathcal{E}}\ y(\lambda z.Zyz)z$$

hold.

2. We prove that $y(\lambda z.\Delta\Delta z)z\ \mathcal{R}_{\mathcal{E}}\ y(\lambda z.Zyz)z$ implies $\lambda z.\Delta\Delta z\ \mathcal{R}_{\mathcal{E}}\ \lambda z.Zyz$ and $z_0 z\ \mathcal{R}_{\Lambda}\ z_0 z$, where z_0 is a (fresh) variable. The former holds by very definition of $\mathcal{R}_{\mathcal{E}}$, whereas the latter holds since $\mathsf{I}_{\Lambda} \subseteq \mathcal{R}_{\Lambda}$.

As already discussed in Example 42, the refinement $Y \preceq^{\mathsf{E}} Z$ is an example of a refinement that cannot be readily established using e.g. the operational methods developed in Chapter 5 (notably CIU approximation and applicative similarity), but whose proof is straightforward using enf-similarity. Additionally, considering the symmetric closure of $\mathcal{R}$ we can convince ourselves that Y and Z are actually enf-bisimilar, although such a notion has not been formally defined yet (see Definition 49).

Finally, in Subsection 7.4.2 we will prove an *up-to context technique* which will allow to reduce the size of $\mathcal{R}$, thus minimising the task of checking that our relation is indeed an enf-simulation. To the best of the author's knowledge, the probabilistic instance of enf-similarity is the first example of a *probabilistic normal form similarity* in the literature. ⊠

We conclude this subsection defining effectful eager normal form bisimilarity with respect to a relator Γ following the same patter of Definition 34.

Definition 49. *A term relation $\mathcal{R}$ is an* effectful eager normal form bisimulation with respect to Γ *(enf-bisimulation, for short) if it is an enf-simulation with respect to $\Gamma \wedge \Gamma^{\circ}$.* Eager normal bisimilarity with respect to Γ *(hereafter enf-bisimilarity) $\simeq^{\mathsf{E}}$ is defined as eager normal similarity with respect to $\Gamma \wedge \Gamma^{\circ}$.*

As usual, a routine proof by coinduction gives us the following characterisation of enf-bisimilarity.

Proposition 20. *Enf-bisimilarity $\simeq^{\mathsf{E}}$ is the largest symmetric enf-simulation with respect to Γ.*

Before proving our main result, namely that $\preceq^{\mathsf{E}}$ is a precongrunce (and that $\simeq^{\mathsf{E}}$ is a congruence), we introduce the notions of *effectful weak head normal form similarity* and *effectful weak head normal form bisimilarity.*

7.3.2 Weak Head Normal Form Similarly and Bisimilarity

Effectful weak head normal form (bi)similarity is the call-by-name counterpart of effectful eager normal form (bi)similarity. In the rest of this subsection we work with $\Lambda_{\Sigma}^{(\mathsf{n})}$. We start by defining the notion of λ-term relation for $\Lambda_{\Sigma}^{(\mathsf{n})}$

Definition 50. *A λ-term relation (term relation, for short) is a pair $\mathcal{R} = (\mathcal{R}_{\Lambda} : \Lambda \nrightarrow \Lambda, \mathcal{R}_{\mathcal{W}} : \mathcal{W} \nrightarrow \mathcal{W})$.*

Definition 51. *A term relation $\mathcal{R} = (\mathcal{R}_{\Lambda} : \Lambda \nrightarrow \Lambda, \mathcal{R}_{\mathcal{W}} : \mathcal{W} \nrightarrow \mathcal{W})$ is a* weak head normal form simulation *with respect to Γ (hereafter whnf-simulation) if:*

$$e\ \mathcal{R}_{\Lambda}\ f \implies [\![e]\!]\ \Gamma\mathcal{R}_{\mathcal{W}}\ [\![f]\!] \qquad \text{(whnf comp)}$$
$$\lambda x.e\ \mathcal{R}_{\mathcal{W}}\ s \implies \exists f.\ s = \lambda x.f \wedge e\ \mathcal{R}_{\Lambda}\ f \qquad \text{(whnf abs)}$$
$$xe_0 \cdots e_n\ \mathcal{R}_{\mathcal{W}}\ s \implies \exists f_0, \ldots, f_k.\ s = x f_0 \cdots f_k \wedge \forall i.\ e_i\ \mathcal{R}_{\Lambda}\ f_i. \qquad \text{(whnf stuck)}$$

We say that $\mathcal{R}$ respects whnfs *if it satisfies clauses* (whnf abs) *and* (whnf stuck).

As for eager normal form similarity, Definition 51 induces a monotone endofunction $\mathcal{R} \mapsto [\mathcal{R}]$ on the complete lattice $\mathrm{Rel}(\Lambda, \Lambda) \times \mathrm{Rel}(\mathcal{W}, \mathcal{W})$. We can characterise whnf-simulations as post-fixed points of $\mathcal{R} \mapsto [\mathcal{R}]$ and thus define *weak head normal form similarity with respect to* Γ (whnf-similarity, for short) $\preceq^{\mathrm{W}}$ as the greatest fixed point of $\mathcal{R} \mapsto [\mathcal{R}]$.

Example 46. Recall Example 43. We prove $P \preceq^{\mathrm{W}} Q$ by coinduction, showing that the term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{W})$ (whose graph is) defined as follows is a whnf-simulation:

$$\mathcal{R}_\Lambda \triangleq \{(P, Q), (y(Pyz) \textbf{ or } z(Pzy), y(Qzy) \textbf{ or } z(Qyz)), (Pyz, Qzy), (Pzy, Qyz)\}$$

$$\mathcal{R}_\mathcal{W} \triangleq \{(\lambda y.\lambda z.y(Pyz) \textbf{ or } z(Pzy), \lambda y.\lambda z.y(Qzy) \textbf{ or } z(Qyz)), (\lambda z.y(Pyz) \textbf{ or } z(Pzy), \lambda z.y(Qzy) \textbf{ or } z(Qyz))$$
$$(y(Pyz), y(Qzy)), (z(Pzy), z(Qyz))\}.$$

The relation $\mathcal{R}$ is built starting from $\{(P, Q)\}$ by progressively adding terms and whnfs going through clauses (whnf comp)-(whnf stuck) in Definition 51. Although straightforward, checking that the relation $\mathcal{R}$ is indeed a whnf simulation is an annoying work. In Subsection 7.4.2 we will use an up-to context technique to prove that $P \preceq^{\mathrm{W}} Q$ with minimum effort. Finally, we notice that we can show that P and Q are actually equivalent by taking the symmetric closure of $\mathcal{R}$. ⊠

Following Definition 49, we define *weak head normal bisimilarity with respect to* Γ (whnf-bisimilarity, for short), denoted by $\simeq^{\mathrm{W}}$, as weak head normal form similarity with respect to $\Gamma \wedge \Gamma^\circ$. Without surprise, $\preceq^{\mathrm{W}}$ can be characterised as the largest symmetric whnf-simulation with respect to Γ.

Proposition 21. *Weak head normal form bisimilarity with respect to* Γ *is the largest weak head normal form simulation with respect to* Γ.

Previous examples give evidences that normal form similarity and bisimilarity are interesting notions of program approximation and equivalence, respectively, even in presence of effects. However, in order to qualify as such, we first have to prove precongruence and congruence theorems for them. The next section is dedicated to such a task.

7.4 Meta-theoretical Properties

In this section we prove congruence and precongruence theorems for normal form bisimilarity and similarity, respectively. Our proofs are based on a generalisation of Lassen's relational construction for the pure call-by-name λ-calculus (S. B. Lassen, 1999) to both $\Lambda_\Sigma^{(\mathrm{n})}$ and $\Lambda_\Sigma^{(\mathrm{v})}$. Such a construction has been previously adapted to the *pure* call-by-value λ-calculus (and its extension with delimited control operators) in (Biernacki, Lenglet, & Polesiuk, 2018), whereas Lassen has proved compatibility of pure eager normal form bisimilarity via a CPS translation (S. B. Lassen, 2005). Both those proofs rely on syntactical properties of the calculus (mostly expressed using a suitable small-step semantics), and thus seem to be hardly adaptable to effectful calculi[5]. On the contrary, our proofs rely on the properties of relators, thereby making our results and techniques more modular and thus valid for a large class of effects.

7.4.1 Congruence and Precongruence Theorems

We begin proving precongruence of enf-similarity, as proving the latter is much harder than proving precongruence of whnf-similarity. As a consequence, unless explicitly stated, in the rest of this section

[5]Lassen (S. B. Lassen, 2006b) studied weak head normal form (bi)similarity for a nondeterministic call-by-name λ-calculus. However, his treatment uses specific features of nondeterminism and, to the best of the author's knowledge, is not extended to the call-by-value case.

we work with the calculus $\Lambda_\Sigma^{(v)}$. As usual, we tacitly assume a Σ-continuous monad $\mathbb{T}$ and a Σ-continuous relator Γ for $\mathbb{T}$ to be fixed. Moreover, in order to have a lighter notation, we reintroduce our sequent-like notation for term relations. In particular, for a term relation $\mathcal{R} = (\mathcal{R}_\Lambda, \mathcal{R}_\mathcal{E})$ and arbitrary terms e, f and values v, w, we use the notations $\vdash^\Lambda e\ \mathcal{R}\ f$ and $e\ \mathcal{R}_\Lambda\ e$, as well as $\vdash^\mathcal{E} e\ \mathcal{R}\ f$ and $e\ \mathcal{R}_\mathcal{E}\ f$, interchangeably. Notice, however, that in writing e.g. $\vdash^\Lambda e\ \mathcal{R}\ f$ we are *not* requiring e and f to be closed terms.

The main (new) tool we will use to prove the wished precongruence theorem is the so-called *substitutive and compatible closure* of a term relation (S. B. Lassen, 1999).

Definition 52. *Let $\mathcal{R}$ be a term relation. The* substitutive and compatible closure *(sc-closure, for short) of $\mathcal{R}$ is the term relation $\mathcal{R}^{\mathrm{SC}}$ defined inductively according to the rules in Figure 7.2, where in rules ($\mathcal{E}$-sc.E) and (Λ-sc.E) z is a fresh variable.*

$$\frac{\vdash^\Lambda e\ \mathcal{R}\ e'}{\vdash^\Lambda e\ \mathcal{R}^{\mathrm{SC}}\ e'}\ (\Lambda\text{-sc}.\mathcal{R}) \qquad \frac{\vdash^\mathcal{E} s\ \mathcal{R}\ s'}{\vdash^\mathcal{E} s\ \mathcal{R}^{\mathrm{SC}}\ s'}\ (\mathcal{E}\text{-sc}.\mathcal{R}) \qquad \frac{\vdash^\mathcal{E} s\ \mathcal{R}^{\mathrm{SC}}\ s'}{\vdash^\Lambda s\ \mathcal{R}^{\mathrm{SC}}\ s'}\ (\mathcal{E}\text{-}\Lambda\text{-sc})$$

$$\frac{}{\vdash^\mathcal{E} x\ \mathcal{R}^{\mathrm{SC}}\ x}\ (\mathcal{E}\text{-sc.var})$$

$$\frac{\vdash^\Lambda E[z]\ \mathcal{R}^{\mathrm{SC}}\ E'[z] \quad \vdash^\mathcal{E} v\ \mathcal{R}^{\mathrm{SC}}\ v'}{\vdash^\mathcal{E} E[xv]\ \mathcal{R}^{\mathrm{SC}}\ E[xv']}\ (\mathcal{E}\text{-sc.E}) \qquad \frac{\vdash^\Lambda E[z]\ \mathcal{R}^{\mathrm{SC}}\ E'[z] \quad \vdash^\Lambda e\ \mathcal{R}^{\mathrm{SC}}\ e'}{\vdash^\Lambda E[e]\ \mathcal{R}^{\mathrm{SC}}\ E'[e']}\ (\Lambda\text{-sc.E})$$

$$\frac{\vdash^\Lambda e\ \mathcal{R}^{\mathrm{SC}}\ f}{\vdash^\mathcal{E} \lambda x.e\ \mathcal{R}^{\mathrm{SC}}\ \lambda x.f}\ (\Lambda\text{-sc.abs}) \qquad \frac{\vdash^\Lambda e\ \mathcal{R}^{\mathrm{SC}}\ e' \quad \vdash^\Lambda f\ \mathcal{R}^{\mathrm{SC}}\ f'}{\vdash^\Lambda ef\ \mathcal{R}^{\mathrm{SC}}\ e'f'}\ (\Lambda\text{-sc.app})$$

$$\frac{\vdash^\mathcal{E} v\ \mathcal{R}^{\mathrm{SC}}\ v' \quad \vdash^\mathcal{E} w\ \mathcal{R}^{\mathrm{SC}}\ w'}{\vdash^\mathcal{E} v[x := w]\ \mathcal{R}^{\mathrm{SC}}\ v'[x := w']}\ (\mathcal{E}\text{-sc.subst}) \qquad \frac{\vdash^\Lambda e\ \mathcal{R}^{\mathrm{SC}}\ e' \quad \vdash^\mathcal{E} v\ \mathcal{R}^{\mathrm{SC}}\ v'}{\vdash^\Lambda e[x := v]\ \mathcal{R}^{\mathrm{SC}}\ e'[x := v']}\ (\Lambda\text{-sc.subst})$$

$$\frac{\vdash^\Lambda e\ \mathcal{R}^{\mathrm{SC}}\ f}{\vdash^\Lambda \mathbf{op}(p, x.e)\ \mathcal{R}^{\mathrm{SC}}\ \mathbf{op}(p, x.f)}\ (\Lambda\text{-sc.op})$$

Figure 7.2: Compatible and substitutive closure construction for $\Lambda_\Sigma^{(v)}$.

We say that term relation $\mathcal{R}$ is *compatible* and *substitutive* if $\mathcal{R} = \mathcal{R}^{\mathrm{SC}}$. This definition obviously agrees with the definitions of substitutivity and compatibility given in Chapter 5.

We immediately notice that thanks to rules (Λ-sc.$\mathcal{R}$) and ($\mathcal{E}$-sc.$\mathcal{R}$), for any term relation $\mathcal{R}$, we have $\mathcal{R} \subseteq \mathcal{R}^{\mathrm{SC}}$, and that the latter is always reflexive. As a consequence, in order to show that a term relation $\mathcal{R}$ is compatible it is sufficient to prove $\mathcal{R}^{\mathrm{SC}} \subseteq \mathcal{R}$. We are now going to prove that if $\mathcal{R}$ is a enf-simulation, then so is $\mathcal{R}^{\mathrm{SC}}$. In particular, we will infer that $(\preceq^{\mathrm{E}})^{\mathrm{SC}}$ is a enf-simulation, and thus it is contained in $\preceq^{\mathrm{E}}$, by coinduction. That allows us to conclude that enf-similarity is compatible and substitutive. For readability, we split the proof in two parts.

Lemma 33. *If $\mathcal{R}$ respects enfs, then so does $\mathcal{R}^{\mathrm{SC}}$.*

Proof. Suppose $\mathcal{R}$ respects enfs. We prove that $\mathcal{R}^{\mathrm{SC}}$ satisfies conditions (enf var), (enf abs), and (enf stuck) in Definition 48.

- We show that $\mathcal{R}^{\mathrm{SC}}$ satisfies (enf var). That is, we prove:

  $$\vdash^\mathcal{E} x\ \mathcal{R}^{\mathrm{SC}}\ s \implies s = x.$$

  We proceed by induction on the derivation of $\vdash^\mathcal{E} x\ \mathcal{R}^{\mathrm{SC}}\ s$. The only relevant cases to consider are those relatives to rules ($\mathcal{E}$-sc.$\mathcal{R}$), ($\mathcal{E}$-sc.var), and ($\mathcal{E}$-sc.subst). The first two are trivial. For the

latter, suppose $\vdash^{\mathcal{E}} x \, \mathcal{R}^{\text{SC}} \, s$ to be the conclusion of a derivation of the form:

$$\frac{\vdash^{\mathcal{E}} w \, \mathcal{R}^{\text{SC}} \, w' \quad \vdash^{\mathcal{E}} v \, \mathcal{R}^{\text{SC}} \, v'}{\vdash^{\mathcal{E}} w[y := v] \, \mathcal{R}^{\text{SC}} \, w'[y := v']} \; (\mathcal{E}\text{-sc.subst})$$

so that we have $x = w[y := v]$ and $s = w'[y := v']$. As a consequence, w must be a variable. If $w \neq y$, then we must have $w = x$ and we conclude the thesis by induction hypothesis on $\vdash^{\mathcal{E}} w \, \mathcal{R}^{\text{SC}} \, w'$. If $w = y$, then we must have $v = x$. By induction hypothesis on $\vdash^{\mathcal{E}} w \, \mathcal{R}^{\text{SC}} \, w'$ we infer $w' = y$ too, whereas from induction hypothesis on $\vdash^{\mathcal{E}} v \, \mathcal{R}^{\text{SC}} \, v'$ we infer $v' = x$, and thus we are done.

- We show that $\mathcal{R}^{\text{SC}}$ satisfies (enf abs), i.e.

$$\lambda x.e \, {\mathcal{R}^{\text{SC}}}_{\mathcal{E}} \, s \implies s = \lambda x.f \wedge \vdash^{\mathcal{E}} e \, \mathcal{R}^{\text{SC}} \, f.$$

  We proceed by induction on the derivation of $\vdash^{\mathcal{E}} \lambda x.e \, \mathcal{R}^{\text{SC}} \, s$. The only relevant cases to consider are those for rules ($\mathcal{E}$-sc.$\mathcal{R}$), (Λ-sc.abs), and ($\mathcal{E}$-sc.subst). The first two are trivial. For the latter, suppose $\vdash^{\mathcal{E}} \lambda x.e \, \mathcal{R}^{\text{SC}} \, s$ to be the conclusion of derivation of the form

$$\frac{\vdash^{\mathcal{E}} w \, \mathcal{R}^{\text{SC}} \, w' \quad \vdash^{\mathcal{E}} v \, \mathcal{R}^{\text{SC}} \, v'}{\vdash^{\mathcal{E}} w[y := v] \, \mathcal{R}^{\text{SC}} \, w'[y := v']} \; (\mathcal{E}\text{-sc.subst})$$

  so that $\lambda x.e = w[y := v]$ and $s = w'[y := v']$. We do case analysis on the structure of w. The following are the relevant cases.

  - Suppose $w = y$. Then we have $v = \lambda x.f$. From $\vdash^{\mathcal{E}} y \, \mathcal{R}^{\text{SC}} \, w'$, since by previous point $\mathcal{R}^{\text{SC}}$ satisfies (enf var), it follows $w' = y$. We conclude the thesis by induction hypothesis on $sv \, \mathcal{R}^{\text{SC}} \, v'$.
  - Suppose $w = \lambda x.g$. Then we have $e = g[y := v]$. By induction hypothesis on $\vdash^{\mathcal{E}} w \, \mathcal{R}^{\text{SC}} \, w'$ we have $w' = \lambda x.g'$ with $g \, {\mathcal{R}^{\text{SC}}}_{\Lambda} \, g'$, and thus $s = \lambda x.g'[y := v']$. Since $sv \, \mathcal{R}^{\text{SC}} \, v'$ by rule (Λ-sc.subst) we conclude $\vdash^{\Lambda} g[x := v] \, \mathcal{R}^{\text{SC}} \, g'[x := v']$, and thus we are done.

- We show that $\mathcal{R}^{\text{SC}}$ satisfies (enf stuck), i.e.

$$\vdash^{\mathcal{E}} E[xv] \, {\mathcal{R}^{\text{SC}}}_{\mathcal{E}} \, s \implies \exists E', v'.\, s = E'[xv'] \wedge \vdash^{\mathcal{E}} v \, \mathcal{R}^{\text{SC}} \, v' \wedge \exists z \notin FV(E) \cup FV(E').\ \vdash^{\Lambda} E[z] \, \mathcal{R}^{\text{SC}} \, E'[z].$$

  As before, we proceed by induction on the derivation of $\vdash^{\mathcal{E}} E[xv] \, \mathcal{R}^{\text{SC}} \, s$. Since $E[xv]$ is not a value, $\vdash^{\mathcal{E}} E[xv] \, \mathcal{R}^{\text{SC}} \, s$ cannot be the conclusion of an instance of rule ($\mathcal{E}$-sc.subst) (as the substitution of value for a variable in a value is itself a value). As a consequence, the only relevant cases are those for rules ($\mathcal{E}$-sc.$\mathcal{R}$) and ($\mathcal{E}$-sc.E), which are both straightforward.

□

Lemma 34 (Main Lemma.). *Let $\mathcal{R} = (\mathcal{R}_{\Lambda}, \mathcal{R}_{\mathcal{E}})$ be a λ-term relation. If $\mathcal{R}$ is a enf-simulation, then so is $\mathcal{R}^{\text{SC}}$.*

Proof. The proof is long and non-trivial. First, we immediately notice that $\mathcal{R}^{\text{SC}}$ respects enfs. This directly follows from Lemma 33, since $\mathcal{R}$ is a enf-simulation (and thus respects enfs). As a consequence, it remains to prove that $e \, {\mathcal{R}^{\text{SC}}}_{\Lambda} \, f$ implies $[\![e]\!] \, \Gamma {\mathcal{R}^{\text{SC}}}_{\mathcal{E}} \, [\![f]\!]$. Since Γ is inductive, the latter follows if

$$\forall n.\ \vdash^{\Lambda} e \, \mathcal{R}^{\text{SC}} \, f \implies [\![e]\!]_n \, \Gamma \mathcal{R}^{\text{SC}}_{\mathcal{E}} \, [\![f]\!].$$

We proceed by lexicographic induction on (1) the natural number n and (2) the derivation $\vdash^{\Lambda} e \, \mathcal{R}^{\text{SC}} \, f$. The case for $n = 0$ is trivial (since Γ is inductive). The remaining cases are nontrivial, and we handle

them either using Lemma 27 or observing that $[\![E[e]]\!] = [\![e]\!] \gg\!= (s \mapsto [\![E[s]]\!])$ (cf. Lemma 10). Both these observations allow us to apply condition (rel bind) to simplify proof obligations (usually relying on part (2) of the induction hypothesis as well). This scheme is iterated until we reach either an enf (in which case we rely on condition (rel unit)) or a pair of expression on which we can apply part (1) of the induction hypothesis. We now give technical details. Assume $n > 0$. We proceed by cases on the last inference rule in the derivation of $e\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ f$.

Case (Λ-sc.$\mathcal{R}$). We have $e\ \mathcal{R}_{\Lambda}\ f$. Since $\mathcal{R}$ is a simulation we have $[\![e]\!]_n \sqsubseteq [\![e]\!]\ \Gamma\mathcal{R}_{\mathcal{E}}\ [\![f]\!]$. We conclude the thesis by condition (ind 2) since $\mathcal{R}_{\mathcal{E}} \subseteq \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}$ and Γ is monotone.

Case ($\mathcal{E}$-Λ-sc). We have to prove $\eta(s)\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ \eta(t)$ given $s\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ t$. This directly follows by condition (rel unit).

Case (Λ-sc.app). We have to prove $[\![eg]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![fh]\!]$, given $e\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ f$ and $g\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ h$. We notice that

$$[\![eg]\!]_n \sqsubseteq [\![e]\!]_n \gg\!= (s \mapsto [\![sg]\!]_n)$$
$$[\![fh]\!] = [\![f]\!] \gg\!= (s \mapsto [\![sh]\!]),$$

where $s \mapsto [\![sh]\!]_n$ denotes the function taking as argument an enf s and giving as result $[\![sh]\!]_n$ (and similarity for $s \mapsto [\![sh]\!]$). By condition (ind 2), it is sufficient to prove

$$[\![e]\!]_n \gg\!= (s \mapsto [\![sg]\!]_n)\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![f]\!] \gg\!= (s \mapsto [\![sh]\!]).$$

which in turn, by condition (rel bind), amounts to show $[\![e]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![f]\!]$ and that $s\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ s'$ implies $[\![sg]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![s'h]\!]$. The former directly follows by part (2) of the induction hypothesis, whereas for the latter we assume $s\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ s'$ and proceed by case analysis on s.

Case $s = x$. Then by Lemma 33 we have $s' = x$ too. As before, we have $[\![xg]\!]_n \sqsubseteq [\![g]\!]_n \gg\!= (t \mapsto [\![xt]\!]_n)$ and $[\![xh]\!] = [\![h]\!] \gg\!= (t \mapsto [\![xt]\!])$. Therefore it is sufficient to prove

$$[\![g]\!]_n \gg\!= (t \mapsto [\![xt]\!]_n)\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![h]\!] \gg\!= (t \mapsto [\![xt]\!]).$$

By condition (rel bind) it is sufficient to show $[\![g]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![h]\!]$ and that $t\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ t'$ implies $[\![xt]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![xt']\!]$. The former follows from part (2) of the induction hypothesis, whereas the latter amounts to prove $\eta(xt)\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ \eta(xt')$ for all enfs t, t' such that $t\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ t'$ holds. By condition (rel unit) it is sufficient to show $xt\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ xt'$ which can be easily seen to be the case by case analysis on t. If t is a value v, then by Lemma 33 t' is a value v' too. But then both $xt = xv$ and $xt' = xv'$ are stuck terms. We conclude the thesis by rule ($\mathcal{E}$-sc.E). If t is a stuck term, say $E[yv]$, then by Lemma 33 t' is a stuck term too, say $E'[yv']$ with $v\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ v'$ and $E[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ E'[z]$. But then $xE[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ xE'[z]$ holds too, and thus we can conclude the desired thesis by rule ($\mathcal{E}$-sc.E), since both xE and xE' are evaluation contexts.

Case $s = \lambda x.c$. By Lemma 33 we have $s = \lambda x.c'$ with $c\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ c'$. Proceeding as in previous case, we reduce the proof to showing that $t\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ t'$ implies $[\![(\lambda x.c)t]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![(\lambda x.c')t']\!]$. We assume $t\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ t'$ and prove the thesis by case analysis on t.

Case $t = v$. Then by Lemma 33 we have that t' is a value v' too. We have to prove

$$[\![(\lambda x.c)v]\!]_n = [\![c[x := v]]\!]_{n-1}\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![c'[x := v']]\!] = [\![(\lambda x.c')v']\!].$$

The latter follows by part (1) of induction hypothesis, since $c\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ c'$, $v\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ v'$ imply $c[x := v]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ c'[x := v']$ by rule (Λ-sc.subst).

Case $t = E[yv]$. By Lemma 33 we have $t' = E'[yv']$ with $v\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ v'$ and $E[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ E'[z]$, for some fresh variable z. We have to prove:

$$[\![(\lambda x.c)E[yv]]\!]_n = \eta((\lambda x.c)E[yv])\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ \eta((\lambda x.c')E'[yv']) = [\![(\lambda x.c')E'[yv']]\!].$$

By condition (rel unit) it is sufficient to show $(\lambda x.c)E[yv]\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ (\lambda x.c')E'[yv']$, which holds by rule ($\mathcal{E}$-sc.E).

Case $s = E[xv]$. By Lemma 33 we have $s' = E'[xv']$ with $v\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ v'$ and $E[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ E'[z]$ for some fresh variable z. We have to prove:

$$[\![E[xv]g]\!]_n = \eta(E[xv]g)\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ \eta(E'[xv']h) = [\![E'[xv']h]\!].$$

By condition (rel unit) it is sufficient to show $E[xv]g\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ E'[xv']h$. This is indeed the case since both $v\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ v'$ and $E[z]g\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ E'[z]h$ hold, the latter being implied by $g\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ h$ and $E[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ E'[z]$ (notice that without loss of generality we can assume z to be fresh in g and h since by rule (Λ-sc.subst) $E[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ E'[z]$ implies $E[z']\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ E'[z']$ for any variable z fresh for g, h, $E[z]$, and $E'[z]$).

Case (Λ-sc.subst). We have to prove $[\![e[x := u]]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![f[x := u']]\!]$, given $e\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ f$ and $u\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ u'$. Let $\sigma \triangleq -[x := u]$, $\tau \triangleq -[x := u']$. By Lemma 27 and condition (ind 2) to prove the thesis it is sufficient to show:

$$[\![e]\!]_n \gg\!= (s \mapsto [\![s^\sigma]\!]_n)\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![f]\!] \gg\!= (s \mapsto [\![s^\tau]\!]).$$

Relying on condition (rel bind) we claim that $[\![e]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![f]\!]$ and that $[\![s^\sigma]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![t^\tau]\!]$ hold for all enfs s, t such that $s\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ t$. The former directly follows by part (2) of the induction hypothesis, whereas for the latter we assume $s\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ t$ and proceed by case analysis on s.

Case $s = v$. By Lemma 33 we have $t = w$ for some value w. Since both v^σ and w^τ are values, we have to show $\eta(v^\sigma)\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ \eta(w^\tau)$. By condition (rel unit) it is sufficient to prove $v^\sigma\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ w^\tau$, which follows by rule ($\mathcal{E}$-sc.subst).

Case $s = E[yv], y \neq x$. We proceed as in previous case.

Case $s = E[xv]$. By Lemma 33 we have $t = F[xw]$ with $v\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ w$ and $E[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ F[z]$ for some fresh variable z. We have to prove

$$[\![E^\sigma[uv^\sigma]]\!]_n\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![F^\tau[u'w^\tau]]\!].$$

We proceed by case analysis on u. If u is a variable, the we proceed as in previous case. Otherwise u is a λ-abstraction $\lambda y.g$. By Lemma 33 we have $u' = \lambda y.h$ with $g\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ h$. By very definition of operational semantics, we have to prove:

$$[\![E^\sigma[g[y := v^\sigma]]]\!]_{n-1}\ \Gamma\mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ [\![F^\tau[h[y := w^\tau]]]\!].$$

The latter follows by part (1) of the induction hypothesis, provided $E^\sigma[g[y := v^\sigma]]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ F^\tau[h[y := w^\tau]]$. This is indeed the case since $E[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ F[z]$ and $u\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ u'$ imply $E^\sigma[z]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ F^\tau[z]$ (without loss of generality we can assume $x \neq z$). By rule ($\mathcal{E}$-sc.subst), from $u\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ u'$ and $v\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ w$ we infer $v^\sigma\ \mathcal{R}^{\mathrm{sc}}{}_{\mathcal{E}}\ w^\tau$, and thus $g[x := v^\sigma]$ and $h[x := w^\tau]$ by rule (Λ-sc.subst). We conclude $E^\sigma[g[y := v^\sigma]]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ F^\tau[h[y := w^\tau]]$ by rule (Λ-sc.E).

Case (Λ-sc.op). We have to prove $[\![\mathbf{op}(p, x.e)]\!]_n \Gamma\mathcal{R}_{\mathcal{E}}{}^{\mathrm{sc}}\ [\![\mathbf{op}(p, x.f)]\!]$, given $e\mathcal{R}^{\mathrm{sc}}{}_{\Lambda}f$. For that, it is sufficient to show

$$[\![\mathbf{op}]\!](p, v \mapsto [\![e[x := v]]\!]_{n-1})\ \Gamma\mathcal{R}_{\mathcal{E}}{}^{\mathrm{sc}}\ [\![\mathbf{op}]\!](p, v \mapsto [\![f[x := v]]\!]).$$

Since $\mathcal{R}_{\mathcal{E}}{}^{\mathrm{sc}}$ is reflexive, from $e\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ f$ we infer $e[x := v]\ \mathcal{R}^{\mathrm{sc}}{}_{\Lambda}\ f[x := v]$, for any v. We can thus conclude the wished thesis using condition (Σ comp) and part (1) of the induction hypothesis.

Case (Λ-sc.E). We have to prove $[\![E[e]]\!]_n \, \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, [\![F[f]]\!]$ given $e \, \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \, f$ and $E[z] \, \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \, F[z]$, for some fresh variable z. As usual, it is sufficient to show:

$$[\![e]\!]_n \gg\!= (s \mapsto [\![E[s]]\!]_n) \; \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \; [\![f]\!] \gg\!= (s \mapsto [\![F[s]]\!]).$$

By condition (rel bind) it is sufficient to prove $[\![e]\!]_n \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} [\![f]\!]$ and that $s \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} t$ implies $[\![E[s]]\!]_n \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} [\![F[t]]\!]$. The former follows by part (2) of the induction hypothesis since $e \, \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \, f$, whereas for the latter we assume $s \, \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, t$ and proceed by case analysis on s.

Case 1. If s is a stuck term, say $A[xv]$, then by Lemma 33 we have $t = B[xw]$ with $A[y] \, \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \, B[y]$ and $v \, \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, w$. As a consequence, proving $[\![E[s]]\!]_n \; \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \; [\![F[t]]\!]$ amounts to prove $\eta(E[A[xv]]) \; \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \; \eta(F[B[xw]])$. By condition (rel unit) to prove the latter it is sufficient to show $E[A[xv]] \; \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \; F[B[xw]]$. This is indeed the case thanks to rule ($\mathcal{E}$-sc.E) since both $E[A[-]]$ and $F[B[-]]$ are evaluation contexts and we have $E[A[y]] \; \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \; F[B[y]]$.

Case 2. If s is a value v, then t must be a value w too, and we have to show $[\![E[v]]\!]_n \, \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, [\![F[w]]\!]$. Since z is fresh in $E[-]$ and $F[-]$ we notice that $E[v] = (E[z])^{\sigma}$ and $F[w] = (F[z])^{\tau}$, where $\sigma \triangleq -[z := v]$ and $\tau \triangleq -[z := w]$. By Lemma 27 to prove the thesis it is thus sufficient to show

$$[\![E[z]]\!]_n \gg\!= (r \mapsto [\![r^{\sigma}]\!]_n) \; \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \; [\![F[z]]\!] \gg\!= (r \mapsto [\![r^{\tau}]\!]).$$

By condition (rel bind) to prove the latter it is sufficient to prove $[\![E[z]]\!]_n \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} [\![F[z]]\!]$ and that $r_1 \, \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, r_2$ implies $[\![r_1^{\sigma}]\!]_n \, \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, [\![r_2^{\tau}]\!]$. As usual the former follows by part (2) of the induction hypothesis since $E[z] \, \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \, F[z]$, whereas for the latter we assume $r_1 \, \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, r_2$ and proceed by case analysis on r_1. If r_1 is a value, then so is r_2 (by Lemma 33) and proving the thesis is straightforward. If r_1 is a stuck term $A_1[xv_1]$, then r_2 must be a stuck term too, say $A_2[xv_2]$, with $A_1[y] \, \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \, A_2[y]$ and $v_1 \, \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, v_2$. If $x \neq z$, then we have to show

$$[\![A_1^{\sigma}[xv_1^{\sigma}]]\!]_n \; \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \; [\![A_2^{\tau}[xv_2^{\tau}]]\!].$$

By condition (rel unit) the latter follows from $A_1^{\sigma}[xv_1^{\sigma}] \, \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \, A_2^{\tau}[xv_2^{\tau}]$ which indeed holds. If $x = z$, then we have to show

$$[\![A_1^{\sigma}[vv_1^{\sigma}]]\!]_n \; \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \; [\![A_2^{\tau}[wv_2^{\tau}]]\!].$$

If v is a variable, then so must be w (since $v \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} w$) and we proceed as above. If $v \equiv \lambda y.g$, then by Lemma 33 we have $w \equiv \lambda y.h$ with $g \, \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \, h$. By very definition of operational semantics we have to show

$$[\![A_1^{\sigma}[g[y := v_1^{\sigma}]]]\!]_{n-1} \; \Gamma \mathcal{R}^{\mathrm{sc}}{}_{\varepsilon} \; [\![A_2^{\tau}[h[y := v_2^{\tau}]]]\!].$$

The latter follows by part (1) of the induction hypothesis, since we have $A_1^{\sigma}[g[y := v_1^{\sigma}]] \, \mathcal{R}^{\mathrm{sc}}{}_{\Lambda} \, A_2^{\tau}[h[y := v_2^{\tau}]]$.

□

Theorem 9. *Enf-similarity is a precongruence relation.*

Proof. From Lemma 34 we know that $\preceq^{\mathsf{E}}$ is a compatible, reflexive, and substitutive term relation. It remains to prove that it is also transitive. By coinduction, it is sufficient to show that $\preceq^{\mathsf{E}} \cdot \preceq^{\mathsf{E}}$ is an enf-simulation, which is a routine exercise (notice, however, that to handle condition (enf stuck) we rely on substitutivity of $\preceq^{\mathsf{E}}$). □

Finally, we can rely on Proposition 20 to prove that enf-bisimilarity is a congruence. In fact, as discussed in Subsection 5.4.1, our proof of Lemma 34 requires the relator Γ to be inductive, a condition which is not satisfied by conversive relators, in general. However, by Proposition 20 we can keep working with the Σ-continuous relator Γ and show that $(\simeq^{E})^{SC}$ is a enf-simulation with respect to Γ. In order to conclude $(\simeq^{E})^{SC} = \simeq^{E}$ we need to show that $(\simeq^{E})^{SC}$ is symmetric. An easy inspection of the rules in Figure 7.2 reveals that for any term relation $\mathcal{R}$, $\mathcal{R}^{SC}$ is symmetric whenever $\mathcal{R}$ is. We have thus proved the following result.

Theorem 10. *Enf-bisimilarity is a congruence relation.*

We conclude this section with a short discussion on congruence and precongruence properties of whnf-(bi)similarity. We can prove that whnf-similarity is a precongruence by mimicking the proof given for enf-similarity. The substitutive and compatible closure of a term relation $\mathcal{R} = (\mathcal{R}_{\Lambda}, \mathcal{R}_{\mathcal{W}})$ is defined according to the rules in Figure 7.3. Due to the simpler form of evaluation contexts (and thus of whnfs) of $\Lambda_{\Sigma}^{(n)}$, the definition of substitutive and compatible closure for $\Lambda_{\Sigma}^{(n)}$ is easier than the one for $\Lambda_{\Sigma}^{(v)}$. Such simplicity is reflected in the proof of the analogous of Lemma 34 which closely follows the one for $\Lambda_{\Sigma}^{(v)}$ but which is substantially easier.

Theorem 11. *Whnf-similarity is a precongruence relation, and whnf-bisimilarity is a congruence relation.*

$$\frac{\vdash^{\Lambda} e\ \mathcal{R}\ f}{\vdash^{\Lambda} e\ \mathcal{R}^{SC}\ f}\ (\Lambda\text{-sc.}\mathcal{R}) \qquad \frac{\vdash^{\mathcal{W}} s\ \mathcal{R}\ t}{\vdash^{\mathcal{W}} s\ \mathcal{R}^{SC}\ t}\ (\mathcal{W}\text{-sc.}\mathcal{R}) \qquad \frac{\vdash^{\mathcal{W}} s\ \mathcal{R}^{SC}\ t}{\vdash^{\Lambda} s\ \mathcal{R}^{SC}\ t}\ (\mathcal{W}\text{-}\Lambda\text{-sc})$$

$$\frac{}{\vdash^{\mathcal{W}} x\ \mathcal{R}^{SC}\ x}\ (\mathcal{W}\text{-sc.var})$$

$$\frac{\vdash^{\mathcal{W}} E[z]\ \mathcal{R}^{SC}\ F[z] \qquad \vdash^{\Lambda} e\ \mathcal{R}^{SC}\ f}{\vdash^{\mathcal{W}} E[xe]\ \mathcal{R}^{SC}\ F[xf]}\ (\mathcal{W}\text{-sc.E})$$

$$\frac{\vdash^{\Lambda} e\ \mathcal{R}^{SC}\ f}{\vdash^{\mathcal{W}} \lambda x.e\ \mathcal{R}^{SC}\ \lambda x.f}\ (\mathcal{W}\text{-sc.abs}) \qquad \frac{\vdash^{\Lambda} e\ \mathcal{R}^{SC}\ f \qquad \vdash^{\Lambda} g\ \mathcal{R}^{SC}\ h}{\vdash^{\Lambda} eg\ \mathcal{R}^{SC}\ fh}\ (\Lambda\text{-sc.app})$$

$$\frac{\vdash^{\Lambda} e\ \mathcal{R}^{SC}\ f \qquad \vdash^{\Lambda} g\ \mathcal{R}^{SC}\ h}{\vdash^{\Lambda} e[x := g]\ \mathcal{R}^{SC}\ f[x := h]}\ (\Lambda\text{-sc.subst}) \qquad \frac{\vdash^{\Lambda} e\ \mathcal{R}^{SC}\ f}{\vdash^{\Lambda} \mathbf{op}(p, x.e)\ \mathcal{R}^{SC}\ \mathbf{op}(p, x.f)}\ (\Lambda\text{-sc.op})$$

Figure 7.3: Compatible and substitutive closure construction for $\Lambda_{\Sigma}^{(n)}$.

Theorem 9, Theorem 10, and Theorem 11 qualify effectful normal form similarity and bisimilarity as good candidate notions for program equivalence and refinement, at least from a structural perspective. Additionally, such notions have been introduced looking at specif examples aimed to show limitations of effectful applicative (bi)similarity. It is then natural to ask what is the relationship between all such notions.

We now present a formal comparison between effectful enf-(bi)similarity and effectful applicative (bi)similarity, as defined in Chapter 5. It is a straightforward exercise to rephrase such a comparison for whnf-bisimilarity and the call-by-name effectful variation of applicative (bi)similarity as sketched at the beginning of Chapter 6. We will say more about that below, and present a summary of the results obtained in Chapter 8.

First of all, we cannot rely on the notation used in Chapter 5, as there would be some inconsistencies. We thus use the following notational convention. Let $\Lambda_0, \mathcal{V}_0$ denote the collections of closed terms

and closed values, respectively. We keep using the notation $[\![-]\!]$ to denote the evaluation function of Definition 45. We notice that if $e \in \Lambda_0$, then $[\![e]\!] \in T\mathcal{V}_0$. Formally, we express that by saying that the evaluation map $[\![-]\!]$ factors as follows, where $\iota : \mathcal{V}_0 \hookrightarrow \mathcal{E}$ is the obvious inclusion map:

$$\begin{array}{ccc} \Lambda_0 & \xrightarrow{|-|} & T\mathcal{V}_0 \\ \downarrow & & \downarrow{\scriptstyle T\iota} \\ \Lambda & \xrightarrow[[\![-]\!]]{} & T\mathcal{E} \end{array}$$

In particular, the map $|-|$ is nothing but the evaluation function of Definition 15. We can thus phrase the definition of effectful applicative similarity (with respect to a relator Γ) as follows.

Definition 53. *A term relation $\mathcal{R} = (\mathcal{R}_\Lambda : \Lambda_0 \nrightarrow \Lambda_0, \mathcal{R}_\mathcal{V} : \mathcal{V}_0 \nrightarrow \mathcal{V}_0)$ is an effectful applicative simulation with respect to a relator Γ (applicative simulation, for short) if:*

$$\vdash^\Lambda e \,\mathcal{R}\, f \implies |e|\ \Gamma\mathcal{R}_\mathcal{V}\ |f| \qquad \text{(app comp}')$$

$$\vdash^\mathsf{v} \lambda x.e \,\mathcal{R}\, \lambda x.f \implies \forall v \in \mathcal{V}_0.\ \vdash^\Lambda e[x := v] \,\mathcal{R}\, f[x := v]. \qquad \text{(app val}')$$

Let $\preceq^\mathsf{A} = (\preceq^\mathsf{A}_\Lambda, \preceq^\mathsf{A}_\mathcal{V})$ denote applicative similarity, and $\preceq^c = (\preceq_\Lambda{}^c, \preceq_\mathcal{E}{}^c)$ denote the restriction of enf-similarity $\preceq^\mathsf{E}$ to closed terms and closed eager normal forms. Obviously $\preceq^\mathsf{A}$ describes effectful applicative similarity as defined in Chapter 5, but adapted to the coarse-grain version of Λ_Σ.

Proposition 22. *Enf-similarity restricted to closed term and enfs is contained in applicative similarity.*

Proof. We show $\preceq^c \subseteq \preceq^\mathsf{A}$ by coinduction, proving that $\preceq^c$ satisfies conditions (app val′) and (app comp′).

- We show:

$$\vdash^\mathcal{E} \lambda x.e \preceq^c \lambda x.f \implies \forall v \in \mathcal{V}_\circ.\ \vdash^\Lambda e[x := v] \preceq^c f[x := v].$$

  We have:

$$\begin{aligned} \vdash^\mathcal{E} \lambda x.e \preceq^c \lambda x.f &\implies \vdash^\mathcal{E} \lambda x.e \preceq^\mathsf{E} \lambda x.f \\ &\quad [\text{Since } \preceq^c \subseteq \preceq^\mathsf{E}] \\ &\implies \vdash^\Lambda e \preceq f \\ &\quad [\text{By (enf abs)}] \\ &\implies \forall v \in \mathcal{V}_\circ.\ \vdash^\Lambda e[x := v] \preceq^\mathsf{E} f[x := v] \\ &\quad [\text{Since } \preceq^\mathsf{E} \text{ is reflexive and substitutive}] \\ &\implies \vdash^\Lambda e[x := v] \preceq^c f[x := v]. \\ &\quad [\text{Since } e[x := v], f[x := v] \text{ are closed}] \end{aligned}$$

- We prove (app comp′). We have:

$$\begin{aligned} \vdash^\Lambda e \preceq^c f &\implies \vdash^\Lambda e \preceq^\mathsf{E} f \\ &\quad [\text{Since } \preceq^c \subseteq \preceq^\mathsf{E}] \\ &\implies [\![e]\!]\ \Gamma \preceq^\mathsf{E}_\mathcal{E}\ [\![f]\!] \\ &\quad [\text{By (enf comp)}] \\ &\implies T\iota|e|\ \Gamma \preceq^\mathsf{E}_\mathcal{E}\ T\iota|f|. \\ &\quad [\text{Since } \forall e \in \Lambda_\circ.\ [\![e]\!] = T\iota|e|] \end{aligned}$$

We conclude the thesis observing that since ${\preceq_{\mathcal{E}}}^{c} = \iota^{\circ} \cdot \preceq_{\mathcal{E}}^{\mathsf{E}} \cdot \iota$, we can use (stability) to calculate:

$$\Big((T\iota \cdot |-|)^{\circ} \cdot \Gamma \preceq_{\mathcal{E}} \cdot T\iota \cdot |-|\Big) = \Big(|-|^{\circ} \cdot (T\iota)^{\circ} \cdot \Gamma \preceq_{\mathcal{E}} \cdot T\iota \cdot |-|\Big) = \Big(|-|^{\circ} \cdot \Gamma(\iota^{\circ} \cdot \preceq_{\mathcal{E}} \cdot \iota) \cdot |-|\Big),$$

and thus conclude $|e|\ \Gamma {\preceq_{\mathcal{E}}}^{c}\ |f|$.

□

By Definition 49 it follows that enf-bisimilarity restricted on closed terms and enfs is contained in applicative bisimilarity (the latter being defined as the largest symmetric applicative simulation). This means that effectful enf-similarity is a sound proof technique for effectful applicative bisimilarity and thus for effectful contextual equivalence, in virtue of the results of Chapter 5. Moreover, Proposition 22 extends to enf-similarity and open applicative bisimilarity (i.e. the open extension of $\preceq^{\mathsf{A}}$). For, let e, f be open terms with free variables $\vec{x}$. Then for all closed values $\vec{v}$ we have:

$$\begin{aligned} e \preceq_{\Lambda}^{\mathsf{E}} f &\implies e[\vec{x} := \vec{v}] \preceq_{\Lambda}^{\mathsf{E}} f[\vec{x} := \vec{v}] \\ &\quad \text{[By substitutivity of } \preceq \text{]} \\ &\implies e[\vec{x} := \vec{v}] \preceq_{\Lambda}^{\mathsf{A}}, f[\vec{x} := \vec{v}] \\ &\quad \text{[By Proposition 22]} \end{aligned}$$

where in applying substitutivity we use the fact that $\vec{v}$ consisting of closed values only, sequential substitution coincide with simultaneous substitution. An analogous result holds for values.

Although enf-(bi)similarity is sound for applicative (bi)similarity, it is *not* fully-abstract. In fact, already in the pure λ-calculus enf-bisimilarity is strictly finer than applicative bisimilarity (and thus strictly finer than contextual equivalence too), as observed in (S. B. Lassen, 2005). To see that, let us consider Example 1.4 in (Durier, Hirschkoff, & Sangiorgi, 2018): the terms xv and $(\lambda y.xv)(xv)$ are obviously applicatively bisimilar but not enf-bisimilar.

Proposition 22 can also be easily adapted to whnf-(bi)similarity and the call-by-name version of applicative (bi)similarity. Again, full abstraction fails. To see that, we can consider Example 2.3.10 in (Abramsky & Ong, 1993), where it is observed that the (open) terms xx and $x(\lambda y.xy)$ are applicatively bisimilar, but not whnf-bisimilar.

7.4.2 Normal Form (Bi)simulation Up-to Context

The up-to context technique (S. B. Lassen, 1999; Pous & Sangiorgi, 2012) is a refinement of the coinduction proof principle of normal form (bi)similarity that allows for handier proofs of equivalence and refinement between terms. When exhibiting a candidate (bi)simulation relation $\mathcal{R}$, it is desirable for $\mathcal{R}$ to be as small as possible, so to minimise the task of verifying that $\mathcal{R}$ is indeed a (bi)simulation.

The motivation behind such a technique can be easily seen by looking at Example 45 and Example 46, where we showed the equivalence between probabilistic fixed point combinators working with relations containing several administrative pairs of terms. The presence of such administrative pairs was forced by Definition 49 and Definition 51, although they appear somehow unnecessary in order to convince that e.g. Y and Z exhibit the same behaviour.

Normal form (bi)simulation up-to context is a refinement of normal form (bi)simulation that allows to check that a relation $\mathcal{R}$ behaves as a (bi)simulation relation up to its substitutive and compatible closure. We first define the notion of an enf-simulation up-to context.

Definition 54. *A term relation* $\mathcal{R} = (\mathcal{R}_{\Lambda} : \Lambda \nrightarrow \Lambda, \mathcal{R}_{\mathcal{E}} : \mathcal{E} \nrightarrow \mathcal{E})$ *is an* effectful eager normal form simulation up-to context with respect to Γ *(hereafter enf-simulation up-to context) if satisfies the following*

conditions.

$$\vdash^{\Lambda} e\,\mathcal{R}\,f \implies [\![e]\!]\,\Gamma\mathcal{R}_{\mathcal{E}}^{\mathrm{SC}}\,[\![f]\!] \qquad \text{(enf up-to ctx comp)}$$

$$\vdash^{\mathcal{E}} x\,\mathcal{R}\,s \implies s = x \qquad \text{(enf up-to ctx var)}$$

$$\vdash^{\mathcal{E}} \lambda x.e\,\mathcal{R}\,s \implies \exists f.\, s = \lambda x.f \wedge \ \vdash^{\Lambda} e\,\mathcal{R}^{\mathrm{SC}}\,f \qquad \text{(enf up-to ctx abs)}$$

$$\vdash^{\mathcal{E}} E[xv]\,\mathcal{R}\,s \implies \exists E', v'.\, s = E'[xv'] \wedge \ \vdash^{\mathcal{E}} v\,\mathcal{R}^{\mathrm{SC}}\,v' \wedge \exists z \notin FV(E) \cup FV(E').\ \vdash^{\Lambda} E[z]\,\mathcal{R}^{\mathrm{SC}}\,E'[z].$$
(enf up-to ctx stuck)

In order for the up-to context technique to be sound, we need to show that every enf-simulation up-to context is contained in enf-similarity. This is a direct consequence of the following variation of Lemma 34.

Lemma 35. *If $\mathcal{R}$ is a enf-simulation up-to context, then $\mathcal{R}^{\mathrm{SC}}$ is a enf-simulation.*

Proof. The proof is structurally identical to the one of Lemma 34, where we simply observe that wherever we use the assumption that $\mathcal{R}$ is an enf-simulation, we can use the weaker assumption that $\mathcal{R}$ is an enf-simulation up-to context. □

In particular, since by Lemma 34 we have that $\preceq^{\mathrm{E}} = (\preceq^{\mathrm{E}})^{\mathrm{SC}}$, we see that enf-similarity is an enf-simulation up-to context. Additionally, by Lemma 35 it is the largest such. Since the same result holds for enf-bisimilarity and enf-bisimilarity up-to context, we have the following result.

Theorem 12. *Enf-similarity is the largest enf-simulation up-to context, and enf-bisimilarity is the largest enf-bisimulation up-to context.*

Example 47. We apply Theorem 12 to simply the proof of the equivalence between Y and Z given in Example 45. In fact, it is sufficient to show that the symmetric closure of term relation $\mathcal{R}$ defined as follows is an enf-bisimulation up-to context.

$$\mathcal{R}_{\Lambda} \triangleq \{(Y, Z), (\Delta\Delta z, Zyz), (\Delta\Delta, y(\lambda z.\Delta\Delta z) \text{ or } y(\lambda z.Zyz))\}$$
$$\mathcal{R}_{\mathcal{E}} \triangleq \mathsf{I}_{\mathcal{E}}.$$

⊠

It is not hard to see that the above results can be rephrased in terms of whnf-similarity and whnf-bisimilarity.

Definition 55. *A term relation $\mathcal{R} = (\mathcal{R}_{\Lambda} : \Lambda \nrightarrow \Lambda, \mathcal{R}_{\mathcal{W}} : \mathcal{W} \nrightarrow \mathcal{W})$ is a* weak head normal form simulation up-to context with respect to Γ *(whnf-simulation up-to context, for short) if:*

$$\vdash^{\Lambda} e\,\mathcal{R}\,f \implies [\![e]\!]\,\Gamma\mathcal{R}_{\mathcal{W}}^{\mathrm{SC}}\,[\![f]\!] \qquad \text{(whnf up-to ctx var)}$$

$$\vdash^{\mathcal{W}} \lambda x.e\,\mathcal{R}\,s \implies \exists f.\, s = \lambda x.f \wedge \vdash^{\Lambda} e\,\mathcal{R}^{\mathrm{SC}}\,f \qquad \text{(whnf up-to ctx abs)}$$

$$\vdash^{\mathcal{W}} xe_0 \cdots e_n\,\mathcal{R}\,s \implies \exists f_0, \ldots, f_k.\, s = xf_0 \cdots f_k \wedge \forall i.\ \vdash^{\Lambda} e_i\,\mathcal{R}^{\mathrm{SC}}\,f_i. \qquad \text{(whnf up-to ctx stuck)}$$

Theorem 13. *Whnf-similarity is the largest whnf-simulation up-to context, and whnf-bisimilarity is the largest whnf-bisimulation up-to context.*

Example 48. We can use Theorem 13 to show the equivalence between the combinators P and Q of Example 46. The reader is invited to compare the (considerably larger) size of the whnf-simulation exhibited in Example 46 with the one defined below.

$$\mathcal{R}_{\Lambda} \triangleq \{(P, Q), (Pyz, Qzy), (Pzy, Qyz)\}$$
$$\mathcal{R}_{\mathcal{W}} \triangleq \emptyset.$$

It is a straightforward exercise to show that $\mathcal{R}$ is indeed a whnf-simulation up to-context, so that we have $P \preceq^{\mathrm{W}} Q$. Moreover, the symmetric closure of $\mathcal{R}$ is whnf-bisimulation up to-context, and thus we have $P \simeq^{\mathrm{W}} Q$. ⊠

We conclude this section spending a couple of words on up-to techniques. In this chapter we focused on normal form (bi)similarity up-to context, as for our purpose the latter is probably the most powerful technique. There are several other up-to techniques, such as (bi)simulation up-to similarity (Milner, 1989) or (bi)simulation up-to improvement (S. Lassen, 1998a). For instance, it is not hard to see the up-to similarity technique to be valid for effectful applicative similarity (the proof is straightforward), where the mapping $\mathcal{R} \mapsto [\mathcal{R}]$ is the one defined in Definition 32.

Proposition 23. *The following law holds for any closed λ-term relation $\mathcal{R}$:*

$$\frac{\mathcal{R} \;\subseteq\; [(\preceq^{\mathrm{A}})^o \cdot \mathcal{R}^o \cdot (\preceq^{\mathrm{A}})^o]}{\mathcal{R} \subseteq \preceq^{\mathrm{A}}} \text{ (up-to } \preceq^{\mathrm{A}})$$

Proving an up-to similarity technique for normal form similarity requires to deal with some technicalities, as normal form similarity is defined on open terms. Nevertheless, if $\mathcal{R}$ is closed under the substitution rule below, where $x \notin FV(E) \cup FV(F)$, then we can easily prove the validity of an up-to similarity technique.

$$\frac{\vdash^{\Lambda} E[x] \,\mathcal{R}\, F[x] \quad z \notin FV(E) \cup FV(F)}{\vdash^{\Lambda} E[z] \,\mathcal{R}\, F[z]}$$

In (S. Lassen, 1998a) several up-to techniques are proved to be sound for *pure* applicative (bi)similarity. Among those, there are applicative similarity up-to similarity and applicative similarity up-to improvement (Sands, 1998), as well as combinations thereof. In particular, the following restricted form of the up-to context technique is proved to be valid for pure applicative similarity (again, the mapping $\mathcal{R} \mapsto [\mathcal{R}]$ is the one defined in Definition 32):

$$\frac{\mathcal{R} \;\subseteq [\widehat{\mathcal{R}}]}{\mathcal{R} \;\subseteq \preceq^{\mathrm{A}}} \text{ (up-to-ctx-closed)}$$

The soundness of the 'full' up-to context technique

$$\frac{\mathcal{R} \;\subseteq [\widehat{(\mathcal{R}^o)}]}{\mathcal{R} \;\subseteq \preceq^{\mathrm{A}}} \text{ (up-to-ctx)}$$

is, to the best of author's knowledge, still an open problem.

7.5 Effectful Lévy-Longo Trees

At the very beginning of this chapter, we made several references to Böhm-tree like equivalences. That was rather informal, as there is no precise definition of such equivalences. In this section we study the formal connection between normal form similarity and Böhm-tree like approximation. Notably, we define and analyse suitable generalisation of *Lévy-Longo trees* for effectful calculi as well as the equivalence and approximation they induce on terms. As we will see, the latter coincide with whnf bisimilarity and similarity, respectively. Finally, we briefly discuss tree-like equivalences associated with enf-(bi)similarity.

Böhm trees (Barendregt, 1984) are infinitary tree-like structures used as mathematical descriptions of the behaviour of pure, untyped λ-terms when evaluated according to the so-called head normal form

reduction (Barendregt, 1984). Lévy-Longo trees (Lévy, 1975; Longo, 1983; C. Ong, 1988) have been introduced as the natural counterpart of Böhm trees in the context of the so-called lazy or weak head normal form reduction, i.e. a call-by-name reduction strategy reducing terms to weak head normal forms (Abramsky, 1990a).

The original formulation of both Böhm and Lévy-Longo trees is of an inductive nature, as witnessed by e.g. (Barendregt, 1984). Moving from ideas developed in concurrency theory[6], in (Sangiorgi, 1992, 1994) Lévy-Longo tree equality has been characterised *coinductively*, introducing the notion of *open bisimilarity*, the latter being nothing but whnf-bisimilarity for $\Lambda_{\mathbb{M}}$. The study of Lévy-Longo tree equality as a suitable notion of bisimulation opened the door to the analysis of tree like equivalences using coinductive techniques, as witnessed by the numerous works on the subject (see e.g. (Biernacki, Lenglet, & Polesiuk, 2018; De Liguoro & Piperno, 1995; S. B. Lassen, 1999, 2005), just to mention a few).

In this section we introduce *effectful Lévy-Longo trees* as an extension of Lévy-Longo trees to $\Lambda_{\Sigma}^{(n)}$ and show that we can define a suitable preorder (resp. equivalence) on such trees which coincides with whnf-similarity (resp. whnf-bisimilarity). Specific notions of effecful Böhm trees have been introduced in the context of nondeterministic (De Liguoro & Piperno, 1995) and probabilistic (Leventis, 2016) λ-calculi, and it is not hard to see that our notion of effectful Lévy-Longo tree is deeply related to such proposals. The major differences are the following.

1. Differently from (De Liguoro & Piperno, 1995) and (Leventis, 2016) we work with lazy calculi, meaning that we do not evaluate under lambdas, and thus we work with generalisations of Lévy-Longo trees rather than Böhm trees.

2. Our approach is parametric over a large class of effects, and not tailored to specific ones.

3. Our definitions and results are mostly of a coinductive nature, whereas both (De Liguoro & Piperno, 1995) and (Leventis, 2016) seem to be more inductively-oriented.

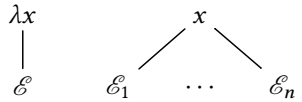
In the rest of this section we work with the calculus $\Lambda_{\Sigma}^{(n)}$ and fix both a Σ-continuous monad $\mathbb{T}$ and a Σ-continuous relator Γ for it. For convenience, we use a compact notation to represent trees.

Definition 56. *The collection $\mathcal{VT}(\mathbb{T})$ of* effectful value Lévy-Longo trees *(with respect to $\mathbb{T}$) is coinductively defined as follows:*

1. *If $\mathscr{E} \in T(\mathcal{VT}(\mathbb{T}))$, then $\lambda x.\mathscr{E} \in \mathcal{VT}(\mathbb{T})$.*
2. *If $\mathscr{E}_1, \ldots, \mathscr{E}_n \in T(\mathcal{VT}(\mathbb{T}))$, then $x\langle \mathscr{E}_1, \ldots, \mathscr{E}_n \rangle \in \mathcal{VT}(\mathbb{T})$.*

The collection of effectful Lévy-Longo trees *(with respect to $\mathbb{T}$) is defined as $T(\mathcal{VT}(\mathbb{T}))$.*

As usual, where no confusion arises we will omit references to the monad $\mathbb{T}$ and refer to effectful value trees and effectful trees as value trees and trees, respectively. Intuitively, a value tree of the form $\lambda x.\mathscr{E}$ represent a tree with root $\lambda x.$ and with the single child $\mathscr{E}$. Similarly, $x\langle \mathscr{E}_1, \ldots, \mathscr{E}_n \rangle$ represents a tree with root x and children $\mathscr{E}_1, \ldots, \mathscr{E}_n$. Pictorially, we can present such trees as:



The elements $\mathscr{E}_1, \ldots, \mathscr{E}_m$ are, for instance, subdistributions of value trees when $\mathbb{T}$ is $\mathbb{DM}$, sets of value trees when $\mathbb{T}$ is $\mathbb{PM}$, or simply trees (including the undefined tree $\bot$) when $\mathbb{T}$ is $\mathbb{M}$. As for usual λ-terms, we work with (value) trees up to renaming of bound variables.

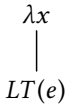
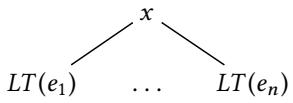
[6] Notably from the theory of open bisimulation for the π-calculus (Sangiorgi, 1993).

Example 49. We immediately notice that for any monad T we have $\bot \in T(\mathcal{VT}(\mathbb{T}))$, so that $\lambda x.\bot$ is a value tree. Moreover, we can define the tree $\top$ as the solution to the equation $\xi = \lambda x.\eta(\xi)$. Therefore, we have $\top = \lambda x.\eta(\lambda x.\eta(\lambda x.\eta(\ldots))))$. Since we work up to renaming of bound variable, $\top$ intuitively captures the behaviour of terms of order ω without side effects (see (C. Ong, 1988; Sangiorgi, 1994) for details). ⊠

Definition 57. *The* effectful Lévy-Longo tree *(hereafter tree or Lévy-Longo tree)* $LT(e)$ *of a term* e *is defined as* $T(\varphi)[\![e]\!]$*, where the map* $\varphi : \mathcal{W} \to \mathcal{VT}(\mathbb{T})$ *is defined by:*

$$\varphi(\lambda x.e) = \lambda x.LT(e)$$
$$\varphi(xe_1 \ldots e_n) \triangleq x\langle LT(e_1) \ldots LT(e_n)\rangle.$$

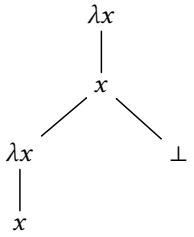
Pictorially:

$LT(e) \triangleq T(\varphi)[\![e]\!]$ $\quad$ $\varphi(\lambda x.e) \triangleq$  $\quad$ $\varphi(xe_1 \ldots e_n) \triangleq$ 

Example 50. Let $\mathbb{T}$ be the partiality monad $\mathbb{M}$. We immediately observe that $LT(\Omega) = \bot$, where as usual $\Omega \triangleq (\lambda x.xx)(\lambda x.xx)$. Similarly, we can calculate the Lévy-Longo tree $LT(\lambda x.xI\Omega)$, for $I \triangleq \lambda x.x$, as follows:

$$\begin{aligned} LT(\lambda x.xI\Omega) &= \mathit{just}\ (\lambda x.LT(xI\Omega)) \\ &= \mathit{just}\ (\lambda x.\mathit{just}\ (x\langle LT(I), LT(\Omega)\rangle)) \\ &= \mathit{just}\ (\lambda x.\mathit{just}\ (x\langle \mathit{just}\ (\varphi(\lambda x.x)), \bot\rangle)) \\ &= \mathit{just}\ (\lambda x.\mathit{just}\ (x\langle \mathit{just}\ (\lambda x.LT(x)), \bot\rangle)) \\ &= \mathit{just}\ (\lambda x.\mathit{just}\ (x\langle \mathit{just}\ (\lambda x.\mathit{just}\ x), \bot\rangle)). \end{aligned}$$

Omitting *just* annotations, we can represent $LT(\lambda x.xI\Omega)$ as:



Let us now consider the ogre combinator $O \equiv YK$, where $K \triangleq \lambda x.\lambda y.x$ and Y is Curry fixed point combinator. Let us write Δ_K for $\lambda x.K(xx)$. Since $[\![O]\!] = \mathit{just}\ \lambda y.\Delta_K\Delta_K$ and $[\![\Delta_K\Delta_K]\!] = \mathit{just}\ \lambda y.\Delta_K\Delta_K$, we see that $LT(O)$ is given as the solution to the equation $\xi = \mathit{just}\ (\lambda y.\xi)$. Omitting *just* annotations, we can represent $LT(O)$ as the following tree, and thus see that $LT(O)$ coincides with $\top$.

⊠

Example 51. Let $\mathbb{T}$ be the partial powerset monad $\mathbb{F}\mathbb{T}$. We have:

$$\begin{aligned} LT(\lambda x.(x \textbf{ or } \Omega)) &= \{just\ (\varphi(\lambda x.(x \textbf{ or } \Omega)))\} \\ &= \{just\ (\lambda x.LT(x \textbf{ or } \Omega))\} \\ &= \{just\ (\lambda x.\{just\ x, \bot\})\} \end{aligned}$$

$$\begin{aligned} LT((\lambda x.x) \textbf{ or } (\lambda x.\Omega)) &= \{just\ (\varphi(\lambda x.x)), just\ (\varphi(\lambda x.\Omega))\} \\ &= \{just\ (\lambda x.\{just\ x\}), just\ (\lambda x.\{\bot\})\}. \end{aligned}$$

Omitting *just* annotations, we can represent $LT(\lambda x.(x \textbf{ or } \Omega))$ and $LT((\lambda x.x) \textbf{ or } (\lambda x.\Omega))$ as follows:

$$LT(\lambda x.(x \textbf{ or } \Omega)) = \left\{ \begin{array}{c} \lambda x \\ | \\ \{x, \bot\} \end{array} \right\} \qquad LT((\lambda x.x) \textbf{ or } (\lambda x.\Omega)) = \left\{ \begin{array}{ccc} \lambda x & & \lambda x \\ | & , & | \\ \{x\} & & \{\bot\} \end{array} \right\}$$

Similarly, considering the partial distribution monad $\mathbb{D}\mathbb{M}$ (and mitting *just* annotations) we can represent $LT(\lambda x.(x \textbf{ or } \Omega))$ and $LT((\lambda x.x) \textbf{ or } (\lambda x.\Omega))$ as follows:

$$LT(\lambda x.(x \textbf{ or } \Omega)) = 1 \cdot \left(\begin{array}{c} \lambda x \\ | \\ \left(\frac{1}{2} \cdot x + \frac{1}{2} \cdot \bot\right) \end{array} \right) \qquad LT((\lambda x.x) \textbf{ or } (\lambda x.\Omega)) = \frac{1}{2} \cdot \left(\begin{array}{c} \lambda x \\ | \\ 1 \cdot x \end{array} \right) + \frac{1}{2} \left(\begin{array}{c} \lambda x \\ | \\ 1 \cdot \bot \end{array} \right)$$

⊠

Comparing effectful Lévy-Longo tree with plain equality can be easily seen to be too discriminating. For instance, $\lambda x.(x \textbf{ or } \Omega)$ and $\lambda x.x$ have different Lévy-Longo trees, but can be easily seen to be whnf-bisimilar. As a consequence, our notions of effectful Lévy-Longo tree equivalence and approximation are defined with respect to a relator Γ.

Definition 58. Lévy-Longo value tree approximation *with respect to* Γ *(Lévy-Longo value tree approximation, for short)* $\preceq^{\mathsf{VLT}} : \mathcal{VT}(\mathbb{T}) \nrightarrow \mathcal{VT}(\mathbb{T})$ *is defined as the largest relation* $\mathcal{R} : \mathcal{VT}(\mathbb{T}) \nrightarrow \mathcal{VT}(\mathbb{T})$ *such that:*

$$\begin{aligned} \lambda x.\mathscr{E}\ \mathcal{R}\ \mathfrak{F} &\implies \exists \mathscr{F} \in T(\mathcal{VT}(\mathbb{T})).\ \mathfrak{F} = \lambda x.\mathscr{F} \wedge \mathscr{E}\ \Gamma\mathcal{R}\ \mathscr{F} \\ x\langle \mathscr{E}_1, \ldots, \mathscr{E}_n \rangle\ \mathcal{R}\ \mathfrak{F} &\implies \exists \mathscr{G}_1, \ldots, \mathscr{G}_n \in T(\mathcal{VT}(\mathbb{T})).\ \mathfrak{F} = x\langle \mathscr{G}_1, \ldots, \mathscr{G}_n \rangle \wedge \forall i \leq n.\ \mathscr{E}_i\ \Gamma\mathcal{R}\ \mathscr{G}_i. \end{aligned}$$

As usual, Definition 58 induces a monotone endofunction $\mathcal{R} \mapsto [\mathcal{R}]$ (monotonicity being a consequence of monotonicity of Γ) on the complete lattice of binary relations on value trees. As a consequence, we see that $\preceq^{\mathsf{VLT}}$ is nothing but the greatest fixed point of $\mathcal{R} \mapsto [\mathcal{R}]$. We can finally extract a λ-term relation from $\preceq^{\mathsf{VLT}}$ that provides an alternative characterisation of whnf-similarity.

Definition 59. *Define* effectful Lévy-Longo tree approximation *with respect to* Γ *(Lévy-Longo tree approximation, for short) as the* λ*-term relation* $\preceq^{\mathsf{LT}} = (\preceq^{\mathsf{LT}}_{\Lambda}, \preceq^{\mathsf{LT}}_{W})$*, defined by:*

$$\begin{aligned} \preceq^{\mathsf{LT}}_{\Lambda} &\triangleq LT^{\circ} \cdot \Gamma\preceq^{\mathsf{VLT}} \cdot LT, \\ \preceq^{\mathsf{LT}}_{W} &\triangleq \varphi^{\circ} \cdot \preceq^{\mathsf{VLT}} \cdot \varphi. \end{aligned}$$

That is:

$$\begin{aligned} e \preceq^{\mathsf{LT}}_{\Lambda} f &\iff LT(e)\ \Gamma\preceq^{\mathsf{VLT}}\ LT(f) \\ s \preceq^{\mathsf{LT}}_{W} t &\iff \varphi(s) \preceq^{\mathsf{VLT}} \varphi(t). \end{aligned}$$

Theorem 14. *Lévy-Longo tree approximation $\preceq^{\mathrm{LT}}$ coincide with whnf-similarity $\preceq^{\mathrm{W}}$.*

Proof. The proof is entirely by coinduction and relies on stability of Γ. Let us start by showing $\preceq^{\mathrm{W}} \subseteq \preceq^{\mathrm{LT}}$, i.e. that the following implications hold:

$$\begin{aligned} e \preceq^{\mathrm{W}}_{\Lambda} f &\implies LT(e)\ \Gamma \preceq^{\mathrm{VLT}} LT(f) \\ s \preceq^{\mathrm{W}}_{\mathcal{W}} t &\implies \varphi(s)\ \preceq^{\mathrm{VLT}} \varphi(t). \end{aligned}$$

Since $\preceq^{\mathrm{VLT}}$ is defined coinductively, to prove the above implications it is sufficient to exhibit a relation $\mathcal{R} : \mathcal{VT}(\mathbb{T}) \to \mathcal{VT}(\mathbb{T})$ satisfying proof obligations in Definition 58 and such that

$$\begin{aligned} \preceq^{\mathrm{W}}_{\Lambda} &\subseteq LT^{\circ} \cdot \Gamma\mathcal{R} \cdot LT \\ \preceq^{\mathrm{W}}_{\mathcal{W}} &\subseteq \varphi^{\circ} \cdot \mathcal{R} \cdot \varphi. \end{aligned}$$

In fact, since $\mathcal{R}$ satisfies proof obligations in Definition 58, by coinduction we have have $\mathcal{R} \subseteq \preceq^{\mathrm{VLT}}$ and thus $\Gamma\mathcal{R} \subseteq \Gamma\preceq^{\mathrm{VLT}}$, as Γ is monotone. Define $\mathcal{R}$ as $(\varphi \times \varphi)(\preceq^{\mathrm{W}}_{\mathcal{W}})$, i.e.

$$\mathcal{R} \triangleq \{(\varphi(s), \varphi(t)) \mid s \preceq_{\mathcal{W}} t\}.$$

We claim that $\mathcal{R}$ satisfies proof obligations in Definition 58.

1. Suppose $\lambda x.\mathscr{E}\ \mathcal{R}\ \varphi(t)$. We prove that there exists $\mathscr{F} \in T(\mathcal{VT}(\mathbb{T}))$ such that $\varphi(t) = \lambda x.\mathscr{F}$ and $\mathscr{E}\ \Gamma\mathcal{R}\ \mathscr{F}$. By very definition of φ, there exists a term e such that $\mathscr{E} = LT(e)$ and $\lambda x.\mathscr{E} = \varphi(\lambda x.e)$. By very definition of $\mathcal{R}$ we have $\lambda x.e \preceq^{\mathrm{W}}_{\mathcal{W}} t$, and thus $t = \lambda x.f$, for some term f such that $e \preceq^{\mathrm{W}}_{\Lambda} f$. In order to conclude the desired thesis we have to show $LT(e)\ \Gamma\mathcal{R}\ LT(f)$. We have:

$$\begin{aligned} LT(e)\ \Gamma\mathcal{R}\ LT(f) &\iff T(\varphi)[\![e]\!]\ \Gamma\mathcal{R}\ T(\varphi)[\![f]\!] \\ &\quad \text{[By Definition 57]} \\ &\iff [\![e]\!]\ \Gamma(\varphi^{\circ} \cdot \mathcal{R} \cdot \varphi)\ [\![f]\!] \\ &\quad \text{[By (stability)]} \\ &\iff [\![e]\!]\ \Gamma\preceq^{\mathrm{W}}_{\mathcal{W}}\ [\![f]\!] \\ &\quad [\varphi^{\circ} \cdot \mathcal{R} \cdot \varphi\ =\ \preceq^{\mathrm{W}}_{\mathcal{W}}] \\ &\impliedby e \preceq^{\mathrm{W}}_{\Lambda} f. \end{aligned}$$

2. We can prove that $x\langle \mathscr{E}_1, \ldots, \mathscr{E}_n \rangle$ implies that there exist $\mathscr{F}_1, \ldots, \mathscr{F}_n \in T(\mathcal{VT}(\mathbb{T}))$ such that $\varphi(t) = x\langle \mathscr{F}_1, \ldots, \mathscr{F}_n \rangle$ and $\mathscr{E}_i\ \Gamma\mathcal{R}\ \mathscr{F}_i$, for any $i \leq n$, as we did in point 1.

Obviously $\preceq^{\mathrm{W}}_{\mathcal{W}} \subseteq \varphi^{\circ} \cdot \mathcal{R} \cdot \varphi$. It remains to prove that $e \preceq_{\Lambda} f$ implies $LT(e)\,\Gamma\mathcal{R}\,LT(f)$, but that is exactly what we showed in point 1 above. We now show that $\preceq^{\mathrm{LT}}$ is a whnf-simulation, from which $\preceq^{\mathrm{LT}} \subseteq \preceq^{\mathrm{W}}$ will follow. Proving that $\preceq^{\mathrm{LT}}$ respects whnfs is straightforward. It remains to show that $e \preceq^{\mathrm{LT}}_{\Lambda} f$ implies $[\![e]\!]\ \Gamma\preceq^{\mathrm{LT}}_{\mathcal{W}}\ [\![f]\!]$. We have:

$$\begin{aligned} e \preceq^{\mathrm{LT}}_{\Lambda} f &\iff LT(e)\ \Gamma \preceq^{\mathrm{VLT}} LT(f) \\ &\quad \text{[By Definition 59]} \\ &\iff T(\varphi)[\![e]\!]\ \Gamma\preceq^{\mathrm{VLT}} T(\varphi)[\![f]\!] \\ &\quad \text{[By Definition 57]} \\ &\iff [\![e]\!]\ \Gamma(\varphi^{\circ} \cdot \preceq^{\mathrm{VLT}} \varphi)\ [\![f]\!] \\ &\quad \text{[By (stability)]} \\ &\iff [\![e]\!]\ \Gamma\preceq^{\mathrm{LT}}_{\mathcal{W}}\ [\![f]\!]. \\ &\quad \text{[By Definition 59]} \end{aligned}$$

□

Finally, we define *effectful Lévy-Longo value tree equivalence with respect to* Γ, denoted as $\simeq^{\mathsf{VLT}}$, as the largest symmetric effectful Lévy-Longo tree approximation. The relation $\simeq^{\mathsf{VLT}}$ induces a λ-term relation $\simeq^{\mathsf{LT}}$ as in Definition 59. Moreover, a slightly modification of Theorem 14 allows us to conclude the following result.

Theorem 15. *Effectful Lévy-Longo tree equivalence* $\simeq^{\mathsf{LT}}$ *coincides with whnf-bisimilarity* $\simeq^{\mathsf{W}}$.

We conclude this section as well as this chapter, with a short discussion on the analogous of effectful Lévy-Longo tree equivalence for enf-bisimilarity. As we remarked above, Lévy-Longo trees have been introduced as the analogous of Böhm trees for the so-called lazy λ-calculus, where one works with weak head normal forms, rather than with head normal forms. To the best of the author's knowledge, the notion of a Böhm tree has not been adapted to the eager λ-calculus of (S. B. Lassen, 2005), where weak head normal forms are replaced by eager normal forms. The reason why this has not been done is rather evident: because such trees would faithfully mimic the bisimulation game associated with eager normal form bisimulation. To see that, let us define the *eager tree* $ET(e)$ of a *pure* λ-term e.

Definition 60. *The* eager tree $ET(e)$ *of a pure term* e *is the (possibly) infinitary tree with edges possibly labelled by variables coinductively defined as follows:*

$$ET(e) \triangleq \begin{cases} \bot & \text{if } [\![e]\!] = \bot \\ \varphi(s) & \text{if } [\![e]\!] = \mathit{just}\ s \end{cases}$$

where the function φ *is defined as follows, where* $z \notin FV(E)$*:*

$$\varphi(x) \triangleq x \qquad \varphi(\lambda x.e) \triangleq \begin{array}{c} \lambda x \\ | \\ ET(e) \end{array} \qquad \varphi(E[xv]) \triangleq \begin{array}{ccc} & & x \\ & z\swarrow & \\ ET(E[z]) & & \varphi(v) \end{array}$$

Definition 60 has been proposed by Ronchi della Rocca to Sangiorgi and Durier[7], and by the latter to the author in a private communication.

Example 52. We can use Definition 60 to see that the applicatively bisimilar terms xv and $(\lambda y.xv)(xv)$ previously discussed have different eager trees.

$$ET(xv) = \left(\begin{array}{ccc} & & x \\ & z\swarrow & \\ z & & \varphi(v) \end{array} \right) \qquad ET(\lambda y.xv)(xv) = \left(\begin{array}{ccccc} & & & & x \\ & & & z\swarrow & \\ & & x & & \varphi(v) \\ & z'\swarrow & & & \\ z' & & \varphi(v) & & \end{array} \right)$$

In a similar fashion, we can prove the equivalence between the (pure) fixed point combinators Y and Z

[7] They refer to $ET(e)$ as the *Lassen tree* of e.

of Examplex 41, observing that they have the same eager tree.

$$LT(Y) = \left(\begin{array}{ccccccc} & & \lambda y & & & & \\ & & | & & & & \\ & & y & & & & \\ & {\scriptstyle y'} \swarrow & & \searrow & & & \\ y' & & & & \lambda z & & \\ & & & & | & & \\ & & & & y & & \\ & & & {\scriptstyle z'} \swarrow & & \searrow & \\ & & z' & & & & \lambda z \\ & {\scriptstyle z''} \swarrow & & \searrow & & & | \\ z'' & & & & z & & y \\ & & & & & {\scriptstyle z'} \swarrow & & \searrow \\ & & & & z' & & & \lambda z \\ & & & {\scriptstyle z''} \swarrow & & \searrow & & | \\ & & z'' & & & & z & \vdots \end{array} \right) = LT(Z)$$

⊠

Finally, it is straightforward to see that we can generalise Definition 60 to full $\Lambda_{\Sigma}^{(v)}$ simply defining $ET(e)$ as $T(\varphi)[\![e]\!]$. It comes with no surprise that we can define notions of eager tree approximation and equivalence, and that the latter coincide with $\preceq^{E}$ and $\simeq^{E}$, respectively.

Chapter 8

Midterm Discussion

μέσον τε καὶ ἄριστον

Aristotle, Nicomachean Ethics

Chapter 7 is the last chapter of this dissertation that deals with notions of program equivalence and program refinement. Chapter 9 to Chapter 12 are entirely dedicated to the study of the so-called *program metrics* or *program distances*.

Here we summarise the main results proved so far, and their relationship. In fact, we studied several notions of program equivalence and approximation, and, although several soundness and completeness (i.e. full abstraction) theorems have been proved, it is useful for the reader to have a conceptual map of the results obtained.

Concerning program equivalence and refinements for *call-by-value* calculi, we introduced:

- Effectful applicative similarity $\preceq^{A}$ and bisimilarity $\simeq^{A}$.
- Effectful contextual approximation $\preceq^{ctx}$ and equivalence $\simeq^{ctx}$.
- Effectful CIU approximation $\preceq^{ciu}$ and equivalence $\simeq^{ciu}$.
- Effectful eager normal form similarity $\preceq^{E}$ and bisimilarity $\simeq^{E}$.

With the exception of CIU approximation (resp. equivalence) and contextual approximation (resp. equivalence), these relations do not coincide, in general. In particular, we have:

- (The open extension of) effectful applicative similarity (resp. bisimilarity) is strictly included in effectful contextual approximation (resp. equivalence). Inclusion follows from Theorem 4 and Theorem 5, whereas strictness has been proved for the nondeterministic calculus $\Lambda_{\mathbb{FM}}$ in (S. Lassen, 1998b) (Example 6.4.4).
- Effectful eager normal form similarity (resp. bisimilarity) is strictly included in (the open extension of) effectful applicative similarity (resp. bisimilarity). Inclusion follows by Proposition 22, whereas strictness has been in shown in (S. B. Lassen, 2005) for the pure, untyped λ-calculus (see also the discussion after Proposition 22).
- (The open extension of) effectful CIU approximation (resp. equivalence) coincides with contextual approximation (resp. equivalence). This follows from Theorem 6.

These results are summarised in Table 8.1. Concerning program equivalence and refinements for *call-by-name* calculi, we introduced:

$\preceq^{E}$	$\subsetneq$	$(\preceq^{A})^{o}$	$\subsetneq$	$(\preceq^{ciu})^{o}$	$=$	$\preceq^{ctx}$
$\simeq^{E}$	$\subsetneq$	$(\simeq^{A})^{o}$	$\subsetneq$	$(\simeq^{ciu})^{o}$	$=$	$\simeq^{ctx}$

Table 8.1: Call-by-value refinement/equivalence spectrum.

- Effectful applicative similarity $\preceq^{A}$ and bisimilarity $\simeq^{A}$ (we only sketched the definition).
- Monadic applicative similarity $\preceq^{M}$ and bisimilarity $\simeq^{M}$.
- Effectful contextual approximation $\preceq^{ctx}$ and equivalence $\simeq^{ctx}$.
- Effectful CIU approximation $\preceq^{ciu}$ and equivalence $\simeq^{ciu}$.
- Effectful weak head normal form similarity $\preceq^{W}$ and bisimilarity $\simeq^{W}$.
- Effectful Lévy-Longo tree approximation $\preceq^{LT}$ and equivalence $\simeq^{LT}$.

The relationship between these equivalences and refinements is the following, where we assume relators to be quasi-flat.

- (The open extension of) monadic applicative similarity (resp. bisimilarity), effectful contextual approximation (resp. equivalence), and (the open extension of) CIU approximation (resp. equivalence) coincide. This follows from Theorem 8, Corollary 4, and Corollary 5.
- Effectful applicative similarity (resp. bisimilarity) is strictly included in monadic applicative similarity (resp. bisimilarity), and thus in effectful contextual approximation (resp. equivalence), and CIU approximation (resp. equivalence). Strictness has been showed for $\Lambda_{\mathbb{FM}}$ at the beginning of Chapter 6.
- Effectful weak head normal form similarity (resp. bisimilarity) is strictly included in (the open extension of) effectful applicative similarity (resp. bisimilarity). Inclusion follows by mimicking the proof of Proposition 22, whereas strictness has been proved in (Abramsky & Ong, 1993), Example 2.3.10.
- Effectful Lévy-Longo tree approximation (resp. equivalence) coincides with effectful weak head normal form similarity (resp. bisimilarity).

These results are summarised in table 8.2.

$\preceq^{LT}$	$=$	$\preceq^{W}$	$\subsetneq$	$(\preceq^{A})^{o}$	$\subsetneq$	$(\preceq^{M})^{o}$	$=$	$(\preceq^{ciu})^{o}$	$=$	$\preceq^{ctx}$
$\simeq^{LT}$	$=$	$\simeq^{W}$	$\subsetneq$	$(\simeq^{A})^{o}$	$\subsetneq$	$(\simeq^{M})^{o}$	$=$	$(\simeq^{ciu})^{o}$	$=$	$\simeq^{ctx}$

Table 8.2: Call-by-name refinement/equivalence spectrum.

Chapter 9

A Theory of Abstract Behavioural Metrics

> Algebra is the intellectual instrument which has been created for rendering clear the quantitative aspects of the world.
>
> Alfred North Whitehead, The Aims of Education

The rest of this dissertation is devoted to the study of effectful applicative (bi)similarity distance, the quantitative-like refinement of effectful applicative (bi)similarity, as defined in Chapter 5. Roughly speaking, such a refinement is obtained by moving from ordinary boolean-valued relations to relations taking values over arbitrary quantitative domains (such as the real extended half-line $[0, \infty]$ or the unit interval $[0, 1]$). Accordingly, the theory of effectful applicative (bi)similarity distance builds on three major improvements of the theory of effectful applicative (bi)similarity of Chapter 5.

1. The first improvement is to move from boolean-valued relations to relations taking values on quantitative domains, such as $[0, \infty]$ or $[0, 1]$, in such a way that restricting these domains to the (carrier of the) boolean algebra $\mathbb{2}$ makes the theory collapse to the usual theory of applicative (bi)similarity. For that, we rely on Lawvere's analysis (F. Lawvere, 1973) of generalised metric spaces and preordered sets as enriched categories. Accordingly, we replace boolean-valued relations with relations taking values over quantales (Rosenthal, 1990) (see Definition 61) $(\mathsf{V}, \leq, \otimes, k)$, i.e. algebraic structures (notably complete lattices equipped with a monoid structure) that play the role of sets of abstract quantities. Examples of quantales include the extended real half-line $([0, \infty], \geq, 0, +)$ ordered by the "greater or equal" relation $\geq$ and with monoid structure given by addition (and its restriction to the unit interval $[0, 1]$), and the extended real half-line $([0, \infty], \geq, 0, \max)$ with monoid structure given by binary maximum (in place of addition), as well as any complete Boolean and Heyting algebra. This allows to develop an algebra of quantale-valued relations, $\mathbb{V}$-relations for short, which provides a general framework for studying both behavioural relations and behavioural distances (for instance, an equivalence $\mathbb{V}$-relation instantiates to an ordinary equivalence relation on the boolean quantale $(\{\mathsf{false}, \mathsf{true}\}, \leq, \wedge, \mathsf{true})$, and to a pseudometric on the quantale $([0, \infty], \geq, 0, +)$).
2. The second improvement is the generalisation of the notion of relator to quantale-valued relators, i.e. relators acting on relations taking values over quantales. Perhaps surprisingly, such

generalisation is at the heart of the filed of *monoidal topology* (Hofmann et al., 2014), a subfield of categorical topology aiming to unify ordered, metric, and topological spaces in categorical terms. Central to the development of monoidal topology is the notion of a $\mathbb{V}$-relator or $\mathbb{V}$-lax extension of a monad $\mathbb{T}$ (with carrier T) which, analogously to the notion of relator, is a construction lifting $\mathbb{V}$-relations on a set X to $\mathbb{V}$-relations on TX. Notable examples of $\mathbb{V}$-relators are obtained from the Hausdorff distance (for the powerset monad) and from the Wasserstein-Kantorovich distance (Villani, 2008) (for the distribution monad).

3. The third improvement (on which we will expand more in Section 9.2) is the development of a *compositional* theory of behavioural $\mathbb{V}$-relations (and thus of behavioural distances). As we are going to see, ensuring compositionality in an higher-order setting is particularly challenging due to the ability of higher-order programs to copy their input several times, a feature that allows them to amplify distances between their inputs *ad libitum*.

The result is an abstract theory of behavioural $\mathbb{V}$-relations that allows to define notions of quantale-valued applicative similarity and bisimilarity parametrised by a quantale-valued relator. The notions obtained, which we refer to as *effectful applicative similarity* and *bisimilarity*, respectively, generalise the existing notions of real-valued applicative (bi)similarity and can be instantiated to concrete calculi to provide new notions of applicative (bi)similarity distance. A remarkable example is the case of probabilistic λ-calculi, where to the best of the author's knowledge a (non-trivial) applicative distance for a universal (i.e. Turing complete) probabilistic λ-calculus is still lacking in the literature (but see Section 13.1).

As for Chapter 5, the main theorem we prove states that under suitable conditions on monads and quantale-valued relators, the abstract notion of effectful applicative similarity distance is a compatible — i.e. compositional — reflexive and transitive $\mathbb{V}$-relation. Under mild conditions such a result extends to effectful applicative bisimilarity distance, which is thus proved to be a compatible, reflexive, symmetric, and transitive $\mathbb{V}$-relation (i.e. a compatible pseudometric).

This chapter aims to provide an extensive, informal introduction to Chapter 10, Chapter 11, and Chapter 12, which, instead, are devoted to the technical exposition of the theory of effectful applicative (bi)similarity distance.

9.1 An Abstract Theory of Distances

The observation that ordered and metric spaces share a common structure dates back at least to Hausdorff (Hausdorff, 1949). In fact, Hausdorff observed that an order on a set X can be described as a function $f : X \times X \to \{<, =, >\}$. Accordingly, Hausdorff presented metric spaces as a direct generalisation of ordered sets where now the map f associates to each pair of points in X the distance between them. Decades later, this simple analogy has been extended by Lawvere (F. Lawvere, 1973) to a formal correspondence. Metric and ordered spaces not only share the same structure, but also the same axioms. Moreover, such axioms are nothing but a concrete instance of the definition of an *enriched category* (Kelly, 2005).

In ordinary category theory, each category $\mathbb{C}$ comes with a map, the so-called hom-map $\mathbb{C}(-,-)$, that associates to each pair of objects X, Y of $\mathbb{C}$ a *set*, namely the set of morphism from X to Y. Such hom-sets must satisfy specific conditions; for any object X there is an identity morphism 1_X belonging to $\mathbb{C}(X, X)$. Additionally, for all objects X, Y, and Z there is a binary operation $\circ$ associating to morphisms $g \in \mathbb{C}(Y, Z)$ and $f \in \mathbb{C}(X, Y)$ a morphism $g \circ f \in \mathbb{C}(X, Z)$, called the composition of f with g. Enriched category theory moves from the observation that there is no formal reason to require hom-sets to be sets. Hom-sets are then replaced by objects in another category V. In that case, we say that $\mathbb{C}$ is enriched over V. However, the category V needs to have enough structure to allow the definition of identity

morphisms and composition operations. For that, it is sufficient to require V to be a monoidal category. The existence of a composition operation can be then formalised as a family of morphisms

$$m_{X,Y,Z} : \mathbb{C}(Y,Z) \otimes \mathbb{C}(X,Y) \to \mathbb{C}(X,Z)$$

in V, whereas identity is given by morphisms

$$i_X : I \to \mathbb{C}(X,X).$$

The coherence conditions of V ensures m and i to behave as composition and identity, respectively. For instance, the category Set being monoidal, it is easy to see that a locally small category is nothing but a category enriched over Set.

A more interesting example of an enriched category is provided by preorders. Recall that the two-element boolean algebra $\mathbb{2}$ can be seen as a category with two objects (0 and 1, or false and true), and a single morphism $x \to y$ if $x \leq y$. Moreover, $\mathbb{2}$ is monoidal, with tensor product given by binary meet $\wedge$ and unit given by true. A preorder is thus a category X enriched over $\mathbb{2}$. The hom-set $X(x,y) \in 2$ denotes the truth value of the dominance of y over x. Accordingly, identity and composition are modelled by the 2-morphisms:

$$\begin{aligned} \mathsf{true} &\leq X(x,x) \\ X(y,z) \wedge X(x,y) &\leq X(x,z), \end{aligned}$$

which are nothing but the usual axioms stating that the order $X(-,-)$ is transitive and reflexive.

Lawvere's key insight is that another example of an enriched category is provided by what he called *generalised metric spaces*, i.e. pairs of the form (X,μ), with $\mu : X \times X \to [0,\infty]$, satisfying the following subset of the usual metric axioms (where x,y,z are universally quantified):

$$\begin{aligned} 0 &\geq \mu(x,x) \\ \mu(x,y) + \mu(y,z) &\geq \mu(x,z). \end{aligned}$$

In order to see the enriched nature of generalised metric spaces, we first observe that the extended non-negative real line $[0,\infty]$ carries a monoidal category structure. Arrows are given according to the opposite of the natural ordering $\geq$ (so that 0 is the terminal object, and ∞ is the initial one), whereas the tensor product is given by addition, so that the unit is 0. In a category X enriched over $[0,\infty]$, the hom-set $X(x,y) \in [0,\infty]$ gives the 'distance' between x and y. Identity and composition are thus given as the morphisms:

$$\begin{aligned} 0 &\geq X(x,x) \\ X(y,z) + X(x,y) &\geq X(x,z), \end{aligned}$$

which are nothing but the usual axioms for identity and triangle inequality. We can also notice that there is another natural way to give the set $[0,\infty]$ a monoidal category structure, namely replacing addition (i.e. the tensor product) with the binary maximum operator max. Composition now gives

$$\max(X(y,z), X(x,y)) \geq X(x,z)$$

which is nothing but the usual *strong triangle inequality* axiom characterising ultrametric spaces.

Moving form these observations Lawvere built a beautiful theory of categories enriched over complete and cocomplete monoidal categories that unifies the theory of locally small categories, preroder spaces, and generalised metric spaces (as well as the logic(s) associated with those structures). Here, we will be more concrete and work with relations taking values over *quantales* (Rosenthal, 1990), the latter being the decategorification of complete and cocomplete monoaidal categories.

More precisely, a quantale is a structure $\mathbb{V} = (\mathsf{V}, \leq, \otimes, k)$ such that $(\mathsf{V}, \leq)$ is a complete lattice and $(\mathsf{V}, \otimes, k)$ is a monoid. We require the complete lattice and monoid structure of quantale to nicely interact, meaning that some suitable distributivity laws has to be satisfied (see Definition 61 for details). For the moment, we simply observe that the structures $\mathbb{2} = (2, \leq, \wedge, \mathrm{true})$, $\mathbb{L} = ([0, \infty], \geq, +, 0)$, and $\mathbb{SL} = ([0, \infty], \geq, \max, 0)$ are quantales. We refer to them as the *boolean*, *Lawvere*, and *strong Lawevere* quantale, respectively. Finally, for a quantale $\mathbb{V}$ as above, we define a $\mathbb{V}$-*relation* between X and Y to be a map $\alpha : X \times Y \to \mathsf{V}$.

Intuitively quantales model abstract quantities, and thus quantale-valued relations can be used as models of abstract distances. On a formal level, the structure of a quantale V allows to generalise the usual algebra of boolean-valued relations to $\mathbb{V}$-relations.

The correspondence between $\mathbb{2}$, $\mathbb{L}$, $\mathbb{SL}$, and an arbitrary quantale $\mathbb{V} = (\mathsf{V}, \leq, \otimes, k)$ is summarised in Table 9.1.

	$\mathbb{2}$ (boolean)	$\mathbb{L}$ (Lawvere)	$\mathbb{L}$ (strong Lawvere)	$\mathbb{V}$ (quantale)
Carrier	2	$[0, \infty]$	$[0, \infty]$	V
Order	$\leq$	$\geq$	$\geq$	$\leq$
Join	$\exists$	inf	inf	$\bigvee$
Meet	$\forall$	sup	sup	$\bigwedge$
Tensor	$\wedge$	$+$	max	$\otimes$
Unit	true	0	0	k

Table 9.1: Correspondence 2-$[0, \infty]$-V.

We can use Table 9.1 as a recipe to lift the algebra of boolean-valued relations to $\mathbb{V}$-relations, as well as to translate standard relational constructions into their quantitative counterparts.

For instance, it is straightforward to generalise the usual notion of relation composition. In fact, the latter is defined using only the join ($\exists$) and binary meet ($\wedge$) of $\mathbb{2}$, and both these notions are defined in any quantale. Applying Table 9.1 to the definition of relation composition we thus obtain the following definitions of (generalised) composition, where $\mathcal{R}, \mathcal{S}$ are $\mathbb{2}$-relations, μ, ν are $\mathbb{L}$-relations, p, s are $\mathbb{SL}$-relations, and α, β are $\mathbb{V}$-relations we have:

$$
\begin{aligned}
(\mathcal{S} \cdot \mathcal{R})(x, z) &\triangleq \exists y.\ \mathcal{R}(x, y) \wedge \mathcal{S}(y, z) \\
(\nu \cdot \mu)(x, z) &\triangleq \inf_y \mu(x, y) + \nu(y, z) \\
(p \cdot s)(x, z) &\triangleq \inf_y \max(s(x, y), p(y, z)) \\
(\beta \cdot \alpha)(x, z) &\triangleq \bigvee_y \alpha(x, y) \otimes \beta(y, z).
\end{aligned}
$$

As a consequence, we can say that a $\mathbb{V}$-relation α is transitive if $\alpha \cdot \alpha \leq \alpha$. It is straightforward to see that $\mathbb{V}$-relation transitivity generalises the usual notion of transitivity of boolean-valued relations as well as the well-known triangle and strong triangle inequality formulas.

Extending this line of reasoning to the notions of symmetry and reflexivity, we come up with pre-order and equivalence $\mathbb{V}$-relations (oftentimes abbreviated as $\mathbb{V}$-preorders and $\mathbb{V}$-equivalences). These generalise standard preorder and equivalence relations, as well as generalised (ultra)metrics, and their symmetric extensions (known as (ultra-)pseudometrics (Steen & Seebach, 1995)). As a consequence, our notions of quantale-valued program equivalence and refinement provide abstract notions of program distance, the latter subsuming both standard, boolean-valued program equivalence and refinement, and program metric (the latter, as it is customary, denoting both program generalised (ultra)metrics and program (ultra-)pseudometrics).

Having clarified the basic intuitions behind the abstract approach to relations and distances used in this work, we now discuss the so-called distance amplification phenomena.

9.2 Compositionality, Distance Amplification, and Linear Types

As usual, we begin our analysis with an informal example giving some insights on the nature of the notions we aim to investigate. The informal example we study here is the calculus Λ_p of Chapter 2 (and actually is equivalent formulation $\Lambda_{\mathbb{DM}}$). Our goal is to define a notion of program distance δ between terms of Λ_p refining probabilistic applicative bisimilarity. Intuitively, $\delta(e, e')$ quantifies the observable differences between (the operational behaviour of) e and (the operational behaviour of) e'. Since we postulated the probability of convergence to be the only observable property of a program, it is natural to let $\delta(e, e')$ be a non-negative real number between 0 and 1. Following Section 9.1, we see that $[0, 1]$ carries the same quantale structure of $\mathbb{L}$, the only difference being that the tensor product on $[0, 1]$ is defined as *truncated addition*[1]. We call the quantale thus obtained *unit interval quantale*, and denote it as $\mathbb{I}$. We have thus obtained our first assumption.

Assumption 1. *A program distance δ in Λ_p is a pair of functions $(\delta^\Lambda, \delta^\mathcal{V})$, where $\delta^\Lambda : \Lambda_\circ \times \Lambda_\circ \to [0, 1]$ and $\delta^\mathcal{V} : \mathcal{V}_\circ \times \mathcal{V}_\circ \to [0, 1]$.*

We now look at some minimal desiderata a candidate distance δ should satisfy. First of all, since we observe the probability of convergence of programs, it is natural to require $\delta^\Lambda(e, e')$ to be at least as big as the absolute value of the difference between the probability of convergence of e and of e'. In fact, the latter is somehow the minimal distance an external observer can detect, even without interacting with e and e'. We are thus requiring δ to be adequate with respect to the observable (operational) behaviour of programs in Λ_p.

Assumption 2. *Writing $\sum[\![e]\!]$ for $\sum_{v \in \mathcal{V}_\circ}[\![e]\!](just\ v)$, we say that a program distance $\delta = (\delta^\Lambda, \delta^\mathcal{V})$ is adequate if:*

$$|\sum[\![e]\!] - \sum[\![e']\!]| \leq \delta(e, e').$$

Next, we ask if and how much the distance $\delta^\Lambda(e, e')$ is modified when e and e' are used inside a context C. Indeed we would like to reason about the distance $\delta^\Lambda(C[e], C[e'])$ *compositionally*, i.e. in terms of the distance $\delta^\Lambda(e, e')$. As we have already remarked, compositionality is at the heart of relational reasoning about program behaviour, and most of the main results proved in previous chapters are concerned with showing compositionality of several forms of (bi)simulation-based equivalence and refinement relations. Formally, we modelled compositionality via the notion of compatibility, a relation being compatible if it is preserved by all language constructors.

Analogous to the idea that compatible relations are preserved by language constructors, we are tempted to define as compatible those distances that are not increased by language constructors. That is, we would like to say that a behavioural distance δ is compatible if the distance $\delta(C[e], C[e'])$ between $C[e]$ and $C[e']$ is always bounded by the distance $\delta(e, e')$, no matter how $C[-]$ uses e and e'. This intuition gives our third assumption.

Assumption 3. *A program distance $\delta = (\delta^\Lambda, \delta^\mathcal{V})$ is compatible if for all programs e, e' and any context C, we have:*

$$\delta^\Lambda(C[e], C[e']) \leq \delta^\Lambda(e, e').$$

[1] Truncated addition is defined as $\min(x + y, 1)$: we overload the notation and denote by $+$ both ordinary and truncated addition.

Equivalently, δ is compatible if for all programs e, e' we have:

$$\sup_C \delta^\Lambda(C[e], C[e']) \leq \delta^\Lambda(e, e').$$

Notice that Assumption 3 requires δ to be *non-expansive* with respect to all language constructors. At this point we encounter a serious problem: as noticed in (Crubillé & Dal Lago, 2015, 2017), requiring Assumption 1, Assumption 2, and Assumption 3 can lead to some *distance trivialisation* phenomena. Roughly speaking, we say that a program distance δ trivialises if for all programs e, e', either $\delta^\Lambda(e, e') = 0$ or $\delta^\Lambda(e, e') = 1$. That is, interpreting 0 as true and 1 as false, δ trivialises if it behaves as a (boolean-valued) relation.

Distance trivialisation is a consequence of higher-features of calculi. In fact, higher-order programs can freely copy and duplicate their input several times, thus having the testing power to amplify distances between their inputs *ad libitum*. Here we are not concerned with distance trivialisation in full generality (the interested reader can refer to (Crubillé & Dal Lago, 2017) for a formal analysis of distance trivialisation). For our purposes, it is sufficient to notice that Assumption 1, Assumption 2, and Assumption 3 make any candidate distance not satisfactory. We clarify this point with the following example taken from (Crubillé & Dal Lago, 2017), and here (re)formulated in the calculus Λ_p.

Example 53. Let δ be a program distance satisfying Assumption 1, Assumption 2, and Assumption 3. Let I be the identity combinator $\lambda x.\mathbf{return}\ x$ and Ω be the purely divergent computation $(\lambda x.xx)(\lambda x.xx)$. As usual, we see that $(\mathbf{return}\ I)\ \mathbf{or}\ \Omega$ evaluates to I with probability $\frac{1}{2}$, and diverges with probability $\frac{1}{2}$. Since we observe the probability of convergence of a program, it speaks by itself that we would expect $\delta^\Lambda(\mathbf{return}\ I, (\mathbf{return}\ I)\ \mathbf{or}\ \Omega) = \frac{1}{2}$. However, we can easily prove $\delta^\Lambda(\mathbf{return}\ I, (\mathbf{return}\ I)\ \mathbf{or}\ \Omega) = 1$. Consider the family of contexts $\{C_n\}_{n\geq 0}$ defined by:

$$C_n \triangleq \lambda x.(\underbrace{(xI)\ldots(xI)}_{n})(\lambda y.[-]).$$

Each context C_n duplicates its input n times, meaning that the distance $\delta^\Lambda(\mathbf{return}\ I, (\mathbf{return}\ I)\ \mathbf{or}\ \Omega)$ is detected n times in C_n. Therefore, the resulting distance $\delta^\Lambda(C_n[\mathbf{return}\ I], C_n[(\mathbf{return}\ I)\ \mathbf{or}\ \Omega])$ is the observed (ground) distance between $\mathbf{return}\ I$ and $(\mathbf{return}\ I)\ \mathbf{or}\ \Omega$ amplified of a factor n. In fact, as n grows, the probability of convergence of $C_n[(\mathbf{return}\ I)\ \mathbf{or}\ \Omega]$ tends to 0, whereas the one of $C_n[\mathbf{return}\ I]$ remains always equal to 1. Formally, we have:

$$\begin{aligned}
\text{Assumption 2} &\implies \forall n \geq 0.\ |\sum [\![C_n[e]]\!] - \sum [\![C_n[e']]\!]| \leq \delta^\Lambda(C_n[e], C_n[e']) \\
&\implies \sup_n |\sum [\![C_n[e]]\!] - \sum [\![C_n[e']]\!]| \leq \sup_n \delta^\Lambda(C_n[e], C_n[e']) \\
&\quad [\text{By definition of sup}] \\
&\implies 1 \leq \sup_n \delta^\Lambda(C_n[e], C_n[e']) \\
&\quad [\text{Since } \sup_n |\sum [\![C_n[e]]\!] - \sum [\![C_n[e']]\!]| = 1] \\
&\implies 1 \leq \sup_C \delta^\Lambda(C[e], C[e']) \\
&\quad [\text{Since } \sup_n \delta^\Lambda(C_n[e], C_n[e']) \leq \sup_C \delta^\Lambda(C[e], C[e'])] \\
&\implies 1 \leq \delta^\Lambda(e, e'). \\
&\quad [\text{By Assumption 3}]
\end{aligned}$$

During its evaluation, every time the context C_n evaluates its inputs the detected distance between the latter is somehow accumulated to the distances previously observed, thus exploiting the *linear* —

as opposed to classical — nature of the act of measuring. Such linearity naturally reflects the monoidal closed structure of categories of metric spaces, in opposition with the cartesian closed structure characterising 'classical' (i.e. boolean-valued) observations. ⊠

The moral of Example 53 is that not only how a context uses a program matters, but also *how much* it uses the program does. As a consequence, if we want to reason compositionally about behavioural distances, we have to take into account *how much* programs can access their inputs.

We can give a precise meaning to the expression *how much* above through the notion of *program sensitivity* (de Amorim et al., 2017; Reed & Pierce, 2010). Intuitively, the sensitivity of program is the law describing how much differences in the output are affected by differences in the input. We thus use the notion of sensitivity to formalise the testing power of contexts. For instance, we can define the sensitivity of a context C in Λ_p as a non-negative real number s, and refine our notion of compatibility accordingly: we now allow a context C with sensitivity s to increase the distance $\delta(e, e')$ between e and e', but of a factor *at most* s. That is, the distance $\delta(C[e], C[e'])$ must be bounded by $s \cdot \delta(e, e')$.

Remark 16. It is important to stress that the sensitivity of a program in Λ_p is *not* the number of times the program accesses its input. For instance, we obviously expect the single-hole context $[-]$ to have sensitivity 1, and the zero-hole context v, for a closed value v, to have sensitivity 0. Up to this point indeed program sensitivity can be identified with the number of times inputs are used. However, it is reasonable to expect the context $[-]$ **or** v to have sensitivity $\frac{1}{2}$, since with probability $\frac{1}{2}$ it will test its input (once), and with probability $\frac{1}{2}$ it will not.

We can thus formulate a new notion of compatibility.

Assumption 4. *A program distance δ is compatible if for all programs e, e' and context C with sensitivity s,*

$$\delta(C[e], C[e']) \leq s \cdot \delta(e, e').$$

We also remark that there are other reasonable notions of program sensitivity: for instance, we could have defined the sensitivity of a context as a polynomial p bounding the testing power of the context. Our approach will be more abstract, allowing the sensitivity of a program to be any function satisfying some minimal structural properties, and thus leaving the freedom to choose the concrete notion of sensitivity that fits best the concrete calculus at hand. In fact, it turned out that program sensitivity can be seen as an instance of a more general notion, namely the one of a *coeffect*[2] (Brunel, Gaboardi, Mazza, & Zdancewic, 2014; Gaboardi, Katsumata, Orchard, Breuvart, & Uustalu, 2016; Petricek, Orchard, & Mycroft, 2014).

The introduction of the notion of program sensitivity seems to allow to resolve many issues related to the notion of compositionality as defined in Assumption 3. However, it also raises an important question: how can we track program sensitivity? To answer this question, we follow (Reed & Pierce, 2010) and refines Λ_p in a linear-like calculus. The resulting calculus has a powerful type systems inspired by bounded linear logic (Girard, Scedrov, & Scott, 1992) and, compared to other linear type systems, has the novelty of introducing types of the form $!_s\sigma$, where s is a non-negative real number modelling program sensitivity. Moreover, allowing the sensitivity of a program to be ∞, meaning that we are working with non-negative extended real numbers, we can model programs testing their input *ad libitum*. As a consequence, we can recover the full bang type $!\sigma$ as $!_\infty\sigma$.

[2] Unfortunately, the author was not aware of the connection between program sensitivity and coeffects until the moment of writing the present dissertation. As a consequence, a proper treatment of such a connection is not treated in this work.

9.3 Structure of the Analysis

At this point the reader should have enough intuitions and motivations to follow the rest of this dissertation. The latter is structured as follows. In Chapter 10 we introduce $\mathbb{V}$-relations and their algebra, as well as the notion of a *change of base functor* (Kelly, 2005; F. Lawvere, 1973). We will use the latter to define our abstract notion of program sensitivity. Having defined a notion of program sensitivity, we define our vehicle calculus $\mathbb{V}$-Fuzz. The latter is a generalisation of Reed's and Pierce's Fuzz (Reed & Pierce, 2010), a PCF-like language with a powerful linear type system enriched with sensitivity-indexed 'bang types' that allow to track program sensitivity. $\mathbb{V}$-Fuzz has two major differences compared to Fuzz. First, $\mathbb{V}$-Fuzz is parametrised by an arbitrary quantale and a signature of change of base functors. This allows $\mathbb{V}$-Fuzz to model several concrete calculi as well as several notions of program sensitivity. Secondly, and most importantly, $\mathbb{V}$-Fuzz is an effectful calculus parametrised by a monad and a signature of (algebraic) operation symbols. As a consequence, $\mathbb{V}$-Fuzz qualifies as a refinement of Λ_{Σ}, which we already argued by means of several examples to be good model for several effectful calculi. For instance, $\mathbb{V}$-Fuzz subsumes imperative, nondeterministic, and probabilistic versions of Fuzz, as well as combinations thereof.

Next, in Chapter 11 we introduce relators on $\mathbb{V}$-relations, which we call $\mathbb{V}$-relators. The resulting theory is the generalisation of the usual theory of relators as exposed in Chapter 4. We give examples of relevant relators, notably relators associated with the Hausdorff (Munkres, 2000) and Wasserstein (Villani, 2008) distance.

Finally, in Chapter 12 we define a notion of effectful applicative (bi)similarity distance for $\mathbb{V}$-Fuzz, i.e. a $\mathbb{V}$-relation (parametric with respect to a $\mathbb{V}$-relator) generalising the notion of effectful applicative (bi)similarity of Chapter 5. Our main theorem (Theorem 16) states that under suitable conditions on $\mathbb{V}$-relators, effectful applicative similarity distance is a compatible (according to a suitable generalisation of Assumption 4), reflexive, and transitive $\mathbb{V}$-relation, and thus gives a compatible generalised metric in the sense of (F. Lawvere, 1973). Similarly, we show that under mild conditions on change of base functors, effectful applicative bisimilarity distance is a compatible equivalence $\mathbb{V}$-relation (Theorem 17), and thus gives a compatible pseudometric.

Chapter 10

An Effectful Calculus with Program Sensitivity

In language there are only differences

Ferdinand de Saussure, Course in General Linguistics

In this chapter we introduce $\mathbb{V}$-relations and their algebra, as well as the notion of a *change of base functor* (Kelly, 2005; F. Lawvere, 1973). The latter provides the mathematical instrument we use to define our abstract notion of program sensitivity.

We use change of base functors to define our vehicle calculus $\mathbb{V}$-Fuzz. The latter is a generalisation of Reed's and Pierce's Fuzz (Reed & Pierce, 2010), a PCF-like language with a powerful linear type system enriched with sensitivity-indexed 'bang types' that allow to track program sensitivity. $\mathbb{V}$-Fuzz has two major differences compared to Fuzz. First, $\mathbb{V}$-Fuzz is parametrised by an arbitrary quantale and a signature of change of base functors. This allows $\mathbb{V}$-Fuzz to model several concrete calculi as well as several notions of program sensitivity. Secondly, and most importantly, $\mathbb{V}$-Fuzz is an effectful calculus parametrised by a monad and a signature of (algebraic) operation symbols. As a consequence, $\mathbb{V}$-Fuzz qualifies as a refinement of Λ_Σ, which we already argued by means of several examples to be good model for several effectful calculi. Finally, following Chapter 3, we give $\mathbb{V}$-Fuzz monadic operational semantics.

10.1 Quantales and Quantale-valued Relations

In this section we recall some necessary background notions on quantales (Rosenthal, 1990) and quantale-valued relations ($\mathbb{V}$-relations) along the lines of (F. Lawvere, 1973). The reader is referred to the monograph (Hofmann et al., 2014) for a comprehensive introduction.

Definition 61. *A (unital) quantale* $\mathbb{V} = (\mathsf{V}, \leq, \otimes, k)$ *consists of a monoid* $(\mathsf{V}, \otimes, k)$ *and a sup-lattice* $(\mathsf{V}, \leq)$ *satisfying the following distributivity laws:*

$$b \otimes \bigvee_{i \in I} a_i = \bigvee_{i \in I} (b \otimes a_i)$$
$$(\bigvee_{i \in I} a_i) \otimes b = \bigvee_{i \in I} (a_i \otimes b).$$

The element k is called unit of the quantale, whereas $\otimes$ is called multiplication or tensor of the quantale. Given quantales $\mathbb{V}, \mathbb{W}$, a quantale lax morphism *is a* monotone *map $h : \mathsf{V} \to \mathsf{W}$ satisfying the following inequalities:*

$$\ell \leq h(k),$$
$$h(a) \otimes h(b) \leq h(a \otimes b),$$

where ℓ is the unit of $\mathbb{W}$.

It is easy to see that $\otimes$ is monotone in both arguments. We denote the top and bottom element of a quantale by $\top$ and $\bot\!\!\!\bot$, respectively (notice that we use the symbol $\bot\!\!\!\bot$ to denote the bottom element of a quantale $\mathbb{V}$, and the symbol $\bot$ to denote the bottom element of a Σ-continuous monad). Moreover, we say that a quantale is commutative if its underlying monoid is such, and that it is non-trivial if $k \neq \bot\!\!\!\bot$. Finally, we observe that for any $a \in \mathsf{V}$, the map $a \otimes (-) : \mathsf{V} \to \mathsf{V}$ has a right adjoint $a \multimap (-) : \mathsf{V} \to \mathsf{V}$ which is uniquely determined by the law:

$$a \otimes b \leq c \iff b \leq a \multimap c.$$

From now on we tacitly assume quantales to be commutative and non-trivial.

Example 54. The following are examples of quantales.

1. The *boolean quantale* $\mathbb{2} = (2, \leq, \wedge, \mathsf{true})$ where $2 = \{\mathsf{true}, \mathsf{false}\}$ and $\mathsf{false} \leq \mathsf{true}$.

2. The *Lawvere quantale* $\mathbb{L} = ([0, \infty], \geq, +, 0)$ consists of the extended real half-line ordered by the "greater or equal" relation $\geq$ and extended[1] addition as monoid multiplication. Notice that in the Lawvere quantale the bottom element is ∞, the top element is 0, whereas infimum and supremum are defined as sup and inf, respectively. Notice also that $\multimap$ is truncated subtraction.

3. Replacing addition with maximum in the Lawvere quantale we obtain the *strong Lawvere quantale* $\mathbb{SL} = ([0, \infty], \geq, \max, 0)$, which has been used to study generalised ultrametric spaces (J. Rutten, 1996). In fact, in the strong Lawvere quantale tensor and meet coincide.

4. The *unit interval quantale* $\mathbb{I} = ([0, 1], \geq, +, 0)$ is obtained by restricting the Lawvere quantale to the unit interval $[0, 1]$ and addition to truncated addition.

5. A left continuous *triangular norm* (t-norm for short) is a binary operator $* : [0, 1] \times [0, 1] \to [0, 1]$ that induces a quantale structure over the complete lattice $([0, 1], \leq)$ in such a way that the quantale is commutative. Examples of t-norms are:

   (a) The *product t-norm*: $x *_p y \triangleq x \cdot y$.

   (b) The *Łukasiewicz t-norm*: $x *_l y \triangleq \max\{x + y - 1, 0\}$.

   (c) The *Gödel t-norm*: $x *_g y \triangleq \min\{x, y\}$.

6. The collection $\mathsf{Rel}(X, X)$ of binary relations on a set X forms a quantale with monoid structure given by relation composition and the identity relation.

7. The structure $(\tau, \subseteq, \cap, X)$, where τ is a collection of open sets on a set X is a quantale. Here the join is defined as $\bigcup$, from which we can define arbitrary meets (notice that these are, in general,

[1] We extend ordinary as follows: $x + \infty \triangleq \infty \triangleq \infty + x$.

different from arbitrary intersections). More generally, recall that a frame (Vickers, 1996) consists of a sup lattice $(V, \leq, \bigvee)$ satisfying the following distributivity laws:

$$b \wedge \bigvee_{i \in I} a_i = \bigvee_{i \in I} (b \wedge a_i)$$
$$(\bigvee_{i \in I} a_i) \wedge b = \bigvee_{i \in I} (a_i \wedge b).$$

We see that any frame is a quantale. More specifically, a frame is nothing but a quantale $\mathbb{V}$ with monoid structure given by the poset structure $(\mathsf{V}, \bigwedge, \top)$.

8. All complete boolean and Heyting algebras are (trivially) quantales.

9. The three-element chain $3 = \{\bot\!\!\bot, k, \top\}$ is a quantale, with $\bot\!\!\bot \leq k \leq \top$. We can use 3 to model scenarios like the following one. Suppose we have a family of contexts C acting as experiments. Given two programs e, e' we can test them against contexts in C. The output of the experiment performed by a context C is, as usual, an observation $obs(C[e])$. Each test can take only a given amount of time, after which the test fails (meaning that we were not able to prove neither the equivalence nor to discriminate between e and e'). In such a case we write $obs(C[e]) \uparrow$. We define the distance $\delta(e, e')$ between e and e' as the result of the whole experiment:

$$\delta(e, e') \triangleq \begin{cases} \bot\!\!\bot & \text{if } \exists C \in \mathsf{C}.\ obs(C[e]) \neq obs(C[e']) \\ \top & \text{if } \forall C \in \mathsf{C}.\ obs(C[e]) = obs(C[e']) \\ k & \text{if } \neg\exists C \in \mathsf{C}.\ obs(C[e]) \neq obs(C[e']) \wedge \exists C \in \mathsf{C}.(obs(C[e]) \uparrow \vee obs(C[e']) \uparrow). \end{cases}$$

Obviously there are cases where performing the same experiment twice gives different results (e.g. consider the program (**return** v) **or** Ω), or we can extend old experiments with new ones (for instance by enlarging C). It thus makes sense to introduce an operation $\otimes$ to collect and update the results of experiments. Obviously, we expect $\bot\!\!\bot \otimes x = \bot\!\!\bot$, as once we find an experiment discriminating between e and e', we can tell them apart. Moreover, if an experiment gives result k at first, but once repeated a second time gives a conclusive result $x \neq k$, then it is desirable to have $k \otimes x = x$. We can summarise these desiderata in the following multiplication table:

$\otimes$	$\bot\!\!\bot$	k	$\top$
$\bot\!\!\bot$	$\bot\!\!\bot$	$\bot\!\!\bot$	$\bot\!\!\bot$
k	$\bot\!\!\bot$	k	$\top$
$\top$	$\bot\!\!\bot$	$\top$	$\top$

It is a straightforward exercise to verify that $(3, \leq, \otimes, k)$ is a quantale.

$\boxtimes$

With the exception of $\mathrm{Rel}(X, X)$ and the three-element 3, all quantales mentioned in Examplex 54 have unit k coinciding with the top element $\top$. Quantales with such property are called *integral quantales*, and are particularly well-behaved. For instance, in an integral quantale $a \otimes b$ is a lower bound of a and b (and thus $a \otimes \bot\!\!\bot = \bot\!\!\bot$, for any $a \in \mathsf{V}$). For instance, by monotonicity of tensor products we have:

$$a \otimes b \leq a \otimes \top = a \otimes k = a.$$

Due to their nice properties, we will work with integral quantales only. In particular, from now on we tacitly assume all quantales to be integral.

Having defined quantales, we can now define quantale-valued relations. As we have already stressed, the notion of a $\mathbb{V}$-relation, for a quantale V, provides an abstraction of the notion of a relation that subsumes both the qualitative (i.e. boolean valued) and the quantitative (i.e. real valued) notion of a relation. Moreover, sets and $\mathbb{V}$-relations form a category which, thanks to the quantale structure of V, behaves essentially like Rel. That allows to develop an algebra of $\mathbb{V}$-relations on the same line of the usual algebra of relations.

Formally, for a quantale $\mathbb{V} = (\mathsf{V}, \leq, \otimes, k)$, a $\mathbb{V}$-relation $\alpha : X \nrightarrow Y$ between sets X and Y is a function $\alpha : X \times Y \to \mathsf{V}$. For any set X we can define the identity $\mathbb{V}$-relation $\mathsf{I}_X : X \nrightarrow X$ mapping diagonal elements (x, x) to k, and all other elements to $\bot\!\!\!\bot$. Moreover, for $\mathbb{V}$-relations $\alpha : X \nrightarrow Y$ and $\beta : Y \nrightarrow Z$, we can define the composition $\beta \cdot \alpha : X \nrightarrow Z$ by the so-called 'matrix multiplication formula':

$$(\beta \cdot \alpha)(x, z) \triangleq \bigvee_{y \in Y} \alpha(x, y) \otimes \beta(y, z).$$

Composition of $\mathbb{V}$-relations is associative, and I is the unit of composition. As a consequence, we have that sets and $\mathbb{V}$-relations form a category, called $\mathbb{V}$-Rel. $\mathbb{V}$-Rel is a monoidal category with unit given by the one-element set and tensor product given by cartesian product of sets with $\alpha \otimes \beta : X \times Y \nrightarrow X' \times Y'$, for $\alpha : X \nrightarrow X'$ and $\beta : Y \to Y'$, defined pointwise. Moreover, for all sets X, Y, the hom-set $\mathbb{V}$-$\mathsf{Rel}(X, Y)$ inherits a complete lattice structure from V according to the pointwise ordering. Actually, the whole quantale structure of V is inherited, in the sense that $\mathbb{V}$-Rel is a *quantaloid* (Hofmann et al., 2014). In particular, for all $\mathbb{V}$-relations $\alpha : X \nrightarrow Y$, $\beta_i : Y \nrightarrow Z$ $(i \in I)$, and $\gamma : Z \nrightarrow W$ we have the following distributivity laws:

$$\gamma \cdot (\bigvee_{i \in I} \beta_i) = \bigvee_{i \in I} (\gamma \cdot \beta_i),$$
$$(\bigvee_{i \in I} \beta_i) \cdot \alpha = \bigvee_{i \in I} (\beta_i \cdot \alpha).$$

There is a bijection $-^\circ : \mathbb{V}\text{-}\mathsf{Rel}(X, Y) \to \mathbb{V}\text{-}\mathsf{Rel}(Y, X)$ that maps each $\mathbb{V}$-relation α to its dual α° defined by $\alpha^\circ(y, x) \triangleq \alpha(x, y)$. It is straightforward to see that $-^\circ$ is monotone (i.e. $\alpha \leq \beta$ implies $\alpha^\circ \leq \beta^\circ$), idempotent (i.e. $(\alpha^\circ)^\circ = \alpha$), and preserves the identity $\mathbb{V}$-relation (i.e. $\mathsf{I}^\circ = \mathsf{I}$). Moreover, since $\mathbb{V}$ is commutative we also have the equality $(\beta \cdot \alpha)^\circ = \alpha^\circ \cdot \beta^\circ$.

Finally, we define the graph functor G from Set to $\mathbb{V}$-Rel acting as the identity on sets and mapping each function f to its graph (so that $G(f)(x, y)$ is equal to k if $y = f(x)$, and $\bot\!\!\!\bot$ otherwise). It is easy to see that since V is non-trivial G is faithful. In light of this observation we will use the notation $f : X \to Y$ in place of $G(f) : X \nrightarrow Y$ in $\mathbb{V}$-Rel.

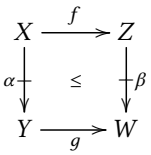
A direct application of the definition of composition gives the equality:

$$(g^\circ \cdot \alpha \cdot f)(x, w) = \alpha(f(x), g(w))$$

for $f : X \to Y$, $\alpha : Y \nrightarrow Z$, and $g : W \to Z$. Moreover, it is useful to keep in mind the following adjunction rules (Hofmann et al., 2014) (for α, β, γ $\mathbb{V}$-relations, and f, g functions with appropriate source and target):

$$g \cdot \alpha \leq \beta \iff \alpha \leq g^\circ \cdot \beta,$$
$$\beta \cdot f^\circ \leq \gamma \iff \beta \leq \gamma \cdot f.$$

The above inequalities turned out to be useful in making pointfree calculations with $\mathbb{V}$-relations. In particular, we can use *lax* commutative diagrams of the form



as diagrammatic representation for the inequality $g \cdot \alpha \leq \beta \cdot f$. By adjunction rules, the latter is equivalent to $\alpha \leq g^\circ \cdot \beta \cdot f$, which, pointwise, gives the following generalised non-expansiveness condition[2]: $\forall (x, y) \in X \times Y.\ \alpha(x, y) \leq \beta(f(x), g(y))$. Among $\mathbb{V}$-relations we are interested in those generalising equivalences and pseudometrics.

Definition 62. *A $\mathbb{V}$-relation $\alpha : X \nrightarrow X$ is* reflexive *if $1_X \leq \alpha$,* transitive *if $\alpha \cdot \alpha \leq \alpha$, and* symmetric *if $\alpha \leq \alpha^\circ$.*

When read pointwise, reflexivity, transitivity, and symmetry give the following inequalities:

$$\begin{aligned} k &\leq \alpha(x, x) \\ \alpha(x, y) \otimes \alpha(y, z) &\leq \alpha(x, z) \\ \alpha(x, y) &\leq \alpha(y, x), \end{aligned}$$

for all $x, y, z \in X$. We call a reflexive and transitive $\mathbb{V}$-relation a preorder $\mathbb{V}$-relation (V-*preorder*, for short), and a reflexive, symmetric, and transitive $\mathbb{V}$-relation an equivalence $\mathbb{V}$-relation (V-*equivalence*, for short).

Example 55. We have already seen that instantiating the transitivity formula on the boolean, Lawvere, and strong Lawvere quantale we recover the notion of (boolean) transitivity, triangle inequality, and strong triangle inequality, respectively. Proceeding in a similar fashion with reflexivity (resp. reflexivity and symmetry), we see that $\mathbb{V}$-preorders (resp. $\mathbb{V}$-equivalences) generalise preorder (resp. equivalence), generalised metric (resp. pseudometric), and generealised ultrametric (resp. ultra-pseudometric) spaces, respectively. These correspondences are summarised in Table 10.1, where $\mathcal{R} : X \nrightarrow X$ is a $\mathbb{2}$-relation, $\mu : X \nrightarrow X$ is a $\mathbb{L}$-relation, and $p : X \nrightarrow X$ is a $\mathbb{SL}$-relation ⊠

Boolean	Lawvere	Strong Lawvere
$\mathsf{true} \leq \mathcal{R}(x, x)$	$0 \geq \mu(x, x)$	$0 \geq p(x, x)$
$\mathcal{R}(x, y) \leq \mathcal{R}(y, x)$	$\mu(x, y) \geq \mu(y, x)$	$p(x, y) \geq p(y, x)$
$\mathcal{R}(x, y) \wedge \mathcal{R}(y, z) \leq \mathcal{R}(x, z)$	$\mu(x, y) + \mu(y, z) \geq \mu(x, z)$	$\max(p(x, y), p(y, z)) \geq p(x, z)$

Table 10.1: Correspondences reflexivity-symmetry-transitivity.

Remark 17 ($\mathbb{V}$-categories)**.** In his seminal paper (F. Lawvere, 1973) Lawvere introduced generalised metric spaces as $\mathbb{L}$-preorder spaces, i.e. pairs (X, α) consisting of a set X and an $\mathbb{L}$-preorder $\alpha : X \nrightarrow X$. Generalising from the Lawvere quantale to an arbitrary quantale $\mathbb{V}$ we obtain the so-called $\mathbb{V}$-categories (Hofmann et al., 2014). In fact, a $\mathbb{V}$-category (X, α) is nothing but a category enriched over V regarded as a bicomplete monoidal category. The notion of a $\mathbb{V}$-enriched functor instantiates to the notion of a non-expansive map between $\mathbb{V}$-categories, so that one can consider the category $\mathbb{V}$-Cat of $\mathbb{V}$-categories

[2] Taking $f = g$ generalised non-expansiveness expresses monotonicity of f in the boolean quantale, and non-expansiveness of f in the Lawvere quantale and its variants (recall that when we instantiate $\mathbb{V}$ as e.g. the Lawvere quantale we have to *reverse* inequalities).

and $\mathbb{V}$-functors. The category $\mathbb{V}$-Cat has a rich structure. In particular, it is a (symmetric) monoidal closed category. Given $\mathbb{V}$-categories $(X,\alpha),(Y,\beta)$, their exponential $(Y^X,[\alpha,\beta])$ is defined by

$$[\alpha,\beta](f,g) \triangleq \bigwedge_{x\in X} \beta(f(x),g(x)),$$

whereas their tensor product $(X\times Y,\alpha\otimes\beta)$ is defined pointwise. Instantiating $\mathbb{V}$-Cat with the boolean, Lawvere, and strong Lawvere quantale we obtain the category of preorders and monotone function, the category of generalised metric spaces and non-expansive maps, and the category of generalised ultrametric spaces and non-expansive maps, respectively. In particular, given monotone/non-expansive functions $f,g : X \to Y$, relations $\mathcal{R} : X \nrightarrow X$, $\mathcal{S} : Y \nrightarrow Y$, and generalised (ultra)metrics $\mu : X \nrightarrow X$, $\nu : Y \nrightarrow Y$, we see that we have:

$$\begin{aligned}[\mathcal{R},\mathcal{S}](f,g) &\iff \forall x.\ f(x)\ \mathcal{S}\ g(x)\\ [\mu,\nu](f,g) &= \sup_x \nu(f(x),g(x)).\end{aligned}$$

Finally, we observed that given $\mathbb{V}$-categories $(X,\alpha),(Y,\beta)$ and $\mathbb{V}$-enriched functors we can equivalently express $[\alpha,\beta](f,g)$ as

$$\bigwedge_{x,x'\in X} \alpha(x,x') \multimap \beta(f(x),g(x')).$$

In fact, since α is reflexive, for all $x\in X$ we have:

$$\begin{aligned}\alpha(x,x) \multimap \beta(f(x),g(x)) &= (\alpha(x,x) \multimap \beta(f(x),g(x))) \otimes k\\ &\le (\alpha(x,x) \multimap \beta(f(x),g(x))) \otimes \alpha(x,x)\\ &\le \beta(f(x),g(x)),\end{aligned}$$

from which we conclude $\bigwedge_{x,x'} \alpha(x,x') \multimap \beta(f(x),g(x')) \le \bigwedge_x \beta(f(x),g(x))$. Dually, since f is a $\mathbb{V}$-enriched functor and β is transitive, for all x,x' we have:

$$\begin{aligned}\alpha(x,x') \otimes \bigwedge_x \beta(f(x),g(x)) &\le \alpha(x,x') \otimes \beta(f(x'),g(x'))\\ &\le \beta(f(x),f(x')) \otimes \beta(f(x'),g(x'))\\ &\le \beta(f(x),g(x')).\end{aligned}$$

By adjunction, we can infer $\bigwedge_x \beta(f(x),g(x)) \le \alpha(x,x') \multimap \beta(f(x),g(x'))$, and thus $\bigwedge_x \beta(f(x),g(x)) \le \bigwedge_{x,x'} \alpha(x,x') \multimap \beta(f(x),g(x'))$.

Although here we do not work with $\mathbb{V}$-categories (we work in $\mathbb{V}$-Rel only), it is sometimes useful to think in terms of $\mathbb{V}$-categories for 'semantical intuitions'.

10.1.1 Operations and Change of Base Functors

Since effects in $\mathbb{V}$-Fuzz will be specified by means of a signature Σ of operation symbols, we need to specify how operations in Σ interact with $\mathbb{V}$-relations (e.g. how they modify distances), and thus how they interact with quantales. For simplicity, we will consider only finitary operation symbols. Although we conjecture our results to hold also for generalised operations, the latter are not very natural in $\mathbb{V}$-Fuzz, as they would require to work in an infinitary type system. Nonetheless, we will give some hints on how to extend our definitions to generalised operations too.

Definition 63. *Let Σ be a signature consisting of finitary operation symbols only. A Σ-quantale is a quantale $\mathbb{V}$ equipped with* monotone *operations $[\![\mathbf{op}]\!]_{\mathsf{V}} : \mathsf{V}^n \to \mathsf{V}$, for each n-ary operation $\mathbf{op} \in \Sigma$, satisfying the following inequalities:*

$$k \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(k, \ldots, k),$$
$$[\![\mathbf{op}]\!]_{\mathsf{V}}(a_1, \ldots, a_n) \otimes [\![\mathbf{op}]\!]_{\mathsf{V}}(b_1, \ldots, b_n) \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(a_1 \otimes b_1, \ldots, a_n \otimes b_n).$$

Example 56. Both in the Lawvere quantale and in the unit interval quantale we can interpret the fair probabilistic choice operation symbol **or** as: $x \,[\![\mathbf{or}]\!]_W\, y \triangleq \frac{1}{2} \cdot x + \frac{1}{2} \cdot y$, where W ranges over $\{[0, \infty], [0, 1]\}$. More generally, we can interpret the (unfair) p-probabilistic choice operation symbol $\mathbf{or}_p$, where p is a rational number in $[0, 1]$, as $x \,[\![\mathbf{or}_p]\!]_W\, y \triangleq p \cdot x + (1 - p) \cdot y$.

In general, for a quantale V we can always interpret $[\![\mathbf{op}]\!]_{\mathsf{V}}(a_1, \ldots, a_n)$ both as $a_1 \otimes \ldots \otimes a_n$ and as $a_1 \wedge \ldots \wedge a_n$. In order to have a clear intuition about the above interpretation of operation symbols, it is useful to look at the result of the evaluation of a program as a computation tree. Recall from Chapter 2 that each node in a computation tree is labelled by an operation symbol **op** which corresponds to the evaluation of a computation of the form $\mathbf{op}(e_1, \ldots, e_n)$. Each subtree of the node represents the result of the evaluation of one of the continuations $e_1, \ldots, e_n$. As a consequence, the interpretation of **op** as a tensor product corresponds to the view that all distances observed in each possible continuation should be accumulated, whereas the interpretation of **op** as binary meet states that the observed distance is obtained by taking the maximal distance observed in each possible continuation. ⊠

It is easy to see that we can extend Definition 63 to generalised operation symbols as follows. Given an operation $\mathbf{op} : P \rightsquigarrow I$, let $\bar{k} : I \to \mathsf{V}$ be the constant function mapping each $i \in I$ to k. Then we require $[\![\mathbf{op}]\!]_{\mathsf{V}}$ to satisfy the following inequalities:

$$k \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(p, \bar{k})$$
$$[\![\mathbf{op}]\!]_{\mathsf{V}}(p, \kappa) \otimes [\![\mathbf{op}]\!]_{\mathsf{V}}(p, \nu) \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(p, \kappa \otimes \nu),$$

where $\kappa \otimes \nu$ is defined pointwise. We also require $[\![\mathbf{op}]\!]_{\mathsf{V}}$ to be monotone in its second argument, where the order on V^I is defined pointwise.

Finally, we introduce the notion of a *change of base functor* (Hofmann et al., 2014; Kelly, 2005; F. Lawvere, 1973). As stressed in Chapter 9, we model program sensitivity as a function giving the 'law' describing how much distances between inputs are modified by the program. The notion of a change of base functor provides a mathematical abstraction to model the concept of program sensitivity with respect to an arbitrary quantale.

Definition 64. *A change of base functor, CBF for short, between quantales $\mathbb{V}, \mathbb{W}$ is a lax quantale morphism $h : \mathsf{V} \to \mathsf{W}$ (see Definition 61). If $\mathsf{V} = \mathsf{W}$ we speak of change of base* endofunctors *(CBEs, for short), and denote them by $s, r \ldots$. Clearly, every CBE s is also a CBF.*

The action $h \circ \alpha$ of a CBF $h : \mathsf{V} \to \mathsf{W}$ on a $\mathbb{V}$-relation $\alpha : X \nrightarrow Y$ is defined by $h \circ \alpha(x, y) \triangleq h(\alpha(x, y))$ (to improve readability we omit brackets). Notice that since $\mathbb{V}$ is integral, CBFs preserve the unit k.

Example 57. 1. We define (extended) multiplication by a constant $c \in [0, \infty]$ as the function $c \cdot (-) : [0, \infty] \to [0, \infty]$ defined as follows, where $x < \infty$:

$$0 \cdot \infty \triangleq 0$$
$$\infty \cdot 0 \triangleq 0$$
$$\infty \cdot x \triangleq \infty$$
$$x \cdot \infty \triangleq \infty.$$

Extended multiplication by a constant is a CBE on the Lawvere quantale. Moreover, it also acts as a CBE on the unit interval quantale, where multiplication is meant to be truncated.

Our definition of extended multiplication is different from the one given in (de Amorim et al., 2017), where extended multiplication is defined as follows (with $y \neq 0$):

$$x \cdot \infty \triangleq \infty$$
$$\infty \cdot 0 \triangleq 0$$
$$\infty \cdot y \triangleq \infty.$$

Despite making extended multiplication non-commutative, the above definition does not fit with our intuition about program sensitivity. In fact, according to our informal reading, the equality $0 \cdot \infty = 0$ expresses the principle that if we test observationally distinguishable objects zero times (i.e. we do not test them at all), then we are not able to tell them apart. Dually, the equality $\infty \cdot 0 = 0$ states that observationally indistinguishable objects cannot be told apart, no matter how many times are tested.

We should also remark that although our definition of extended multiplication agrees with our intuition of program sensitivity, it has the drawback of being non-continuous. For instance, consider the set $X = \{\varepsilon \mid \varepsilon > 0\}$. Continuity requires $\infty \cdot \inf X = \inf(\infty \cdot X)$, which is obviously not the case since $\infty \cdot \inf X = \infty \cdot 0 = 0$ and $\inf\{\infty \cdot \varepsilon \mid \varepsilon > 0\} = \inf \infty = \infty$.

2. Another interesting example of a CBE both on the Lawvere and the unit interval quantale is provided by polynomials P such that $P(0) = 0$. Such polynomials can be used to express sensitivity of context having only polynomial testing power (meaning that contexts are required to be able to discriminate programs efficiently).

3. More generally, given a quantale $\mathbb{V}$ we can define CBEs $n, \infty : \mathsf{V} \to \mathsf{V}$, for $n < \omega$ as follows:

$$0(a) \triangleq k$$
$$(n+1)(a) \triangleq a \otimes n(a)$$
$$\infty(a) \triangleq \mathbb{1}.$$

Notice that 1 acts as the identity function, and that on the Lawvere and unit interval quantale we have $n(c) = n \cdot c$ and $\infty(c) = \infty$.

4. An important example of CBFs is given by the map $\psi : 2 \to \mathsf{V}$ and its right adjoint $\varphi : \mathsf{V} \to 2$ defined by:

$$\varphi(k) \triangleq \mathsf{true} \qquad \psi(\mathsf{true}) \triangleq$$
$$\varphi(a) \triangleq \mathsf{false} \qquad \psi(\mathsf{false}) \triangleq \mathbb{1}.$$

Intuitively, the map φ is used to collapse distances into relations, whereas ψ is used to translate a relation into a (trivial) distance.

⊠

Finally, we observe that the action of CBFs on a $\mathbb{V}$-relation obeys the following laws:

$$(h \cdot h')(\alpha) = h \circ (h' \circ \alpha),$$
$$(h \circ \alpha) \cdot (h \circ \beta) \leq h \circ (\alpha \cdot \beta).$$

Remark 18. We saw that $\mathbb{V}$-categories generalise both generalised metric spaces and preorders, and that $\mathbb{V}$-functors generalise both monotone and non-expansive functions. However, when dealing with metric-like spaces, besides non-expansive functions a prominent role is played by *Lipshitz-continuous* functions (Searcóid, 2006).

Given metric-like spaces (X, μ) and (Y, ν), a function $f : X \to Y$ is said to be *c-continuous*, for $c \in \mathbb{R}_{\geq 0}$, if the inequality

$$c \cdot \mu(x, x') \geq \nu(f(x), f(x'))$$

holds, for all $x, x' \in X$. Example 57 shows that multiplication by a (possibly extended) non-negative real number c is a change of base endofunctor on the Lawvere quantale, meaning that using CBEs we can generalise the notion of Lipshitz-continuity to $\mathbb{V}$-categories. In fact, easy calculations show that for any $\mathbb{V}$-category (X, α) and any CBE s on $\mathbb{V}$, $(X, s \circ \alpha)$ is a $\mathbb{V}$-category. Moreover, since CBEs are monotone, such an operation is functorial (with morphism left unchanged). In particular, we can define s-continuous functions from (X, α) to (Y, β) as $\mathbb{V}$-functors from $(X, s \circ \alpha)$ to (Y, β). That is, we say that a function $f : X \to Y$ is s-continuous if

$$s \circ \alpha(x, x') \leq \beta(f(x), f(x'))$$

holds, for all $x, x' \in X$.

We conclude this section with the following useful result on the algebra of CBEs.

Lemma 36. *Let V be a Σ-quantale. CBEs are closed under the following operations (where* **op** *is an n-ary operation symbol in Σ):*

$$\begin{aligned}
(s \otimes r)(a) &\triangleq s(a) \otimes r(a) \\
(r \cdot s)(a) &\triangleq r(s(a)) \\
(s \wedge r)(a) &= s(a) \wedge s(b) \\
[\![\mathbf{op}]\!]_{\mathsf{V}}(s_1, \ldots, s_n)(a) &\triangleq [\![\mathbf{op}]\!]_{\mathsf{V}}(s_1(a), \ldots, s_n(a)).
\end{aligned}$$

The proof of Lemma 36 is straightforward. Moreover, we can extend its content to generalised operations as follows. Let $\mathbf{op} : P \rightsquigarrow I$ be a generalised operation, and let κ be a function mapping each $i \in I$ to a CBE $\kappa(i)$. We define the CBE $[\![\mathbf{op}]\!]_{\mathsf{V}}(p, \kappa)$ by:

$$[\![\mathbf{op}]\!]_{\mathsf{V}}(p, \kappa)(a) \triangleq [\![\mathbf{op}]\!]_{\mathsf{V}}(p, i \mapsto \kappa_i(a)).$$

It is easy to see that $[\![\mathbf{op}]\!]_{\mathsf{V}}(p, \kappa)$ is indeed a CBE. For instance, suppose $a \leq b$. Since κ_i is a CBE, we have $\kappa_i(a) \leq \kappa_i(b)$, for any $i \in I$. It follows that $i \mapsto \kappa_i(a) \leq i \mapsto \kappa_i(b)$, and thus $[\![\mathbf{op}]\!]_{\mathsf{V}}(p, i \mapsto \kappa_i(a)) \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(p, i \mapsto \kappa_i(b))$, since $[\![\mathbf{op}]\!]_{\mathsf{V}}$ is monotone.

10.2 The $\mathbb{V}$-Fuzz Language

As already observed in Chapter 9, when dealing with behavioural $\mathbb{V}$-relations a crucial parameter in distance trivialisation is program sensitivity. To deal with such parameter we now introduce $\mathbb{V}$-Fuzz, a higher-order fine-grain λ-calculus extending Fuzz (Reed & Pierce, 2010) with algebraic operations. As Fuzz, $\mathbb{V}$-Fuzz is characterised by a powerful linear type system with sum and recursive types inspired by *bounded linear logic* (Girard et al., 1992) giving syntactic information on program sensitivity.

The syntax of $\mathbb{V}$-Fuzz is parametrised by a signature Σ of operation symbols, a Σ-quantale $\mathbb{V}$, and a family Π of CBEs. From now on we assume Σ, $\mathbb{V}$, and Π to be fixed. Moreover, we assume Π to contain

$$
\begin{array}{lll}
\text{Types } \sigma, \tau \ ::= \ t & \text{Values } v, w \ ::= \ x & \text{Computations } e, f \ ::= \ \mathbf{return}\ v \\
\quad \mid \sum_{i \in I} \sigma_i & \quad \mid \lambda x.e & \quad \mid v w \\
\quad \mid \sigma \multimap \tau & \quad \mid \langle \hat{\imath}, v \rangle & \quad \mid \mathbf{case}\ v\ \mathbf{of}\ \{\langle i, x \rangle \to e_i\} \\
\quad \mid \mu t.\sigma & \quad \mid \mathbf{fold}\ v & \quad \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \\
\quad \mid !_s \sigma & \quad \mid\ !v & \quad \mid \mathbf{case}\ v\ \mathbf{of}\ \{!x \to e\} \\
 & & \quad \mid \mathbf{case}\ v\ \mathbf{of}\ \{\mathbf{fold}\ x \to e\} \\
 & & \quad \mid \mathbf{op}(e, \ldots, e).
\end{array}
$$

Figure 10.1: Types, values, and computations of $\mathbb{V}$-Fuzz.

at least CBEs n, ∞ in Example 57 and to be closed under operations in Lemma 36. Types, (raw) values, and (raw) computations of $\mathbb{V}$-Fuzz are defined in Figure 10.1, where t denotes a type variable, I is a *finite* set (whose elements are denoted by $\hat{\imath}, \hat{\jmath}, \ldots$), and s belongs to Π.

Free and bound variables in computations and values are defined as usual, and we adopt the same syntactical conventions defined in Chapter 3. We extend such conventions to types. In particular, we denote by $\sigma[\tau/t]$ the result of capture-avoiding substitution of type τ for the type variable t in σ. We also write $\mathbf{0}$ for the empty sum type, $\mathbf{1}$ for $\mathbf{0} \multimap \mathbf{0}$, and nat for $\mu t.\mathbf{1} + t$.

In order to define the class of well-defined $\mathbb{V}$-Fuzz terms we equip $\mathbb{V}$-Fuzz with a suitable type system. Intuitively, the latter is based on judgments of the form $x_1 :_{s_1} \sigma_1, \ldots, x_n :_{s_n} \sigma_n \vdash e : \sigma$, where $s_1, \ldots, s_n$ are CBEs. The informal meaning of such judgment is that on input x_i ($i \leq n$), the computation e has sensitivity s_i. That is, e amplifies the (behavioural) distance between two input values v_i, v_i' of *at most* a factor s_i. Symbolically, we have:

$$s_i \circ \alpha(v_i, v_i') \leq \alpha(e[x_i := v_i'], e[x_i := v_i']).$$

In order to define typing judgments formally, we need the notion of an *environment*. An environment Γ is a finite sequence $x_1 :_{s_1} \sigma_1, \ldots, x_n :_{s_n} \sigma_n$ of distinct variables with associated closed types and CBEs (we denote the empty environment by $\emptyset$ and oftentimes writes $\vdash e : \sigma$ in place of $\emptyset \vdash e : \sigma$). We can lift operations on CBEs in Lemma 36 to environments as follows, where $\mathbf{op}$ in an m-ary operation symbol in Σ:

$$
\begin{aligned}
r \cdot \Gamma &= x_1 :_{r \cdot s_1} \sigma_1, \ldots, x_n :_{r \cdot s_n} \sigma_n, \\
\Gamma \otimes \Delta &= x_1 :_{s_1 \otimes r_1} \sigma_1, \ldots, x_n :_{s_n \otimes r_n} \sigma_n, \\
[\![\mathbf{op}]\!]_{\mathbb{V}}(\Gamma^1, \ldots, \Gamma^m) &= x_1 :_{[\![\mathbf{op}]\!]_{\mathbb{V}}(s_1^1, \ldots, s_1^m)} \sigma_1, \ldots, x_n :_{[\![\mathbf{op}]\!]_{\mathbb{V}}(s_n^1, \ldots, s_n^m)} \sigma_n,
\end{aligned}
$$

for $\Gamma = x_1 :_{s_1} \sigma_1, \ldots, x_n :_{s_n} \sigma_n$, $\Delta = x_1 :_{r_1} \sigma_1, \ldots, x_n :_{r_n} \sigma_n$, and $\Gamma^i = x_1 :_{s_1^i} \sigma_1, \ldots, x_n :_{s_n^i} \sigma_n$. Notice that the above operations are defined for environments having the same structure (i.e. differing only on CBEs). This is not a real restriction, since we can always add the missing variables $y :_{\bar{k}} \sigma$, where $\bar{k}$ is the constant function returning the unit of the quantale.

Having at our disposal operations on environments we can define a type system for $\mathbb{V}$-Fuzz. Such a type system is based on two kinds of judgment (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 computations. The system is defined in Figure 10.2.

As for Λ_Σ, we denote by $\mathcal{V}_\circ^\sigma$ and $\Lambda_\circ^\sigma$ for the set of closed values and terms of type σ, respectively. Accordingly, we will also use the notation $\Lambda_{\Gamma \vdash^{\Lambda} \sigma}$ for the set $\{e \in \Lambda \mid \Gamma \vdash^{\Lambda} e : \sigma\}$ (and similarity for values).

$$\frac{s \leq 1}{\Gamma, x :_s \sigma \vdash^{\mathsf{v}} x : \sigma}$$

$$\frac{\Gamma, x :_1 \sigma \vdash^{\Lambda} e : \tau}{\Gamma \vdash^{\mathsf{v}} \lambda x.e : \sigma \multimap \tau} \quad \frac{\Gamma \vdash^{\mathsf{v}} v : \sigma \multimap \tau \quad \Delta \vdash^{\mathsf{v}} w : \sigma}{\Gamma \otimes \Delta \vdash^{\Lambda} vw : \tau}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} v : \sigma_i}{\Gamma \vdash^{\mathsf{v}} \langle i, v \rangle : \sum_{i \in I} \sigma_i} \quad \frac{\Gamma \vdash^{\mathsf{v}} v : \sum_{i \in I} \sigma_i \quad \Delta, x :_s \sigma_i \vdash^{\Lambda} e_i : \tau \quad (\forall i \in I)}{s \cdot \Gamma \otimes \Delta \vdash^{\Lambda} \mathbf{case}\ v\ \mathbf{of}\ \{\langle i, x \rangle \to e_i\} : \tau}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} v : \sigma}{\Gamma \vdash^{\Lambda} \mathbf{return}\ v : \sigma} \quad \frac{\Gamma \vdash^{\Lambda} e : \sigma \quad \Delta, x :_s \sigma \vdash^{\Lambda} f : \tau}{(s \wedge 1) \cdot \Gamma \otimes \Delta \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f : \tau}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} v : \sigma}{s \cdot \Gamma \vdash^{\mathsf{v}} !v : !_s \sigma} \quad \frac{\Gamma \vdash^{\mathsf{v}} v : !_r \sigma \quad \Delta, x :_{s \cdot r} \sigma \vdash^{\Lambda} e : \tau}{s \cdot \Gamma \otimes \Delta \vdash^{\Lambda} \mathbf{case}\ v\ \mathbf{of}\ \{!x \to e\} : \tau}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} v : \sigma[\mu t.\sigma/t]}{\Gamma \vdash^{\mathsf{v}} \mathbf{fold}\ v : \mu t.\sigma} \quad \frac{\Gamma \vdash^{\mathsf{v}} v : \mu t.\sigma \quad \Delta, x :_s \sigma[\mu t.\sigma/t] \vdash^{\Lambda} e : \tau}{s \cdot \Gamma \otimes \Delta \vdash^{\Lambda} \mathbf{case}\ v\ \mathbf{of}\ \{\mathbf{fold}\ x \to e\} : \tau}$$

$$\frac{\Gamma_1 \vdash^{\Lambda} e_1 : \sigma \quad \cdots \quad \Gamma_n \vdash^{\Lambda} e_n : \sigma}{[\![\mathbf{op}]\!]_{\mathsf{V}}(\Gamma_1, \ldots, \Gamma_n) \vdash^{\Lambda} \mathbf{op}(e_1, \ldots, e_n) : \sigma}$$

Figure 10.2: Typing rules for $\mathbb{V}$-Fuzz.

Before giving examples of concrete instances of $\mathbb{V}$-Fuzz, let us comment some of the typing rules in Figure 10.2. Most of these rules are similar to those of Fuzz (e.g. in the variable rule we require $s \leq 1$, meaning that the open value x can access x at least once) with the exception of the rule for sequencing and the rule for operation symbols (which is just not present in Fuzz, the latter not having algebraic operations). Concerning the former, let us consider the following instance of the sequencing rule on the unit interval quantale:

$$\frac{x :_1 \sigma \vdash^{\Lambda} e : \sigma \quad y :_0 \sigma \vdash^{\Lambda} f : \tau}{x :_{\max(0,1)\cdot 1} \sigma \vdash^{\Lambda} \mathbf{let}\ y = e\ \mathbf{in}\ f : \tau}$$

where f is a closed term of type τ, so that we can assume it to have sensitivity 0 on all variables. According to our informal intuition, e has sensitivity 1 on input x, meaning that (i) e can possibly detect (behavioural) differences between input values v, w, and (ii) e cannot amplify their behavioural distance of a factor bigger than 1. Formally, point (ii) states that we have the inequality $\alpha(v, w) \geq \alpha(e[x := v], e[x := w])$, where α denotes a suitable behavioural $\mathbb{I}$-relation. On the contrary, f is a closed term and thus has sensitivity 0 on any input, meaning that it cannot detect any observable difference between input values. In particular, for all values v, w we have $\alpha(f[y := v], f[y := w]) = \alpha(f, f) = 0$ (provided that α is reflexive). Replacing $\max(0, 1)$ with 0 in the above rule (i.e. $s \wedge 1$ with s in the general case) would allow to infer the judgment $x :_0 \sigma \vdash^{\Lambda} \mathbf{let}\ y = e\ \mathbf{in}\ f : \tau$, and thus to conclude $\alpha(\mathbf{let}\ y = e[x := v]\ \mathbf{in}\ f, \mathbf{let}\ y = e[x := w]\ \mathbf{in}\ f) = 0$. The latter equality is unsound as evaluating $\mathbf{let}\ y = e[x := v]\ \mathbf{in}\ f$ (resp. $\mathbf{let}\ y = e[x := w]\ \mathbf{in}\ f$) requires to *first* evaluate $e[x := v]$ (resp. $e[x := w]$) thus making observable differences between v and w detectable (see also Section 12.1 for a formal explanation).

10.2.1 Relevant Examples

Example 58. Instantiating $\mathbb{V}$-Fuzz with the empty signature, the Lawvere quantale, and CBEs $\Pi = \{c \cdot (-) \mid c \in [0, \infty]\}$ we obtain the original Fuzz of (Reed & Pierce, 2010). Actually, to recover full Fuzz we should add a basic type for real numbers, as well as primitive operations on its inhabitant. We can also add nondeterminism considering the signature $\Sigma_{\mathbb{FM}}$. ⊠

Example 59. A more interesting calculus is obtained by instantiating $\mathbb{V}$-Fuzz with the signature $\Sigma_{\mathbb{DM}}$, the unit interval quantale, and CBEs $\Pi = \{c \cdot (-) \mid c \in [0, \infty]\}$] (as usual we are actually referring to truncated multiplication). We refer to the calculus those obtained as P-Fuzz. The interpretation of the operation symbol **or** on $[0, 1]$ is defined as in Example 56. In particular, we have the following typing rule:

$$\frac{\Gamma \vdash^{\Lambda} e : \sigma \quad \Delta \vdash^{\Lambda} f : \sigma}{\frac{1}{2} \cdot \Gamma + \frac{1}{2} \cdot \Delta \vdash^{\Lambda} e\ \mathbf{or}\ f : \sigma}$$

As a consequence, we see that we have

$$\frac{\dfrac{x :_1 \sigma \vdash^{\mathsf{V}} x : \sigma}{x :_1 \sigma \vdash^{\Lambda} \mathbf{return}\ x : \sigma} \quad \dfrac{x :_0 \sigma \vdash^{\mathsf{V}} v : \sigma}{x :_0 \sigma \vdash^{\Lambda} \mathbf{return}\ v : \sigma}}{x :_{\frac{1}{2}} \sigma \vdash^{\Lambda} (\mathbf{return}\ x)\ \mathbf{or}\ (\mathbf{return}\ v)}$$

for any value $v \in \mathcal{V}_{\circ}^{\sigma}$. This witnesses that the notion of program sensitivity *cannot* be identified with the number of times programs access their inputs, as already (informally) remarked in Chapter 9. ⊠

Example 60. Let $\mathbb{V}$ be a frame, i.e. a quantale such that $\wedge$ and $\otimes$ coincide. Taking the empty signature, we see that $\mathbb{V}$-Fuzz gives a linear-like λ-calculus as follows. We take $\Pi \triangleq \{0, 1, \infty\}$. Since V is a frame, obviously Π is closed under operations of Lemma 36. The type $!_0$ has as inhabitants those values that cannot be used, whereas the types $!_1$ and $!_\infty$ have as inhabitants those values that can be use linearly and *as libitum*, respectively. ⊠

Example 61. Another interesting example is obtained by instantiating $\mathbb{V}$-Fuzz with the signature $\Sigma_{\mathbb{O}^\infty}$ and the *strong* Lawvere quantale. The latter is a frame, and thus we obtain a linear-like λ-calculus. We interpret the operation symbol $\mathbf{print}_c$ as the identity function on $[0,\infty]$. We will come back on this example later. ⊠

The typing system of Figure 10.2 makes $\mathbb{V}$-Fuzz a refinement of the linear λ-calculus (Maraist et al., 1999). For instance, for any type σ we have the term $I \triangleq \mathbf{return}\ (\lambda x.\mathbf{return}\ x)$ of type $\sigma \multimap \sigma$. Moreover, as the above examples suggest, $\mathbb{V}$-Fuzz is a Turing complete language. In fact, we can encode the linear λ-calculus in $\mathbb{V}$-Fuzz by translating the type $!\sigma$ into $!_\infty\sigma$. For instance, for any type σ we have the purely divergent program $\Omega \triangleq \omega!(\mathbf{fold}\ \omega)$ of type σ, where $\omega \in \Lambda_{!_\infty(\mu t.!_\infty t \multimap \sigma)\multimap\sigma}$ is defined by:

$$\omega \triangleq \lambda x.\mathbf{case}\ x\ \mathbf{of}\ \{!y \to \mathbf{case}\ y\ \mathbf{of}\ \{\mathbf{fold}\ z \to z!(\mathbf{fold}\ z)\}\}.$$

Finally, we remark that the syntactic distinction between terms and values gives the following (obvious) result.

Lemma 37. *The following equalities hold:*

$$\begin{aligned}
\mathcal{V}_\circ^{\sigma\multimap\tau} &= \{\lambda x.e \mid x :_1 \sigma \vdash^\Lambda e : \tau\}\\
\mathcal{V}_\circ^{\sum_{i\in I}\sigma_i} &= \bigcup_{i\in I}\{\langle i, v\rangle \mid v \in \mathcal{V}_\circ^{\sigma_i}\}\\
\mathcal{V}_\circ^{!_s\sigma} &= \{!v \mid v \in \mathcal{V}_\circ^{\sigma}\}.
\end{aligned}$$

Remark 19. Before defining the operational semantics of $\mathbb{V}$-Fuzz, we briefly expand on the possibility of considering generalised operation symbols. First, given a generalised operation symbol $\mathbf{op} : P \rightsquigarrow I$ with interpretation $[\![\mathbf{op}]\!]_\mathsf{V}$ on a quantale V (extended to CBEs according to Lemma 36), we extend $[\![\mathbf{op}]\!]_\mathsf{V}$ to environments as follows. Given a map γ associating to each $i \in I$ an environment $\Gamma^i = x_1 :_{s_1^i} \sigma_1, \ldots, x_n :_{s_n^i} \sigma_n$, we define

$$[\![\mathbf{op}]\!]_\mathsf{V}(p,\gamma) \triangleq x_1 :_{[\![\mathbf{op}]\!]_\mathsf{V}(p, i\mapsto s_1^i)} \sigma_1, \ldots, x_n :_{[\![\mathbf{op}]\!]_\mathsf{V}(p, i\mapsto s_n^i)} \sigma_n.$$

We then extend the typing system of Figure 10.2 with the infinitary rule:

$$\frac{\forall i \in I.\ \Gamma_i, x :_s \tau \vdash^\Lambda e_i : \sigma}{[\![\mathbf{op}]\!]_\mathsf{V}(p, i \mapsto s \cdot \Gamma_i) \vdash^\Lambda \mathbf{op}(p, i \mapsto e_i) : \sigma}$$

We can now define an operational semantics for $\mathbb{V}$-Fuzz.

10.2.2 Operational Semantics

We give $\mathbb{V}$-Fuzz monadic operational semantics in the style of Chapter 3. Let Σ be an algebraic signature for a Σ-continuous monad $\mathbb{T} = \langle T, \eta, -^\dagger\rangle$. Operational semantics is defined by means of an evaluation function $[\![-]\!]^\sigma$ indexed over closed types, associating to any computation in $\Lambda_\circ^\sigma$ a monadic value in $T\mathcal{V}_\circ^\sigma$. The evaluation function $[\![-]\!]^\sigma$ is itself defined by means of the $\mathbb{N}$-indexed family of functions $[\![-]\!]_n^\sigma$.

Definition 65. *The $\mathbb{N}$-indexed family of functions $[\![-]\!]^\sigma_n : \Lambda^\sigma_\circ \to T\mathcal{V}^\sigma_\circ$ is inductively defined as follows:*

$$\begin{aligned}
[\![e]\!]^\sigma_0 &\triangleq \bot_{\mathcal{V}_\sigma} \\
[\![\mathbf{return}\ v]\!]^\sigma_{n+1} &\triangleq \eta_{\mathcal{V}_\sigma}(v) \\
[\![(\lambda x.e)v]\!]^\sigma_{n+1} &\triangleq [\![e[x := v]]\!]^\sigma_n \\
[\![\mathbf{case}\ \langle \hat{\imath}, v\rangle\ \mathbf{of}\ \{\langle i, x\rangle \to e_i\}]\!]^\sigma_{n+1} &\triangleq [\![e_i[x := v]]\!]^\sigma_n \\
[\![\mathbf{case}\ (\mathbf{fold}\ v)\ \mathbf{of}\ \{\mathbf{fold}\ x \to e\}]\!]^\sigma_{n+1} &\triangleq [\![e[x := v]]\!]^\sigma_n \\
[\![\mathbf{case}\ !v\ \mathbf{of}\ \{!x \to e\}]\!]^\sigma_{n+1} &\triangleq [\![e[x := v]]\!]^\sigma_n \\
[\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!]^\sigma_{n+1} &\triangleq ([\![f[x := -]]\!]^{\tau,\sigma}_n)^\dagger [\![e]\!]^\tau_n \\
[\![\mathbf{op}(e_1, \dots, e_j)]\!]^\sigma_{n+1} &\triangleq [\![\mathbf{op}]\!]_{\mathcal{V}_\sigma}([\![e_1]\!]^\sigma_n, \dots, [\![e_j]\!]^\sigma_n).
\end{aligned}$$

The definition of $[\![-]\!]^\sigma_n$ is rather standard and closely resembles Definition 15. Moreover, it is straightforward to observe that indeed $[\![-]\!]^\sigma_n$ a function from $\Lambda^\sigma_\circ$ to $T\mathcal{V}^\sigma_\circ$.

Let us expand on the definition of $[\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!]^\sigma_{n+1}$. Since $\mathbf{let}\ x = e\ \mathbf{in}\ f \in \Lambda^\sigma_\circ$, the judgment $\vdash^\Lambda \mathbf{let}\ x = e\ \mathbf{in}\ f : \sigma$ must be the conclusion of a derivation of the form:

$$\frac{\vdash^\Lambda e : \tau \quad x :_s \tau \vdash^\Lambda f : \sigma}{\vdash^\Lambda \mathbf{let}\ x = e\ \mathbf{in}\ f : \sigma}$$

As a consequence, for any $v \in \mathcal{V}^\tau_\circ$, we have $[\![f[x := v]]\!]^\sigma_n \in T\mathcal{V}^\sigma_\circ$. This induces a map $[\![f[x := -]]\!]^{\tau,\sigma}_n$ from $\mathcal{V}^\tau_\circ$ to $T\mathcal{V}^\sigma_\circ$ whose Kleisli extension can be applied to $[\![e]\!]^\tau_n \in T\mathcal{V}^\tau_\circ$.

Finally, it is easy to see that $([\![e]\!]_n)_n$ forms an ω-chain in $T\mathcal{V}^\sigma_\circ$. As for Λ_Σ, we can thus define the evaluation map $[\![-]\!]^\sigma : \Lambda^\sigma_\circ \to T\mathcal{V}^\sigma_\circ$ as $[\![e]\!]^\sigma \triangleq \bigsqcup_n [\![e]\!]^\sigma_n$. In order to improve readability we oftentimes omit type superscripts in $[\![e]\!]^\sigma$. We also notice that because $\mathbb{T}$ is Σ-continuous, $[\![-]\!]^\sigma$ is itself continuous.

Proposition 24. *The following identities hold:*

$$\begin{aligned}
[\![\mathbf{return}\ v]\!] &= \eta(v) \\
[\![(\lambda x.e)v]\!] &= [\![e[x := v]]\!] \\
[\![\mathbf{case}\ \langle \hat{\imath}, v\rangle\ \mathbf{of}\ \{\langle i, x\rangle \to e_i\}]\!] &= [\![e_i[x := v]]\!] \\
[\![\mathbf{case}\ (\mathbf{fold}\ v)\ \mathbf{of}\ \{\mathbf{fold}\ x \to e\}]\!] &= [\![e[x := v]]\!] \\
[\![\mathbf{case}\ !v\ \mathbf{of}\ \{!x \to e\}]\!] &= [\![e[x := v]]\!] \\
[\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!] &= [\![f[x := -]]\!]^\dagger([\![e]\!]) \\
[\![\mathbf{op}(e_1, \dots, e_j)]\!] &= [\![\mathbf{op}]\!]_{\mathcal{V}_\sigma}([\![e_1]\!], \dots, [\![e_j]\!]).
\end{aligned}$$

Having defined the syntax and semantics of $\mathbb{V}$-Fuzz, it is time to look at program distances for it. In order to do so, we first have to generalise the theory of relators of Chapter 4 to $\mathbb{V}$-relations. We refer to relators for $\mathbb{V}$-relations as $\mathbb{V}$-*relators*.

Chapter 11

Barr Meets Lawvere

> It is indifferent to me where I am to begin, for there shall I return again
>
> Parmenides

In this chapter we define the notion of a $\mathbb{V}$-relator (Hofmann et al., 2014) for a quantale $\mathbb{V}$. The latter is somehow the 'quantitative' generalisation of the concept of a relator. Analogously to ordinary relators, $\mathbb{V}$-relators for a set endofunctor T are abstractions meant to capture the possible ways a $\mathbb{V}$-relation on a set X can be turned into a $\mathbb{V}$-relation on TX, and thus provide ways to lift a behavioural distance between programs to a (behavioural) distance between monadic values. On a formal level, we say that a $\mathbb{V}$-relator extends T from Set to $\mathbb{V}$-Rel, laxly.

$\mathbb{V}$-valued relators have been introduced in the context of monoidal topology (Hofmann et al., 2014) (where are usually called *lax extension* (Hoffman, 2015)), in order to provide a unifying account of Lawvere's theory of generalised metric spaces (F. Lawvere, 1973) and Barr's categorical analysis of topological spaces (Barr, 1970). Using the notion of $\mathbb{V}$-relator (which leads to the notion of lax $\mathbb{V}$-algebra (Hofmann et al., 2014)), it is possible to unify the notion of a (generalised) metric, preorder, topological, and approach space, in purely categorical terms.

According to (Hofmann et al., 2014), the observation that a similar unification should have been possible dates back to Lawvere. The reader can also consult (Hofmann & Reis, 2018) for an historical introduction to the subject.

11.1 Quantale-valued Relators

In this section we introduce the notion of a $\mathbb{V}$-relator for a (strong) monad, which will be a central tool for our analysis of effectful program distance. After having introduced some basic notions and definitions, we focus on some specific examples of $\mathbb{V}$-relators, giving special attention to the so-called Wasserstein-Kantorovich lifting (Villani, 2008). As usual, in the rest of this chapter we assume all functors (and thus monads) to be on Set, unless explicitly stated.

Definition 66. *For a fucunctor T a* $\mathbb{V}$-relator *for T is a mapping $(\alpha : X \nrightarrow Y) \mapsto (\Gamma\alpha : TX \nrightarrow TY)$ satisfying conditions* ($\mathbb{V}$-rel 1)-($\mathbb{V}$-rel 4). *We say that Γ is conversive if it additionally satisfies condition*

(𝕍-rel 5).

$$\mathsf{I}_{TX} \leq \Gamma(1_X) \tag{𝕍-rel 1}$$
$$\Gamma\beta \cdot \Gamma\alpha \leq \Gamma(\beta \cdot \alpha) \tag{𝕍-rel 2}$$
$$Tf \leq \Gamma f, \quad (Tf)^\circ \leq \Gamma f^\circ \tag{𝕍-rel 3}$$
$$\alpha \leq \beta \implies \Gamma\alpha \leq \Gamma\beta \tag{𝕍-rel 4}$$
$$\Gamma(\alpha^\circ) = (\Gamma\alpha)^\circ. \tag{𝕍-rel 5}$$

It is immediate to see that when instantiated with $\mathbb{V} = \mathbb{2}$, the above definition gives the usual notion of relator as defined in Chapter 4. We also observe that any $\mathbb{V}$-relator Γ for T induces an endomap T_Γ on $\mathbb{V}$-Rel that acts as T on sets and as Γ as $\mathbb{V}$-relation. It is easy to check that conditions in Definition 66 makes T_Γ a *lax endofunctor*.

Before giving examples of $\mathbb{V}$-relators it is useful to observe that the collection of $\mathbb{V}$-relators is closed under specific operations.

Proposition 25. *Let T, U be functors, UT be their composition, Γ, Δ be relators for T and U, respectively, and $\{\Gamma_i\}_{i\in I}$ be a family of relators for T. Then:*

1. *The map $\Delta \cdot \Gamma$ defined by $(\Delta \cdot \Gamma)\alpha \triangleq \Delta\Gamma\alpha$ is a $\mathbb{V}$-relator for UT.*
2. *The map $\bigwedge_{i\in I} \Gamma_i$ defined by $(\bigwedge_{i\in I} \Gamma_i)\alpha \triangleq \bigwedge_{i\in I} \Gamma_i\alpha$ is a $\mathbb{V}$-relator for T.*
3. *The map Γ° defined by $\Gamma^\circ\alpha \triangleq (\Gamma\alpha^\circ)^\circ$ is a $\mathbb{V}$-relator for T.*
4. *The map $\Gamma \wedge \Gamma^\circ$ is the greatest conversive $\mathbb{V}$-relator smaller than Γ.*

Proof. The proof consists of a number of straightforward calculations. As an example, we show that $\bigwedge_{i\in I} \Gamma_i$ in point 2 satisfies condition (𝕍-rel 2). Concretely, we have to prove

$$\bigwedge_{i\in I} \Gamma_i\beta \cdot \bigwedge_{i\in I} \Gamma_i\alpha \leq \bigwedge_{i\in I} \Gamma_i(\beta \cdot \alpha).$$

For that it is sufficient to prove that for any $j \in I$ we have:

$$\bigwedge_{i\in I} \Gamma_i\beta \cdot \bigwedge_{i\in I} \Gamma_i\alpha \leq \Gamma_j(\beta \cdot \alpha).$$

Observe that we have $\bigwedge_{i\in I} \Gamma_i\beta \leq \Gamma_j\beta$ and $\bigwedge_{i\in I} \Gamma_i\alpha \leq \Gamma_j\alpha$, so that by monotonicity of composition (recall that $\mathbb{V}$-Rel is a quantaloid) we infer $\bigwedge_{i\in I} \Gamma_i\beta \cdot \bigwedge_{i\in I} \Gamma_i\alpha \leq \Gamma_j\beta \cdot \Gamma_j\alpha$. The thesis now follows from (𝕍-rel 2). □

As we have already seen working with Λ_Σ, relators for a functor do not have enough structure to guarantee effectful applicative (bi)similarity to be compatible, so that we worked with the richer notion of a relator for a monad. This essentially holds for $\mathbb{V}$-Fuzz too. However, what we need in order to prove applicative distance(s) to be compatible are not relators for monads, but relators for *strong* monads.

The reason behind such requirement can be intuitively understood as follows. Recall that by Proposition 24 we have (for readability we omit types) $[\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!] = [\![f[x := -]]\!]^\dagger [\![e]\!]$. This operation can be described using the bind operator $\gg\!=\!: TX \times (X \to TY) \to TY$, so that we have $[\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!] = [\![e]\!] \gg\!= [\![f[x := -]]\!]$. Let now $f, g : X \to Y$ be functions, $\alpha : X \nrightarrow X, \beta : Y \nrightarrow Y$ be $\mathbb{V}$-relations, and Γ be a $\mathbb{V}$-relator for T. Considering the compound V relation $[\alpha, \Gamma\beta] \otimes \Gamma\alpha$ (see Remark 17) and ignoring issues about sensitivity, it is then natural to require $\gg\!=$ to be non-expansive. That is, we require the inequality

$$[\alpha, \Gamma\beta](f, g) \otimes \Gamma\alpha(\mathfrak{x}, \mathfrak{y}) \leq \Gamma\beta(\mathfrak{x} \gg\!= f, \mathfrak{y} \gg\!= g)$$

i.e.

$$\bigwedge_{x \in X} \Gamma\beta(f(x), g(x)) \otimes \Gamma\alpha(\mathfrak{x}, \mathfrak{y}) \leq \Gamma\beta(\mathfrak{x} \gg\!= f, \mathfrak{y} \gg\!= g).$$

Informally, we are requiring the behavioural distance between sequential compositions of programs to be bounded by the behavioural distances between their components (this is of course a too strong requirement, but at this point it should be clear to the reader that it is sufficient to require $\gg\!=$ to be Lipshitz-continuous, rather than non-expansive). Since $\gg\!=$ is nothing but the strong Kleisli extension apply^* of the application function $\mathrm{apply} : (X \to TY) \times X \to TY$ (which is defined by $\mathrm{apply}(f, x) \triangleq f(x)$), what we need to do is indeed to extend strong monads from Set to $\mathbb{V}$-Rel, laxly.

Definition 67. *Let $\mathbb{T} = \langle T, \eta, -^* \rangle$ be a strong monad, and Γ be a $\mathbb{V}$-relator for T. We say that Γ is an L-continuous $\mathbb{V}$-relator for $\mathbb{T}$ if it satisfies the following conditions for any CBE $s \leq 1$.*

$$s \circ \Gamma\alpha = \Gamma(s \circ \alpha) \tag{L-dist}$$

$$\alpha \leq \eta_Y^\circ \cdot \Gamma\alpha \cdot \eta_X \tag{Lax unit}$$

$$\gamma \otimes (s \circ \alpha) \leq g^\circ \cdot \Gamma\beta \cdot f \implies \gamma \otimes (s \circ \Gamma\alpha) \leq (g^*)^\circ \cdot \Gamma\beta \cdot f^* \tag{Strong lax bind}$$

We can represent conditions (Lax unit) *and* (Strong lax bind) *with the following lax commutative diagrams:*

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & TX \\ {\scriptstyle\alpha}\downarrow & \leq & \downarrow{\scriptstyle\Gamma\alpha} \\ Y & \xrightarrow[\eta_Y]{} & TY \end{array}$$

$$\begin{array}{ccc} Z \times X & \xrightarrow{f} & TY \\ {\scriptstyle\gamma\otimes\alpha}\downarrow & \leq & \downarrow{\scriptstyle\Gamma\beta} \\ Z' \times X' & \xrightarrow[g]{} & TY' \end{array} \implies \begin{array}{ccc} Z \times TX & \xrightarrow{f^*} & TY \\ {\scriptstyle\gamma\otimes\Gamma\alpha}\downarrow & \leq & \downarrow{\scriptstyle\Gamma\beta.} \\ Z' \times TX' & \xrightarrow[g^*]{} & TY' \end{array}$$

The condition $s \leq 1$ reflects the presence of $s \wedge 1$ in the typing rule for sequencing and it will be fundamental for our distances to be sound. Conditions (Lax unit) and (Strong lax bind) state non-expansiveness of unit and strong Kleisli extension (this is particularly clear when instantiated on the Lawvere quantale). Condition (L-dist) makes strong Kleisli extension Lipshitz-continuous with respect to CBEs. In fact, we can replace conditions (L-dist) and (Strong lax bind) with the following condition:

$$\gamma \otimes (s \circ \alpha) \leq g^\circ \cdot \Gamma(r \circ \beta) \cdot f \implies \gamma \otimes (s \circ \Gamma\alpha) \leq (g^*)^\circ \cdot (r \circ \Gamma\beta) \cdot f^* \tag{L-Strong lax bind}$$

where $s, r \leq 1$. Diagrammatically, we can express (L-Strong lax bind) as follows:

$$\begin{array}{ccc} Z \times X & \xrightarrow{f} & TY \\ {\scriptstyle\gamma\otimes s\circ\alpha}\downarrow & \leq & \downarrow{\scriptstyle\Gamma(r\circ\beta)} \\ Z' \times X' & \xrightarrow[g]{} & TY' \end{array} \implies \begin{array}{ccc} Z \times TX & \xrightarrow{f^*} & TY \\ {\scriptstyle\gamma\otimes(s\circ\Gamma\alpha)}\downarrow & \leq & \downarrow{\scriptstyle r\circ\Gamma\beta.} \\ Z' \times TX' & \xrightarrow[g^*]{} & TY' \end{array}$$

We immediately see that (L-dist) and (Strong lax bind) implies (L-Strong lax bind):

$$\begin{aligned} \gamma \otimes (s \circ \alpha) \leq g^\circ \cdot \Gamma(r \circ \beta) \cdot f &\implies \gamma \otimes \Gamma(s \circ \alpha) \leq (g^*)^\circ \cdot \Gamma(r \circ \beta) \cdot f^* \\ &\quad \text{[By (Strong lax bind)]} \\ &\implies \gamma \otimes (s \circ \Gamma\alpha) \leq (g^*)^\circ \cdot (r \circ \Gamma\beta) \cdot f^* \\ &\quad \text{[By (L-dist)].} \end{aligned}$$

Dually, we can decompose (*L*-Strong lax bind) in the following two conditions, where $s \leq 1$:

$$\gamma \otimes (s \circ \alpha) \leq g^\circ \cdot \Gamma\beta \cdot f \implies \gamma \otimes (s \circ \Gamma\alpha) \leq (g^*)^\circ \cdot \Gamma\beta \cdot f^* \qquad (L\text{-Strong lax bind 1})$$

$$\gamma \otimes \alpha \leq g^\circ \cdot \Gamma(s \circ \beta) \cdot f \implies \gamma \otimes \Gamma\alpha \leq (g^*)^\circ \cdot (s \circ \Gamma\beta) \cdot f^* \qquad (L\text{-Strong lax bind 2})$$

Instantiating (*L*-Strong lax bind 1) with Kleisli extension (rather than strong Kleisli extension) we obtain:

$$\begin{aligned}
\text{(Lax unit)} &\implies s \circ \alpha \leq \eta_Y^\circ \cdot \Gamma(s \circ \alpha) \cdot \eta_X \\
&\implies \Gamma(s \circ \alpha) \leq (\eta_Y^\dagger)^\circ \cdot (s \circ \Gamma\alpha) \cdot \eta_X^\dagger \\
&\quad \text{[By (}L\text{-Strong lax bind 1)]} \\
&\implies \Gamma(s \circ \alpha) \leq s \circ \Gamma\alpha.
\end{aligned}$$

In a similar fashion, using condition (*L*-Strong lax bind 2) we can prove $\Gamma(s \circ \alpha) \leq s \circ \Gamma\alpha$, and thus conclude (L-dist).

Before giving some relevant examples of $\mathbb{V}$-relator, we observe that we can extend Proposition 10 to $\mathbb{V}$-relators, provided that the CBEs are lattice morphisms, i.e. provided that the equality $s(a \wedge b) = s(a) \wedge s(b)$ (and thus $s \circ (\alpha \wedge \beta) = (s \circ \alpha) \wedge (s \circ \beta)$) holds, for any CBE s and elements $a, b \in \mathsf{V}$. In fact, if that is the case, then $\Gamma(s \circ \alpha) = s \circ \Gamma\alpha$ and $\Delta(s \circ \alpha) = s \circ \Delta\alpha$ imply:

$$\begin{aligned}
(\Gamma \wedge \Delta)(s \circ \alpha) &= \Gamma(s \circ \alpha) \wedge \Delta(s \circ \alpha) \\
&= (s \circ \Gamma\alpha) \wedge (s \circ \Delta\alpha) \\
&= s \circ (\Gamma\alpha \wedge \Delta\alpha) \\
&= s \circ ((\Gamma \wedge \Delta)\alpha).
\end{aligned}$$

To see that $s \circ \Gamma^\circ\alpha = \Gamma^\circ(s \circ \alpha)$ it is sufficient to observe that $s \circ \alpha^\circ = (s \circ \alpha)^\circ$. This shows that if Γ, Δ are relators (for a functor T) satisfying condition (L-dist), then so are $\Gamma \wedge \Delta$ and Γ°. A similar result can be proved for conditions (Lax unit) and (Strong lax bind) along the lines of the proof of Proposition 10.

11.1.1 Relevant Examples

Example 62. For the partiality monad $\mathbb{M}$ we define the $\mathbb{V}$-relator $\widetilde{\mathbb{M}}$ as:

$$\widetilde{\mathbb{M}}\alpha(\mathfrak{x}, \mathfrak{y}) \triangleq \begin{cases} \alpha(x, y) & \text{if } \mathfrak{x} = \mathit{just}\ x, \mathfrak{y} = \mathit{just}\ y \\ k & \text{if } \mathfrak{x} = \bot \\ \perp\!\!\!\perp & \text{otherwise.} \end{cases}$$

It is easy to see that $\widetilde{\mathbb{M}}$ defines a relator for M satisfying conditions (Lax unit) and (Strong lax bind). We also see that $\widetilde{\mathbb{M}}$ satisfies condition (L-dist), where the condition $s \leq 1$ turns out to be crucial. In fact, we see that $\widetilde{\mathbb{M}}(s \circ \alpha)(\mathit{just}\ x, \bot) = \perp\!\!\!\perp$ and $(s \circ \widetilde{\mathbb{M}}\alpha)(\mathit{just}\ x, \bot) = s(\perp\!\!\!\perp)$. Since $s \leq 1$, we have $s(\perp\!\!\!\perp) \leq \perp\!\!\!\perp$, and thus $s(\perp\!\!\!\perp) = \perp\!\!\!\perp$, as desired.

To see the problem related with CBEs greater than 1, let us consider the Lawvere quantale with CBEs given by extended multiplication by a constant. Let us consider a convergent program e (meaning that $[\![e]\!] = \mathit{just}\ v$, for some value v) and a divergent program f (meaning that $[\![f]\!] = \bot$). Given a $[0, \infty]$-relation α, we have $0 \cdot \widetilde{\mathbb{M}}\alpha(e, f) = 0 \cdot \infty = 0$, although $\widetilde{\mathbb{M}}(0 \cdot \alpha)(e, f) = \infty$, which is a rather undesired behaviour. Finally, we also have the conversive relator $\widetilde{\mathbb{M}} \wedge \widetilde{\mathbb{M}}^\circ$.

In Chapter 4 we showed how to sum monads and relators. It is immediate to see that the sum of monads extends to strong monads (we simply replace Kleisli extensions with strong Kleisli extensions). Similarly, we can also define the sum of $\mathbb{V}$-relators.

Proposition 26. *Given a strong monad $\mathbb{T} = \langle T, \varepsilon, -^{\mathbb{T}} \rangle$ with a $\mathbb{V}$-relator Γ for it, if Γ satisfies condition ($\text{ind}_{\mathbb{V}}$ 1) in Definition 69, then $\Gamma\widetilde{\mathbb{M}}$ is a relator for $\mathbb{T}\mathbb{M} = \langle TM, \eta, -^{\mathbb{T}\mathbb{M}} \rangle$.*

Proving Proposition 26 is rather straightforward. For instance, we have:

$$\begin{aligned} s \circ \Gamma\widetilde{\mathbb{M}}\alpha &= s \circ \Gamma(\widetilde{\mathbb{M}}\alpha) \\ &= \Gamma(s \circ \widetilde{\mathbb{M}}\alpha) \\ &= \Gamma(\widetilde{\mathbb{M}}(s \circ \alpha)) \\ &= \Gamma\widetilde{\mathbb{M}}(s \circ \alpha). \end{aligned}$$

In order to show that $\Gamma\widetilde{\mathbb{M}}$ satisfies condition (Strong lax bind), however, we need Γ to satisfy condition ($\text{ind}_{\mathbb{V}}$ 1). Let us assume such a condition and show that $\Gamma\widetilde{\mathbb{M}}$ satisfies (Strong lax bind). For that, we assume $\gamma \otimes \alpha \leq g^\circ \cdot \Gamma\widetilde{\mathbb{M}}\beta \cdot f$, and show $\gamma \otimes \Gamma\widetilde{\mathbb{M}}\alpha \leq g^\circ \cdot \Gamma\widetilde{\mathbb{M}}\beta \cdot f$, where f, g are functions with suitable source and target. As for the case of (boolean-valued) relators, we can easily derive the conclusion once we prove

$$\gamma(z, z') \otimes \widetilde{\mathbb{M}}\alpha(\mathfrak{x}, \mathfrak{x}') \leq \Gamma\widetilde{\mathbb{M}}\beta(f_\perp(z, \mathfrak{x}), g_\perp(z', \mathfrak{x}')), \tag{11.1}$$

where for any function $h : Z \times X \to TMY$ we define $h_\perp : Z \times MX \to TMY$ by:

$$h_\perp(z, \mathfrak{x}) \triangleq \begin{cases} h(z, x) & \text{if } \mathfrak{x} = \mathit{just}\ x \\ \varepsilon_{MY}(\perp) & \text{otherwise.} \end{cases}$$

Again, proving (11.1) is mostly trivial, except for $\mathfrak{x} = \perp$, in which case we have to prove $\gamma(z, z') \otimes k \leq \Gamma(\widetilde{\mathbb{M}}\beta)(\perp, g_\perp(z', \mathfrak{x}'))$. This is the case exactly if condition ($\text{ind}_{\mathbb{V}}$ 1) holds. ⊠

Example 63. For the monad $\mathbb{F}$ we define the $\mathbb{V}$-relator $\widetilde{\mathbb{F}}$ (called generalised Hausdorff lifting) as

$$\widetilde{\mathbb{F}}\alpha(\mathfrak{X}, \mathfrak{Y}) \triangleq \bigwedge_{x \in \mathfrak{X}} \bigvee_{y \in \mathfrak{Y}} \alpha(x, y).$$

If we instantiate $\mathbb{V}$ as the Lawvere quantale, then $\widetilde{\mathbb{F}}$ gives the (non-symmetric) Hausdorff lifting (Searcóid, 2006) whereas $\widetilde{\mathbb{F}} \wedge \widetilde{\mathbb{F}}^\circ$ gives the usual Hausdorff lifting. Similarly, when instantiating V as $\mathfrak{2}$, $\widetilde{\mathbb{F}}$ gives the ($\mathfrak{2}$-)relator $\hat{\mathbb{F}}$ of Chapter 4. It is easy to verify that $\widetilde{\mathbb{F}}$ is a relator for F that satisfies conditions (Lax unit) and (Strong lax bind). However, $\widetilde{\mathbb{F}}$ does not satisfy (L-dist), in general. In fact, for that to be the case we need CBEs to be continuous. ⊠

Example 64. We now introduce one of the most interesting $\mathbb{V}$-relators we will deal with. Let as instantiate $\mathbb{V}$ as the unit interval quantale $\mathbb{I}$. We define the map $\widetilde{\mathbb{D}}$ as the *Wasserstein-Kantorovich lifting* (Villani, 2008). Recall that for $\mu \in D(X), \nu \in D(Y)$, we denote by $\Omega(\mu, \nu)$ the set of *couplings* of μ and ν. Define $\widetilde{\mathbb{D}}\alpha$, for $\alpha : X \nrightarrow Y$ as follows:

$$\widetilde{\mathbb{D}}\alpha(\mu, \nu) \triangleq \inf_{\omega \in \Omega(\mu,\nu)} \sum_{x,y} \alpha(x, y) \cdot \omega(x, y).$$

Remarkably, $\widetilde{\mathbb{D}}\alpha(\mu, \nu)$ attains its infimum and has a dual characterisation.

Proposition 27. *Let $\mu \in D(X), \nu \in D(Y)$ be countable distributions and $\alpha : X \nrightarrow Y$ be a $\mathbb{I}$-relation. Then:*

$$\begin{aligned} \widetilde{\mathbb{D}}\alpha(\mu, \nu) &= \min\{\sum_{x,y} \alpha(x, y) \cdot \omega(x, y) \mid \omega \in \Omega(\mu, \nu)\} \\ &= \max\{\sum_x a_x \cdot \mu(x) + \sum_y b_y \cdot \nu(y) \mid a_x + b_y \leq \alpha(x, y), a_x, b_y \text{ bounded}\}, \end{aligned}$$

where a_x, b_y bounded means that there exist $\bar{a}, \bar{b} \in \mathbb{R}$ such that $\forall x.\ a_x \leq \bar{a}$, and $\forall y.\ b_y \leq \bar{b}$.

Proof. The proof is a direct consequence of the following duality theorem for countable transportation problems (Kortanek & Yamasaki, 1995) (Theorem 2.1 and 2.2).

Fact 1. Let $i, j, \ldots$ range over natural numbers. Let m_i, n_j, c_{ij} be non-negative real numbers, for all i, j. Define

$$M \triangleq \inf\{\sum_{i,j} c_{ij}x_{ij} \mid x_{ij} \geq 0, \sum_j x_{ij} = m_i, \sum_i x_{ij} = n_j, \}$$
$$M^* \triangleq \sup\{\sum_i m_i a_i + \sum_j n_j b_j \mid a_i + b_j \leq c_{ij}, a_i, b_j \text{ bounded}\}.$$

Then the following hold:
1. $M = M^*$.
2. The linear problem P induced by M has optimal solution.
3. The linear problem P^* induced by M^* has optimal solution.

First of all we notice that $\widetilde{\mathbb{D}}\alpha(\mu, \nu)$ is nothing but

$$\inf\{\sum_{x,y} \alpha(x, y) \cdot \omega(x, y) \mid \omega(x, y) \geq 0, \sum_y \omega(x, y) = \mu(x), \sum_x \omega(x, y) = \nu(y)\}.$$

In fact, $\omega(x, y) \geq 0$, $\sum_y \omega(x, y) = \mu(x)$ and $\sum_x \omega(x, y) = \nu(y)$ imply $\omega \in D(X \times Y)$. Moreover, since α is a $\mathbb{I}$-relation, $\alpha(x, y) \in [0, 1]$ (recall that Fact 1 requires c_{ij} to be a non-negative real number). We conclude the thesis by Fact 1. In particular, it follows that there exists $\omega \in \Omega(\mu, \nu)$ such that:

$$\widetilde{\mathbb{D}}\alpha(\mu, \nu) = \sum_{x,y} \alpha(x, y) \cdot \omega(x, y).$$

Since $\alpha(x, y), \omega(x, y) \in [0, 1]$ we have $\alpha(x, y) \cdot \omega(x, y) \leq \omega(x, y)$, for all x, y. It follows

$$0 \leq \sum_{x,y} \alpha(x, y) \cdot \omega(x, y) \leq \sum_{x,y} \omega(x, y) = 1$$

so that $\widetilde{\mathbb{D}}\alpha$ is indeed a $[0, 1]$-relation. □

Using Proposition 27 we can show that $\widetilde{\mathbb{D}}$ indeed defines a relator for D (but see Remark 20). We now prove that $\widetilde{\mathbb{D}}$ is also a relator for $\mathbb{D}$.

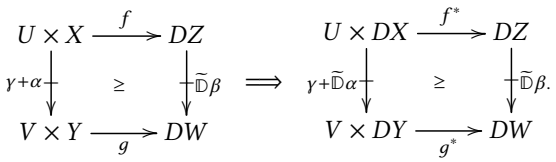
Proposition 28. *Wasserstein lifting $\widetilde{\mathbb{D}}$ satisfies all conditions in Definition 67.*

Proof. We start by showing that $\widetilde{\mathbb{D}}$ satisfies condition (Lax unit). It is convenient to work with the following notation: given an element $x \in X$ we denote by $|x\rangle$ the Dirac distribution on x, i.e. $\eta_X(x)$, where η is the unit of $\mathbb{D}$. We have to show that for any $z \in X, w \in Y$, $\alpha(z, w) \geq \widetilde{\mathbb{D}}\alpha(|z\rangle, |w\rangle)$ holds. By duality (Proposition 27) we have:

$$\widetilde{\mathbb{D}}\alpha(|z\rangle, |w\rangle) = \max\{\sum_x a_x \cdot |z\rangle(x) + \sum_y b_y \cdot |w\rangle(y) \mid a_x + b_y \leq \alpha(x, y)\},$$

where a_x, b_y are bounded. Clearly $\widetilde{\mathbb{D}}\alpha(|z\rangle, |w\rangle) = a_x + b_y$, for suitable $x \in X$ and $y \in Y$. Since $a_x + b_y \leq \alpha(x, y)$ we are done.

We now show that $\widetilde{\mathbb{D}}$ satisfies condition (Strong lax bind). Concretely, that amounts to prove the following implication:



We show that for any $u \in U, v \in V, \mu \in DX, \nu \in DY$ we have:

$$\widetilde{\mathbb{D}}\beta(f^*(u,\mu), g^*(v,\nu)) \leq \gamma(u,v) + \widetilde{\mathbb{D}}\alpha(\mu,\nu)$$

(notice that in the right hand side of the above equations we can assume without loss of generality to have ordinary addition in place of truncated sum). By very definition of strong Kleisli extension we have:

$$f^*(u,\mu)(z) = \sum_x \mu(x) \cdot f(u,x)(z)$$
$$g^*(v,\nu)(w) = \sum_y \nu(y) \cdot g(v,y)(w).$$

Let $\bar{m} \triangleq \widetilde{\mathbb{D}}\beta(f^*(u,\mu), g^*(v,\nu))$. By duality we have:

$$\bar{m} = \max\{\sum_z a_z \cdot \sum_x \mu(x) \cdot f(u,x)(z) + \sum_w b_w \cdot \sum_y \nu(y) \cdot g(v,y)(w) \mid a_z + b_w \leq \beta(z,w)\},$$

where a_z and b_w are bounded. By Proposition 27, there exists $\omega \in \Omega(\mu,\nu)$ such that $\widetilde{\mathbb{D}}\alpha(\mu,\nu) = \sum_{x,y} \omega(x,y) \cdot \alpha(x,y)$. We have to prove:

$$\bar{m} \leq \gamma(u,v) + \sum_{x,y} \omega(x,y) \cdot \alpha(x,y).$$

From $\omega \in \Omega(\mu,\nu)$ we obtain $\mu(x) = \sum_y \omega(x,y)$, $\nu(y) = \sum_x \omega(x,y)$. We apply the above equalities to $\bar{m}$, obtaining (for readability we omit the constraint $a_z + b_w \leq \beta(z,w)$):

$$\begin{aligned}
\bar{m} &= \max\{\sum_z a_z \cdot \sum_x \mu(x) \cdot f(u,x)(z) + \sum_w b_w \cdot \sum_y \nu(y) \cdot g(v,y)(w)\} \\
&= \max\{\sum_z a_z \cdot \sum_{x,y} \omega(x,y) \cdot f(u,x)(z) + \sum_w b_w \cdot \sum_{x,y} \omega(x,y) \cdot g(v,y)(w)\} \\
&= \max\{\sum_{x,y} \omega(x,y)(\sum_z a_z \cdot f(u,x)(z) + \sum_w b_w \cdot g(v,y)(w))\} \\
&= \sum_{x,y} \omega(x,y) \cdot \max\{\sum_z a_z \cdot f(u,x)(z) + \sum_w b_w \cdot g(v,y)(w)\} \\
&= \sum_{x,y} \omega(x,y) \cdot \widetilde{\mathbb{D}}\beta(f(u,x), g(v,y)).
\end{aligned}$$

We are now in position to use our hypothesis, namely the inequality:

$$\widetilde{\mathbb{D}}\beta(f(u,x), g(v,y)) \leq \gamma(u,v) + \alpha(f(u,x), g(v,y))$$

(notice that we actually have a stronger hypothesis, as the latter gives an inequality for truncated addition). We conclude:

$$\begin{aligned}\bar{m} &\leq \sum_{x,y} \omega(x,y) \cdot (\gamma(u,v) + \alpha(x,y)) \\ &= \sum_{x,y} \omega(x,y) \cdot \gamma(u,v) + \sum_{x,y} \omega(x,y) \cdot \alpha(x,y) \\ &= \gamma(u,v) + \sum_{x,y} \omega(x,y) \cdot \alpha(x,y)\end{aligned}$$

(where in the last equality we used the fact that $\omega(x,y) \in \Omega(\mu,\nu)$ implies $\sum_{x,y} \omega(x,y) = 1$). We are done.

Proving that $\widetilde{\mathbb{D}}$ satisfies condition (L-dist) is straightforward. In fact, given $c \in [0,1]$ we have:

$$\begin{aligned}c \cdot \widetilde{\mathbb{D}}\alpha(\mu,\nu) &= c \cdot \min\{\textstyle\sum_{x,y} \alpha(x,y) \cdot \omega(x,y) \mid \omega \in \Omega(\mu,\nu)\} \\ &= \min\{\textstyle\sum_{x,y} c \cdot \alpha(x,y) \cdot \omega(x,y) \mid \omega \in \Omega(\mu,\nu)\} \\ &= \widetilde{\mathbb{D}}(c \cdot \alpha)(\mu,\nu).\end{aligned}$$

□

Finally, we sum $\widetilde{\mathbb{D}}$ and $\widetilde{\mathbb{M}}$ obtaining a new relator $\widetilde{\mathbb{DM}}$ for $\mathbb{DM}$. ⊠

Example 65. Let us instantiate $\mathbb{V}$ as the strong Lawvere quantale. Given an alphabet $\mathcal{A}$, define the $\mathbb{SL}$-relation $\lambda : \mathcal{A}^\infty \nrightarrow \mathcal{A}^\infty$ by:

$$\lambda(w,u) \triangleq \begin{cases} 0 & \text{if } u = w \\ 2^{-\mathsf{lcp}(w,u)} & \text{otherwise,} \end{cases}$$

where $\mathsf{lcp}(w,u)$ is the length of the longest common prefix of u and w. We then define the map $\widetilde{\mathbb{O}^\infty}$ as follows, where $\alpha : X \nrightarrow Y$:

$$\widetilde{\mathbb{O}^\infty}\alpha(\langle w, \mathfrak{x}\rangle, \langle u, \mathfrak{y}\rangle) \triangleq \max(\lambda(u,w), (\widetilde{\mathbb{M}} \wedge \widetilde{\mathbb{M}^\circ})\alpha(\mathfrak{x},\mathfrak{y})).$$

It is not hard to see that $\widetilde{\mathbb{O}^\infty}$ is a conversive $[0,\infty]$-relator for $\mathbb{O}^\infty$. ⊠

Remark 20. Following Subsection 4.3.1 we might ask whether there exist canonical $\mathbb{V}$-relators for a monad. This is only partially the case. In fact, recall that Definition 22 crucially relies on the double nature of a relation, which can be viewed both as a morphism in Rel and as an object in Set. This is no longer the case for a $\mathbb{V}$-relation, and thus it is not clear how to define the Barr extension of a functor T from Set to $\mathbb{V}$-Rel. However, the Barr extension of T can be characterised in an alternative way if we assume T to preserves weak pullback diagrams (although the reader can see (Hofmann, 2007; E. G. Manes, 2002) for more general conditions). Let $\xi : T2 \to 2$ be the map defined by $\xi(\mathfrak{x}) = \mathsf{true}$ if and only if $\mathfrak{x} \in T\{\mathsf{true}\}$, where $T\{\mathsf{true}\}$ is the image of the map $T\iota$ for the inclusion $\iota : \{\mathsf{true}\} \to 2$. That is, $\xi(\mathfrak{x}) = \mathsf{true}$ if and only if there exists an element $\mathfrak{y} \in T\{\mathsf{true}\}$ such that $T\iota(\mathfrak{y}) = \mathfrak{x}$. Notice that this makes sense since T preserves monomorphisms (recall that we can describe monomorphism as weak pullbacks) and thus $T\iota : T\{\mathsf{true}\} \to T2$ is a monomorphism. We can now characterise $\overline{T}\mathcal{R}$ without mentioning the graph of $\mathcal{R}$:

$$\overline{T}\,\mathcal{R}\,(\mathfrak{x},\mathfrak{y}) = \mathsf{true} \iff \exists \mathfrak{w} \in T(X \times Y).\ \begin{cases} T\pi_1(\mathfrak{w}) & = \mathfrak{x}, \\ T\pi_2(\mathfrak{w}) & = \mathfrak{y}, \\ \xi \cdot T\,\mathcal{R}\,(\mathfrak{w}) & = \mathsf{true}. \end{cases}$$

Since the existential quantification is nothing but the join of the boolean quantale $\mathbb{2}$, the above characterisation of $\overline{T}$ can be turned into a definition of an extension of T to $\mathbb{V}$-Rel parametric with respect to an map $\xi : T\mathsf{V} \to \mathsf{V}$.

Definition 68. *For a set endofunctor T and a map $\xi : T\mathsf{V} \to \mathsf{V}$ define the* $\mathbb{V}$-Barr extension $\overline{T}_\xi$ *of T to* $\mathbb{V}$-Rel *with respect to ξ as follows:*

$$\overline{T}_\xi \alpha(\mathfrak{x}, \mathfrak{y}) \triangleq \bigvee_{\mathfrak{w} \in \Omega(\mathfrak{x},\mathfrak{y})} \xi \cdot T\alpha(\mathfrak{w}),$$

for $\mathfrak{x} \in TX, \mathfrak{y} \in TY$, where the set $\Omega(\mathfrak{x}, \mathfrak{y})$ of generalised couplings of $\mathfrak{x}, \mathfrak{y}$ is defined by:

$$\Omega(\mathfrak{x}, \mathfrak{y}) \triangleq \{\mathfrak{w} \in T(X \times Y) \mid T\pi_1(\mathfrak{w}) = \mathfrak{x},\ T\pi_2(\mathfrak{w}) = \mathfrak{y}\}.$$

We refer to ξ as the *structure map*, since it gives V the structure of a T-algebra. For instance, taking $\xi : F\mathsf{V} \to \mathsf{V}$ defined by $\xi(\mathfrak{X}) \triangleq \bigwedge \mathfrak{X}$ we recover the Hausdorff lifting $\widetilde{\mathbb{F}} \wedge \widetilde{\mathbb{F}^\circ}$. Similarly, taking ξ as the expectation function, so that $\xi : D[0,1] \to [0,1]$ is defined by $\xi(\mu) \triangleq \sum_x x \cdot \mu(x)$, we recover $\widetilde{\mathbb{D}}$.

Using the map $\xi : T\mathsf{V} \to \mathsf{V}$ we can define an extension of T to $\mathbb{V}$-Rel. However, such extension is in general not a $\mathbb{V}$-relator. Nonetheless, under mild conditions on ξ and assuming T to preserve weak pullback, it is possible to show that $\overline{T}_\xi$ is indeed a $\mathbb{V}$-relator. The following proposition has been proved in (Clementino & Tholen, 2014; Hofmann, 2007) (a similar result for real-valued pseudometric spaces has been proved in (Baldan, Bonchi, Kerstan, & König, 2014, 2015)).

Proposition 29. *Let $\mathbb{T}$ be a monad with carrier T preserving weak pullbacks, and $\xi : T\mathsf{V} \to \mathsf{V}$ be a structure map. If ξ satisfies the following conditions, then $\overline{T}_\xi$ is a conversive $\mathbb{V}$-relator for $\mathbb{T}$. Moreover, if conditions 1 and 2 belows are equalities, then $\overline{T}_\xi$ is functorial.*

1. *ξ respect quantale multiplication:*

$$\begin{array}{ccc} T(\mathsf{V} \times \mathsf{V}) & \xrightarrow{T\otimes} & T\mathsf{V} \\ {\scriptstyle \langle \xi \cdot T\pi_1, \xi \cdot T\pi_2 \rangle}\downarrow & \leq & \downarrow{\scriptstyle \xi} \\ \mathsf{V} \times \mathsf{V} & \xrightarrow[\otimes]{} & \mathsf{V} \end{array}.$$

2. *ξ respects the unit of the quantale:*

$$\begin{array}{ccc} T1 & \xrightarrow{Tk} & T\mathsf{V} \\ {\scriptstyle !}\downarrow & \leq & \downarrow{\scriptstyle \xi} \\ 1 & \xrightarrow[k]{} & \mathsf{V} \end{array}.$$

3. *ξ respects the order of the quantale. That is, the map $\varphi \mapsto \xi \cdot T\varphi$, for $\varphi : X \to \mathsf{V}$, is monotone.*

Finally, an open question (which the author has not yet investigated) concerns whether non-conversive $\mathbb{V}$-relators can be defined along the lines of Subsection 4.3.1. This is conjectured to be the case, provided that we replace stable preorders with stable generalised metrics, i.e. with functors F mapping each set X to a $\mathbb{V}$-category (FX, α_X) such that for each map $f : X \to Y$ in Set, $Ff : (FX, \alpha_X) \to (FY, \alpha_y)$ is a $\mathbb{V}$-enriched functor.

At this point it should be clear to the reader that most of the relators studied in Section 4.3 are instances of more general $\mathbb{V}$-relators when $\mathbb{V} = \mathbb{2}$. We now make this observation formal. Recall the change of base functor $\psi : 2 \to \mathsf{V}$ and its right adjoint $\varphi : \mathsf{V} \to 2$ of Example 57:

$$\psi(\mathsf{true}) \triangleq k \qquad \varphi(k) \triangleq \mathsf{true}$$
$$\psi(\mathsf{false}) \triangleq \bot\!\!\!\bot \qquad \varphi(a) \triangleq \mathsf{false}$$

Using φ and ψ we associate to every $\mathbb{V}$-relation α its kernel 2-relation $\varphi \circ \alpha$ and to any 2-relation $\mathcal{R}$ the $\mathbb{V}$-relation $\psi \circ \mathcal{R}$. Similarly, we can associate to each $\mathbb{V}$-relator Γ the 2-relator $\Delta_\Gamma \mathcal{R} \triangleq \varphi \circ \Gamma(\psi \circ \mathcal{R})$. Moreover, since φ is the right adjoint of ψ we have the inequalities:

$$\psi \circ \Delta_\Gamma \mathcal{R} \leq \Gamma(\psi \circ \mathcal{R})$$
$$\Delta_\Gamma(\varphi \circ \alpha) \leq \varphi \circ \Gamma\alpha.$$

If $\Delta_\Gamma(\varphi \circ \alpha) = \varphi \circ \Gamma\alpha$, we say that Γ is compatible with φ.

Example 66. For $\mathbb{T} \in \{\mathbb{M}, \mathbb{F}, \mathbb{D}\}$ we see that $\Delta_{\widetilde{\mathbb{T}}} = \hat{\mathbb{T}}$ and $\Delta_{\widetilde{\mathbb{T}} \wedge \widetilde{\mathbb{T}}^\circ} = \hat{\mathbb{T}} \wedge \hat{\mathbb{T}}^\circ$. Moreover, straightforward calculations show that $\Delta_{\widetilde{\mathbb{T}}}(\varphi \circ \alpha) = \varphi \circ \widetilde{\mathbb{T}}\alpha$. ⊠

Finally, we see that if Γ is compatible with φ, then we have the following inequalities.

Lemma 38. *Let Γ be $\mathbb{V}$-relator compatible with φ. Then the following hold:*

$$\begin{array}{ccc} X & \xrightarrow{f} & TZ \\ {\scriptstyle\alpha}\downarrow & \leq & \downarrow{\scriptstyle\Gamma\beta} \\ Y & \xrightarrow[g]{} & TW \end{array} \implies \begin{array}{ccc} X & \xrightarrow{f} & TZ \\ {\scriptstyle\varphi\circ\alpha}\downarrow & \leq & \downarrow{\scriptstyle\Delta_\Gamma(\varphi\circ\beta)} \\ Y & \xrightarrow[g]{} & TW \end{array},$$

$$\begin{array}{ccc} X & \xrightarrow{f} & TZ \\ {\scriptstyle\mathcal{R}}\downarrow & \leq & \downarrow{\scriptstyle\Delta_\Gamma\mathcal{S}} \\ Y & \xrightarrow[g]{} & TW \end{array} \implies \begin{array}{ccc} X & \xrightarrow{f} & TZ \\ {\scriptstyle\psi\circ\mathcal{R}}\downarrow & \leq & \downarrow{\scriptstyle\Gamma(\psi\circ\mathcal{S})} \\ Y & \xrightarrow[g]{} & TW \end{array}.$$

Proof. The proof is straightforward. As an illustrative example, we show

$$\alpha \leq g^\circ \cdot \Gamma\beta \cdot f \implies \varphi \circ \alpha \leq g^\circ \cdot \Delta_\Gamma(\varphi \circ \beta) \cdot f.$$

We have:

$$\begin{aligned} \alpha \leq g^\circ \cdot \Gamma\beta \cdot f &\implies \varphi \circ \alpha \leq g^\circ \cdot (\varphi \circ \Gamma\beta) \cdot f \\ &\quad \text{[By monotonicity of } \varphi\text{]} \\ &\implies \varphi \circ \alpha \leq g^\circ \cdot \Delta_\Gamma(\varphi \circ \beta) \cdot f \\ &\quad \text{[By compatibility of } \Gamma \text{ with } \varphi\text{]}. \end{aligned}$$

□

As for ordinary, boolean-valued relators we required some compatibility conditions with the Σ-continuous structure of monads, we do the same for $\mathbb{V}$-relators.

Definition 69. *Let $\mathbb{T}$ be a Σ-continuous monad, $\mathbb{V}$ be a Σ-quantale, and Γ be a $\mathbb{V}$-relator for $\mathbb{T}$.*

1. *We say that* Γ *is* inductive *if the following inequalities hold:*

$$k \leq \Gamma\alpha(\bot_X, \mathfrak{y}) \tag{$\text{ind}_\mathbb{V}$ 1}$$

$$\bigwedge_n \Gamma\alpha(\mathfrak{x}_n, \mathfrak{y}) \leq \Gamma\alpha(\bigsqcup_n \mathfrak{x}_n, \mathfrak{y}) \tag{$\text{ind}_\mathbb{V}$ 2}$$

for any ω*-chain* $(\mathfrak{x}_n)_n$ *in* TX*, element* $\mathfrak{y} \in TY$*, and* $\mathbb{V}$*-relation* $\alpha : X \nrightarrow Y$*.*

2. *We say that* Γ *is* Σ-compatible *if it satisfies the following condition, for all elements* $\mathfrak{u}_1, \ldots, \mathfrak{u}_n \in TX$*,* $\mathfrak{y}_1, \ldots, \mathfrak{y}_n \in TY$*, and* n*-ary operation symbol* $\mathbf{op} \in \Sigma$*.*

$$[\![\mathbf{op}]\!]_\mathbb{V}(\Gamma\alpha(\mathfrak{u}_1, \mathfrak{y}_1), \ldots \Gamma\alpha(\mathfrak{u}_n, \mathfrak{y}_n)) \leq \Gamma\alpha([\![\mathbf{op}]\!]_X(\mathfrak{u}_1, \ldots, \mathfrak{u}_n), [\![\mathbf{op}]\!]_Y(\mathfrak{y}_1, \ldots, \mathfrak{y}_n)) \qquad (\Sigma\ \text{comp}_\mathbb{V})$$

Notice that if Γ is inductive then we have the following 'induction-like' principle:

$$\forall n \geq 0.\ a \leq \Gamma\alpha(\bigsqcup_n \mathfrak{x}_n, \mathfrak{y}) \implies a \leq \Gamma\alpha(\mathfrak{x}_n, \mathfrak{y}).$$

We refer to inductive and Σ-compatible $\mathbb{V}$-relators for a monad $\mathbb{T}$ as Σ*-continuous* $\mathbb{V}$*-relators.*

Example 67. Easy calculations show that $\widetilde{\mathbb{M}}$ and $\widetilde{\mathbb{FM}}$ are inductive and Σ-compatible. Using results from (Villani, 2008) and (Clément & Desch, 2008) (notably Lemma 5.2) it is possible to show that $\widetilde{\mathbb{DM}}$ is inductive, the relevant inequality being $\widetilde{\mathbb{DM}}\alpha(\sup_n \mu_n, \nu) \leq \sup_n \widetilde{\mathbb{DM}}\alpha(\mu_n, \nu)$. Proving Σ-compatibility of $\widetilde{\mathbb{D}}$ and $\widetilde{\mathbb{DM}}$ is straightforward, as it amounts to show $\Gamma\alpha(\mu_1 \oplus \nu_1, \mu_2 \oplus \nu_2) \leq \Gamma\alpha(\mu_1, \mu_2) \oplus \Gamma\alpha(\nu_1, \nu_2)$, for $\Gamma \in \{\widetilde{\mathbb{D}}, \widetilde{\mathbb{DM}}\}$. ⊠

This concludes our exposition of the general theory of $\mathbb{V}$-relators. We are now going to apply such a theory to define abstract notions of program distance for $\mathbb{V}$-Fuzz.

Chapter 12

Effectful Applicative Distances

> Là onde, come sarebe male la abolizione ed il non essere di questo mondo, cossì' non sarebe buono il non essere de innumerabili altri.
>
> Giordano Bruno, De l'infinito, universo e mondi

In this chapter we define effectful applicative similarity and bisimilarity distance and prove pre-congruence and congruence results for them, respectively. In order to achieve these results, we first generalise the relational framework of Chapter 5 to $\mathbb{V}$-Fuzz and $\mathbb{V}$-relations, and the instantiate it to define our notions of effectful distance.

Effectful applicative similarity and bisimilarity distance being defined coinductively, we easily prove them to be preorder and equivalence $\mathbb{V}$-relations, respectively. Additionally, relying on Lemma 38 we prove that the kernel of effectful applicative (bi)similarity distance coincides with a refinement of effectful applicative (bi)similarity (Proposition 31).

(Pre)congruence results are proved using a refinement of abstract Howe's method as developed in Chapter 5, taking inspiration from (Crubillé & Dal Lago, 2015). As for the proof of Lemma 16, the proof of Lemma 43 relies on the axioms of $\mathbb{V}$-relators.

In the rest of this chapter we assume a signature Σ, a Σ-quantale $\mathbb{V}$, a collection of CBEs Π (according to Section 10.2), a Σ-continuous (strong) monad $\mathbb{T}$, and a Σ-continuous relator for $\mathbb{T}$ to be fixed.

12.1 Behavioural $\mathbb{V}$-relations

We begin our analysis extending the relational framework developed in Chapter 5 to $\mathbb{V}$-Fuzz and $\mathbb{V}$-relations. Since the development of this section closely follows Section 5.1, we will be rather succinct, not commenting routine definitions and results (the reader should already be familiar with them from Chapter 5).

Definition 70. *A* closed *λ-term $\mathbb{V}$-relation $\alpha = (\alpha^{\wedge}, \alpha^{\vee})$ associates to each closed type σ, binary $\mathbb{V}$-relations $\alpha^{\vee}_{\sigma}, \alpha^{\wedge}_{\sigma}$ on closed values and computations inhabiting it, respectively.*

Since the syntactic shape of expressions determines whether we are dealing with computations or values, oftentimes we will write $\alpha_{\sigma}(e, f)$ (resp. $\alpha_{\sigma}(v, w)$) in place of $\alpha^{\wedge}_{\sigma}(e, f)$ (resp. $\alpha^{\vee}_{\sigma}(v, w)$).

Definition 71. *An* open *λ-term $\mathbb{V}$-relation α associates to each sequent $\Gamma \vdash^{\wedge} \sigma$ a $\mathbb{V}$-relation $\Gamma \vdash^{\wedge} \alpha(-,-) : \sigma$ on computations inhabiting it, and to each sequent $\Gamma \vdash^{\vee} \sigma$ a $\mathbb{V}$-relation $\Gamma \vdash^{\vee} \alpha(-,-) : \sigma$ on values inhabiting it. We require open λ-term $\mathbb{V}$-relations to be closed under weakening, i.e. for any environment Δ we require:*

$$(\Gamma \vdash^{\wedge} \alpha(e,f) : \sigma) \leq (\Gamma \otimes \Delta \vdash^{\wedge} \alpha(e,f) : \sigma)$$
$$(\Gamma \vdash^{\vee} \alpha(v,w) : \sigma) \leq (\Gamma \otimes \Delta \vdash^{\vee} \alpha(v,w) : \sigma).$$

As for closed λ-term $\mathbb{V}$-relations, we will often write $\Gamma \vdash \alpha(v,w) : \sigma$ in place of $\Gamma \vdash^{\vee} \alpha(v,w) : \sigma$ (and similarity for computations) and simply refer to open λ-term $\mathbb{V}$-relations as λ-term $\mathbb{V}$-relations (whenever relevant we will explicitly mention whether we are dealing with open or closed λ-term $\mathbb{V}$-relations).

We notice that the collection of open λ-term $\mathbb{V}$-relations carries a complete lattice structure (ordered pointwise), meaning that we can define λ-term $\mathbb{V}$-relations both inductively and coinductively. Moreover, we can always extend a closed λ-term $\mathbb{V}$-relation $\alpha = (\alpha^{\wedge}, \alpha^{\vee})$ to an open one (as well as restricting an open λ-term $\mathbb{V}$-relation to a closed one).

Definition 72. *Let $\Gamma \triangleq x_1 :_{s_1} \sigma_1, \dots, x_n :_{s_n} \sigma_n$ be an environment. For values $\vec{v} \triangleq v_1, \dots, v_n$ we write $\vec{v} : \Gamma$ if for any $i \leq n$, $\vdash^{\vee} v_i : \sigma_i$ holds. Given a closed λ-term $\mathbb{V}$-relation $\alpha = (\alpha^{\wedge}, \alpha^{\vee})$ we define its open extension α^o as follows:*

$$\Gamma \vdash^{\wedge} \alpha^o(e,f) : \tau \triangleq \bigwedge\nolimits_{\vec{v}:\Gamma} \alpha^{\wedge}_{\tau}(e[\vec{x} := \vec{v}], f[\vec{x} := \vec{v}])$$
$$\Gamma \vdash^{\vee} \alpha^o(v,w) : \tau \triangleq \bigwedge\nolimits_{\vec{u}:\Gamma} \alpha^{\vee}_{\tau}(v[\vec{u}/\vec{x}], w[\vec{u}/\vec{x}]).$$

In the following we will adopt the notational conventions introduced in Chapter 5. We now define the notions of *substitutivity* and *compatibility* for λ-term $\mathbb{V}$-relations. Due to the presence of program sensitivity annotations in $\mathbb{V}$-Fuzz, these notions are quite different than their boolean-valued counterparts for Λ_{Σ}.

Definition 73. *We say that an open λ-term $\mathbb{V}$-relation α is* value substitutive *if for all values $\Gamma, x :_s \sigma \vdash^{\vee} v, w : \tau$ and computations $\Gamma, x :_s \sigma \vdash^{\wedge} e, f : \tau$ we have:*

$$(\Gamma, x :_s \sigma \vdash^{\vee} \alpha(v,w) : \tau) \leq (\Gamma \vdash^{\vee} \alpha(v[u/x], w[u/x]) : \tau)$$
$$(\Gamma, x :_s \sigma \vdash^{\wedge} \alpha(e,f) : \tau) \leq (\Gamma \vdash^{\wedge} \alpha(e[x := u], f[x := u]) : \tau)$$

for any closed value $u \in \mathcal{V}^{\sigma}_{\circ}$. We say that α is substitutive *if for all values $\Gamma, x :_s \sigma \vdash^{\vee} v, w : \tau$ and computations $\Gamma, x :_s \sigma \vdash^{\wedge} e, f : \tau$ we have:*

$$(\Gamma, x :_s \sigma \vdash^{\vee} \alpha(v,w) : \tau) \otimes (s \circ \alpha^{\vee}_{\sigma}(u,u')) \leq \Gamma \vdash^{\vee} \alpha(v[u/x], w[u'/x]) : \tau$$
$$(\Gamma, x :_s \sigma \vdash^{\wedge} \alpha(e,f) : \tau) \otimes (s \circ \alpha^{\vee}_{\sigma}(u,u')) \leq \Gamma \vdash^{\wedge} \alpha(e[x := u], f[x := u']) : \tau$$

for all closed values u, u' of type σ.

The notion of *compatibility* captures an abstract form of Lipshitz-continuity with respect to $\mathbb{V}$-Fuzz constructors. Formally, we define the notion of a compatible λ-term $\mathbb{V}$-relation by means of the *compatible refinement* operator.

Definition 74. *The* compatible refinement *$\widehat{\alpha}$ of an open λ-term $\mathbb{V}$-relation α is defined by:*

$$(\Gamma \vdash \widehat{\alpha}(e,f) : \sigma) \triangleq \bigvee \{a \mid \Gamma \models^{\wedge} a \leq \widehat{\alpha}(e,f) : \sigma\}$$
$$(\Gamma \vdash^{\vee} \widehat{\alpha}(v,w) : \sigma) \triangleq \bigvee \{a \mid \Gamma \models^{\vee} a \leq \widehat{\alpha}(v,w) : \sigma\}$$

where judgments $\Gamma \models^{\wedge} a \leq \widehat{\alpha}(e,f) : \sigma$ and $\Gamma \models^{\vee} a \leq \widehat{\alpha}(v,w) : \sigma$ are inductively defined for $a \in \mathsf{V}$, $\Gamma \vdash^{\wedge} e, f : \sigma$, and $\Gamma \vdash^{\vee} v, w : \sigma$ by rules in Figure 12.1. We say that α is compatible if $\widehat{\alpha} \leq \alpha$.

$$\frac{}{\Gamma, x :_s \sigma \models^{\wedge} k \leq \widehat{\alpha}(x, x) : \sigma} \ (\text{comp}_{\vee}\text{-var})$$

$$\frac{a \leq \Gamma, x :_1 \sigma \vdash \alpha(e, f) : \tau}{\Gamma \models^{\vee} a \leq \widehat{\alpha}(\lambda x.e, \lambda x.f) : \sigma \multimap \tau} \ (\text{comp}_{\vee}\text{-abs})$$

$$\frac{a \leq \Gamma \vdash^{\vee} \alpha(v, v') : \sigma \multimap \tau \quad b \leq \Delta \vdash^{\vee} \alpha(w, w') : \sigma}{\Gamma \otimes \Delta \models^{\wedge} a \otimes b \leq \widehat{\alpha}(vw, v'w') : \tau} \ (\text{comp}_{\vee}\text{-app})$$

$$\frac{a \leq \Gamma \vdash^{\vee} \alpha(v, w) : \sigma_{\hat{\imath}}}{\Gamma \models^{\vee} a \leq \widehat{\alpha}(\langle \hat{\imath}, v\rangle, \langle \hat{\imath}, w\rangle) : \sum_{i \in I} \sigma_i} \ (\text{comp}_{\vee}\text{-inj})$$

$$\frac{a \leq \Gamma \vdash^{\vee} \alpha(\langle \hat{\imath}, v\rangle, \langle \hat{\imath}, w\rangle) : \sum_{i \in I} \sigma_i \quad b_i \leq \Delta, x :_{s_i} \sigma_i \vdash \leq \alpha(e_i, f_i) : \tau \quad (\forall i \in I)}{s \cdot \Gamma \otimes \Delta \models^{\wedge} s(a) \otimes b_i \leq \widehat{\alpha}(\textbf{case}\ \langle \hat{\imath}, v\rangle\ \textbf{of}\ \{\langle i, x\rangle \to e_i\}, \textbf{case}\ \langle \hat{\imath}, w\rangle\ \textbf{of}\ \{\langle i, x\rangle \to f_i\}) : \tau} \ (\text{comp}_{\vee}\text{-sum-cases})$$

$$\frac{a \leq \Gamma \vdash^{\vee} \alpha(v, w) : \sigma}{\Gamma \models^{\wedge} a \leq \widehat{\alpha}(\textbf{return}\ v, \textbf{return}\ w) : \sigma} \ (\text{comp}_{\vee}\text{-val})$$

$$\frac{a \leq \Gamma \vdash \alpha(e, e') : \sigma \quad b \leq \Delta, x :_s \sigma \vdash \alpha(f', f') : \tau}{(s \wedge 1) \cdot \Gamma \otimes \Delta \models^{\wedge} (s \wedge 1)(a) \otimes b \leq \widehat{\alpha}(\textbf{let}\ x = e\ \textbf{in}\ f, \textbf{let}\ x = e'\ \textbf{in}\ f') : \tau} \ (\text{comp}_{\vee}\text{-let})$$

$$\frac{a \leq \Gamma \models^{\wedge} \alpha(v, w) : \sigma}{s \cdot \Gamma \models^{\vee} s(a) \leq \alpha(!v, !w) : !_s\sigma} \ (\text{comp}_{\vee}\text{-bang})$$

$$\frac{a \leq \Gamma \vdash^{\vee} \alpha(v, w) : !_r\sigma \quad b \leq \Delta, x :_{s \cdot r} \sigma \vdash \alpha(e, f) : \tau}{s \cdot \Gamma \otimes \Delta \models^{\wedge} s(a) \otimes b \leq \widehat{\alpha}(\textbf{case}\ v\ \textbf{of}\ \{!x \to e\}, \textbf{case}\ w\ \textbf{of}\ \{!x \to f\}) : \tau} \ (\text{comp}_{\vee}\text{-bang-cases})$$

$$\frac{a\Gamma \vdash^{\vee} \alpha(v, w) : \sigma[\mu t.\sigma/t]}{\Gamma \models^{\vee} a \leq \widehat{\alpha}(\textbf{fold}\ v, \textbf{fold}\ w) : \mu t.\sigma} \ (\text{comp}_{\vee}\text{-fold})$$

$$\frac{a \leq \Gamma \vdash^{\vee} \alpha(v, w) : \mu t.\sigma \quad b \leq \Delta, x :_s \sigma[\mu t.\sigma/t] \vdash b \leq \alpha(e, f) : \tau}{s \cdot \Gamma \otimes \Delta \models^{\wedge} s(a) \otimes b \leq \widehat{\alpha}(\textbf{case}\ v\ \textbf{of}\ \{\textbf{fold}\ x \to e\}, \textbf{case}\ w\ \textbf{of}\ \{\textbf{fold}\ x \to f\}) : \tau} \ (\text{comp}_{\vee}\text{-fold-cases})$$

$$\frac{a_1 \leq \Gamma_1 \vdash \alpha(e_1, f_1) : \sigma \quad \cdots \quad a_n \leq \Gamma_n \vdash \alpha(e_n, f_n) : \sigma}{[\![\textbf{op}]\!]_{\vee}(\Gamma_1, \ldots, \Gamma_n) \models^{\wedge} [\![\textbf{op}]\!]_{\vee}(a_1, \ldots, a_n) \leq \widehat{\alpha}(\textbf{op}(e_1, \ldots, e_n), \textbf{op}(e_1, \ldots, e_n)) : \sigma} \ (\text{comp}_{\vee}\text{-op})$$

Figure 12.1: Compatible refinement.

$$k \leq (\Gamma \vdash^{\mathsf{v}} \alpha(x,x) : \sigma)$$

$$\Gamma, x :_1 \sigma \vdash^{\Lambda} \alpha(e,f) : \tau \leq \Gamma \vdash^{\mathsf{v}} \alpha(\lambda x.e, \lambda x.f) : \sigma \multimap \tau$$

$$(\Gamma \vdash^{\mathsf{v}} \alpha(v,v') : \sigma \multimap \tau) \otimes (\Delta \vdash^{\mathsf{v}} \alpha(w,w') : \sigma) \leq (\Gamma \otimes \Delta \vdash^{\Lambda} \alpha(vw, v'w') : \tau)$$

$$\Gamma \vdash^{\mathsf{v}} \alpha(v,w) : \sigma_{\hat{\imath}} \leq \Gamma \vdash^{\mathsf{v}} \alpha(\langle \hat{\imath}, v\rangle, \langle \hat{\imath}, w\rangle) : \sum_{i\in I} \sigma_i$$

$$s \circ (\Gamma \vdash^{\mathsf{v}} \alpha(\langle \hat{\imath}, v\rangle, \langle \hat{\imath}, w\rangle) : \sum_{i\in I} \sigma_i) \otimes (\Delta, x :_s \sigma \vdash^{\Lambda} \alpha(e_{\hat{\imath}}, f_{\hat{\imath}}) : \tau) \leq$$
$$s \cdot \Gamma \otimes \Delta \vdash^{\Lambda} \alpha(\mathbf{case}\ \langle \hat{\imath}, v\rangle\ \mathbf{of}\ \{\langle i, x\rangle \to e_i\}, \mathbf{case}\ \langle \hat{\imath}, w\rangle\ \mathbf{of}\ \{\langle i, x\rangle \to f_i\}) : \tau$$

$$\Gamma \vdash^{\mathsf{v}} \alpha(v,w) : \sigma \leq \Gamma \vdash^{\Lambda} \alpha(\mathbf{return}\ v, \mathbf{return}\ w) : \sigma$$

$$(s \wedge 1) \circ (\Gamma \vdash^{\Lambda} \alpha(e,e') : \sigma) \otimes (\Delta, x :_s \sigma \vdash^{\Lambda} \alpha(f,f') : \tau) \leq$$
$$(s \wedge 1) \cdot \Gamma \otimes \Delta \vdash^{\Lambda} \alpha(\mathbf{let}\ x = e\ \mathbf{in}\ f, \mathbf{let}\ x = e'\ \mathbf{in}\ f') : \tau$$

$$s \circ (\Gamma \vdash^{\mathsf{v}} \alpha(v,w) : \sigma) \leq s \cdot \Gamma \vdash^{\mathsf{v}} \alpha(!v, !w) : !_s\sigma$$

$$s \circ (\Gamma \vdash^{\mathsf{v}} \alpha(v,w) : !_r\sigma) \otimes (\Delta, x :_{s\cdot r} \sigma \vdash^{\Lambda} \alpha(e,f) : \tau) \leq$$
$$s \cdot \Gamma \otimes \Delta \vdash^{\Lambda} \alpha(\mathbf{case}\ v\ \mathbf{of}\ \{!x \to e\}, \mathbf{case}\ w\ \mathbf{of}\ \{!x \to f\}) : \tau$$

$$\Gamma \vdash^{\mathsf{v}} \alpha(v,w) : \sigma[\mu t.\sigma/t] \leq \Gamma \vdash^{\mathsf{v}} \alpha(\mathbf{fold}\ v, \mathbf{fold}\ w) : \mu t.\sigma$$

$$s \circ (\Gamma \vdash^{\mathsf{v}} \alpha(v,w) : \mu t.\sigma) \otimes (\Delta, x :_s \sigma[\mu t.\sigma/t] \vdash^{\Lambda} \alpha(e,f) : \tau) \leq$$
$$s \cdot \Gamma \otimes \Delta \vdash^{\Lambda} \alpha(\mathbf{case}\ v\ \mathbf{of}\ \{\mathbf{fold}\ x \to e\}, \mathbf{case}\ w\ \mathbf{of}\ \{\mathbf{fold}\ x \to f\}) : \tau$$

$$[\![\mathbf{op}]\!]_{\mathsf{V}}(\Gamma_1 \vdash^{\Lambda} \alpha(e_1,f_1) : \sigma, \ldots, \Gamma_n \vdash^{\Lambda} \alpha(e_n,f_n) : \sigma) \leq$$
$$[\![\mathbf{op}]\!]_{\mathsf{V}}(\Gamma_1, \ldots, \Gamma_n) \vdash^{\Lambda} \alpha(\mathbf{op}(e_1, \ldots, e_n), \mathbf{op}(f_1, \ldots, f_n)) : \sigma$$

Figure 12.2: Compatibility clauses.

It is easy to see that if α is compatible, then it satisfies inequalities in Figure 12.2. Actually, α is compatible precisely if it satisfies such inequalities. Notice also that in the clause for sequential composition the presence of $s \wedge 1$, instead of s, ensures that for terms like $e \triangleq \mathbf{let}\ x = (\mathbf{return}\ I)\ \mathbf{in}\ f$ and $e' \triangleq \mathbf{let}\ x = \Omega\ \mathbf{in}\ f$, where f is a *closed* computation, the distance $\alpha(e,e')$ is determined *before* sequencing (which captures the idea that although f will not 'use' any input, **return** I and Ω will be still evaluated, thus producing observable differences between e and e'). In fact, if we replace $s \wedge 1$ with s, then by taking $s \triangleq 0$ compatibility would imply $\alpha(e,e') = k$, which is clearly unsound.

We can finally define *effectful applicative similarity distance*, the generalisation of effectful applicative similarity of Chapter 5 to $\mathbb{V}$-relations.

Definition 75. *Let Γ be a $\mathbb{V}$-relator for $\mathbb{T}$ and $\alpha = (\alpha^{\Lambda}, \alpha^{\mathsf{V}})$ be a closed λ-term $\mathbb{V}$-relation. Define the closed*

λ-term $\mathbb{V}$-relation $[\alpha] = ([\alpha]^{\wedge}, [\alpha]^{\vee})$ as follows:

$$
\begin{aligned}
[\alpha]^{\wedge}_{\sigma}(e,f) &\triangleq \Gamma\alpha^{\vee}_{\sigma}([\![e]\!],[\![f]\!]) \\
[\alpha]^{\vee}_{\sigma\multimap\tau}(v,w) &\triangleq \bigwedge\nolimits_{u\in\mathcal{V}_\circ^{\sigma}} \alpha^{\wedge}_{\tau}(vu,wu) \\
[\alpha]^{\vee}_{\sum_{i\in I}\sigma_i}(\langle \hat{\imath}, v\rangle, \langle \hat{\imath}, w\rangle) &\triangleq \alpha^{\vee}_{\sigma_i}(v,w) \\
[\alpha]^{\vee}_{\sum_{i\in I}\sigma_i}(\langle \hat{\imath}, v\rangle, \langle \hat{\jmath}, w\rangle) &\triangleq \perp\!\!\!\perp \\
[\alpha]_{\mu t.\sigma}(\mathbf{fold}\ v, \mathbf{fold}\ w) &\triangleq \alpha_{\sigma[\mu t.\sigma/t]}(v,w) \\
[\alpha]_{!_s\sigma}(!v, !w) &\triangleq (s\circ\alpha_{\sigma})(v,w).
\end{aligned}
$$

(notice that the definition of $[\alpha]^{\vee}$ is by case analysis on $\vdash^{\mathsf{v}} v, w : \sigma$). A λ-term $\mathbb{V}$-relation α is an effectful applicative simulation distance with respect to Γ *if $\alpha \leq [\alpha]$.*

The clause for $\sigma \multimap \tau$ generalises the usual applicative clause, whereas the clause for $!_s\sigma$ 'scale' $\alpha^{\vee}_{\sigma}$ by s. It is easy to see that the above definition induces a map $\alpha \mapsto [\alpha]$ on the complete lattice of closed λ-term $\mathbb{V}$-relations. Moreover, such map is monotone since both Γ and CBEs are.

Definition 76. *Define* effectful applicative similarity distance with respect to Γ, *denoted as δ, as the greatest fixed point of $\alpha \mapsto [\alpha]$. That is, δ is the greatest (closed) λ-term $\mathbb{V}$-relation satisfying the equation $\alpha = [\alpha]$ (such greatest solution exists by the Knaster-Tarski Theorem).*

We apply notational convention of Remark 9 and simply refer to applicative simulation (resp. similarity) distance in place of effectful applicative simulation (resp. similarity) distance with respect to Γ. Applicative similarity distance comes with an associated coinduction principle: for any closed λ-term $\mathbb{V}$-relation α, if $\alpha \leq [\alpha]$, then $\alpha \leq \delta$. Symbolically:

$$\frac{\alpha \leq [\alpha]}{\alpha \leq \delta}\ (\delta\text{-coind.})$$

Example 68. Instantiating Definition 76 with $\widetilde{\mathbb{DM}}$, we obtain the quantitative refinement of probabilistic applicative similarity for P-Fuzz. In particular, for two computations $e, f \in \Lambda_{\sigma}$, $\delta(e,f)$ is (for readability we omit subscripts):

$$\min_{\omega\in\Omega([\![e]\!],[\![f]\!])} \sum_{v,w\in\mathcal{V}_\circ} \omega(just\ v, just\ w)\cdot\delta^{\vee}(v,w) + \sum_{v\in\mathcal{V}_\circ}\omega(just\ v,\perp)$$

where we have implicitly used the following identities:

$$
\begin{aligned}
\widetilde{\mathbb{M}}\delta^{\vee}(\perp,\perp) &= 0 \\
\widetilde{\mathbb{M}}\delta^{\vee}(just\ v,\perp) &= 1 \\
\widetilde{\mathbb{M}}\delta^{\vee}(\perp, just\ w) &= 0.
\end{aligned}
$$

We immediately notice that δ is adequate in the following sense: for all terms $e, f \in \Lambda_{\sigma}$ we have the inequality

$$\sum[\![e]\!] - \sum[\![f]\!] \leq \delta^{\wedge}(e,f),$$

where $\sum[\![e]\!]$ is the probability of convergence of e, i.e. $\sum_{v\in\mathcal{V}_\circ}[\![e]\!](just\ v)$, and subtraction is actually truncated subtraction.

Lemma 39. *For all $\mu \in DMX$ and $\nu \in DMY$, and $\mathbb{I}$-relation $\alpha : X \nrightarrow Y$, we have:*

$$\sum \mu - \sum \nu \leq \widetilde{\mathbb{DM}}\alpha(\mu, \nu),$$

where $-$ denotes truncated subtraction.

Proof. By Proposition 27 we have:

$$\widetilde{\mathbb{DM}}\alpha(\mu,\nu) = \max\{\sum_{x\in X} a_x \cdot \mu(just\ x) + a_{\perp_X} \cdot \mu(\perp_X) + \sum_{y\in Y} b_y \cdot \nu(just\ y) + b_{\perp_Y} \cdot \nu(\perp_Y)\},$$

where $a_x, a_{\perp_X}, b_y, b_{\perp_Y}$ are bounded and satisfy the following constraints (already simplified according to the definition of $\widetilde{\mathbb{M}}\alpha$):

$$a_x + b_y \leq \alpha(x,y), \qquad a_{\perp_X} + b_y \leq 0, \qquad a_x + b_{\perp_Y} \leq 1, \qquad a_{\perp_X} + b_{\perp_Y} \leq 0.$$

Choosing $a_x \triangleq 1$, $b_y \triangleq -1$, $a_{\perp_X} \triangleq b_{\perp_Y} \triangleq 0$ we obtain the desired inequality. □

Let now e, f be **return** I and (**return** I) **or** Ω, respectively. We claim that $\delta^{\Lambda}(e,f) = \frac{1}{2}$. By adequacy we immediately see that $\frac{1}{2} \leq \delta^{\Lambda}(e,f)$. We prove $\delta^{\Lambda}(e,f) \leq \frac{1}{2}$. Let us consider the coupling ω defined by:

$$\omega(just\ I, just\ I) = \frac{1}{2}, \quad \omega(just\ I, \perp) = \frac{1}{2}$$

and zero for the rest. Indeed ω is a coupling of $[\![e]\!]$ and $[\![f]\!]$. Moreover, by very definition of δ and $\widetilde{\mathbb{DM}}$ we have:

$$\delta^{\Lambda}(e,f) \leq \omega(just\ I, just\ I) \cdot \delta^{\mathcal{V}}(I,I) + \omega(just\ I, \perp).$$

The right hand side of the above inequality gives exactly $\frac{1}{2}$, provided that $\delta^{\mathcal{V}}(I,I) = 0$. This indeed holds in full generality. ⊠

Proposition 30. *Applicative similarity distance δ is a reflexive and transitive λ-term $\mathbb{V}$-relation.*

Proof. The proof is by coinduction. Let us show that δ is transitive, i.e. that $\delta \cdot \delta \leq \delta$. We prove that the λ-term $\mathbb{V}$-relation $(\delta^{\Lambda} \cdot \delta^{\Lambda}, \delta^{\mathcal{V}} \cdot \delta^{\mathcal{V}})$ is an applicative simulation distance. We split the proof into five cases:

1. We show that for all terms $e, f \in \Lambda^{\sigma}_{\circ}$ we have:

$$\bigvee_{g\in\Lambda_\sigma} \delta^{\Lambda}_{\sigma}(e,g) \otimes \delta^{\Lambda}_{\sigma}(g,f) \leq \Gamma(\delta^{\mathcal{V}}_{\sigma} \cdot \delta^{\mathcal{V}}_{\sigma})([\![e]\!],[\![f]\!]).$$

   By (𝕍-rel 2) it is sufficient to prove:

$$\bigvee_{g\in\Lambda^{\sigma}_{\circ}} \delta^{\Lambda}_{\sigma}(e,g) \otimes \delta^{\Lambda}_{\sigma}(g,f) \leq \bigvee_{\mathfrak{v}\in T\mathcal{V}^{\sigma}_{\circ}} \Gamma\delta^{\mathcal{V}}_{\sigma}([\![e]\!],\mathfrak{v}) \otimes \Gamma\delta^{\mathcal{V}}_{\sigma}(\mathfrak{v},[\![f]\!]).$$

   For any $g \in \Lambda^{\sigma}_{\circ}$ instantiate $\mathfrak{v}$ as $[\![g]\!]$. Since $\delta^{\Lambda}_{\sigma}(e,g) \leq \Gamma\delta^{\mathcal{V}}_{\sigma}([\![e]\!],[\![g]\!])$ and $\delta^{\Lambda}_{\sigma}(g,f) \leq \Gamma\delta^{\mathcal{V}}_{\sigma}([\![g]\!],[\![f]\!])$, we are done by very definition of δ.

2. We prove that

$$(\delta^{\mathcal{V}}_{\sigma\multimap\tau} \cdot \delta^{\mathcal{V}}_{\sigma\multimap\tau})(v,w) \leq \bigwedge_{u\in\mathcal{V}^{\sigma}_{\circ}} (\delta^{\Lambda}_{\tau} \cdot \delta^{\Lambda}_{\tau})(vu, wu)$$

holds for all values $v, w \in \mathcal{V}_\circ^{\sigma \multimap \tau}$. For that it is sufficient to prove that for any $u \in \mathcal{V}_\circ^{\sigma}$ and for any $z \in \mathcal{V}_\circ^{\sigma \multimap \tau}$ there exists a term $e \in \Lambda_\circ^{\tau}$ such that:

$$\delta^{\mathcal{V}}_{\sigma \multimap \tau}(v, z) \otimes \delta^{\mathcal{V}}_{\sigma \multimap \tau}(z, w) \leq \delta^{\Lambda}_{\tau}(vu, e) \otimes \delta^{\Lambda}_{\tau}(e, wu).$$

By very definition of $\delta^{\mathcal{V}}_{\sigma \multimap \tau}$ we have:

$$\begin{aligned}\delta^{\mathcal{V}}_{\sigma \multimap \tau}(v, z) \otimes \delta^{\mathcal{V}}_{\sigma \multimap \tau}(z, w) &\leq \bigwedge_{u' \in \mathcal{V}_\circ^{\sigma}} \delta^{\Lambda}_{\tau}(vu', zu') \otimes \bigwedge_{u' \in \mathcal{V}_\circ^{\sigma}} \delta^{\Lambda}_{\tau}(zu', wu') \\ &\leq \delta^{\Lambda}_{\tau}(vu, zu) \otimes \delta^{\Lambda}_{\tau}(zu, wu),\end{aligned}$$

so that it is sufficient to instantiate e as zu.

3. We prove that

$$\begin{aligned}(\delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i} \cdot \delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i})(\langle \hat{\imath}, v\rangle, \langle \hat{\jmath}, u\rangle) &\leq \bot\!\!\!\bot \\ (\delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i} \cdot \delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i})(\langle \hat{\imath}, v\rangle, \langle \hat{\imath}, w\rangle) &\leq (\delta^{\mathcal{V}}_{\sigma_i} \cdot \delta^{\mathcal{V}}_{\sigma_i})(v, w)\end{aligned}$$

hold for all $v, w \in \mathcal{V}_\circ^{\sigma_i}$ and $u \in \mathcal{V}_\circ^{\sigma_j}$, with $\hat{\imath} \neq \hat{\jmath}$. We have

$$(\delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i} \cdot \delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i})(\langle \hat{\imath}, v\rangle, \langle \hat{\jmath}, u\rangle) = \bigvee_{\langle \hat{\ell}, z\rangle \in \mathcal{V}_\circ^{\sum_{i \in I} \sigma_i}} \delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i}(\langle \hat{\imath}, v\rangle, \langle \hat{\ell}, z\rangle) \otimes \delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i}(\langle \hat{\ell}, z\rangle, \langle \hat{\jmath}, w\rangle).$$

Since $\hat{\imath} \neq \hat{\jmath}$ at least one among $\hat{\imath} \neq \hat{\ell}$ and $\hat{\ell} \neq \hat{\jmath}$ holds, for any $\langle \hat{\ell}, z\rangle \in \mathcal{V}_\circ^{\sum_{i \in I} \sigma_i}$. As a consequence, by very definition of δ, the right hand side of the above inequality is equal to something of the form $\bot\!\!\!\bot \otimes a$, which is itself equal to $\bot\!\!\!\bot$. To prove the second inequality, we have to show that for any $\langle \hat{\imath}, u\rangle \in \mathcal{V}_\circ^{\sum_{i \in I} \sigma_i}$ there exists $z \in \mathcal{V}_\circ^{\sigma_i}$ such that

$$\delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i}(\langle \hat{\imath}, v\rangle, \langle \hat{\imath}, u\rangle) \otimes \delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i}(\langle \hat{\imath}, u\rangle, \langle \hat{\imath}, w\rangle) \leq \delta^{\mathcal{V}}_{\sigma_i}(v, z) \otimes \delta^{\mathcal{V}}_{\sigma_i}(z, w).$$

Notice that for a value $\langle \hat{\jmath}, u\rangle \in \mathcal{V}_\circ^{\sum_{i \in I} \sigma_i}$ with $\hat{\jmath} \neq \hat{\imath}$ we would have, by very definition of δ, $\delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i}(\langle \hat{\imath}, v\rangle, \langle \hat{\jmath}, u\rangle) = \bot\!\!\!\bot$, and thus we would be trivially done. Proving the above inequality is straightforward: simply instantiate z as u and observe that by definition of δ we have

$$\begin{aligned}\delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i}(\langle \hat{\imath}, v\rangle, \langle \hat{\imath}, u\rangle) &\leq \delta^{\mathcal{V}}_{\sigma_i}(v, u) \\ \delta^{\mathcal{V}}_{\sum_{i \in I} \sigma_i}(\langle \hat{\imath}, u\rangle, \langle \hat{\imath}, w\rangle) &\leq \delta^{\mathcal{V}}_{\sigma_i}(u, w).\end{aligned}$$

4. The case for $\mu t.\sigma$ follows the same pattern of the above one.

5. We prove:

$$(\delta^{\mathcal{V}}_{!_s \sigma} \cdot \delta^{\mathcal{V}}_{!_s \sigma})(!v, !w) \leq s \circ (\delta^{\mathcal{V}}_{\sigma} \cdot \delta^{\mathcal{V}}_{\sigma})(v, w).$$

For that we notice that for every $!u \in \mathcal{V}_\circ^{!_s \sigma}$ we have:

$$\begin{aligned}\delta^{\mathcal{V}}_{!_s \sigma}(!v, !u) \otimes \delta^{\mathcal{V}}_{!_s \sigma}(!u, !w) &\leq (s \circ \delta^{\mathcal{V}}_{\sigma})(v, u) \otimes (s \circ \delta^{\mathcal{V}}_{\sigma})(u, w) \\ &\leq ((s \circ \delta^{\mathcal{V}}_{\sigma}) \cdot (s \circ \delta^{\mathcal{V}}_{\sigma}))(v, w) \\ &\leq s \circ (\delta^{\mathcal{V}}_{\sigma} \cdot \delta^{\mathcal{V}}_{\sigma})(v, w).\end{aligned}$$

□

In light of Example 66 we can look at the kernel of δ and recover effectful applicative similarity of Chapter 5 (suitably generalised to $\mathbb{V}$-Fuzz).

Proposition 31. *Define applicative Δ_Γ-similarity $\preceq$ by instantiating Definition 75 with the 2-relator Δ_Γ and replacing the clause for types of the form $!_s\sigma$ as follows:*

$$!v \ \mathcal{R}_{!_s\sigma} \ !w \implies (\varphi \cdot s \cdot \psi) \circ \mathcal{R}_\sigma(v, w).$$

Then the kernel $\varphi \circ \delta$ of δ coincide with $\preceq$.

Proof. The proof is by coinduction. We start proving that $\varphi \circ \delta$ is an applicative Δ_Γ-simulation. Since $\delta^\Lambda_\sigma(e, f) \leq \Gamma\delta^{\mathcal{V}}_\sigma([\![e]\!], [\![f]\!])$ holds for all terms $e, f \in \Lambda^\sigma_\circ$, we can apply Lemma 38 and infer the inequality $\varphi \circ \delta^\Lambda_\sigma(e, f) \leq \Delta_\Gamma(\varphi \circ \delta^{\mathcal{V}}_\sigma)([\![e]\!], [\![f]\!])$. Let us now move to the value clauses.

1. We prove that for all values $v, w \in \mathcal{V}^{\sigma \multimap \tau}_\circ$ we have:

$$\varphi \circ \delta^{\mathcal{V}}_{\sigma \multimap \tau}(v, w) \leq \bigwedge_{u \in \mathcal{V}^\sigma_\circ} \varphi \circ \delta^\Lambda_\tau(vu, wu).$$

   Suppose $\varphi \circ \delta^{\mathcal{V}}_{\sigma \multimap \tau}(v, w) = \mathsf{true}$, so that $\delta^{\mathcal{V}}_{\sigma \multimap \tau}(v, w) = k$. We show that $\varphi \circ \delta^\Lambda_\tau(vu, wu) = \mathsf{true}$ holds for any $u \in \mathcal{V}^\sigma_\circ$. By very definition of applicative similarity distance, $\delta^{\mathcal{V}}_{\sigma \multimap \tau}(v, w) = k$ implies $\bigwedge_{u \in \mathcal{V}^\sigma_\circ} \delta^\Lambda_\tau(vu, wu) = k$. Since V is integral (i.e. $k = \top$), we must have $\delta^\Lambda_\tau(vu, wu) = k$ (and thus $\varphi \circ \delta^\Lambda_\tau(vu, wu) = \mathsf{true}$) for any $u \in \mathcal{V}^\sigma_\circ$.
2. Clauses for sum and recursive types are straightforward.
3. We show that for all values $!v, !w \in \mathcal{V}^{!_s\sigma}_\circ$, $\varphi \circ \delta^{\mathcal{V}}_{!_s\sigma}(!v, !w) = \mathsf{true}$ implies $(\varphi \cdot s \cdot \psi) \circ (\varphi \circ \delta^{\mathcal{V}}_\sigma)(v, w) = \mathsf{true}$. By algebra of CBFs we have:

$$\begin{aligned}(\varphi \cdot s \cdot \psi) \circ (\varphi \circ \delta^{\mathcal{V}}_\sigma) &= (\varphi \cdot s \cdot \psi \cdot \varphi) \circ \delta^{\mathcal{V}}_\sigma \\ &= (\varphi \cdot s) \circ \delta^{\mathcal{V}}_\sigma \\ &= \varphi \circ (s \circ \delta^{\mathcal{V}}_\sigma).\end{aligned}$$

   Since $\varphi \circ \delta^{\mathcal{V}}_{!_s\sigma}(!v, !w) = \mathsf{true}$, and thus $\delta^{\mathcal{V}}_{!_s\sigma}(!v, !w) = k$, by very definition of δ we infer $s \circ \delta^{\mathcal{V}}_\sigma(v, w) = k$. We conclude $(\varphi \circ (s \circ \delta^{\mathcal{V}}_\sigma))(v, w) = \mathsf{true}$.

We now prove by coinduction $(\psi \circ \preceq) \leq \delta$, from which follows $((\varphi \cdot \psi) \circ \preceq) \subseteq (\varphi \circ \delta)$ and thus $\preceq \subseteq (\varphi \circ \delta)$. The clause for terms directly follows from Lemma 38. The clauses for values follow the same structure of the previous part of the proof. We show the case for values of type $!_s\sigma$. Suppose $\psi \circ \preceq^{\mathcal{V}}_{!_s\sigma}(!v, !w) = k$ to hold (otherwise we are trivially done), meaning that $!v \preceq^{\mathcal{V}}_{!_s\sigma} !w$ holds as well. As a consequence, we have $((\varphi \cdot s \cdot \psi) \circ \preceq^{\mathcal{V}}_\sigma)(v, w) = \mathsf{true}$, and thus $s \circ (\psi \circ \preceq^{\mathcal{V}}_\sigma)(v, w) = k$. □

Remark 21. Notice that if $\mathcal{R}_\sigma(v, w)$ holds, then so does $(\varphi \cdot s \cdot \psi) \circ \mathcal{R}_\sigma(v, w)$, but the vice versa is not true, in general. For instance, taking $s \triangleq 0$ we see that

$$(\varphi \cdot 0 \cdot \psi) \circ \mathcal{R}_\sigma(v, w) = \varphi(0(\psi(\mathsf{false}))) = \varphi(0 \cdot \infty) = \varphi(0) = \mathsf{true},$$

which essentially means we identify distinguishable values if they are not used. Nonetheless, the reader should notice that the encoding of a 'standard' λ-calculus Λ in $\mathbb{V}$-Fuzz can be obtained via the usual encoding of Λ in its linear refinement (Maraist et al., 1999) which corresponds to the fragment of $\mathbb{V}$-Fuzz based on CBEs 1 and ∞, thus avoiding the above undesired result.

In order to make applicative similarity distance a useful tool for reasoning about program distance, we need to prove it to be compositional. Formally, that amounts to prove that applicative similarity distance is compatible.

12.2 Howe's Method

In order to prove compatibility of applicative similarity distance we design a refinement of Howe's technique as presented in Section 5.4, taking inspiration from (Crubillé & Dal Lago, 2015). We start by defining the notion of *Howe's extension*, a construction extending a λ-term $\mathbb{V}$-open relation to a compatible and substitutive λ-term $\mathbb{V}$-relation.

Definition 77 (Howe extension (1))**.** *The* Howe extension α^H *of an open λ-term $\mathbb{V}$-relation α is defined as the least solution to the equation $\beta = \alpha \cdot \widehat{\beta}$.*

It is easy to see that compatible refinement $\widehat{-}$ is monotone, and thus so is the map Φ_α defined by $\Phi_\alpha(\beta) \triangleq \alpha \cdot \widehat{\beta}$. As a consequence, we can define α^H as the least fixed point of Φ_α. Since open extension $-^o$ is monotone as well, we can define the Howe's extension of a closed λ-term $\mathbb{V}$-relation α as $(\alpha^o)^H$. It is also useful to spell out the above definition concretely.

Definition 78 (Howe extension (2))**.** *The* Howe extension α^H *of an open λ-term $\mathbb{V}$-relation α is defined by:*

$$(\Gamma \vdash \alpha^H(e,f) : \sigma) \triangleq \bigvee \{a \mid \Gamma \models^\wedge a \leq \alpha^H(e,f) : \sigma\},$$
$$(\Gamma \vdash^\vee \alpha^H(v,w) : \sigma) \triangleq \bigvee \{a \mid \Gamma \models^\vee a \leq \alpha^H(v,w) : \sigma\},$$

where judgments $\Gamma \models^\wedge a \leq \alpha^H(e,f) : \sigma$ and $\Gamma \models^\vee a \leq \alpha^H(v,w) : \sigma$ are inductively defined for $a \in \mathsf{V}$, $\Gamma \vdash^\wedge e, f : \sigma$, and $\Gamma \vdash^\vee v, w : \sigma$ by rules in Figure 12.3.

The next lemma is useful for proving properties of Howe's extension. It states that α^H attains its value via the rules in Figure 12.3.

Lemma 40. *The following hold:*

1. *Given well-typed values $\Gamma \vdash^\vee v, w : \sigma$, let*

   $$A \triangleq \{a \mid \Gamma \models^\vee a \leq \alpha^H(v,w) : \sigma\}$$

   be non-empty. Then $\Gamma \models^\vee \bigvee A \leq \alpha^H(v,w)$ is derivable.

2. *Given well-typed terms $\Gamma \vdash^\wedge e, f : \sigma$, let*

   $$A \triangleq \{a \mid \Gamma \models^c a \leq \alpha^H(e,f) : \sigma\}$$

   be non-empty. Then $\Gamma \models^c \bigvee A \leq \alpha^H(e,f)$ is derivable.

Proof sketch. We simultaneously prove statements 1 and 2 by induction on (v, e). We show a couple of cases as illustrative examples:

1. Suppose

   $$A \triangleq \{a \mid \Gamma \models^\vee a \leq \alpha^H(x,w) : \sigma\}$$

   to be non-empty. If the judgment $\Gamma \models^\vee a \leq \alpha^H(x,w) : \sigma$ is provable, then it must be the conclusion of an instance of rule (H-var) from the premise:

   $$a \leq (\Delta, x :_s \sigma \vdash^\vee \alpha(x,w) : \sigma)$$

   so that $\Gamma = \Delta, x :_s \sigma$. As a consequence, we see that the set A is just $\{a \mid a \leq (\Delta, x :_s \sigma \vdash^\vee \alpha(x,w) : \sigma)\}$. In particular, we have $\Delta, x :_s \sigma \vdash^\vee \alpha(x,w) : \sigma = \bigvee A \in A$.

$$\frac{a \le \Gamma, x :_s \sigma \vdash^{\vee} \alpha(x, w) : \sigma}{\Gamma, x :_s \sigma \models^{\wedge} a \le \alpha^H(x, w) : \sigma} \text{ (H-var)}$$

$$\frac{\Gamma, x :_1 \sigma \models^{\wedge} a \le \alpha^H(e, g) : \tau \quad c \le \Gamma \vdash \alpha(\lambda x.g, f) : \sigma \multimap \tau}{\Gamma \models^{\wedge} a \otimes c \le \alpha^H(\lambda x.e, f) : \sigma \multimap \tau} \text{ (H-abs)}$$

$$\frac{\Gamma \models^{\wedge} a \le \alpha^H(v, v') : \sigma \multimap \tau \quad \Delta \models^{\wedge} b \le \alpha^H(w, w') : \sigma \quad c \le \Gamma \otimes \Delta \vdash \alpha(v'w', f) : \tau}{\Gamma \otimes \Delta \models^{\wedge} a \otimes b \otimes c \le \alpha^H(vw, f) : \tau} \text{ (H-app)}$$

$$\frac{\Gamma \models^{\vee} a \le \alpha^H(v, w) : \sigma_i \quad b \le \Gamma \vdash^{\vee} \alpha(\langle \hat{\imath}, w \rangle, u) : \sum_{i \in I} \sigma_i}{\Gamma \models^{\vee} a \otimes b \le \alpha(\langle \hat{\imath}, v \rangle, u) : \sum_{i \in I} \sigma_i} \text{ (H-inj)}$$

$$\frac{\Gamma \models^{\vee} a \le \alpha^H(\langle \hat{\imath}, v \rangle, \langle \hat{\imath}, w \rangle) : \sum_{i \in I} \sigma_i \qquad \begin{array}{c} c \le s \cdot \Gamma \otimes \Delta \vdash^{\wedge} \alpha(\mathbf{case}\ \langle \hat{\imath}, w \rangle\ \mathbf{of}\ \{\langle i, x \rangle \to f_i\}, g) : \tau \\ \forall i \in I.\ \Delta, x :_s \sigma_i \models^{\wedge} b_i \le \alpha^H(e_i, f_i) : \tau \end{array}}{s \cdot \Gamma \otimes \Delta \models^{\wedge} s(a) \otimes b_{\hat{\imath}} \otimes c \le \alpha^H(\mathbf{case}\ \langle \hat{\imath}, v \rangle\ \mathbf{of}\ \{\langle i, x \rangle \to e_i\}, g) : \tau} \text{ (H-sum-cases)}$$

$$\frac{\Gamma \models^{\wedge} a \le \alpha^H(v, w) : \sigma \quad c \le \Gamma \vdash \alpha(\mathbf{return}\ w, f) : \sigma}{\Gamma \models^{\wedge} a \otimes c \le \alpha^H(\mathbf{return}\ v, f) : \sigma} \text{ (H-val)}$$

$$\frac{\Gamma \models^{\wedge} a \le \alpha^H(e, g) : \sigma \quad \Delta, x :_s \sigma \models^{\wedge} b \le \alpha^H(e', g') : \tau \quad c \le (s \wedge 1) \cdot \Gamma \otimes \Delta \vdash \alpha(\mathbf{let}\ x = g\ \mathbf{in}\ g', f) : \tau}{(s \wedge 1) \cdot \Gamma \otimes \Delta \models^{\wedge} (s \wedge 1)(a) \otimes b \otimes c \le \alpha^H(\mathbf{let}\ x = e\ \mathbf{in}\ e', f) : \tau} \text{ (H-let)}$$

$$\frac{\Gamma \models^{\wedge} a \le \alpha^H(v, w) : \sigma \quad c \le s \cdot \Gamma \vdash \alpha(!w, z) : !_s\sigma}{s \cdot \Gamma \models^{\wedge} s(a) \otimes c \le \alpha^H(!v, z) : !_s\sigma} \text{ (H-bang)}$$

$$\frac{\Gamma \models^{\wedge} a \le \alpha^H(v, w) : !_r\sigma \qquad \begin{array}{c} c \le s \cdot \Gamma \otimes \Delta \vdash \alpha(\mathbf{case}\ w\ \mathbf{of}\ \{!x \to g\}, f) : \tau \\ \Delta, x :_{s \cdot r} \sigma \models^{\wedge} b \le \alpha^H(e, g) : \tau \end{array}}{s \cdot \Gamma \otimes \Delta \models^{\wedge} s(a) \otimes b \otimes c \le \alpha^H(\mathbf{case}\ v\ \mathbf{of}\ \{!x \to e\}, f) : \tau} \text{ (H-bang-cases)}$$

$$\frac{\Gamma \models^{\wedge} a \le \alpha^H(v, w) : \mu t.\sigma \qquad \begin{array}{c} c \le s \cdot \Gamma \otimes \Delta \vdash \alpha(\mathbf{case}\ w\ \mathbf{of}\ \{\mathbf{fold}\ x \to g\}, f) : \tau \\ \Delta, x :_s \sigma[\mu t.\sigma/t] \models^{\wedge} b \le \alpha^H(e, g) : \tau \end{array}}{s \cdot \Gamma \otimes \Delta \models^{\wedge} s(a) \otimes b \otimes c \le \alpha^H(\mathbf{case}\ v\ \mathbf{of}\ \{\mathbf{fold}\ x \to e\}, f) : \tau} \text{ (H-fold-cases)}$$

$$\frac{\Gamma \models^{\wedge} a \le \alpha^H(v, w) : \sigma[\mu t.\sigma/t] \quad c \le \Gamma \vdash \alpha(\mathbf{fold}\ w, z) : \mu t.\sigma}{\Gamma \models^{\wedge} a \otimes c \le \alpha^H(\mathbf{fold}\ v, z) : \mu t.\sigma} \text{ (H-fold)}$$

$$\frac{\forall i \le n.\ \Gamma_i \models^{\wedge} a_i \le \alpha^H(e_i, g_i) : \sigma \quad c \le [\![\mathbf{op}]\!]_{\vee}(\Gamma_1 \ldots, \Gamma_n) \vdash \alpha(\mathbf{op}(g_1, \ldots, g_n), f) : \sigma}{[\![\mathbf{op}]\!]_{\vee}(\Gamma_1, \ldots, \Gamma_n) \models^{\wedge} [\![\mathbf{op}]\!]_{\vee}(a_1, \ldots, a_n) \otimes c \le \alpha^H(\mathbf{op}(e_1, \ldots, e_n), f) : \sigma} \text{ (H-op)}$$

Figure 12.3: Howe extension of α.

2. Suppose
$$A \triangleq \{a \mid \Gamma \models^{\wedge} a \leq \alpha^H(\textbf{let } x = e \textbf{ in } f, g) : \tau\}$$
to be non-empty. That means there exists $a \in \mathsf{V}$ such that $\Gamma \models^{\wedge} a \leq \alpha^H(\textbf{let } x = e \textbf{ in } f, g) : \tau$ is derivable. The latter judgment must be the conclusion of an instance of rule (H-let) from premises:
$$\begin{aligned}
&\Sigma \models^{\wedge} b \leq \alpha^H(e, e') : \sigma \\
&\Delta, x :_s \sigma \models^{\wedge} c \leq \alpha^H(f, f') : \tau \\
&d \leq (s \wedge 1) \cdot \Sigma \otimes \Delta \vdash \alpha(\textbf{let } x = e' \textbf{ in } f', g) : \tau
\end{aligned}$$
so that $\Gamma = (s \wedge 1) \cdot \Sigma \otimes \Delta$ and $a = (s \wedge 1)(b) \otimes c \otimes d$. In particular, the sets
$$\begin{aligned}
B &= \{b \mid \Sigma \models^{\wedge} b \leq \alpha^H(e, e') : \sigma\} \\
C &= \{c \mid \Delta, x :_s \sigma \models^{\wedge} c \leq \alpha^H(f, f') : \tau\}
\end{aligned}$$
are non-empty. By induction hypothesis we have $\bigvee B \in B$ and $\bigvee C \in C$. Let $\underline{d} = (s \wedge 1) \cdot \Sigma \otimes \Delta \vdash \alpha(\textbf{let } x = e' \textbf{ in } f', g) : \tau$. We can now apply rule (H-let) obtaining $(s \wedge 1)\Big(\bigvee B\Big) \otimes \Big(\bigvee C\Big) \otimes \underline{d} \in A$. To see that the latter is actually $\bigvee A$ it is sufficient to show that for any $a \in A$ we have $a \leq (s \wedge 1)\Big(\bigvee B\Big) \otimes \Big(\bigvee C\Big) \otimes \underline{d}$. But any $a \in A$ (with $a \neq \perp\!\!\!\perp$) is of the form $(s \wedge 1)(b) \otimes c \otimes d$ for $b \in B$, $c \in C$, and $d \leq \underline{d}$. We are done since both $(s \wedge 1)$ and $\otimes$ are monotone.

□

It is not hard to convince ourselves that Definition 77 and Definition 78 gives the same λ-term $\mathbb{V}$-relation. In particular, for an open λ-term $\mathbb{V}$-relation α, α^H is the least compatible λ-term $\mathbb{V}$-relation satisfying the inequality $\alpha^o \cdot \beta \leq \beta$. The following are standard results. Proofs are straightforward but tedious (they closely resemble their relational counterparts), and thus are omitted.

Lemma 41. *Let α be a reflexive and transitive closed λ-term $\mathbb{V}$-relation. Then the following hold:*
1. *α^H is reflexive.*
2. *$\alpha^o \leq \alpha^H$.*
3. *$\alpha \cdot \alpha^H \leq \alpha^H$.*
4. *α^H is compatible.*

We refer to property 1 as pseudo-transitivity. In particular, by very definition of $\mathbb{V}$-relator we also have the following inequality.
$$\Gamma\alpha \cdot \Gamma\alpha^H \leq \Gamma\, \alpha^H\,. \qquad (\Gamma \text{ pseudo-trans.})$$

Notice that Proposition 30 implies that $(\delta^o)^H$ is compatible and bigger than δ^o. Finally, we notice that the Howe extension of a λ-term $\mathbb{V}$-relation is substitutive.

Lemma 42 (Substitutivity)**.** *Let α be a value substitutive λ-term $\mathbb{V}$-preorder. For all values, $\Gamma, x :_s \sigma \vdash^{\mathsf{v}} u, z : \tau$ and $\vdash v, w : \sigma$, and terms $\Gamma, x :_s \sigma \vdash e, f : \tau$, let $\underline{a} \triangleq \vdash^{\mathsf{v}} \alpha^H(v, w) : \sigma$. Then:*
$$\begin{aligned}
(\Gamma, x :_s \sigma \vdash^{\mathsf{v}} \alpha^H(u, z) : \tau) \otimes s(\underline{a}) &\leq \Gamma \vdash^{\mathsf{v}} \alpha^H(u[v/x], z[w/x]) : \tau \\
(\Gamma, x :_s \sigma \vdash \alpha^H(e, f) : \tau) \otimes s(\underline{a}) &\leq \Gamma \vdash \alpha^H(e[x := v], f[x := w]) : \tau.
\end{aligned}$$

Proof. We simultaneously prove the following statements.

1. For any $a \in \mathsf{V}$ if $\Gamma, x :_s \sigma \models^{\wedge} a \leq \alpha^H(e, f) : \tau$ is derivable, then
$$a \otimes s(\underline{a}) \leq \Gamma \vdash \alpha^H(e[x := v], f[x := w]) : \tau.$$

2. For any $a \in \mathsf{V}$ if $\Gamma, x :_s \sigma \models^{\mathsf{v}} a \leq \alpha^H(u,z) : \tau$ is derivable, then

$$a \otimes s(\underline{a}) \leq \Gamma \vdash \alpha^H(u[v/x], z[w/x]) : \tau.$$

The proof is by induction on the derivation of the judgments:

$$\mathcal{J} \triangleq \Gamma, x :_s \sigma \models^{\wedge} a \leq \alpha^H(e,f) : \tau$$
$$\mathcal{J}' \triangleq \Gamma, x :_s \sigma \models^{\mathsf{v}} a \leq \alpha^H(u,z) : \tau.$$

1. Suppose $\mathcal{J}'$ has been inferred via an instance of rule (H-var). We have two subcases to consider.

   1.1 $\mathcal{J}'$ has been inferred via an instance of rule (H-var) from premises:

   $$\frac{a \leq \Gamma, x :_s \sigma \vdash^{\mathsf{v}} \alpha(x,u) : \sigma}{\Gamma, x :_s \sigma \models^{\mathsf{v}} a \leq \alpha^H(x,u) : \sigma} \text{ (H-var)},$$

   so that $s \leq 1$ and $\mathcal{J}'$ is $\Gamma, x :_s \sigma \models^{\mathsf{v}} a \leq \alpha^H(x,u) : \sigma$. We have to prove $a \otimes s \circ (\vdash^{\mathsf{v}} \alpha^H(v,w)) \leq \Gamma \vdash^{\mathsf{v}} \alpha^H(v, u[w/x]) : \sigma$. Since α is value substitutive, from $\Gamma, x :_s \sigma \models^{\mathsf{v}} a \leq \alpha^H(x,u) : \sigma$ we infer $a \leq \Gamma \vdash^{\mathsf{v}} \alpha(w, u[w/x]) : \sigma$. Moreover, since α^H is an open λ-term $\mathbb{V}$-relation (and thus closed under weakening), we have $\vdash^{\mathsf{v}} \alpha^H(v,w) : \sigma \leq \Gamma \vdash^{\mathsf{v}} \alpha^H(v,w) : \sigma$. We can now conclude the thesis as follows:

   $$\begin{aligned} a \otimes s(\underline{a}) &\leq (\Gamma \vdash^{\mathsf{v}} \alpha^H(v,w) : \sigma) \otimes s \circ (\Gamma \vdash^{\mathsf{v}} \alpha(w, u[w/x]) : \sigma) \\ &\leq (\Gamma \vdash^{\mathsf{v}} \alpha^H(v,w) : \sigma) \otimes (\Gamma \vdash^{\mathsf{v}} \alpha(w, u[w/x]) : \sigma) \\ &\quad [\text{ since } s \leq 1 \text{ }] \\ &\leq \Gamma \vdash^{\mathsf{v}} \alpha^H(v, u[w/x]) : \sigma \\ &\quad [\text{ by pseudo-transitivity }]. \end{aligned}$$

   1.2 $\mathcal{J}'$ has been inferred via an instance of rule (H-var) from premises:

   $$\frac{a \leq \Gamma, y :_r \tau, x :_s \sigma \vdash^{\mathsf{v}} \alpha(y,u) : \tau}{\Gamma, y :_r \tau, x :_s \sigma \models^{\mathsf{v}} a \leq \alpha^H(y,u) : \tau} \text{ (H-var)}$$

   so that $\mathcal{J}'$ is $\Gamma, y :_r \tau, x :_s \sigma \models^{\mathsf{v}} a \leq \alpha^H(y,u) : \tau$. We have to prove $a \otimes s \circ (\emptyset \vdash^{\mathsf{v}} \alpha^H(v,w)) \leq \Gamma, y :_r \tau \vdash^{\mathsf{v}} \alpha^H(y, u[w/x]) : \tau$. As V is integral and α is value-substitutive, we have:

   $$a \otimes s \circ (\emptyset \vdash^{\mathsf{v}} \alpha^H(v,w)) \leq a \leq \Gamma, y :_r \tau \vdash^{\mathsf{v}} \alpha(y, u[w/x]).$$

   Since $\alpha \leq \alpha^H$ we are done.

2. Suppose $\mathcal{J}$ has been inferred via an instance of rule (H-let) from premises:

   $$\Gamma, x :_s \sigma \models^{\wedge} a \leq \alpha^H(e, e') : \sigma' \tag{12.1}$$
   $$\Delta, x :_r \sigma, y :_p \sigma' \models^{\wedge} b \leq \alpha^H(f, f') : \tau \tag{12.2}$$
   $$c \leq (p \wedge 1) \cdot (\Gamma, x :_s \sigma) \otimes (\Delta, x :_r \sigma) \vdash \alpha^H(\mathbf{let}\ y = e'\ \mathbf{in}\ f', g) : \tau. \tag{12.3}$$

   so that J is:

   $$(p \wedge 1) \cdot \Gamma \otimes \Delta, x :_{(p \wedge 1) \cdot s \otimes r} \sigma \models^{\wedge} (p \wedge 1)(a) \otimes b \otimes c \leq \alpha^H(\mathbf{let}\ y = e\ \mathbf{in}\ f, g) : \tau.$$

We have to prove:

$$(p \wedge 1)(s(\underline{a})) \otimes r(\underline{a}) \otimes (p \wedge 1)(a) \otimes b \otimes c \leq (p \wedge 1) \cdot \Gamma \otimes \Delta \vdash \alpha^H(\mathbf{let}\ y = e[x := v]\ \mathbf{in}\ f[x := v], g[x := w]) : \tau.$$

We apply the induction hypothesis on (12.1) and (12.2) obtaining:

$$s(\underline{a}) \otimes a \leq \Gamma \vdash \alpha^H(e[x := v], e'[x := w]) : \sigma' \tag{12.4}$$

$$r(\underline{a}) \otimes b \leq \Delta, y :_p \sigma' \vdash \alpha^H(f[x := v], f[x := w]) : \tau. \tag{12.5}$$

From (12.4) and (12.5) by compatibility of α^H (and lax equations of change of base functors) we infer:

$$\begin{aligned}(p \wedge 1)(s(\underline{a})) \otimes (p \wedge 1)(a) \otimes r(\underline{a}) \otimes b &\leq (p \wedge 1) \cdot \Gamma \otimes \Delta \\ &\vdash \alpha^H(\mathbf{let}\ y = e[x := v]\ \mathbf{in}\ f[x := a], \mathbf{let}\ y = e'[x := w]\ \mathbf{in}\ f'[x := w]) : \tau.\end{aligned} \tag{12.6}$$

Finally, since α is value-substitutive, from (12.3) we obtain:

$$c \leq (p \wedge 1) \cdot \Gamma \otimes \Delta \vdash \alpha^H(\mathbf{let}\ y = e'[x := w]\ \mathbf{in}\ f'[x := w], g) : \tau$$

and thus conclude the thesis from the latter and (12.6) by pseudo-transitivity.

3. Suppose $\mathcal{J}$ has been inferred via an instance of rule (H-op) from premises (as usual we write $\vec{x_i}$ for items $x_1, \ldots, x_n$):

$$\forall i.\ \Gamma_i, x :_{s_i} \sigma \models^{\wedge} a_i \leq \alpha^H(e_i, e_i') : \tau \tag{12.7}$$

$$b \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{\Gamma_i}), x :_{[\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{s_i})} \sigma \vdash \alpha(\mathbf{op}(\vec{e_i'}), f) : \tau \tag{12.8}$$

so that J is

$$[\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{\Gamma_i}), x :_{[\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{s_i})} \sigma \models^{\wedge} [\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{a_i}) \otimes b \leq \alpha^H(\mathbf{op}(\vec{e_i}), f) : \tau.$$

We have to prove

$$[\![\mathbf{op}]\!]_{\mathsf{V}}(\overrightarrow{s_i(\underline{a})}) \otimes [\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{a_i}) \otimes b \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{\Gamma_i}) \vdash \alpha^H(\overrightarrow{e_i[x := v]}, f[x := w]) : \tau.$$

We apply the induction hypothesis on (12.7) obtaining:

$$\forall i.\ s(\underline{a}) \otimes a_i \leq \Gamma_i \vdash \alpha^H(e_i[x := v], e_i'[x := w]) : \tau. \tag{12.9}$$

Monotonicity of $[\![\mathbf{op}]\!]_{\mathsf{V}}$ on (12.9) followed by compatibility gives:

$$[\![\mathbf{op}]\!]_{\mathsf{V}}(\overrightarrow{s_i(\underline{a}) \otimes a_i}) \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{\Gamma_i}) \vdash \alpha^H(\mathbf{op}(\overrightarrow{e_i[x := v]}), \mathbf{op}(\overrightarrow{e_i'[x := w]})). \tag{12.10}$$

Finally, as α is value-substitutive, from (12.8) we obtain:

$$b \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{\Gamma_i}), \vdash \alpha(\mathbf{op}(\overrightarrow{e_i'[x := w]})f[x := w]) : \tau.$$

The latter together with (12.5) implies

$$[\![\mathbf{op}]\!]_{\mathsf{V}}(\overrightarrow{s_i(\underline{a}) \otimes a_i}) \otimes b \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{\Gamma_i}) \vdash \alpha^H(\mathbf{op}(\overrightarrow{e_i[x := v]}), f[x := w])$$

by pseudo-transitivity. We conclude the thesis as Definition 63 entails:

$$[\![\mathbf{op}]\!]_{\mathsf{V}}(\overrightarrow{s_i(\underline{a})}) \otimes [\![\mathbf{op}]\!]_{\mathsf{V}}(\vec{a_i}) \leq [\![\mathbf{op}]\!]_{\mathsf{V}}(\overrightarrow{s_i(\underline{a}) \otimes a_i}).$$

The remaining cases follow the same pattern. □

Notice that the open extension of any closed λ-term $\mathbb{V}$-relation is value-substitutive. We can now prove the main result of Howe's method, the so-called *Key Lemma*. The latter states the Howe extension of applicative similarity distance (restricted to closed terms) is an applicative simulation distance. By coinduction, we conclude $\delta = (\delta^H)^c$ (the latter being the closed restriction of δ^H), and thus $\delta^o = \delta^H$, meaning that the former is compatible.

Lemma 43 (Key Lemma)**.** *Recall that we assume Γ to be Σ-continuous. Let α be a reflexive and transitive applicative simulation distance. Then the Howe extension of α restricted to closed terms/values in an applicative simulation distance.*

Proof. Let us write α^H for the Howe extension of α restricted to closed computations/values. It is easy to see that α^H satisfies the simulation clauses for values. For instance, we prove the inequality $\alpha^H_{!_s\sigma}(!v, !w) \leq s \circ \alpha^H_\sigma(v, w)$, where for readability we omit values superscript in α and α^H. It is sufficient to show that for any $a \in \mathsf{V}$ such that the judgment $\mathcal{J} \triangleq (\models^\wedge a \leq \alpha^H(!v, !w) : !_s\sigma)$ is derivable, the inequality $a \leq s \circ \alpha^H_\sigma(v, w)$ holds. The judgment $\mathcal{J}$ must have been inferred via an instance of rule (H-bang), so that without loss of generality we can assume $a = s(b) \otimes \alpha_{!_s\sigma}(!u, !w)$, with $\models^\wedge b \leq \alpha^H(v, u) : \sigma$ derivable, for some value u. We conclude the thesis as follows:

$$
\begin{aligned}
a &\leq s \circ \alpha^H_\sigma(v, u) \otimes \alpha_{!_s\sigma}(!u, !w) \\
&\leq s \circ \alpha^H_\sigma(v, u) \otimes s \circ \alpha_\sigma(u, w) \\
&\quad [\alpha \text{ is an applicative simulation distance}] \\
&\leq s \circ (\alpha^H_\sigma(v, u) \otimes \alpha_\sigma(u, w)) \\
&\leq s \circ (\alpha_\sigma \cdot \alpha^H_\sigma)(v, w) \\
&\leq s \circ \alpha^H_\sigma(v, w). \\
&\quad [\text{pseudo-transitivity}]
\end{aligned}
$$

The crucial part of the proof is to show that α^H satisfies the clause for computations. We prove that for any $n \geq 0$, and all computations $e, f \in \Lambda^\sigma_\circ$, we have:

$$(\alpha^H)^\wedge_\sigma(e, f) \leq \Gamma(\alpha^H)^\mathcal{V}_\sigma([\![e]\!]_n, [\![f]\!]).$$

Since Γ is inductive the above inequality gives the thesis as follows:

$$
\begin{aligned}
(\alpha^H_\sigma)^\wedge(e, f) &\leq \bigwedge_n \Gamma(\alpha^H_\sigma)^\mathcal{V}([\![e]\!]_n, [\![f]\!]) \\
&\leq \Gamma(\alpha^H_\sigma)^\mathcal{V}(\bigsqcup_n [\![e]\!]_n, [\![f]\!]) \\
&= \Gamma(\alpha^H_\sigma)^\mathcal{V}([\![e]\!], [\![f]\!]).
\end{aligned}
$$

The proof is by induction on n with a case analysis on the term structure in the inductive case. For readability we simply write α in place of $\alpha^\wedge$ and $\alpha^\mathcal{V}$. Moreover, to avoid confusion we remark that we explicitly distinguish between (ordinary) Kleisli extension and strong Kleisli extension. Given a monoidal category $\langle \mathbb{C}, I, \otimes \rangle$, recall that we denote by $f^* : Z \otimes TX \to TY$ the *strong* Kleisli extension of $f : Z \otimes X \to TY$ and by $g^\dagger : TX \to TY$ the Kleisli extension of $g : X \to TY$. Moreover, the latter can be defined in terms of the former as $g^\dagger \triangleq (g \cdot \lambda_X)^* \cdot \lambda^{-1}_{TX}$, where $\lambda_X : I \otimes X \xrightarrow{\cong} X$ is the natural isomorphism given by the monoidal structure of $\mathbb{C}$. Notice also that $g^\dagger \cdot \lambda_{TX} = (g \cdot \lambda_X)^*$. We now enter the details of the induction.

1. We have to prove:
$$\alpha^H_\sigma(e, f) \leq \Gamma\alpha^H_\sigma([\![e]\!]_0, [\![f]\!]).$$
Since Γ is inductive and $[\![e]\!]_0 = \bot_{\mathcal{V}^\sigma_\circ}$, it is sufficient to prove $\alpha^H_\sigma(e, f) \leq k$. Because the quantale is integral the latter trivially holds.

2. We have to prove:
$$\alpha^H_\sigma(\mathbf{return}\ v, w) \leq \Gamma\alpha^H_\sigma([\![\mathbf{return}\ v]\!]_{n+1}, [\![w]\!]).$$
Since $[\![\mathbf{return}\ v]\!]_{n+1} = \eta(v)$, it is sufficient to prove that for any a such that the judgment $\models^\wedge a \leq \alpha^H(\mathbf{return}\ v, w) : \sigma$ is derivable, $a \leq \Gamma\alpha^H_\sigma(\eta(v), [\![w]\!])$ holds. Suppose $\models^\wedge a \leq \alpha^H(\mathbf{return}\ v, w) : \sigma$ to be derivable. The latter must have been inferred via an instance of rule (H-val) from premises:
$$\models^\wedge b \leq \alpha^H(v, v') : \sigma \tag{12.11}$$
$$c \leq \alpha_\sigma(\mathbf{return}\ v', w). \tag{12.12}$$
In particular, we have $b \leq \alpha^H_\sigma(v, v')$ and thus, by condition (Lax unit), $b \leq \Gamma\alpha^H_\sigma(\eta(v), \eta(v'))$. From, (12.12) we infer, by very definition of applicative simulation distance, $c \leq \Gamma\alpha_\sigma(\eta(v'), [\![w]\!])$, and thus $b \otimes c \leq \Gamma\alpha^H_\sigma(\eta(v), \eta(v')) \otimes \Gamma\alpha_\sigma(\eta(v'), [\![w]\!])$. We conclude the thesis by (Γ pseudo-trans.).

3. We have to prove:
$$\alpha^H_\tau((\lambda x.e)v, f) \leq \Gamma\alpha^H_\tau([\![(\lambda x.e)v]\!]_{n+1}, [\![f]\!]).$$
As $[\![(\lambda x.e)v]\!]_{n+1} = [\![e[x := v]]\!]_n$, it is sufficient to show that for any a such that $\models^\wedge a \leq \alpha^H((\lambda x.e)v, f) : \tau$ holds, we have $a \leq \Gamma\alpha^H_\tau([\![e[x := v]]\!]_n, [\![f]\!])$. Assume $\models^\wedge a \leq \alpha^H((\lambda x.e)v, f) : \tau$. The latter must have been inferred via an instance of rule (H-app) from premises:
$$\models^\wedge b \leq \alpha^H(v, w) : \sigma \tag{12.13}$$
$$\models^\wedge c \leq \alpha^H(\lambda x.e, u) : \sigma \multimap \tau \tag{12.14}$$
$$d \leq \alpha_\tau(uw, f). \tag{12.15}$$
Let us examine premise (12.14). First of all, since u is a closed value of type $\sigma \multimap \tau$ it must be of the form $\lambda x.g$. Moreover, (12.14) must have been inferred via an instance rule rule (H-abs) from premises:
$$x :_1 \sigma \models^\wedge c_1 \leq \alpha^H(e, h) : \tau \tag{12.16}$$
$$c_2 \leq \alpha_{\sigma\multimap\tau}(\lambda x.h, \lambda x.g). \tag{12.17}$$
In particular, we have the equality $c_1 \otimes c_2 = c$. From (12.16) we deduce $c_1 \leq x :_1 \sigma \vdash \alpha^H(e, h) : \tau$, whereas from (12.13) we infer $b \leq \alpha^H_\sigma(v, w)$. We are now in position to apply Lemma 42, obtaining $c_1 \otimes b \leq \alpha^H_\tau(e[x := v], h[x := w])$. By very definition of applicative simulation distance, (12.17) implies the inequality $c_2 \leq \alpha_\tau(h[x := w], g[x := w])$. Applying pseudo-transitivity followed by the induction hypothesis we obtain:
$$c_1 \otimes c_2 \otimes b \leq \alpha^H_\tau(e[x := v], g[x := w]) \leq \Gamma\alpha^H_\tau([\![e[x := v]]\!]_n, [\![g[x := w]]\!]).$$
Finally, from (12.15), by definition of applicative simulation distance we infer
$$d \leq \Gamma\alpha_\tau([\![g[x := w]]\!], [\![f]\!])$$
(recall that $u = \lambda x.g$, so that $[\![uw]\!] = [\![g[x := w]]\!]$). We can now conclude the thesis by (Γ pseudo-trans.).

4. Cases for pattern matching against folds and sums are standard (they follow the same pattern of point 5, but are simpler).

5. We have to prove:

$$\alpha^H_\tau(\textbf{case}\ !v\ \textbf{of}\ \{!x \to e\}, f) \leq \Gamma\alpha^H_\tau([\![\textbf{case}\ !v\ \textbf{of}\ \{!x \to e\}]\!]_{n+1}, [\![f]\!]).$$

As $[\![\textbf{case}\ !v\ \textbf{of}\ \{!x \to e\}]\!]_{n+1} = [\![e[x := v]]\!]_n$, we show that for any a such that $\models^\wedge a \leq \alpha^H(\textbf{case}\ !v\ \textbf{of}\ \{!x \to e\}, f) : \tau$ is derivable, the inequality $a \leq \Gamma\alpha^H_\tau([\![e[x := v]]\!]_n, [\![f]\!])$ holds. Suppose $\models^\wedge a \leq \alpha^H(\textbf{case}\ !v\ \textbf{of}\ \{!x \to e\}, f) : \tau$. The latter must have been inferred via an instance of rule (H-bang-cases) from premises:

$$\emptyset \models^\wedge b \leq \alpha^H(!v, u) : !_s\sigma \tag{12.18}$$

$$x :_{r\cdot s} \sigma \models^\wedge c \leq \alpha^H(e, e') : \tau \tag{12.19}$$

$$d \leq \alpha_\tau(\textbf{case}\ u\ \textbf{of}\ \{!x \to e'\}, f). \tag{12.20}$$

In particular, we have $a = r(b) \otimes c \otimes d$. Let us examine premise (12.18). First of all, since u is a closed value of type $!_s\sigma$, it must be of the form $!v'$. Moreover, (12.18) must have been inferred via an instance of rule (H-bang) from premises:

$$\emptyset \models^\wedge b_1 \leq \alpha^H(v, w) : \sigma \tag{12.21}$$

$$b_2 \leq \alpha_{!_s\sigma}(!w, !v'). \tag{12.22}$$

In particular, $b = s(b_1) \otimes b_2$. From (12.22), by definition of applicative simulation distance we infer $b_2 \leq s \circ \alpha_\sigma(w, v')$. Since (12.21) implies $b_1 \leq \alpha^H_\sigma(v, w)$, we have:

$$\begin{aligned} b &= s(b_1) \otimes b_2 \\ &\leq s \circ \alpha^H_\sigma(v, w) \otimes s \circ \alpha_\sigma(w, v') \\ &\leq s \circ (\alpha^H_\sigma(v, w) \otimes \alpha_\sigma(w, v')) \\ &\leq s \circ \alpha^H_\sigma(v, v') \end{aligned}$$

where the last inequality follows by pseudo-transitivity. From (12.19) we infer the inequality $c \leq x :_{r\cdot s} \sigma \vdash \alpha^H(e, e') : \tau$. We are now in position to apply Lemma 42 obtaining:

$$(r \cdot s) \circ \alpha^H_\sigma(v, v') \otimes c \leq \alpha^H_\tau(e[x := v], e'[x := v']).$$

The latter, together with the inequality $b \leq s \circ \alpha^H_\sigma(v, v')$, implies

$$r(b) \otimes c \leq \alpha^H_\tau(e[x := v], e'[x := v']).$$

Applying the induction hypothesis we conclude:

$$r(b) \otimes c \leq \Gamma\alpha^H_\tau([\![e[x := v]]\!]_n, [\![e'[x := v']]\!]).$$

Finally, from (12.20), by definition of applicative simulation distance we infer

$$d \leq \Gamma\alpha_\tau([\![e'[x := v']]\!], [\![f]\!])$$

(recall that $u = !v'$) and thus conclude the thesis by (Γ pseudo-trans.).

6. We have to prove:

$$\alpha^H_\tau(\textbf{let}\ x = e\ \textbf{in}\ f, g) \leq \Gamma\alpha^H_\tau([\![\textbf{let}\ x = e\ \textbf{in}\ f]\!]_{n+1}, [\![g]\!]).$$

As $[\![\textbf{let}\ x = e\ \textbf{in}\ f]\!]_{n+1} = [\![f[x := \_]]\!]^\dagger_n [\![e]\!]_n$, it is sufficient to prove that for any a such that $\models^\wedge a \leq \alpha^H(\textbf{let}\ x = e\ \textbf{in}\ f, g) : \tau$ is derivable, we have $a \leq \Gamma\alpha^H_\tau([\![f[x := \_]]\!]^\dagger_n [\![e]\!]_n, [\![g]\!])$. Suppose

$\models^{\Lambda} a \leq \alpha^H(\mathbf{let}\ x = e\ \mathbf{in}\ f, g) : \tau$. The latter must have been inferred via an instance of rule (H-let) from premises:

$$\models^{\Lambda} b \leq \alpha^H(e, e') : \sigma \tag{12.23}$$

$$x :_s \sigma \models^{\Lambda} c \leq \alpha^H(f, f') : \tau \tag{12.24}$$

$$d \leq \alpha_\tau(\mathbf{let}\ x = e'\ \mathbf{in}\ f', g). \tag{12.25}$$

In particular, we have $a = (s \wedge 1)(b) \otimes c \otimes d$. We now claim to have:

$$(x :_s \sigma \vdash \alpha^H(f, f') : \tau) \otimes (s \wedge 1) \circ \alpha^H_\sigma(e, e') \leq \Gamma\alpha^H_\tau([\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!]_{n+1}, [\![\mathbf{let}\ x = e'\ \mathbf{in}\ f']\!]). \tag{12.26}$$

By very definition of Howe extension, the latter obviously entails

$$(s \wedge 1)(b) \otimes c \leq \Gamma\alpha^H_\tau([\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!]_{n+1}, [\![\mathbf{let}\ x = e'\ \mathbf{in}\ f']\!]).$$

Moreover, by definition of applicative simulation distance, (12.25) implies $d \leq \Gamma\alpha_\tau([\![\mathbf{let}\ x = e'\ \mathbf{in}\ f']\!], [\![g]\!])$, which allows to conclude the thesis by (Γ pseudo-trans.). Let us now turn to the proof of (12.25). First of all we apply the induction hypothesis on $\alpha^H_\sigma(e, e')$. By monotonicity of $s \wedge 1$ we have thus reduced the proof of (12.25) to proving the inequality:

$$(x :_s \sigma \vdash \alpha^H(f, f') : \tau) \otimes (s \wedge 1) \circ \Gamma\alpha^H_\sigma([\![e]\!]_n, [\![e']\!]) \leq \Gamma\alpha^H_\tau([\![f[x := \_]]\!]^\dagger_n [\![e]\!]_n, [\![f'[x := \_]]\!]^\dagger [\![e']\!]). \tag{12.27}$$

Consider the diagram:

$$\begin{array}{ccc} I \times T\mathcal{V}^\sigma_\circ & \xrightarrow{[\![f[x:=\_]]\!]^\dagger_n \cdot \lambda_{T\mathcal{V}^\sigma_\circ}} & T\mathcal{V}_\tau \\ \downarrow{\scriptstyle \gamma \otimes (s\wedge 1) \circ \Gamma\alpha^H_\sigma} & \leq & \downarrow{\scriptstyle \Gamma\alpha^H_\tau} \\ I \times T\mathcal{V}^\sigma_\circ & \xrightarrow[[\![f'[x:=\_]]\!]^\dagger \cdot \lambda_{T\mathcal{V}^\sigma_\circ}]{} & T\mathcal{V}^\tau_\circ \end{array} \tag{12.28}$$

where $I = \{*\}$ and $\gamma(*, *) = (x :_s \sigma \vdash \alpha^H(f, f') : \tau)$. It is easy to see that (12.27) follows from (12.28), since e.g.:

$$([\![f[x := \_]]\!]^\dagger_n \cdot \lambda_{T\mathcal{V}^\sigma_\circ})(*, [\![e]\!]_n) = [\![e[x := \_]]\!]^\dagger_n [\![e]\!]_n.$$

To prove (12.28) we first observe that by very definition of strong monad we have $[\![f[x := \_]]\!]^\dagger_n \cdot \lambda_{T\mathcal{V}^\sigma_\circ} = ([\![f[x := \_]]\!]_n \cdot \lambda_{\mathcal{V}^\sigma_\circ})^*$. We can now apply condition (*L*-Strong lax bind). As a consequence, to prove (12.28) it is sufficient to prove that for all closed values v, w of type σ, we have:

$$(x :_s \sigma \vdash \alpha^H(f, f') : \tau) \otimes (s \wedge 1) \circ \alpha^H_\sigma(v, w) \leq \Gamma\alpha^H_\tau([\![f[x := v]]\!]_n, [\![f'[x := w]]\!]).$$

By Lemma 42 and induction hypothesis we have:

$$(x :_s \sigma \vdash \alpha^H(f, f') : \tau) \otimes s \circ \alpha^H_\sigma(v, w) \leq \Gamma\alpha^H_\tau([\![f[x := v]]\!]_n, [\![f'[x := w]]\!]).$$

We conclude the thesis since $s \wedge 1 \leq s$.

7. We have to prove:

$$\alpha^H_\sigma(\mathbf{op}(e_1, \ldots, e_m), f) \leq \Gamma\alpha^H_\sigma([\![\mathbf{op}(e_1, \ldots, e_m)]\!]_{n+1}, [\![f]\!])$$

where **op** is an m-ary operation symbol in Σ. As usual, we use the notation $\vec{x_i}$ for items $x_1, \ldots, x_m$. We show that for any a such that $\models^\wedge a \leq \alpha^H(\mathbf{op}(\vec{e_i}), f) : \sigma$ is derivable, $a \leq \Gamma\alpha^H_\tau([\![\mathbf{op}(\vec{e_i})]\!]_n [\![f]\!])$ holds. Suppose to have $\models^\wedge a \leq \alpha^H(\mathbf{op}(\vec{e_i}), f) : \tau$. The latter must have been inferred via an instance of rule (H-op) from premises:

$$\forall i \leq m.\ \models^\wedge a_i \leq \alpha^H(e_i, f_i) : \sigma \tag{12.29}$$

$$b \leq \alpha_\tau(\mathbf{op}(f_1, \ldots, f_m), f). \tag{12.30}$$

In particular, we have $a = [\![\mathbf{op}]\!]_\vee(a_1, \ldots, a_m) \otimes b$. We apply the induction hypothesis on (12.29) obtaining, for each $i \leq m$, the inequality $a_i \leq \Gamma\alpha^H([\![e_i]\!]_n, [\![f_i]\!])$. By monotonicity of $[\![\mathbf{op}]\!]_\vee$ we thus infer:

$$\begin{aligned}
[\![\mathbf{op}]\!]_\vee(\vec{a_i}) &\leq [\![\mathbf{op}]\!]_\vee(\Gamma\alpha^H([\![e_1]\!]_n, [\![f_1]\!]), \ldots, \Gamma\alpha^H([\![e_m]\!]_n, [\![f_m]\!])) \\
&\leq \Gamma\alpha^H_\sigma([\![\mathbf{op}]\!]_{\mathcal{V}_\sigma}([\![e_1]\!]_n, \ldots, [\![e_m]\!]_n), [\![\mathbf{op}]\!]_{\mathcal{V}_\sigma}([\![f_1]\!], \ldots, [\![f_m]\!])) \\
&= \Gamma\alpha^H_\sigma([\![\mathbf{op}(e_1, \ldots, e_m)]\!]_{n+1}, [\![\mathbf{op}(f_1, \ldots, f_m)]\!]),
\end{aligned}$$

where the second inequality follows since Γ is Σ-compatible. We conclude the thesis from (12.30) by (Γ pseudo-trans.) and definition of applicative simulation distance.

□

Theorem 16 (Compatibility)**.** *Applicative similarity distance is compatible.*

Proof. We have to prove that δ^o is compatible. By Lemma 41 we know that $\delta^o \leq (\delta^o)^H$ and that $(\delta^o)^H$ is compatible. Therefore, to conclude the thesis it is sufficient to prove $(\delta^o)^H \leq \delta^o$. The Key Lemma implies that the restriction of $(\delta^o)^H$ on closed terms is an applicative simulation distance, and thus smaller or equal than δ. We can thus show that for all $\Gamma \vdash^\wedge e, e' : \sigma$, the inequality $\Gamma \vdash^\wedge (\delta^o)^H(e, e') : \sigma \leq \Gamma \vdash^\wedge \delta^o(e, e') : \sigma$ holds. In fact, since $(\delta^o)^H$ is substitutive, and thus value substitutive[1], we have:

$$\begin{aligned}
\Gamma \vdash^\wedge (\delta^o)^H(e, e) : \sigma &\leq \bigwedge_{\vec{v}:\Gamma} \vdash^\wedge (\delta^o)^H(e[\vec{x} := \vec{v}], e'[\vec{x} := \vec{v}]) : \sigma \\
&\leq \bigwedge_{\vec{v}} \delta^\wedge_\sigma(e[\vec{x} := \vec{v}], e'[\vec{x} := \vec{v}]) \\
&= \Gamma \vdash^\wedge \delta^o(e, e') : \sigma.
\end{aligned}$$

A similar argument holds for values. □

It is worth noticing that Theorem 16 gives the following generalisation of the so-called *metric preservation* (de Amorim et al., 2017; Reed & Pierce, 2010).

Corollary 6 (Metric Preservation)**.** *For any environment* $\Gamma \triangleq x_1 :_{s_1} \sigma, \ldots, x_n :_{s_n} \sigma$*, values* $\vec{v}, \vec{w} : \Gamma$*, and* $\Gamma \vdash^\wedge e : \sigma$ *we have:*

$$s_1 \circ \delta^{\mathcal{V}}_{\sigma_1}(v_1, w_1) \otimes \cdots \otimes s_n \circ \delta^{\mathcal{V}}_{\sigma_n}(v_n, w_n) \leq \delta^\wedge_\sigma(e[\vec{x} := \vec{v}], e[\vec{x} := \vec{w}]).$$

Having proved that applicative similarity distance is a compatible generalised metric, we now move to applicative bisimilarity distance.

[1] Notice that in Definition 73 we substitute *closed* values (in computations and values) meaning that simultaneous substitution and sequential substitution coincide. In particular, value substitutivity implies e.g.

$$(\Gamma \vdash^\wedge \alpha(e, f') : \tau) \leq \bigwedge_{\vec{v}:\Gamma} \alpha^\wedge_\tau(e[\vec{x} := \vec{v}], f[\vec{x} := \vec{v}]).$$

.

12.3 Effectful Applicative bisimilarity Distance

In previous section we proved that applicative similarity distance is a compatible $\mathbb{V}$-preorder. However, in the context of programming language semantics it is often desirable to work with $\mathbb{V}$-equivalences (cf. pseudometrics). In this section we discuss two natural program $\mathbb{V}$-equivalences: *effectful applicative bisimilarity distance* (applicative bisimilarity distance, for short) and *two-way effectful applicative similarity distance* (two-way applicative similarity distance, for short). We prove that under suitable conditions on CBEs, both applicative bisimilarity distance and two-way applicative similarity distance are compatible $\mathbb{V}$-equivalences. Proving compatibility of the latter is straightforward, whereas proving compatibility of the former is non-trivial, and requires a generalisation of the *transitive closure trick* of Chapter 5.

Before entering formalities, let us remark that so far we have mostly worked with inequation and inequalities. That was fine since we have been interested in non-symmetric $\mathbb{V}$-relations. However, for symmetric $\mathbb{V}$-relations inequalities seem to be not powerful enough, and often plain equalities are needed in order to make proofs work. For that reason in the rest of this section we assume CBFs to be monotone *monoid (homo)morphism*. That is, we modify Definition 64 requiring the equalities:

$$h(k) = \ell \qquad\qquad h(a \otimes b) = h(a) \otimes h(b).$$

Notice that we do not require CBEs to be join-preserving (i.e. continuous). We also require operations $[\![\mathbf{op}]\!]_{\mathbb{V}}$ to be *quantale (homo)morphism*, i.e. to preserves unit, tensor, and joins. It is easy to see that the new requirements are met by most of the examples considered so far. We start with two-way applicative similarity distance.

Proposition 32. *For a $\mathbb{V}$-relator Γ, define two-way applicative similarity distance as $\delta \otimes \delta^{\circ}$. Then two-way applicative similarity distance is a compatible $\mathbb{V}$-equivalence.*

Proof. Clearly $\delta \otimes \delta^{\circ}$ is symmetric. Moreover, since CBEs are monoid (homo)morphism it is also compatible. □

We now move to the more interesting case of applicative bisimilarity distance.

Definition 79. *Define* effectful applicative bisimilarity distance with respect to Γ *(applicative bisimilarity distance, for short), denoted as γ, as effectful applicative similarity distance with respect to $\Gamma \wedge \Gamma^{\circ}$.*

Proposition 30 implies that γ is reflexive and transitive. Moreover, if CBEs preserve binary meet, i.e. $s(a) \wedge s(b) = s(a \wedge b)$ for any CBE s in Π, then γ is also symmetric, ad thus a $\mathbb{V}$-equivalence. Finally we observe that γ is the greatest λ-term $\mathbb{V}$-relation α such that both α and α° are applicative simulation distance (with respect to Γ).

We now look at the proof of compatibility of γ. As usual, we cannot rely on Lemma 43 since $\Gamma \wedge \Gamma^{\circ}$ being conversive is, in general, not inductive. To overcome this problem, we proceed as in Chapter 5 and characterise applicative bisimilarity distance differently.

Lemma 44. *Let Γ be a $\mathbb{V}$-relator. Define the λ-term $\mathbb{V}$-relation γ' as follows:*

$$\gamma' \triangleq \bigvee \{\alpha \mid \alpha^{\circ} = \alpha,\ \alpha \leq [\alpha]\}.$$

Then:

1. *γ' is a symmetric applicative simulation distance (with respect to Γ), and therefore the largest such.*
2. *γ' coincides with γ.*

Proof. Obviously γ is an applicative simulation distance (with respect to Γ). Moreover, γ is symmetric and thus we have $\gamma \leq \gamma'$. To see that $\gamma' \leq \gamma$ it is sufficient to prove that γ' is an applicative simulation distance with respect to $\Gamma \wedge \Gamma^{\circ}$. Clauses on values are trivially satisfied. We now show that for any symmetric applicative simulation α, we have the inequality $\alpha^{\wedge}_{\sigma}(e, e') \leq \Gamma\alpha^{\vee}_{\sigma}([\![e]\!], [\![e']\!]) \wedge \Gamma(\alpha^{\vee}_{\sigma})^{\circ}([\![e']\!], [\![e]\!])$ for all terms $e, e' \in \Lambda_{\sigma}$. For that it is sufficient to prove $\alpha^{\wedge}_{\sigma}(e, e') \leq \Gamma(\alpha^{\vee}_{\sigma})^{\circ}([\![e']\!], [\![e]\!])$, which obviously holds since α is symmetric. □

Lemma 44 allows to apply the Key Lemma on γ, thus showing that γ^H is compatible. However, as we already know there is little hope to prove γ^H to be symmetric. We thus look at its transitive closure. The latter is in general not symmetric, but it is so for a specific class of CBEs.

Definition 80. *We say that a CBE s is* finitely continuous, *if $s \neq \infty$ implies $s(\bigvee A) = \bigvee\{s(a) \mid a \in A\}$, for any set $A \subseteq \mathsf{V}$.*

Example 69. All concrete CBEs considered in previous examples satisfy the conditions mentioned in this section, and thus are finitely continuous. Moreover, it is easy to prove the all CBEs defined from the CBEs n, ∞ of Example 57 using operations in Lemma 36 are finitely continuous[2] provided that V is completely distributive (Hofmann et al., 2014) and $[\![\mathbf{op}]\!]_{\mathsf{V}}(a_1, \ldots, \bot, \ldots, a_n) = \bot$ (which is the case for most of the concrete operations we considered). ⊠

The following is the central result of our argument.

Lemma 45. *Assume CBEs in Π to be finitely continuous. Define the transitive closure α^T of a $\mathbb{V}$-relation α as $\alpha^T \triangleq \bigvee_n \alpha^{(n)}$, where $\alpha^{(0)} \triangleq \mathsf{I}$, and $\alpha^{(n+1)} \triangleq \alpha^{(n)} \cdot \alpha$.*

1. *Let α be a reflexive and transitive λ-term $\mathbb{V}$-relation. Then $(\alpha^H)^T$ is compatible.*
2. *Let α be an reflexive, symmetric, and transitive open λ-term $\mathbb{V}$-relation. Then $(\alpha^H)^T$ is symmetric.*

Proof. We start with point 1. First of all observe that by Lemma 41 α^H is compatible. To prove compatibility of $(\alpha^H)^T$ we have to check that it satisfies all clauses in Figure 12.2. We show the case for sequential composition as an illustrative example (the other cases are proved in a similar, but easier, way). We have to prove:

$$(s \wedge 1) \circ (\Gamma \vdash (\alpha^H)^T(e, e') : \sigma) \otimes (\Delta, x :_s \sigma \vdash (\alpha^H)^T(f, f') : \tau) \\ \leq (s \wedge 1) \cdot \Gamma \vdash (\alpha^H)^T(\mathbf{let}\ x = e\ \mathbf{in}\ f, \mathbf{let}\ x = e'\ \mathbf{in}\ f') : \tau.$$

Let $c \triangleq ((s \wedge 1) \cdot \Gamma \vdash (\alpha^H)^T(\mathbf{let}\ x = e\ \mathbf{in}\ f, \mathbf{let}\ x = e'\ \mathbf{in}\ f') : \tau)$. By definition of transitive closure we have to prove:

$$(s \wedge 1) \circ \bigvee_n (\Gamma \vdash (\alpha^H)^{(n)}(e, e') : \sigma) \otimes \bigvee_m (\Delta, x :_s \sigma \vdash (\alpha^H)^{(m)}(f, f') : \tau) \leq c.$$

By finite continuity, either $s \wedge 1 = \infty$ or it is continuous with respect to joints. In the former case we are trivially done. So suppose the latter case, so that thesis becomes:

$$\bigvee_n (s \wedge 1) \circ (\Gamma \vdash (\alpha^H)^{(n)}(e, e') : \sigma) \otimes \bigvee_m (\Delta, x :_s \sigma \vdash (\alpha^H)^{(m)}(f, f') : \tau) \leq c.$$

In particular, we also have $s \neq \infty$. We prove that for any $n, m \geq 0$ the following holds: for all e, e', f, f' (of appropriate type),

$$(s \wedge 1) \circ (\Gamma \vdash (\alpha^H)^{(n)}(e, e') : \sigma) \otimes (\Delta, x :_s \sigma \vdash (\alpha^H)^{(m)}(f, f') : \tau) \\ \leq ((s \wedge 1) \cdot \Gamma \vdash (\alpha^H)^T(\mathbf{let}\ x = e\ \mathbf{in}\ f, \mathbf{let}\ x = e'\ \mathbf{in}\ f') : \tau)$$

[2] Recall that since a is integral we have the inequality $a \otimes \bot = \bot$ for any $a \in \mathsf{V}$.

holds. First of all we observe that since α^H is reflexive, we can assume $n = m$. In fact, if e.g. $n = m + l$, then we can 'complete' $(\alpha^H)^{(m)}$ as follows:

$$(\alpha^H)^{(m)} = (\alpha^H)^{(m)} \underbrace{\cdot \mathbf{I} \cdot \cdots \cdot \mathbf{I}}_{l\text{-times}} \leq (\alpha^H)^{(m)} \underbrace{\cdot \alpha^H \cdot \cdots \cdot \alpha^H}_{l\text{-times}} = (\alpha^H)^{(n)}.$$

We now do induction on n. The base case is trivial. Let us turn on the inductive step. We have to prove:

$$(s \wedge 1) \circ \Big(\bigvee_{e''} (\Gamma \vdash \alpha^H(e, e'') : \sigma) \otimes (\Gamma \vdash (\alpha^H)^{(n)}(e'', e') : \sigma) \Big) \\ \otimes \bigvee_{f''} (\Delta, x :_s \sigma \vdash \alpha^H(f, f'') : \tau) \otimes (\Delta, x :_s \sigma \vdash (\alpha^H)^{(n)}(f'', f') : \tau) \leq c.$$

Since $s \wedge 1$ is continuous it is sufficient to prove that for all terms e'', f'' we have:

$$(s \wedge 1) \circ (\Gamma \vdash \alpha^H(e, e'') : \sigma) \otimes (s \wedge 1) \circ (\Gamma \vdash (\alpha^H)^{(n)}(e'', e') : \sigma) \\ \otimes (\Delta, x :_s \sigma \vdash \alpha^H(f, f'') : \tau) \otimes (\Delta, x :_s \sigma \vdash (\alpha^H)^{(n)}(f'', f') : \tau) \leq c,$$

i.e.

$$(s \wedge 1) \circ (\Gamma \vdash \alpha^H(e, e'') : \sigma) \otimes (\Delta, x :_s \sigma \vdash \alpha^H(f, f'') : \tau) \otimes (s \wedge 1) \circ (\Gamma \vdash (\alpha^H)^{(n)}(e'', e') : \sigma) \\ \otimes (\Delta, x :_s \sigma \vdash (\alpha^H)^{(n)}(f'', f') : \tau) \leq c.$$

We can now apply compatibility of α^H plus the induction hypothesis, thus reducing the thesis to:

$$\Big((s \wedge 1) \cdot \Gamma \otimes \Delta \vdash \alpha^H(\mathbf{let}\ x = e\ \mathbf{in}\ f, \mathbf{let}\ x = e''\ \mathbf{in}\ f'') : \sigma \Big) \\ \otimes \Big((s \wedge 1) \cdot \Gamma \otimes \Delta \vdash (\alpha^H)^T(\mathbf{let}\ x = e''\ \mathbf{in}\ f'', \mathbf{let}\ x = e'\ \mathbf{in}\ f) : \sigma) \Big) \leq c.$$

We can now conclude the thesis by very definition of $(\alpha^H)^T$.

To prove point 2 we have to show $(\alpha^H)^T \leq ((\alpha^H)^T)^\circ$. For that it is sufficient to show $\alpha^H \leq ((\alpha^H)^T)^\circ$. That amounts to prove that for all computations $\Gamma \vdash e, e' : \sigma$ and values $\Gamma \vdash^{\mathsf{v}} v, v'$, and for any $a \in \mathsf{V}$ such that $\Gamma \models^{\wedge} a \leq \alpha^H(e, e') : \sigma$ is derivable we have $a \leq \Gamma \vdash (\alpha^H)^T(e, e') : \sigma$ (and similarity for $\Gamma \vdash^{\mathsf{v}} v, v' : \sigma$). The proof is by induction on the derivation of $\Gamma \models^{\wedge} a \leq \alpha^H(e, e') : \sigma$ using point 1. □

Finally, we can prove that applicative bisimilarity distance is compatible.

Theorem 17. *If any CBE in Π is finitely continuous, then applicative bisimilarity distance is compatible.*

Proof. From Lemma 45 we know that $(\gamma^H)^T$ is compatible. Therefore, it is sufficient to prove $((\gamma)^H)^T = \gamma$. One inequality follows from Lemma 41 as follows: $\gamma \leq \gamma^H \leq (\gamma)^T$. For the other inequality, we rely on the coinduction proof principle associated with γ, meaning that it is sufficient to prove that $((\gamma)^H)^T$ is a symmetric applicative simulation (with respect to Γ). Symmetry is given by Lemma 45. From Lemma 43 (Key Lemma) we know that γ^H is an applicative simulation distance. Since the identity λ-term $\mathbb{V}$-relation is an applicative simulation distance, and the composition of applicative simulation distances is itself an applicative simulation distance (see Proposition 30), we see that $(\gamma^H)^T$ is itself an applicative simulation distance (with respect to Γ). □

We conclude this chapter by noticing that we can rely on Theorem 17 to come up with concrete notions of compatible applicative bisimilarity distance. For instance, we obtain a compatible pseudometric for for P-Fuzz (observe that CBEs are indeed finitely continuous). To the best of the author's knowledge, this is the first example of a compatible *applicative* pseudometric for a *Turing complete* higher-order probabilistic sequential language (but see Section 13.2).

Chapter 13

Conclusion

> What if some day or night a demon were to steal into your loneliest loneliness and say to you: 'This life as you now live it and have lived it you will have to live once again and innumerable times again; and there will be nothing new in it, but every pain and every joy and every thought and sigh and everything unspeakably small or great in your life must return to you, all in the same succession and sequence - even this spider and this moonlight between the trees, and even this moment and I myself. The eternal hourglass of existence is turned over again and again, and you with it, speck of dust!'
>
> Friedrich Nietzsche, The Gay Science

Chapter 12 concludes our investigation on coinductive equivalences and metrics for languages with algebraic effects. Before discussing some related and future works, it is useful to briefly summarise the results we have achieved.

In the first part of this dissertation we have defined abstract notions of program equivalence and refinement for higher-order languages with algebraic effects, aiming to answer the question of whether two effectful programs can be deemed as operationally equivalent. To answer such a question we have defined the notions of effectful applicative (bi)similarity, monadic applicative (bi)similarity, and effectful normal form (bi)similarity. For all these notions we proved congruence and precongruence theorems, as well several results about their relationship.

In order to define such notions of equivalence and refinement, we have developed an abstract relational framework, based on monads and relators, which we have also used to define the notions of effectful contextual approximation (resp. equivalence) and effectful CIU approximation (resp. equivalence). The aforementioned congruence and precongruence theorems, directly lead to soundness of our notions of (bi)similarity for effectful contextual approximation (resp. equivalence).

In order to prove soundness of effectful applicative (bi)similarity, in Chapter 5 we have developed a non-trivial generalisation of Howe's method, the standard relational construction used to prove con-

gruence properties of applicative bisimilarity. We have argued that our abstract Howe's method is of interest by itself, as it sheds light on the very essence of Howe's construction itself. Additionally, we have showed a further application of such a method by proving full abstraction of effectful CIU approximation (resp. equivalence) for effectful contextual approximation (resp. equivalence).

These results hold both for call-by-name and call-by-value calculi. In Chapter 6 we have showed that we can strengthen our techniques and achieve, under mild conditions, full abstraction of monadic applicative (bi)similarity for effectful contextual equivalence. That holds, however, only for call-by-name calculi.

In Chapter 7 we have defined effectful normal form (bi)similarity, which gives the notion of effectful eager normal form (bi)similarity, in call-by-value calculi, and effectful weak head normal form, in call-by-name calculi. We have proved congruence and precongruence theorems for all these notions, as well as their (strict) inclusion in effectful applicative (bi)similarity.

Compared to other standard operational techniques, normal form bisimilarity has the major advantage of being an *intensional* program equivalence, equating programs according to the syntactic structure of their (possibly infinitary) normal forms. As a consequence, in order to deem two programs as normal form bisimilar, it is sufficient to test them in isolation, i.e. independently of their interaction with the environment. These features make normal form bisimilarity a powerful technique for program equivalence. Additionally, we make such a technique even more powerful by proving suitable up-to context techniques.

In the second part of this dissertation we have tackled a different problem, namely the one of quantifying operational differences between programs. That directly led to the investigation of notions of effectful program metric (or distance). In Chapter 10, Chapter 11, and Chapter 12 we have defined effectful applicative (bi)similarity distance for the calculus $\mathbb{V}$-Fuzz. The latter is a linear λ-calculus with a powerful type system expressing program sensitivity.

In order to define effectful applicative (bi)similarity distance, which is *de facto* the refinement of effectful applicative (bi)similarity to quantale-valued relations, in Chapter 12 we have designed an abstract relational framework for studying effectful program distances. Such a framework builds on Lawvere's insights on the categorical nature of metric spaces as well as on results from monoidal topology. In particular, of a central importance is the notion of a $\mathbb{V}$-relator (Chapter 11), i.e. of a relator acting on quantale-valued relations. Prime examples of $\mathbb{V}$-relators are provided by the well-known Hausdorff and Wasserstein-Kantorovich distances.

Finally, in Chapter 12 we have proved our main results, namely congruence and precongruence theorems for effectful applicative bisimilarity and similarity distance, respectively. Such results are proved with a further generalisation of Howe's method.

13.1 Related Work

The results presented in this dissertation are definitely not the first account of program equivalences, refinements, and metrics for effectful languages. However, our results are, to the best of the author's knowledge, the first abstract accounts of operational *coinductive* techniques for languages with algebraic effects.

Concerning applicative notions of equivalence, to the best of the author's knowledge, no abstract account exists on applicative coinductive techniques for calculi with algebraic effects. Nevertheless, there are several works that investigate such techniques for calculi with specific effects. Among them, prime examples are given by (C. L. Ong, 1993) and (S. Lassen, 1998b), to which this dissertation is in great debt, for nondeterministic calculi, and (Crubillé & Dal Lago, 2014; Dal Lago et al., 2014) for probabilistic calculi. An abstract account of logic-based equivalences for higher-order languages with algebraic effects has been developed in (Simpson & Voorneveld, 2018), where congruence properties of the equivalences

defined are proved using our abstract Howe's method. Applicative bisimilarity has also been investigated for languages with non-algebraic effects, notably for languages with control operators (Biernacki & Lenglet, 2014). However, we have to remark that applicative bisimilarity has been proved to be fragile in presence of non-algebraic effects, and actually unsound in presence of information hiding (Koutavas et al., 2011).

Normal form bisimulations have been extensively investigated for pure λ-calculi (S. B. Lassen, 1999, 2005; C. L. Ong, 1993; Sangiorgi, 1994), as well as for calculi with non-algebraic effects, notably continuations (Biernacki, Lenglet, & Polesiuk, 2018) and control operators (S. B. Lassen, 2006a; Støvring & Lassen, 2007). Until Lassen's work on eager normal form bisimulation (S. B. Lassen, 2005), normal form bisimilarity has been mainly investigated for call-by-name languages, due to its equivalence with Lévy-Longo tree equivalence, and for the so-called classical theory of λ-calculus (Barendregt, 1984), due to its equivalence with Böhm tree equivalence.

To the best of the author's knowledge, no abstract account of normal form bisimilarity in the context of languages with algebraic effects has been given so far. However, there are some works on specific effectful extensions of Böhm trees, notably the work of de Liguoro and Piperno on nondeterministic Böhm trees (De Liguoro & Piperno, 1995), and the work of Leventis on probabilistic Böhm trees (Leventis, 2018), as well as works on nondeterministic extensions of weak head normal form bisimulation, notably (S. B. Lassen, 2006b).

The situation is rather different if one looks at non-coinductive abstract theories of program equivalence. Denotational semantics of effectful calculi has been studied since Moggi's seminal work (Moggi, 1989), thus implicitly providing a notion of program equivalence. All this has been given a more operational flavour starting with Plotkin and Power account of adequacy for algebraic effects (G. D. Plotkin & Power, 2001, 2002). The literature also offers abstract accounts on logical relations for effectful calculi. The first of them is due to Goubault-Larrecq, Lasota and Nowak (Goubault-Larrecq, Lasota, & Nowak, 2008), which is noticeably able to deal with nondeterministic and probabilistic effects, but also with dynamic name creation, for which applicative bisimilarity is known to be unsound (Koutavas et al., 2011). Similar in spirit to our approach, the work of Katsumata and Sato (Katsumata & Sato, 2013) (as well as (Katsumata, 2013)) analyses monadic lifting of relations in the context of $\top\top$-lifting.

Another piece of work which is related to ours is due to Johann, Simpson, and Voigtländer (Johann, Simpson, & Voigtländer, 2010), who focused on algebraic effects and observational equivalence, and their characterisation via CIU theorems and a form of logical relation based $\top\top$-lifting. In both cases, the target language is a call-by-name typed λ-calculus. A further account on logical relations for languages with algebraic effects is given in (Biernacki, Piróg, Polesiuk, & Sieczkowski, 2018). An extensive analysis of observational equivalences and preorders for languages with algebraic effects (with a case studied on combinations of probability with nondeterminism) is given in (Lopez & Simpson, 2018).

Concerning program metrics, several works have been done in the past years on quantitative reasoning in the context of programming language semantics. In particular, several authors have used (cartesian) categories of *ultrametric spaces* as a foundation for denotational semantics of both concurrent (Arnold & Nivat, 1980; de Bakker & Zucker, 1982) and sequential programming languages (Escardo, 1999). Bisimilarity-based distances have been proposed for probabilistic *first-order* calculi (Du, Deng, & Gebler, 2016), where, due to the absence of higher-order features, amplification phenomena do not occur (but see (Gebler, K.G., & Tini, 2016)). A different approach is investigated in (de Amorim et al., 2017), where a denotational semantics combining ordinary metric spaces and domains is given to *pure* (i.e. effect-free) Fuzz. The main theorem of (de Amorim et al., 2017) is a denotational version of the so-called *metric preservation* (Reed & Pierce, 2010) (whose original proof requires a suitable *step-indexed metric logical relation*). Corollary 6 is the operational counterpart of such result generalised to arbitrary algebraic effects.

A different but deeply related line of research has been recently proposed in (Crubillé & Dal Lago, 2015, 2017), where coinductive, operationally-based distances have been studied for probabilistic λ-

calculi. In particular, in (Crubillé & Dal Lago, 2015) a notion of applicative distance based on the Wasserstein lifting is proposed for a probabilistic *affine* λ-calculus. Restricting to affine programs only makes the calculus strongly normalising and removes copying capabilities of programs by construction. That way programs cannot amplify distances between their inputs and therefore are forced to behave as non-expansive functions. This limitation is overcome in (Crubillé & Dal Lago, 2017), where a coinductive notion of distance is proposed for a full linear λ-calculus, and distance trivialisation phenomena are studied in depth. The price to pay for such generality is that the distance proposed is not applicative, but a tuple distance somehow resembling environmental bisimilarity (Sangiorgi et al., 2011).

13.2 Future Work

Topics for further work are plentiful. Here we list some major lines of research.

13.2.1 Handlers

This dissertation analyses calculi with algebraic effects. As remarked in the introduction of this work, the theory of algebraic effects has been extended with *effect handlers*, giving rise to the theory of *algebraic effects and handlers* (Bauer & Pretnar, 2015; G. D. Plotkin & Pretnar, 2013; Pretnar, 2015). An *handler* is a syntactical constructor for manipulating the control flow of the program by specifying how to interpret operation symbols. For instance, consider the program $\mathbf{print}_c.e$ that prints the character c and continues as e. As usual, the operational semantics of $\mathbf{print}_c.e$ is to *first* print c, and *then* continue as e. We can use handlers to modify the control flow of computation, so that we *first* evaluate e, and *then* we print c. For that we use the handler:

$$\mathbf{handler}\ \{\mathbf{print}_c.k \mapsto \mathbf{let}\ x = k\ \mathbf{in}\ \mathbf{print}_c.(\mathbf{return}\ x)\}.$$

Accordingly, the handler acts as way to redefine the operational behaviour of $\mathbf{print}_c$, and thus behaves as a kind of algebraic homomorphism.

It is an open question whether the techniques developed in this dissertation can be extended to languages with algebraic effects and handlers. This is certainly the case for some (restricted) handlers. An example is provided by Example 21, where a restricted exception handling constructor is introduced. Although non-algebraic, such operation can be easily shown to satisfy condition (Σ comp), meaning that we can apply our abstract soundness results to that calculus.

Extending such a result to arbitrary effect handlers, however, seems to require non-trivial extension of the framework designed, as the semantics of an operation symbol is now determined by the context in which it is evaluated. A promising route to achieve such a goal seems to look at specific normal form bisimulations.

13.2.2 Full Abstraction and Non-algebraic Effects

Both effectful applicative (bi)similarity and effectful normal form (bi)similarity have been proved to be *sound*, but not *fully abstract* for effectful contextual equivalence/approximation. An interesting question is whether there exist conditions on monads and relators guaranteeing full abstraction results.

Full abstraction of applicative bisimilarity is known to hold for the pure λ-calculus (Abramsky, 1990a), as well as for the probabilistic λ-calculus (Crubillé & Dal Lago, 2014), but to fail for the non-deterministic λ-calculus (S. Lassen, 1998b). The latter perfectly fits the abstract theory of Chapter 5, meaning that proving full abstraction results for effectful applicative (bi)similarity seems to require a severe restriction on our axioms of Σ-continuous monads and relators. A promising route towards this

challenge would be to understand which class of *tests* (if any) characterise effectful applicative bisimilarity, depending on the underlying monad and relator, this way generalising results in (Van Breugel, Mislove, Ouaknine, & Worrell, 2005) and (Crubillé & Dal Lago, 2014).

Another interesting line of research is to investigate whether the theory of normal form bisimulation can be extended to languages with non-algebraic effects. In particular, normal form bisimilarity has been extended to languages with local states (Støvring & Lassen, 2007), control operators (Biernacki, Lenglet, & Polesiuk, 2018), and both control operators and local states, where it has additionally been proved to be fully abstract for contextual equivalence (Støvring & Lassen, 2007). In light of such results, it is natural to ask whether there exists an abstract notion of normal form bisimulation for languages with local effects. Achieving such a result would be a major improvement in the theory of normal form bisimulation, especially considering that applicative bisimilarity has been proved to be unsound for several languages with non-algebraic effects (Koutavas et al., 2011).

13.2.3 Effectful Logical Relations

Recalling the distinction between observational, applicative (or extensional), intensional, and logical program equivalence made in the introduction, it is clear that a major issue that has not been addressed in this work concerns effectful logical equivalences. Logical relations techniques for effectful languages have been extensively investigated both for languages with specific algebraic effects (Bizjak & Birkedal, 2015; Culpepper & Cobb, 2017) and for effectful languages in general (Goubault-Larrecq et al., 2008; Katsumata, 2013; Katsumata & Sato, 2013). In particular, the latter work uses relation lifting constructions resembling relators, and thus seems to suggest the applicability of our relational framework to study effectful logical relations.

We also remark that logical relations for general algebraic effects have been studied in (Johann et al., 2010) and (Biernacki, Piróg, et al., 2018). The latter work seems to have some limitations for our purposes, since the calculus studied does not take into account pure algebraic operations, i.e. (algebraic) operations whose semantics is defined independently of handlers. Having such operations give calculi more expressive power, as handlers alone do not allow for a fine analysis of e.g. true probabilistic nondeterminism (the latter requires to have operations, either internal or external to the calculus, that act as sources of probabilistic nondeterminism). More relevant for us is (Johann et al., 2010), where logical relations for a call-by-name polymorphic PCF (G. Plotkin, 1977) enriched with algebraic operations are defined. It is not hard to realise that the definition of such logical relations can be mimicked in our framework relying on the observation function of Chapter 6 and on a suitable step-indexing (Appel & McAllester, 2001), and that the logical relations thus obtained can be used to provide an alternative characterisation of effectful contextual equivalence/approximation. That holds for both call-by-name and call-by-value calculi.

An abstract understanding of the relationship between logical relations and notions of bisimulation is, however, still missing.

13.2.4 Program Distances and Coeffects

Compared to the results achieved on program equivalence and refinements, our investigation on program distance is at its very beginning. A first extension of the theory developed is the design of effectful contextual and CIU distances. Such distances should be definable by generalising the results of Chapter 5 to $\mathbb{V}$-Fuzz and $\mathbb{V}$-relations. The author conjectures that to be possible, provided one restricts to finitely-continuous change of base functors. On a similar line of research, the development of effectful distances based on logical relations, along the lines of (Reed & Pierce, 2010), should be rather straightforward.

A potentially more interesting notion of program distance might be the generalisation of normal form bisimilarity to the abstract setting of quantale-valued relations.

On a different side, we observe that program sensitivity is an instance of the more general notion of a *coeffect*. Coeffects (Brunel et al., 2014; Gaboardi et al., 2016; Petricek et al., 2014) are a formalism introduced to model context-dependent computations, i.e. computations whose behaviour depends on the context in which they happen. From a mathematical perspective, coeffects are the dual of effects. As a consequence, as effects are modelled as monads, coeffects are modelled as comonads. This suggests the possibility of looking at more general notions of distances (as well as equivalences), for effectful and coeffectful calculi.

Finally, recalling the informal introduction to program metric of Chapter 9, we notice that the development of the theory of applicative bisimulation distances is rooted in the refinement of Assumption 3 (non-expansiveness) into Assumption 4 (Lipshitz-continuity). Such a refinement allowed us to bypass problems raised by distance trivialisation. However, that was only *one* possible route to define well-behaved notions of program distance. Another possibility is the one investigated in (Crubillé & Dal Lago, 2017), where the proposed notion of *tuple distance* can be seen as a way to refine Assumption 1 by defining distances not on pairs of terms, but on pairs of tuples. Looking at other possible 'non-standard' notions of program distance is a stimulating topic for further research.

Appendix A

Names of Categories

The following is a list of the categories mentioned in this dissertation.

- Set: the category of sets and functions. Objects are denoted by letters $X, Y, \ldots$ whereas morphisms by letters $f : X \to Y, g : X \to Y, \ldots$.
- Rel: the category of sets and relations. Objects are denoted by letters $X, Y, \ldots$ whereas morphisms by letters $\mathcal{R} : X \nrightarrow Y, \mathcal{S} : X \nrightarrow Y, \ldots$.
- PreOrd: the category of preordered sets and monotone functions. A preorder on a set X is a reflexive and transitive relation $\leq_X : X \nrightarrow X$. We denote objects of PreOrd as $(X, \leq_X), (Y, \leq_Y), \ldots$. A morphism $f : (X, \leq_X) \to (Y, \leq_Y)$ is a morphism $f : X \to Y$ in Set such that $f \cdot \leq_X \subseteq \leq_Y \cdot f$ holds in Rel.
- $\mathbb{V}$-Rel is the category of sets and $\mathbb{V}$-relations, for a quantale V. Objects are denoted by letters $X, Y, \ldots$ whereas morphisms by letters $\alpha : X \nrightarrow Y, \beta : X \nrightarrow Y, \ldots$.
- $\mathbb{V}$-Cat: the category of $\mathbb{V}$-categories and $\mathbb{V}$-functor. A $\mathbb{V}$-category (X, α) consists of a set X together with a reflexive and transitive $\mathbb{V}$-relation $\alpha : X \nrightarrow X$. We denote objects of $\mathbb{V}$-Cat as $(X, \alpha), (Y, \beta), \ldots$. A morphism $f : (X, \alpha) \to (Y, \beta)$ is a $\mathbb{V}$-functor, i.e. a morphism $f : X \to Y$ in Set such that $f \cdot \alpha \leq \beta \cdot f$ holds in $\mathbb{V}$-Rel.

Appendix B

Spans

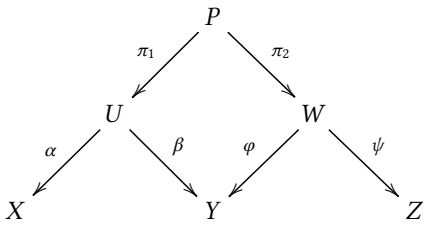
In order to have a better grasp of the Barr extension of a functor, it is useful to look at relations using the notion of a *span* (Bruni & Gadducci, 2003; Carboni, Kasangian, & Street, 1984).

Definition 81. *A span on a category $\mathbb{C}$ is given by an ordered pair of $\mathbb{C}$-morphisms $(\alpha : U \to X, \beta : U \to Y)$. The object U is called the source of the span.*

We use the notation $\langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y\rangle$ for spans. Given spans $\langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y\rangle$ and $\langle Y \xleftarrow{\varphi} W \xrightarrow{\psi} Z\rangle$, their composition can be defined as follows, provided that $\mathbb{C}$ has enough pullbacks. Let $\langle U \xleftarrow{\pi_1} P \xrightarrow{\pi_2} W\rangle$ be a pullback of $\beta : U \to Y$ and $\varphi : W \to Y$, then define:

$$\langle Y \xleftarrow{\varphi} W \xrightarrow{\psi} Z\rangle \cdot \langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y\rangle \triangleq \langle X \xleftarrow{\alpha\cdot\pi_1} P \xrightarrow{\psi\cdot\pi_2} Z\rangle$$

Diagrammatically:



Since pullbacks are unique modulo isomorphism, we need to work with equivalence classes of spans. For any $\mathbb{C}$-object U the span $\langle U \xleftarrow{1} U \xrightarrow{1} U\rangle$ acts as identity for composition. In fact, given a span $\langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y\rangle$, the composition $\langle Y \xleftarrow{1} Y \xrightarrow{1} Y\rangle \cdot \langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y\rangle$ is obtained by taking a pullback of $\beta : U \to Y$ and $1 : Y \to Y$. Obviously $\langle U \xleftarrow{1} U \xrightarrow{\beta} Y\rangle$ is such a pullback, so that we have $\langle Y \xleftarrow{1} Y \xrightarrow{1} Y\rangle \cdot \langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y\rangle = \langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y\rangle$.

Spans can be used as set-theoretic presentation of relations. In fact, any relation $\mathcal{R}$ can be represented as the span $\langle X \xleftarrow{\pi_1} G_{\mathcal{R}} \xrightarrow{\pi_2} Y\rangle$, and any span $\langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y\rangle$ defines a relation, namely $\beta \cdot \alpha^\circ : X \nrightarrow Y$. Spans, however, retain more information than relations, as they are sensitive to multiplicity (in fact, there is a correspondence between spans and relations over multisets; see (Bruni & Gadducci, 2003) for details).

Barr's construction can be seen as a construction on spans. In fact, we can define the Barr extension of a functor T as the map $\overline{T}$ on spans thus defined:

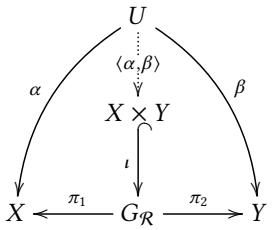
$$\overline{T} : \langle X \xleftarrow{\alpha} Z \xrightarrow{\beta} Y \rangle \mapsto \langle TX \xleftarrow{T\alpha} TZ \xrightarrow{T\beta} Y \rangle$$

This abstract perspective on Barr's construction gives several insights on the structure of $\overline{T}$. For instance, given the prime role of pullbacks in the definition of span composition, it comes with no surprise that in order for $\overline{T}$ to satisfy (rel 2) (properly generalised to spans), T need preserve weak pullbacks.

We conclude remarking that we can indeed safely look at Barr's construction in terms of span, rather than relation. This is not evident, in principle, as spans and relations are not in a one-to-one correspondence, and thus different spans may represent different relations. Nonetheless, we can easily prove that Barr's construction is independent from the span representation of a relation.

Lemma 46. *Let $\mathcal{R} : X \nrightarrow Y$ be a relation, and let $\langle X \xleftarrow{\pi_1} G_{\mathcal{R}} \xrightarrow{\pi_2} Y \rangle$ be the span associated with $\mathcal{R}$. Let $\langle X \xleftarrow{\alpha} U \xrightarrow{\beta} Y \rangle$ be another span representing $\mathcal{R}$, that is such that $\beta \cdot \alpha^\circ = \pi_2 \cdot \pi_1^\circ$ holds. Then $\langle TX \xleftarrow{T\pi_1} TG_{\mathcal{R}} \xrightarrow{T\pi_2} TY \rangle$ and $\langle TX \xleftarrow{T\alpha} TU \xrightarrow{T\beta} TY \rangle$ represent the same relation too; that is, $T\beta \cdot (T\alpha)^\circ = T\pi_2 \cdot (T\pi_1)^\circ$ holds.*

Proof. First of all notice that the map $e = \iota \cdot \langle \alpha, \beta \rangle$ is surjective.



In fact, for any $(x, y) \in G_{\mathcal{R}}$ we have $x \, (\pi_2 \cdot \pi_1^\circ) \, y$, and thus $x \, (\beta \cdot \alpha^\circ) \, y$. That means that there exists $u \in U$ such that $\alpha(u) = x$ and $\beta(u) = y$, meaning that $e(u) = (x, y)$. Since e is a surjection, by the axiom of choice Te is a surjection too. In particular, we have $Te \cdot (Te)^\circ = \mathsf{I}$, and thus we can calculate as follows:

$$T\beta \cdot (T\alpha)^\circ = T(\pi_2 \cdot e) \cdot (T(\pi_1 \cdot e))^\circ = T\pi_2 \cdot Te \cdot (Te)^\circ \cdot (T\pi_1)^\circ = T\pi_2 \cdot (T\pi_1)^\circ.$$

□

References

Abramsky, S. (1990a). The lazy lambda calculus. In D. Turner (Ed.), *Research topics in functional programming* (pp. 65–117). Addison Wesley.

Abramsky, S. (1990b). Research topics in functional programming. In D. A. Turner (Ed.), (pp. 65–116). Addison-Wesley Longman Publishing Co., Inc.

Abramsky, S., & Jung, A. (1994). Domain theory. In *Handbook of logic in computer science* (pp. 1–168). Clarendon Press.

Abramsky, S., & Ong, C. L. (1993). Full abstraction in the lazy lambda calculus. *Inf. Comput.*, *105*(2), 159–267.

Appel, A., & McAllester, D. (2001). An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, *23*(5), 657–683.

Arnold, A., & Nivat, M. (1980). Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theor. Comput. Sci.*, *11*, 181–205.

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., … Woodger, M. (1960). Report on the algorithmic language algol 60. *Commun. ACM*, *3*(5), 299–314.

Baldan, P., Bonchi, F., Kerstan, H., & König, B. (2014). Behavioral metrics via functor lifting. In *Proc. of FSTTCS* (pp. 403–415).

Baldan, P., Bonchi, F., Kerstan, H., & König, B. (2015). Towards trace metrics via functor lifting. In *Proc. of CALCO 2015* (pp. 35–49).

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

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

Bauer, A., & Pretnar, M. (2015). Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, *84*(1), 108–123.

Benton, N., Hughes, J., & Moggi, E. (2000). Monads and effects. In *Applied semantics, international summer school, APPSEM, advanced lectures* (pp. 42–122).

Biernacki, D., & Lenglet, S. (2014). Applicative bisimilarities for call-by-name and call-by-value $\lambda\mu$-calculus. *Electr. Notes Theor. Comput. Sci.*, *308*, 49–64.

Biernacki, D., Lenglet, S., & Polesiuk, P. (2018). Proving soundness of extensional normal-form bisimilarities. *Electr. Notes Theor. Comput. Sci.*, *336*, 41–56.

Biernacki, D., Piróg, M., Polesiuk, P., & Sieczkowski, F. (2018). Handle with care: relational interpretation of algebraic effects and handlers. *PACMPL*, *2*(POPL), 8:1–8:30.

Bizjak, A., & Birkedal, L. (2015). Step-indexed logical relations for probability. In *Proc. of FOSSACS 2015* (pp. 279–294).

Böhm, C. (1968). Alcune proprietà delle forme $\beta\eta$-normali del λk-calcolo. *Pubblicazioni dell'Istituto per le Applicazioni del Calcolo*, *696*.

Brachthäuser, J. I., & Schuster, P. (2017). Effekt: Extensible algebraic effects in scala (short paper). In *Proceedings of the 8th acm sigplan international symposium on scala* (pp. 67–72).

Brunel, A., Gaboardi, M., Mazza, D., & Zdancewic, S. (2014). A core quantitative coeffect calculus. In *Proc. of ESOP 2014* (pp. 351–370).

Bruni, R., & Gadducci, F. (2003). Some algebraic laws for spans (and their connections with multirelations). *Electronic Notes in Theoretical Computer Science*, *44*(3), 175–193. (RelMiS 2001, Relational Methods in Software)

Carboni, A., Kasangian, S., & Street, R. (1984). Bicategories of spans and relations. *Journal of Pure and Applied Algebra*, *33*(3), 259–267.

Carraro, A., & Guerrieri, G. (2014). A semantical and operational account of call-by-value solvability. In *Foundations of software science and computation structures - 17th international conference, FOSSACS 2014, held as part of the european joint conferences on theory and practice of software, ETAPS 2014, grenoble, france, april 5-13, 2014, proceedings* (pp. 103–118).

Church, A. (1985). *The calculi of lambda conversion. (am-6) (annals of mathematics studies).* Princeton, NJ, USA: Princeton University Press.

Clément, P., & Desch, W. (2008). *Wasserstein metric and subordination* (Vol. 189) (No. 1).

Clementino, M., & Tholen, W. (2014). From lax monad extensions to topological theories. , *46*, 99–123.

Crubillé, R., & Dal Lago, U. (2014). On probabilistic applicative bisimulation and call-by-value lambda-calculi. In *Proc. of ESOP 2014* (pp. 209–228).

Crubillé, R., & Dal Lago, U. (2015). Metric reasoning about lambda-terms: The affine case. In *Proc. of LICS 2015* (pp. 633–644).

Crubillé, R., & Dal Lago, U. (2017). Metric reasoning about lambda-terms: The general case. In *Proc. of ESOP 2017* (pp. 341–367).

Culpepper, R., & Cobb, A. (2017). Contextual equivalence for probabilistic programs with continuous random variables and scoring. In *Proceedings of ESOP 2017* (pp. 368–392).

Dal Lago, U., Sangiorgi, D., & Alberti, M. (2014). On coinductive equivalences for higher-order probabilistic functional programs. In *Proc. of POPL 2014* (pp. 297–308).

Dal Lago, U., & Zorzi, M. (2012). Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, *46*(3), 413–450.

Davey, B., & Priestley, H. (1990). *Introduction to lattices and order.* Cambridge University Press.

De Nicola, R., & Hennessy, M. (1983). Testing equivalence for processes. In *Automata, languages and programming, 10th colloquium, barcelona, spain, july 18-22, 1983, proceedings* (pp. 548–560).

de Amorim, A., Gaboardi, M., Hsu, J., Katsumata, S., & Cherigui, I. (2017). A semantic account of metric preservation. In *Proc. of POPL 2017* (pp. 545–556).

de Bakker, J., & Zucker, J. (1982). Denotational semantics of concurrency. In *Stoc* (pp. 153–158).

De Liguoro, U., & Piperno, A. (1995). Non deterministic extensions of untyped lambda-calculus. *Inf. Comput.*, *122*(2), 149-177.

de Saussure, F., & Baskin, W. (2011). *Course in general linguistics: Translated by wade baskin. edited by perry meisel and haun saussy.* Columbia University Press.

de Vink, E. P., & Rutten, J. J. M. M. (1997). Bisimulation for probabilistic transition systems: A coalgebraic approach. In *Automata, languages and programming, 24th international colloquium, icalp'97, bologna, italy, 7-11 july 1997, proceedings* (pp. 460–470).

Dolan, S., Eliopoulos, S., Hillerström, D., Madhavapeddy, A., Sivaramakrishnan, K. C., & White, L. (2017). Concurrent system programming with effect handlers. In *Trends in functional programming - 18th international symposium, TFP 2017, canterbury, uk, june 19-21, 2017, revised selected papers* (pp. 98–117).

Donkel, D. (2001). *The theory of difference: Readings in contemporary continental thought.* State University of New York Press.

Du, W., Deng, Y., & Gebler, D. (2016). Behavioural pseudometrics for nondeterministic probabilistic systems. In *Proc. of SETTA 2016* (pp. 67–84).

Durier, A., Hirschkoff, D., & Sangiorgi, D. (2018). Eager functions as processes. In *Proceedings of the 33rd annual ACM/IEEE symposium on logic in computer science, LICS 2018* (pp. 364–373).

Dwork, C. (2006). Differential privacy. In *Automata, languages and programming, 33rd international*

colloquium, ICALP 2006, venice, italy, july 10-14, 2006, proceedings, part II (pp. 1–12).
Escardo, M. (1999). A metric model of pcf. In *Workshop on realizability semantics and applications.*
Felleisen, M., & Hieb, R. (1992). The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, *103*(2), 235–271.
Gabbrielli, M., & Martini, S. (2010). *Programming languages: Principles and paradigms.* Springer.
Gaboardi, M., Katsumata, S., Orchard, D. A., Breuvart, F., & Uustalu, T. (2016). Combining effects and coeffects via grading. In *Proc. of ICFP 2016* (pp. 476–489).
Gebler, D., K.G., L., & Tini, S. (2016). Compositional bisimulation metric reasoning with probabilistic process calculi. *Logical Methods in Computer Science*, *12*(4).
Girard, J., Scedrov, A., & Scott, P. (1992). Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.*, *97*, 1–66.
Goguen, J. A., Thatcher, J. W., Wagner, E. G., & Wright, J. B. (1977). Initial algebra semantics and continuous algebras. *J. ACM*, *24*(1), 68–95.
Gordon, A. (1994, September). A tutorial on co-induction and functional programming. In *Workshops in computing* (p. 78-95). Springer London.
Goubault-Larrecq, J., Lasota, S., & Nowak, D. (2008). Logical relations for monadic types. *Mathematical Structures in Computer Science*, *18*(6), 1169–1217.
Hasuo, I., Jacobs, B., & Sokolova, A. (2007). Generic trace semantics via coinduction. *Logical Methods in Computer Science*, *3*(4).
Hausdorff, F. (1949). *Grundzüge der mengenlehre.* Chelsea.
Hermida, C., & Jacobs, B. (1998). Structural induction and coinduction in a fibrational setting. *Inf. Comput.*, *145*(2), 107–152.
Hoffman, D. (2015). A cottage industry of lax extensions. *Categories and General Algebraic Structures with Applications*, *3*(1), 113-151.
Hofmann, D. (2007). Topological theories and closed objects. *Adv. Math.*, *215*, 789-824.
Hofmann, D., & Reis, C. (2018). Convergence and quantale-enriched categories. *Categories and General Algebraic Structures with Applications*, *9*(1), 77-138.
Hofmann, D., Seal, G., & Tholen, W. (Eds.). (2014). *Monoidal topology. a categorical approach to order, metric, and topology* (No. 153). Cambridge University Press.
Howard, R. (2007). *Dynamic probabilistic systems.* Dover Publications.
Howe, D. (1996). Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, *124*(2), 103-112.
Hughes, J., & Jacobs, B. (2004). Simulations in coalgebra. *Theor. Comput. Sci.*, *327*(1-2), 71–108.
Hyland, M., Levy, P. B., Plotkin, G. D., & Power, J. (2007). Combining algebraic effects with continuations. *Theor. Comput. Sci.*, *375*(1-3), 20–40.
Hyland, M., Plotkin, G. D., & Power, J. (2006). Combining effects: Sum and tensor. *Theor. Comput. Sci.*, *357*(1-3), 70–99.
Hyland, M., & Power, J. (2007). The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electr. Notes Theor. Comput. Sci.*, *172*, 437–458.
Jacobs, B. (2016). *Introduction to coalgebra: Towards mathematics of states and observation* (Vol. 59). Cambridge University Press.
Jacobs, B., Silva, A., & Sokolova, A. (2012). Trace semantics via determinization. In *Coalgebraic methods in computer science - 11th international workshop, CMCS 2012, colocated with ETAPS 2012* (pp. 109–129).
Jech, T. (1997). *Set theory, second edition.* Springer.
Johann, P., Simpson, A., & Voigtländer, J. (2010). A generic operational metatheory for algebraic effects. In *Proc. of LICS 2010* (pp. 209–218). IEEE Computer Society.
Katsumata, S. (2013). Relating computational effects by $\top\top$-lifting. *Inf. Comput.*, *222*, 228–246.
Katsumata, S., & Sato, T. (2013). Preorders on monads and coalgebraic simulations. In *Foundations of*

software science and computation structures - 16th international conference, FOSSACS 2013, held as part of the european joint conferences on theory and practice of software, ETAPS 2013, rome, italy, march 16-24, 2013. proceedings (pp. 145–160).

Kelly, G. M. (2005). Basic concepts of enriched category theory. *Reprints in Theory and Applications of Categories*(10), 1–136.

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

Kortanek, K., & Yamasaki, M. (1995). Discrete infinite transportation problems. *Discrete Applied Mathematics*(58), 19–33.

Koutavas, V., Levy, P. B., & Sumii, E. (2011). From applicative to environmental bisimulation. *Electr. Notes Theor. Comput. Sci.*, *276*, 215–235.

Kurz, A., & Velebil, J. (2016). Relation lifting, a survey. *J. Log. Algebr. Meth. Program.*, *85*(4), 475–499.

Landin, P. (1965a). Correspondence between ALGOL 60 and church's lambda-notation: part I. *Commun. ACM*, *8*(2), 89–101.

Landin, P. (1965b). A correspondence between ALGOL 60 and church's lambda-notations: Part II. *Commun. ACM*, *8*(3), 158–167.

Larsen, K. G., & Skou, A. (1989). Bisimulation through probabilistic testing. In *Proceedings of POPL 1989* (pp. 344–352).

Lassen, S. (1998a). Relational reasoning about contexts. In A. D. Gordon & A. M. Pitts (Eds.), *Higher order operational techniques in semantics* (pp. 91–136).

Lassen, S. (1998b). *Relational reasoning about functions and nondeterminism* (Unpublished doctoral dissertation). Dept. of Computer Science, University of Aarhus.

Lassen, S. B. (1999). Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context. *Electr. Notes Theor. Comput. Sci.*, *20*, 346–374.

Lassen, S. B. (2005). Eager normal form bisimulation. In *Proceedings of LICS 2005* (pp. 345–354).

Lassen, S. B. (2006a). Head normal form bisimulation for pairs and the \lambda\mu-calculus. In *Proceedings of LICS 2006* (pp. 297–306).

Lassen, S. B. (2006b). Normal form simulation for mccarthy's amb. *Electr. Notes Theor. Comput. Sci.*, *155*, 445–465.

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

Lawvere, W. F. (2004). Functorial Semantics of Algebraic Theories. *Reprints in Theory and Applications of Categories*, *4*, 1–121.

Leijen, D. (2017). Implementing algebraic effects in c. In B.-Y. E. Chang (Ed.), *Programming languages and systems* (pp. 339–363). Cham: Springer International Publishing.

Leventis, T. (2016). *Probabilistic lambda-theories* (Unpublished doctoral dissertation). Aix-Marseille Université.

Leventis, T. (2018). Probabilistic böhm trees and probabilistic separation. In *Proc. of lics.* (To appear.)

Lévy, J. (1975). An algebraic interpretation of the lambda beta - calculus and a labeled lambda - calculus. In *Lambda-calculus and computer science theory, proceedings of the symposium held in rome, italy, march 25-27, 1975* (pp. 147–165).

Levy, P. (2006). Infinitary howe's method. *Electr. Notes Theor. Comput. Sci.*, *164*(1), 85–104.

Levy, P. (2011). Similarity quotients as final coalgebras. In *Proc. of FOSSACS 2011* (Vol. 6604, pp. 27–41).

Levy, P., Power, J., & Thielecke, H. (2003). Modelling environments in call-by-value programming languages. *Inf. Comput.*, *185*(2), 182–210.

Levy, P. B. (2007). Amb breaks well-pointedness, ground amb doesn't. *Electronic Notes in Theoretical Computer Science*, *173*, 221–239. (Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIII))

Longo, G. (1983). Set-theoretical models of lambda calculus: Theories, expansions, isomorphisms. *Annals of Pure and Applied Logic*, *24*, 153–188.

Lopez, A., & Simpson, A. (2018). Basic operational preorders for algebraic effects in general, and for combined probability and nondeterminism in particular. In *27th EACSL annual conference on computer science logic, CSL 2018, september 4-7, 2018, birmingham, UK* (pp. 29:1–29:17).
MacLane, S. (1971). *Categories for the working mathematician.* Springer-Verlag.
Manes, E. (1969). A triple theoretic construction of compact algebras. In *Seminar on triples and categorical homology theory* (pp. 91–118). Springer Berlin Heidelberg.
Manes, E. G. (2002). Taut monads and t0-spaces. *Theor. Comput. Sci.*, *275*(1-2), 79–109.
Maraist, J., Odersky, M., Turner, D., & Wadler, P. (1999). Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theor. Comput. Sci.*, *228*(1-2), 175–210.
Mason, I. A., & Talcott, C. L. (1991). Equivalence in functional languages with effects. *J. Funct. Program.*, *1*(3), 287–327.
McLaughlin, C., McKinna, J., & Stark, I. (2018). Triangulating context lemmas. In *Proceedings of the 7th acm sigplan international conference on certified programs and proofs* (pp. 102–114).
Milner, R. (1977). Fully abstract models of typed *lambda*-calculi. *Theor. Comput. Sci.*, *4*(1), 1–22.
Milner, R. (1989). *Communication and concurrency.* Prentice Hall.
Milner, R. (1992). Functions as processes. *Mathematical Structures in Computer Science*, *2*(2), 119–141.
Milner, R., & Tofte, M. (1991). Co-induction in relational semantics. *Theor. Comput. Sci.*, *87*(1), 209–220.
Milner, R., Tofte, M., & Harper, R. (1990). *Definition of standard ML.* MIT Press.
Mitchell, J. C. (2002). *Concepts in programming languages.* Cambridge University Press.
Moggi, E. (1989). Computational lambda-calculus and monads. In *Proc. of LICS 1989* (pp. 14–23). IEEE Computer Society.
Moggi, E. (1991). Notions of computation and monads. *Inf. Comput.*, *93*(1), 55–92.
Morris, J. (1969). *Lambda calculus models of programming languages* (Unpublished doctoral dissertation). MIT.
Munkres, J. (2000). *Topology.* Prentice Hall, Incorporated.
Ong, C. (1988). *The lazy lambda calculus: An investigation into the foundations of functional programming.* University of London. Imperial College of Science and Technology.
Ong, C. L. (1993). Non-determinism in a functional setting. In *Proc. of LICS 1993* (pp. 275–286). IEEE Computer Society.
Park, D. (1981). Concurrency and automata on infinite sequences. In *Theoretical computer science* (pp. 167–183). Berlin, Heidelberg: Springer Berlin Heidelberg.
Petricek, T., Orchard, D. A., & Mycroft, A. (2014). Coeffects: a calculus of context-dependent computation. In *Proc. of ICFP 2014* (pp. 123–135).
Pitts, A. (2011). Howe's method for higher-order languages. In D. Sangiorgi & J. Rutten (Eds.), *Advanced topics in bisimulation and coinduction* (Vol. 52, pp. 197–232). Cambridge University Press.
Plotkin, G. (1973). *Lambda-definability and logical relations.* (Technical Report SAI-RM-4, School of A.I., University of Edinburgh)
Plotkin, G. (1975). Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, *1*(2), 125 - 159.
Plotkin, G. (1977). Lcf considered as a programming language. *Theoretical Computer Science*, *5*(3), 223 - 255.
Plotkin, G. D., & Power, J. (2001). Adequacy for algebraic effects. In *Proc. of FOSSACS 2001* (pp. 1–24).
Plotkin, G. D., & Power, J. (2002). Notions of computation determine monads. In *Proc. of FOSSACS 2002* (pp. 342–356).
Plotkin, G. D., & Power, J. (2003). Algebraic operations and generic effects. *Applied Categorical Structures*, *11*(1), 69–94.
Plotkin, G. D., & Pretnar, M. (2013). Handling algebraic effects. *Logical Methods in Computer Science*, *9*(4).
Pous, D., & Sangiorgi, D. (2012). Enhancements of the bisimulation proof method. In D. Sangiorgi &

J. Rutten (Eds.), *Advanced Topics in Bisimulation and Coinduction.* Cambridge University Press.
Pretnar, M. (2015). An introduction to algebraic effects and handlers. invited tutorial paper. *Electr. Notes Theor. Comput. Sci.*, *319*, 19–35.
Rabin, M. O., & Scott, D. S. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, *3*(2), 114–125.
Reed, J., & Pierce, B. (2010). Distance makes the types grow stronger: a calculus for differential privacy. In *Proc. of ICFP 2010* (pp. 157–168).
Reynolds, J. (1983). Types, abstraction and parametric polymorphism. In *IFIP congress* (pp. 513–523).
Rosenthal, K. (1990). *Quantales and their applications.* Longman Scientific & Technical.
Rutten, J. (1996). Elements of generalized ultrametric domain theory. *Theor. Comput. Sci.*, *170*(1-2), 349–381.
Rutten, J. J. M. M. (2000). Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, *249*(1), 3–80.
Sands, D. (1998). Improvement theory and its applications. In A. D. Gordon & A. M. Pitts (Eds.), *Higher Order Operational Techniques in Semantics* (pp. 275–306). Cambridge University Press.
Sangiorgi, D. (1992). The lazy lambda calculus in a concurrency scenario (extended abstract). In *Proceedings of the seventh annual symposium on logic in computer science (LICS '92), santa cruz, california, usa, june 22-25, 1992* (pp. 102–109).
Sangiorgi, D. (1993). A theory of bisimulation for the pi-calculus. In *CONCUR '93, 4th international conference on concurrency theory, hildesheim, germany, august 23-26, 1993, proceedings* (pp. 127–142).
Sangiorgi, D. (1994). The lazy lambda calculus in a concurrency scenario. *Inf. Comput.*, *111*(1), 120–153.
Sangiorgi, D. (2011). *Introduction to bisimulation and coinduction.* Cambridge University Press.
Sangiorgi, D., Kobayashi, N., & Sumii, E. (2011). Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, *33*(1), 5:1–5:69.
Sangiorgi, D., & Vignudelli, V. (2016). Environmental bisimulations for probabilistic higher-order languages. In *Proceedings of POPL 2016* (pp. 595–607).
Sangiorgi, D., & Walker, D. (2001). *The π-calculus - a theory of mobile processes.* Cambridge University Press.
Schrijver, A. (1986). *Theory of linear and integer programming.* New York, NY, USA: John Wiley & Sons, Inc.
Searcóid, M. Ó. (2006). *Metric spaces.* Springer London.
Sieber, K. (1992). Reasoning about sequential functions via logical relations. In J. P. T. Fourman M. P. & A. M. Pitts (Eds.), *Applications of categories in computer science* (Vol. 177, pp. 258–269). Cambridge University Press.
Simpson, A., & Voorneveld, N. (2018). Behavioural equivalence via modalities for algebraic effects. In *Proc. of ESOP 2018* (pp. 300–326).
Steen, L., & Seebach, J. (1995). *Counterexamples in topology.* Dover Publications.
Støvring, K., & Lassen, S. B. (2007). A complete, co-inductive syntactic theory of sequential control and state. In *Proceedings of POPL 2007* (pp. 161–172).
Strassen, V. (1965). The existence of probability measures with given marginals. *Ann. Math. Statist.*, *36*(2), 423–439.
Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, *5*(2), 285–309.
Thijs, A. (1996). *Simulation and fixpoint semantics.* Rijksuniversiteit Groningen.
Turing, A. M. (1937). Computability and λ-definability. *The Journal of Symbolic Logic*, *2*(4), 153–163.
Van Breugel, F., Mislove, M., Ouaknine, J., & Worrell, J. (2005). Domain theory, testing and simulation for labelled markov processes. *Theor. Comput. Sci.*, *333*(1-2), 171–197.
Vickers, S. (1996). *Topology via logic.* Cambridge University Press.
Villani, C. (2008). *Optimal transport: Old and new.* Springer Berlin Heidelberg.

Xu, L., Chatzikokolakis, K., & Lin, H. (2014). Metrics for differential privacy in concurrent systems. In *Formal techniques for distributed objects, components, and systems - 34th IFIP WG 6.1 international conference, FORTE 2014, held as part of the 9th international federated conference on distributed computing techniques, discotec 2014. proceedings* (pp. 199–215).