UNIVERSITY OF LJUBLJANA
FACULTY OF MATHEMATICS AND PHYSICS
DEPARTMENT OF MATHEMATICS

Mathematics – 3rd cycle

Niels Voorneveld

EQUALITY BETWEEN PROGRAMS WITH EFFECTS

Doctoral dissertation

Adviser: prof. dr. Alex Simpson

Ljubljana, 2019

UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO
ODDELEK ZA MATEMATIKO

Matematika – 3. stopnja

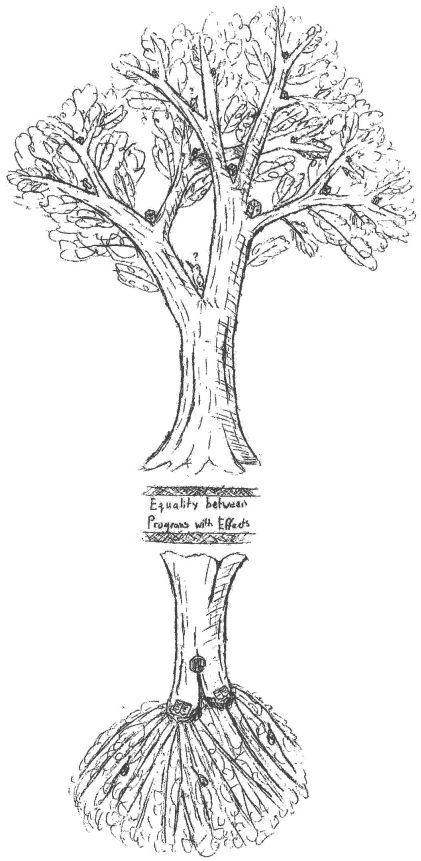
Niels Voorneveld

ENAKOST MED PROGRAMI Z UČINKI

Doktorska disertacija

Mentor: prof. dr. Alex Simpson

Ljubljana, 2019



Acknowledgements

I am very grateful to my supervisor, Alex Simpson, whose guidance was of paramount importance to my research, and who helped me in all of my academic writings. I would like to thank the other members of my thesis committee, Ugo Dal Lago and Marko Petkovšek, for their evaluation of my thesis. Thanks to Philipp Haselwarter and Brett Chenoweth who, as my officemates, I could always bother with an odd question or two. Thanks to Andrej Bauer, Matija Pretnar and Žiga Lukšič, for further helpful discussions and guidance, and especially Anja Petković for helping me with Slovene translations. Thanks to the academic visitors to Ljubljana throughout the years, including Francesco Gavazzo, Aliaume Lopez, and my master's supervisor Jaap van Oosten, whose insights helped me a lot. Last but not least, I would like to thank my parents, Ellen and Frits Voorneveld, for all their support, and the rest of my family whose constant stream of photos never failed to lift my mood. Of course, thanks to all whom I failed to mention, and to the rest of humanity for contributing to this marvellous world.

This dissertation is based upon work supported by the Air Force Office of Scientific Research under award numbers FA9550-14-1-0096 and FA9550-17-1-0326. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.

Abstract

This thesis studies notions of program equivalence for a call-by-push-value functional language with algebraic effects and general recursion. We mainly focus on behavioural equivalence, where program behaviour is specified by a collection of effect-specific formulas. Two programs of the same type are deemed equivalent if they satisfy the same formulas. To interpret effectful behaviour in a generic way, computation terms are evaluated to trees built from effect operators. These trees are then interpreted in a logic using modalities, which lift predicates on value types to predicates on computation types.

One of the main contributions of this thesis is identifying conditions on the modalities under which the behavioural equivalence induced by the logic is a congruence. This means equivalent terms cannot be distinguished by programs. To prove this property, we show that the behavioural equivalence coincides with an appropriate notion of applicative bisimilarity, where effects are interpreted using relators (which lift relations). This allows us to prove the aforementioned congruence using a variation of Howe's method.

The algebraic effects to which the results apply include error, nondeterminism, probability, global store, input/output, and timer. Several combinations of these effects can also be described with the logic. However, in order to combine effects more easily, and to give more natural descriptions of program behaviour, the logic is generalised to a logic with quantitative formulas. Once again, the congruence property and connections with applicative bisimilarity are established.

Finally, we show that similar results hold also if the language is extended with additional type constructors. In particular, we consider universal polymorphic and recursive types.

Math. Subj. Class. (2010): 68Q55, 03B70, 06B35, 68Q10, 68Q85

Keywords: Program equivalence, functional programming, call-by-push-value, behavioural logic, modalities, algebraic effects, applicative bisimilarity, Howe's method, complete lattices

Izvleček

Disertacija preučuje pojem enakovrednosti programov za funkcijski programski jezik z algebrajskimi učinki in splošno rekurzijo, ki uporablja klic po naloženi vrednosti (call-by-push-value oz. na kratko cbpv). Osredotočamo se predvsem na vedenjsko ekvivalenco, pri čemer je obnašanje programa določeno z zbirko formul, ki so odvisne od učinkov. Dva programa istega tipa imamo za enakovredna, če zadoščata istim formulam. Za splošno interpretacijo obnašanja programa z učinki izračune ovrednotimo z drevesi, zgrajenimi iz učinkovnih operacij. Ta drevesa potem interpretiramo v logiki z modalnostmi, kar dvigne predikate na tipih vrednosti do predikatov na tipih izračunov.

Eden glavnih prispevkov disertacije je določitev pogojev na modalnostih, pri katerih je vedenjska ekvivalenca, ki jo inducira logika, kongruenca. To pomeni, da enakovrednih izrazov ni mogoče razlikovati s programi. Za dokaz te lastnosti pokažemo, da vedenjska ekvivalenca sovpada z ustreznim pojmom aplikativne bipodobnosti, kjer učinke interpretiramo z uporabo relatorjev (ki dvigajo relacije). To nam omogoči, da dokažemo omenjeno kongruenco s pomočjo različice Howejeve metode.

Algebrajski učinki, za katere veljajo rezultati, vključujejo napake, nedeterminizem, verjetnost, globalni pomnilnik, vhod/izhod in časomer. Z logiko lahko opišemo tudi različne kombinacije teh učinkov. Da bi lažje kombinirali učinke in naravneje opisovali obnašanje programov, posplošimo logiko na kvantitativno logiko. Tudi tu pokažemo lastnost kongruence in povezavo z aplikativno bipodobnostjo.

Na koncu pokažemo, da podobni rezultati veljajo tudi, če jezik razširimo z dodatnimi konstruktorji tipov. Posebej preučimo univerzalne polimorfne in rekurzivne tipe.

Math. Subj. Class. (2010): 68Q55, 03B70, 06B35, 68Q10, 68Q85

Ključne besede: Enakovrednost programov, funkcijsko programiranje, klic po naloženi vrednosti (call-by-push-value), vedenjska logika, modalnosti, algebrajski učinki, aplikativna bipodobnost, Howejeva metoda, polne mreže

Contents

1

Introduction

1.1 Foreword

Not everything is as you imagine it to be. Human creations, conjured in the confines of the brain, may behave differently when brought into existence. Nowhere has this become as apparent as in the field of computer science. Coders meticulously craft tools to aid their fellow human beings. But once downloaded on the users phone, all manner of different things may alter the apps behaviour. Can the downloaded app on the users phone still be considered the same as the original code on the designer's laptop?

This depends on a variety of circumstances. What phone is the user using? What operating system? Is it connected to the internet? Was it dropped on the floor recently? The real world has many ways to effect the behaviour of computer programs. When you type in a calculator '2+2=' it will say '4', unless the calculator ran out of batteries, or a dog mistook it for a biscuit. A calculator fresh from the box is surely different from a calculator which has been chewed in half.

Once you are aware of such effects the world can have on a program, there is a choice to be made. Do you avoid the effects as much as you can, or do you harness them for your own gain? Whatever decision you make, real world effects are a fact of life. So in order to verify the behaviour of, and equality between, programs with effects, one needs to understand how effects influence program behaviour.

This understanding needs to be built on a suitable level of abstraction. Fundamentally, we can consider a program to be a 'function', something that takes an input and computes from it an output. Effects may then be considered as any real world influence which will make the program behave 'non-functionally'. This can be anything; something that stops the program from outputting a result, a potential random influence which makes the program output different results at different runs, or some interaction with an outside memory source like the internet.

In order to understand the behaviour of the program, one could measure under what circumstances and how often the program outputs a result with certain properties. Such measurements can be conceptualised with the notion of *observation*, which are formalised using *modalities*. In case of some random interference, such a modality might be constructed to measure the frequency at which certain things are outputted

by the program.

We consider two different programs to be equal if there is no measurement, e.g., a modality, which can distinguish the two. These measurements are abstractly defined, and as such need to be reasonably formulated to express actual things which the users of the program can observe. An important property the resulting notion of equality needs to satisfy, is that no other program can distinguish between two equal programs. Establishing this for different effects and languages is a main contribution of this thesis.

1.2 Technical introduction

The title of this dissertation, "Equality between Programs with Effects", is quite general. Therefore, as an introduction to the material presented in this thesis, we give an outline of what exactly is meant by the three concepts mentioned in it.

Equality: There are many different notions of equality between programs in computer science. Such equalities are normally called *equivalences*, where the word 'equality' is often-times reserved for *syntactic equality*: two programs are equal if their codes are exactly the same. There are however multiple ways to code the same 'function', abstracting away things like execution time. No one notion of equality could be considered the 'be-all and end-all', which is why we will talk about 'a' notion of equivalence, instead of 'the' notion of equality.

Firstly, we consider the notion of *denotational equivalence*, transporting a program from its real world implementation to an abstract space of mathematical denotations. These are mathematical models describing a program as the user might imagine it, a black box, a function, an abstract representation of the program. If the representations of two programs are equal, we consider the two programs denotationally equivalent. Programs which receive inputs and return outputs can for instance be interpreted as continuous mathematical functions, which is done in *domain theory* [93, 98]. Alternatively, one can consider as denotation functions that are realized by codes, as in realizability [48, 104], or consider strategies of a game as denotation, as done in game semantics [4, 32].

Another notion of equivalence is *bisimilarity* [27, 61, 71]. Two programs are related if there is a simulation relating the two. Such a simulation needs to satisfy some local properties of behaviour, doing similar modifications on a pair of programs related by the simulation must result in a new pair related by the simulation. Bisimilarity itself is then defined coinductively, as the largest symmetric simulation satisfying the appropriate properties. A particular version of bisimilarity of interest in this thesis is applicative bisimilarity [2], where the main modification tested by simulations are applications to arguments, whilst abstracting away evaluation time of programs.

Thirdly, there is the notion of *contextual equivalence* [61, 62, 64]. When a program outputs basic data, like natural numbers, it is easy to see when two such programs are equal; which is if they output the same thing. We can define *observations* for such programs, e.g., the program outputs the number 3. Programs of more complicated types

can be tested at such basic data types, by plugging them into a *context*, a program of a basic type with a hole. Two programs are contextually equivalent if within each context they satisfy the same observations. This is the largest 'reasonable' notion of equivalence, since it is the largest relation which is both preserved under *program composition*, and satisfies basic observations. It is also the most mysterious notion of equivalence, since its formulation does not give a clear description of the equivalence at higher types.

The main focus of this thesis however, will be the fourth and final notion, *logical equivalence*. This notion has primarily been developed in concurrency theory [27]. Using the notion of observation at basic types used for contextual equivalence as a foundation, we define observations for any type of programs. These describe behavioural properties of programs, and are defined as formulas in a behavioural logic. Two programs of the same type are equivalent if they satisfy the same formulas of that type. Unlike contexts with observations, these formulas give more intuitive tests on programs, like applying a program to some argument, akin to what is done in applicative bisimilarity.

Effects: The matter of defining equivalences becomes more complicated once effects are involved. If the behaviour of a program is altered by an outside influence, such that it for instance might output different things on different evaluations, observations used for effect-free programs become insufficient for interpreting program behaviour. Observations and denotational models of programs need to be adapted to account for the way the program can be affected by the real world.

A program can for instance be affected by some random process, introducing *probability*, in which case the likelihood that a certain result is produced becomes relevant. A program might interact with some outside memory source, a *global store*, in which case what is stored there becomes important, both before and after the program is evaluated. Maybe the program communicates to the human user, with *input-output*, in which case any possible line of communication with the user must be taken into account. It can also be that whatever the program interacts with is completely unpredictable, *nondeterministic*, so one can only check what is and is not possible.

The equivalence needs to be modelled to suit the situation. In denotational models, we can for instance change what we consider as outputs the program can produce. In case of nondeterminism [79, 97] for instance, sets of results are considered, whereas for probability [36] it is distributions over results. A general way to describe effects in such a denotational way is with the use of monads [63, 106]. Efforts have also been exerted to adapt the notion of applicative bisimilarity to different effects [12, 16, 41], with recently an important formulation for generic effects using monads and relators [14].

In the case of contextual equivalence, we can alter the notion of observations on basic types, inspired by the denotational models described above. For instance, in [35] observations are defined on the type of natural numbers, and using the method of top-top closure [35, 75] properties are established for the resulting contextual equivalence. Inspired by such effectful observations, we define the notion of *modality* for lifting predicates and defining behavioural properties for higher types. These modalities are the foundation for the formulation of logical equivalence used in this thesis.

Programs: To study the notions of equivalence in sufficient detail, we need to choose a specific programming language. This language is chosen to feature a wide range of program behaviour, but is otherwise kept as simple as possible in order to focus on the interesting details. Fundamental to this language is the inclusion of higher-order programs, allowing us to construct programs which use other programs as input. Moreover, we like the language to be Turing-complete, which can be accomplished by including general recursion. This allows us to code up any function which is computable in a mathematical sense.

We could consider Plotkin's PCF [80], a functional programming language based on Scott's logic for computable functions [25, 94]. The behaviour of such a programming language based on lambda-calculus is heavily dependent on the strategy used to evaluate terms. When applying a function term to an argument, which can be a program, this argument can either first be evaluated or directly be substituted in the function term. The choice of such evaluation strategies, respectively called call-by-value (CBV) and call-by-name (CBN), becomes vital once either divergence or effects are present in the language. In order to study both aspects of effect behaviour, we use Levy's call-by-push-value (CBPV) [42, 43] with general recursion as our vehicle of study.

We add algebraic effects [82, 84] to the language, representing effects concretely as algebraic operators. For instance, $\mathsf{pr}(\underline{M}, \underline{N})$ is a program which either continues evaluation with $\underline{M}$ or $\underline{N}$, with equal probability. So we see $\mathsf{pr}(-,-)$ as a probabilistic algebraic operators on programs of the language. In order to keep our treatment of effects as generic as possible, the behaviour of effects will however not be directly implemented in the formulation of program evaluation, the operational semantics of the language. The behaviour will instead be specified in the logic, using the aforementioned modalities.

Call-by-push-value with general recursion and algebraic effects will form the concrete basis of our studies. We will however consider extensions of this language separately. Such staples of functional languages like universal type polymorphism and type recursion will also be studied. With such extensions, we hope the language is general enough to cover a wide range of (real-world) variations of programming languages.

1.3 Contributions

Behavioural equivalence: In this thesis, we study the notion of logical equivalence, where two programs are equated when they satisfy the same formulas. Programs can be seen as models which may or may not satisfy predicates describing patterns of program behaviour. We will focus in particular on classifying the correct formulas, such that they can be used to formulate behavioural properties from the literature, and such that the resulting logical equivalence between programs has the right properties. We call such a logical equivalence induced by formulas describing behavioural properties, the *behavioural equivalence*.

Fundamental to the set-up is how we treat effects. When a program is evaluated, if an effect is encountered, the effect operator is marked down and all possible continuations

are taken into consideration. As such, we see a program of a *producer* type, containing computations which can produce a result, as constructing what is called an *effect tree*, whose leaves are given by values the program can produce, and whose nodes are given by effect operators. We then use *modalities* to lift formulas on return values to formulas on programs of the producer type. The interpretation of effects is completely specified by such modalities, and other formula constructors are there to capture other aspects of call-by-push-value programs.

A main result of this thesis is given in Theorem 3.3.8 (The Compatibility Theorem), which states: given that the set of modalities satisfies certain properties, the resulting logical equivalence is preserved under program composition. This partially motivates the relevance of the logical equivalence, and shows it is a good notion of program equivalence. Contributions of this thesis include the identification of those sufficient properties on modalities, dealing with *continuity* and *sequencing*, together with the proof of the theorem itself. It is particularly interesting that numerous examples of algebraic effects can be described by suitable *unary* modalities, where by 'suitable' we mean they satisfy the properties required by the theorem.

Some combinations of effects however, like nondeterminism with probability or global store, cannot be described by such modalities. This can be proven by studying which equational theories can be described by suitable modalities. To remedy this problem, we generalise the logical equivalence to use quantitative formulas. These express behavioural properties which can be satisfied to a certain quantitative degree. In some ways, they more naturally describe properties of effectful programs. For instance, they can describe the probability that a program terminates or the set of suitable starting states required for termination. Modalities for such a quantitative logic are then quantitative themselves, and the theorem from before can be generalised appropriately to Theorem 6.3.15 (The Generalised Compatibility Theorem).

Relationships between equivalences: Though we focus on one notion of program equivalence, its relationship to other notions of equivalence is important. For instance, if we use modalities to specify observations for contextual equivalence, Theorems 3.3.8 and 6.3.15 (The Compatibility Theorems) imply that the logical equivalence is included in the corresponding contextual equivalence. We could also potentially relate the logical equivalence with denotational equivalence, by noting that the modalities defined on effect trees (the free monad) in a sense capture the behavioural aspects of the more general monads used in some denotational models. But this is not done in this thesis.

The most important relationship established in the thesis is the relationship between logical equivalence and applicative bisimilarity. Using the modalities, a *relator* can be defined in the sense of [14]. This allows us to define a notion of applicative bisimilarity for the call-by-push-value language with effects, which coincides with the logical equivalence. Establishing this connection between two prominent notions of program equivalence (Theorem 4.2.8, The Coincidence Theorem) is one of the main contributions of this thesis. Moreover, this connection allows us to prove the Compatibility Theorems using a variation on Howe's method [31, 44], as done in [14].

Other contributions of this thesis involve variations on the logical equivalence. First and foremost, we can choose to include or exclude negation in our logic. This may change the resulting logical equivalence, in particular when it is used to describe non-deterministic effects. The logical equivalence in the absence of negation coincides with a notion of mutual applicative similarity. Other variations on the logic can be carried out without changing the resulting logical equivalence. This includes a 'pure' variation, where the formulas do not reference program terms directly. Other variations entail reducing the class of formulas without changing the equivalence. In particular, by default we close the formulas under countable conjunction and disjunction, which can be changed to finitary (or binary) conjunction and disjunction for most examples of effects.

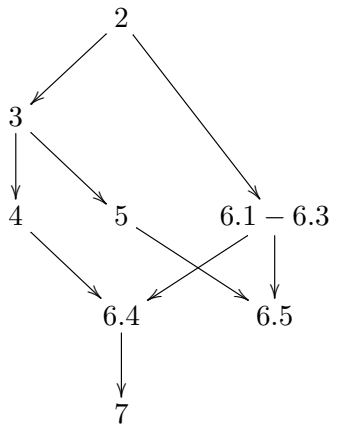


Figure 1.1: Chapter dependencies

Overview: In Chapter 2 we set up the base language of study, a call-by-push-value lambda calculus with general recursion and algebraic effects. We also define its operational semantics, including the use of effect trees constructed with the algebraic effect operators.

In Chapter 3 we define the formulas used to describe behavioural properties of the language, and in particular the modalities used to describe each effect. We formulate Theorem 3.3.8 (The Compatibility Theorem), and define the properties on modalities used in the theorem. The chapter ends with a study of equational theories resulting from such modalities.

Chapter 4 establishes the aforementioned connection with applicative bisimilarity via the Coincidence Theorems 4.2.7 and 4.2.8, together with an explicit proof of the Compatibility Theorem 3.3.8, using Howe's method.

Chapter 5 is used to explore the variations on the logic discussed before, together with which combinations of effects are possible in a Boolean logic with unary modalities.

In Chapter 6 we generalise the logic from Chapter 3 to a quantitative logic, capable of describing more examples of (combinations of) effects. It also generalises other parts of the thesis. In particular, Sections 6.4 and 6.5 partially generalise Chapters 4 and 5 respectively.

Last but not least, Chapter 7 extends the language and logic to include universal polymorphic and recursive types, and Chapter 8 concludes the thesis with some related work and potential topics for future research.

A lot of the material covered in this thesis has been featured in previous publications. The conference paper [95] and its journal version [96], co-written with Alex Simpson, contains most material from Chapters 3 and 4, though there the underlying language is a more modest fine-grained call-by-value lambda calculus (with algebraic effects and general recursion). The paper [105] describes most of Chapter 6, and uses the same underlying CBPV language as defined in Chapter 2 of this thesis. Material in this dissertation which has not yet been published before includes: Section 3.5, most of Chapter 5, Section 6.5 and Chapter 7, together with minor extra results spread out throughout the thesis.

1.4 Published papers

Below is a list of the author's publications, on the basis of which this thesis has been prepared:

[96] Alex Simpson and Niels Voorneveld. 2019. Behavioural Equivalence via Modalities for Algebraic Effects. *ACM Trans. Program. Lang. Syst.* 42, 1, Article 4 (2020), 45 pages.
`https://doi.org/10.1145/3363518`

[105] Niels Voorneveld. Quantitative logics for equivalence of effectful programs. *In Proc. of MFPS XXXV (Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics)*, ENTCS. Elsevier, 2019. To appear.

[104] Niels Voorneveld. Non-deterministic Effects in a Realizability Model. *Electronic Notes in Theoretical Computer Science.* 336 : 299 - 314, 2018. (MFPS XXXIII).
`https://doi.org/10.1016/j.entcs.2018.03.029`.

[95] Alex Simpson and Niels Voorneveld. Behavioural equivalence via modalities for algebraic effects. In *Programming Languages and Systems (ESOP 2018)*, pages 300-326, 2018.
`https://link.springer.com/book/10.1007%2F978-3-319-89884-1`

2

Language and operational semantics

In this chapter we give the core functional language considered in this thesis. We look at a general language, incorporating different evaluation strategies and a variety of different data types. To keep the technical details clear and concise, we focus on a simply typed language for now. But later on in Chapter 7 we will go beyond simple types, and consider possible parametric language extensions.

Two of the main features of the language are algebraic effects and general recursion. In the presence of these features, as discussed in the introduction, the particular way of resolving function application becomes fundamental for our interpretation of program behaviour. This amounts to the following question:

When we apply a functional lambda term to an argument, do we evaluate that argument before we substitute it? Or do we substitute the argument as is?

The two options correspond to two different reduction strategies, specifically call-by-value (CBV) and call-by-name (CBN) [47, 78, 80] respectively. This choice becomes important when, for instance, the argument diverges or invokes other effects.

Take for instance the following example. Let $\mathsf{write}(\overline{n}; \underline{M})$ be a computation which prints the numeral n on the screen, after which it continues with the evaluation of the computation $\underline{M}$. We look at the following computation:

$$(\lambda x.\, \mathsf{write}(\overline{0}; x))\ \ \mathsf{write}(\overline{1}; \mathsf{return}(*))$$

Under the CBN reduction strategy, '0 1' is printed on the screen, whereas with CBV '1 0' is printed. We desire a language which accommodates both these attitudes towards effects.

In order to incorporate both behaviours, we choose to use Paul Levy's call-by-push-value language as in [42, 43]. This language combines CBV and CBN terms by translating them into different call-by-push-value terms of different types.

2.1 Effect-free core language

We give an overview of the language and its semantics. The types are divided into two categories, *value types* and *computation types*. Value types contain value terms that are

passive; they will not compute anything on their own and can be substituted into other terms. Computation types contain computation terms which are *active*, which means they either start evaluating or wait for an input.

Value types consist of:

$$\mathbf{A}, \mathbf{B} \ ::= \ \mathbf{U}\,\underline{\mathbf{C}} \mid \mathbf{1} \mid \mathbf{N} \mid \mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i \mid \mathbf{A} \times \mathbf{B}$$

where I is any *finite* indexing set[1] as in [43]. In [42], infinitary indexing sets are used.

$\mathbf{U}\,\underline{\mathbf{C}}$ is a *thunk type*, which consists of computation terms which are 'frozen'. These terms were initially computation terms but are made inactive by wrapping them into a *thunk*. $\mathbf{1}$ is the unit type, only containing one value $*$. This type forms a foundation for testing behavioural properties. $\mathbf{N}$ is the type of natural numbers, containing the non-negative integers. With this type, we can program any computable function on the natural numbers like in PCF [80]. Lastly, $\mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i$ is the sum type over a finite collection of types and $\mathbf{A} \times \mathbf{B}$ contains pairs of values.

The computation types consist of:

$$\underline{\mathbf{C}}, \underline{\mathbf{D}} \ ::= \ \mathbf{F}\,\mathbf{A} \mid \mathbf{A} \to \underline{\mathbf{C}} \mid \mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i$$

with I again being a finite indexing set. $\mathbf{F}\,\mathbf{A}$ is a *producer* type. When terms of this type are evaluated, they perform a computation which may invoke effects and either produces a value term of type $\mathbf{A}$ or diverges (the evaluation procedure never terminates). $\mathbf{A} \to \mathbf{B}$ is the type of functions, which is considered a computation type since its terms are actively awaiting an input, mainly the argument of the function. Lastly, $\mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i$ is the product type over a finite collection of types. Unlike $\mathbf{A} \times \mathbf{B}$, this type combines computations and is considered active, awaiting an index i from the set I as argument, before continuing its computation.

Note that there is a mismatch between the binary pair type constructor $\mathbf{A} \times \mathbf{B}$ and the indexed product type constructor $\mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i$, since we follow the type system presented in [42]. Alternatively, we could replace the indexed type constructors by binary type constructors $\mathbf{A} + \mathbf{B}$, $\mathbf{A}$ & $\mathbf{B}$ and the empty type $\mathbf{0}$. Vice versa, we could replace the pair type constructor by an indexed version. This change will not make any significant difference.

A term's type also depends on its *context* denoting which variables (from some countable list of variables $x, y, z, \dots$) the term may refer to. Contexts consist of sequences of distinct variables of value types:

$$\Gamma \ ::= \ \emptyset \ \mid \ \Gamma, x : \mathbf{A} \ .$$

The contexts only contain value types, meaning that we can only ever substitute value terms for variables. Despite this restriction, we can still substitute computation terms by turning them into values of a thunk type.

[1] It is of course natural to have finite indexing sets for sum and product types, as it produces a finite syntax. As a result, the sets of terms are countable, which has technical benefits as it is helpful in the formulation of our logic and the Howe's method used in Chapter 4.

The value terms consist of:

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

Whereas the computation terms contain:

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

In several terms, variables are being bound: $\mathsf{case}\ V\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\}$ and $\underline{M}\ \mathsf{to}\ x.\, \underline{N}$ bind x in $\underline{N}$, and both $\mathsf{let}\ x\ \mathsf{be}\ V.\, \underline{M}$ and $\lambda x.\, \underline{M}$ bind x in $\underline{M}$, whereas $\mathsf{pm}\ V\ \mathsf{as}\ \{\ldots, (i.x).\underline{M}_i, \ldots\}$ binds x in $\underline{M}_i$, and $\mathsf{pm}\ V\ \mathsf{as}\ (x, y).\underline{M}$ binds x and y in $\underline{M}$. Terms are identified up to α-equivalence ([7, 77]), allowing us to change variable names when necessary.

We write $\underline{\underline{\mathbf{E}}}, \underline{\underline{\mathbf{F}}}$ for general types, which may be either value or computation types. We write $\underline{\underline{P}}, \underline{\underline{R}}, \underline{\underline{Q}}$ for general terms, which can either be value or computation terms.

The typing rules for the terms are given in Figure 2.1. There are two typing judgments, a value typing rule $\Gamma \vdash V : \mathbf{A}$ which says V is a value of type $\mathbf{A}$ with context Γ, and a computation typing rule $\Gamma \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}$ which says $\underline{M}$ is a computation of type $\underline{\mathbf{C}}$ with context Γ. We write $\Gamma \mathrel{\underline{\underline{\vdash}}} \underline{\underline{P}} : \underline{\underline{\mathbf{E}}}$ in case the type of the term is generic (i.e. value or computation). A term is called *closed* if it is typed in the empty context. For any type $\underline{\underline{\mathbf{E}}}$, we write $\mathit{Terms}(\underline{\underline{\mathbf{E}}})$ for the set of closed terms of type $\underline{\underline{\mathbf{E}}}$.

Lemma 2.1.1. *If* $\Gamma \mathrel{\underline{\underline{\vdash}}} \underline{\underline{P}} : \underline{\underline{\mathbf{E}}}$ *and* x *is not mentioned in* $\underline{\underline{P}}$ *or* Γ*, then* $x : \mathbf{A}, \Gamma \mathrel{\underline{\underline{\vdash}}} \underline{\underline{P}} : \underline{\underline{\mathbf{E}}}$.

This can be proven by a routine induction on the typing judgements.

Given a general term $\underline{\underline{P}}$, a value term V and a variable x, we write $\underline{\underline{P}}[V/x]$ for the substitution of V for any free occurrence of x in $\underline{\underline{P}}$. We have the following result which can be established by a routine induction.

Lemma 2.1.2. *If* $\Gamma, x : \mathbf{A} \mathrel{\underline{\underline{\vdash}}} \underline{\underline{P}} : \underline{\underline{\mathbf{E}}}$, $\Gamma \vdash V : \mathbf{A}$, *and* x *is not bound in* $\underline{\underline{P}}$, *then* $\Gamma \mathrel{\underline{\underline{\vdash}}} \underline{\underline{P}}[V/x] : \underline{\underline{\mathbf{E}}}$.

2.1.1 Operational semantics

Computation terms are active, and may evaluate if they are not in their terminal form. What they evaluate to must be specified by certain rules. We give the semantics of this language by specifying a reduction strategy in the style of a CK-machine [18, 42].

We distinguish a special class of computation terms, called terminal computation terms. They are the terms which will not reduce further.

Definition 2.1.3. A *terminal computation term* is a computation term of the form:

$$\mathsf{return}(V), \quad \lambda x.\, \underline{M}, \quad \text{or} \quad \langle \underline{M}_i \mid i \in I \rangle$$

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

$$\frac{\Gamma \vdash V : \mathbf{N} \qquad \Gamma \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}} \qquad \Gamma, x : \mathbf{N} \mathrel{\underline{\vdash}} \underline{N} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{case}\ V\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} : \underline{\mathbf{C}}}$$

$$\frac{}{\Gamma, x : \mathbf{A}, \Gamma' \vdash x : \mathbf{A}} \qquad \frac{\Gamma \vdash V : \mathbf{A} \qquad \Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{let}\ x\ \mathsf{be}\ V.\underline{M} : \underline{\mathbf{C}}}$$

$$\frac{\Gamma \vdash V : \mathbf{A}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{return}(V) : \mathbf{F}\,\mathbf{A}} \qquad \frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} : \mathbf{F}\,\mathbf{A} \qquad \Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{N} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \underline{M}\ \mathsf{to}\ x.\underline{N} : \underline{\mathbf{C}}}$$

$$\frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \vdash \mathsf{thunk}(\underline{M}) : \mathbf{U}\,\underline{\mathbf{C}}} \qquad \frac{\Gamma \vdash V : \mathbf{U}\,\underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{force}(V) : \underline{\mathbf{C}}}$$

$$\frac{\Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \lambda x.\underline{M} : \mathbf{A} \to \underline{\mathbf{C}}} \qquad \frac{\Gamma \vdash V : \mathbf{A} \qquad \Gamma \mathrel{\underline{\vdash}} \underline{M} : \mathbf{A} \to \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \underline{M}\ V : \underline{\mathbf{C}}}$$

$$\frac{j \in I \quad \Gamma \vdash V : \mathbf{A}_j}{\Gamma \vdash (j, V) : \mathbf{\Sigma}_{i \in I}\,\mathbf{A}_i} \qquad \frac{\Gamma \vdash V : \mathbf{\Sigma}_{i \in I}\,\mathbf{A}_i \qquad \Gamma, x : \mathbf{A}_i \mathrel{\underline{\vdash}} \underline{M}_i : \underline{\mathbf{C}} \ \text{ for each } \ i \in I}{\Gamma \mathrel{\underline{\vdash}} \mathsf{pm}\ V\ \mathsf{as}\ \{\ldots, (i.x).\underline{M}_i, \ldots\} : \underline{\mathbf{C}}}$$

$$\frac{\Gamma \vdash V : \mathbf{A} \qquad \Gamma \vdash W : \mathbf{B}}{\Gamma \vdash (V, W) : \mathbf{A} \times \mathbf{B}} \qquad \frac{\Gamma \vdash V : \mathbf{A} \times \mathbf{B} \qquad \Gamma, x : \mathbf{A}, y : \mathbf{B} \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{pm}\ V\ \mathsf{as}\ (x, y).\underline{M} : \underline{\mathbf{C}}}$$

$$\frac{\Gamma \mathrel{\underline{\vdash}} \underline{M}_i : \underline{\mathbf{C}}_i \ \text{ for each } \ i \in I}{\Gamma \mathrel{\underline{\vdash}} \langle \underline{M}_i \mid i \in I \rangle : \mathbf{\Pi}_{i \in I}\,\underline{\mathbf{C}}_i} \qquad \frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} : \mathbf{\Pi}_{i \in I}\,\underline{\mathbf{C}}_i \qquad j \in I}{\Gamma \mathrel{\underline{\vdash}} \underline{M}\ j : \underline{\mathbf{C}}_j}$$

$$\frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} : \mathbf{U}\,\underline{\mathbf{C}} \to \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{fix}(\underline{M}) : \underline{\mathbf{C}}}$$

Figure 2.1: Typing rules

For most computation types $\underline{\mathbf{C}}$, there are closed terminal computation terms $\underline{M}$ such that $\underline{M} \in \mathit{Terms}(\underline{\mathbf{C}})$. We denote by $\mathit{Tct}(\underline{\mathbf{C}})$ the set of such terms of type $\underline{\mathbf{C}}$. Note for instance that $\mathit{Tct}(\mathbf{F}\,\mathbf{A}) = \{\mathsf{return}(V) \mid V \in \mathit{Terms}(\mathbf{A})\}$.

We first give the rules for terms we can directly reduce. These correspond to β-reduction rules, where the continuation of the computation is apparent.

Definition 2.1.4. We define the relation $\rightsquigarrow$ on closed computation terms, which gives a partial function $\mathit{Terms}(\underline{\mathbf{C}}) \to \mathit{Terms}(\underline{\mathbf{C}})$ according to the following rules:

1. $\mathsf{case}\ \mathsf{Z}\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} \ \rightsquigarrow\ \underline{M}$
2. $\mathsf{case}\ \mathsf{S}(V)\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} \ \rightsquigarrow\ \underline{N}[V/x]$
3. $\mathsf{let}\ x\ \mathsf{be}\ V.\underline{M} \ \rightsquigarrow\ \underline{M}[V/x]$

4. $\mathsf{force}(\mathsf{thunk}(\underline{M})) \rightsquigarrow \underline{M}$
5. $\mathsf{pm}\ (j, V)\ \mathsf{as}\ \{\ldots, (i.x).\underline{M}_i, \ldots\} \rightsquigarrow \underline{M}_j[V/x]$
6. $\mathsf{pm}\ (V, W)\ \mathsf{as}\ (x, y).\underline{M} \rightsquigarrow \underline{M}[V/x, W/y]$
7. $\mathsf{fix}(\underline{M}) \rightsquigarrow \underline{M}\ \mathsf{thunk}(\mathsf{fix}(\underline{M}))$

Given a routine induction we can establish the following result.

Lemma 2.1.5. *If* $\vdash \underline{M} : \underline{\mathbf{C}}$ *and* $\underline{M} \rightsquigarrow \underline{N}$, *then* $\vdash \underline{N} : \underline{\mathbf{C}}$.

Sometimes, the reduction of a subterm is required in order to continue the evaluation of the term. For instance in the case of $\underline{M}\ V$, we need to first reduce the subterm $\underline{M}$ to a terminal like $\lambda x.\, \underline{N}$ before we can apply it to V. To handle this, we define *stacks* which are used to express future continuations of the computation. Such stacks have a similar roles as evaluation contexts where the term considered for evaluation is highlighted [19, 107]. The stacks are given by:

$$S, Z \quad ::= \quad \varepsilon \ \mid\ S \circ ((-)\ \mathsf{to}\ x.\, \underline{M}) \ \mid\ S \circ ((-)\ V) \ \mid\ S \circ ((-)\ j)\ ,$$

where j is an element of an indexing set for product types. We denote by $S@Z$ the result of appending stack S with stack Z. A stack accepts terms of one computation type, and returns a term of another computation type. We can denote the types of stacks by $S : \underline{\mathbf{C}} \Rightarrow \underline{\mathbf{D}}$ where:

$\varepsilon : \underline{\mathbf{C}} \Rightarrow \underline{\mathbf{C}}$.
If $S : \underline{\mathbf{C}} \Rightarrow \underline{\mathbf{D}}$ and $x : \mathbf{A} \vdash \underline{M} : \underline{\mathbf{C}}$, then $S \circ (-)\ \mathsf{to}\ x.\, \underline{M} : \mathbf{F}\,\mathbf{A} \Rightarrow \underline{\mathbf{D}}$.
If $S : \underline{\mathbf{C}} \Rightarrow \underline{\mathbf{D}}$ and $V : \mathbf{A}$, then $S \circ (-)\ V : (\mathbf{A} \to \underline{\mathbf{C}}) \Rightarrow \underline{\mathbf{D}}$.
If $S : \underline{\mathbf{C}} \Rightarrow \underline{\mathbf{D}}$ and $\underline{\mathbf{C}}_j = \underline{\mathbf{C}}$, then $S \circ (-)\ j : \mathbf{\Pi}_{I \in \underline{\mathbf{C}}_j} \Rightarrow \underline{\mathbf{C}}$

Let $\mathit{Stack}(\underline{\mathbf{C}}, \underline{\mathbf{D}})$ be the set of stacks with type $\underline{\mathbf{C}} \Rightarrow \underline{\mathbf{D}}$. For any pair $(S, \underline{M}) \in \mathit{Stack}(\underline{\mathbf{C}}, \underline{\mathbf{D}}) \times \mathit{Terms}(\underline{\mathbf{C}})$, we can substitute $\underline{M}$ into the stack S.

Definition 2.1.6. We denote by $S\{\underline{M}\} : \underline{\mathbf{D}}$ the computation term resulting from the substitution of a term $\underline{M} : \underline{\mathbf{C}}$ in a stack $S \in \mathit{Stack}(\underline{\mathbf{C}}, \underline{\mathbf{D}})$, according to the rules:

$$\varepsilon\{\underline{M}\} := \underline{M}$$

$$(S \circ (-)\ \mathsf{to}\ x.\, \underline{N})\{\underline{M}\} := S\{\underline{M}\ \mathsf{to}\ x.\, \underline{N}\}$$

$$(S \circ (-)\ V)\{\underline{M}\} := S\{\underline{M}\ V\}$$

$$(S \circ (-)\ i)\{\underline{M}\} := S\{\underline{M}\ i\}$$

We can interpret a pair $(S, \underline{M})$ as the computation term $S\{\underline{M}\}$, where currently the reduction of the computation is focussed on reducing $\underline{M}$. Whenever you encounter a computation of which you need to first evaluate a subterm, you unfold the outermost term constructor into the Stack and continue evaluating the subterm.

Definition 2.1.7. For each computation type $\underline{\mathbf{C}}$, the *stack reduction relation* $\rightarrowtail$ gives a partial function $\bigcup_{\underline{\mathbf{D}}}(\mathit{Stack}(\underline{\mathbf{D}},\underline{\mathbf{C}}) \times \mathit{Terms}(\underline{\mathbf{D}})) \to \bigcup_{\underline{\mathbf{D}}}(\mathit{Stack}(\underline{\mathbf{D}},\underline{\mathbf{C}}) \times \mathit{Terms}(\underline{\mathbf{D}}))$ according to the following rules:

1. If $\underline{M} \rightsquigarrow \underline{N}$, then $(S, \underline{M}) \rightarrowtail (S, \underline{N})$.
2. $(S, \underline{M}\ \mathsf{to}\ x.\, \underline{N}) \rightarrowtail (S \circ (-)\ \mathsf{to}\ x.\, \underline{N}, \underline{M})$.
3. $(S \circ (-)\ \mathsf{to}\ x.\, \underline{N}, \mathsf{return}(V)) \rightarrowtail (S, \underline{N}[V/x])$.
4. $(S, \underline{M}\ V) \rightarrowtail (S \circ (-)\ V, \underline{M})$.
5. $(S \circ (-)\ V, \lambda x.\, \underline{M}) \rightarrowtail (S, \underline{M}[V/x])$
6. $(S, \underline{M}\ j) \rightarrowtail (S \circ (-)\ j, \underline{M})$.
7. $(S \circ (-)\ j, \langle \underline{M}_i \mid i \in I\rangle) \rightarrowtail (S, \underline{M}_j)$

This definition is well-defined because of the type preservation properties, as given in Lemmas 2.1.2 and 2.1.5.

For a relation $\mathcal{R}$, we write $\mathcal{R}^*$ for the reflexive and transitive closure of $\mathcal{R}$. When we want to evaluate a computation $\underline{M}$, we proceed by evaluating the pair $(\varepsilon, \underline{M})$. So if $(\varepsilon, \underline{M}) \rightarrowtail^* (\varepsilon, \underline{N})$ where $\underline{N} : \underline{\mathbf{C}}$ is a terminal computation term, we say that $\underline{M}$ *reduces to* $\underline{N}$. We can observe the following fact.

Lemma 2.1.8. *For each $\underline{M} : \underline{\mathbf{C}}$ there is at most one terminal computation term $\underline{N} : \underline{\mathbf{C}}$ such that $(\varepsilon, \underline{M}) \rightarrowtail^* (\varepsilon, \underline{N})$.*

Proof. This is an immediate consequence of the fact that any pair $(S, \underline{M})$ has at most one pair $(Z, \underline{N})$ it can reduce to via $\rightarrowtail$. Moreover, for any terminal computation term $\underline{N} : \underline{\mathbf{C}}$, the pair $(\varepsilon, \underline{N})$ cannot reduce to anything. Given these facts, the result follows from a routine induction. □

With the inclusion of fixpoint operators, computations may diverge. We specify for every computation type, a canonical nonterminating computation

$$\Omega := \mathsf{fix}(\lambda x.\, \mathsf{force}(x)) : \underline{\mathbf{C}}\ .$$

This term diverges, because for any stack S the reduction relation creates a cycle:

$$(S, \Omega) \rightarrowtail (S, \lambda x.\, \mathsf{force}(x)\ \mathsf{thunk}(\Omega)) \rightarrowtail (S \circ (-)\ \mathsf{thunk}(\Omega), \lambda x.\, \mathsf{force}(x)) \rightarrowtail$$

$$\rightarrowtail (S, \mathsf{force}(\mathsf{thunk}(\Omega))) \rightarrowtail (S, \Omega)\ .$$

Since the reduction relation is deterministic, by Lemma 2.1.8, we can conclude that Ω does not reduce to a terminal computation term.

We write $\circledast$ as short for the terminal computation term $\mathsf{return}(*)$. We can observe the difference between call-by-name and call-by-value by looking at a program '$(\lambda x.\, \circledast)\ \Omega$'. The call-by-name interpretation of this term in our language is '$(\lambda x.\, \circledast)\ \mathsf{thunk}(\Omega)$' which

reduces neatly to $\circledast$. The call-by-value interpretation is 'Ω to $x.\,(\lambda y.\,\circledast)\;x$', which reduces to the pair '$(\varepsilon \circ (-)$ to $x.\,(\lambda y.\,\circledast)\;x, \Omega)$', which diverges. See [42] for a precise embedding of call-by-value and call-by-name PCF into this language.

For simplicity in our examples, we add some more syntactic sugar.

- $\mathbf{rt}(\underline{M}) := \mathsf{return}(\mathsf{thunk}(\underline{M}))$.
- $(\underline{M}; \underline{N}) := \underline{M}$ to $x.\,\underline{N}$, when is x not free in $\underline{N}$. This is a special kind of sequencing where first $\underline{M}$ is evaluated and then $\underline{N}$ is evaluated. The final result will be the result produced by $\underline{N}$, whereas the result produced by $\underline{M}$ is discarded.
- $\overline{0} := \mathsf{Z}$ and for all natural numbers $n \in \mathbb{N}$, $\overline{n+1} := \mathsf{S}(\overline{n})$.

2.2 Adding algebraic effects

To the language defined in the previous section, we add a collection of algebraic effects in the style of [82]. Effects are those aspects of computation that involve a program interacting with the world 'outside'; for example: nondeterminism, probabilistic choice (in both cases, the choice is deferred to the environment); input/output; and mutable store (the machine state is modified). Algebraic effects in particular are effects modelled by effect-triggering operations, whose 'algebraic' nature means that effects act independently of the continuation. In particular, it holds that stacks distribute over algebraic effect operations.

What moreover makes these effects algebraic is the way in which they are generated. In general, an effect is specified using an *effect signature* Σ, which is a set containing one or more effect *operators*. Each operator combines a tuple of arguments given by computations into a single computation. How such a combination behaves depends on the specific effect in question. An effect operator has a specified *arity*. We consider arities of the following forms:

$$\mathbf{N}^n \times \underline{\alpha}^m \to \underline{\alpha} \qquad\qquad \mathbf{N}^n \times (\underline{\alpha}^{\mathbf{N}}) \to \underline{\alpha}$$

where n and m are non-negative integers. Normally in the literature, the word arity is used to designate the number of arguments an operation has. Using that formulation, we can see the above 'arities' as describing a family of operations (indexed by $\mathbf{N}^n$) of arity m and $\mathbb{N}$ respectively. For convenience, we use the word arity in this thesis to mean the more descriptive functional forms above. There, the symbol $\underline{\alpha}$ is used as a mnemonic device, which can be instantiated by any computation type. The arity shows us how computations can be combined into a single computation of type $\underline{\alpha}$. With the inclusion of type polymorphism in Chapter 7, the effect operators can be seen as polymorphic functional terms, where $\underline{\alpha}$ is a polymorphic type.

Note that in the arities, both n and m can be zero, in which case no arguments of that type are necessary. An example of an effect operator is $\mathsf{or}(-,-) : \mathbf{N}^0 \times \underline{\alpha}^2 \to \underline{\alpha}$, which combines two computations $\underline{M}$ and $\underline{N}$ into a single computation $\mathsf{or}(\underline{M}, \underline{N})$. This new computation will, when evaluated, continue evaluation with either $\underline{M}$ or $\underline{N}$. Which

of the two computations is evaluated depends on the behaviour of the effect; for instance it can be determined by some probabilistic procedure, or by a scheduler of the operating system. As another example of effect operator, consider $\mathsf{update}_l(-;-) : \mathbf{N}^1 \times \underline{\alpha}^1 \to \underline{\alpha}$, where $\mathsf{update}_l(\overline{n}; \underline{M})$ stores the number n at some global store location l, and continues evaluation with $\underline{M}$.

The effect operators pertaining to the effects we want to study are collected in an *effect signature* Σ, giving a set of effect operators. Given such a signature, new computation terms can be constructed according to the typing rules in Fig. 2.2. There, $\mathsf{op}(V_1, \ldots, V_n, x \mapsto \underline{M})$ binds x in $\underline{M}$.

$$\frac{\Gamma \vdash V_i : \mathbf{N} \text{ for each } 1 \leq i \leq n \quad \Gamma \vdash \underline{M}_i : \underline{\mathbf{C}} \text{ for each } 1 \leq i \leq m}{\Gamma \vdash \mathsf{op}(V_1, \ldots, V_n, \underline{M}_1, \ldots, \underline{M}_m) : \underline{\mathbf{C}}} \; \mathsf{op} : \mathbf{N}^n \times \underline{\alpha}^m \to \underline{\alpha} \in \Sigma$$

$$\frac{\Gamma \vdash V_i : \mathbf{N} \;\text{ for each }\; 1 \leq i \leq n \qquad \Gamma, x : \mathbf{N} \vdash \underline{M} : \underline{\mathbf{C}}}{\Gamma \vdash \mathsf{op}(V_1, \ldots, V_n, x \mapsto \underline{M}) : \underline{\mathbf{C}}} \; \mathsf{op} : \mathbf{N}^n \times \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha} \in \Sigma$$

Figure 2.2: Effect typing rules

Remark: We use $\mathbf{N}$ for arguments of algebraic effect operations, since it is the only non-trivial base type in the language. Other cases, in which we for example want to used Boolean-valued arguments, can be simulated using natural numbers.

In the presence of algebraic effects, computation terms may be reduced to trees of effect operators. Whenever we encounter an effect operator in the evaluation of a computation term, we continue the reduction for each of its arguments, collecting all the continuations into a tree. This way, we define an operational semantics which reduces a computation term to a tree whose internal nodes are effect operators and leaves are either labelled by $\bot$ (representing divergence) or by terminal terms.

Definition 2.2.1. An *effect tree* (henceforth *tree*), over a set X, determined by a signature Σ of effect operations, is a labelled and possibly infinite depth tree whose nodes have the following possible forms:

1. A leaf node labelled with $\bot$ (representing divergence).

2. A leaf node labelled with $\langle x \rangle$ where $x \in X$.

3. A node labelled $\mathsf{op}_{l_1,\ldots,l_n}$ with children $t_1, \ldots, t_m$, when the operator $\mathsf{op} \in \Sigma$ has arity $\mathbf{N}^n \times \underline{\alpha}^m \to \underline{\alpha}$ and $l_1, \ldots, l_n \in \mathbb{N}$. In this case, we write the subtree at that node as $\mathsf{op}_{l_1,\ldots,l_n}\langle t_1, \ldots, t_m \rangle$.

4. A node labelled $\mathsf{op}_{l_1,\ldots,l_n}$ with an infinite sequence $t_0, t_1, \ldots$ of children, when the operator $\mathsf{op} \in \Sigma$ has arity $\mathbf{N}^n \times \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$ and $l_1, \ldots, l_n \in \mathbb{N}$. In this case, we write the subtree at that node as $\mathsf{op}_{l_1,\ldots,l_n}\langle m \mapsto t_m \rangle$.

The set of effect trees over a set X determined by Σ is denoted by $T_\Sigma(X)$, but we tend to just write it as $T(X)$ in case Σ can be inferred from the context. We define a partial ordering on $T(X)$ where $t_1 \leq t_2$, if t_1 can be obtained by pruning t_2, removing a possibly infinite number of subtrees of t_2 and putting a leaf node labelled $\bot$ in their place. In particular, $\bot \leq t$ for any tree t. This forms an ω-*complete* partial order, meaning that every ascending sequence $t_1 \leq t_2 \leq \ldots$ has a least upper bound $\bigsqcup_n t_n$.

Given $f : X \to Y$ and a tree $t \in T(X)$, we write $t[x \mapsto f(x)] \in T(Y)$ for the tree whose leaves labelled by $\langle x \rangle$ for any $x \in X$ are renamed to leaves labelled $\langle f(x) \rangle$. We define $f^* := T(f) : T(X) \to T(Y)$ for the function sending t to $t[x \mapsto f(x)]$, so $T(-)$ can be considered a functor in the category of sets. We have a function $\mu : T(T(X)) \to T(X)$, which takes a tree r of trees and 'flattens' it to a tree $\mu r \in T(X)$, by taking the labelling tree at each non-$\bot$ leaf of r as the subtree at the corresponding node in μr. The function μ is the multiplication associated with the monad structure of the $T(-)$ operation. The unit of the monad is the map $\eta : X \to T(X)$ which takes an element $x \in X$ and returns the tree whose root is the leaf labelled $\langle x \rangle$.

Proposition 2.2.2. *The triple $(T(-), \eta, \mu)$ is a monad on the category of sets.*

We will define the operational semantics of our language in terms of such effect trees. These trees may carry information that would not be detectable by an observer of the program. However, in order to treat algebraic effects in a generic way, we choose to define the operational semantics using trees dependent only on the algebraic signature Σ of the effects. We postpone the interpretation of behaviour of effects to the next chapter, where we define behavioural properties of programs in terms of effect trees. Those properties will be defined in such a way that they do not distinguish between behaviourally equivalent terms.

For a computation type $\underline{\mathbf{C}}$ we write $T(\underline{\mathbf{C}}) := T(Tct(\underline{\mathbf{C}}))$, and $T(\mathbf{A}) := T(Tct(\mathbf{F}\,\mathbf{A}))$ $(= T(\mathbf{F}\,\mathbf{A}))$. The latter is isomorphic to $T(Terms(\mathbf{A}))$ (e.g., $T(\mathbf{1}) = T(\{*\})$), and the two may be used interchangeably. We will define a reduction relation from computations to trees of terminal computation terms. The operational mapping from a computation $\underline{M} \in Terms(\underline{\mathbf{C}})$ to an effect tree $T(\underline{\mathbf{C}})$ is defined intuitively as follows. Start evaluating the $\underline{M}$ in the empty stack ε, until the evaluation process (which is deterministic) terminates (if this never happens, the tree is $\bot$). If the evaluation process terminates at a configuration of the form $(\varepsilon, \underline{N})$, where $\underline{N}$ is a terminal computation term, then the tree consists of one leaf labelled $\langle \underline{N} \rangle$. Otherwise the evaluation process can only terminate at a configuration of the form $(S, \mathsf{op}(\ldots))$ for some effect operation $\mathsf{op} \in \Sigma$. In this case, create an internal node in the tree of the appropriate kind (depending on op) and continue generating each child tree of this node, repeating the above process on the appropriate subcomputation using stack S.

Formally, we define the tree reduction for computation terms as a function $|-| : Terms(\underline{\mathbf{C}}) \to T(\underline{\mathbf{C}})$, which returns a tree whose leaves are either $\bot$ or terminal computation terms of $\underline{\mathbf{C}}$ [2]. We do this inductively by first defining approximations

[2] In the case of an operator of arity zero, we can have an 'internal end node' given by that operator, which we do not count as a leaf, even though it has no children

of this tree.

Definition 2.2.3. The *approximating tree reduction* is a function $|-,-|_{(-)}$ from $\bigcup_{\underline{\mathbf{D}}} \mathit{Stack}(\underline{\mathbf{D}},\underline{\mathbf{C}}) \times \mathit{Terms}(\underline{\mathbf{D}}) \times \mathbb{N}$ to $T(\underline{\mathbf{C}})$ defined using the following rules:

1. $|S,\underline{M}|_0 := \bot$
2. $|\varepsilon,\underline{M}|_{n+1} := \langle \underline{M} \rangle$ if $\underline{M}$ is a terminal computation term.
3. $|S,\underline{M}|_{n+1} := |S',\underline{M}'|_n$ if $(S,\underline{M}) \rightarrowtail (S',\underline{M}')$.
4. $|S,\mathsf{op}(\overline{n_1},\ldots,\overline{n_k},\underline{M}_1,\ldots,\underline{M}_m)|_{n+1} := \mathsf{op}_{n_1,\ldots,n_k}\langle |S,\underline{M}_1|_n,\ldots,|S,\underline{M}_m|_n\rangle$ if $\mathsf{op} : \mathbf{N}^k \times \alpha^m \to \alpha$.
5. $|S,\mathsf{op}(\overline{n_1},\ldots,\overline{n_k},x \mapsto \underline{M})|_{n+1} := \mathsf{op}_{n_1,\ldots,n_k}\langle \overline{m} \mapsto |S,\underline{M}[\overline{m}/x]|_{\max(0,n-m)}\rangle$ if $\mathsf{op} : \mathbf{N}^k \times \alpha^{\mathbf{N}} \to \alpha$.

A tree t is *finite* if it has only finitely many nodes and non-$\bot$ leaves. Equivalently:

Definition 2.2.4. A tree $t \in T(X)$ is *finite* if the set $\{r \in T(X) \mid r \leq t\}$ is finite.

Note that for any pair $(S,\underline{M})$ and any natural number $n \in \mathbb{N}$, $|S,\underline{M}|_n$ is finite. In particular, the definition for $|-|_{n+1}$ for effect nodes with countably many children has been formulated in such a way, using 'max', to ensure that this property holds.

Lemma 2.2.5. *For any pair* $(S,\underline{M}) \in \mathit{Stack}(\underline{\mathbf{D}},\underline{\mathbf{C}}) \times \mathit{Terms}(\underline{\mathbf{D}})$ *and any natural number* $n \in \mathbb{N}$, $|S,\underline{M}|_n \leq |S,\underline{M}|_{n+1}$.

Proof. We prove this by an induction on n. If $n = 0$, then $|S,\underline{M}|_0 = \bot \leq |S,\underline{M}|_1$. Assume the statement holds for all k smaller or equal to n. We do a case analysis.

1. If $(S,\underline{M}) = (\varepsilon,\underline{N})$ with $\underline{N}$ terminal, then $|S,\underline{M}|_{n+1} = \langle \underline{N} \rangle = |S,\underline{M}|_{(n+1)+1}$.
2. If $(S,\underline{M}) \rightarrowtail (Z,\underline{N})$, then $|S,\underline{M}|_{n+1} = |Z,\underline{N}|_n$, which by induction hypothesis is smaller or equal to $|Z,\underline{N}|_{n+1} = |S,\underline{M}|_{n+2}$.
3. If $\underline{M} = \mathsf{op}(\overline{n_1},\ldots,\overline{n_k},\underline{M}_1,\ldots,\underline{M}_m)$, then $|S,\underline{M}|_{n+1} =$ $\mathsf{op}_{n_1,\ldots,n_k}\langle |S,\underline{M}_1|_n,\ldots,|S,\underline{M}_m|_n\rangle$. By the induction hypothesis, $|S,\underline{M}_i|_n \leq |S,\underline{M}_i|_{n+1}$ for each $1 \leq i \leq k$. Hence $|S,\underline{M}|_{n+1} \leq$ $\mathsf{op}_{n_1,\ldots,n_k}\langle |S,\underline{M}_1|_{n+1},\ldots,|S,\underline{M}_m|_{n+1}\rangle = |S,\underline{M}|_{n+2}$.
4. If $\underline{M} = \mathsf{op}(\overline{n_1},\ldots,\overline{n_k},x \mapsto \underline{N})$, then $|S,\underline{M}|_{n+1} = \mathsf{op}_{n_1,\ldots,n_k}\langle \overline{m} \mapsto |S,\underline{N}[\overline{m}/x]|_{\max(0,n-m)}\rangle$. By induction hypothesis, for each m, $|S,\underline{N}[\overline{m}/x]|_{\max(0,n-m)} \leq |S,\underline{N}[\overline{m}/x]|_{\max(0,n+1-m)}$, hence $|S,\underline{M}|_{n+1} \leq$ $\mathsf{op}_{n_1,\ldots,n_k}\langle \overline{m} \mapsto |S,\underline{N}[\overline{m}/x]|_{\max(0,n+1-m)}\rangle = |S,\underline{M}|_{n+2}$.

□

Because of the previous Lemma, we can take the supremum of the sequence, defining $|S, \underline{M}| := \bigsqcup_n |S, \underline{M}|_n$. Using this, we define $|\underline{M}|_n := |\varepsilon, \underline{M}|_n$ and $|\underline{M}| := |\varepsilon, \underline{M}| = \bigsqcup_n |\underline{M}|_n$. For instance, since Ω diverges, any approximation $|\Omega|_n$ will yield $\bot$ and hence $|\Omega| = \bot$. We look at some other general results, which are quite technical. The reader who would like to skip the technical material on first reading may continue reading at Section 2.3.

Lemma 2.2.6. *For any triple* $(S, Z, \underline{M}) \in \mathit{Stack}(\underline{\mathbf{C}}', \underline{\mathbf{C}}) \times \mathit{Stack}(\underline{\mathbf{D}}, \underline{\mathbf{C}}') \times \mathit{Terms}(\underline{\mathbf{D}})$ *and any natural number* n, $|S@Z, \underline{M}|_n \leq \mu(|Z, \underline{M}|_n[\underline{N} \mapsto |S, \underline{N}|_n])$.

Proof. We prove this by induction on n.

Suppose $n = 0$, then $|S@Z, \underline{M}|_0 = \bot \leq \mu(|Z, \underline{M}|_n[\underline{N} \mapsto |S, \underline{N}|_n])$.

Suppose the statement holds for any $k \leq n$, we prove it holds for $n+1$ by case analysis on $(Z, \underline{M})$.

1. If $(Z, \underline{M}) = (\varepsilon, \underline{N})$ with $\underline{N}$ a terminal computation term, then $|S@Z, \underline{M}|_{n+1} = |S, \underline{N}|_{n+1} = \mu(\langle \underline{N} \rangle[\underline{N}' \mapsto |S, \underline{N}'|_{n+1}]) = \mu(|\underline{M}|_{n+1}[\underline{N}' \mapsto |S, \underline{N}'|_{n+1}]) = \mu(|Z, \underline{M}|_{n+1}[\underline{N}' \mapsto |S, \underline{N}'|_{n+1}])$.

2. If $(Z, \underline{M}) \rightarrowtail (Z', \underline{M}')$, then by induction $|S@Z, \underline{M}|_{n+1} = |S@Z', \underline{M}'|_n$, which by induction hypothesis is smaller or equal to $\mu(|Z', \underline{M}'|_n[\underline{N} \mapsto |S, \underline{N}|_n]) = \mu(|Z, \underline{M}|_{n+1}[\underline{N} \mapsto |S, \underline{N}|_n])$. By Lemma 2.2.5, this is smaller or equal to $\mu(|Z, \underline{M}|_{n+1}[\underline{N} \mapsto |S, \underline{N}|_{n+1}])$.

3. If $\underline{M} = \mathsf{op}(\overline{n_1}, \ldots, \overline{n_k}, \underline{M}_1, \ldots, \underline{M}_m)$, then $|S@Z, \underline{M}|_{n+1} =$

   $\mathsf{op}_{n_1,\ldots,n_k}\langle |S@Z, \underline{M}_1|_n, \ldots, |S@Z, \underline{M}_m|_n \rangle$. Each $|S@Z, \underline{M}_i|_n$ is by induction hypothesis smaller or equal to $\mu(|Z, \underline{M}_i|_n[\underline{N} \mapsto |S, \underline{N}|_n])$. Hence $|S@Z, \underline{M}|_{n+1} \leq$
   $\mathsf{op}_{n_1,\ldots,n_k}\langle \mu(|Z, \underline{M}_1|_n[\underline{N} \mapsto |S, \underline{N}|_n]), \ldots, \mu(|Z, \underline{M}_m|_n[\underline{N} \mapsto |S, \underline{N}|_n]) \rangle \leq$
   $\mu(\mathsf{op}_{n_1,\ldots,n_k}\langle |Z, \underline{M}_1|_n[\underline{N} \mapsto |S, \underline{N}|_n], \ldots, |Z, \underline{M}_m|_n[\underline{N} \mapsto |S, \underline{N}|_n] \rangle) \leq$
   $\mu(\mathsf{op}_{n_1,\ldots,n_k}\langle |Z, \underline{M}_1|_n, \ldots, |Z, \underline{M}_m|_n \rangle[\underline{N} \mapsto |S, \underline{N}|_n]) \leq$
   $\mu(|Z, \underline{M}|_{n+1}[\underline{N} \mapsto |S, \underline{N}|_n])$. By Lemma 2.2.5, this is smaller or equal to $\mu(|Z, \underline{M}|_{n+1}[\underline{N} \mapsto |S, \underline{N}|_{n+1}])$.

4. If $\underline{M} = \mathsf{op}(\overline{n_1}, \ldots, \overline{n_k}, x \mapsto \underline{N})$, the proof goes similarly to 3.

□

Corollary 2.2.7. $|S\{\underline{M} \text{ to } x.\, \underline{N}\}|_{n+1} \leq \mu(|\underline{M}|_n[\mathsf{return}(V) \mapsto |S\{\underline{N}[V/x]\}|_n])$.

Proof. If S is of length $l \in \mathbb{N}$, then $|S\{\underline{M} \text{ to } x.\, \underline{N}\}|_{n+1} = |S, \underline{M} \text{ to } x.\, \underline{N}|_{n+1-l} = |S \circ - \text{ to } x.\, \underline{N}, \underline{M}|_{n-l}$. If $n \leq l$, then $|S\{\underline{M} \text{ to } x.\, \underline{N}\}|_{n+1} = \bot$ and the statement holds. So we may assume that $n > l$.

$|S \circ - \text{ to } x.\, \underline{N}, \underline{M}|_{n-l} \leq \mu(|\underline{M}|_{n-l}[\mathsf{return}(V) \mapsto |S \circ - \text{ to } x.\, \underline{N}, \mathsf{return}(V)|_{n-l}])$ by Lemma 2.2.6. Since for any V, $|S \circ - \text{ to } x.\, \underline{N}, \mathsf{return}(V)|_{n-l} = |S, \underline{N}[V/x]|_{n-1-l} = |S\{\underline{N}[V/x]\}|_{n-1}$, it holds that $|S \circ - \text{ to } x.\, \underline{N}, \underline{M}|_{n-l} \leq \mu(|\underline{M}|_{n-l}[\mathsf{return}(V) \mapsto |S\{\underline{N}[V/x]\}|_{n-1}])$, which is lower or equal to $\mu(|\underline{M}|_n[\mathsf{return}(V) \mapsto |S\{\underline{N}[V/x]\}|_n])$. □

Lemma 2.2.8. *For any triple* $(S, Z, \underline{M}) \in \mathit{Stack}(\underline{\mathbf{C}'}, \underline{\mathbf{C}}) \times \mathit{Stack}(\underline{\mathbf{D}}, \underline{\mathbf{C}'}) \times \mathit{Terms}(\underline{\mathbf{D}})$ *and any two natural numbers* n, k, $\mu(|Z, \underline{M}|_n[\underline{N} \mapsto |S, \underline{N}|_k]) \leq |S@Z, \underline{M}|_{n+k+1}$.

Proof. We prove this by induction on n.

Suppose $n = 0$, then $\mu(|Z, \underline{M}|_0[\underline{N} \mapsto |S, \underline{N}|_k]) = \mu(\bot) = \bot \leq |S@Z, \underline{M}|_{n+k+1}$.

Suppose the statement holds for any $n' \leq n$, we prove it holds for $n+1$ by case analysis on $(Z, \underline{M})$.

1. If $(Z, \underline{M}) = (\varepsilon, \underline{M'})$ with $\underline{M'}$ terminal, then $\mu(|Z, \underline{M}|_{n+1}[\underline{N} \mapsto |S, \underline{N}|_k]) = \mu(\langle \underline{M'} \rangle[\underline{N} \mapsto |S, \underline{N}|_k]) = |S, \underline{M'}|_k = |S@Z, \underline{M}|_k \leq |S@Z, \underline{M}|_{n+k+2}$.

2. If $(Z, \underline{M}) \rightarrowtail (Z', \underline{M'})$, then $\mu(|Z, \underline{M}|_{n+1}[\underline{N} \mapsto |S, \underline{N}|_k]) = \mu(|Z', \underline{M'}|_n[\underline{N} \mapsto |S, \underline{N}|_k])$ which by induction hypothesis is smaller or equal to $|S@Z', \underline{M'}|_{n+k+1} \leq |S@Z, \underline{M}|_{n+k+2}$.

3. If $\underline{M} = \mathsf{op}(\overline{l_1}, \ldots, \overline{l_l}, \underline{M}_1, \ldots, \underline{M}_m)$, then $\mu(|Z, \underline{M}|_{n+1}[\underline{N} \mapsto |S, \underline{N}|_k]) =$
$\mu(\mathsf{op}_{l_1,\ldots,l_l}\langle |Z, \underline{M}_1|_n, \ldots, |Z, \underline{M}_m|_n \rangle[\underline{N} \mapsto |S, \underline{N}|_k]) =$
$\mu(\mathsf{op}_{l_1,\ldots,l_l}\langle |Z, \underline{M}_1|_n[\underline{N} \mapsto |S, \underline{N}|_k], \ldots, |Z, \underline{M}_m|_n[\underline{N} \mapsto |S, \underline{N}|_k] \rangle) =$
$\mathsf{op}_{l_1,\ldots,l_l}\langle \mu(|Z, \underline{M}_1|_n[\underline{N} \mapsto |S, \underline{N}|_k]), \ldots, \mu(|Z, \underline{M}_m|_n[\underline{N} \mapsto |S, \underline{N}|_k]) \rangle \leq$
$\mathsf{op}_{l_1,\ldots,l_l}\langle |S@Z, \underline{M}_1|_{n+k+1}, \ldots, |S@Z, \underline{M}_m|_{n+k+1} \rangle = |S@Z, \underline{M}|_{n+k+2}$.

4. If $\underline{M} = \mathsf{op}(\overline{l_1}, \ldots, \overline{l_k}, x \mapsto \underline{N})$, the proof goes similarly to 3.

□

As a consequence of the previous two lemmas we can conclude with the following proposition:

Proposition 2.2.9. *For any triple* $(S, Z, \underline{M}) \in \mathit{Stack}(\underline{\mathbf{C}'}, \underline{\mathbf{C}}) \times \mathit{Stack}(\underline{\mathbf{D}}, \underline{\mathbf{C}'}) \times \mathit{Terms}(\underline{\mathbf{D}})$, $|S@Z, \underline{M}| = \mu(|Z, \underline{M}|[\underline{N} \mapsto |S, \underline{N}|])$.

Proof. By Lemma 2.2.6 and Lemma 2.2.8, it holds for any $n \in \mathbb{N}$ that:

$|S@Z, \underline{M}|_n \leq \mu(|Z, \underline{M}|_n[\underline{N} \mapsto |S, \underline{N}|_n]) \leq |S@Z, \underline{M}|_{2n+1}$, so the suprema of the two sequences $\{|S@Z, \underline{M}|_n\}_{n \in \mathbb{N}}$ and $\{\mu(|Z, \underline{M}|_n[\underline{N} \mapsto |S, \underline{N}|_n])\}_{n \in \mathbb{N}}$ are identical. □

Since $|\varepsilon, S\{\underline{M}\}| = |S, \underline{M}|$ we can derive with Proposition 2.2.9:

Corollary 2.2.10. *The following three statements hold:*

1. $|\underline{M}\ \mathsf{to}\ x.\ \underline{N}| = \mu(|\underline{M}|[\mathsf{return}(V) \mapsto |\underline{N}[V/x]|])$.

2. $|\underline{M}\ V| = \mu(|\underline{M}|[\lambda x.\ \underline{N} \mapsto |\underline{N}[V/x]|])$.

3. $|\underline{M}\ j| = \mu(|\underline{M}|[\langle \underline{N}_i \mid i \in I \rangle \mapsto |\underline{N}_j|])$.

In the future, we will sometimes use $t[V \mapsto f(V)]$ as shorthand for $t[\mathsf{return}(V) \mapsto f(V)]$.

As discussed before, at Definition 2.2.4, a tree t is finite if and only if it has finitely many non-$\bot$ nodes. Finite trees are exactly the finite elements of the domain $T(X)$,

which has the following consequence: if $r \leq \bigsqcup_n t_n$ for r finite and $\{t_n\}_{n\in\mathbb{N}}$ some ascending sequence, then $r \leq t_n$ for some $n \in \mathbb{N}$. Moreover, if $\mathsf{op}_{l_1,\dots,l_m}(t_0, t_1, t_2, \dots)$ is finite, then for some $n \in \mathbb{N}$, it holds that $\forall k \in \mathbb{N}, k > n \implies t_k = \bot$.

Moreover, any finite tree in $T(\underline{\mathbf{C}})$ occurs as the computation tree of a computation term in the empty stack.

Lemma 2.2.11. *For any finite tree $t \in T(Terms(\underline{\mathbf{C}}))$, there is a term $\underline{M} : \underline{\mathbf{C}}$ such that $|\underline{M}| = \mu(t[\underline{N} \mapsto |\underline{N}|])$.*

Proof. Any such finite tree $t \in T(\underline{\mathbf{C}})$ can be transformed into a term by the following inductively defined function $f : T(\underline{\mathbf{C}}) \to Terms(\underline{\mathbf{C}})$:

$$
\begin{aligned}
&f(\langle \underline{M} \rangle) = \underline{M}.\\
&f(\bot) = \Omega.\\
&f(\mathsf{op}_{l_1,\dots,l_m}(t_1, \dots, t_k)) := \mathsf{op}(\overline{l_1}, \dots, \overline{l_m}, f(t_1), \dots, f(t_k)).\\
&f(\mathsf{op}_{l_1,\dots,l_m}(t_0, \dots, t_k, \bot, \dots)) := \mathsf{op}(\overline{l_1}, \dots, \overline{l_m}, x_0 \mapsto\\
&\qquad\qquad \mathsf{case}\ x_0\ \mathsf{of}\ \{f(t_0), \mathsf{S}(x_1) \Rightarrow\\
&\qquad\qquad \mathsf{case}\ x_1\ \mathsf{of}\ \{f(t_1), \mathsf{S}(x_2) \Rightarrow \dots\\
&\qquad\qquad\qquad \vdots\\
&\qquad\qquad \mathsf{case}\ x_k\ \mathsf{of}\ \{f(t_k), \mathsf{S}(x_{k+1}) \Rightarrow \Omega\}\}\}).
\end{aligned}
$$

The last part uses some iteration of case distinctions on the natural numbers. If t is finite, then the function f is defined, and it can be proven that $|f(t)| = \mu(t[\underline{N} \mapsto |\underline{N}|])$ with a simple induction on finite trees. □

Corollary 2.2.12. *For any finite tree $t \in T(\underline{\mathbf{C}}) = T(Tct(\underline{\mathbf{C}}))$, there is a term $\underline{M} : \underline{\mathbf{C}}$ such that $|\underline{M}| = t$.*

2.3 Examples of effects

We will look at some examples of effects and their signatures of effect operators. Our focus here is to define the particular effect operators needed to describe certain effects, giving only a rough idea of the behaviour of the effect. The operational semantics are given in terms of trees, which in general gives a lot of information irrelevant to the behaviour of an effect. The precise interpretation of behaviour for the different effects will be given in Chapter 3. The examples given here are mainly the standard examples of algebraic effects, taken partially from [35, 82, 84].

2.3.1 Pure

The first example is the case where there are no effects present, resulting in a language of *pure functional computations*, a call-by-push-value variant of PCF [80]. This entails taking the empty signature $\Sigma_\emptyset := \emptyset$, in which case the set of trees over a set X is given by the disjoint union $T_{\Sigma_\emptyset}(X) := X_\bot = X \cup \{\bot\}$. When describing elements of $T_{\Sigma_\emptyset}(X)$,

we will omit the use of 'inleft' and 'inright'. Here $T(X) = X_\perp$ has the expected domain structure where $x \leq y \iff x \in \{\perp, y\}$.

2.3.2 Error

We can extend the language with a set of error messages. We take a set of error messages Err, and for each of these messages $e \in \mathsf{Err}$ we add an effect operator $\mathsf{raise}_e()$ with arity $\mathbf{N}^0 \times \underline{\alpha}^0 \to \underline{\alpha}$, which simplifies to $\mathbf{1} \to \underline{\alpha}$. This arity describes the fact that no continuation is possible after the error has been raised. Our signature is given by $\Sigma_{\text{er}} := \{\mathsf{raise}_e() : \mathbf{1} \to \underline{\alpha} \mid e \in \mathsf{Err}\}$, where the computation $\mathsf{raise}_e()$ aborts evaluation and displays e as an error message.

Alternatively, we may allow the program to give some more feedback about the particular error to accompany the message. To this end we may give some of the algebraic effect operators an extra natural number as input, taking $\mathsf{raise}_e() : \mathbf{N} \to \underline{\alpha}$. For simplicity, we will only consider the former definition.

The trees over a set X can be given by the disjoint union $(X \cup \mathsf{Err})_\perp = X \cup \mathsf{Err} \cup \{\perp\}$ (we will omit the use of 'inleft' and 'inright'). An example of a possible use for error messages is the 'safe' predecessor function, which raises an error if you ask for the predecessor of zero.

$$\mathsf{Pred} \;:=\; \lambda y : \mathbf{N}.\mathbf{case}\ y\ \mathbf{of}\ \{Z \Rightarrow \mathsf{raise}_{Negative}(); S(x) \Rightarrow \mathsf{return}(x)\}\ .$$

This generates the following 'trees':

$$|\mathsf{Pred}\,\overline{0}| = \mathsf{raise}_{Negative} \qquad |\mathsf{Pred}\,(\overline{n+1})| = \langle \overline{n} \rangle\ .$$

2.3.3 Nondeterminism

Nondeterminism can arise in many ways. It may arise as the result of decisions made by some external scheduler, or from the particular environment in which the program is run. The key characteristic of nondeterminism is that we cannot predict how a computation will continue its reduction.

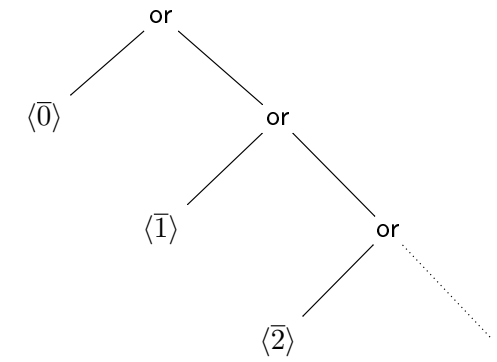
One way of implementing this algebraically is by taking a binary choice operator $\mathsf{or}(-,-) : \underline{\alpha}^2 \to \underline{\alpha}$ which gives two options for continuing the computation. Given two computations $\underline{M}$ and $\underline{N}$, $\mathsf{or}(\underline{M}, \underline{N})$ is the computation which will either continue evaluation with $\underline{M}$ or with $\underline{N}$. Which of the two it will be is however unknown. The choice of continuation can be understood as being under the control of an external agent, which one may wish to model as being cooperative (*angelic*), antagonistic (*demonic*), or completely unpredictable (*neutral*).

Our signature in this case only contains this binary choice operator, with $\Sigma_{\text{nd}} := \{\mathsf{or}(-,-) : \underline{\alpha}^2 \to \underline{\alpha}\}$. Trees over X are given by binary trees whose leaves are either labelled by $\perp$ or labelled by elements from X.

An example of a nondeterministic computation is a term of type $\mathbf{F}\,\mathbf{N}$ which may return any natural number:

$$? := \mathsf{fix}(\lambda x : \mathbf{1} \to \mathbf{N}.\, \mathsf{or}(\lambda y : \mathbf{1}.\, Z\,, \lambda y : \mathbf{1}.\, xy \text{ to } z.\, \mathsf{return}(\mathsf{S}(z)))) \text{ to } w.\, w*$$

$|?| =$



Alternatively, we may want to implement countable choice using an operator $\mathsf{c\text{-}or}(-) : \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$. However, with the tools used in this thesis, we will not be able to prove that the behavioural equivalence for demonic or neutral nondeterminism with countable choice is compatible (Theorem 3.3.8). We will discuss this in more detail in Subsection 3.3.2.

2.3.4 Probabilistic choice

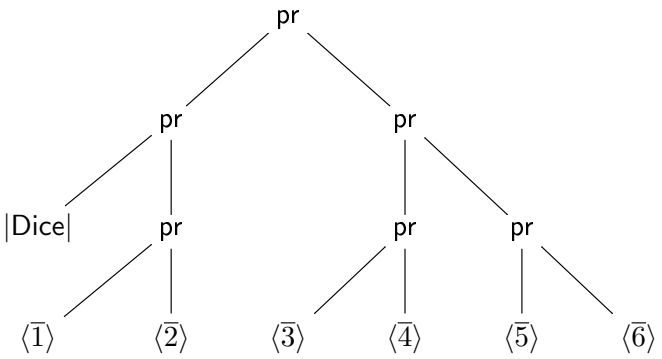
Similarly to the previous example, this illustrates the scenario where a computation's results may differ on different runs. However, here this variation is controlled by chance, and hence we can speak of the probability that a computation produces a certain output.

Again we implement this using a single binary choice operator $\mathsf{pr}(-,-) : \underline{\alpha}^2 \to \underline{\alpha}$ which gives two options for continuing the computation, with effect signature $\Sigma_{\mathrm{pr}} := \{\mathsf{pr}(-,-) : \underline{\alpha}^2 \to \underline{\alpha}\}$. In this case, the choice of continuation is probabilistic, where $\mathsf{pr}(\underline{M}, \underline{N})$ has an equal probability of either continuing with the computation $\underline{M}$, or with the computation $\underline{N}$. This is different from nondeterminism, in which we cannot associate a probability to the continuations. Trees over X are given by binary trees whose leaves are either $\bot$ or from X.

The following gives an example of how a dice throw can be simulated using the fixpoint operator and fair choice. We use $\mathsf{ret}(V)$ as shorthand for $\mathsf{return}(V)$. Notice that the tree is defined recursively in terms of itself, leading to an infinite tree.

$$\mathsf{Dice} := \mathsf{fix}(\lambda x.\, \mathsf{pr}(\mathsf{pr}(\mathsf{force}(x), \mathsf{pr}(\mathsf{ret}(\overline{1}), \mathsf{ret}(\overline{2}))), \mathsf{pr}(\mathsf{pr}(\mathsf{ret}(\overline{3}), \mathsf{ret}(\overline{4})), \mathsf{pr}(\mathsf{ret}(\overline{5}), \mathsf{ret}(\overline{6})))))$$

$|\mathsf{Dice}| =$



We could alternatively give our program easier control over the probability by implementing it with a weighted operator $\mathsf{w\text{-}or}(-,-,-,-) : \mathbf{N}^2 \times \underline{\alpha}^2 \to \underline{\alpha}$. We interpret $\mathsf{w\text{-}or}(\overline{n}, \overline{m}, \underline{M}, \underline{N})$ as the computation which has $\frac{n}{n+m}$ probability of continuing with $\underline{M}$ and $\frac{m}{n+m}$ probability of continuing with $\underline{N}$. If $n = m = 0$, the computation is resolved via fair choice. This is just one way of incorporating weighted probabilistic choice. The particular choice of definition does not matter however, since any type of weighted probabilistic choice operator can be programmed in terms of the fair choice operator and recursion. See for instance how the computation Dice defined above recursively models a fair choice between six possible continuations.

2.3.5 Global store

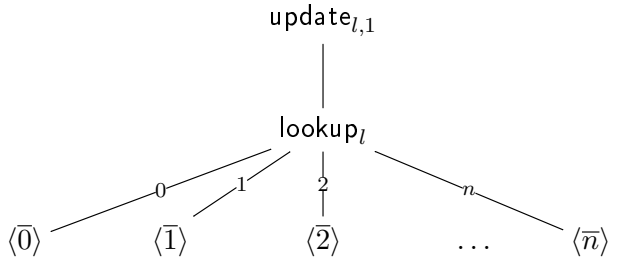
Here we look at the case where programs can interact with some global memory, being able to read from it and write to it.

We take a set of locations Loc for storing natural numbers. For each $l \in \mathsf{Loc}$ we have two effect operators $\mathsf{lookup}_l(-) : \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$ and $\mathsf{update}_l(-;-) : \mathbf{N} \times \underline{\alpha} \to \underline{\alpha}$, so its signature is given by $\Sigma_{\mathrm{gs}} := \{\mathsf{lookup}_l(-), \mathsf{update}_l(-;-) \mid l \in \mathsf{Loc}\}$. The computation $\mathsf{lookup}_l(x \mapsto \underline{M})$ looks up the number at location l and substitutes it for x in $\underline{M}$, and $\mathsf{update}_l(\overline{n}; \underline{M})$ stores n at location l and then continues with the computation $\underline{M}$.

The following computation accepts as an argument a natural number, which it will store at location l. After that, it will check the same location l and produce whatever is stored there.

$$V := \lambda y : \mathbf{N}.\mathsf{update}_l(y; \mathsf{lookup}_l(\lambda x : \mathbf{N}.\, \mathbf{return}(x))) \quad : \quad \mathbf{N} \to \mathbf{N}$$

$|V\overline{1}| =$



We see that the resulting tree exhibits redundancies with respect to the expected model of computation with global store. Since the update_l operation sets the value of location l to 1, the ensuing lookup_l operation will retrieve the value 1, and so execution will proceed down the branch labelled 1 resulting in the return value 1. The other infinitely many leaves of the tree are redundant. These redundancies are a result of the operational semantics being defined independently of the effect behaviour, as motivated before. In the next chapter, the behaviour of such computations will be specified accordingly, keeping the above considerations in mind.

There are several variations we might consider. We look at three, though they are not exhaustive. Firstly we can consider partiality, where some store locations may be empty, and we have a way to clear the contents of a location with the operator $\mathsf{clear}_l(-) : \underline{\alpha} \to \underline{\alpha}$. Secondly we may limit the way with which we are able to update

a store, e.g., only allowing to add one to the store with $\mathsf{add}_l(-) : \underline{\alpha} \to \underline{\alpha}$. Lastly, we might number the store locations with natural numbers, and make choosing the location interactive with $\mathsf{lookup}_{(-)}(-) : \mathbf{N} \times \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$ and $\mathsf{update}_{(-)}(-;-) : \mathbf{N}^2 \times \underline{\alpha} \to \underline{\alpha}$. Such variations considers stores as heaps allowing for pointer arithmetics, like the ones considered in the separation logic [68, 69, 89]. Though the above examples of variations on global store are very natural, they are not considered further on in the thesis. These examples may be studied in the future.

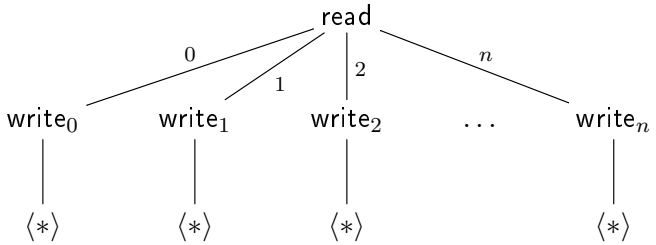
2.3.6 Input/Output

Here we consider an interaction with some external entity, for instance another computer or a human user. In this case, we have no way of predicting how this outside source will behave. However, unlike in the case of nondeterminism, the particular choices made are observable. For example, the users of a computer know which buttons they pressed to get a certain result.

Here we have two operators, $\mathsf{read}(-) : \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$ which reads a number from an input channel and passes it as the argument to a computation, and $\mathsf{write}(-;-) : \mathbf{N} \times \underline{\alpha} \to \underline{\alpha}$ which prints a number (the first argument) on the screen and then continues as the computation given as the second argument. Combined they have the signature $\Sigma_{\text{io}} := \{\mathsf{read}(-) : \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}, \mathsf{write}(-;-) : \mathbf{N} \times \underline{\alpha} \to \underline{\alpha}\}$.

The following example simply displays the number from some input on the screen.

$$\mathsf{Echo} := \mathsf{read}(x \mapsto x) \text{ to } y.\, \mathsf{write}(y; \circledast)$$



We take this opportunity to emphasize that the natural numbers, elements of type $\mathbf{N}$, are used as an abstraction of types of decidable datasets. One could easily extend the language to include other decidable discrete types, like strings of symbols, which are more appropriate for input/output. However for simplicity, we only use its mathematically isomorphic counterpart $\mathbf{N}$.

2.3.7 Timer

This effect controls the passage of time, and can be used to study two different effectful phenomena. Firstly, we may want to delay the computation for a particular amount of time. Secondly, we might want to keep track of time spent on an evaluation by explicitly tagging particular subterms of a program we know to take a long time to evaluate.

We make some arbitrary choices in our implementation of timer. Firstly, we do not model inherent computation timing. We only look at time we explicitly flag. One could

use a similar mechanism to flag some other resource, like monetary cost or memory. As such, we see this example as an illustrative case of more general phenomena.

We define a countable set of positive rational time increments $\mathsf{Inc} \subset \mathbb{Q}_{>0}$,[3] where for each $c \in \mathsf{Inc}$ we have an operator $\mathsf{tick}_c(-) : \underline{\alpha} \to \underline{\alpha}$. Our effect signature is $\Sigma_{\mathrm{ti}} := \{\mathsf{tick}_c(-) : \underline{\alpha} \to \underline{\alpha} \mid c \in \mathsf{Inc}\}$, where $\mathsf{tick}_c(\underline{M})$ is the computation where evaluation gets postponed for c units of time, after which it continues with $\underline{M}$. Note that the same operator could alternatively be used to keep track of the evaluation time of a computation, using the operator as a token. If a certain part of the evaluation takes a certain amount of time, this operator can be used to mark the computation of that evaluation, so one can keep track of how many times that evaluation has occurred.

As an example, consider the functional term $\underline{M} := \lambda x.\, (\mathsf{force}(x); \mathsf{force}(x))$ and apply it to the term $V := \mathsf{thunk}(\mathsf{tick}_1(\mathsf{return}(W)))$.

$$|\underline{M}\ V| := \begin{array}{c} \mathsf{tick}_1 \\ | \\ \mathsf{tick}_1 \\ | \\ \langle W \rangle \end{array}$$

Since $\underline{M}$ forces evaluation of its argument twice, and V invokes a 'tick' when evaluated, the resulting tree exhibits two 'ticks'.

Alternatively, we can have the program determine the number of milliseconds the computation gets postponed using a single sleep operator $\mathsf{sleep}(-;-) : \mathbf{N} \times \underline{\alpha} \to \underline{\alpha}$. Moreover, as mentioned before, this example is part of a bigger group of examples modelling costs, where other resources like money or memory can be considered. These can be implemented in a similar way.

[3] The choice of using rationals is arbitrary. Instead, we could have used the natural numbers $\mathbb{N}$ or the positive reals $\mathbb{R}_{>0}$. Other total orders with commutative operations could potentially also work.

3

Behavioural equivalence

The main aim of this thesis is to study notions of equality between programs that feature effectful behaviour. This is done in an axiomatic way, establishing for an effect signature appropriate notions of behaviour for effects in the form of *modalities*. Though by necessity, this investigation is done for one language, one hopes this would be applicable to a wide range of languages with effects.

In the previous chapter, we have given operational semantics where we see computations as generating trees. However, such trees do not capture the behaviour of effectful programs, and tend to contain information which is not 'observable'. Take for instance the nondeterministic terms $\mathsf{or}(\underline{M}, \underline{N})$ and $\mathsf{or}(\underline{N}, \underline{M})$ which may operationally generate different trees, but are behaviourally indistinguishable.

This brings us to the main notion of equality in this thesis, *behavioural equivalence*. Here we look at the behaviour of a program externally. Two programs are different when they have different behaviours. To formalise this, we need to establish what the possible properties are for describing this behaviour. In this chapter, we define this notion of *behavioural property* and express them using formulas from a *logic*.

3.1 Design criteria

For every type $\underline{\underline{\mathbf{E}}}$, we are going to define a set $\mathit{Form}(\underline{\underline{\mathbf{E}}})$ of *formulas*. For each $\underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$, we use $\underline{P} \models \underline{\phi}$ to denote that the term $\underline{P}$ exhibits the property expressed by $\underline{\phi}$. In that case, we say $\underline{P}$ *satisfies* formula $\underline{\phi}$. The formulas together with the satisfaction relation $\models$ will induce a *behavioural equivalence* on terms, where two terms $\underline{P}, \underline{R}$ of the same type are behaviourally equivalent, written $\underline{P} \equiv \underline{R}$, if they satisfy the same formulas (see Definition 3.3.2 later on).

We identify three properties we desire the logic to have. This will guide our formulation of the logic and its formulas.

1. Each formula of the logic should express a property which is *behaviourally meaningful*. The formulas should align with the natural understanding and interpretation of program behaviour of effectful computation, i.e., if $\underline{P}$ has the same behaviour as $\underline{R}$, then $\underline{P} \models \underline{\phi}$ if and only if $\underline{R} \models \underline{\phi}$.

2. No program should be able to distinguish between behaviourally equivalent programs. In other words, substituting two equivalent programs $\underset{\cdot\cdot}{P} \equiv \underset{\cdot\cdot}{R}$ in another program $C[-]$ should yield equivalent programs $C[\underset{\cdot\cdot}{P}] \equiv C[\underset{\cdot\cdot}{R}]$.

3. The logic should be as expressive as possible, given the desires expressed by the previous two points.

According to the first criterion, we add to each type *basic formulas*, which are either atomic or constructed from formulas of other types. Such basic formulas express properties pertinent to the type, and thus the choice of formula depends on the type in question.

Besides these basic formulas, we close each set $\mathit{Form}(\underset{\cdot\cdot}{\mathbf{E}})$ under infinitary propositional logic. Boolean connectives can be used to combine particular facets of computational behaviour, so following criterion 3 it is reasonable to close our sets of behavioural properties under such connectives. In particular, we add countable disjunction, countable conjunction and negation to our logic[1].

We now consider the basic formulas on a type by type basis, starting with the natural numbers type. Since we have a 'case' operator for natural numbers in the language, it is possible for programs to distinguish any two different natural numbers. As such, criterium 2 motivates us to distinguish between different numbers. Of course, such a distinction is also behaviourally meaningful, as statements like $1 \neq 2$ align with our natural understanding of numbers. So both criteria 1 and 2 motivate us too include a basic value formula $\{n\} \in \mathit{Form}(\mathbf{N})$ for every $n \in \mathbb{N}$. This property checks whether a numeral expressed in the language is equal to some natural number, with the semantics of this formula given by:

$$V \models \{n\} \qquad \Longleftrightarrow \qquad V = \overline{n} \ .$$

By the closure of $\mathit{Form}(\mathbf{N})$ under countable disjunction, every subset of $\mathbb{N}$ can be represented as some value formula.

For the unit type $\mathbf{1}$, we do not require any basic value formulas. The unit type has only one value, $*$. The two subsets of this singleton set of values are identified by the formulas $\bot$ ('falsum', given as an empty disjunction), and $\top$ (the truth constant, given as an empty conjunction). These two formulas exist for any type, being satisfied respectively by all terms and by no terms.

For a function type $\mathbf{A} \to \underline{\mathbf{C}}$, the main way of observing the behaviour of its terms is to give it a specific argument and check the behaviour of the resulting computation. Given a computation $\underline{M}$ of this type, and a value V of type $\mathbf{A}$, the application $\underline{M}\ V$ gives a computation of type $\underline{\mathbf{C}}$. This motivates us to include, for every value $V \in \mathit{Terms}(\mathbf{A})$ and computation formula $\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})$, a basic value formula $(V \mapsto \underline{\phi}) \in \mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})$ with the semantics:

$$\underline{M} \models (V \mapsto \underline{\phi}) \qquad \Longleftrightarrow \qquad \underline{M}\ V \models \underline{\phi} \ .$$

[1] Since there are only countably many terms, adding countable connectives to the logic is equivalent to adding arbitrary connectives to the logic. See Lemma 3.3.1

Other natural behavioural properties can be expressed using these basic formulas together with the infinitary propositional logic. For example, given $\phi \in \mathit{Form}(\mathbf{A})$ and $\underline{\psi} \in \mathit{Form}(\underline{\mathbf{C}})$, we can construct a formula $(\phi \mapsto \underline{\psi})$ with the semantics:

$$\underline{M} \models (\phi \mapsto \underline{\psi}) \quad \iff \quad \forall V \in \mathit{Terms}(\mathbf{A}),\ V \models \phi \text{ implies } \underline{M}\ V \models \underline{\psi}\ .$$

This formula can be defined in the logic as a conjunction of $V \to \underline{\psi}$ formulas over all values $V \in \mathit{Terms}(\mathbf{A})$ such that $V \models \phi$, so

$$(\phi \mapsto \underline{\psi}) \ := \ \bigwedge\{V \to \underline{\psi} \mid \forall V \in \mathit{Terms}(\mathbf{A}), V \models \phi\}\ .$$

In Section 5.4, we shall consider the possibility of changing the basic value formulas in $\mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})$ to formulas $(\phi \mapsto \underline{\psi})$.

The basic formulas for $\mathbf{U}\,\underline{\mathbf{C}}$, $\mathbf{\Sigma}_{i\in I}\,\mathbf{A}_i$, $\mathbf{A} \times \mathbf{B}$, and $\mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}}_i$ are relatively straightforward interpretations of the behaviour of the terms at such types. In case of $\mathbf{U}\,\underline{\mathbf{C}}$ for instance, each formula $\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})$ is lifted to a formula $\langle\underline{\phi}\rangle \in \mathit{Form}(\mathbf{U}\,\underline{\mathbf{C}})$ with the semantics:

$$V \models \langle\underline{\phi}\rangle \quad \iff \quad \mathsf{force}(V) \models \underline{\phi}$$

The behavioural properties of $\mathbf{\Sigma}_{i\in I}\,\mathbf{A}_i$, $\mathbf{A} \times \mathbf{B}$, and $\mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}}_i$ are given by observing properties of the components of their terms. The precise formulation and semantics of the basic formulas of these types are given in Section 3.3.

3.2 Modalities for effects

The crucial design decision in the logic is the choice of how basic computation formulas in $\mathit{Form}(\mathbf{F}\,\mathbf{A})$ are formed. Terms of this type return values of type $\mathbf{A}$ when converging. The moment of convergence is observable, and is possibly influenced by effects. This lies in stark contrast with other computation types like $\mathbf{A} \to \underline{\mathbf{C}}$, whose terms only evaluate when necessary and the particular moment of the convergence (to a lambda term) of the evaluation is deemed unobservable. This is according to the call-by-name interpretation the call-by-push-value language gives the function type.

The only moment when effects can be observed is when terms of type $\mathbf{F}\,\mathbf{A}$ are evaluated. To interpret the way these effects can be observed, in a behaviourally meaningful way, we require a given set $\mathcal{O}$ of *modalities* depending on the algebraic effects contained in the language. The basic computation formulas in $\mathbf{F}\,\mathbf{A}$ have the form $o(\phi)$, where $o \in \mathcal{O}$ is a modality, and ϕ is a value formula from $\mathit{Form}(\mathbf{A})$. Thus a modality 'lifts' properties of values of type $\mathbf{A}$ to properties of computations of type $\mathbf{F}\,\mathbf{A}$.

Take for instance nondeterminism as defined in Subsection 2.3.3. A computation of type $\mathbf{F}\,\mathbf{A}$ will, after a (possibly infinite) sequence of binary nondeterministic decisions, return a value or diverge. One possible modality for this language is the 'diamond' modality $\Diamond$, which given a formula $\phi \in \mathit{Form}(\mathbf{A})$ creates a formula $\Diamond(\phi) \in \mathit{Form}(\mathbf{F}\,\mathbf{A})$, which describes the possibility of satisfaction of ϕ, with the informal definition:

$$\underline{M} \models \Diamond(\phi) \quad \iff \quad \underline{M} \text{ '}\mathit{may}\text{' return a value } V \text{ such that } V \models \phi\ .$$

The formal semantics of a modality o is given by a single subset $[\![o]\!] \subseteq T(\mathbf{1}) = T(Tct(\mathbf{F}\,\mathbf{1})) = T(\{*\})$ (here there is a minor abuse of notation, since the second equality is formally an isomorphism) such that for $\underline{M} : \mathbf{F}\,\mathbf{1}$, $\underline{M} \models o(\top) \iff |\underline{M}| \in [\![o]\!]$. We call $[\![o]\!]$ the *denotation* of the modality o.

In general, given a tree $t \in T(X)$ and some subset D of X, we define:

$$t[\in D] \quad := \quad t\left[x \mapsto \begin{cases} \langle * \rangle & \text{if } x \in D \\ \bot & \text{otherwise} \end{cases}\right] \quad \in T(\mathbf{1}).$$

If $t \in T(\mathbf{F}\,\mathbf{A}) = T(Tct(\mathbf{F}\,\mathbf{A}))$ and $\phi \in Form(\mathbf{A})$, we write:

$$t[\models \phi] \quad := \quad t[\in \{\mathsf{return}(V) \mid V \in Terms(\mathbf{A}), V \models \phi\}].$$

Given this, formulas of the form $o(\phi)$ have the following semantics:

$$\underline{M} \models o(\phi) \quad \iff \quad |\underline{M}|[\models \phi] \in [\![o]\!] \tag{3.1}$$

The particular modalities needed depend on the effects under consideration, and what kind of property should be considered observable. In all cases of effects under consideration, we shall asses the appropriateness of our choice of modalities using three criteria.

1. The modalities correspond to behaviourally meaningful properties, which represent behaviour patterns observable externally.
2. The modalities relate to already existing models of effects, e.g., the resulting equivalences satisfy the correct equations, as discussed in Section 3.5.
3. The modalities satisfy properties such that the resulting behavioural equivalence is compatible. This will be discussed in Subsection 3.3.2.

We now introduce suitable modalities for each of our running examples of effects.

3.2.1 Pure computation: Termination

In the case where there are no effects present, when the effect signature is $\Sigma_\emptyset := \emptyset$ as in Subsection 2.3.1, there is only one fundamental thing we can observe; that the computation has terminated/converged and returned a value.

We define our set of modalities as $\mathcal{O}_\emptyset = \{\downarrow\}$, with one *termination modality* $\downarrow$; where $\downarrow(\phi)$ asserts that a computation terminates with a return value V satisfying ϕ. The following gives both an informal and formal definition of the semantics of the formula:

$$\begin{aligned} \underline{M} \models \downarrow(\phi) \quad &\Leftrightarrow \quad \underline{M} \text{ returns a value } V \text{ such that } V \models \phi \\ &\Leftrightarrow \quad |\underline{M}| \text{ is a leaf } \langle \mathsf{return}(V) \rangle \text{ such that } V \models \phi \ . \end{aligned}$$

Note that, in the case of pure functional computation, all trees are leaves; either term leaves of the form $\langle \mathsf{return}(V) \rangle$, or nontermination leaves labelled $\bot$. The denotation of the termination modality, which determines via (3.1) the formal semantics of $\downarrow(\phi)$ stated above, is given by:

$$[\![\downarrow]\!] \ = \ \{\, \langle * \rangle \,\} \qquad \text{(where } \langle * \rangle \text{ is the tree with a single leaf } *).$$

3.2.2 Error: Detecting the message

When the language is extended with a set of possible error messages Err as in Subsection 2.3.2, giving the signature $\Sigma_{\mathrm{er}} := \{\mathsf{raise}_e() \mid e \in \mathsf{Err}\}$, the set of modalities must be extended in an appropriate manner. We define

$$\mathcal{O}_{\mathrm{er}} = \{\downarrow\} \cup \{\mathsf{E}_e \mid e \in \mathsf{Err}\},$$

using the semantics of the termination modality $\downarrow$ defined above. We have some new *error detecting modalities*, each E_e detecting error e:[2]

$$\begin{aligned} \underline{M} \models \mathsf{E}_e(\phi) \quad &\Leftrightarrow \quad \underline{M} \text{ raises error message } e \\ &\Leftrightarrow \quad |\underline{M}| \text{ is a node labelled } \mathsf{raise}_e \ . \end{aligned}$$

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

To determine the right semantics for our new modality, we define its denotation in the following way:

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

3.2.3 Nondeterminism: May and Must

As introduced in Subsection 2.3.3, we consider nondeterminism with a single effect operator $\Sigma_{\mathrm{nd}} := \{\mathsf{or}(-,-) : \alpha^2 \to \alpha\}$, which describes a binary choice made by some unknown external scheduler.

We define $\mathcal{O}_{\mathrm{nd}} = \{\Diamond, \Box\}$ consisting of two modalities. The $\Diamond$-modality determines what is possible, where we look at which formulas may be satisfied given some possible resolution of nondeterministic choices. The $\Box$-modality determines what must happen, where we look at which formulas will always be satisfied regardless of how the nondeterministic choices are resolved. We have the following definitions:

$$\begin{aligned} \underline{M} \models \Diamond(\phi) \quad &\Leftrightarrow \quad \underline{M} \text{ may return a value } V \text{ such that } V \models \phi \\ &\Leftrightarrow \quad |\underline{M}| \text{ has some leaf } \mathsf{return}(V) \text{ such that } V \models \phi \ , \\ \underline{M} \models \Box(\phi) \quad &\Leftrightarrow \quad \underline{M} \text{ must return a value } V \text{ such that } V \models \phi \\ &\Leftrightarrow \quad |\underline{M}| \text{ is finite and every leaf is of the form } \mathsf{return}(V) \text{ s.t. } V \models \phi \quad . \end{aligned}$$

Including both modalities amounts to a *neutral* view of nondeterminism, considering both the best case and the worst case scenario in a single model. Only including the

[2] Because $\mathsf{raise}_e()$ is an operation of arity 0, a raise_e node in a tree has 0 children.

$\Diamond$ modality amounts to *angelic* nondeterminism, and only including the $\Box$ modality amounts to *demonic nondeterminism*.

Because of the way the semantic definitions interact with termination, divergence and finiteness, the modalities $\Box$ and $\Diamond$ are not De Morgan duals. For instance, $\underline{\phi} := \neg(\Diamond(\neg(\phi)))$ expresses the fact that if a computation returns a value V, that value will satisfy ϕ. However, to satisfy $\underline{\phi}$, termination need not be guaranteed, so the formula is not equivalent to $\Box(\phi)$. The two modalities are not completely unrelated however, since we do have the following nice equivalence of formulas:

$$\neg(\Diamond(\neg(\phi))) \wedge \Box(\top) \quad \equiv \quad \Box(\phi).$$

The above semantics are implemented by defining the denotations of the modalities as follows:

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

Each of the three possibilities $\{\Diamond, \Box\}, \{\Diamond\}, \{\Box\}$ for $\mathcal{O}$ leads to a logic with a different behavioural equivalence. See Subsection 3.4.1 for examples of such differences.

The diamond modality $\Diamond$ can be adapted to work for countable nondeterminism, with effect operator $\mathsf{c\text{-}or}(-) : \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$, in the obvious way. We could also give a natural definition of the box modality for the countable choice operator, where e.g., $[\![\Box]\!]$ contains precisely the trees $t \in T(\mathbf{1})$ which are *well-founded* and contain only leaves labelled $\langle * \rangle$. However, this $\Box$ modality violates one of the properties introduced in Subsection 3.3.2, and can as such not be used in our proof of the forthcoming Compatibility Theorem.

Other behavioural properties of nondeterministic programs can be expressed in terms of the above modalities with the use of negation. For example:

- $\underline{M} \models \neg(\Box(\neg(\phi)))$ holds if, when $\underline{M}$ is guaranteed to terminate, then it may produce a value satisfying ϕ.
- $\underline{M} \models \neg(\Diamond(\top))$ iff $\underline{M}$ must diverge.
- $\underline{M} \models \neg(\Box(\top))$ iff $\underline{M}$ may diverge, possibly by an infinite sequence of or-operators. For instance, this holds for the term $\underline{M}$ such that $|\underline{M}| = \mathsf{or}(|\underline{M}|, \langle * \rangle)$.

3.2.4 Probability: Expected satisfaction

In the case of probability, as in Subsection 2.3.4, we take as effect signature the singleton $\Sigma_{\text{pr}} := \{\mathsf{pr}(-,-) : \underline{\alpha}^2 \to \underline{\alpha}\}$. We define:

$$\mathcal{O}_{\text{pr}} = \{\mathsf{P}_{>q} \mid q \in \mathbb{Q}, 0 \leq q \leq 1\},$$

where the modality $\mathsf{P}_{>q}$ has the semantics:

$$\underline{M} \models \mathsf{P}_{>q}(\phi) \quad \Leftrightarrow \quad \text{The probability for } \underline{M} \text{ of returning a value } V$$

$$\begin{aligned} & \text{such that } V \models \phi \text{ is larger than } q, \\ \Leftrightarrow \quad & \mathbf{P}(|\underline{M}|[\models \phi]) > q \,. \end{aligned}$$

Here, $\mathbf{P} : T(\mathbf{1}) \to [0,1]$ calculates the probability that a run through the tree t, starting at the root, and making independent fair probabilistic choices at each branching node, ends up with a leaf $\langle * \rangle$. Formally, we define for each $n \in \mathbb{N}$ a function $\mathbf{P}_n : T(\mathbf{1}) \to [0,1]$ in the following way:

$$\begin{aligned} \mathbf{P}_0(t) &:= 0 \\ \mathbf{P}_{n+1}(\langle * \rangle) &:= 1 \\ \mathbf{P}_{n+1}(\bot) &:= 0 \\ \mathbf{P}_{n+1}(\mathsf{pr}(t,r)) &:= (\mathbf{P}_n(t) + \mathbf{P}_n(r))/2 \ . \end{aligned}$$

These give lower approximations of the true probability of termination, and for each $t \in T(\mathbf{1})$ we define $\mathbf{P}(t) := \sup\{\mathbf{P}_n \mid n \in \mathbb{N}\}$. The denotation of $\mathsf{P}_{>q}$ is:

$$[\![\mathsf{P}_{>q}]\!] \ = \ \{t \in T(\mathbf{1}) \mid \mathbf{P}(t) > q\} \,.$$

Since our logic is closed under countable disjunction and conjunction, our choice of only having modalities for rational strict thresholds q is immaterial, as, for any real r with $0 \leq r \leq 1$, we can define:

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

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

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

Also, we can use negation to define modalities expressing strict and non-strict upper bounds on probabilities.

$$\mathsf{P}_{<r}(\phi) \ := \ \neg(\mathsf{P}_{\geq r}(\phi)), \qquad \mathsf{P}_{\leq r}(\phi) \ := \ \neg(\mathsf{P}_{>r}(\phi)) \ .$$

We shall see in Subsection 3.3.3 that, because of continuity issues, it is important that we include only strict lower-bound modalities in our set $\mathcal{O}_{\mathrm{pr}}$ of primitive modalities.

3.2.5 Global Store: Initial and final state

For global store, as introduced in Subsection 2.3.5, we specify a set of locations Loc, and have as algebraic effect signature $\Sigma_{\mathrm{gs}} := \{\mathsf{lookup}_l(-), \mathsf{update}_l(-;-) \mid l \in \mathsf{Loc}\}$. We define the set of global states $\mathsf{State} := \mathbb{N}^{\mathsf{Loc}}$, where for $s \in \mathsf{State}$, $s(l) = m$ means the number m is stored at location l. The modalities are given by the set:

$$\mathcal{O}_{\mathrm{gs}} = \{(s \rightarrowtail r) \mid s, r \in \mathsf{State}\},$$

where informally:

$$\underline{M} \models (s \rightarrowtail r)(\phi) \quad \Leftrightarrow \quad \text{the execution of } \underline{M}, \text{ starting in state } s, \text{ terminates in}$$

final state r, and returns a value V such that $V \models \phi$.

We make the above definition precise using the effect tree of $\underline{M}$. We define

$$exec : T(X) \times \mathsf{State} \to X \times \mathsf{State}\,,$$

for any set X, to be the least partial function satisfying for each $s \in \mathsf{State}$:

$$\begin{array}{rcl} exec(\langle x \rangle, s) & := & (x, s) \quad \text{for any } x \in X \\ exec(\mathsf{lookup}_l(n \mapsto t_n), s) & := & exec(t_{s(l)}, s) \\ exec(\mathsf{update}_{l,n}(t), s) & := & exec(t, s[l := n]) \ . \end{array}$$

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

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

To obtain the above semantics, we define the denotation of the modalities using *exec* function on $\{*\}$:

$$[\![(s \rightarrowtail r)]\!] \ = \ \{t \in T(\mathbf{1}) \mid exec(t, s) = (*, r)\} \ .$$

The modality implements a notion of *total correctness*, in the sense that $\underline{M} \models (s \rightarrowtail r)(\phi)$ holds if and only if $\underline{M}$, starting with state s, *will terminate* with state r producing a value V such that $V \models \phi$. It is also possible to define *partial correctness* assertions in the logic. For instance,

$$\underline{M} \ \models \ \neg(s \rightarrowtail r)(\neg(\phi)) \ \wedge \bigwedge_{r' \in \mathsf{State}, r' \neq r} \neg(s \rightarrowtail r')(\top)$$

holds if and only if $\underline{M}$, starting with state s, either diverges or terminates with state r producing a value V such that $V \models \phi$.

3.2.6 Input/Output: Traces of communication

Consider the input/output effect from Subsection 2.3.6, with the signature given by $\Sigma_{\text{io}} := \{\mathsf{read}(-) : \underline{\alpha}^{\mathbf{N}} \to \alpha, \mathsf{write}(-;-) : \mathbf{N} \times \underline{\alpha} \to \underline{\alpha}\}$. We define an *i/o-trace* to be a finite word w over the alphabet of characters

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

The idea is that such a word represents an input/output sequence, where $?n$ means the number n is given in response to an input prompt given by the read operator, and

$!n$ means that the program outputs n via the write_n operator. We define the set of modalities:

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

The intuitive semantics of these modalities are as follows:

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

In order to define the semantics of formulas precisely, we first define satisfaction relations $t \models \langle w\rangle\!\downarrow P$ and $t \models \langle w\rangle_{\ldots}$, between $t \in T(X)$ and $P \subseteq X$, by induction on words. (Note that we are overloading the $\models$ symbol.) In the following, we write ε for the empty word, and we use textual juxtaposition for concatenation of words.

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

The formal semantics of modalities is now defined as follows:

$$\begin{aligned}
\underline{M} \models \langle w\rangle\!\downarrow(\phi) \quad &\Leftrightarrow \quad |\underline{M}| \models \langle w\rangle\!\downarrow\{\mathsf{return}(V) \mid V \models \phi\} \\
\underline{M} \models \langle w\rangle_{\ldots}(\phi) \quad &\Leftrightarrow \quad |\underline{M}| \models \langle w\rangle_{\ldots}\,.
\end{aligned}$$

We may write $\langle w\rangle i$ where i is a variable ranging over $\{\downarrow, \ldots\}$. Note that $\langle w\rangle_{\ldots}$, like E_e in Example 2.3.2, has a redundant formula argument. Also, note that our modalities for input/output could naturally be formed by combining the termination modality $\downarrow$, which lifts value formulas to computation formulas, with sequences of atomic modalities $\langle ?n\rangle$ and $\langle !n\rangle$ acting directly on computation formulas.

The above semantics is formally obtained by defining:

$$\begin{aligned}
[\![\langle w\rangle\!\downarrow]\!] &= \{t \mid t \models \langle w\rangle\!\downarrow\{*\}\} \\
[\![\langle w\rangle_{\ldots}]\!] &= \{t \mid t \models \langle w\rangle_{\ldots}\}\,.
\end{aligned}$$

Input/Output is the only example given in this thesis where trees of unit type are behaviourally equivalent if and only if they are the same. In particular, for any tree $t \in T(\{*\})$ there is some set of modalities $A \subseteq \mathcal{O}$ such that $\bigcap_{o\in A}[\![o]\!] = \{r \in T(\{*\}) \mid t \leq r\}$. This distinction between different trees is behaviourally meaningful, since with a correct series of inputs, any branch in the effect tree can be visited and can be observed by whoever is choosing the inputs.

3.2.7 Timer: Time taken

As given in Subsection 2.3.7, the signature of the timer effect is given by $\Sigma_{\mathrm{ti}} := \{\mathsf{tick}_c(-) : \underline{\alpha} \to \underline{\alpha} \mid c \in \mathsf{Inc}\}$, where $\mathsf{Inc} \subseteq \mathbb{Q}_{>0}$ is some set of time increments. For each rational number $q \in \mathbb{Q}$, we consider three types of modalities: $\mathsf{C}_{\leq q}$, $\mathsf{C}_{\geq q}$ and $\mathsf{C}^{\uparrow}_{>q}$, where the modalities are informally defined as follows:

$$\begin{aligned}
\underline{M} \models \mathsf{C}_{\leq q}(\phi) &\quad\Leftrightarrow\quad \underline{M} \text{ terminates with } V \text{ in at most } q\text{-time, where } V \models \phi.\\
\underline{M} \models \mathsf{C}_{\geq q}(\phi) &\quad\Leftrightarrow\quad \underline{M} \text{ terminates with } V \text{ after at least } q\text{-time, where } V \models \phi.\\
\underline{M} \models \mathsf{C}^{\uparrow}_{>q}(\phi) &\quad\Leftrightarrow\quad \underline{M} \text{ delays computation for more than } q.
\end{aligned}$$

This is formally defined using a function $\mathit{clock} : \mathbb{N} \times T(X) \to (X \cup \{\bot\}) \times \mathbb{Q}$ where:

$$\begin{aligned}
\mathit{clock}_0(t) = \mathit{clock}_{n+1}(\bot) &:= (\bot, 0)\\
\mathit{clock}_{n+1}(\langle x \rangle) &:= (x, 0)\\
\mathit{clock}_{n+1}(\mathsf{tick}_c(t)) &:= (\pi_0(\mathit{clock}_n(t)), \pi_1(\mathit{clock}_n(t)) + c)\ .
\end{aligned}$$

Here π_0 and π_1 used for left and right projections respectively. We can define the denotation as:

$$[\![\mathsf{C}_{\leq q}]\!] := \{t \in T(\{*\}) \mid \exists n \in \mathbb{N}, \mathit{clock}_n(t) = (*, p) \wedge p \leq q\}$$

$$[\![\mathsf{C}_{\geq q}]\!] := \{t \in T(\{*\}) \mid \exists n \in \mathbb{N}, \mathit{clock}_n(t) = (*, p) \wedge p \geq q\}$$

$$[\![\mathsf{C}^{\uparrow}_{>q}]\!] := \{t \in T(\{*\}) \mid \exists n \in \mathbb{N}, \pi_1(\mathit{clock}_n(t)) > q\}\ .$$

We take three different sets of modalities, corresponding to three different theories for the timer effects.

1. The down-modalities $\mathcal{O}^{\downarrow}_{\mathrm{ti}} = \{\mathsf{C}_{\leq q} \mid q \in \mathbb{Q}_{\geq 0}\}$.
2. The up-modalities $\mathcal{O}^{\uparrow}_{\mathrm{ti}} = \{\mathsf{C}_{\geq q}, \mathsf{C}^{\uparrow}_{>q} \mid q \in \mathbb{Q}_{\geq 0}\}$.
3. The combined modalities $\mathcal{O}_{\mathrm{ti}} = \mathcal{O}^{\downarrow}_{\mathrm{ti}} \cup \mathcal{O}^{\uparrow}_{\mathrm{ti}}$.

We see the down-modalities as the primary way of interpreting this effect, since it looks at the passage of time as a negative thing. Moreover, any diverging computation takes an infinite amount of time, and hence $\mathsf{tick}_c(\Omega)$ should be seen as indistinguishable from Ω. The down-modalities formally implement this sentiment.

Of course, there might be other interpretations of the timer effect for which the other modalities are more natural. E.g. one could see $\mathsf{tick}_{1000}(\Omega)$ as dispensing a cash price to the user of the computer, which can be interpreted as a positive event. In such a case, the up-modalities may be a better interpretation of the effect.

3.3 Behavioural preorders

We have defined all the tools needed to construct our logic of behavioural properties, and the resulting behavioural equivalence. In this section, we assume that we have an effect signature Σ and a set of modalities $\mathcal{O}$, each with an associated denotation $[\![-]\!]$. We will first give a precise formulation of the construction of the formulas, where $\mathit{Form}(\underline{\underline{\mathbf{E}}})$ gives the set of formulas of type $\underline{\underline{\mathbf{E}}}$.

We use $\phi, \psi, \ldots$ for formulas over value types, $\underline{\phi}, \underline{\psi}$ for formulas over computation types, and $\underline{\underline{\phi}}, \underline{\underline{\psi}}, \ldots$ for formulas over a non-specific type $\underline{\underline{\mathbf{E}}}$ (i.e. either value or computation). Figure 3.1 gives the inductive rules for generating these formulas.

$$(1)\ \frac{n \in \mathbb{N}}{\{n\} \in \mathit{Form}(\mathbf{N})}$$

$$(2)\ \frac{\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})}{\langle \underline{\phi} \rangle \in \mathit{Form}(\mathbf{U}\,\underline{\mathbf{C}})} \qquad (3)\ \frac{j \in I \qquad \phi \in \mathit{Form}(\mathbf{A}_j)}{(j, \phi) \in \mathit{Form}(\mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i)}$$

$$(4)\ \frac{\phi \in \mathit{Form}(\mathbf{A})}{\pi_0(\phi) \in \mathit{Form}(\mathbf{A} \times \mathbf{B})} \qquad (5)\ \frac{\phi \in \mathit{Form}(\mathbf{B})}{\pi_1(\phi) \in \mathit{Form}(\mathbf{A} \times \mathbf{B})}$$

$$(6)\ \frac{V \in \mathit{Terms}(\mathbf{A}) \qquad \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})}{(V \mapsto \underline{\phi}) \in \mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})} \qquad (7)\ \frac{o \in \mathcal{O} \qquad \phi \in \mathit{Form}(\mathbf{A})}{o(\phi) \in \mathit{Form}(\mathbf{F}\,\mathbf{A})}$$

$$(8)\ \frac{\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}_j)}{(j \mapsto \underline{\phi}) \in \mathit{Form}(\mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i)} \qquad (9)\ \frac{X \subseteq_{\text{countable}} \mathit{Form}(\underline{\underline{\mathbf{E}}})}{\bigvee X \in \mathit{Form}(\underline{\underline{\mathbf{E}}})}$$

$$(10)\ \frac{X \subseteq_{\text{countable}} \mathit{Form}(\underline{\underline{\mathbf{E}}})}{\bigwedge X \in \mathit{Form}(\underline{\underline{\mathbf{E}}})} \qquad (11)\ \frac{\underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})}{\neg(\underline{\underline{\phi}}) \in \mathit{Form}(\underline{\underline{\mathbf{E}}})}$$

Figure 3.1: Formula constructors

Satisfaction of the formulas is given by the following rules:

$$\begin{aligned}
V \models \{n\} &\iff V = \overline{n}. \\
V \models \langle \underline{\phi} \rangle &\iff \mathsf{force}(V) \models \underline{\phi}. \\
(i, V) \models (j, \phi) &\iff i = j \text{ and } V \models \phi. \\
(V, W) \models \pi_0(\phi) &\iff V \models \phi. \\
(V, W) \models \pi_1(\phi) &\iff W \models \phi. \\
\underline{M} \models (V \mapsto \underline{\phi}) &\iff \underline{M}\ V \models \underline{\phi}. \\
\underline{M} \models o(\phi) &\iff |\underline{M}|[\models \phi] \in [\![o]\!]. \\
\underline{M} \models (j \mapsto \underline{\phi}) &\iff \underline{M}\ j \models \underline{\phi}. \\
\underline{\underline{P}} \models \bigvee X &\iff \exists \underline{\underline{\phi}} \in X.\ \underline{\underline{P}} \models \underline{\underline{\phi}}. \\
\underline{\underline{P}} \models \bigwedge X &\iff \forall \underline{\underline{\phi}} \in X.\ \underline{\underline{P}} \models \underline{\underline{\phi}}. \\
\underline{\underline{P}} \models \neg(\underline{\underline{\phi}}) &\iff \neg(\underline{\underline{P}} \models \underline{\underline{\phi}}).
\end{aligned}$$

Note that conjunctions and disjunctions are formed over countable sets of formulas only. The choice to not include connectives over larger sets is inconsequential, since there are only countably many terms, any disjunction or conjunctions over a set of formulas can be reduced to a disjunction or conjunction over a countable set of formulas.

Lemma 3.3.1. *For any type* $\underline{\underline{\mathbf{E}}}$ *and any set* $X \subseteq \mathit{Form}(\underline{\underline{\mathbf{E}}})$*, there are countable subsets* $X_\vee, X_\wedge \subseteq X$ *such that:*

1. $\underline{\underline{P}} \models \bigvee X_\vee \iff \exists \underline{\underline{\phi}} \in X.\ \underline{\underline{P}} \models \underline{\underline{\phi}}$.
2. $\underline{\underline{P}} \models \bigwedge X_\wedge \iff \forall \underline{\underline{\phi}} \in X.\ \underline{\underline{P}} \models \underline{\underline{\phi}}$.

Proof. Let $f : \mathbb{N} \to \mathit{Terms}(\underline{\underline{\mathbf{E}}})$ be an enumeration of all terms, which exists since the number of terms is countable. We construct sequences of sets $X_\vee^n$ and $X_\wedge^n$ inductively, with $X_\vee^0 = X_\wedge^0 = \emptyset$. For each $n \in \mathbb{N}$ we do the following:

If there is a $\underline{\underline{\phi}} \in X$ such that $f(n) \models \underline{\underline{\phi}}$, then choose such a $\underline{\underline{\phi}}$ and define $X_\vee^{n+1} = X_\vee^n \cup \{\underline{\underline{\phi}}\}$. If such a $\underline{\underline{\phi}}$ does not exist, define $X_\vee^{n+1} = X_\vee^n$. If there is a $\underline{\underline{\phi}} \in X$ such that $f(n) \not\models \underline{\underline{\phi}}$, then define $X_\wedge^{n+1} = X_\wedge^n \cup \{\underline{\underline{\phi}}\}$, else define $X_\wedge^{n+1} = X_\wedge^n$.

The resulting sets $X_\vee := \bigcup_n X_\vee^n$ and $X_\wedge := \bigcup_n X_\wedge^n$ have the desired properties. □

A *basic formula* is a formula which on the top level does not have a conjunction, disjunction or negation. A *¬-basic* formula, is either a basic formula or the negation of a basic formula. We will distinguish between two different logics, one with and one without negations:

- All formulas together form the *general logic* $\mathcal{V}$, where the 'V' stands for the fact that functions are tested by checking for a specific *Value* argument[3].
- The formulas without negation symbols ¬ form the *positive logic* $\mathcal{V}^+$. Of course, the positive logic forms a subset of the general logic.

3.3.1 Logical preorders

We can define the preorders resulting from the logic. For a subset $\mathcal{L} \subseteq \mathcal{V}$, also called a *fragment* of $\mathcal{V}$, and a type $\underline{\underline{\mathbf{E}}}$, we write $\mathit{Form}(\underline{\underline{\mathbf{E}}})_\mathcal{L}$ for $\mathit{Form}(\underline{\underline{\mathbf{E}}}) \cap \mathcal{L}$.

Definition 3.3.2. For any subset (fragment) of the logic $\mathcal{L} \subseteq \mathcal{V}$, we define the *logical preorder* $\sqsubseteq_\mathcal{L}$ such that:

$$\forall \underline{\underline{P}}, \underline{\underline{R}} : \underline{\underline{\mathbf{E}}},\quad \underline{\underline{P}} \sqsubseteq_\mathcal{L} \underline{\underline{R}} \iff (\forall \underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})_\mathcal{L},\ \underline{\underline{P}} \models \underline{\underline{\phi}} \Rightarrow \underline{\underline{R}} \models \underline{\underline{\phi}}).$$

Here we use $\underline{\underline{P}} : \underline{\underline{\mathbf{E}}}$ as a shorthand for $\underline{\underline{P}} \in \mathit{Terms}(\underline{\underline{\mathbf{E}}})$.

The *general behavioural preorder* $\sqsubseteq$ is the logical preorder $\sqsubseteq_\mathcal{V}$, whereas the *positive behavioural preorder* $\sqsubseteq^+$ is the logical preorder $\sqsubseteq_{\mathcal{V}^+}$. Because these preorders are

[3] As opposed to the pure logic $\mathcal{F}$, where 'F' stands for *Formula*, introduced in Section 5.4

fundamental, we leave out the subscripts. We write $\equiv_{\mathcal{L}}$, $\equiv$ and $\equiv^{+}$ for the equivalence induced by the preorders $\sqsubseteq_{\mathcal{L}}$, $\sqsubseteq$, and $\sqsubseteq^{+}$ respectively:

$$\forall \underline{P}, \underline{R} : \underline{\mathbf{E}}, \quad \underline{P} \equiv_{\mathcal{L}} \underline{R} \iff (\forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{E}})_{\mathcal{L}},\ \underline{P} \models \underline{\phi} \Leftrightarrow \underline{R} \models \underline{\phi}).$$

Note that for any fragment $\mathcal{L} \subseteq \mathcal{V}$, we always have that $(\sqsubseteq) \subseteq (\sqsubseteq_{\mathcal{L}})$ simply because fewer properties are tested by $\mathcal{L}$ then by $\mathcal{V}$. We study some other general results.

Lemma 3.3.3. *For any fragment of the logic $\mathcal{L} \subseteq \mathcal{V}$, $\sqsubseteq_{\mathcal{L}}$ is reflexive and transitive.*

Proof. This is a consequence of the reflexivity and transitivity of the logical implication '$\Rightarrow$'. □

Lemma 3.3.4. *The general behavioural preorder $\sqsubseteq$ is symmetric, so $(\sqsubseteq) = (\equiv)$.*

Proof. Assume $\underline{P} \sqsubseteq \underline{R}$ and $\underline{R} \models \underline{\phi}$. Then if $\underline{P} \not\models \underline{\phi}$, it holds that $\underline{P} \models \neg\underline{\phi}$ and hence $\underline{R} \models \neg\underline{\phi}$ which leads to a contradiction. We conclude that $\underline{P} \models \underline{\phi}$, which is for all such $\underline{\phi}$, so $\underline{R} \sqsubseteq \underline{P}$. □

Henceforth we will use $\equiv$ instead of $\sqsubseteq$, since the two coincide.

The following result can be established from the fact that satisfaction of conjunctions, disjunctions and negations are completely determined by the satisfaction of the subformulas (the formulas over which the connectives are taken).

Lemma 3.3.5. *The preorders $\equiv$ and $\sqsubseteq^{+}$ are completely determined by basic formulas. For example, we can state the following classifications of the preorders:*

1. *$V \sqsubseteq^{+}_{\mathbf{U}\,\underline{\mathbf{C}}} W$ if and only if $\forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})_{\mathcal{V}^{+}}$ we have $V \models \langle\underline{\phi}\rangle \Rightarrow W \models \langle\underline{\phi}\rangle$.*
2. *$\underline{M} \equiv_{\mathbf{F}\,\mathbf{A}} \underline{N}$ if and only if $\forall o \in \mathcal{O}$ and $\phi \in \mathit{Form}(\mathbf{A})$ we have $\underline{M} \models o(\phi) \Leftrightarrow \underline{N} \models o(\phi)$.*

Similar properties hold at the other types.

Proof. Let $\underline{\mathbf{E}}$ be some type, and assume that for two terms $\underline{P}, \underline{R} : \underline{\mathbf{E}}$, and for any basic formula $\underline{\phi}$, $\underline{P} \models \underline{\phi} \Leftrightarrow \underline{R} \models \underline{\phi}$. We prove by induction on $\underline{\psi} \in \mathit{Form}(\underline{\mathbf{E}})$ that $\underline{P} \models \underline{\psi} \Leftrightarrow \underline{R} \models \underline{\psi}$. The induction base is the case where $\underline{\psi}$ is a basic formula. Hence the desired statements holds by assumption.

Now for the induction step. Assume by induction hypothesis that for a countable subset $X \subseteq \mathit{Form}(\underline{\mathbf{E}})$, it holds that for any $\underline{\phi} \in X, \underline{P} \models \underline{\phi} \Leftrightarrow \underline{R} \models \underline{\phi}$. Then $\underline{P} \models \bigvee X \Leftrightarrow \exists \underline{\phi} \in X, \underline{P} \models \underline{\phi} \Leftrightarrow \exists \underline{\phi} \in X, \underline{R} \models \underline{\phi} \Leftrightarrow \underline{R} \models \bigvee X$. Similarly, $\underline{P} \models \bigwedge X \Leftrightarrow \forall \underline{\phi} \in X, \underline{P} \models \underline{\phi} \Leftrightarrow \forall \underline{\phi} \in X, \underline{R} \models \underline{\phi} \Leftrightarrow \underline{R} \models \bigwedge X$. Lastly, assume as induction hypothesis that the statement holds for $\underline{\phi}$, then since $\underline{P} \models \underline{\phi} \Leftrightarrow \underline{R} \models \underline{\phi}$, it holds that $\underline{P} \models \neg\underline{\phi} \Leftrightarrow \underline{R} \models \neg\underline{\phi}$.

This finishes the induction. The proof for the positive behavioural preorder $\sqsubseteq^{+}$ is similar, and can be obtained by replacing all instances of '$\Leftrightarrow$' in the above proof by '$\Rightarrow$', and removing the case of negation. □

Lemma 3.3.6. *Suppose $\mathcal{L}$ is either $\mathcal{V}$ or $\mathcal{V}^+$. For any term $\underline{\underline{P}}$ there is a formula $\chi_{\underline{\underline{P}}}$ such that $\underline{\underline{R}} \models \chi_{\underline{\underline{P}}} \iff \underline{\underline{P}} \sqsubseteq_{\mathcal{L}} \underline{\underline{R}}$. Moreover, $\chi_{\underline{\underline{P}}}$ can be expressed as a conjunction over $\neg$-basic formulas in case of $\mathcal{V}$ (and over basic formulas in case of $\mathcal{V}^+$).*

Proof. We prove the statement for $\mathcal{L} = \mathcal{V}$, where $\sqsubseteq_{\mathcal{L}} = \equiv$. For any $\underline{\underline{R}}$ such that $\neg(\underline{\underline{P}} \equiv \underline{\underline{R}})$, we can find by Lemma 3.3.5 a basic formula $\underline{\underline{\phi}}$ such that $\underline{\underline{P}} \models \underline{\underline{\phi}} \not\Leftrightarrow \underline{\underline{R}} \models \underline{\underline{\phi}}$. So we can find a $\neg$-basic formula $\underline{\underline{\psi}}^{\neg \underline{\underline{R}}}$, which is either $\underline{\underline{\phi}}$ or its negation, such that $\underline{\underline{P}} \models \underline{\underline{\psi}}^{\neg \underline{\underline{R}}}$ and $\underline{\underline{R}} \not\models \underline{\underline{\psi}}^{\neg \underline{\underline{R}}}$. We choose such a formula for each $\underline{\underline{R}}$ for which $\neg(\underline{\underline{P}} \sqsubseteq_{\mathcal{L}} \underline{\underline{R}})$, and define $X_{\underline{\underline{P}}} := \{\underline{\underline{\psi}}^{\neg \underline{\underline{R}}} \mid \neg(\underline{\underline{P}} \sqsubseteq_{\mathcal{L}} \underline{\underline{R}})\}$. We prove that $\chi_{\underline{\underline{P}}} := \bigwedge X_{\underline{\underline{P}}}$ has the desired properties, noting that it is a conjunction over $\neg$-basic formulas. The set $X_{\underline{\underline{P}}}$ is countable, as it is a subset of the countable set of terms.

If $\underline{\underline{P}} \sqsubseteq_{\mathcal{L}} \underline{\underline{Q}}$, then for any $\underline{\underline{\phi}} \in X_{\underline{\underline{P}}}$, since $\underline{\underline{P}} \models \underline{\underline{\phi}}$, it holds that $\underline{\underline{Q}} \models \underline{\underline{\phi}}$. Conversely, suppose $\underline{\underline{Q}} \models \bigwedge X_{\underline{\underline{P}}}$, and assume $\underline{\underline{P}} \not\sqsubseteq_{\mathcal{L}} \underline{\underline{Q}}$. Then $\underline{\underline{\psi}}^{\neg \underline{\underline{Q}}} \in X_{\underline{\underline{P}}}$ and $\neg(\underline{\underline{Q}} \models \underline{\underline{\psi}}^{\neg \underline{\underline{Q}}})$, which directly contradicts $\underline{\underline{Q}} \models \bigwedge X_{\underline{\underline{P}}}$. We can conclude that $\underline{\underline{P}} \sqsubseteq_{\mathcal{L}} \underline{\underline{Q}}$, so $\underline{\underline{P}} \sqsubseteq_{\mathcal{L}} \underline{\underline{Q}} \Leftrightarrow \underline{\underline{Q}} \models \chi_{\underline{\underline{P}}}$.

The proof for $\mathcal{L} = \mathcal{V}^+$ is a simplification of the above proof. □

3.3.2 Compatibility

The defined formulas only test properties of closed terms (terms with empty context). To deal with open terms, we will make use of the *open extension* of our behavioural equivalence on closed terms.

Definition 3.3.7. A relation $\mathcal{R}$ on terms is *well-typed*, if it only relates terms of the same type and context. Given a well-typed relation $\mathcal{R}$ on closed terms, the *open extension* $\mathcal{R}^\circ$ of $\mathcal{R}$ is the well-typed relation on open terms such that:

$$x_1 : \mathbf{A}_1, \ldots, x_n : \mathbf{A}_n \vdash \underline{\underline{P}} \mathcal{R}^\circ \underline{\underline{R}} : \underline{\underline{\mathbf{E}}} \iff$$

$$\forall V_1 : \mathbf{A}_1, \ldots, V_n : \mathbf{A}_n, \underline{\underline{P}}[V_1/x_1, \ldots, V_n/x_n] \,\mathcal{R}\, \underline{\underline{R}}[V_1/x_1, \ldots, V_n/x_n] : \underline{\underline{\mathbf{E}}}$$

We desire the open extension of the behavioural preorders to be *compatible*, in the sense of [24, 41, 75]. In other words, the preorders should be precongruences and hence be preserved under program composition. Compatibility can be formulated using an operation on well-typed relations of open terms, the *compatible refinement* $\widehat{\mathcal{R}}$, whose rules are given in Figure 3.2. A relation $\mathcal{R}$ is *compatible* if $\widehat{\mathcal{R}} \subseteq \mathcal{R}$. Note that the compatible refinement $\widehat{\mathcal{R}}$ of a well-typed relation is not necessarily compatible.

We now state the main theorem of this chapter, that under sufficient conditions, which we will define later, the behavioural equivalence and the positive behavioural preorder are compatible. The proof of this theorem will be concluded at the end of Chapter 4.

Theorem 3.3.8 (The Compatibility Theorem)**.** *If $\mathcal{O}$ is a decomposable set of Scott-open modalities then the open extensions of the full behavioural equivalence $\equiv$ and the positive behavioural preorder $\sqsubseteq^+$ are both compatible.*

We work towards defining the two properties (decomposability and Scott openness) mentioned in the theorem.

$$\frac{}{\Gamma \vdash * \,\widehat{\mathcal{R}}\, * : \mathbf{1}}\mathbf{C1} \qquad \frac{}{\Gamma \vdash \mathsf{Z}\,\widehat{\mathcal{R}}\,\mathsf{Z} : \mathbf{N}}\mathbf{C2} \qquad \frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{N}}{\Gamma \vdash \mathsf{S}(V)\,\widehat{\mathcal{R}}\,\mathsf{S}(V') : \mathbf{N}}\mathbf{C3}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{N} \qquad \Gamma \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \underline{\mathbf{C}} \qquad \Gamma, x : \mathbf{N} \mathrel{\underline{\vdash}} \underline{N} \,\mathcal{R}\, \underline{N'} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} (\mathsf{case}\ V\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\})\,\widehat{\mathcal{R}}\,(\mathsf{case}\ V'\ \mathsf{of}\ \{\underline{M'}, \mathsf{S}(x) \Rightarrow \underline{N'}\}) : \underline{\mathbf{C}'}}\mathbf{C4}$$

$$\frac{}{\Gamma, x : \mathbf{A}, \Gamma' \vdash x\,\widehat{\mathcal{R}}\,x : \mathbf{A}}\mathbf{C5} \qquad \frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{A} \qquad \Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} (\mathsf{let}\ x\ \mathsf{be}\ V.\, \underline{M})\,\widehat{\mathcal{R}}\,(\mathsf{let}\ x\ \mathsf{be}\ V'.\, \underline{M'}) : \underline{\mathbf{C}}}\mathbf{C6}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{A}}{\Gamma \vdash \mathsf{return}(V)\,\widehat{\mathcal{R}}\,\mathsf{return}(V') : \mathbf{F}\,\mathbf{A}}\mathbf{C7} \qquad \frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \mathbf{F}\,\mathbf{A} \qquad \Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{N} \,\mathcal{R}\, \underline{N'} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} (\underline{M}\ \mathsf{to}\ x.\, \underline{N})\,\widehat{\mathcal{R}}\,(\underline{M'}\ \mathsf{to}\ x.\, \underline{N'}) : \underline{\mathbf{C}}}\mathbf{C8}$$

$$\frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \underline{\mathbf{C}}}{\Gamma \vdash \mathsf{thunk}(\underline{M})\,\widehat{\mathcal{R}}\,\mathsf{thunk}(\underline{M'}) : \mathbf{U}\,\underline{\mathbf{C}}}\mathbf{C9} \qquad \frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{U}\,\underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{force}(V)\,\widehat{\mathcal{R}}\,\mathsf{force}(V') : \underline{\mathbf{C}}}\mathbf{C10}$$

$$\frac{\Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} (\lambda x.\, \underline{M})\,\widehat{\mathcal{R}}\,(\lambda x.\, \underline{M'}) : \mathbf{A} \to \underline{\mathbf{C}}}\mathbf{C11} \qquad \frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{A} \qquad \Gamma \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \mathbf{A} \to \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} (\underline{M}\ V)\,\widehat{\mathcal{R}}\,(\underline{M'}\ V') : \underline{\mathbf{C}}}\mathbf{C12}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{A}_j \qquad j \in I}{\Gamma \vdash (j, V)\,\widehat{\mathcal{R}}\,(j, V') : \mathbf{\Sigma}_{i \in I}\,\mathbf{A}_i}\mathbf{C13}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{\Sigma}_{i \in I}\,\mathbf{A}_i \qquad \Gamma, x : \mathbf{A}_j \mathrel{\underline{\vdash}} \underline{M_j} \,\mathcal{R}\, \underline{M'_j} : \underline{\mathbf{C}} \ \text{ for each } \ j \in I}{\Gamma \mathrel{\underline{\vdash}} (\mathsf{pm}\ V\ \mathsf{as}\ \{\ldots, (i.x).\underline{M_i}, \ldots\})\,\widehat{\mathcal{R}}\,(\mathsf{pm}\ V'\ \mathsf{as}\ \{\ldots, (i.x).\underline{M'_i}, \ldots\}) : \underline{\mathbf{C}}}\mathbf{C14}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{A} \qquad \Gamma \vdash W \,\mathcal{R}\, W' : \mathbf{A}'}{\Gamma \vdash (V, W)\,\widehat{\mathcal{R}}\,(V', W') : \mathbf{A} \times \mathbf{A}'}\mathbf{C15}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{A} \times \mathbf{A}' \qquad \Gamma, x : \mathbf{A}, y : \mathbf{A}' \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{pm}\ V\ \mathsf{as}\ (x, y).\underline{M}\,\widehat{\mathcal{R}}\,\mathsf{pm}\ V'\ \mathsf{as}\ (x, y).\underline{M} : \underline{\mathbf{C}}}\mathbf{C16}$$

$$\frac{\Gamma \mathrel{\underline{\vdash}} \underline{M_i} \,\mathcal{R}\, \underline{M'_i} : \underline{\mathbf{C}_i} \ \text{ for each } \ i \in I}{\Gamma \mathrel{\underline{\vdash}} \langle \underline{M_i} \mid i \in I \rangle\,\widehat{\mathcal{R}}\,\langle \underline{M'_i} \mid i \in I \rangle : \mathbf{\Pi}_{i \in I}\,\underline{\mathbf{C}_i}}\mathbf{C17} \qquad \frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \mathbf{\Pi}_{i \in I}\,\underline{\mathbf{C}_i} \qquad j \in I}{\Gamma \mathrel{\underline{\vdash}} (\underline{M}\ j)\,\widehat{\mathcal{R}}\,(\underline{M'}\ j) : \underline{\mathbf{C}_j}}\mathbf{C18}$$

$$\frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \mathbf{U}\,\underline{\mathbf{C}} \to \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{fix}(\underline{M})\,\widehat{\mathcal{R}}\,\mathsf{fix}(\underline{M'}) : \underline{\mathbf{C}}}\mathbf{C19}$$

$$\frac{\Gamma \vdash V_i \,\mathcal{R}\, V'_i : \mathbf{N} \ \text{ for each } \ 1 \leq i \leq n \qquad \Gamma \mathrel{\underline{\vdash}} \underline{M_j} \,\mathcal{R}\, \underline{M'_j} : \underline{\mathbf{C}} \ \text{ for each } \ 1 \leq j \leq m}{\Gamma \mathrel{\underline{\vdash}} \mathsf{op}(V_1, \ldots, V_n, \underline{M_1}, \ldots, \underline{M_m})\,\widehat{\mathcal{R}}\,\mathsf{op}(V'_1, \ldots, V'_n, \underline{M'_1}, \ldots, \underline{M'_m}) : \underline{\mathbf{C}}}\mathbf{C20}$$

$$\frac{\Gamma \vdash V_i \,\mathcal{R}\, V'_i : \mathbf{N} \ \text{ for each } \ 1 \leq i \leq n \qquad \Gamma, x : \mathbf{N} \mathrel{\underline{\vdash}} \underline{M} \,\mathcal{R}\, \underline{M'} : \mathbf{N} \to \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{op}(V_1, \ldots, V_n, x \mapsto \underline{M})\,\widehat{\mathcal{R}}\,\mathsf{op}(V'_1, \ldots, V'_n, x \mapsto \underline{M'}) : \underline{\mathbf{C}}}\mathbf{C21}$$

Figure 3.2: Compatible refinement.

Definition 3.3.9. We say that $o \in \mathcal{O}$ is *upwards closed* if $[\![o]\!]$ is an upper-closed subset of $T\mathbf{1}$; i.e., if $t \in [\![o]\!]$ and $t \leq t'$ implies $t' \in [\![o]\!]$.

Definition 3.3.10. We say that $o \in \mathcal{O}$ is *Scott-open* if $[\![o]\!]$ is an open subset in the Scott topology on $T(\mathbf{1})$; i.e., $[\![o]\!]$ is upwards closed and, whenever $t_1 \leq t_2 \leq \ldots$ is an ascending chain in $T\mathbf{1}$ with supremum $\bigsqcup_i t_i \in [\![o]\!]$, we have $t_n \in [\![o]\!]$ for some n.

The main use for Scott openness is expressed by the following lemma.

Lemma 3.3.11. *If $o \in \mathcal{O}$ is a Scott-open modality, then $\underline{M} \models o(\phi)$ holds if and only if there is an $n \in \mathbb{N}$ such that $|\underline{M}|_n[\models \phi] \in [\![o]\!]$.*

Proof. This is a simple consequence of the fact that $\bigsqcup_n(|\underline{M}|_n[\models \phi]) = |\underline{M}|[\models \phi]$. □

We will show in Subsection 3.3.3 that all the modalities given for the examples are Scott-open. This is because for any tree $t \in [\![o]\!]$ there is a finite subtree $t' \leq t$ such that $t' \in [\![o]\!]$. E.g., if $t \in [\![\Diamond]\!]$, then t has a leaf labelled $\langle * \rangle$, and there is a finite subtree t' of t which contains that leaf as well, hence $t' \in [\![\Diamond]\!]$.

An example of a modality which is not Scott-open is the nondeterministic must modality $\Box$ as defined for *countable* nondeterministic choice in Subsection 3.2.3. In a finite tree, any node of countable nondeterministic choice will have infinitely many diverging $\bot$-leaves as children. As such, guaranteed termination can not be checked at a finite level. This is the reason why we only consider binary nondeterministic choice.

A potential *open question* to investigate in the future may be to check whether the theory established in this thesis, for instance Theorem 3.3.8, still applies to modalities that are not Scott open. Such a result would be applicable to countable nondeterminism.

Though Scott openness will be required in the main inductive proof of the Compatibility Theorem (see Lemma 4.3.2), it is not a necessary condition in a lot of auxiliary proofs. In fact, a much weaker version of upwards closure seems sufficient for most results, for instance for the Coincidence Theorem (Theorem 4.2.7).

Definition 3.3.12. A modality $o \in \mathcal{O}$ is *leaf-upwards closed* if for any $t \in [\![o]\!]$ and $r \in T(\mathbf{1})$, if $t \leq r \leq t[\bot \mapsto *]$ (equivalently, r can be obtained from t by replacing a choice of $\bot$-leaves with $*$-leaves), then $r \in [\![o]\!]$.

Decomposability

We now focus on the much more technical notion of decomposability. The main purpose of the decomposability property is to enable us to prove that the logical preorder is preserved over the **to** constructor (cf. case 4 in the proof of Lemma 4.4.18). Semantically, the sequencing of programs by the **to** constructor is implemented using the monad multiplication map $\mu : TTX \to TX$ defined in Chapter 2. As such, we formulate decomposability as a preservation property of μ. It turns out to be sufficient to formulate decomposability as a property of μ only in the case that X is the singleton set $\{*\}$. As such, the formulation involves trees of unit type, that is trees in $T\mathbf{1}$, as well as *double*

trees, that is trees in $T T\mathbf{1}$. The decomposability condition is formulated using preorders $\preccurlyeq$ and $\preccurlyeq$ on such trees and double trees respectively.

The relation $\preccurlyeq$ is an extension of the positive behavioural preorder $\sqsubseteq^+$ at type $\mathbf{F}\,\mathbf{1}$, from a relation on closed computation terms to a relation $\preccurlyeq$ on arbitrary effect trees[4]. To accommodate this extension, we introduce a new notation; for $A \subseteq X$ and $o \in \mathcal{O}$ we write $o_X(A)$ for the subset $\{t \in TX \mid t[\in A] \in [\![o]\!]\} \subseteq TX$. We will often write $o(A)$ instead of $o_X(A)$ when X is clear from the context. For instance, if $t \in TX$ then $t \in o(A)$ means $t \in o_X(A)$. One case of particular interest is when $A = X = \{*\}$, for which we note that $o(\{*\}) = [\![o]\!]$.

As a preliminary to extending $\sqsubseteq^+$ to a relation on arbitrary effect trees, we extend the class of positive formulas of type $\mathbf{F}\,\mathbf{1}$, and interpret them over arbitrary $T\mathbf{1}$ computation trees rather than just terms. The class of *tree formulas* $TF(\mathbf{1})$ is the smallest class of formulas closed under arbitrary (not necessarily countable) disjunctions $\bigvee$ and conjunctions $\bigwedge$, such that for any $o \in \mathcal{O}$, $o(\top) \in TF(\mathbf{1})$ and $o(\bot) \in TF(\mathbf{1})$[5]. For each $\underline{\phi} \in TF(\mathbf{1})$, we define its denotation $[\![\underline{\phi}]\!] \subseteq T\mathbf{1}$ inductively, according to the following rules:

$$[\![o(\top)]\!] := o(\{*\}), \qquad [\![o(\bot)]\!] := o(\emptyset),$$
$$\left[\!\!\left[\bigvee X\right]\!\!\right] := \bigcup\{[\![\underline{\phi}]\!] \mid \underline{\phi} \in X\}, \qquad \left[\!\!\left[\bigwedge X\right]\!\!\right] := \bigcap\{[\![\underline{\phi}]\!] \mid \underline{\phi} \in X\}.$$

Note that tree formulas subsume positive formulas of type $\mathbf{F}\,\mathbf{1}$. Moreover, for any closed term $M : \mathbf{F}\,\mathbf{1}$ and formula $\underline{\phi} \in \mathit{Form}(\mathbf{F}\,\mathbf{1})$ it holds that $M \models \underline{\phi} \Leftrightarrow |M| \in [\![\underline{\phi}]\!]$.

We use tree formulas to define a behavioural preorder between arbitrary trees of type $\mathbf{1}$.

Definition 3.3.13. We define the preorder $\preccurlyeq$ on $T\mathbf{1}$ by: for any two trees $t, t' \in T\mathbf{1}$,

$$t \preccurlyeq t' \quad :\Longleftrightarrow \quad \forall \underline{\phi} \in TF(\mathbf{1}),\ t \in [\![\underline{\phi}]\!] \Rightarrow t' \in [\![\underline{\phi}]\!].$$

The simple result below characterises the preorder on trees determined by tree formulas.

Proposition 3.3.14. *For any $t, t' \in T\mathbf{1}$, the following three statements are equivalent:*

1. $t \preccurlyeq t'$.
2. $t \preccurlyeq t' \quad \wedge \quad t[\in \emptyset] \preccurlyeq t'[\in \emptyset]$.
3. $\forall o \in \mathcal{O},\ (t \in o(\{*\}) \Rightarrow t' \in o(\{*\})) \wedge (t \in o(\emptyset) \Rightarrow t' \in o(\emptyset))$.

Proof. The equivalence (1) $\Leftrightarrow$ (3) follows from a straightforward induction on the structure of tree formulas. The equivalence (1) $\Leftrightarrow$ (2) is obvious considering the equivalence (1) $\Leftrightarrow$ (3). □

[4] This extension is necessary because there are uncountably many effect trees, whereas there are only countably many terms of type $\mathbf{F}\,\mathbf{1}$.

[5] We could have alternatively taken the size of disjunctions and conjunctions to be the size of the set $T\mathbf{1}$, in which case $TF(\mathbf{1})$ forms a set.

It is now immediate that $\preccurlyeq$ is a conservative extension of the positive behavioural preorder on computation terms of type $\mathbf{F}\,\mathbf{1}$.

Proposition 3.3.15. *For computation terms $\underline{M}, \underline{N} \in \mathit{Terms}(\mathbf{F}\,\mathbf{1})$, it holds that $|\underline{M}| \preccurlyeq |\underline{N}|$ if and only if $\underline{M} \sqsubseteq^{+} \underline{N}$.*

Proof. The implication $|\underline{M}| \preccurlyeq |\underline{N}| \Rightarrow \underline{M} \sqsubseteq^{+} \underline{N}$ holds because of the inclusion of formulas $\mathit{Form}(\mathbf{F}\,\mathbf{1}) \subseteq \mathit{TF}(\mathbf{1})$. The implication $\underline{M} \sqsubseteq^{+} \underline{N} \Rightarrow |\underline{M}| \preccurlyeq |\underline{N}|$ follows from the equivalence (3) $\Leftrightarrow$ (1) in Proposition 3.3.14, and the fact that $o(\top), o(\bot) \in \mathit{Form}(\mathbf{F}\,\mathbf{1})$. □

Corollary 3.3.16. *For any $\underline{\phi} \in \mathit{TF}(\mathbf{1})$, there is a $\underline{\phi'} \in \mathit{Form}(\mathbf{F}\,\mathbf{1})$ such that:* $\forall \underline{M} \in \mathit{Terms}(\mathbf{F}\,\mathbf{1})$, $\underline{M} \models \underline{\phi'} \Leftrightarrow |\underline{M}| \in [\![\underline{\phi}]\!]$.

Proof. Using characteristic formulas from Lemma 3.3.6, we can prove using Proposition 3.3.15 that $\underline{\phi'} := \bigvee\{\chi_{\underline{M}} \mid \underline{M} \in \mathit{Terms}(\mathbf{F}\,\mathbf{1}), |\underline{M}| \in [\![\underline{\phi}]\!]\}$ has the right properties. □

In all of the examples of sets of modalities $\mathcal{O}$ given in Section 3.2, the preorder $\preccurlyeq$ based on $\mathcal{O}$ can be characterised in a simpler way, where:

$$t \preccurlyeq t' \quad \Longleftrightarrow \quad \forall o \in \mathcal{O},\ t \in [\![o]\!] \Rightarrow t' \in [\![o]\!]. \tag{3.2}$$

This is a consequence of the fact that each of the particular modalities o of the examples satisfy one of the following two properties:

(i) $o(\emptyset) = \emptyset$. The modalities with this property are: $\downarrow, \Diamond, \Box, \mathsf{P}_{>q}, (s \rightarrowtail s')$, $\langle w\rangle\!\downarrow$, $\mathsf{C}_{\geq q}$, and $\mathsf{C}_{\leq q}$.

(ii) $\forall t \in T\mathbf{1}, t \in o(\{*\}) \Leftrightarrow t \in o(\emptyset)$. The modalities with this property are E_e, $\langle w\rangle_{\ldots}$, and $\mathsf{C}^{\uparrow}_{>q}$.

There do however exist sets of Scott open modalities for which the characterisation of $\preccurlyeq$ via (3.2) does not hold. For example, for $\Sigma = \{\mathsf{raise} : \alpha^0 \to \alpha\}$ and $\mathcal{O} = \{o\}$ where $[\![o]\!] = \{\langle * \rangle, \mathsf{raise}\}$. There we have that $\mathsf{raise} \not\preccurlyeq \langle * \rangle$, since $\mathsf{raise} \in o(\emptyset)$ and $\langle * \rangle \notin o(\emptyset)$. However, it does hold that for all $o \in \mathcal{O}$, $\mathsf{raise} \in [\![o]\!] \Rightarrow \langle * \rangle \in [\![o]\!]$.

We next define the relation $\preccurlyeq$ on double trees, which is an abstraction of the positive behavioural preorder on type $\mathbf{F}\,(\mathbf{U}\,(\mathbf{F}\,\mathbf{1}))$.

Definition 3.3.17. We define the preorder $\preccurlyeq$ on $TT\mathbf{1}$ by: for any two double trees $r, r' \in TT\mathbf{1}$,

$$r \preccurlyeq r' \quad :\Longleftrightarrow \quad \forall o \in \mathcal{O}, \forall \underline{\phi} \in \mathit{TF}(\mathbf{1}),\ r \in o([\![\underline{\phi}]\!]) \Rightarrow r' \in o([\![\underline{\phi}]\!]).$$

Proposition 3.3.18. *For any two terms $\underline{M}, \underline{N} \in \mathit{Terms}(\mathbf{F}\,(\mathbf{U}\,(\mathbf{F}\,\mathbf{1})))$,*

$$\underline{M} \sqsubseteq^{+} \underline{N} \quad \Longleftrightarrow \quad |\underline{M}|[V \mapsto |\mathsf{force}(V)|] \preccurlyeq |\underline{M}|[V \mapsto |\mathsf{force}(V)|].$$

Proof. For the left to right implication, assume $\underline{M} \sqsubseteq^+ \underline{N}$ and $|\underline{M}|[V \mapsto |\mathsf{force}(V)|] \in o([\![\underline{\phi}]\!])$. Take $\underline{\phi}' \in \mathit{Form}(\mathbf{F}\,\mathbf{1})$ from Corollary 3.3.16, then $o(\langle \underline{\phi}' \rangle) \in \mathit{Form}(\mathbf{F}\,(\mathbf{U}\,(\mathbf{F}\,\mathbf{1})))$ and $\underline{M} \models o(\langle \underline{\phi}' \rangle)$. So $\underline{N} \models o(\langle \underline{\phi}' \rangle)$ and hence $|\underline{N}|[V \mapsto |\mathsf{force}(V)|] \in o([\![\underline{\phi}]\!])$. We conclude that $|\underline{M}|[V \mapsto |\mathsf{force}(V)|] \preccurlyeq |\underline{N}|[V \mapsto |\mathsf{force}(V)|]$.

For the right to left implication, assume $|\underline{M}|[V \mapsto |\mathsf{force}(V)|] \preccurlyeq |\underline{M}|[V \mapsto |\mathsf{force}(V)|]$. Using Lemma 3.3.5, we know that $\underline{M} \sqsubseteq^+ \underline{N}$ if for any $o \in \mathcal{O}$ and $\phi \in \mathit{Form}(\mathbf{U}\,(\mathbf{F}\,\mathbf{1}))$, $\underline{M} \models o(\phi) \Rightarrow \underline{N} \models o(\phi)$. Assume $\underline{M} \models o(\phi)$. Since the $\langle - \rangle$ formula constructor distributes over $\bigvee$ and $\bigwedge$, and formulas $\langle \top \rangle$ and $\langle \bot \rangle$ are equivalent to $\top$ and $\bot$ respectively, we can find (by induction on ϕ) a formula $\underline{\phi} \in \mathit{Form}(\mathbf{F}\,\mathbf{1}) \subseteq TF(\mathbf{1})$ such that $\phi \equiv \langle \underline{\phi} \rangle$. It follows that $\underline{M} \models o(\langle \underline{\phi} \rangle)$ which leads to $|\underline{M}|[V \mapsto |\mathsf{force}(V)|] \in o([\![\underline{\phi}]\!])$, hence $|\underline{N}|[V \mapsto |\mathsf{force}(V)|] \in o([\![\underline{\phi}]\!])$ and $\underline{N} \models o(\phi)$. We conclude that $\underline{M} \sqsubseteq^+ \underline{N}$. □

We give some alternative characterisations of $\preccurlyeq$. These use the notion of *right-set*, that is; for any relation $\mathcal{R} \subseteq X \times Y$ and any subset $A \subseteq X$, we write $(\mathcal{R}^{\uparrow}[A]) := \{y \in Y \mid \exists x \in A, x\,\mathcal{R}\,y\}$ for the *right-set* of A under the relation $\mathcal{R}$.

Lemma 3.3.19. *If $\mathcal{O}$ is a set of leaf-upwards closed modalities, then for all $r, r' \in TT\mathbf{1}$, the following are equivalent:*

1. $r \preccurlyeq r'$.
2. $\forall A \subseteq T\mathbf{1},\ r[\in A] \preccurlyeq r'[\in(\preccurlyeq^{\uparrow}[A])]$.
3. $\forall o \in \mathcal{O}, \forall A \subseteq T\mathbf{1},\ r \in o(A) \Rightarrow r' \in o(\preccurlyeq^{\uparrow}[A])$.

Proof. (1) ⇒ (2). Assume $r[\in A] \in o(\{*\})$, then since $A \subseteq (\preccurlyeq^{\uparrow}[A])$ and o is leaf-upwards closed, $r[\in(\preccurlyeq^{\uparrow}[A])] \in o(\{*\})$, which means $r \in o(\preccurlyeq^{\uparrow}[A])$. Let $\underline{\phi}_A := \bigvee_{t \in A}((\bigwedge_{o \in \mathcal{O}, t \in o(\{*\})} o(\top)) \wedge (\bigwedge_{o \in \mathcal{O}, t \in o(\emptyset)} o(\bot))) \in TF(\mathbf{1})$, then $[\![\underline{\phi}_A]\!] = (\preccurlyeq^{\uparrow}[A])$ as $\underline{\phi}_A$ perfectly replicates the condition of membership of $(\preccurlyeq^{\uparrow}[A])$. So by (1) it holds that $r' \in o(\preccurlyeq^{\uparrow}[A])$, hence $r'[\in(\preccurlyeq^{\uparrow}[A])] \in o(\{*\})$. If $r[\in A] \in o(\emptyset)$, then $r \in o(\emptyset)$ hence $r' \in o(\emptyset)$ (since $(\bigvee \emptyset) \in TF(\mathbf{1})$ and $[\![\bigvee \emptyset]\!] = \emptyset$), so $r'[\in(\preccurlyeq^{\uparrow}[A])] \in o(\emptyset)$. With Proposition 3.3.14, we conclude that $r[\in A] \preccurlyeq r'[\in(\preccurlyeq^{\uparrow}[A])]$.

(2) ⇒ (3). Since $r[\in A] \preccurlyeq r'[\in(\preccurlyeq^{\uparrow}[A])]$, then for any $o \in \mathcal{O}$ it holds that $r[\in A] \in o(\{*\}) \Rightarrow r'[\in(\preccurlyeq^{\uparrow}[A])] \in o(\{*\})$, which is identical to the statement $r \in o(A) \Rightarrow r' \in o(\preccurlyeq^{\uparrow}[A])$.

(3) ⇒ (1). If $r \in o([\![\underline{\phi}]\!])$ with $\underline{\phi} \in TF(\mathbf{1})$, then by (3) it holds that $r' \in o(\preccurlyeq^{\uparrow}[[\![\underline{\phi}]\!]])$. By the definition of $\preccurlyeq$ it holds that $(\preccurlyeq^{\uparrow}[[\![\underline{\phi}]\!]]) \subseteq [\![\underline{\phi}]\!]$, and since $\preccurlyeq$ is reflexive $[\![\underline{\phi}]\!] \subseteq (\preccurlyeq^{\uparrow}[[\![\underline{\phi}]\!]])$. Hence $(\preccurlyeq^{\uparrow}[[\![\underline{\phi}]\!]]) = [\![\underline{\phi}]\!]$ and we conclude that $r' \in o([\![\underline{\phi}]\!])$. □

We can now finally define the promised notion of decomposability.

Definition 3.3.20 (Decomposability)**.** We say that $\mathcal{O}$ is *decomposable* if, for all double trees $r, r' \in TT\mathbf{1}$, $r \preccurlyeq r'$ implies $\mu r \preccurlyeq \mu r'$.

As an immediate consequence of Propositions 3.3.15 and 3.3.18, we can see that decomposability implies the preservation of $\sqsubseteq^+$ over the **to** constructor for sequencing over type $\mathbf{F}\,(\mathbf{U}\,(\mathbf{F}\,\mathbf{1}))$:

Lemma 3.3.21. *If $\mathcal{O}$ is decomposable, then for any two closed terms $\underline{M}, \underline{N} \in Terms(\mathbf{F}(\mathbf{U}(\mathbf{F}\,\mathbf{1})))$,*

$$\underline{M} \sqsubseteq^{+} \underline{N} \quad \Longrightarrow \quad \underline{M} \text{ to } x.\,\mathsf{force}(x) \;\sqsubseteq^{+}\; \underline{N} \text{ to } x.\,\mathsf{force}(x).$$

Proof. Use that $|\underline{M} \text{ to } x.\,\mathsf{force}(x)| = \mu(|\underline{M}|[V \mapsto |\mathsf{force}(V)|])$ from Corollary 2.2.10. □

The two lemmas below provide different characterisations of decomposability, which hold if all modalities are upwards closed. The first provides a reformulation that is immediate in the case of our examples, where the statement $t \preccurlyeq t'$ can be simplified via (3.2). The general case, however, requires a rather technical proof.

Lemma 3.3.22. *A set of leaf-upwards closed modalities $\mathcal{O}$ is decomposable if and only if for all $r, r' \in TT\mathbf{1}$, if $r \preccurlyeq r'$, then $\forall o \in \mathcal{O},\ \mu r \in o(\{*\}) \Rightarrow \mu r' \in o(\{*\})$.*

Proof. ($\Rightarrow$) The result follows by observing that $\mu r \preccurlyeq \mu r'$ implies $\forall o \in \mathcal{O},\ \mu r \in o(\{*\}) \Rightarrow \mu r' \in o(\{*\})$.

($\Leftarrow$) Assume:

$$\text{(I)}\ \ \forall r, r' \in TT\mathbf{1},\ r \preccurlyeq r' \quad \Longrightarrow \quad \forall o \in \mathcal{O},\ \mu r \in o(\{*\}) \Rightarrow \mu r' \in o(\{*\}).$$

Take some $r, r' \in TT\mathbf{1}$, and suppose that $r \preccurlyeq r'$, which with Lemma 3.3.19 means:

$$\text{(II)}\ \ \forall o \in \mathcal{O}, A \subseteq T\mathbf{1},\ r \in o(A) \Rightarrow r' \in o(\preccurlyeq^{\uparrow}[A]).$$

We need to prove that $\mu r \preccurlyeq \mu r'$. By (I) we derive that $\forall o \in \mathcal{O},\ \mu r \in o(\{*\}) \Rightarrow \mu r' \in o(\{*\})$. To prove $\mu r \preccurlyeq \mu r'$ we need only prove $\forall o \in \mathcal{O},\ \mu r \in o(\emptyset) \Rightarrow \mu r' \in o(\emptyset)$.

Assume $\mu r \in o'(\emptyset)$ for $o' \in \mathcal{O}$, then $\mu(r[t \mapsto t[\in\emptyset]]) = (\mu r)[\in\emptyset] \in o'(\{*\})$. We prove that $r[t \mapsto t[\in\emptyset]] \preccurlyeq r'[t' \mapsto t'[\in\emptyset]]$ using equivalent notion (3) from Lemma 3.3.19. Suppose for some $o \in \mathcal{O}$ and $A \subseteq T\mathbf{1}$, it holds that $r[t \mapsto t[\in\emptyset]] \in o(A)$. Let $B := \{t \in T\mathbf{1} \mid t[\in\emptyset] \in A\}$, then $r \in o(B)$. By (II) it holds that $r' \in o(\preccurlyeq^{\uparrow}[B])$.

For $t' \in (\preccurlyeq^{\uparrow}[B])$, there is a $t \in B$ such that $t \preccurlyeq t'$. Since $t \in B$, $t[\in\emptyset] \in A$ and hence $t'[\in\emptyset] \in (\preccurlyeq^{\uparrow}[A])$. So $(\preccurlyeq^{\uparrow}[B]) \subseteq \{t' \in T\mathbf{1} \mid t'[\in\emptyset] \in (\preccurlyeq^{\uparrow}[A])\}$, and we can use leaf-upwards closure of o' to derive $r'[t' \mapsto t'[\in\emptyset]] \in o(\preccurlyeq^{\uparrow}[A])$. Hence by Lemma 3.3.19, $r[t \mapsto t[\in\emptyset]] \preccurlyeq r'[t' \mapsto t'[\in\emptyset]]$.

We can apply (I) to derive $\mu(r[t \mapsto t[\in\emptyset]]) \preccurlyeq \mu(r'[t' \mapsto t'[\in\emptyset]])$. So since $\mu(r[t \mapsto t[\in\emptyset]]) = (\mu r)[\in\emptyset] \in o'(\{*\})$, it holds that and $\mu(r'[t' \mapsto t'[\in\emptyset]]) \in o'(\{*\})$ and hence $\mu r \in o'(\emptyset)$. We conclude that $\mu r \preccurlyeq \mu r'$, so we are finished. □

The second formulation of decomposability shows that it is equivalent to being able to 'decompose' statements of the form $\mu r \in o(\{*\})$ into a collection of modal properties of the double tree r.

Proposition 3.3.23. *A set of leaf-upwards closed modalities $\mathcal{O}$ is decomposable if and only if for any $r \in TT\mathbf{1}$ and $o \in \mathcal{O}$ such that $\mu r \in o(\{*\})$, there is a collection of pairs $\{(o_i, \underline{\phi}_i)\}_{i \in I}$, with each $o_i \in \mathcal{O}$ and $\underline{\phi}_i \in TF(\mathbf{1})$, satisfying the following two properties:*

1. $\forall i \in I,\ r \in o_i([\![\underline{\phi}_i]\!])$.

2. $\forall r' \in TT\mathbf{1},\ (\forall i \in I,\ r' \in o_i([\![\underline{\phi}_i]\!])) \ \Rightarrow\ \mu r' \in o(\{*\})$.

Proof. We use the equivalent statement for decomposability established in Lemma 3.3.22.

($\Rightarrow$) Assume decomposability, and that for some $r \in TT\mathbf{1}$ and $o \in \mathcal{O}$, it holds that $\mu r \in o(\{*\})$. Take as set of pairs the following set: $\{(o', \underline{\phi}') \mid o' \in \mathcal{O}, \underline{\phi}' \in TF(\mathbf{1}), r \in o'([\![\underline{\phi}']\!])\}$. This collection by definition satisfies condition (1). For any $r' \in TT\mathbf{1}$, if $r' \in o'([\![\underline{\phi}']\!])$ for any pair in the collection, then $r \preccurlyeq r'$. Hence by decomposability it holds that $\forall o'' \in \mathcal{O}, \mu r \in o''(\{*\}) \Rightarrow \mu r' \in o''(\{*\})$, so in particular $r' \in o(\{*\})$, and thus condition (2) holds.

($\Leftarrow$) Assume the statement given in the lemma, we need to prove decomposability. For some $r, r' \in TT\mathbf{1}$, suppose that $r \preccurlyeq r'$. Now let $o \in \mathcal{O}$ such that $\mu r \in o(\{*\})$, then there is a collection of pairs $\{(o_i, \underline{\phi}_i)\}_{i \in I}$ satisfying the properties given above. So by property (1), $\forall i \in I,\ r \in o_i([\![\underline{\phi}_i]\!])$, and since $r \preccurlyeq r'$ it holds that $r' \in o_i([\![\underline{\phi}_i]\!])$. By property (2) we conclude that $\mu r' \in o(\{*\})$ which is what we needed to prove. □

The following result is a direct consequence of the above proposition.

Corollary 3.3.24. *If $\mathcal{O}$ and $\mathcal{O}'$ are both decomposable sets of Scott open modalities for the same signature Σ, then $\mathcal{O} \cup \mathcal{O}'$ is a decomposable set of Scott open modalities.*

The following stronger notion of decomposability simplifies the property given in Proposition 3.3.23. Our running examples turn out to all satisfy this stronger property, as we shall verify below.

Definition 3.3.25 (Strong decomposability)**.** We say that $\mathcal{O}$ is *strongly decomposable* if, for every $r \in TT\mathbf{1}$ and $o \in \mathcal{O}$ for which $\mu r \in o(\{*\})$, there exists a collection $\{(o_i, o_i')\}_{i \in I}$ of pairs of modalities such that:

1. $\forall i \in I,\ r \in o_i(o_i'(\{*\}))$; and
2. For every $r' \in TT\mathbf{1}$, if for all $i \in I$, $r' \in o_i(o_i'(\{*\}))$ then $\mu r' \in o(\{*\})$.

Proposition 3.3.26. *If $\mathcal{O}$ is a strongly decomposable set of leaf-upwards closed modalities, then $\mathcal{O}$ is decomposable.*

Proof. Using Proposition 3.3.23, this result is a simple consequence of the fact that for any $o_i \in \mathcal{O}$, $o_i(\top) \in TF(\mathbf{1})$ and $[\![o_i(\top)]\!] = o_i(\{*\})$. □

The converse statement is not always true, even if we assume Scott openness.

Lemma 3.3.27. *There is a decomposable set of Scott open modalities (moreover satisfying property 3.2) which is not strongly decomposable.*

Proof. Take the following decomposable set of Scott open modalities; $\mathcal{O} := \{\downarrow, \Box'\}$ for the signature $\Sigma := \{\mathsf{or}(-,-) : \alpha \times \alpha \to \alpha\}$. Here, $[\![\downarrow]\!] := \{\langle * \rangle\}$ (termination without any nondeterministic choice) and $[\![\Box']\!] := [\![\Box]\!] - \{\langle * \rangle\}$ (MUST termination with at least one

nondeterministic choice). These two modalities are obviously Scott open. For $r \in TT\mathbf{1}$, it holds that:

$\mu r \in [\![\downarrow]\!] \iff r \in \downarrow(\downarrow(\{*\}))$.

$\mu r \in [\![\Box']\!] \iff r \in \downarrow(\Box'(\{*\})) \ \vee \ r \in \Box'(\Box'\{*\} \vee \downarrow(\{*\}))$.

Since $\downarrow(\top)$ and $\Box'(\top) \vee \downarrow(\top)$ are both tree formulas, we can use Proposition 3.3.23 to derive that $\mathcal{O}$ is decomposable.

However, consider the following tree; $r := \mathsf{or}(\langle\langle * \rangle\rangle, \langle \mathsf{or}(\langle * \rangle, \langle * \rangle)\rangle) \in TT\mathbf{1}$. This tree has the property that $\mu r = \mathsf{or}(\langle * \rangle, \mathsf{or}(\langle * \rangle, \langle * \rangle)) \in [\![\Box']\!]$. However, $\langle * \rangle$ and $\mathsf{or}(\langle * \rangle, \langle * \rangle)$ do not have a modality they satisfy in common. As such, there is no pair $o_i, o_i' \in \mathcal{O}$ such that $r \in o_i(o_i'(\{*\}))$, so any collection of pairs satisfying property (1) of strong decomposability must be empty. But then, for property (2) to be satisfied, it must hold that $\forall r' \in TT\mathbf{1}, \mu r' \in [\![\Box']\!]$, which is obviously not true. So we cannot find a collection of pairs of modalities satisfying both properties, and we conclude that $\mathcal{O}$ is not strongly decomposable. □

3.3.3 The examples have the correct properties

In this subsection, we prove that the sets of modalities for the examples of Section 3.2 satisfy the Scott openness and decomposability property. As such, the Compatibility Theorem, Theorem 3.3.8, holds for those sets of modalities.

In the examples, it is straightforward to observe that the modalities are upwards closed, so the proofs of this are left out. Decomposability will be proven by proving the strong decomposability property, which will take the following form. Given $t \in T(T(\mathbf{1}))$, the observation $\mu t \in [\![o]\!]$ will be expressed as an equivalent expression using observations of the form $t \in o'(o''(\{*\}))$. As such, if $\forall o', o'', t \in o'(o''(\{*\})) \implies t' \in o'(o''(\{*\}))$, then $\mu t \in [\![o]\!] \implies \mu t' \in [\![o']\!]$, and strong decomposability is proven.

Pure computation: (Subsection 3.2.1)
Here $\Sigma_\emptyset = \emptyset$, and the modalities are $\mathcal{O}_\emptyset = \{\downarrow\}$. All trees $T_{\Sigma_\emptyset}(\mathbf{1})$ are finite, so Scott openness holds. $\mathcal{O}_\emptyset$ is strongly decomposable, since for any $t \in T_{\Sigma_\emptyset}(T_{\Sigma_\emptyset}(\mathbf{1}))$,

$$\mu t \in [\![\downarrow]\!] \quad \iff \quad t \in \downarrow(\downarrow(\{*\})).$$

This means t terminates, and returns x which terminates with $*$.

Error: (Subsection 3.2.2)
Here $\Sigma_{\mathrm{er}} := \{\mathsf{raise}_e() \mid e \in \mathsf{Err}\}$, and the modalities are $\mathcal{O}_{\mathrm{er}} = \{\downarrow\} \cup \{\mathsf{E}_e \mid e \in \mathsf{Err}\}$. Here too Scott openness holds because all trees of $T_{\Sigma_{\mathrm{er}}}(\mathbf{1})$ are finite. $\mathcal{O}_{\mathrm{er}}$ is strongly decomposable, as we can observe that for any $t \in T_{\Sigma_{\mathrm{er}}}(T_{\Sigma_{\mathrm{er}}}(\mathbf{1}))$ it holds that:

$$\mu t \in [\![\downarrow]\!] \quad \iff \quad t \in \downarrow(\downarrow(\{*\})).$$

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

$$\mu t \in [\![\mathsf{E}_e]\!] \quad \iff \quad t \in \mathsf{E}_e(\mathsf{E}_e(\{*\})) \ \vee \ t \in \downarrow(\mathsf{E}_e(\{*\})).$$

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

Nondeterminism: (Subsection 3.2.3)
Here $\Sigma_{\mathrm{nd}} := \{\mathsf{or}(-,-) : \alpha \times \alpha \to \alpha\}$ and $\mathcal{O}_{\mathrm{nd}} = \{\Diamond, \Box\}$. Any element of $[\![\Box]\!]$ is finite, so it is Scott open. If $t \in [\![\Diamond]\!]$, then t has a $\langle * \rangle$-leaf, and there is a finite subtree $t' \leq t$ containing that leaf. Hence $t' \in [\![\Diamond]\!]$, since it has a $\langle * \rangle$-leaf, and we can conclude that $\Diamond$ is Scott open. $\mathcal{O}_{\mathrm{nd}}$ is strongly decomposable, since for any $t \in T_{\Sigma_{\mathrm{nd}}}(T_{\Sigma_{\mathrm{nd}}}(\mathbf{1}))$ it holds that:

$$\mu t \in [\![\Diamond]\!] \quad \Longleftrightarrow \quad t \in \Diamond(\Diamond(\{*\})).$$

This means t may return a tree, which may return $*$.

$$\mu t \in [\![\Box]\!] \quad \Longleftrightarrow \quad t \in \Box(\Box(\{*\})).$$

This means t is finite and must return a finite tree, which must return $*$.

Probability: (Subsection 3.2.4)
Here $\Sigma_{\mathrm{pr}} = \{\mathsf{pr}(-,-) : \alpha \times \alpha \to \alpha\}$ and modalities are $\mathcal{O}_{\mathrm{pr}} = \{\mathsf{P}_{>q} \mid q \in \mathbb{Q}, 0 \leq q < 1\}$. If $t \in [\![\mathsf{P}_{>q}]\!]$, then $\bigsqcup_n \mathsf{P}_n(t) = \mathsf{P}(t) > q$. There must be an n such that $\mathsf{P}_n(t) > q$, since otherwise $\mathsf{P}(t) \leq q$. Now $\mathsf{P}_n(t)$ only looks at a finite part of t, the part of t bounded by tree-depth n. So defining $t^n \leq t$ to be the finite subtree of t bounded at layer n (replacing nodes at that layer with $\perp$ leaves), it hold that $\mathsf{P}_n(t^n) = \mathsf{P}_n(t) > q$. Hence $t^n \in [\![\mathsf{P}_{>q}]\!]$, showing that the modality is Scott open.

To establish that $\mathcal{O}_{\mathrm{pr}}$ is strongly decomposable, an intricate argument must be given. Given $t \in T_{\Sigma_{\mathrm{pr}}}(T_{\Sigma_{\mathrm{pr}}}(\mathbf{1}))$, we define a real-valued function $f_t : [0,1] \to [0,1]$ where for each rational number $q \in [0,1]$ we define $f_t(p) := \mathsf{P}(t[\in [\![\mathsf{P}_{>p}]\!]]) = \sup\{r \in [0,1] \text{ rational } \mid t \in \mathsf{P}_{>r}(\mathsf{P}_{>p}(\{*\}))\}$. The set $[\![\mathsf{P}_{>p}]\!]$ gets smaller as p gets bigger, and as such, the function f_t on the rationals is monotone decreasing. We can extend the definition of f_t to a function on the reals.

The proof of decomposability relies on the fact that $\mathsf{P}(\mu t) = \int_0^1 f_t(p)dp$, which can be observed by studying how the probability of μt is calculated: its the sum of $2^{-n}\mathsf{P}(x)$ over all leaves x of t where n is the depth of x in t. The integral allows us to express this sum by using only $t \in \mathsf{P}_{>r}(\mathsf{P}_{>p}(\{*\}))$ statements. We can conclude that:

$$\mu t \in [\![\mathsf{P}_{>q}]\!] \quad \Longleftrightarrow \quad \int_0^1 f_t(p)dp > q \quad \Longleftrightarrow \quad \exists n, \left(\sum_{i=1,2,\ldots,n} f_t(i/n)/n \right) > q \ .$$

Since f_t is defined using pairs $\mathsf{P}_{>p}, \mathsf{P}_{>r}$, this gives us a strong decomposition for $\mu t \in [\![\mathsf{P}_{>q}]\!]$, so we can conclude that $\mathcal{O}_{\mathrm{pr}}$ is strongly decomposable.

Global Store: (Subsection 3.2.5)
$\Sigma_{\mathrm{gs}} := \{\mathsf{lookup}_l(-) : \alpha^{\mathbb{N}} \to \alpha, \mathsf{update}_l(-;-) : \mathbb{N} \times \alpha \to \alpha \mid l \in \mathsf{Loc}\}$ and $\mathcal{O}_{\mathrm{gs}} = \{(s \rightarrowtail r) \mid s, r \in \mathsf{State} = \mathbb{N}^{\mathsf{Loc}}\}$. Each $[\![(s \rightarrowtail r)]\!]$ is Scott open, since for each tree $t \in [\![(s \rightarrowtail r)]\!]$, there is a single finite branch which determines its satisfaction. $\mathcal{O}_{\mathrm{gs}}$ is strongly decomposable, since for any $t \in T_{\Sigma_{\mathrm{gs}}}(T_{\Sigma_{\mathrm{gs}}}(\mathbf{1}))$:

$$\mu t \in [\![(s \rightarrowtail r)]\!] \quad \Longleftrightarrow \quad \exists c \in \mathbb{N}^{\mathsf{Loc}}, t \in (s \rightarrowtail c)((c \rightarrowtail r)(\{*\})).$$

This means that for some c, $exec(t,s) = (x,c)$ and $exec(x,c) = (*,r)$.

Input/Output: (Subsection 3.2.6)
$\Sigma_{\mathrm{io}} = \{\mathsf{read}(-) : \alpha^{\mathbb{N}} \to \alpha, \mathsf{write}(-;-) : \mathbb{N} \times \alpha \to \alpha\}$ and
$\mathcal{O}_{\mathrm{io}} = \{\langle w\rangle\!\downarrow, \langle w\rangle_{\dots} \mid w \text{ an i/o-trace}\}$. The modalities are Scott open, since only a single finite branch, exactly following the specified i/o trace, is checked by the modalities. So any finite subtree containing that branch is contained in the modality. Moreover, for each modality o, there is a finite tree t such that $[\![o]\!] = \{r \mid t \leq r\}$. $\mathcal{O}_{\mathrm{io}}$ is strongly decomposable, since for any $t \in T_{\Sigma_{\mathrm{io}}}(T_{\Sigma_{\mathrm{io}}}(\mathbf{1}))$:

$$\mu t \in [\![\langle w\rangle\!\downarrow]\!] \quad \Longleftrightarrow \quad \exists v, v', \text{ s.t. } w = vv' \wedge t \in \langle v\rangle\!\downarrow (\langle v'\rangle\!\downarrow (\{*\})).$$

This means t follows trace v returning x, and x follows trace v' returning $*$.

$$\mu t \in [\![\langle w\rangle_{\dots}]\!] \quad \Longleftrightarrow \quad t \in \langle w\rangle_{\dots}(\langle w\rangle_{\dots}(\{*\})) \ \vee \ (\exists v, v', \ w = vv' \wedge t \in \langle v\rangle\!\downarrow (\langle v'\rangle_{\dots}(\{*\}))).$$

This means either t follows trace w immediately, or it follows v returning a tree which follows v'.

Timer: (Subsection 3.2.7)
Here $\Sigma_{\mathrm{ti}} = \{\mathsf{tick}_c(-) : \alpha \to \alpha \mid c \in \mathsf{Inc}\}$ and $\mathcal{O}^{\downarrow}_{\mathrm{ti}} = \{\mathsf{C}_{\leq q} \mid q \in \mathbb{Q}_{\geq 0}\}$, $\mathcal{O}^{\uparrow}_{\mathrm{ti}} = \{\mathsf{C}_{\geq q}, \mathsf{C}^{\uparrow}_{>q} \mid q \in \mathbb{Q}_{\geq 0}\}$ and $\mathcal{O}_{\mathrm{ti}} = \mathcal{O}^{\downarrow}_{\mathrm{ti}} \cup \mathcal{O}^{\uparrow}_{\mathrm{ti}}$. Note that for each q, $[\![\mathsf{C}_{\leq q}]\!]$, $[\![\mathsf{C}_{\geq q}]\!]$ and $[\![\mathsf{C}^{\uparrow}_{>q}]\!]$ are Scott open, since for any $n \in \mathbb{N}$ and any tree $t \in T(\mathbf{1})$, there is a finite subtree $t^n \leq t$ such that $\mathit{clock}_n(t) = \mathit{clock}_n(t^n)$. All three sets of modalities are strongly decomposable, since for any $t \in T_{\Sigma_{\mathrm{ti}}}(T_{\Sigma_{\mathrm{ti}}}(\mathbf{1}))$ it holds that:

$$\begin{aligned}
\mu t \in [\![\mathsf{C}_{\leq q}]\!] &\iff \exists a, b \in \mathbb{Q}, (a+b) \leq q \ \wedge \ t \in \mathsf{C}_{\leq a}(\mathsf{C}_{\leq b}(\{*\})).\\
\mu t \in [\![\mathsf{C}_{\geq q}]\!] &\iff \exists a, b \in \mathbb{Q}, (a+b) \geq q \ \wedge \ t \in \mathsf{C}_{\geq a}(\mathsf{C}_{\geq b}(\{*\})).\\
\mu t \in [\![\mathsf{C}^{\uparrow}_{>q}]\!] &\iff t \in \mathsf{C}^{\uparrow}_{>q}(\mathsf{C}^{\uparrow}_{>q}(\{*\})) \ \vee \ \exists a, b \in \mathbb{Q}, ((a+b) \geq q \wedge t \in \mathsf{C}_{\geq a}(\mathsf{C}^{\uparrow}_{>b}(\{*\})).
\end{aligned}$$

Remark: The set of modalities, only containing modalities of the form $\mathsf{C}^{\uparrow}_{>q}$ is not decomposable, which is why we do not consider it as a valid interpretation of the effect.

3.4 Properties of the preorders

In the next two sections we will investigate some properties of the behavioural preorders $\sqsubseteq^{+}$ and $\equiv$. In Section 3.5, we look at the equations and inequations for effect trees of computations of type $\mathbf{FA}$ in particular. In this section, we will look at other properties and classifications of the behavioural preorders. Moreover, we will investigate the different behavioural preorders for the effect of nondeterminism as a case study, e.g., to see how the general and positive behavioural equivalence can differ.

Firstly we observe that equality of operational semantics implies the general behavioural preorder.

Lemma 3.4.1. *For any $\underline{M}, \underline{N} : \underline{\mathbf{C}}$, if $|\underline{M}| = |\underline{N}|$ then $\underline{M} \equiv \underline{N}$.*

Proof. We prove by induction on computation formulas $\underline{\phi} \in Form(\underline{\mathbf{C}})$, that for any closed terms $\underline{M}, \underline{N} : \underline{\mathbf{C}}$, if $|\underline{M}| = |\underline{N}|$ and $\underline{M} \models \underline{\phi}$, then $\underline{N} \models \underline{\phi}$.

In the case that $\underline{\phi}$ is not a basic formula, the result can be easily proven. E.g., if $\underline{\phi} = \bigwedge X$ where for any $\underline{\psi} \in X$, $\underline{M} \models \underline{\psi} \Rightarrow \underline{N} \models \underline{\psi}$, then $\underline{M} \models \bigwedge X \Rightarrow \underline{N} \models \bigwedge X$. If $\underline{\phi} = \neg(\underline{\psi})$, then since $|\underline{N}| = |\underline{M}|$ it holds that $\underline{N} \models \underline{\psi} \Rightarrow \underline{M} \models \underline{\psi}\ \underline{M}$, hence $\underline{M} \models \underline{\phi} \Rightarrow \underline{N} \models \underline{\phi}$.

If $\underline{\phi} := o(\phi)$, then $\underline{M} \models \underline{\phi}$ means $|\underline{M}|[\models \phi] \in [\![o]\!]$. So since $|\underline{N}| = |\underline{M}|$ we can derive that $|\underline{N}|[\models \phi] \in [\![o]\!]$, hence $\underline{N} \models o(\phi)$.

If $\underline{\phi} := (V \mapsto \underline{\psi})$, assume as induction hypothesis that the statement holds for $\underline{\psi}$. Suppose $\underline{M} \models (V \mapsto \underline{\psi})$, then $\underline{M}\ V \models \underline{\psi}$. Now, $|\underline{M}\ V| = \mu(|\underline{M}|[\lambda x.\, \underline{M'} \mapsto |\underline{M'}[V/x]])$ by Corollary 2.2.10, which is equal to $\mu(|\underline{N}|[\lambda x.\, \underline{M'} \mapsto |\underline{M'}[V/x]]) = |\underline{N}\ V|$. So by induction hypothesis on $\underline{\psi}$ and $|\underline{M}\ V| = |\underline{N}\ V|$ we can conclude that $\underline{N}\ V \models \underline{\psi}$ and hence $\underline{N} \models (V \mapsto \underline{\psi})$.

The case where $\underline{\phi} := (i \mapsto \underline{\psi})$ can be proven in the same way as the previous case using Corollary 2.2.10.

This finishes the induction and we can conclude that $\underline{M} \equiv \underline{N}$.

□

Corollary 3.4.2. $\underline{M} \equiv \mathsf{force}(\mathsf{thunk}(\underline{M}))$, *and hence* $(\mathsf{thunk}(\underline{M}) \models \langle\underline{\phi}\rangle) \Leftrightarrow (\underline{M} \models \underline{\phi})$.

Using Lemma 3.3.5, the following classification of the behavioural preorders can be established by unfolding the definitions of the basic formulas:

Lemma 3.4.3. *For $\mathcal{R}$ either $\underline{\sqsubseteq}^+$ or $\equiv$, it holds that:*

1. $V\, \mathcal{R}_{\mathbf{N}}\, W \iff V = W$.
2. $V\, \mathcal{R}_{\mathbf{U}\,\underline{\mathbf{C}}}\, W \iff \mathsf{force}(V)\, \mathcal{R}_{\underline{\mathbf{C}}}\, \mathsf{force}(W)$.
3. $(j, V)\, \mathcal{R}_{\mathbf{\Sigma}_{i\in I}\, \mathbf{A}_i}\, (k, W) \iff (j = k) \wedge V\, \mathcal{R}_{\mathbf{A}_j}\, W$.
4. $(V, V')\, \mathcal{R}_{\mathbf{A}\times\mathbf{B}}\, (W, W') \iff V\, \mathcal{R}_{\mathbf{A}}\, W \wedge V'\, \mathcal{R}_{\mathbf{B}}\, W'$.
5. $\underline{M}\, \mathcal{R}_{\mathbf{A}\to\underline{\mathbf{C}}}\, \underline{N} \iff \forall V \in Terms(\mathbf{A}), (\underline{M}\ V)\, \mathcal{R}_{\underline{\mathbf{C}}}\, (\underline{N}\ V)$.
6. $\underline{M}\, \mathcal{R}_{\mathbf{\Pi}_{i\in I}\, \underline{\mathbf{C}}_i}\, \underline{N} \iff \forall j \in I, (\underline{M}\ j)\, \mathcal{R}_{\underline{\mathbf{C}}_j}\, (\underline{N}\ j)$.

Proof. We use Lemma 3.3.5.

1. If $\overline{n}\, \mathcal{R}_{\mathbf{N}}\, \overline{m}$, then since $\overline{n} \models \{n\}$ it holds that $\overline{m} \models \{n\}$ hence $\overline{n} = \overline{m}$.

   If $V = W$, then $V\, \mathcal{R}_{\mathbf{N}}\, W$ by reflexivity (Lemma 3.3.3).

2. $V\, \mathcal{R}_{\mathbf{U}\,\underline{\mathbf{C}}}\, W \iff \forall \underline{\phi} \in Form(\underline{\mathbf{C}}), (V \models \langle\underline{\phi}\rangle \Rightarrow W \models \langle\underline{\phi}\rangle) \iff$
   $\forall \underline{\phi} \in Form(\underline{\mathbf{C}}), (\mathsf{force}(V) \models \underline{\phi} \Rightarrow \mathsf{force}(W) \models \underline{\phi}) \iff$
   $\mathsf{force}(V)\, \mathcal{R}_{\underline{\mathbf{C}}}\, \mathsf{force}(W)$.

3. If $(j,V)\,\mathcal{R}_{\mathbf{\Sigma}_{i\in I}\,\mathbf{A}_i}\,(k,W)$, then from $(j,V) \models (j,\top)$ we derive that $(k,W) \models (j,\top)$, so $k = j$. For any $\phi \in \mathit{Form}(\mathbf{A}_j)$ we now have $V \models \phi \iff (j,V) \models (j,\phi) \implies (j,W) \models (j,\phi) \iff W \models \phi$, so we conclude that $V\,\mathcal{R}_{V_j}\,W$.

   If $V\,\mathcal{R}_{V_j}\,W$ and $j = k$, then $(j,V) \models (l,\phi)$ means $j = l$ and $V \models \phi$, hence $W \models \phi$ so $(k,W) \models (l,\phi)$. We conclude that $(j,V)\,\mathcal{R}_{\mathbf{\Sigma}_{i\in I}\,\mathbf{A}_i}\,(k,W)$.

4. $(V,V')\,\mathcal{R}_{\mathbf{A}\times\mathbf{B}}\,(W,W') \iff$
   $(\forall \phi \in \mathit{Form}(\mathbf{A}), (V,V') \models \pi_0(\phi) \Rightarrow (W,W') \models \pi_0(\phi))$
   $\wedge(\forall \phi \in \mathit{Form}(\mathbf{B}), (V,V') \models \pi_1(\phi) \Rightarrow (W,W') \models \pi_1(\phi))$,
   which is true if and only if $V\,\mathcal{R}_{\mathbf{A}}\,W$ and $V'\,\mathcal{R}_{\mathbf{B}}\,W'$.

5. $\underline{M}\,\mathcal{R}_{\mathbf{A}\to\underline{\mathbf{C}}}\,\underline{N} \iff$
   $\forall V \in \mathit{Terms}(\mathbf{A}), \forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}), (\underline{M} \models (V \mapsto \underline{\phi}) \Rightarrow \underline{N} \models (V \mapsto \underline{\phi})) \iff$
   $\forall V \in \mathit{Terms}(\mathbf{A}), \forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}), (\underline{M}\ V \models \underline{\phi} \Rightarrow \underline{N}\ V \models \underline{\phi}) \iff$
   $\forall V \in \mathit{Terms}(\mathbf{A}), \underline{M}\ V\,\mathcal{R}_{\underline{\mathbf{C}}}\,\underline{N}\ V$.

6. $\underline{M}\,\mathcal{R}_{\mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}}_i}\,\underline{N} \iff$
   $\forall j \in I, \forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}_i), (\underline{M} \models (j \mapsto \underline{\phi}) \Rightarrow \underline{N} \models (j \mapsto \underline{\phi})) \iff$
   $\forall j \in I, \forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}_j), (\underline{M}\ j \models \underline{\phi} \Rightarrow \underline{N}\ j \models \underline{\phi}) \iff$
   $\forall j \in I, \underline{M}\ j\,\mathcal{R}_{\underline{\mathbf{C}}_j}\,\underline{N}\ V_j$.

□

Many other properties can be derived from Lemma 3.4.1 and 3.4.3. Here we look at a selection of three interesting examples, though this list is not comprehensive.

- $\mathsf{thunk}(\underline{M}) \equiv_{\mathbf{U}\,\underline{\mathbf{C}}} \mathsf{thunk}(\underline{N}) \iff \underline{M} \equiv_{\underline{\mathbf{C}}} \underline{N}$.
- $x : \mathbf{A} \vdash \underline{M} \equiv^{\circ}_{\underline{\mathbf{C}}} \underline{N} : \underline{\mathbf{C}} \iff \lambda x.\,\underline{M} \equiv_{\mathbf{A}\to\underline{\mathbf{C}}} \lambda x.\,\underline{N}$.
- $\langle \underline{M}_i \mid i \in I\rangle \equiv_{\mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}}_i} \langle \underline{N}_i \mid i \in I\rangle \iff \forall j \in I, \underline{M}_j \equiv_{\underline{\mathbf{C}}_j} \underline{N}_j$.

3.4.1 Differences between equivalences for nondeterminism

The various logics for nondeterminism give arguably the most diverse collection of logical equivalences out of all the examples given in this chapter. Not only does the logical equivalence depend on the chosen modalities, the intersection of the may equivalence created by the $\Diamond$ modality and the must equivalence created by the $\Box$ modality does not yield the neutral equivalence given by the $\mathcal{O} = \{\Diamond, \Box\}$. Moreover, including or excluding negation changes the equivalence for any type of nondeterminism.

For terms of type $\mathbf{F}\,\mathbf{1}$, we have the basic equivalences which hold for both the general and the positive logic:

$$\mathsf{return}(*) \quad \equiv_{\{\Diamond\}} \quad \mathsf{or}(\mathsf{return}(*), \Omega) \quad \sqsubseteq^{+}_{\{\Diamond\}} \quad \Omega$$

$$\mathsf{return}(*) \quad \sqsubseteq^{+}_{\{\Box\}} \quad \mathsf{or}(\mathsf{return}(*), \Omega) \quad \equiv_{\{\Box\}} \quad \Omega$$

$(-)$	$[\langle(\ -\ )(\top)\rangle]$	$\underline{M} := \mathbf{rt}(\mathsf{or}(\Omega, \circledast))$	$\underline{N} := \mathsf{or}(\mathbf{rt}(\Omega), \mathbf{rt}(\circledast))$	$\underline{K} := \underline{M}\ \mathsf{or}\ \underline{N}$
$\Diamond$	$\Diamond$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\Box$	$\Box$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\Diamond$	$\Box$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\Box$	$\Diamond$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\Diamond$	$\neg(\Diamond)$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\Box$	$\neg(\Box)$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\Diamond$	$\neg(\Box)$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\Box$	$\neg(\Diamond)$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\Diamond$	$\Diamond \wedge \neg(\Box)$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{T}$
$\Diamond$	$\Box \vee \neg(\Diamond)$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\Box$	$\Diamond \wedge \neg(\Box)$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\Box$	$\Box \vee \neg(\Diamond)$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{F}$

$(-)$	$[\langle(\ -\ )(\top)\rangle]$	$\mathsf{or}(\Omega, \mathbf{rt}(\circledast))$	$\mathsf{or}(\Omega, \mathsf{or}(\mathbf{rt}(\Omega), \mathbf{rt}(\circledast)))$
$\Diamond$	$\Diamond$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\Box$	$\Box$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\Diamond$	$\Box$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\Box$	$\Diamond$	$\boldsymbol{F}$	$\boldsymbol{F}$
$\Diamond$	$\neg(\Diamond)$	$\boldsymbol{F}$	$\boldsymbol{T}$
$\Box$	$\neg(\Box)$	$\boldsymbol{F}$	$\boldsymbol{F}$

Table 3.1: Differences between the nondeterministic equivalences

For type **FUF1**, more differences between the logics can be observed. Examples of these differences are adapted from [41, 70] using the coincidence between the behavioural equivalence and applicative bisimilarity proven in the forthcoming Chapter 4.

We study five terms and the logical formulas they satisfy in Table 3.1. The logical formulas are given in a systematic way, in the shape of $(-)[\langle(\ -\ )(\top)\rangle]$ with two slots that can be filled with logical combinations of modalities.

The first column only contains base modalities, since Lemma 3.3.5 establishes that the behavioural equivalence and positive behavioural preorder only depend on basic formulas. The second column contain more complex logical expressions of modalities. In particular, the second column of the first table features all possible distinct logical combinations of modalities: any other expression is equivalent to one featured in the table. For example, $\Box \wedge \Diamond \equiv \Box$. The two columns together give all pairings of expressions

relevant to our study of the behavioural preorders.

From the first table of 3.1, we can deduce that the three terms $\underline{M}$, $\underline{N}$ and $\underline{K}$ are equal according to the equivalences $\equiv^+_{\{\Diamond\}}$ and $\equiv^+_{\{\Box\}}$ as they satisfy precisely the same set of formulas from the table, and any basic positive formula of type $\mathbf{F}\,\mathbf{U}\,\mathbf{F}\,\mathbf{1}$ is equivalent to a formula in the table. However, $\underline{M}$ and $\underline{N}$ are not equal according to the equivalences $\equiv^+_{\{\Diamond,\Box\}}$, $\equiv_{\{\Diamond\}}$ and $\equiv_{\{\Box\}}$, while for these equivalences $\underline{N}$ and $\underline{K}$ are still equal. The only equivalence for which all three terms are considered different, is the general behavioural equivalence $\equiv_{\{\Diamond,\Box\}}$, where composite formulas using $\vee$ or $\wedge$ are necessary to prove the difference between $\underline{N}$ and $\underline{K}$.

The second table in 3.1 gives an example of a difference between the relations $\equiv^+_{\{\Diamond,\Box\}}$ and $\equiv_{\{\Diamond\}} \cap \equiv_{\{\Box\}}$. We can conclude that the following equivalences are all different:

$$\equiv^+_{\{\Diamond\}},\ \equiv^+_{\{\Box\}},\ \equiv^+_{\{\Diamond\}} \cap \equiv^+_{\{\Box\}},\ \equiv^+_{\{\Diamond,\Box\}},\ \equiv_{\{\Diamond\}},\ \equiv_{\{\Box\}},\ \equiv_{\{\Diamond\}} \cap \equiv_{\{\Box\}},\ \text{and}\ \equiv_{\{\Diamond,\Box\}}\ .$$

This subsection does not show all sorts of nondeterminism. One example which is excluded is the notion of *ambiguous nondeterminism* as studied in [45, 55], which considers concurrent evaluations. This type of nondeterminism does not fit into the framework of this thesis, as it breaks monotonicity, i.e. there are no upwards closed modalities for specifying the behaviour of ambiguous nondeterministic programs.

3.5 Equational theories

In the study of algebraic effects, equational axiomatisations play a major role. These equational axiomatisations are specified between *algebraic terms* formed from variables and effect operations. In our setting, we have used modalities as a foundation to derive semantic preorders on the set of closed terms. Using the modalities, we can derive which equations are valid for the semantic preorders. This process of going from modalities to equations is quite natural, whereas extracting modalities from equational axiomatisations is not as straightforward.

In this section we explore which equations and inequations are valid with respect to these behavioural preorders, and we relate them to the standard axiomatisations of the effects. Note that by inequation we do not mean a non-equation, but a notion of equation which is not necessarily symmetric.

Instead of looking at equations and inequations between algebraic terms, we look at equations and inequations between effect trees whose leaves are variables, as in [49]. As a basis of exploring validity of equations and inequations with respect to our preorders, we want to extend the concept of behavioural preorder $\sqsubseteq^+$ and $\equiv$ on computation terms to a preorder on effect trees. This has already been done for unit type effect trees in Subsection 3.3.2. Here we generalise this to a preorder on trees of any value type.

Definition 3.5.1. For any value type $\mathbf{A}$, given $t, r \in T(\mathbf{A})$ we say:

- $t \sqsubseteq^+_{\mathbf{A}} r$ if $\forall o \in \mathcal{O}, \forall \phi \in \mathit{Form}(\mathbf{A})_{\mathcal{V}^+}, t[\models \phi] \in [\![o]\!] \Rightarrow r[\models \phi] \in [\![o]\!]$.
- $t \equiv_{\mathbf{A}} r$ if $\forall o \in \mathcal{O}, \forall \phi \in \mathit{Form}(\mathbf{A})_{\mathcal{V}}, t[\models \phi] \in [\![o]\!] \Leftrightarrow r[\models \phi] \in [\![o]\!]$.

Note by Lemma 3.3.5 it holds that for any $\underline{M}, \underline{N} \in \mathit{Terms}(\mathbf{F}\,\mathbf{A})$:

$$\underline{M} \sqsubseteq^{+} \underline{N} \iff |\underline{M}| \sqsubseteq^{+}_{\mathbf{A}} |\underline{N}|, \qquad \underline{M} \equiv \underline{N} \iff |\underline{M}| \equiv_{\mathbf{A}} |\underline{N}|.$$

We now formalise the notion of equation between effect trees. We use the natural numbers $\mathbb{N}$ to enumerate a countable set of variables $x, y, z, \ldots$. As such, we see $e \in T(\mathbb{N})$ as a *variable expression* of effect operators. An *equation* or *inequation* is simply a pair of such effect trees, where for $e, e' \in T(\mathbb{N})$ we can state the equation $e = e'$ and we can state the inequation $e \leq e'$. We will study what it means for such (in)equations to be valid for our behavioural preorders.

Given an expression $e \in T(\mathbb{N})$ and a sequence $f : \mathbb{N} \to T(X)$, we denote by $e\{f(n)/n\}_n$ the substitution of f in e given by $\mu(e[n \mapsto f(n)]) = \mu(f^*(e))$. In the following technical development, we use variables and their associated number interchangeably, e.g., we view $\mathsf{or}(0,1)$ as $\mathsf{or}(x,y)$.

There are two notions of validity of (in)equations we will study; a natural version and an abstract version, and we will show that they coincide if $\mathcal{O}$ is a decomposable set of leaf-upwards closed modalities. We first define a notion of validity for (in)equations with respect to substitution of computation terms. This is the most natural notion of validity with respect to the behavioural preorders, as it directly refers to the extended notions of behavioural preorders $\sqsubseteq^{+}$ and $\equiv$ defined in Definition 3.5.1.

Definition 3.5.2. For $e, e' \in T(\mathbb{N})$ and Value Type $\mathbf{A}$:

- $e \mathrel{\widehat{\sqsubseteq}^{+}_{\mathbf{A}}} e'$ if for any $\underline{M}_{(-)} : \mathbb{N} \to \mathit{Terms}(\mathbf{F}\,\mathbf{A})$, $e\{|\underline{M}_n|/n\}_n \sqsubseteq^{+}_{\mathbf{A}} e'\{|\underline{M}_n|/n\}_n$.
- $e \mathrel{\widehat{\equiv}_{\mathbf{A}}} e'$ if for any $\underline{M}_{(-)} : \mathbb{N} \to \mathit{Terms}(\mathbf{F}\,\mathbf{A})$, $e\{|\underline{M}_n|/n\}_n \equiv_{\mathbf{A}} e'\{|\underline{M}_n|/n\}_n$.

Moreover, the general inequation $e \mathrel{\widehat{\sqsubseteq}^{+}} e'$ holds iff for all $\mathbf{A}$, $e \mathrel{\widehat{\equiv}_{\mathbf{A}}} e'$, and $e \mathrel{\widehat{\equiv}} e'$ holds iff for all $\mathbf{A}$, $e \mathrel{\widehat{\equiv}_{\mathbf{A}}} e'$.

Alternatively, we can define validity of (in)equations with respect to the substitution of unit type trees. This notion is useful, as it only refers to the modalities, and does not reference the programming language and behavioural logic.

Definition 3.5.3. For $e, e' \in T(\mathbb{N})$:

- $e \mathrel{\widehat{\leq}} e'$ if $\forall f : \mathbb{N} \to T(\mathbf{1}), \forall o \in \mathcal{O}$, $e\{f(n)/n\}_n \in [\![o]\!] \Rightarrow e\{f(n)/n\}_n \in [\![o]\!]$.
- $e \mathrel{\widehat{=}} e'$ if $e \mathrel{\widehat{\leq}} e'$ and $e' \mathrel{\widehat{\leq}} e$.

3.5.1 Relating equation definitions

We will see in this subsection, that under sufficient conditions, an (in)equation is valid with respect to term substitutions (Definition 3.5.2) if and only if it is valid with respect to unit type tree substitutions (Definition 3.5.3). In fact, we show that there are more notions of validity which are equivalent to the given two.

Lemma 3.5.4. *For any* $e, e' \in T(\mathbb{N})$, $e \mathrel{\widehat{\leq}} e' \implies e \mathrel{\widehat{\sqsubseteq}^{+}} e'$, *and* $e \mathrel{\widehat{=}} e' \implies e \mathrel{\widehat{\equiv}} e'$.

Proof. Suppose $e \mathrel{\widehat{\leq}} e'$. Let $\mathbf{A}$ be a value type, $\underline{M}_{(-)} : \mathbb{N} \to \mathit{Terms}(\mathbf{F}\,\mathbf{A})$, and take $o \in \mathcal{O}$ and $\phi \in \mathit{Form}(\mathbf{A})_{\mathcal{V}}$ (a general formula) such that $e\{|\underline{M}_n|/n\}_n[\models \phi] \in [\![o]\!]$. Note that $e\{|\underline{M}_n|/n\}_n[\models \phi] = e\{|\underline{M}_n|[\models \phi]/n\}_n$. Since $|\underline{M}_{(-)}|[\models \phi]$ gives a map from $\mathbb{N}$ to $T(\mathbf{1})$, we get (since $e \mathrel{\widehat{\leq}} e'$) that $e'\{|\underline{M}_n|[\models \phi]/n\}_n \in [\![o]\!]$. We conclude that $e \mathrel{\widehat{\leq}} e'$ implies:

$$\forall o \in \mathcal{O}, \forall \phi \in \mathit{Form}(\mathbf{A})_{\mathcal{V}}, (e'\{|\underline{M}_n|/n\}_n[\models \phi] \in [\![o]\!] \Rightarrow e'\{|\underline{M}_n|/n\}_n[\models \phi] \in [\![o]\!])$$

Since $\mathcal{V}^+ \subseteq \mathcal{V}$, we can derive that $e \mathrel{\widehat{\leq}} e'$ implies $e \mathrel{\widehat{\sqsubseteq}^+_{\mathbf{A}}} e'$ for any value type $\mathbf{A}$, so we conclude that $e \mathrel{\widehat{\sqsubseteq}^+} e'$.

If $e \mathrel{\widehat{=}} e'$, then $e \mathrel{\widehat{\leq}} e'$ and $e' \mathrel{\widehat{\leq}} e$, so we can derive with the above result that $e \mathrel{\widehat{\equiv}_{\mathbf{A}}} e'$ for any value type $\mathbf{A}$, so $e \mathrel{\widehat{\equiv}} e'$. □

Lemma 3.5.5. *Suppose $\mathcal{O}$ is a decomposable set of leaf-upwards closed modalities, and $e, e' \in T(\mathbb{N})$, then $e \mathrel{\widehat{\leq}} e'$ holds if and only if:*

$$\forall f : \mathbb{N} \to \{\bot, \langle * \rangle\}, \forall o \in \mathcal{O}, \quad e\{f(n)/n\}_n \in [\![o]\!] \quad \Rightarrow \quad e'\{f(n)/n\}_n \in [\![o]\!].$$

Proof. The left to right direction is trivial, since $\{\bot, \langle * \rangle\} \subseteq T(\mathbf{1})$.

The other direction makes use of decomposability. Suppose:

(I) $\forall f : \mathbb{N} \to \{\bot, \langle * \rangle\}$ and $o \in \mathcal{O}$, $e\{f(n)/n\}_n \in [\![o]\!] \Rightarrow e'\{f(n)/n\}_n \in [\![o]\!]$.

Take some substitution of unit trees $g : \mathbb{N} \to T(\mathbf{1})$ and a modality $o \in \mathcal{O}$ such that $e\{g(n)/n\}_n \in [\![o]\!]$, we want to prove that $e'\{g(n)/n\}_n \in [\![o]\!]$.

Note that $\mu(e[n \mapsto g(n)]) = e\{g(n)/n\}_n$. We want to apply decomposability on the two double trees $e[n \mapsto g(n)]$, $e'[n \mapsto g(n)] \in T(T(\mathbf{1}))$. Assume $D \subseteq T(\mathbf{1})$ and $o' \in \mathcal{O}$ such that $e[n \mapsto g(n)] \in o'(D)$. We define the following function, $f : \mathbb{N} \to \{\bot, \langle * \rangle\}$, where $f(n) = \langle * \rangle$ precisely when $g(n) \in D$. Then $e\{f(n)/n\}_n = e[n \mapsto g(n)][\in D]$, so $e\{f(n)/n\}_n \in [\![o']\!]$.

By assumption (I), $e'\{f(n)/n\}_n \in [\![o']\!]$, so $e'[n \mapsto g(n)] \in o'(D)$. Since $D \subseteq (\preccurlyeq^{\uparrow}[D])$ and o' is leaf-upwards closed, we can derive that $e'[n \mapsto g(n)] \in o'(\preccurlyeq^{\uparrow}[D])$.

By Lemma 3.3.19, we have derived that $e[n \mapsto g(n)] \preccurlyeq e'[n \mapsto g(n)]$. So by decomposability, $e\{g(n)/n\}_n = \mu(e[n \mapsto g(n)]) \preccurlyeq \mu(e'[n \mapsto g(n)]) = e'\{g(n)/n\}_n$, hence $e'\{g(n)/n\}_n \in [\![o]\!]$. We can conclude that $e \mathrel{\widehat{\leq}} e'$.

□

Combining the previous lemma with Lemma 3.5.4 we get:

Proposition 3.5.6. *Suppose $\mathcal{O}$ is a decomposable set of leaf-upwards closed modalities, $e, e' \in T(\mathbb{N})$, and $\mathbf{A}$ an inhabited value type (there is a term of type $\mathbf{A}$), then the following 5 statements are equivalent:*

1. $e \mathrel{\widehat{\leq}} e'$.
2. $e \mathrel{\widehat{\sqsubseteq}^+} e'$.
3. $e \mathrel{\widehat{\sqsubseteq}^+_{\mathbf{A}}} e'$.
4. $\forall f : \mathbb{N} \to \{\bot, \langle * \rangle\}, \forall o \in \mathcal{O}, e\{f(n)/n\}_n \in [\![o]\!] \Rightarrow e'\{f(n)/n\}_n \in [\![o]\!]$.
5. $\forall f : \mathbb{N} \to \{\bot, x\}$, $e\{f(n)/n\}_n \mathrel{\widehat{\leq}} e'\{f(n)/n\}_n$ *(where $x = 0 \in \mathbb{N}$).*

Proof. We prove that the first four statements are equivalent, and then prove they are equivalent to the fifth statement.

(1. $\Rightarrow$ 2.) This case is given by Lemma 3.5.4.

(2. $\Rightarrow$ 3.) This case holds by definition of $\widehat{\sqsubseteq}^{+}$.

(3. $\Rightarrow$ 4.) Assume that $e \,\widehat{\sqsubseteq}^{+}_{\mathbf{A}} e'$ holds. Let $f : \mathbb{N} \to \{\bot, \langle * \rangle\}$ and $o \in \mathcal{O}$ such that $e\{f(n)/n\}_n \in [\![o]\!]$. Let V : $\mathbf{A}$ some value term, $\phi := \top \in \mathit{Form}(\mathbf{A})_{\mathcal{V}^+}$ and $\underline{M}_{(-)} : \mathbb{N} \to \mathit{Terms}(\mathbf{F}\,\mathbf{A})$ such that $\underline{M}_n := \Omega$ if $f(n) = \bot$, else $\underline{M}_n := \mathsf{return}(V)$. Then $e\{|\underline{M}_n|/n\}_n[\models \phi] = e\{f(n)/n\}_n \in [\![o]\!]$. Since $e \,\widehat{\sqsubseteq}^{+}_{\mathbf{A}} e'$, we can conclude that $e'\{f(n)/n\}_n = e'\{|\underline{M}_n|/n\}_n[\models \phi] \in [\![o]\!]$.

(4. $\Rightarrow$ 1.) This case holds because of Lemma 3.5.5.

(5. $\Leftrightarrow$ 4.) For this case, we use the already established equivalence (1. $\Leftrightarrow$ 4.) to unfold statement 5. to: $\forall f : \mathbb{N} \to \{\bot, 0\}, \forall g : \mathbb{N} \to \{\bot, \langle * \rangle\}, \forall o \in \mathcal{O}$,
$e\{g(f(n))/n\}_n \in [\![o]\!] \Rightarrow e'\{g(f(n))/n\}_n \in [\![o]\!]$,
which is equivalent to statement 4.

□

With similar proofs, using symmetry, we have the following analogous result.

Proposition 3.5.7. *Suppose $\mathcal{O}$ is a decomposable set of leaf-upwards closed modalities, $e, e' \in T(\mathbb{N})$, and $\mathbf{A}$ an inhabited value type, then the following 5 statements are equivalent:*

1. $e \mathrel{\widehat{=}} e'$. 2. $e \mathrel{\widehat{\equiv}} e'$. 3. $e \mathrel{\widehat{\equiv}}_{\mathbf{A}} e'$.

4. $\forall f : \mathbb{N} \to \{\bot, *\}$, $\forall o \in \mathcal{O}$, $e\{f(n)/n\}_n \in [\![o]\!] \Leftrightarrow e'\{f(n)/n\}_n \in [\![o]\!]$.

5. $\forall f : \mathbb{N} \to \{\bot, x\}$, $e\{f(n)/n\}_n \mathrel{\widehat{=}} e'\{f(n)/n\}_n$ *(where $x = 0 \in \mathbb{N}$).*

If alternatively, Scott openness holds, we have a slightly weaker result:

Proposition 3.5.8. *Suppose all $o \in \mathcal{O}$ are Scott open, then $e \,\widehat{\sqsubseteq}^{+}_{\mathbf{1}} e' \implies e \mathrel{\widehat{\leq}} e'$.*

Proof. Let $f : \mathbb{N} \to T(\mathbf{1})$, and for each $m \in \mathbb{N}$ we write $f(n)_m$ for the m-th finite approximation of the tree $f(n)$. Suppose that for $o \in \mathcal{O}$ we have that $e\{f(n)/n\}_n \in [\![o]\!]$. Since o is Scott open, there must be an $m \in \mathbb{N}$ such that $e\{f(n)_m/n\}_n \in [\![o]\!]$ (since $\bigsqcup_m e\{f(n)_m/n\}_n = e\{f(n)/n\}_n$). For each n, $f(n)_m$ is finite, so by Lemma 2.2.11 there is a term $\underline{M}_n$ such that $|\underline{M}_n| = f(n)_m$. Define $\underline{M}_{(-)} : \mathbb{N} \to \mathbf{F}\,\mathbf{1}$ by choosing the $\underline{M}_n$ as above, then $e\{|\underline{M}_n|/n\}_n = e\{f(n)_m/n\}_n \in [\![o]\!]$. Since by assumption $e \,\widehat{\sqsubseteq}^{+}_{\mathbf{1}} e'$, it holds that $e'\{f(n)_m/n\}_n = e'\{|\underline{M}_n|/n\}_n \in [\![o]\!]$, hence with o being Scott open (hence upwards closed) this implies that $e'\{f(n)/n\}_n \in [\![o]\!]$. We conclude that $e \mathrel{\widehat{\leq}} e'$. □

3.5.2 General properties

We show that the properties of *admissibility* and *compositionality* from [35] for $\widehat{\leq}$ are consequences of the properties of the modalities.

Definition 3.5.9. The preorder $\widehat{\leq}$ is *admissible*, if for any two sequences of expressions $e_n, e'_n \in T(\mathbb{N})$ such that $\forall n \in \mathbb{N}$, $e_n \leq e_{n+1}$ and $e'_n \leq e'_{n+1}$,

$$(\forall n \in \mathbb{N}, e_n \mathrel{\widehat{\leq}} e'_n) \quad \Rightarrow \quad (\sqcup_n e_n) \mathrel{\widehat{\leq}} (\sqcup_n e'_n).$$

Proposition 3.5.10. *If all $o \in \mathcal{O}$ are Scott open, then $\widehat{\leq}$ is admissible.*

Proof. A simple derivation:
$(\sqcup_n e_n)\{f(m)/m\}_m \in [\![o]\!] \Rightarrow \exists n, e_n\{f(m)/m\}_m \in [\![o]\!] \Rightarrow$
$\exists n, e'_n\{f(m)/m\}_m \in [\![o]\!] \Rightarrow (\sqcup_n e_n)\{f(m)/m\}_m \in [\![o]\!]$. □

Definition 3.5.11. The preorder $\widehat{\leq}$ is *compositional*, if for any $e \mathrel{\widehat{\leq}} e'$ and any $E, E' : \mathbb{N} \to T(\mathbb{N})$ such that $\forall n \in \mathbb{N}, E(n) \mathrel{\widehat{\leq}} E'(n)$, $e\{E(n)/n\}_n \mathrel{\widehat{\leq}} e'\{E'(n)/n\}_n$.

Proposition 3.5.12. *If $\mathcal{O}$ is a decomposable set of leaf-upwards closed modalities, then $\widehat{\leq}$ is compositional.*

Proof. Let $e \mathrel{\widehat{\leq}} e'$ and $E, E' : \mathbb{N} \to T(\mathbb{N})$ such that $\forall n \in \mathbb{N}, E(n) \mathrel{\widehat{\leq}} E'(n)$. We use the equivalence (1. $\Leftrightarrow$ 4.) from Proposition 3.5.6.

Take $g : \mathbb{N} \to \{\bot, \langle * \rangle\}$ and $o \in \mathcal{O}$ such that $(e\{E_n/n\}_n)\{g(m)/m\}_m = e\{E_n\{g(m)/m\}_m/n\}_n \in [\![o]\!]$. Note that $\mu(e[n \mapsto E_n\{g(m)/m\}_m]) = e\{E_n\{g(m)/m\}_m/n\}_n$. We prove the requirement of decomposability, using Lemma 3.3.19, that for any $D \subseteq T(\mathbf{1})$ and $o \in \mathcal{O}$ we have
$e[n \mapsto E_n\{g(m)/m\}_m] \in o(D) \implies e[n \mapsto E_n\{g(m)/m\}_m] \in o(\preccurlyeq^{\uparrow}[D])$,
to conclude that $e'\{E_n\{g(m)/m\}_m/n\}_n = \mu(e'[n \mapsto E_n\{g(m)/m\}_m]) \in [\![o]\!]$.

Take some arbitrary $D \subseteq T(\mathbf{1})$. Note first that for any $n \in \mathbb{N}$, it holds that $E_n\{g(m)/m\}_m \preccurlyeq E'_n\{g(m)/m\}_m$ (since $E_n\{g(m)/m\}_m[\in \emptyset] = E_n\{g(m)[\in \emptyset]/m\}_m$ and $(n \mapsto g(n)[\in \emptyset])$ is a function from $\mathbb{N}$ to $T(\mathbf{1})$). Hence $\forall n, E_n\{g(m)/m\}_m \in D \Rightarrow E'_n\{g(m)/m\}_m \in (\preccurlyeq^{\uparrow}[D])$.

Let $h : \mathbb{N} \to \{\bot, \langle * \rangle\}$, where $h(n) := \langle * \rangle$ if $E_n\{g(m)/m\}_m \in D$, otherwise $h(n) := \bot$. So in particular, (I): $h(n) = \langle * \rangle$ implies $E'_n\{g(m)/m\}_m \in (\preccurlyeq^{\uparrow}[D])$. Remember that $e \mathrel{\widehat{\leq}} e'$, so suppose for some $o' \in \mathcal{O}$ such that $e\{h(n)/n\}_n \in o'(\{*\})$, $e[n \mapsto E_n\{f(m)/m\}_m] \in o'(D)$ holds, so it follows that $e'\{h(n)/n\}_n \in o'(\{*\})$. Now by observation (I) and leaf-upwards closure of o', $e'[n \mapsto E'_n\{f(m)/m\}_m] \in o'(\preccurlyeq^{\uparrow}[D])$. This proves that $e[n \mapsto E_n\{f(m)/m\}_m] \preccurlyeq e'[n \mapsto E'_n\{f(m)/m\}_m]$.

By decomposability, $\mu(e[n \mapsto E_n\{g(m)/m\}_m]) \preccurlyeq \mu(e'[n \mapsto E'_n\{g(m)/m\}_m])$ and hence $e'\{E'_n\{g(m)/m\}_m/n\}_n \in [\![o]\!]$. This is for all $g : \mathbb{N} \to \{\bot, \langle * \rangle\}$ and $o \in \mathcal{O}$, so we can conclude with Proposition 3.5.6 that $e\{E_n/n\}_n \mathrel{\widehat{\leq}} e'\{E'_n/n\}_n$.

□

Hence if $\mathcal{O}$ is a decomposable set of Scott open modalities, then $\widehat{\leq}$ is admissible, and both $\widehat{\leq}$ and $\mathrel{\widehat{=}}$ are compositional and completely determined by substitutions of $\langle * \rangle$ and $\bot$.

3.5.3 The equations and inequations of the examples

In this subsection, we look at the traditional axiomatic equations and inequations for effects from, e.g., [81, 83], and prove that they are valid for our behavioural equivalence, according to any notion of validity established above. An inequation which holds for any effect is:

$$(\mathrm{B}): \quad \bot \leq x$$

This inequation will hold for $\mathrel{\widehat{\leq}}$ whenever all modalities are upwards closed. More generally, we expect admissability and compositionality to hold. We will have a look at the other traditional axiomatic equations used for defining effects. The point is to establish that these axioms/rules hold for our derived equations $\mathrel{\widehat{=}}$ and inequations $\mathrel{\widehat{\leq}}$. Lemmas 3.5.6 and 3.5.7 make this easier to establish, since we only need to check whether they hold when substituting either $\bot$ or $\langle * \rangle$.

For error and input/output, there are no particular equations or inequations besides (B). This reflects the fact that any two different effect trees of unit type are considered behaviourally different. The other effects do have traditional axiomatic equations.

Nondeterminism: Subsections 2.3.3 and 3.2.3.

$$(\mathrm{N1}): \quad \mathsf{or}(x,x) = x.$$
$$(\mathrm{N2}): \quad \mathsf{or}(x,y) = \mathsf{or}(y,x).$$
$$(\mathrm{N3}): \quad \mathsf{or}(x,\mathsf{or}(y,z)) = \mathsf{or}(\mathsf{or}(x,y),z).$$
$$(\mathrm{NA}): \quad x \leq \mathsf{or}(x,y).$$
$$(\mathrm{ND}): \quad \mathsf{or}(x,y) \leq x.$$

where NA and ND are for angelic and demonic nondeterminism respectively.

Probabilistic: Subsections 2.3.4 and 3.2.4.

$$(\mathrm{P1}): \quad \mathsf{pr}(x,x) = x.$$
$$(\mathrm{P2}): \quad \mathsf{pr}(x,y) = \mathsf{pr}(y,x).$$
$$(\mathrm{P3}): \quad \mathsf{pr}(\mathsf{pr}(x,y),\mathsf{pr}(z,w)) = \mathsf{pr}(\mathsf{pr}(x,z),\mathsf{pr}(y,w)).$$
$$(\mathrm{P4}): \quad \mathsf{pr}(\mathsf{pr}(\mathsf{pr}(\ldots,x),x),x) = x.$$

Where $\mathsf{pr}(\mathsf{pr}(\mathsf{pr}(\ldots,x),x),x)$ is the infinite tree t such that $t = \mathsf{pr}(t,x)$.

Global Store: Subsections 2.3.5 and 3.2.5. For all $l, r \in \mathsf{Loc}$:

$$(\mathrm{G1}): \quad \mathsf{lookup}_l(i \mapsto x) = x.$$
$$(\mathrm{G2}): \quad \mathsf{update}_l(n;\mathsf{update}_l(m;x)) = \mathsf{update}_l(m;x).$$
$$(\mathrm{G3}): \quad \mathsf{update}_l(n;\mathsf{lookup}_l(i \mapsto x_i)) = \mathsf{update}_l(n;x_n).$$

$$(\mathrm{G4}): \quad \mathsf{lookup}_l\,(i \mapsto \mathsf{update}_l(i; x_i)) = \mathsf{lookup}_l(i \mapsto x_i).$$

$$(\mathrm{G5}): \quad \mathsf{update}_l(n; \mathsf{update}_r(m; x)) = \mathsf{update}_r(m; \mathsf{update}_l(n; x)) \quad \text{if } l \neq r.$$

$$(\mathrm{G6}): \quad \mathsf{update}_l(n; \mathsf{lookup}_r(i \mapsto x_i)) = \mathsf{lookup}_r(i \mapsto \mathsf{update}_l(n; x_i)) \quad \text{if } l \neq r.$$

$$(\mathrm{G7}): \quad \mathsf{update}_l(n; \bot) = \bot.$$

(G7) is actually a consequence of the other six rules, rule (B), and compositionality. This is proven in the case of a single Boolean state in Lemma 3.5.14.

Timer: Subsections 2.3.7 and 3.2.7. We look at some the equations for the down-interpretation of this effect. For all $c, d \in \mathsf{Inc}$:

$$(T1): \quad \mathsf{tick}_c(\mathsf{tick}_d(x)) = \mathsf{tick}_{c+d}(x) \qquad \text{if } (c+d) \in \mathsf{Inc}.$$

$$(T2): \quad \mathsf{tick}_d(x) \leq \mathsf{tick}_c(x) \qquad \text{if } d > c.$$

$$(T3): \quad \mathsf{tick}_c(\bot) = \bot$$

Lemma 3.5.13. *For each effect, with its effect signature Σ and chosen modalities $\mathcal{O}$, the resulting inequation $\mathrel{\widehat{\leq}}$ satisfies the effect specific inequalities given above in the following way:*

If $e \leq e'$ as above, then $e \mathrel{\widehat{\leq}} e'$.

If $e = e'$ as above, then $e \mathrel{\widehat{=}} e'$.

Proof. We use equivalence (1. $\Leftrightarrow$ 4.) from Propositions 3.5.6 and 3.5.7, so we may assume the variables are instantiated by either $\bot$ or $\langle * \rangle$. We need to prove that for any given inequality, if the left satisfies some modality then the right does, and for any given equality, the left satisfies a modality if and only if the right does. First note that (B) holds for all effect examples since all modalities are upwards closed. Moreover, $\mathrel{\widehat{\leq}}$ and $\mathrel{\widehat{=}}$ are admissable and compositional too, since we have a decomposable set of Scott open modalities for each effect.

Nondeterminism: Note first that by the definition of the modalities, $\mathsf{or}(x, y) \in [\![\Diamond]\!] \iff x \in [\![\Diamond]\!] \vee y \in [\![\Diamond]\!]$ and $\mathsf{or}(x, y) \in [\![\Box]\!] \iff x \in [\![\Box]\!] \wedge y \in [\![\Box]\!]$. So (N1), (N2) and (N3) hold because of the reflexivity, symmetry and transitivity of the connectives $\vee$ and $\wedge$. (NA) hold since if predicate P holds, then $P \vee Q$ holds. (ND) is true, since if $P \wedge Q$ holds, then P holds.

Probability: Remember that $t \in [\![\mathsf{P}_{>q}]\!] \iff \mathsf{P}(t) > q$, where $\mathsf{P}(t)$ is the probability of termination. Therefore, an equation holds if for any substitution of $\bot$-s and unit leaves, both sides have the same probability of termination. Note that $\mathsf{P}(\mathsf{pr}(x, y)) = (\mathsf{P}(x) + \mathsf{P}(y))/2$ because of how P is formulated, so it is easy to verify that (P1), (P2) and (P3) hold. For (P4), observe that that $\mathsf{P}(\mathsf{pr}(\mathsf{pr}(\ldots, x), x), x)) = \sup_n \mathsf{P}_n(\mathsf{pr}(\mathsf{pr}(\ldots, x), x)) = \sup_n(\sum_{1 \leq i < n} \mathsf{P}_n(x)/2^i) = \sup_n((1 - 2^{-n})\mathsf{P}_n(x)) = \mathsf{P}(x)$.

Global Store: A tree $t \in T(\mathbf{1})$ satisfies the modality $(s \rightarrowtail r)$ if the function application $exec(t, s)$ results in $(*, r)$. So $e \mathrel{\widehat{=}} e'$ holds if and only if $\forall f : \mathbb{N} \to \{\bot, \langle * \rangle\}$, and all $s \in \mathsf{State}$, $exec(e\{f(n)/n\}_n, s) = exec(e'\{f(n)/n\}_n, s)$ (if one side is undefined, the other side is too). It can be verified that for each of the given equations, the result of

applying the *exec* function is the same. E.g. (G3): $exec(\mathsf{update}_l(n; \mathsf{lookup}_l(i \mapsto x_i)), s) = exec(\mathsf{lookup}_l(i \mapsto x_i), s[l := n]) = exec(x_{s[l:=n](l)}, s[l := n]) = exec(\mathsf{update}_l(n; x_n), s)$.

Timer: Note that $\langle * \rangle$ satisfies any $\mathsf{C}_{\leq d}$, $\bot$ satisfies no $\mathsf{C}_{\leq d}$, and $\mathsf{tick}_c(t)$ satisfies $\mathsf{C}_{\leq d}$ if and only if $c \leq d$ and t satisfies $\mathsf{C}_{\leq d-c}$. Given these facts, the equations can be easily verified. E.g for (T2), take $e \in \mathbb{Q}$:

For $x := \langle * \rangle$, $\mathsf{tick}_d(x) \in [\![\mathsf{C}_{\leq e}]\!]$ implies $d \leq e$, hence $c \leq e$ and $\mathsf{tick}_c(x) \in [\![\mathsf{C}_{\leq e}]\!]$.

For $x := \bot$, $\mathsf{tick}_d(\bot) \notin [\![\mathsf{C}_{\leq e}]\!]$. □

3.5.4 Problematic effect combinations

In this subsection, we give examples of two inequational theories for computationally interesting effect combinations, that cannot be represented by a decomposable set of upwards closed modalities. This motivates the transition to a quantitative logic formulated in Chapter 6.

Demonic nondeterminism + Global Store

We combine the effects of demonic nondeterminism with global store. For simplicity, we only have one store location, which contains a Boolean value. The problem however carries over to the more complex case of multiple $\mathbb{N}$-valued stores. The signature Σ is comprised of four operators:

$$\{\mathsf{or}(-,-) : \alpha \times \alpha \to \alpha, \mathsf{up}_{\boldsymbol{T}}(-) : \alpha \to \alpha, \mathsf{up}_{\boldsymbol{F}}(-) : \alpha \to \alpha, \mathsf{look}(-,-) : \alpha \times \alpha \to \alpha\}$$

$\mathsf{up}_{\boldsymbol{T}}(\underline{M})$ stores $\boldsymbol{T}$ in the global store, and continues with $\underline{M}$. $\mathsf{up}_{\boldsymbol{F}}(\underline{M})$ stores $\boldsymbol{F}$ in the global store, and continues with $\underline{M}$. $\mathsf{look}(\underline{M}, \underline{N})$ continues with $\underline{M}$ if $\boldsymbol{F}$ is stored in the global store, else it continues with $\underline{N}$.

We assume that we have a decomposable set of upwards closed modalities $\mathcal{O}$, such that $\widehat{\leq}$ (from Definition 3.5.3) satisfies the desirable inequations. Firstly, note that

$$(1) : \bot \leq x$$

holds because all modalities are upwards closed. This inequation is expected to hold regardless of that fact.

Secondly, we assume that inequations related to demonic nondeterminism and related to global store hold for the combination of the two effects. As such, from demonic nondeterminism we want the following inequations:

$$(2) : \mathsf{or}(x, y) \leq x, \quad \mathsf{or}(x, y) \leq y, \qquad (3) : x = \mathsf{or}(x, x) \ .$$

From global store, adapting the general equations to this single Boolean store location case, we want the following equations:

$$(4) : \mathsf{up}_a(\mathsf{up}_b(x)) = \mathsf{up}_b(x), \qquad (5) : \mathsf{look}(x, x) \mathrel{\widehat{=}} x$$
$$(6) : \mathsf{look}(\mathsf{up}_{\boldsymbol{F}}(x), y) = \mathsf{look}(x, y), \quad \mathsf{look}(x, \mathsf{up}_{\boldsymbol{T}}(y)) = \mathsf{look}(x, y)$$

$$(7) : \mathsf{up}_{\boldsymbol{F}}(\mathsf{look}(x,y)) = \mathsf{up}_{\boldsymbol{F}}(x), \quad \mathsf{up}_{\boldsymbol{T}}(\mathsf{look}(x,y)) = \mathsf{up}_{\boldsymbol{T}}(y)$$

where (4) holds for any $a, b \in \{\boldsymbol{T}, \boldsymbol{F}\}$. These are not all the inequations one would expect from the effects, but listed here are the ones used in the proofs. We also do not need to include equations for how the two effects interact, the following results are a consequence of the above inequations and equations only.

Lemma 3.5.14. *Suppose $\leq'$ is compositional and satisfies rules (1),(4),(5) and (6), then it holds that $\mathsf{up}_{\boldsymbol{T}}(\bot) =' \bot =' \mathsf{up}_{\boldsymbol{F}}(\bot)$, where $(='') = (\leq') \cap (\geq')$.*

Proof. It holds that $\bot \leq' \mathsf{up}_{\boldsymbol{T}}(\bot)$ by rule (1), hence by using compositionality on the inequation $\mathsf{up}_{\boldsymbol{F}}(x) \leq' \mathsf{up}_{\boldsymbol{F}}(x)$ we can derive that $\mathsf{up}_{\boldsymbol{F}}(\bot) \leq' \mathsf{up}_{\boldsymbol{F}}(\mathsf{up}_{\boldsymbol{T}}(\bot))$. By rule (4) it holds that $\mathsf{up}_{\boldsymbol{F}}(\mathsf{up}_{\boldsymbol{T}}(\bot)) = \mathsf{up}_{\boldsymbol{T}}(\bot)$, hence $\mathsf{up}_{\boldsymbol{F}}(\bot) \leq' \mathsf{up}_{\boldsymbol{T}}(\bot)$. Similarly, $\mathsf{up}_{\boldsymbol{T}}(\bot) \leq' \mathsf{up}_{\boldsymbol{F}}(\bot)$, so $\mathsf{up}_{\boldsymbol{F}}(\bot) =' \mathsf{up}_{\boldsymbol{T}}(\bot)$.

Using compositionality on $\mathsf{look}(x,y) =' \mathsf{look}(x,y)$, we can derive that $\mathsf{look}(\mathsf{up}_{\boldsymbol{F}}(\bot), \mathsf{up}_{\boldsymbol{F}}(\bot)) =' \mathsf{look}(\mathsf{up}_{\boldsymbol{F}}(\bot), \mathsf{up}_{\boldsymbol{T}}(\bot))$. By rule (5), $\mathsf{look}(\mathsf{up}_{\boldsymbol{F}}(\bot), \mathsf{up}_{\boldsymbol{F}}(\bot)) = \mathsf{up}_{\boldsymbol{F}}(\bot)$. By rule (6) and (5), it holds that $\mathsf{look}(\mathsf{up}_{\boldsymbol{F}}(\bot), \mathsf{up}_{\boldsymbol{T}}(\bot)) =' \mathsf{look}(\bot, \bot) =' \bot$.

We can conclude that $\mathsf{up}_{\boldsymbol{F}}(\bot) =' \bot$, and $\mathsf{up}_{\boldsymbol{T}}(\bot) =' \bot$. □

This is a perfectly natural thing to expect, and holds in the example where one only considers the global store effect. It reflects the fact that when a computation diverges, the (final) global state is not observable.

By applying the previous lemma to $\widehat{\leq}$, we can now derive the following result.

Proposition 3.5.15. *Given a decomposable set $\mathcal{O}$ of upwards closed modalities, such that $\widehat{\leq}$ satisfies rules (1) to (7), then it holds that: $\mathsf{up}_{\boldsymbol{T}}(x) \widehat{\leq} x$ and $\mathsf{up}_{\boldsymbol{F}}(x) \widehat{\leq} x$.*

Proof. We use equivalence (1. $\Leftrightarrow$ 5.) from Proposition 3.5.7 and Lemma 3.5.14 to prove that $\mathsf{or}(\mathsf{up}_{\boldsymbol{T}}(y), z) \widehat{\leq} y$. There are only four different substitutions of $\{\bot, x\}$ for y and z. In the following table, we check that for each substitution, the inequation holds:

y	z	$\mathsf{or}(\mathsf{up}_{\boldsymbol{T}}(y), z)$				y
$\bot$	$\bot$	$\mathsf{or}(\mathsf{up}_{\boldsymbol{T}}(\bot), \bot)$		$\widehat{\leq}^{(2)}$		$\bot$
x	$\bot$	$\mathsf{or}(\mathsf{up}_{\boldsymbol{T}}(x), \bot)$	$\widehat{\leq}^{(2)}$	$\bot$	$\widehat{\leq}^{(1)}$	x
$\bot$	x	$\mathsf{or}(\mathsf{up}_{\boldsymbol{T}}(\bot), x)$	$\widehat{\leq}^{(2)}$	$\mathsf{up}_{\boldsymbol{T}}(\bot)$	$\widehat{=}$	$\bot$
x	x	$\mathsf{or}(\mathsf{up}_{\boldsymbol{T}}(x), x)$		$\widehat{\leq}^{(2)}$		x

Table 3.2: Unwanted equation for global store + demonic nondeterminism

The $\widehat{=}$ in the table uses Lemma 3.5.14. We can conclude that $\mathsf{or}(\mathsf{up}_{\boldsymbol{T}}(y), z) \widehat{\leq} y$. Substituting $z := \mathsf{up}_{\boldsymbol{T}}(y)$, we get that: $\mathsf{up}_{\boldsymbol{T}}(y) \widehat{=}^{(3)} \mathsf{or}(\mathsf{up}_{\boldsymbol{T}}(y), \mathsf{up}_{\boldsymbol{T}}(y)) \widehat{\leq} y$. Similarly, we can prove that $\mathsf{up}_{\boldsymbol{F}}(y) \widehat{\leq} y$.

□

Lemma 3.5.16. *For any compositional relation $\leq'$, satisfying rules (1) to (7), and both $\mathsf{up}_{\boldsymbol{T}}(x) \leq' x$ and $\mathsf{up}_{\boldsymbol{F}}(x) \leq' x$, then it holds that $x =' \bot$.*

Proof. By compositionality, composing $\mathsf{up}_{\boldsymbol{T}}(x) \leq' x$ with $\mathsf{up}_{\boldsymbol{F}}(x) \leq' \mathsf{up}_{\boldsymbol{F}}(x)$, we derive that $\mathsf{up}_{\boldsymbol{F}}(\mathsf{up}_{\boldsymbol{T}}(x)) \leq' \mathsf{up}_{\boldsymbol{F}}(x)$, so with rule (4), $\mathsf{up}_{\boldsymbol{T}}(x) \leq' \mathsf{up}_{\boldsymbol{F}}(x)$. We can derive in a similar way that $\mathsf{up}_{\boldsymbol{F}}(x) \leq' \mathsf{up}_{\boldsymbol{T}}(x)$, hence $\mathsf{up}_{\boldsymbol{T}}(x) =' \mathsf{up}_{\boldsymbol{F}}(x)$.

Using Lemma 3.5.14 and rule (7):

$$\bot =' \mathsf{up}_{\boldsymbol{F}}(\bot) =' \mathsf{up}_{\boldsymbol{F}}(\mathsf{look}(\bot, x)) =' \mathsf{up}_{\boldsymbol{T}}(\mathsf{look}(\bot, x)) =' \mathsf{up}_{\boldsymbol{T}}(x).$$

Similarly, $\mathsf{up}_{\boldsymbol{F}}(x) = \bot$, so we can conclude by rule (5) and (6) that $\bot =' \mathsf{look}(\bot, \bot) =' \mathsf{look}(\mathsf{up}_{\boldsymbol{F}}(x), \mathsf{up}_{\boldsymbol{T}}(x)) =' \mathsf{look}(x, x) =' x$. □

We can combine all the results together in the following negative result.

Proposition 3.5.17. *Given a decomposable set of upwards closed modalities, such that rules (2) to (7) are valid for $\mathrel{\widehat{\leq}}$, then $x \mathrel{\widehat{=}} y$ (hence all expressions are equal).*

Proof. Rule (1) follows from the fact that all modalities are upwards closed. So from the previous two results, we can derive that $x \mathrel{\widehat{=}} \bot \mathrel{\widehat{=}} y$.

□

Angelic nondeterminism + probability

Another problematic combination of effects is the combination of nondeterminism and probability. The fact that it is difficult to formulate such a combination of effects with Boolean valued predicates is a common observation. Take for instance [49], where it was observed that quantitative expectations used for interpreting this effect combination could not be replaced by Boolean probabilities. Here we will look at a similar but slightly different observation, that this combination cannot be adequately modelled by a decomposable set of Scott open modalities.

We will look at angelic nondeterminism in particular, with the effect signature: $\Sigma := \{\mathsf{or}(-,-) : \alpha \times \alpha \to \alpha, \mathsf{pr}(-,-) : \alpha \times \alpha \to \alpha\}$. We assume a decomposable set of upwards closed modalities $\mathcal{O}$, such that we have the following equations and inequations from the theory of probability and angelic nondeterminism are satisfied by $\mathrel{\widehat{=}}$ and $\mathrel{\widehat{\leq}}$:

$$\begin{aligned} (1) &: \mathsf{or}(x,x) = x & (2) &: x \leq \mathsf{or}(x,y) \\ (3) &: \mathsf{pr}(x,x) = x & (4) &: y \leq \mathsf{or}(x,y) \\ (5) &: \mathsf{pr}(x,y) = \mathsf{pr}(y,x) \end{aligned}$$

Note that because all modalities are upwards closed, we have $\bot \mathrel{\widehat{\leq}} x$. Moreover, $\mathrel{\widehat{\leq}}$ is compositional, so by rule (2); $x \mathrel{\widehat{\leq}} \mathsf{or}(x, \bot) \mathrel{\widehat{\leq}} \mathsf{or}(x,x) \mathrel{\widehat{\leq}} x$, and hence

$$(6) : x \mathrel{\widehat{=}} \mathsf{or}(x, \bot) \mathrel{\widehat{=}} \mathsf{or}(\bot, x) \ .$$

Proposition 3.5.18. *Suppose $\mathcal{O}$ is a decomposable set of upwards closed modalities such that the above (in)equations hold for $\widehat{\leq}$, then $\mathsf{pr}(\mathsf{or}(x,y),\mathsf{or}(x,z)) \widehat{\leq} \mathsf{or}(x,\mathsf{pr}(y,z))$ (which is very undesirable).*

Proof. We use equivalence (1. $\Leftrightarrow$ 5.):

x	y	z	$\mathsf{pr}(\mathsf{or}(x,y),\mathsf{or}(x,z))$				$\mathsf{or}(x,\mathsf{pr}(y,z))$
$\bot$	$\bot$	$\bot$	$\mathsf{pr}(\mathsf{or}(\bot,\bot),\mathsf{or}(\bot,\bot))$	$\widehat{=}^{(1)}$	$\mathsf{pr}(\bot,\bot)$	$\widehat{=}^{(6)}$	$\mathsf{or}(\bot,\mathsf{pr}(\bot,\bot))$
w	$\bot$	$\bot$	$\mathsf{pr}(\mathsf{or}(w,\bot),\mathsf{or}(w,\bot))$	$\widehat{=}^{(3)}$	$\mathsf{or}(w,\bot)$	$\widehat{\leq}$	$\mathsf{or}(w,\mathsf{pr}(\bot,\bot))$
$\bot$	w	$\bot$	$\mathsf{pr}(\mathsf{or}(\bot,w),\mathsf{or}(\bot,\bot))$	$\widehat{=}^{(6,1)}$	$\mathsf{pr}(w,\bot)$	$\widehat{=}^{(6)}$	$\mathsf{or}(\bot,\mathsf{pr}(w,\bot))$
w	w	$\bot$	$\mathsf{pr}(\mathsf{or}(w,w),\mathsf{or}(w,\bot))$	$\widehat{=}^{(1,6)}$	$\mathsf{pr}(w,w) \widehat{=}^{(3)} w$	$\widehat{\leq}^{(2)}$	$\mathsf{or}(w,\mathsf{pr}(w,\bot))$
$\bot$	w	w	$\mathsf{pr}(\mathsf{or}(\bot,w),\mathsf{or}(\bot,w))$	$\widehat{=}^{(3)}$	$\mathsf{or}(\bot,w)$	$\widehat{=}^{(3)}$	$\mathsf{or}(\bot,\mathsf{pr}(w,w))$
w	w	w	$\mathsf{pr}(\mathsf{or}(w,w),\mathsf{or}(w,w))$	$\widehat{=}^{(3)}$	$\mathsf{or}(w,w) \widehat{=}^{(1)} w$	$\widehat{\leq}^{(2)}$	$\mathsf{or}(w,\mathsf{pr}(w,w))$

Table 3.3: Unwanted equation for probability + angelic nondeterminism

So we conclude that $\mathsf{pr}(\mathsf{or}(x,y),\mathsf{or}(x,z)) \widehat{\leq} \mathsf{or}(x,\mathsf{pr}(y,z))$. □

The inequation $\mathsf{pr}(\mathsf{or}(x,y),\mathsf{or}(x,z)) \widehat{\leq} \mathsf{or}(x,\mathsf{pr}(y,z))$ is undesirable. We give an informal argument for this claim. Say x is a computation which has a probability of $1/4$ of terminating, y always diverges and z always terminates. Since nondeterministic choices are made cooperatively, the computation with the highest probability of termination is chosen. So with the above inequation:

$5/8 = (1/4+1)/2 = \mathsf{pr}(1/4,1) = \mathsf{pr}(\mathsf{or}(1/4,0),\mathsf{or}(1/4,1)) \leq \mathsf{or}(1/4,\mathsf{pr}(0,1)) = \mathsf{or}(1/4,1/2) = 1/2$.

Since informally, $1/2 \leq 5/8$, we can conclude that a computation, whose probability of termination is $5/8$, is equal to a computation whose probability of termination is $1/2$, which contradicts the behaviour of probabilistic programs. This undesirable equality can be proven more formally:

Proposition 3.5.19. *Suppose $\mathcal{O}$ is a decomposable set of upwards closed modalities such that the above (in)equations hold for $\widehat{\leq}$, then $\mathsf{pr}(\mathsf{pr}(\mathsf{pr}(w,\bot),\bot),w) \widehat{=} \mathsf{pr}(w,\bot)$.*

Proof. To see this, we substitute $x := (1/4)w := \mathsf{pr}(\mathsf{pr}(w,\bot),\bot)$ ($1/4$ probability of getting w), $y := \bot$ and $z := w$, in $\mathsf{pr}(\mathsf{or}(x,y),\mathsf{or}(x,z)) \widehat{\leq} \mathsf{or}(x,\mathsf{pr}(y,z))$. The left-hand side is bigger then:
$\mathsf{pr}(\mathsf{or}((1/4)w,\bot),\mathsf{or}((1/4)w,w)) \widehat{\geq}^{(2,4)} \mathsf{pr}((1/4)w,w)$

The right hand side is smaller then:
$\mathsf{or}(\mathsf{pr}(\mathsf{pr}(w,\bot),\bot),\mathsf{pr}(\bot,w)) \widehat{\leq} \mathsf{or}(\mathsf{pr}(\mathsf{pr}(w,w),\bot),\mathsf{pr}(\bot,w)) \widehat{=}^{(3)}$
$\mathsf{or}(\mathsf{pr}(w,\bot),\mathsf{pr}(\bot,w)) \widehat{=}^{(5)} \mathsf{or}(\mathsf{pr}(w,\bot),\mathsf{pr}(w,\bot)) \widehat{=}^{(1)} \mathsf{pr}(w,\bot)$

So we can conclude that $\mathsf{pr}((1/4)w,w) \widehat{\leq} \mathsf{pr}(w,\bot)$. By compositionality and $\bot \widehat{\leq} x$ we have $\mathsf{pr}(w,\bot) \widehat{\leq} \mathsf{pr}(w,(1/4)w)$, so with rule (5) we can conclude that:

$$\mathsf{pr}((1/4)w,w) = \quad \mathsf{pr}(\mathsf{pr}(\mathsf{pr}(w,\bot),\bot),w) \widehat{=} \mathsf{pr}(w,\bot) \ .$$

□

In both examples given in this subsection, we show that the use of a Boolean valued logic forces undesirable equations and inequations to hold. In Chapter 6, we solve this problem by considering a generalisation of the logic, using quantitative valued predicates.

4

Applicative bisimilarity

In the previous chapter, we have established an equivalence relation for terms by inductively defining sets of formulas acting as unary predicates. In this chapter, we look at an alternative approach to equivalence. We will define an equivalence coinductively, as the largest relation exhibiting suitable properties. This is the notion of *bisimilarity* [59, 60], where in particular, we will look at *applicative bisimilarity* in the sense of Abramsky [2, 12, 15, 41].

In a recent paper [14], it is shown how this approach can be used for establishing an equivalence for an untyped lambda calculus with algebraic effects. There, *relators* [46, 102] are used to capture the notion of behaviour of effects, which we will here adapt to lift relations on $\mathbf{A}$ to relations on $\mathbf{F}\,\mathbf{A}$. We will show that our behavioural equivalence is identical to the notion of applicative bisimilarity determined by a relator defined using our modalities. This applicative bisimilarity will be proven to be compatible using a variation of Howe's method [31]. This approach to Howe's method follows closely the methods from [14].

4.1 Relators

In this section, we define the operation that lifts relations on terms of value type $\mathbf{A}$, to relations on terms of computation type $\mathbf{F}\,\mathbf{A}$. This operation, called a *relator*, is fundamental for interpreting effects in the context of applicative bisimilarity. First, we introduce some notation for relations.

A relation on sets X and Y is a subset $\mathcal{R} \subseteq X \times Y$, where we write $x\,\mathcal{R}\,y$ for $x \in X$ and $y \in Y$ precisely when $(x, y) \in \mathcal{R}$. We define three standard operations on relations:

- Given $\mathcal{R} \subseteq X \times Y$, $\mathcal{R}^{op} \subseteq Y \times X$ is the relation such that $y\,\mathcal{R}^{op}\,x \iff x\,\mathcal{R}\,y$.
- Given $\mathcal{R} \subseteq X \times Y$ and $\mathcal{S} \subseteq Y \times Z$, we write $\mathcal{R}\mathcal{S} \subseteq X \times Z$ for relation composition, where $x\,\mathcal{R}\mathcal{S}\,y$ iff there is a $y \in Y$ such that $x\,\mathcal{R}\,y$ and $y\,\mathcal{S}\,z$.
- Given $\mathcal{R} \subseteq X \times Y$, and two functions $f : Z \to X$ and $g : W \to Y$, we define the relation $(f \times g)^{-1}(\mathcal{R}) \subseteq Z \times W$, where $z\ [(f \times g)^{-1}(\mathcal{R})]\ w \iff f(z)\,\mathcal{R}\,g(w)$.

Given a set of modalities $\mathcal{O}$ for some signature Σ, we define a *relator* in the sense of Levy and Thijs [46, 102], which in our setting is an operation that lifts relations on sets to relations on Σ-trees. Unlike the modalities, which lift unary predicates on terms, these are designed to lift relations on terms. We define an operator given by a family of maps $\mathcal{O}_{X,Y}(-) : \mathcal{P}(X \times Y) \to \mathcal{P}(T(X) \times T(Y))$, indexed by pairs of sets X and Y, such that:

$$t \; \mathcal{O}_{X,Y}(\mathcal{R}) \; r \quad \iff \quad \forall D \subseteq X, o \in \mathcal{O}, (t \in o(D) \Rightarrow r \in o(\mathcal{R}^{\uparrow}[D])).$$

Henceforth, we do not write the indices X and Y as subscripts for the $\mathcal{O}(-)$ operator.

Remember that $(\mathcal{R}^{\uparrow}[D]) := \{y \in Y \mid \exists x \in X, x \,\mathcal{R}\, y\}$ (introduced just before Lemma 3.3.19). We require this lifting of relations to satisfy the defining properties of relators from [46].

Definition 4.1.1. A family of maps $\Gamma(-) : \mathcal{P}(X \times Y) \to \mathcal{P}(T(X) \times T(Y))$ indexed by pairs of sets X, Y, is a *relator* (in the sense of [46]) if the following four properties hold.

1. For any set X, $=_{T(X)} \subseteq \Gamma(=_X)$.
2. $\forall \mathcal{R} \subseteq X \times Y, \forall \mathcal{S} \subseteq Y \times Z, \quad \Gamma(\mathcal{R})\Gamma(\mathcal{S}) \subseteq \Gamma(\mathcal{R}\mathcal{S})$.
3. $\forall \mathcal{R}, \mathcal{S} \subseteq X \times Y, \quad \mathcal{R} \subseteq \mathcal{S} \Rightarrow \Gamma(\mathcal{R}) \subseteq \Gamma(\mathcal{S})$.
4. $\forall f : X \to Z, g : Y \to W, \mathcal{R} \subseteq Z \times W, \; \Gamma((f \times g)^{-1}\mathcal{R}) = (T(f) \times T(g))^{-1}\Gamma(\mathcal{R})$.

As a direct consequence of point 1 and 3, it holds that for any reflexive relation $\mathcal{R} \subseteq X \times X$, $\Gamma(\mathcal{R})$ is a reflexive relation.

To prove that $\mathcal{O}(-)$ is a relator, we need the condition that all modalities are leaf-upwards closed, from Definition 3.3.12.

Lemma 4.1.2. *If the modalities from $\mathcal{O}$ are leaf-upwards closed, then $\mathcal{O}(-)$ is a relator.*

Proof. We prove each of the four properties individually.

1. For any set $D \subseteq X$ it holds that $D = (=_X{}^{\uparrow}[D])$. So for any $t \in T(X)$, if $t \in o(D)$ then $t \in o(=_X{}^{\uparrow}[D])$. We can conclude that for all $t \in T(X)$, $t \,\mathcal{O}(=_X)\, t$, and hence $=_{T(X)} \subseteq \mathcal{O}(=_X)$.

2. For any subset $D \subseteq X$, it holds that $((\mathcal{R}\mathcal{S})^{\uparrow}[D]) = \{z \in Z \mid \exists x \in D, x \,\mathcal{R}\mathcal{S}\, z\} = \{z \in Z \mid \exists x \in D, \exists y \in Y, x \,\mathcal{R}\, y \wedge y \,\mathcal{S}\, z\} = \{z \in Z \mid \exists y \in (\mathcal{R}^{\uparrow}[D]), y \,\mathcal{S}\, z\} = (\mathcal{S}^{\uparrow}[(\mathcal{R}^{\uparrow}[D])])$. So with $t \,\mathcal{O}(\mathcal{R})\, r \,\mathcal{O}(\mathcal{S})\, l$ and $o \in \mathcal{O}$ it holds that:

   $t \in o(D) \implies r \in o(\mathcal{R}^{\uparrow}[D]) \implies l \in o(\mathcal{S}^{\uparrow}[(\mathcal{R}^{\uparrow}[D])]) = o((\mathcal{R}\mathcal{S})^{\uparrow}[D])$.

3. If $\mathcal{R} \subseteq \mathcal{S}$, then for any set D it holds that $(\mathcal{R}^{\uparrow}[D]) \subseteq (\mathcal{S}^{\uparrow}[D])$. Assume $t \,\mathcal{O}(\mathcal{R})\, r$ and $t \in o(D)$, then $r \in o(\mathcal{R}^{\uparrow}[D])$. Hence by leaf-upwards closure of o, it holds that $r \in o(\mathcal{S}^{\uparrow}[D])$.

4. Note that $((f \times g)^{-1}\mathcal{R}^{\uparrow}[D]) = g^{-1}(\mathcal{R}^{\uparrow}[f(D)])$, where $f(D) = \{f(x) \mid x \in D\}$.

   If $t\,\mathcal{O}((f \times g)^{-1}\mathcal{R})\,r$ then for all $D \subseteq X$ and $o \in \mathcal{O}$ it holds that $t \in o(D) \Rightarrow r \in o((f \times g)^{-1}\mathcal{R}^{\uparrow}[D]) \Rightarrow r \in o(g^{-1}(\mathcal{R}^{\uparrow}[f(D)]))$. So, given a set $E \subseteq Z$ and a modality o such that $T(f)(t) \in o(E)$, then $t \in o(f^{-1}(E))$ so $r \in o(g^{-1}(\mathcal{R}^{\uparrow}[f(f^{-1}E)]))$. Since $f(f^{-1}E) \subseteq E$, and hence $g^{-1}(\mathcal{R}^{\uparrow}[f(f^{-1}E)]) \subseteq g^{-1}(\mathcal{R}^{\uparrow}[E])$, we use leaf-upwards closure of o to derive that $r \in o(g^{-1}(\mathcal{R}^{\uparrow}[E]))$. Hence $T(g)(r) \in o(\mathcal{R}^{\uparrow}[E])$ and we conclude that $T(f)(t)\,\mathcal{O}(\mathcal{R})\,T(g)(r)$.

   If $T(f)(t)\,\mathcal{O}(\mathcal{R})\,T(g)(r)$ and $t \in o(D)$, then since $T(f)(t) \in o(f(D))$ it holds that $T(g)(r) \in o(\mathcal{R}^{\uparrow}[f(D)])$ so $r \in o(g^{-1}(\mathcal{R}^{\uparrow}[f(D)]))$. This is for all such D and o, so we can conclude that $t\ \mathcal{O}((f \times g)^{-1}\mathcal{R})\ r$. □

We will call $\mathcal{O}(-)$ the *$\mathcal{O}$-relator*, since it is a relator specified by the set of modalities $\mathcal{O}$. The next property together with the previous lemma establishes that $\mathcal{O}(-)$ is a *monotone relator* in the sense of Thijs [102][1].

Lemma 4.1.3. *If the modalities from $\mathcal{O}$ are leaf-upwards closed, then for any two functions $f : X \to Z$, $g : Y \to W$, two trees $t \in T(X), r \in T(Y)$, and relations $\mathcal{R} \subseteq X \times Y$, $\mathcal{S} \subseteq Z \times W$:*

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

Proof. Let $\mathcal{R} \subseteq X \times Y$ and $\mathcal{S} \subseteq Z \times W$. Assume:

(I) $\forall x, y, x\,\mathcal{R}\,y \Rightarrow f(x)\,\mathcal{O}(\mathcal{S})\,g(y)$.

(II) $t\,\mathcal{O}(\mathcal{R})\,r$.

Take $o \in \mathcal{O}$ and $D \subseteq T(Z)$ such that $t[x \mapsto f(x)] \in o(D)$. Take $E := f^{-1}(D)$, then $t \in o(E)$. So by (II), $r \in o(\mathcal{R}^{\uparrow}[E])$. For $y \in (\mathcal{R}^{\uparrow}[E])$, there is an $x \in E$ such that $x\,\mathcal{R}\,y$, hence by (I) it holds that $f(x)\,\mathcal{S}\,g(y)$. Since $x \in E$ implies $f(x) \in D$, it holds that $g(y) \in (\mathcal{S}^{\uparrow}[D])$. It follows that $(\mathcal{R}^{\uparrow}[E]) \subseteq g^{-1}(\mathcal{S}^{\uparrow}[D])$. Using leaf-upwards closure, $r \in o(g^{-1}(\mathcal{S}^{\uparrow}[D]))$. Since this is for all $o \in \mathcal{O}$ and $D \subseteq T(Z)$ with $t[x \mapsto f(x)] \in o(D)$, we conclude that $t[x \mapsto f(x)]\,\mathcal{O}(\mathcal{S})\,r[y \mapsto g(y)]$. □

A relation $\mathcal{R}$ on $\mathit{Terms}(\mathbf{A})$ can be considered a relation on terminal computation terms $\mathit{Tct}(\mathbf{F}\,\mathbf{A})$. So we can consider its lifting $\mathcal{O}(\mathcal{R})$ to be a relation on $T(\mathit{Tct}(\mathbf{F}\,\mathbf{A}))$. This relator characterises the equivalence for $\mathbf{F}\,\mathbf{A}$-types.

Lemma 4.1.4. *If all modalities of $\mathcal{O}$ are leaf-upwards closed, then for $\sqsubseteq^{+}$, it holds that $\underline{M}\ \sqsubseteq^{+}_{\mathbf{F}\,\mathbf{A}}\ \underline{N} \iff |\underline{M}|\ \mathcal{O}(\sqsubseteq^{+}_{\mathbf{A}})\ |\underline{N}|$.*

Proof. Assume $\underline{M}\ \sqsubseteq^{+}_{\mathbf{F}\,\mathbf{A}}\ \underline{N}$, $o \in \mathcal{O}$ and $D \subseteq \mathit{Terms}(\mathbf{A})$ such that $|\underline{M}| \in o(D)$. Defining $\chi_D = \bigvee\{\chi_V \mid V \in D\}$ (with χ_V from Lemma 3.3.6) it holds that

$$W \models \chi_D \iff (\exists V \in D, V \sqsubseteq^{+} W) \iff (W \in (\sqsubseteq^{+\uparrow}[D])).$$

[1] [102] defines a monotonic relator as a family of relation lifting operations satisfying conditions 1, 2 and 3 from Definition 4.1.1, and satisfying the condition stated in Lemma 4.1.3.

By reflexivity of $\sqsubseteq^+$ it holds that $D \subseteq (\sqsubseteq^{+\uparrow}[D])$, hence if $|\underline{M}| \in o(D)$ then by leaf-upwards closure of o we get $|\underline{M}| \in o(\sqsubseteq^{+\uparrow}[D])$. Hence $\underline{M} \models o(\chi_D)$ from which we know that $\underline{N} \models o(\chi_D)$ and we can derive that $|\underline{N}| \in o(\sqsubseteq^{+\uparrow}[D])$. So we conclude that $|\underline{M}|\ \mathcal{O}(\sqsubseteq^+_{\mathbf{A}})\ |\underline{N}|$.

Assume $|\underline{M}|\ \mathcal{O}(\sqsubseteq^+_{\mathbf{A}})\ |\underline{N}|$ and $\underline{M} \models o(\phi)$, so $|\underline{N}| \in o(\sqsubseteq^{+\uparrow}[\phi])$. If $W \in (\sqsubseteq^{+\uparrow}[\phi])$ then there is a term V such that $V \models \phi$ and $V \sqsubseteq^+_{\mathbf{A}} W$, hence $W \models \phi$. So it holds that $W \in (\sqsubseteq^{+\uparrow}[\phi]) \implies W \models \phi$. Since o is leaf-upwards closed, we can conclude that $|\underline{N}|[\models \phi] \in [\![o]\!]$, so by Lemma 3.3.5 it holds that $\underline{M} \sqsubseteq^+_{\mathbf{F}\,\mathbf{A}} \underline{N}$. □

By a similar proof, using the symmetry of $\equiv$ we can prove the following.

Corollary 4.1.5. *If all modalities of $\mathcal{O}$ are leaf-upwards closed, then for $\equiv$, it holds that: $\underline{M} \equiv_{\mathbf{F}\,\mathbf{A}} \underline{N} \iff |\underline{M}|\ \mathcal{O}(\equiv_{\mathbf{A}})\ |\underline{N}| \wedge |\underline{N}|\ \mathcal{O}(\equiv_{\mathbf{A}})\ |\underline{M}|$.*

Proof. By the same proof as in Lemma 4.1.4, $\underline{M} \equiv_{\mathbf{F}\,\mathbf{A}} \underline{N} \Rightarrow |\underline{M}|\ \mathcal{O}(\equiv_{\mathbf{A}})\ |\underline{N}|$. Since $\underline{M} \equiv_{\mathbf{F}\,\mathbf{A}} \underline{N} \Rightarrow \underline{N} \equiv_{\mathbf{F}\,\mathbf{A}} \underline{M}$, it also hold that $\underline{M} \equiv_{\mathbf{F}\,\mathbf{A}} \underline{N} \Rightarrow |\underline{N}|\ \mathcal{O}(\equiv_{\mathbf{A}})\ |\underline{M}|$.

Given $|\underline{M}|\ \mathcal{O}(\equiv_{\mathbf{A}})\ |\underline{N}| \wedge |\underline{N}|\ \mathcal{O}(\equiv_{\mathbf{A}})\ |\underline{M}|$, it holds by the same proof as in Lemma 4.1.4, that $\underline{M} \models o(\phi)$ iff $\underline{N} \models o(\phi)$. Hence also, $\underline{M} \models \neg(o(\phi))$ iff $\underline{N} \models \neg(o(\phi))$, so by Lemma 3.3.5 it follows that $\underline{M} \equiv_{\mathbf{F}\,\mathbf{A}} \underline{N}$. □

4.1.1 Relators of the examples

We study what the relators look like for all of the given examples of effects with their modalities from Section 3.2.

Pure functional computation: (Subsection 3.2.1)
For $\mathcal{O}_\emptyset = \{\downarrow\}$, the statement $t\,\mathcal{O}_\emptyset(\mathcal{R})\,r$ holds if and only if:

- When t is equal to a leaf labelled $x \in X$, then r is equal to a leaf labelled $y \in Y$ such that $x\,\mathcal{R}\,y$.

Error: (Subsection 3.2.2)
For $\mathcal{O}_{\mathrm{er}} = \{\downarrow\} \cup \{\mathsf{E}_e \mid e \in \mathsf{Err}\}$, the statement $t\ \mathcal{O}_{\mathrm{er}}(\mathcal{R})\ r$ holds precisely when the following two statements hold:

- When t is equal to a leaf labelled $x \in X$, then r is equal to a leaf labelled $y \in Y$ such that $x\,\mathcal{R}\,y$.
- When t raises an error $e \in \mathsf{Err}$, then r raises the same error e.

Nondeterminism: (Subsection 3.2.3)
For $\mathcal{O}_{\mathrm{nd}} = \{\Diamond, \Box\}$, the statement $t\ \mathcal{O}_{\mathrm{nd}}(\mathcal{R})\ r$ holds precisely when the following two things are true:

1. If $x \in X$ is a leaf of t, then r has a leaf $y \in Y$ such that $x\,\mathcal{R}\,y$.
2. If t is finite and has no $\bot$-leaves, then:

- r is finite and has no $\bot$-leaves.
- If $y \in Y$ is a leaf of r then there is a leaf $x \in X$ of t such that $x \mathcal{R} y$.

The relator for angelic nondeterminism $\{\Diamond\}(\mathcal{R})$ is characterised by rule 1 only, whereas the relator for demonic nondeterminism $\{\Box\}(\mathcal{R})$ is characterised by rule 2.

Probabilistic choice: (Subsection 3.2.4)
For $\mathcal{O}_{\mathrm{pr}} = \{\mathsf{P}_{>q} \mid q \in \mathbb{Q}, 0 \leq q \leq 1\}$, the statement $t\ \mathcal{O}_{\mathrm{pr}}(\mathcal{R})\ r$ holds if and only if:

- For any $A \subseteq X$, the probability of t terminating with an element of A is less than or equal to the probability of r terminating with a y such that there is an $x \in A$ with $x \mathcal{R} y$. In other words, $\forall A \subseteq X.\ \mathbf{P}(t[\in A]) \leq \mathbf{P}(r[\in (\mathcal{R}^{\uparrow}[A])])$.

Global store: (Subsection 3.2.5)
For $\mathcal{O}_{\mathrm{gs}} = \{(s \rightarrowtail r) \mid s, r \in \mathit{State}\}$, the statement $t\ \mathcal{O}_{\mathrm{gs}}(\mathcal{R})\ r$ holds if:

- For any $s \in \mathsf{State}$, if $\mathit{exec}(t, s) = (x, s')$ then $\mathit{exec}(r, s) = (y, s')$ for some y such that $x \mathcal{R} y$.

Input/output: (Subsection 3.2.6)
For $\mathcal{O}_{\mathrm{io}} = \{\langle w\rangle\!\downarrow, \langle w\rangle_{\ldots} \mid w \text{ an i/o-trace}\}$, $t\ \mathcal{O}_{\mathrm{io}}(\mathcal{R})\ r$ holds precisely when the following two statements hold:

- If w is an initial i/o-trace for t, then w is an initial i/o-trace for r. Formally, $t \models \langle w\rangle_{\ldots} \implies r \models \langle w\rangle_{\ldots}$.
- If w is a complete i/o trace for t resulting in termination/leaf $x \in X$, the w is a complete i/o trace for r resulting in termination/leaf $y \in Y$ such that $x \mathcal{R} y$. Formally, $t \models \langle w\rangle\!\downarrow \{x\} \implies r \models \langle w\rangle\!\downarrow (\mathcal{R}^{\uparrow}[\{x\}])$.

Timer: (Subsection 3.2.7)
For $\mathcal{O}_{\mathrm{ti}} = \{\mathsf{C}_{\leq q}, \mathsf{C}_{\geq q}, \mathsf{C}^{\downarrow}_{>q} \mid q \in \mathbb{Q}\}$, then the statement $t\ \mathcal{O}_{\mathrm{ti}}(\mathcal{R})\ r$ holds if and only if:

1. If t evaluates to x in c-time, then r evaluates to a y within c-time s.t. $x \mathcal{R} y$.
2. If t evaluates to x in c-time, then r evaluates to a y in at least c-time s.t. $x \mathcal{R} y$.
3. If t delays computation for c-time, then r delays computation for at least c-time.

The relator for the down-interpretation of delays $\mathcal{O}^{\downarrow}_{\mathrm{ti}}(\mathcal{R})$ is characterised by rule 1 only, whereas the relator for the up-interpretation of delay $\mathcal{O}^{\uparrow}_{\mathrm{ti}}(\mathcal{R})$ is characterised by the combination of rules 2 and 3.

In [14], relators can be defined for monads which carry a continuous Σ-algebra, which the tree monad $T_{\Sigma}(-)$ does. The examples of relators in [14] for specific effects are, unlike here, defined on monads which are specifically designed for the effect in question. Those relators can however be compared to the relators given above. Let M

be a monad which carries a continuous Σ-algebra, then for each set X we can define a function $f : T_\Sigma(X) \to MX$ which respects the monadic and Σ-algebraic structure. For any relator Γ_M on M given as an example in [14], the relator Γ_T on T_Σ defined as, $\forall \mathcal{R} \subseteq X \times Y.\ t\ \Gamma_T(\mathcal{R})\ r \iff f_X(t)\ \Gamma_M(\mathcal{R})\ f_Y(r)$, coincides with the relator for the same effects as given above.

Take for instance the example of probability, which in [14] is described using the subdistribution functor $\mathcal{D}$ which sends a set X to the set of subdistributions $\mu : \mathcal{P}(X) \to [0, 1]$. The continuous Σ-algebra carried by the monad provides us with a function $f : T(X) \to \mathcal{D}X$, given by $f(t)(U) = \mathbf{P}(t[\in U])$. So with the relator $\Gamma_\mathcal{D}$ from [14] given by $\mu\ \Gamma_\mathcal{D}(\mathcal{R})\ \nu :\Leftrightarrow \forall U \subseteq X.\mu(U) \leq \mu(\mathcal{R}^\uparrow[U])$, we see that $f_X(t)\ \Gamma_M(\mathcal{R})\ f_Y(r)$ holds precisely if $t\ \mathcal{O}_{\mathrm{pr}}(\mathcal{R})\ r$.

We could potentially generalize the notion of modality to work for any monad M which carries a continuous Σ-algebra. However, in such a generalisation, any such modality γ on M denoted by a subset $[\![\gamma]\!] \subseteq M(\{*\})$ can be appropriately specified by a modality o_γ on T_Σ denoted by $[\![o_\gamma]\!] = f^{-1}([\![\gamma]\!]) \subseteq T_\Sigma(\{*\})$. As such, there is no real loss of generality when studying tree monads only. So we choose to capture behaviour of effectful programs with modalities on T_Σ. Such modalities implicitly carry the information of more effect-specific monads. See [54] for a setting in which more general classes of 'well-behaved' predicate liftings induce relators.

As an illustration of how we do not lose any generality when studying tree monads, we look at the example of global store, with a monad on the partial global store functor $MX := (\mathsf{State} \times X)^{\mathsf{State}}_\bot$. A modality γ on M would be denoted by a set $[\![\gamma]\!] \subseteq M\{*\} \simeq (\mathsf{State})^{\mathsf{State}}_\bot$, so a set of partial endofunctions on State. Generalising the Scott opennes property, this set $[\![\gamma]\!]$ will need to *open* with respect to the domain structure of $M\{*\}$ (where $g \leq h :\iff \forall s \in \mathsf{State}.g(s) \neq \bot \Rightarrow g(s) = h(s)$). The function $f : T(X) \to MX$ given by the continuous Σ-algebra carried by M can be defined as $f(t) = \lambda s.swap(exec(t, s))$ where $swap(x, s') = (s', x)$. Given that $[\![\gamma]\!]$ is open (hence upwards closed), the modality o_γ given by $[\![o_\gamma]\!] = f^{-1}([\![\gamma]\!])$ can be constructed in terms of the modalities $(s \rightarrowtail s')$ defined in this thesis:

$$[\![o_\gamma]\!] = f^{-1}([\![\gamma]\!]) = \bigvee_{g \in [\![\gamma]\!]} \bigwedge_{s \in \mathsf{State}, g(s) \neq \bot} [\![(s \rightarrowtail g(s))]\!] \ .$$

We conclude that the logic defined for the state monad M can be expressed within our logic defined for the tree monad.

4.2 Applicative simulations

Following [14], but adapted to our call-by-push-value typed setting, we use relators to define notions of applicative similarity and bisimilarity.

Definition 4.2.1. A well-typed relation $\mathcal{R}$ on closed terms is an *applicative $\mathcal{O}$-simulation* if:

1. $V\ \mathcal{R}_\mathbf{N}\ W \quad \Longrightarrow \quad V = W$.

2. $\mathsf{thunk}(\underline{M})\,\mathcal{R}_{\mathbf{U}\,\underline{\mathbf{C}}}\,\mathsf{thunk}(\underline{N}) \quad \Longrightarrow \quad \underline{M}\,\mathcal{R}_{\underline{\mathbf{C}}}\,\underline{N}$.
3. $(j, V)\,\mathcal{R}_{\mathbf{\Sigma}_{i\in I}\,\mathbf{A}_i}\,(k, W) \quad \Longrightarrow \quad (j = k)\ \wedge\ V\,\mathcal{R}_{\mathbf{A}_j}\,W$.
4. $(V, V')\,\mathcal{R}_{\mathbf{A}\times\mathbf{B}}\,(W, W') \quad \Longrightarrow \quad V\,\mathcal{R}_{\mathbf{A}}\,W\ \wedge\ V'\,\mathcal{R}_{\mathbf{B}}\,W'$.
5. $\underline{M}\,\mathcal{R}_{\mathbf{A}\to\underline{\mathbf{C}}}\,\underline{N} \quad \Longrightarrow \quad \forall V \in \mathit{Terms}(\mathbf{A}),\ \underline{M}\ V\,\mathcal{R}_{\underline{\mathbf{C}}}\,\underline{N}\ V$.
6. $\underline{M}\,\mathcal{R}_{\mathbf{F}\,\mathbf{A}}\,\underline{N} \quad \Longrightarrow \quad |\underline{M}|\ \mathcal{O}(\mathcal{R}_{\mathbf{A}})\ |\underline{N}|$
7. $\underline{M}\,\mathcal{R}_{\mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}}_i}\,\underline{N} \quad \Longrightarrow \quad \forall j \in I,\ \underline{M}\ j\,\mathcal{R}_{\underline{\mathbf{C}}_j}\,\underline{N}\ j$.

The *applicative $\mathcal{O}$-similarity* is the largest $\mathcal{O}$-simulation.

Applicative $\mathcal{O}$-similarity exists when all modalities of $\mathcal{O}$ are leaf-upwards closed, since the union over all $\mathcal{O}$-simulations is also an $\mathcal{O}$-simulation. We will prove in Theorem 4.2.7 that the positive behavioural preorder $\sqsubseteq^+$ is applicative $\mathcal{O}$-similarity, which gives an alternative proof for the existence of applicative $\mathcal{O}$-similarity. We define *mutual applicative $\mathcal{O}$-similarity* as the largest symmetric subset of applicative $\mathcal{O}$-similarity, which is equal to $\equiv^+$ given Theorem 4.2.7. Lastly, we define applicative $\mathcal{O}$-bisimilarity, which by Theorem 4.2.8 coincides with the full behavioural equivalence.

Definition 4.2.2. A well-typed relation $\mathcal{R}$ on closed terms is an *applicative $\mathcal{O}$-bisimulation* if it is a symmetric applicative $\mathcal{O}$-simulation. The *applicative $\mathcal{O}$-bisimilarity* is the largest applicative $\mathcal{O}$-bisimulation.

Sometimes a more general notion of applicative bisimulation is used in the literature.

Definition 4.2.3. A well-typed relation $\mathcal{R}$ on closed terms is a *generalised applicative $\mathcal{O}$-bisimulation* if both $\mathcal{R}$ and $\mathcal{R}^{op}$ are applicative $\mathcal{O}$-simulations.

If $\mathcal{O}(-)$ is a relator, and hence has the relator property that for any relations $\mathcal{R} \subseteq \mathcal{S}$, $\mathcal{O}(\mathcal{R}) \subseteq \mathcal{O}(\mathcal{S})$, we have the following relatively simple result (see [15] for similar observations in the context of probabilistic applicative bisimulations):

Lemma 4.2.4. *If $\mathcal{O}(-)$ is a relator, then applicative $\mathcal{O}$-bisimilarity is the largest generalised applicative $\mathcal{O}$-bisimulation.*

Proof. Note that any applicative $\mathcal{O}$-bisimulation $\mathcal{R}$ is a generalised applicative $\mathcal{O}$-bisimulation, since $\mathcal{R}^{op} = \mathcal{R}$.

Given a generalised applicative $\mathcal{O}$-bisimulation $\mathcal{R}$. Then $\mathcal{R} \cup \mathcal{R}^{op}$ is an applicative $\mathcal{O}$-bisimulation. □

This allows for a more general proof technique for establishing that two terms are related via the bisimilarity.

We will now establish the connection between our positive behavioural preorder and applicative $\mathcal{O}$-similarity.

Lemma 4.2.5. *If all modalities are leaf-upwards closed, then the positive behavioural preorder $\sqsubseteq^+$ is an applicative $\mathcal{O}$-simulation.*

Proof. This is a consequence of Lemma 3.4.3, Corollary 3.4.2, and Lemma 4.1.4. □

Lemma 4.2.6. *If all modalities are leaf-upwards closed, then any applicative $\mathcal{O}$-simulation is a subset of $\sqsubseteq^+$.*

Proof. For $\mathcal{R}$ an $\mathcal{O}$-simulation, we prove that $\mathcal{R} \subseteq (\sqsubseteq^+)$. Specifically, we will prove that $\mathcal{R}$ preserves all formulas $\underline{\phi}$, meaning: If $\underline{\mathbf{E}}$ is the type of $\underline{\phi}$, then for any $\underline{P}, \underline{R} \in \mathit{Terms}(\underline{\mathbf{E}})$, if $\underline{P}\, \mathcal{R}_{\underline{\mathbf{E}}}\, \underline{R}$ and $\underline{P} \models \underline{\phi}$ then $\underline{R} \models \underline{\phi}$.

We prove this by an induction on $\underline{\phi}$, which can be done because all formulas are well-founded. Hence, for any formula $\underline{\phi}$, we need to prove that $\mathcal{R}$ preserves $\underline{\phi}$, given that it preserves any subformula[2] of $\underline{\phi}$. We perform the induction by case of $\underline{\phi}$.

1. If $\underline{\phi} = \{n\}$, then $\underline{\phi} \in \mathit{Form}(\mathbf{N})$. Assume $V\, \mathcal{R}_{\mathbf{N}}\, W$ and $V \models \underline{\phi}$, then by simulation rule (1), $V = W$, from which it is apparent that $W \models \underline{\phi}$. Hence we can conclude that $\mathcal{R}$ preserves $\underline{\phi}$.

2. If $\underline{\phi} = \langle \underline{\phi} \rangle$, then $\underline{\phi} \in \mathit{Form}(\mathbf{U}\, \underline{\mathbf{C}})$. Assume $\mathsf{thunk}(\underline{M})\, \mathcal{R}_{\mathbf{U}\, \underline{\mathbf{C}}}\, \mathsf{thunk}(\underline{N})$ and $\mathsf{thunk}(\underline{M}) \models \langle \underline{\phi} \rangle$ hence $\mathsf{force}(\mathsf{thunk}(\underline{M})) \models \underline{\phi}$, and by Corollary 3.4.2, $\underline{M} \models \underline{\phi}$. By simulation rule (2), $\underline{M}\, \mathcal{R}_{\underline{\mathbf{C}}}\, \underline{N}$. So by induction hypothesis $\underline{N} \models \underline{\phi}$ and we conclude (by Corollary 3.4.2) that $\mathsf{thunk}(\underline{N}) \models \langle \underline{\phi} \rangle$.

3. If $\underline{\phi} = (j, \psi)$, then $\underline{\phi} \in \mathit{Form}(\mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i)$. Assume $(a, V)\, \mathcal{R}_{\mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i}\, (b, W)$ and $(a, V) \models \underline{\phi}$, then $a = j$ and $V \models \psi$. By simulation rule (3), $b = a = j$ and $V\, \mathcal{R}_{\mathbf{A}_j}\, W$, hence by induction hypothesis $W \models \psi$. We conclude that $(b, W) \models (j, \psi)$.

4. If $\underline{\phi} = \pi_0(\psi)$, then $\underline{\phi} \in \mathit{Form}(\mathbf{A} \times \mathbf{B})$. Assume $(V, V')\, \mathcal{R}_{\mathbf{A} \times \mathbf{B}}\, (W, W')$ and $(V, V') \models \underline{\phi}$, hence $V \models \psi$. By simulation rule (4), $V\, \mathcal{R}_{\mathbf{A}}\, W$, so by the induction hypothesis $W \models \psi$. We conclude that $(W, W') \models \pi_0(\psi)$.

   The case where $\underline{\phi} = \pi_1(\psi)$ goes similarly.

5. If $\underline{\phi} = (V \mapsto \underline{\phi})$, then $\underline{\phi} \in \mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})$. Assume $\underline{M}\, \mathcal{R}_{\mathbf{A} \to \underline{\mathbf{C}}}\, \underline{N}$ and $\underline{M} \models \underline{\phi}$, hence $\underline{M}\, V \models \underline{\phi}$. By simulation rule (5), $\underline{M}\, V\, \mathcal{R}_{\underline{\mathbf{C}}}\, \underline{N}\, V$, so by induction hypothesis $\underline{N}\, V \models \underline{\phi}$. We can conclude that $\underline{N} \models (V \mapsto \underline{\phi})$.

6. If $\underline{\phi} = o(\psi)$, then $\underline{\phi} \in \mathit{Form}(\mathbf{F}\, \mathbf{A})$. Assume $\underline{M}\, \mathcal{R}_{\mathbf{F}\, \mathbf{A}}\, \underline{N}$ and $\underline{M} \models o(\psi)$, hence $|\underline{M}[\models \psi] \in [\![o]\!]$. By simulation rule (6), it holds that $|\underline{M}|\ \mathcal{O}(\mathcal{R}_{\mathbf{A}})\ |\underline{N}|$, so $|\underline{N}|[\in (\mathcal{R}_{\mathbf{A}}{}^{\uparrow}[\{V \mid V \models \psi\}])] \in [\![o]\!]$. By induction hypothesis, $\mathcal{R}$ preserves ψ, hence for each $W \in (\mathcal{R}_{\mathbf{A}}{}^{\uparrow}[\{V \mid V \models \psi\}])$, since there is a V such that $V \models \psi$ and $V\, \mathcal{R}_{\mathbf{A}}\, W$, it holds that $W \models \psi$. So $(\mathcal{R}_{\mathbf{A}}{}^{\uparrow}[\{V \mid V \models \psi\}]) \subseteq \{V \mid V \models \psi\}$, and hence by leaf-upwards closure of o it holds that $|\underline{N}|[\models \psi] \in [\![o]\!]$. We can conclude that $\underline{N} \models o(\psi)$.

7. If $\underline{\phi} = (j \mapsto \underline{\phi})$, then $\underline{\phi} \in \mathit{Form}(\mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i)$. Assume $\underline{M}\, \mathcal{R}_{\mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i}\, \underline{N}$ and $\underline{M} \models \underline{\phi}$, hence $\underline{M}\, j \models \underline{\phi}$. By simulation rule (7), $\underline{M}\, j\, \mathcal{R}_{\underline{\mathbf{C}}_j}\, \underline{N}\, j$, so by induction hypothesis $\underline{N}\, j \models \underline{\phi}$. We can conclude that $\underline{N} \models (j \mapsto \underline{\phi})$.

[2] With subformula of $\underline{\phi}$, we mean any formula used in the construction of $\underline{\phi}$.

8. If $\underline{\underline{\phi}} = \bigvee_{i\in I} \underline{\underline{\psi}}_i$, then $\underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$ (it can be of any type). Assume $\underline{\underline{P}}\ \mathcal{R}_{\underline{\underline{\mathbf{E}}}}\ \underline{\underline{R}}$ and $\underline{\underline{P}} \models \underline{\underline{\phi}}$, hence there is an $i \in I$ such that $\underline{\underline{P}} \models \underline{\underline{\psi}}_i$. By induction hypothesis, $\underline{\underline{R}} \models \underline{\underline{\psi}}_i$, so we can conclude that $\underline{\underline{R}} \models \bigvee_{i\in I} \underline{\underline{\psi}}_i$.

9. If $\underline{\underline{\phi}} = \bigwedge_{i\in I} \underline{\underline{\psi}}_i$, then $\underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$. Assume $\underline{\underline{P}}\ \mathcal{R}_{\underline{\underline{\mathbf{E}}}}\ \underline{\underline{R}}$ and $\underline{\underline{P}} \models \underline{\underline{\phi}}$, hence for any $i \in I$ it holds that $\underline{\underline{P}} \models \underline{\underline{\psi}}_i$. By induction hypothesis, $\underline{\underline{R}} \models \underline{\underline{\psi}}_i$ for all $i \in I$, so we can conclude that $\underline{\underline{R}} \models \bigwedge_{i\in I} \underline{\underline{\psi}}_i$.

This finishes the induction on formulas, so by the principle of well-founded induction, it holds that $\mathcal{R}$ preserves all formulas. We can conclude that $\mathcal{R} \subseteq (\underline{\underline{\sqsubseteq}}^+)$. □

This proof is slightly different from the proof in [95], which this chapter is based on, which uses types as the basis for the induction. The proof is changed in this dissertation, to accommodate situations where an induction on types is impossible (e.g., polymorphic and recursive types). This accommodation will be needed in Chapter 7.

Note that the previous lemma provides us with a proof technique for proving that two terms are related via the positive behavioural preorder, by relating them via some simulation. We can conclude from Lemmas 4.2.5 and 4.2.6 that $\underline{\underline{\sqsubseteq}}^+$ is the largest applicative $\mathcal{O}$-simulation, hence we can conclude that the following theorem holds:

Theorem 4.2.7 (The Coincidence Theorem I)**.** *If all modalities are leaf-upwards closed, then the positive behavioural preorder $\underline{\underline{\sqsubseteq}}^+$ is applicative $\mathcal{O}$-similarity.*

Adapting the proofs of Lemmas 4.2.5 and 4.2.6 slightly, we also get the following:

Theorem 4.2.8 (The Coincidence Theorem II)**.** *If all modalities are leaf-upwards closed, then the full behavioural equivalence $\equiv$ is applicative $\mathcal{O}$-bisimilarity.*

Proof. By Lemma 3.4.3 and Corollary 4.1.5 it holds that $\equiv$ is an $\mathcal{O}$-simulation, and obviously $\equiv$ is symmetric, so it is an $\mathcal{O}$-bisimulation.

Now, let $\mathcal{R}$ be an $\mathcal{O}$-bisimulation. We prove that it preserves all formulas $\underline{\underline{\phi}} \in \mathcal{V}$ by an induction on formulas. Since $\mathcal{R}$ is a simulation, we can copy the proof of Lemma 4.2.6 containing all the cases for $\underline{\underline{\phi}}$ except $\underline{\underline{\phi}} = \neg(\underline{\underline{\psi}})$. We need only add this negation case.

10. If $\underline{\underline{\phi}} = \neg(\underline{\underline{\psi}})$, then $\underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$. Assume $\underline{\underline{P}}\ \mathcal{R}_{\underline{\underline{\mathbf{E}}}}\ \underline{\underline{R}}$ and $\underline{\underline{P}} \models \underline{\underline{\phi}}$. Since $\mathcal{R}$ is symmetric, $\underline{\underline{R}}\ \mathcal{R}_{\underline{\underline{\mathbf{E}}}}\ \underline{\underline{P}}$, so if $\underline{\underline{R}} \models \underline{\underline{\psi}}$ then by induction hypothesis $\underline{\underline{P}} \models \underline{\underline{\psi}}$. However, $\underline{\underline{P}} \models \neg(\underline{\underline{\psi}})$ so we have a contradiction. We conclude that $\underline{\underline{R}} \models \neg(\underline{\underline{\psi}})$.

This concludes the induction, so $\mathcal{R}$ preserves all formulas from $\mathcal{V}$. So $\equiv$ is an $\mathcal{O}$-bisimulation containing all $\mathcal{O}$-bisimulations, hence it is the largest $\mathcal{O}$-bisimulation. □

The connection to applicative simulations will allow us to prove that the open extensions of the behavioural equivalences are compatible: Theorems 4.5.2 and 4.5.5. The rest of this chapter is devoted to the proofs of those theorems.

4.3 Relator properties

In this section, we look at other properties of relators needed to establish compatibility of the open extensions of applicative $\mathcal{O}$-similarity and applicative $\mathcal{O}$-bisimilarity. These properties were identified in [14], and require that we have a decomposable set of Scott open modalities.

The next lemma states that the relator interacts well with the structure of the monad given by the triple $(T(-), \eta, \mu)$.

Lemma 4.3.1. *If $\mathcal{O}$ is a decomposable set of leaf-upwards closed modalities, then:*

1. *For all $\mathcal{R} \subseteq X \times Y$, $x \in X$ and $y \in Y$, $x\,\mathcal{R}\,y \;\Rightarrow\; \langle x\rangle\,\mathcal{O}(\mathcal{R})\,\langle y\rangle$.*
2. *For $\mathcal{R} \subseteq X \times Y$, $t \in T(T(X))$ and $r \in T(T(Y))$, $t\,\mathcal{O}(\,\mathcal{O}(\mathcal{R})\,)\,r \;\Rightarrow\; \mu t\,\mathcal{O}(\mathcal{R})\,\mu r$.*

Proof. We prove the properties in sequence.

1. Let $D \subseteq X$ and $\langle x\rangle \in o(D)$, then either: $x \in D$ and $\langle *\rangle \in [\![o]\!]$, or $\bot \in [\![o]\!]$.

   By leaf-upwards closure of o, if $\bot \in [\![o]\!]$ then $\langle *\rangle \in [\![o]\!]$ and hence, since the tree $\langle y\rangle[\in (\mathcal{R}^{\uparrow}[D])]$ either is equal to $\bot$ or $\langle *\rangle$, $\langle y\rangle \in o((\mathcal{R}^{\uparrow}[D]))$.

   If $\langle *\rangle \in [\![o]\!]$ and $x \in D$, then $y \in (\mathcal{R}^{\uparrow}[D])$. So $\langle y\rangle[\in (\mathcal{R}^{\uparrow}[D])] = \langle *\rangle \in [\![o]\!]$.

2. Assume:

   $$\text{(I)}\quad t\,\mathcal{O}(\,\mathcal{O}(\mathcal{R})\,)\,r \text{ and } \mu t \in o(E), \text{ where } o \in \mathcal{O} \text{ and } E \subseteq X.$$

   Take $D \subseteq T(\mathbf{1})$ and $\gamma \in \mathcal{O}$ such that $t[a \mapsto a[\in E]] \in \gamma(D)$. We define $S := \{a \in TX \mid a[\in E] \in D\}$, so it holds that $t \in \gamma(S)$. By $t\,\mathcal{O}(\,\mathcal{O}(\mathcal{R})\,)\,r$ we get $r \in \gamma(\,\mathcal{O}(\mathcal{R})^{\uparrow}[S])$. We prove that for $b \in (\,\mathcal{O}(\mathcal{R})^{\uparrow}[S])$, $b[\in (\mathcal{R}^{\uparrow}[E])] \in (\preccurlyeq^{\uparrow}[D])$.

   Suppose $b \in (\,\mathcal{O}(\mathcal{R})^{\uparrow}[S])$, then there is an $a \in S$ such that $a[\in E] \in D$ and $a\,\mathcal{O}(\mathcal{R})\,b$. If $a[\in E] \in [\![\delta]\!]$ for some $\delta \in \mathcal{O}$, then $b[\in (\mathcal{R}^{\uparrow}[E])] \in [\![\delta]\!]$. If $a[\in E][\in \emptyset] = a[\in \emptyset] \in [\![\delta]\!]$ for some $\delta \in \mathcal{O}$ it holds that $b[\in (\mathcal{R}^{\uparrow}[E])][\in \emptyset] = b[\in \emptyset] = b[\in (\mathcal{R}^{\uparrow}[\emptyset])] \in [\![\delta]\!]$. So $a[\in E] \preccurlyeq b[\in (\mathcal{R}^{\uparrow}[E])]$, and since $a[\in E] \in D$ it holds that $b[\in (\mathcal{R}^{\uparrow}[E])] \in (\preccurlyeq^{\uparrow}[D])$. It follows that:

   $$(\,\mathcal{O}(\mathcal{R})^{\uparrow}[S]) \quad\subseteq\quad \{b \in T(Y) \mid b[\in (\mathcal{R}^{\uparrow}[E])] \in (\preccurlyeq^{\uparrow}[D])\}$$

   By leaf-upwards closure of γ and since $r \in \gamma(\,\mathcal{O}(\mathcal{R})^{\uparrow}[S])$, it holds that $r \in \gamma(\{b \mid b[\in (\mathcal{R}^{\uparrow}[E])] \in (\preccurlyeq^{\uparrow}[D])\})$ and hence $r[b \mapsto b[\in (\mathcal{R}^{\uparrow}[E])]] \in \gamma(\preccurlyeq^{\uparrow}[D])$. So we have derived that for all γ and D:

   $$t[a \mapsto a[\in E]] \in \gamma(D) \quad\Longrightarrow\quad r[b \mapsto b[\in (\mathcal{R}^{\uparrow}[E])]] \in \gamma(\preccurlyeq^{\uparrow}[D]).$$

   By decomposability, $(\mu t)[\in E] = \mu(t[a \mapsto a[\in E]]) \preccurlyeq \mu(r[b \mapsto b[\in (\mathcal{R}^{\uparrow}[E])]]) = (\mu r)[\in (\mathcal{R}^{\uparrow}[E])]$. With $(\mu t) \in o(E)$ from our starting assumption (I), we get $(\mu r) \in o(\mathcal{R}^{\uparrow}[E])$. So we conclude that $\mu t\,\mathcal{O}(\mathcal{R})\,\mu r$.

□

The following properties show that the relator behaves well with respect to the domain structure on trees. These properties will be used in our application of Howe's method, particularly in Lemma 4.4.17, whose proof contains an induction on the operational semantics.

Lemma 4.3.2. *If $\mathcal{O}$ only contains Scott open modalities, then:*

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

Proof.

1. If $\mathcal{R}$ is reflexive then $\mathcal{O}(\mathcal{R})$ is reflexive by Lemma 4.1.2. Now for $t \leq r$, it holds that $t[\in D] \leq r[\in D]$. Hence if $t \in o(D)$, by upwards closure of o, $r \in o(D)$. By reflexivity, $r\ \mathcal{O}(\mathcal{R})\ r$, so $r \in o(\mathcal{R}^{\uparrow}[D])$.
2. Suppose that for all $n \in \mathbb{N}$, $u_n\,\mathcal{O}(\mathcal{R})\,v_n$. Take $D \subseteq X$ and $o \in \mathcal{O}$ such that $(\sqcup_n u_n) \in o(D)$. Now $(\sqcup_n u_n)[\in D] = \sqcup_n(u_n[\in D])$, so by Scott openness there is an $m \in \mathbb{N}$ such that $u_m \in o(D)$. Using the assumption that $u_m\,\mathcal{O}(\mathcal{R})\,v_m$, we derive $v_m \in o(\mathcal{R}^{\uparrow}[D])$. Since $v_m \leq \sqcup_n v_n$, $v_m[\in (\mathcal{R}^{\uparrow}[D])] \leq (\sqcup_n v_n)[\in (\mathcal{R}^{\uparrow}[D])]$, hence by upwards closure of o, we conclude that $(\sqcup_n v_n) \in o(\mathcal{R}^{\uparrow}[D])$.

□

The lemmas above list the properties of the relator sufficient for establishing compatibility, which are satisfied when our set $\mathcal{O}$ is a decomposable set of Scott open modalities. The results below follow from those above, specifically the next result combines Lemma 4.3.1 with the fact that $\mathcal{O}(\mathrm{id}_{\mathbf{1}})$ is actually the same relation as $\preccurlyeq$, and $\mathcal{O}(\preccurlyeq)$ is the same relation as $\preccurlyeq$.

Corollary 4.3.3. *If $\mathcal{O}$ contains only leaf-upwards closed modalities, then $\mathcal{O}$ is decomposable iff $\forall \mathcal{R} \subseteq X \times Y, \forall t, r \in T(T(\mathbf{1})),\ t\,\mathcal{O}(\,\mathcal{O}(\mathcal{R})\,)\,r \Rightarrow \mu t\ \mathcal{O}(\mathcal{R})\ \mu r$.*

This provides yet another characterisation of decomposability, complementing Lemma 3.3.22 and Proposition 3.3.23.

Lastly, we verify preservation over sequencing and algebraic effect operators.

Corollary 4.3.4. *If $\mathcal{O}$ is a decomposable set of leaf-upwards closed modalities, then:*

1. *Given $f : X \to Z$, $g : Y \to W$, $\mathcal{R} \subseteq X \times Y$ and $\mathcal{S} \subseteq Z \times W$, if for all $x \in X$ and $y \in Y$ it holds that $x\,\mathcal{R}\,y \Rightarrow f(x)\ \mathcal{O}(\mathcal{S})\ g(y)$ then $\forall t \in T(T(X)), r \in T(T(Y))$:*

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

2. *For an effect operator* $\mathsf{op} : \mathbf{N}^n \times \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$ *or* $\mathsf{op} : \mathbf{N}^n \times \underline{\alpha}^m \to \underline{\alpha}$, *and* $u_i \mathcal{S} v_i$ *for all* $i \in I$, *where* $I = \mathbb{N}$ *or* $I = \{0, \dots, m-1\}$ *respectively, it holds that:* $\mathsf{op}_{l_1,\dots,l_n}(u_0, u_1, \dots)\, \mathcal{O}(\mathcal{S})\, \mathsf{op}_{l_1,\dots,l_n}(v_0, v_1, \dots)$.

Proof. We prove the properties in order.

1. Using Lemma 4.1.3 on the assumptions we get $t[x \mapsto f(x)]\ \mathcal{O}(\,\mathcal{O}(\mathcal{S})\,)\ r[y \mapsto g(y)]$. We can then apply property 2 of Lemma 4.3.1 to get the correct result.

2. We apply the previous property to the following data; $t = r = \mathsf{op}(0, 1, 2, \dots) \in T(\mathbb{N})$, $f(n) = u_n$, $g(n) = v_n$ and $\mathcal{R} = id_{\mathbb{N}}$. The conclusion follows directly.

□

Point 2 of Corollary 4.3.4 has been stated in such a way that it contains both the infinite arity case $\mathbf{N}^n \times \alpha^{\mathbf{N}} \to \alpha$ and the finite arity case $\mathbf{N}^n \times \alpha^m \to \alpha$ for algebraic effect operators. So it states that any lifted relation is preserved under any of the algebraic effect operators.

4.3.1 Relator observations

We finish this section with some minor results about our particular construction of relators. These are not needed in Section 4.4 for the proof of the Compatibility theorem. However, they do further our understanding of $\mathcal{O}$-relators. First we show an equivalent definition of $\mathcal{O}$-relator.

Lemma 4.3.5. *If all modalities of* $\mathcal{O}$ *are leaf-upwards closed, then* $t\, \mathcal{O}(\mathcal{R})\, r$ *holds if and only if:*

$$\forall A \subseteq X, B \subseteq Y, \quad (\forall x \in X, y \in Y, (x \mathcal{R} y \wedge x \in A) \Rightarrow y \in B) \quad \Rightarrow \quad t[\in A] \preccurlyeq r[\in B]$$

Proof. This holds since if $\forall x \in X, y \in Y, (x \mathcal{R} y \wedge x \in A) \Rightarrow y \in B$, then $(\mathcal{R}^{\uparrow}[A]) \subseteq B$. □

The following lemma holds since $\mathcal{O}(\mathcal{R})$ is defined using a universal quantification over the modalities of $\mathcal{O}$.

Lemma 4.3.6. *Let* $\mathcal{O}$ *and* $\mathcal{O}'$ *be two sets of modalities for the same signature* Σ, *then* $\mathcal{O}(\mathcal{R}) \cap \mathcal{O}'(\mathcal{R}) = (\mathcal{O} \cup \mathcal{O}')(\mathcal{R})$.

We relate the notion of relator to the notion of behavioural property.

Definition 4.3.7. Suppose $\mathcal{R} \subseteq X \times X$. A subset $D \subseteq X$ is called $\mathcal{R}$-*closed*, if:

$$\forall x, y \in X, \quad x \mathcal{R} y \wedge x \in D \quad \Rightarrow \quad y \in D.$$

Note that if $\mathcal{R}$ is an equivalence relation, then any $\mathcal{R}$-closed set is a union of equivalence classes of $\mathcal{R}$.

The notion of $\mathcal{R}$-closedness generalises the notion of behavioural property, since the $\equiv_{\underset{\sim}{\mathbf{E}}}$-closed sets are given precisely by formulas $\underset{\sim}{\phi} \in \mathit{Form}(\underset{\sim}{\mathbf{E}})$ in the following sense:

Lemma 4.3.8. *Suppose $D \subseteq Terms(\underline{\mathbf{E}})$, then D is $\equiv_{\underline{\mathbf{E}}}$-closed if and only if there is a formula $\underline{\phi} \in Form(\underline{\mathbf{E}})$ such that $D = \{\underline{P} \in Terms(\underline{\mathbf{E}}) \mid \underline{P} \models \underline{\phi}\}$.*

Proof. If D is a $\equiv_{\underline{\mathbf{E}}}$-closed set, then it is characterised by $\bigvee\{\chi_{\underline{P}} \mid \underline{P} \in D\}$, where $\chi_{\underline{P}}$ is as in Lemma 3.3.6.

If $D = \{\underline{P} \in Terms(\underline{\mathbf{E}}) \mid \underline{P} \models \underline{\phi}\}$, then for $\underline{P} \in D$, $\underline{R} \in Terms(\underline{\mathbf{E}})$, if $\underline{P} \equiv_{\underline{\mathbf{E}}} \underline{R}$ then since $\underline{P} \models \phi$ it holds that $\underline{R} \models \phi$. So $\underline{R} \in D$ and we conclude that D is $\equiv_{\underline{\mathbf{E}}}$-closed. □

We can make two observations about $\mathcal{R}$-closed sets in general:

- If $\mathcal{R} \subseteq X \times X$ is transitive, then for any $D \subseteq X$, $(\mathcal{R}^{\uparrow}[D])$ is $\mathcal{R}$-closed.
- If $\mathcal{R} \subseteq X \times X$ is reflexive, then for any $\mathcal{R}$-closed set $D \subseteq X$, $(\mathcal{R}^{\uparrow}[D]) = D$.

These observations can be used to give a last characterisation of the $\mathcal{O}$-relator, in the case that it is acting on endorelations.

Lemma 4.3.9. *If $\mathcal{R} \subseteq X \times X$ is a preorder and all $o \in \mathcal{O}$ are leaf-upwards closed, then for any $t, r \in T(X)$:*

$$t\,\mathcal{O}(\mathcal{R})\,r \quad \iff \quad \forall \mathcal{R}\text{-closed } D \subseteq X, \forall o \in \mathcal{O}, \quad (t \in o(D) \Rightarrow r \in o(D)).$$

Proof. ($\Rightarrow$) Assume $t \in o(D)$ for D an $\mathcal{R}$-closed set, then $r \in o(\mathcal{R}^{\uparrow}[D])$, and since $(\mathcal{R}^{\uparrow}[D]) = D$ we conclude that $r \in o(D)$.

($\Leftarrow$) Assume $t \in o(D)$ for some $D \subseteq X$. Since $\mathcal{R}$ is reflexive, $D \subseteq (\mathcal{R}^{\uparrow}[D])$, and since o is leaf-upwards closed, $t \in o(\mathcal{R}^{\uparrow}[D])$. Since $(\mathcal{R}^{\uparrow}[D])$ is $\mathcal{R}$-closed, we can conclude that $r \in o(\mathcal{R}^{\uparrow}[D])$. □

4.4 Howe's method

In this section, we use Howe's method, first developed in [30, 31], to establish that the open extension of the applicative similarity and bisimilarity are compatibility, in Theorems 4.5.2 and 4.5.5.

Given a well-typed relation $\mathcal{R}$ on closed terms, we define its *Howe closure* $\mathcal{R}^{\bullet}$, which is compatible and contains the open extension $\mathcal{R}^{\circ}$ (Definition 3.3.7). Our proof makes fundamental use of the relator properties from Sections 4.1 and 4.3, closely following the approach of [14].

We will go into the proof of the Compatibility Theorem, using Howe's method, in more detail. Remember the definitions of open extension $\mathcal{R}^{\circ}$ (Definition 3.3.7) and compatible refinement $\widehat{\mathcal{R}}$ (Subsection 3.3.2). For any well-typed open relation $\mathcal{R}$, we define the *closed limitation* $\mathcal{R}^{\vdash}$ as the relation of $\mathcal{R}$ limited to closed terms. So in particular, we observe that for any relation $\mathcal{R}$ on closed terms, $\mathcal{R}^{\circ\vdash} = \mathcal{R}$.

Definition 4.4.1. For a closed relation $\mathcal{R}$ we define the *Howe closure* $\mathcal{R}^{\bullet}$ as the smallest open relation $\mathcal{S}$ closed under the rule:

$$\frac{\Gamma \vdash \underline{P}\ \widehat{\mathcal{S}}\ \underline{R} \qquad \Gamma \vdash \underline{R}\ \mathcal{R}^{\circ}\ \underline{Q}}{\Gamma \vdash \underline{P}\ \mathcal{S}\ \underline{Q}}\ \textbf{(H)}$$

In other words, the Howe closure is the least solution for $\mathcal{S}$ to the inclusion $\widehat{\mathcal{S}}\,\mathcal{R}^\circ \subseteq \mathcal{S}$. Moreover, it is also the least solution for $\mathcal{S}$ to the equation $\mathcal{S} = \widehat{\mathcal{S}}\,\mathcal{R}^\circ$. In particular, it holds that $\mathcal{R}^\bullet = \widehat{\mathcal{R}^\bullet}\mathcal{R}^\circ$. We look at some preliminary results, mostly from Lassen [41]:

Lemma 4.4.2. *If $\mathcal{R}$ is a well-typed reflexive closed relation, then:*

1. *$\mathcal{R}^\bullet$ is compatible, hence a reflexive open relation.*
2. *$\mathcal{R}^\circ \subseteq \mathcal{R}^\bullet$.*

Proof. We prove the properties separately.

1. Since $\mathcal{R}$ is reflexive, so is $\mathcal{R}^\circ$. Hence $\widehat{\mathcal{R}^\bullet} = \widehat{\mathcal{R}^\bullet}\mathit{id} \subseteq \widehat{\mathcal{R}^\bullet}\mathcal{R}^\circ = \mathcal{R}^\bullet$. Any compatible relation is reflexive, since the typing judgements line up with the compatible refinement rules.
2. Note that the compatible refinement of a reflexive relation is reflexive. Hence $\widehat{\mathcal{R}^\bullet}$ is reflexive, since $\mathcal{R}^\bullet$ is. So $\mathcal{R}^\circ = \mathit{id}\,\mathcal{R}^\circ \subseteq \widehat{\mathcal{R}^\bullet}\mathcal{R}^\circ = \mathcal{R}^\bullet$. □

The next lemma proves that the Howe closure of a reflexive set is *substitutive*, as in for example [75]. We give a sketch of the proof in terms of the coinductive definition of the Howe closure (see, e.g., [15] for a more complete proof for a probabilistic language).

Lemma 4.4.3 (Substitutivity)**.** *If $\mathcal{R}$ is a reflexive, transitive, and closed well-typed relation, and suppose that $(\Gamma; x{:}\mathbf{B}; \Gamma' \vdash \underline{P}\ \mathcal{R}^\bullet\ \underline{R} : \underline{\mathbf{E}})$ and $(\Gamma' \vdash V\ \mathcal{R}^\bullet\ W : \mathbf{B})$ hold, then $(\Gamma; \Gamma' \vdash \underline{P}[V/x]\ \mathcal{R}^\bullet\ \underline{R}[W/x] : \underline{\mathbf{E}})$.*

Proof. We perform an induction on the shape of $\underline{P}$ (which may be a value or a computation). If $(\Gamma; x : \mathbf{B}; \Gamma' \vdash \underline{P}\mathcal{R}^\bullet\underline{R} : \underline{\mathbf{E}})$ then by **H** it holds that $(\Gamma; x : \mathbf{B}; \Gamma' \vdash \underline{P}\widehat{\mathcal{R}^\bullet}\,\underline{Q})$ and $(\Gamma; x : \mathbf{B}; \Gamma' \vdash \underline{Q}\,\mathcal{R}^\circ\,\underline{R})$ for some $\underline{Q}$. So we know that $(\Gamma; \Gamma' \vdash \underline{Q}[W/x]\mathcal{R}^\circ\underline{R}[W/x])$. We need to prove that $(\Gamma; \Gamma' \vdash \underline{P}[V/x]\widehat{\mathcal{R}^\bullet}\underline{Q}[W/x])$. In each of the cases of $\underline{P}$, $(\Gamma; x{:}\mathbf{B}; \Gamma' \vdash \underline{P}\widehat{\mathcal{R}^\bullet}\,\underline{Q})$ is derived from rule **Cn** for some number n. This rule has as its premise some sequence of relations $\underline{P}_i\mathcal{R}^\bullet\underline{Q}_i$. By induction hypothesis it holds that $\underline{P}_i[V/x]\mathcal{R}^\bullet\underline{Q}_i[W/x]$, this is also trivially true in the base cases $n \in \{1, 2\}$ since then the sequence is empty. Using **Cn** we can then derive that $(\Gamma; \Gamma' \vdash \underline{P}[V/x]\widehat{\mathcal{R}^\bullet}\underline{Q}[W/x])$. One can verify that this argument works for each of the cases of **Cn**. So $(\Gamma; \Gamma' \vdash \underline{P}[V/x]\widehat{\mathcal{R}^\bullet}\underline{Q}[W/x])$ and $(\Gamma; \Gamma' \vdash \underline{Q}[W/x]\mathcal{R}^\circ\underline{R}[W/x])$, hence $\Gamma \vdash \underline{P}[V/x]\mathcal{R}^\bullet\underline{R}[W/x]$. □

Another result concerns the composition of the Howe closure with the open extension of the original relation. The first point of this result is featured for example in [41].

Lemma 4.4.4. *If $\mathcal{R}$ is a well-typed preorder on closed terms, and $\mathcal{O}(-)$ is a relator, then we have:*

1. *$\mathcal{R}^\bullet\mathcal{R}^\circ \subseteq \mathcal{R}^\bullet$.*
2. *For closed terms $\underline{M}, \underline{N} : \mathbf{F}\,\mathbf{A}$ and $t \in T(\mathbf{A})$ such that $|\underline{M}|\ \mathcal{O}(\mathcal{R}^\bullet)\ t$ and $t\ \mathcal{O}(\mathcal{R}^\circ)\ |\underline{N}|$, it holds that $|\underline{M}|\ \mathcal{O}(\mathcal{R}^\bullet)\ |\underline{N}|$.*

Proof. We prove the properties individually.

1. We use that $\mathcal{R}$ is transitive, hence $\mathcal{R}^\circ$ is transitive meaning $\mathcal{R}^\circ\mathcal{R}^\circ \subseteq \mathcal{R}^\circ$. Hence with $\mathcal{R}^\bullet = \widehat{\mathcal{R}^\bullet}\mathcal{R}^\circ$ it holds that $\mathcal{R}^\bullet\mathcal{R}^\circ = (\widehat{\mathcal{R}^\bullet}\mathcal{R}^\circ)\mathcal{R}^\circ \subseteq \widehat{\mathcal{R}^\bullet}\mathcal{R}^\circ = \mathcal{R}^\bullet$.

2. This follows from applying property 2 of Definition 4.1.1 to the previous statement. □

4.4.1 The Howe closure of an applicative $\mathcal{O}$-simulation

We assume $\mathcal{O}$ is a decomposable set of Scott open modalities, hence by Lemma 4.1.2, $\mathcal{O}(-)$ is a relator. Let $\subseteq$ be a well-typed preorder on closed terms, and assume $\subseteq$ is an an applicative $\mathcal{O}$-simulation. We look at the Howe closure of $\subseteq$. The lemmas stated before are satisfied, hence in particular it holds that $(\subseteq^\circ) \subseteq (\subseteq^\bullet)$ by Lemma 4.4.2. We prove that $\subseteq^{\bullet\vdash}$ is an $\mathcal{O}$-simulation. All terms in the following lemmas are assumed to be closed, unless stated otherwise, and the lemmas hold for any well-typed preorder on closed terms $\subseteq$.

Lemma 4.4.5. *If for closed terms $V, W : \mathbf{N}$ it holds that $V \subseteq^\bullet W$, then $V = W$.*

Proof. Using the definition of $\subseteq^\bullet$ there must be an $L : \mathbf{N}$ such that $V\ \widehat{\subseteq^\bullet}\ L$ and $L \subseteq W$, the latter implying $L = W$ because of simulation property (1). We do an induction on the shape of V.

Induction basis, if $V = \mathsf{Z}$, then $V\ \widehat{\subseteq^\bullet}\ L$ could only have come from rule **C3**, so $L = \mathsf{Z}$ and hence $V = L = W$.

Induction step, if $V = \mathsf{S}(V')$, then the statement $V\ \widehat{\subseteq^\bullet} L$ could only have come from rule **C4**, so $L = \mathsf{S}(L')$ where $V' \subseteq^\bullet L'$. By induction hypothesis, $V' = L'$. Hence $V = \mathsf{S}(V') = \mathsf{S}(L') = L = W$. □

Two other simulation properties follow directly from Lemma 4.4.2.

Lemma 4.4.6. *By compatibility of $\subseteq^\bullet$ the following three statements are apparent:*

1. *For $\underline{M}\ \underline{\subseteq}^\bullet_{\mathbf{A}\to\mathbf{C}}\ \underline{N}$ it holds that $\forall V \in \mathit{Terms}(\mathbf{A})$, $\underline{M}\ V\ \underline{\subseteq}^\bullet_{\mathbf{C}}\ \underline{N}\ V$.*

2. *For $\underline{M}\ \underline{\subseteq}^\bullet_{\mathbf{\Pi}_{i\in I}\mathbf{C}_i}\ \underline{N}$ it holds that $\forall j \in I, \underline{M}\ j\ \underline{\subseteq}^\bullet_{\mathbf{C}_j}\ \underline{N}\ j$.*

Lemma 4.4.7. *For $\mathsf{thunk}(\underline{M})\ \underline{\subseteq}^\bullet_{\mathbf{U}\,\mathbf{C}}\ \mathsf{thunk}(\underline{N})$ it holds that $\underline{M}\ \underline{\subseteq}^\bullet_{\mathbf{C}}\ \underline{N}$.*

Proof. There is a term $\mathsf{thunk}(\underline{K})$ such that $\mathsf{thunk}(\underline{M})\ \widehat{\underline{\subseteq}^\bullet_{\mathbf{U}\,\mathbf{C}}}\ \mathsf{thunk}(\underline{K}) \subseteq^\circ \mathsf{thunk}(\underline{N})$. By simulation property, $\underline{K} \subseteq^\circ \underline{N}$. The statement $\mathsf{thunk}(\underline{M})\ \widehat{\underline{\subseteq}^\bullet_{\mathbf{U}\,\mathbf{C}}}\ \mathsf{thunk}(\underline{K})$ could only have come from compatible refinement rule **C19**, so $\underline{M} \subseteq^\bullet \underline{K}$. We now use Lemma 4.4.4 to conclude that $\underline{M} \subseteq^\bullet \underline{N}$. □

Lemma 4.4.8. *If $(j, V)\ \underline{\subseteq}^\bullet_{\mathbf{\Sigma}_{i\in I}\mathbf{A}_i}\ (k, W)$ then $j = k$ and $V\ \underline{\subseteq}^\bullet_{\mathbf{A}_j}\ W$.*

Proof. There is a pair (l, L) such that $(j, V)\,\widehat{\subseteq^\bullet_{\Sigma_{i\in I}\,\mathbf{A}_i}}\,(l, L)\subseteq^\circ(k, W)$. The latter implies $l = k$ and $L\subseteq W$ by simulation property. The former statement can only have come from compatible refinement rule **C13**, so $j = l = k$ and $V \subseteq^\bullet L$. We now use Lemma 4.4.4 to conclude that $V \subseteq^\bullet W$. □

Lemma 4.4.9. *If* $(V, V')\subseteq^\bullet_{\mathbf{A}\times\mathbf{B}}(W, W')$ *then* $V \subseteq^\bullet_{\mathbf{A}} W$ *and* $V' \subseteq^\bullet_{\mathbf{B}} W'$.

Proof. There is a pair (L, L') such that $(V, V')\,\widehat{\subseteq^\bullet_{\mathbf{A}\times\mathbf{B}}}\,(L, L')\subseteq^\circ(W, W')$. The latter implies $L\subseteq W$ and $L'\subseteq W'$ by simulation property. The former statement can only have come from compatible refinement rule **C15**, so $V \subseteq^\bullet L$ and $V' \subseteq^\bullet L'$. We use Lemma 4.4.4 to conclude that $V \subseteq^\bullet W$ and $V' \subseteq^\bullet W'$. □

So all conditions except 6 of being an $\mathcal{O}$-simulation (Definition 4.2.1) are proven to be satisfied by the Howe closure of an $\mathcal{O}$-simulation.

4.4.2 Effectful behaviour and the proof by induction

It is most difficult to prove that the Howe closure of an $\mathcal{O}$-simulation satisfies condition of Definition 4.2.1. The proof needs an induction on the reduction relation of terms. It requires us to look at terms $\underline{P}$, $\underline{R}$ of type $\mathbf{F}\,\mathbf{A}$ such that $\underline{P}\subseteq^\bullet \underline{R}$, and prove that $|\underline{P}|\,\mathcal{O}(\subseteq^\bullet)\,|\underline{R}|$. Using continuity, this can be reduced to proving that $|\underline{P}|_n\,\mathcal{O}(\subseteq^\bullet)\,|\underline{R}|$ for all n (see Lemma 4.4.17). So the property can be proven using an induction on n. In this proof by induction, one would look at the shape of $\underline{P}$ and see what it reduces to after one step, so one can use the induction hypothesis on that result. This is a relatively straightforward investigation in the fine-grained call-by-value case considered in the paper this chapter is based on [95].

However, in our operational semantics of call-by-push-value, we see cases where we do not know directly how to reduce a term. In such cases, we used a system of stacks to focus the evaluation of a program on a subterm (see Subsection 2.1.1). For instance, to evaluate $\underline{M}\;V$, we first need to evaluate $\underline{M}$. However, $\underline{M}$ may invoke effects, and as a term of function type, such effects are not 'directly' observable. So we need to give ourselves a bit more information on $\underline{M}$ before we can properly perform the proof by induction.

This is also necessary for terms of the form $\underline{M}\;i$. One program packed in a stack $S\{\underline{M}\}$ for which we do already have enough information, is a program of the shape $S\{\underline{M}\} = \underline{M}\;\mathsf{to}\;x.\,\underline{N}$. Here, $\underline{M}$ is of a producer-type, at which effects are observed with modalities as usual. So the induction hypothesis will give us sufficient information to carry out the induction. Note that this $(-)\;\mathsf{to}\;x.\,\underline{N}$ operator is the only sort of stack used in the operational semantics of a fine-grained call-by-value language.

So in order to give us sufficient information during our induction proof of Lemma 4.4.17, we need to give ourselves more information on the terms we are investigating.

Definition 4.4.10. A stack S is called a *frame* if it only consists of $(-)\;V$ and $(-)\;i$ parts. A computation term is called *informative* if it is not of the form $\underline{M}\;V$ or $\underline{M}\;i$.

We use the term 'informative' to simply label terms which give us enough information to perform the forthcoming proof by induction directly. For instance, all fine-grained call-by-value terms as embedded in the call-by-push-value language are informative. Examples of informative terms include; $\underline{M}$ to $x.\,\underline{N}$, $\mathsf{return}(V)$, $\mathsf{force}(V)$, $\lambda x.\,\underline{M}$ and $\mathsf{or}(\underline{M},\underline{N})$.

Doing structural induction on terms, we observe the following result.

Lemma 4.4.11. *Any computation term $\underline{M}$ is uniquely of the form $S\{\underline{M'}\}$ where S is a frame and $\underline{M'}$ is an informative term.*

This lemma is motivation for the introduction of frames. It tells us that each term has an informative core, which will yield enough information to carry out our proof by induction. Before that proof however, we need to establish a plethora of smaller results.

Lemma 4.4.12. *For any frame $S \in \mathit{Stack}(\underline{\mathbf{C}},\underline{\mathbf{D}})$, if $\underline{M}\subseteq_{\underline{\mathbf{C}}}\underline{N}$ then $S\{\underline{M}\}\subseteq_{\underline{\mathbf{D}}}S\{\underline{N}\}$.*

Proof. We do an induction on the shape of S. For the induction basis, where $S = \varepsilon$ the conclusion follows from $S\{\underline{M}\} = \underline{M}$ and $S\{\underline{N}\} = \underline{N}$. For the induction step, assume as induction hypothesis that the result holds for $Z \in \mathit{Stack}(\underline{\mathbf{C}'},\underline{\mathbf{D}})$.

If $S = Z\circ(-)\ V$, then $\underline{\mathbf{C}}$ is of the form $\mathbf{A}\to\underline{\mathbf{C}'}$. Since $\subseteq$ is a simulation, $\underline{M}\subseteq_{\mathbf{A}\to\underline{\mathbf{C}'}}\underline{N}$ implies $\underline{M}\ V\subseteq_{\underline{\mathbf{C}'}}\underline{N}\ V$, so by induction hypothesis $S\{\underline{M}\} = Z\{\underline{M}\ V\}\subseteq_{\underline{\mathbf{D}}}Z\{\underline{N}\ V\} = S\{\underline{N}\}$.

If $S = Z\circ(-)\ j$, then $\underline{\mathbf{C}}$ is of the form $\mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}_i}$ where $\underline{\mathbf{C}_j} = \underline{\mathbf{C}'}$. Since $\subseteq$ is a simulation, $\underline{M}\subseteq_{\mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}_i}}\underline{N}$ implies $\underline{M}\ j\subseteq_{\underline{\mathbf{C}'}}\underline{N}\ j$, so by induction hypothesis $S\{\underline{M}\} = Z\{\underline{M}\ j\}\subseteq_{\underline{\mathbf{D}}}Z\{\underline{N}\ j\} = S\{\underline{N}\}$. □

Definition 4.4.13. Given two frames S and Z, we say Z *dominates* S if:

1. If $S = \varepsilon$ then $Z = \varepsilon$.
2. If $S = S'\circ(-)\ V$ then $Z = Z'\circ(-)\ V'$ where $V\subseteq^{\bullet}V'$, and Z' dominates S'.
3. If $S = S'\circ(-)\ i$ then $Z = Z'\circ(-)\ i$ and Z' dominates S'.

Note that the above relation on frames is not symmetric. Dominating frames are very handy, since they can make use of compatibility:

Lemma 4.4.14. *If frame Z dominates frame S, and $\underline{M}\subseteq^{\bullet}\underline{N}$, then $S\{\underline{M}\}\subseteq^{\bullet}Z\{\underline{N}\}$.*

Proof. This follows from compatibility of $\subseteq^{\bullet}$, using an induction on frames. □

Note that $((-)\ V\circ S)\{\underline{M}\} = S\{\underline{M}\}\ V$, which can be proven by a simple induction on stacks. Similar properties hold for the other Stack constructors. We have the following fundamental property.

Lemma 4.4.15. *Given a frame S and two closed computation terms $\underline{M'}$ and $\underline{N}$ such that $S\{\underline{M'}\}\subseteq^{\bullet}\underline{N}$, then there is a frame Z and a term $\underline{N'}$ such that; Z dominates S, $\underline{M'}\ \widehat{\subseteq^{\bullet}}\ \underline{N'}$ and $Z\{\underline{N'}\}\subseteq\underline{N}$.*

Proof. We do this by induction on frame S. If $S = \varepsilon$, then the statements hold by taking $Z = \varepsilon$ and $\underline{N'}$ such that $\underline{M'} \mathrel{\widehat{\subseteq^\bullet}} \underline{N'} \subseteq \underline{N}$, which exists since $(\subseteq^\bullet) = (\mathrel{\widehat{\subseteq^\bullet}}) \circ (\subseteq^\circ)$. Now for the induction step, assume the statement holds for any smaller frame S'.

1. If $S = ((-)\ V) \circ S'$, then $S\{\underline{M'}\} = (S'\{\underline{M'}\}\ V)$. Now, there is a term $\underline{K}$ such that $S\{\underline{M'}\} \mathrel{\widehat{\subseteq^\bullet}} \underline{K} \subseteq \underline{N}$. The statement $(S'\{\underline{M'}\}\ V) \mathrel{\widehat{\subseteq^\bullet}} \underline{K}$ could only have been derived from rule **C12**, so we know there are $\underline{K'}$ and W such that $\underline{K} = (\underline{K'}\ W)$, $S'\{\underline{M'}\} \subseteq^\bullet \underline{K'}$, and $V \subseteq^\bullet W$.

   We use the induction hypothesis on $S'\{\underline{M'}\} \subseteq^\bullet \underline{K'}$ to find a term $\underline{N'}$ and frame Z' dominating S' such that $\underline{M'} \mathrel{\widehat{\subseteq^\bullet}} \underline{N'}$ and $Z'\{\underline{N'}\} \subseteq \underline{K'}$. Let $Z := ((-)\ W) \circ Z'$, then Z dominates S. From $Z'\{\underline{N'}\} \subseteq \underline{K'}$ and simulation rule 5 it holds that $Z\{\underline{N'}\} = (Z'\{\underline{N'}\}\ W) \subseteq (\underline{K'}\ W) = \underline{K}$. With $\underline{K} \subseteq \underline{N}$ we can conclude that $Z\{\underline{N'}\} \subseteq \underline{N}$ because $\subseteq$ is a preorder. From earlier, $\underline{M'} \mathrel{\widehat{\subseteq^\bullet}} \underline{N'}$, so Z and $\underline{N'}$ have the desired properties.

2. If $S = ((-)\ i) \circ S'$, then $S\{\underline{M'}\} = (S'\{\underline{M'}\}\ i)$. Now, there is a term $\underline{K}$ such that $S\{\underline{M'}\} \mathrel{\widehat{\subseteq^\bullet}} \underline{K} \subseteq \underline{N}$. The statement $(S'\{\underline{M'}\}\ i) \mathrel{\widehat{\subseteq^\bullet}} \underline{K}$ could only have been derived from rule **C18**, so we know there is a $\underline{K'}$ such that $\underline{K} = (\underline{K'}\ i)$ and $S'\{\underline{M'}\} \subseteq^\bullet \underline{K'}$.

   We use the induction hypothesis on $S'\{\underline{M'}\} \subseteq^\bullet \underline{K'}$ to find a term $\underline{N'}$ and frame Z' dominating S' such that $\underline{M'} \mathrel{\widehat{\subseteq^\bullet}} \underline{N'}$ and $Z'\{\underline{N'}\} \subseteq \underline{K'}$. Let $Z := ((-)\ i) \circ Z'$, then Z dominates S. From $Z'\{\underline{N'}\} \subseteq \underline{K'}$ and simulation rule 7 it holds that $Z\{\underline{N'}\} = (Z'\{\underline{N'}\}\ i) \subseteq (\underline{K'}\ i) = \underline{K}$. With $\underline{K} \subseteq \underline{N}$ we can conclude that $Z\{\underline{N'}\} \subseteq \underline{N}$ and from earlier $\underline{M'} \mathrel{\widehat{\subseteq^\bullet}} \underline{N'}$. So Z and $\underline{N'}$ have the desired properties. □

The last important property of frames is that they act nicely with respect to the reduction relation, which can be proven by a simple induction on frames.

Lemma 4.4.16. *By induction on frames S, we can derive the following facts:*

1. *If $\underline{M} \rightsquigarrow \underline{N}$, then for any $k \in \mathbb{N}$ and any frame S, $|S\{\underline{M}\}|_{k+1} = |S\{\underline{N}\}|_k$.*
2. *If $(S, \underline{M}) \rightarrowtail (Z, \underline{K}) \rightarrowtail (S, \underline{N})$, then $|S\{\underline{M}\}|_{k+2} = |S\{\underline{N}\}|_k$.*
3. *$|S\{\mathsf{op}(\overline{l_1}, \ldots, \overline{l_m}; \underline{M_1}, \ldots, \underline{M_k})\}|_{n+1} \leq \mathsf{op}_{l_1,\ldots,l_m}\langle |S\{\underline{M_1}\}|_n, \ldots, |S\{\underline{M_k}\}|_n \rangle$ and $|S\{\mathsf{op}(\overline{l_1}, \ldots, \overline{l_m}; \underline{M_1}, \ldots, \underline{M_k})\}| = \mathsf{op}_{l_1,\ldots,l_m}\langle |S\{\underline{M_1}\}|, \ldots, |S\{\underline{M_k}\}| \rangle$, for any effect operator $\mathsf{op} : \mathbf{N}^m \times \alpha^k \to \alpha \in \Sigma$.*
4. *$|S\{\mathsf{op}(\overline{l_1}, \ldots, \overline{l_m}; x \mapsto \underline{M})\}|_{n+1} \leq \mathsf{op}_{l_1,\ldots,l_m}\langle k \mapsto |S\{\underline{M}[\overline{k}/x]\}|_{\max(0,n-k)} \rangle$ and $|S\{\mathsf{op}(\overline{l_1}, \ldots, \overline{l_m}; x \mapsto \underline{M})\}| = \mathsf{op}_{l_1,\ldots,l_m}\langle k \mapsto |S\{\underline{M}[\overline{k}/x]\}| \rangle$, for any effect operator $\mathsf{op} : \mathbf{N}^m \times \alpha^{\mathbb{N}} \to \alpha \in \Sigma$.*

Proof. If S has length m, then $(\varepsilon, S\{\underline{M}\})$ reduces in m steps to $(S, \underline{M})$. All the above results can then be established by reducing $\underline{M}$. If the index of $|-|_{(-)}$ is too small, properties 1 and 2 still hold because both sides will be $\bot$, and properties 3 and 4 hold because of the use of the inequality $\leq$.

□

We have finally have the necessary tools to prove the Key Lemma, which contains the proof by induction we eluded to in this subsection.

Lemma 4.4.17. *(Key Lemma) Let $n \in \mathbb{N}$. Given two closed terms $\underline{M}$, $\underline{N}$ of type $\mathbf{F}\,\mathbf{A}$ such that $\underline{M} \subseteq^\bullet \underline{N}$, then it holds that $|\underline{M}|_n \, \mathcal{O}(\subseteq^\bullet) \, |\underline{N}|$.*

Proof. We do an induction on n.

Base case. If $n = 0$, then $|\underline{M}|_0 = \bot$. Hence $|\underline{M}|_0 \, \mathcal{O}(\subseteq^\bullet) \, |\underline{N}|$ by Lemma 4.3.2.

Induction step $(n+1)$. We assume as the *induction hypothesis* that for any $k \leq n$ and $\underline{M'} \subseteq^\bullet \underline{N'}$ it holds that $|\underline{M'}|_k \, \mathcal{O}(\subseteq^\bullet) \, |\underline{N'}|$. To prove, for any $\underline{M} \subseteq^\bullet \underline{N}$ it holds that $|\underline{M}|_{n+1} \, \mathcal{O}(\subseteq^\bullet) \, |\underline{N}|$.
Assume $\underline{M} \subseteq^\bullet \underline{N}$. We use Lemma 4.4.11 to find a frame S and an informative term $\underline{M'}$ such that $\underline{M} = S\{\underline{M'}\}$. We then use Lemma 4.4.15 to find a frame Z dominating S together with a term $\underline{N'}$ such that: $Z\{\underline{N'}\} \subseteq \underline{N}$ and $\underline{M'} \mathrel{\widehat{\subseteq^\bullet}} \underline{N'}$.

In the next lemma, Lemma 4.4.18, we prove that under the current circumstances, $|\underline{M}|_{n+1} = |S\{\underline{M'}\}|_{n+1} \, \mathcal{O}(\subseteq^\bullet) \, |Z\{\underline{N'}\}|$ holds[3]. Given this, since $Z\{\underline{N'}\} \subseteq \underline{N}$ and hence by simulation property 6 $|Z\{\underline{N'}\}| \, \mathcal{O}(\subseteq) \, |\underline{N}|$, we can conclude via Lemma 4.4.4 that $|\underline{M}|_{n+1} \, \mathcal{O}(\subseteq^\bullet) \, |\underline{N}|$. □

Lemma 4.4.18. *Suppose that for all $k \in \mathbb{N}$, $k \leq n$, and for all closed terms $\underline{P}, \underline{Q} : \mathbf{F}\,\mathbf{A}$:*

$$(IH) \qquad \underline{P} \subseteq^\bullet \underline{Q} \implies |\underline{P}|_k \, \mathcal{O}(\subseteq^\bullet) \, |\underline{Q}| \ .$$

For any two closed terms $S\{\underline{M}\}, Z\{\underline{N}\}$ with $\underline{M}$ informative, and Z a frame dominating the frame S, it holds that $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N} \implies |S\{\underline{M}\}|_{n+1} \, \mathcal{O}(\subseteq^\bullet) \, |Z\{\underline{N}\}|$.

Proof. We assume $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ and do a case distinction on $\underline{M}$, which is informative, so not equal to either $(\underline{P}\ V)$ or $(\underline{P}\ i)$. We start with the three cases where the frame S is actively used.

1. If $\underline{M} = \mathsf{return}(V) : \underline{\mathbf{C}}$, then $\underline{\mathbf{C}} = \mathbf{F}\,\mathbf{B}$ for some $\mathbf{B}$, so frame S must be ε as no other frames accept a term of this type. Hence $\underline{M} = S\{\underline{M}\}$, $\underline{\mathbf{C}} = \mathbf{F}\,\mathbf{A}$, and $|S\{\underline{M}\}| = |\underline{M}| = \langle V \rangle$. The dominating frame Z must be ε too, so $\mathsf{return}(V) \mathrel{\widehat{\subseteq^\bullet}} \underline{N} = Z\{\underline{N}\}$. This is only possible from compatibility rule **C7**, meaning $\underline{N} = \mathsf{return}(W)$ for some W such that $V \subseteq^\bullet W$. By Lemma 4.3.1, $\langle V \rangle \ \mathcal{O}(\subseteq^\bullet) \ \langle W \rangle$, hence:

   $|S\{\underline{M}\}|_{n+1} = |\mathsf{return}(V)|_{n+1} = \langle V \rangle \ \mathcal{O}(\subseteq^\bullet) \ \langle W \rangle = |\mathsf{return}(W)| = |\underline{N}| = |Z\{\underline{N}\}|$.

2. If $\underline{M} = \lambda x.\, \underline{P}$, then for $S\{\underline{M}\}$ to be of type $\mathbf{F}\,\mathbf{A}$, S must be non-empty, and hence of the form $S' \circ (-)\ V$. Since Z dominates S, $Z = Z' \circ (-)\ W$ where Z' dominates S', and $V \subseteq^\bullet W$. The statement $\underline{M} = (\lambda x.\, \underline{P}) \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ could only have been derived via compatibility rule **C11**, so $\underline{N} = \lambda x.\, \underline{Q}$ for some $\underline{Q}$ where $(x : \mathbf{B}) \vdash \underline{P} \subseteq^\bullet \underline{Q} : \underline{\mathbf{D}}$. By Lemma 4.4.3 it holds that $\underline{P}[V/x] \subseteq^\bullet \underline{Q}[W/x]$, so we can do the following derivation using (IH) on $k = (n-1)$ and Lemma 4.4.16:

   $|S\{\underline{M}\}|_{n+1} = |S'\{\lambda x.\, \underline{P}\ V\}|_{n+1} = |S'\{\underline{P}[V/x]\}|_{n-1} \ \mathcal{O}(\subseteq^\bullet) \ |Z'\{\underline{Q}[W/x]\}| = |Z\{\underline{N}\}|$.

[3] In order to more easily add cases for $\underline{M}$ when extending the language in Chapter 7, we separate this result from the current lemma.

3. If $\underline{M} = \langle \underline{P}_i \mid i \in I \rangle$, then for $S\{\underline{M}\}$ to be of type $\mathbf{F}\,\mathbf{A}$, S must be non-empty, hence of the form $S' \circ (-)\ j$. Since Z dominates S, $Z = Z' \circ (-)\ j$ where Z' dominates S'. The statement $\underline{M} = \langle \underline{P}_i \mid i \in I \rangle \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ could only have been derived from compatibility rule **C17**, so $\underline{N} = \langle \underline{Q}_i \mid i \in I \rangle$ for some sequence of $\underline{Q}_i$-s, where $\forall i, \underline{P}_i \subseteq^\bullet \underline{Q}_i$. We can do the following derivation using (IH) on $k = (n-1)$ and Lemma 4.4.16:

   $|S\{\underline{M}\}|_{n+1} = |S'\{\langle \underline{P}_i \mid i \in I \rangle\ j\}|_{n+1} = |S'\{\underline{P}_j\}|_{n-1}\ \mathcal{O}(\subseteq^\bullet)\ |Z'\{\underline{Q}_j\}| = |Z\{\underline{N}\}|.$

Computation sequencing is a special case, and needs the decomposability property.

4. If $\underline{M} = \underline{P}\ \mathsf{to}\ x.\underline{Q}$, then by Corollary 2.2.7 it holds that $|S\{\underline{M}\}|_{n+1} \leq |\underline{P}|_n[\mathsf{return}(V) \mapsto |S\{\underline{Q}[V/x]\}|_n]$. Now $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ results only from compatibility rule **C8**, hence $\underline{N} = \underline{P}'\ \mathsf{to}\ x.\underline{Q}'$, where $\underline{P} \subseteq^\bullet \underline{P}'$ and $(x{:}\mathbf{B}) \vdash \underline{Q} \subseteq^\bullet \underline{Q}'$. By a generalisation of Corollary 2.2.10, $|Z\{\underline{N}\}| = \mu(|\underline{P}'|[\mathsf{return}(W) \mapsto |Z\{\underline{Q}'[W/x]\}|])$, and by (IH) it holds that $|\underline{P}|_n\ \mathcal{O}(\subseteq^\bullet)\ |\underline{P}'|$.

   Let $t := |\underline{P}|_n$ and $r := |\underline{P}'|$, which are trees from $T(Tct(\mathbf{F}\,\mathbf{B}))$.

   For any $V \subseteq^\bullet W$ we get via Lemma 4.4.3 that $\underline{Q}[V/x] \subseteq^\bullet \underline{Q}'[W/x]$, and since Z dominates S it holds that $S\{\underline{Q}[V/x]\} \subseteq^\bullet Z\{\underline{Q}'[W/x]\}$ and hence by (IH) we get $|S\{\underline{Q}[V/x]\}|_n\ \mathcal{O}(\subseteq^\bullet)\ |Z\{\underline{Q}'[W/x]\}|$. So define $f, g : Tct(\mathbf{F}\,\mathbf{B}) \to T(\mathbf{A})$, where:

   $f(\mathsf{return}(V)) := |S\{\underline{Q}[V/x]\}|_n$ and $g(\mathsf{return}(V)) := |Z\{\underline{Q}'[V/x]\}|$.

   By using Corollary 4.3.4 on t, r, f, g we conclude that:

$$\begin{array}{c} |S\{\underline{M}\}|_{n+1} \ \leq\ |\underline{P}|_n[\mathsf{return}(V) \mapsto |S\{\underline{Q}[V/x]\}|_n]\ \ \mathcal{O}(\subseteq^\bullet) \\ |\underline{P}'|[\mathsf{return}(W) \mapsto |Z\{\underline{Q}'[W/x]\}|]\ =\ |Z\{\underline{N}\}| \quad . \end{array}$$

Next are the terms which we can reduce with $\rightsquigarrow$:

5. If $\underline{M} = \mathsf{force}(V) : \underline{\mathbf{C}}$, then since V must be a closed value of type $\mathbf{U}\,\underline{\mathbf{C}}$ it must be of the form $\mathsf{thunk}(\underline{P})$ for some term $\underline{P}$, and hence $\underline{M} \rightsquigarrow \underline{P}$. Now $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ can only come from compatibility rule **C10**, hence $\underline{N} = \mathsf{force}(W)$ where $V \subseteq^\bullet W$. From **H** we get $\mathsf{thunk}(\underline{P}) = V \mathrel{\widehat{\subseteq^\bullet}} L \subseteq W$ for some L. The first relation we can only be from **C9** so $L = \mathsf{thunk}(\underline{R})$ where $\underline{P} \subseteq^\bullet \underline{R}$. Since W is closed, it must be of the form $\mathsf{thunk}(\underline{Q})$, hence $\mathsf{thunk}(\underline{R}) \subseteq \mathsf{thunk}(\underline{Q})$. Since $\subseteq$ is a simulation, we get $\underline{R} \subseteq \underline{Q}$ hence by Lemma 4.4.4 we get $\underline{P} \subseteq^\bullet \underline{Q}$ where $\underline{N} = \mathsf{force}(W) = \mathsf{force}(\mathsf{thunk}(\underline{Q}))$. Using the fact that Z dominates S, it holds that $S\{\underline{P}\} \subseteq^\bullet Z\{\underline{Q}\}$ which by (IH) and Lemma 4.4.16 means:

   $|S\{\underline{M}\}|_{n+1} = |S\{\underline{P}\}|_n\ \mathcal{O}(\subseteq^\bullet)\ |Z\{\underline{Q}\}| = |Z\{\underline{N}\}|.$

6. If $\underline{M} = \mathsf{case}\ V\ \mathsf{of}\ \{\underline{P}, \mathsf{S}(x) \Rightarrow \underline{Q}\}$, then $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ can only come from compatibility rule **C4**, hence $\underline{N} = \mathsf{case}\ V'\ \mathsf{of}\ \{\underline{P}', \mathsf{S}(x) \Rightarrow \underline{Q}'\}$ for which it holds that $V \subseteq^\bullet V'$, $\underline{P} \subseteq^\bullet \underline{P}'$ and $(x{:}\mathbf{B}) \vdash \underline{Q} \subseteq^\bullet \underline{Q}'$. Since $V \subseteq^\bullet V'$ it holds that $V = V'$ by Lemma 4.4.5. We do a case distinction on V.

a) If $V = \mathsf{Z} = V'$, then $\underline{M} \rightsquigarrow \underline{P}$ so $|S\{\underline{M}\}|_{n+1} = |S\{\underline{P}\}|_n$ by Lemma 4.4.16, and since Z dominates S, by (IH), $\underline{P} \subseteq^\bullet \underline{P}'$, and Lemma 4.4.14 we have $|S\{\underline{P}\}|_n \; \mathcal{O}(\subseteq^\bullet) \; |Z\{\underline{P}'\}| = |Z\{\mathsf{case}\ \mathsf{Z}\ \mathsf{of}\ \{\underline{P}', \mathsf{S}(x) \Rightarrow \underline{Q}'\}\}| = |Z\{\underline{N}\}|$.

b) If $V = \mathsf{S}(W) = V'$, then $\underline{M} \rightsquigarrow \underline{Q}[W/x]$ so $|S\{\underline{M}\}|_{n+1} = |S\{\underline{Q}[W/x]\}|_n$ by Lemma 4.4.16, and since $(x{:}\mathbf{B}) \vdash \underline{Q} \subseteq^\bullet \underline{Q}'$ it holds that $\underline{Q}[W/x] \subseteq^\bullet \underline{Q}'[W/x]$ (Lemma 4.4.3). Since Z dominates S, we can use Lemma 4.4.14 and (IH) to derive:
$|S\{\underline{M}\}|_{n+1} = S\{\underline{Q}[W/x]\}|_n \; \mathcal{O}(\subseteq^\bullet) \; |Z\{\underline{Q}'[W/x]\}| = |Z\{\underline{N}\}|$.

7. If $\underline{M} = \mathsf{let}\ V\ \mathsf{be}\ x.\, \underline{P} : \underline{\mathbf{C}}$, then the only premise for $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ is given by compatible refinement rule **C6**, from which we know $\underline{N} = \mathsf{let}\ W\ \mathsf{be}\ x.\, \underline{Q}$, where $V \subseteq^\bullet W$ and $(x{:}\mathbf{B}) \vdash \underline{P} \subseteq^\bullet \underline{Q}$. From this and Lemma 4.4.3 it holds that $\underline{P}[V/x] \subseteq^\bullet \underline{Q}[W/x]$ hence since Z dominates S, $S\{\underline{P}[V/x]\} \subseteq^\bullet Z\{\underline{Q}[W/x]\}$, so by (IH) it holds that $|S\{\underline{P}[V/x]\}|_n \; \mathcal{O}(\subseteq^\bullet) \; |Z\{\underline{Q}[W/x]\}|$. We have $\underline{M} \rightsquigarrow \underline{P}[V/x]$ hence $|S\{\underline{M}\}|_{n+1} = |S\{\underline{P}[V/x]\}|_n$, similarly $|Z\{\underline{N}\}| = |Z\{\underline{Q}[W/x]\}|$ so $|S\{\underline{M}\}|_{n+1} \; \mathcal{O}(\subseteq^\bullet) \; |Z\{\underline{N}\}|$.

8. If $\underline{M} = \mathsf{fix}(\underline{P}) : \underline{\mathbf{C}}$, then $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ can only be from **C19**, hence $\underline{N} = \mathsf{fix}(\underline{P}')$ where $\underline{P} \subseteq^\bullet \underline{P}'$. By compatibility, it holds that $\mathsf{thunk}(\underline{P}) \subseteq^\bullet \mathsf{thunk}(\underline{P}')$. Look at the computation term $\underline{Q} = (\mathsf{force}(x)\ \mathsf{thunk}(\mathsf{fix}(x)))$ in context $(x : \mathbf{U}\,(\mathbf{B} \to \underline{\mathbf{C}}))$. For any $\underline{M}' : \mathbf{B} \to \underline{\mathbf{C}}$, $\mathsf{fix}(\underline{M}') \rightsquigarrow \underline{Q}[\mathsf{thunk}(\underline{M}')/x]$. By reflexivity it holds that $\underline{Q} \subseteq^\bullet \underline{Q}$ and hence by Lemma 4.4.3 it holds that $\underline{Q}[\mathsf{thunk}(\underline{P})/x] \subseteq^\bullet \underline{Q}[\mathsf{thunk}(\underline{P}')/x]$. Using that Z dominates S and (IH) holds we derive that $|S\{\underline{Q}[\mathsf{thunk}(\underline{P})/x]\}|_n \; \mathcal{O}(\subseteq^\bullet) \; |Z\{\underline{Q}[\mathsf{thunk}(\underline{P}')/x]\}|$ and the desired conclusion follows from $|S\{\underline{M}\}|_{n+1} = |S\{\underline{Q}[\mathsf{thunk}(\underline{P})/x]\}|_n$ and $|Z\{\underline{N}\}| = |Z\{\underline{Q}[\mathsf{thunk}(\underline{P}')/x]\}|$.

The pattern match cases:

9. If $\underline{M} = (\mathsf{pm}\ V\ \mathsf{as}\ \{(i.x).\underline{P}_i\}_{i\in I})$, then $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ can only come from rule **C14**, hence $\underline{N} = (\mathsf{pm}\ W\ \mathsf{as}\ \{(i.x).\underline{Q}_i\}_{i\in I})$ where $V \subseteq^\bullet W$ and for each i it holds that $(x{:}\mathbf{B}) \vdash (\underline{P}_i \subseteq^\bullet \underline{Q}_i)$. As values from a sum-type, V and W must be of the form (j, V') and (k, W') respectively, where by the simulation rule proven in Lemma 4.4.8 it holds that $j = k$ and $V' \subseteq^\bullet W'$. Hence $\underline{M} \rightsquigarrow \underline{P}_j[V'/x]$ and $\underline{N} \rightsquigarrow \underline{Q}_j[W'/x]$, where by compatibility $\underline{P}_j[V'/x] \subseteq^\bullet \underline{Q}_j[W'/x]$. We conclude by (IH) that:
$|S\{\underline{M}\}|_{n+1} = |S\{\underline{P}_j[V'/x]\}|_n \; \mathcal{O}(\subseteq^\bullet) \; |Z\{\underline{Q}_j[W'/x]\}| = |Z\{\underline{N}\}|$.

10. If $\underline{M} = (\mathsf{pm}\ V\ \mathsf{as}\ (x, y).\underline{P})$, then $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ can only be from rule **C16**, hence $\underline{N} = (\mathsf{pm}\ W\ \mathsf{as}\ (x, y).\underline{Q})$ where $V \subseteq^\bullet W$ and $(x{:}\mathbf{B}; y : \mathbf{B}') \vdash (\underline{P} \subseteq^\bullet \underline{Q})$. As values of a pair-type, V and W must be of the form (V_0, V_1) and (W_0, W_1) respectively, where by the simulation rule proven in Lemma 4.4.9 it holds that $V_0 \subseteq^\bullet W_0$ and $V_1 \subseteq^\bullet W_1$. Hence by $\underline{M} \rightsquigarrow \underline{P}[V_0/x, V_1/y]$ and $\underline{N} \rightsquigarrow \underline{Q}[W_0/x, W_1/y]$, and by applying Lemma 4.4.3 twice, $\underline{P}[V_0/x, V_1/y] \subseteq^\bullet \underline{Q}[W_0/x, W_1/y]$. We conclude by (IH) that:
$|S\{\underline{M}\}|_{n+1} = |S\{\underline{P}[V_0/x, V_1/y]\}|_n \; \mathcal{O}(\subseteq^\bullet) \; |Z\{\underline{Q}[W_0/x, W_1/y]\}| = |Z\{\underline{N}\}|$.

Lastly the effect operator cases.

11. If $\underline{M} = \mathsf{op}(V_1, \ldots, V_m; \underline{P}_1, \ldots, \underline{P}_k) : \underline{\mathbf{C}}$ where $\mathsf{op} : \mathbf{N}^m \times \alpha^k \to \alpha$, then $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ can only be from **C20**, hence $\underline{N} = \mathsf{op}(V'_1, \ldots, V'_m; \underline{P}'_1, \ldots, \underline{P}'_k)$ where for all i between 1 and m, $\underline{P}_i \subseteq^\bullet \underline{P}'_i$, and for all j between 1 and k, $V_j = V'_j$ because of Lemma 4.4.5. Since Z dominates S and (IH) holds, $|S\{\underline{P}_i\}|_n \, \mathcal{O}(\subseteq^\bullet) \, |Z\{\underline{P}'_i\}|$, hence by Lemma 4.4.16 and Lemma 4.3.4:

$$|S\{\underline{M}\}|_{n+1} \leq \mathsf{op}_{V_1,\ldots,V_m}\langle |S\{\underline{P}_1\}|_n, \ldots, |S\{\underline{P}_k\}|_n \rangle \; \mathcal{O}(\subseteq^\bullet) \\ \mathsf{op}_{V_1,\ldots,V_m}\{|Z\{\underline{P}'_1\}|_n, \ldots, |Z\{\underline{P}'_k\}|_n\} = |Z\{\underline{N}\}|.$$

12. If $\underline{M} = \mathsf{op}(V_1, \ldots, V_m; x \mapsto \underline{P}) : \underline{\mathbf{C}}$ where $\mathsf{op} : \mathbf{N}^m \times \alpha^{\mathbf{N}} \to \alpha$, then $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ can only be from **C21**, hence $\underline{N} = \mathsf{op}(V_1, \ldots, V_m; x \mapsto \underline{P}')$ where $x : \mathbf{N} \vdash \underline{P} \subseteq^\bullet \underline{Q}$. Hence by Lemma 4.4.3, for all $\overline{k} : \mathbf{N}$ it holds that $\underline{P}[\overline{k}/x] \subseteq^\bullet \underline{Q}[\overline{k}/x]$ and hence since Z dominates S and (IH) holds, $|S\{\underline{P}[\overline{k}/x]\}|_n \, \mathcal{O}(\subseteq^\bullet) \, |Z\{\underline{Q}[\overline{k}/x]\}|$. So by Lemmas 4.4.16, 4.3.4, and 4.3.2:

$$|S\{\underline{M}\}|_{n+1} \leq \mathsf{op}_{V_1,\ldots,V_m}(k \mapsto |S\{\underline{P}[\overline{k}/x]\}|_{\max(0,n-k)}) \; \mathcal{O}(\subseteq^\bullet) \\ \mathsf{op}_{V_1,\ldots,V_m}(k \mapsto |Z\{\underline{Q}[\overline{k}/x]\}|) = |Z\{\underline{N}\}|.$$

Those were all the cases for $\underline{M}$, so we know that $|S\{\underline{M}\}|_{n+1} \; \mathcal{O}(\subseteq^\bullet) \; |Z\{\underline{N}\}|$ always holds. □

Using Lemma 4.3.2, we can conclude that $\underline{M} \subseteq^\bullet \underline{N} \Rightarrow |\underline{M}| \; \mathcal{O}(\subseteq^\bullet) \; |\underline{N}|$ for closed terms of type $\mathbf{F}\,\mathbf{A}$. We combine this result with Lemmas 4.4.5, 4.4.6, 4.4.7, 4.4.8, and 4.4.9 to get the following fundamental proposition:

Proposition 4.4.19. *Suppose $\mathcal{O}$ is a decompositional set Scott open modalities. Then for any preorder $\mathcal{R}$ on closed terms forming an $\mathcal{O}$-simulation, $\mathcal{R}^{\bullet\vdash}$ is an $\mathcal{O}$ simulation.*

4.5 Compatibility results

Using Proposition 4.4.19, we can derive that the open extensions of applicative $\mathcal{O}$-similarity and $\mathcal{O}$-bisimilarity are compatible. The proof of this fact is finished in this section. As a start, we look at the following standard result:

Lemma 4.5.1. *Suppose $\mathcal{R}$ is a reflexive, transitive, and closed open relation, and $\mathcal{S}$ is a well-typed closed relation. Then $\mathcal{R}^\vdash \subseteq \mathcal{S}$ implies $\mathcal{R} \subseteq \mathcal{S}^\circ$.*

Proof. Assume $\overrightarrow{x} : \overrightarrow{\mathbf{A}} \vdash \underline{\underline{P}} \; \mathcal{R} \; \underline{\underline{R}}$, then, because of substitutivity from Lemma 4.4.3, for any sequence of values $\overrightarrow{V} : \overrightarrow{\mathbf{A}}$ it holds that $\underline{\underline{P}}[\overrightarrow{V}/\overrightarrow{x}] \; \mathcal{R} \; \underline{\underline{R}}[\overrightarrow{V}/\overrightarrow{x}]$. Hence if $\mathcal{R}^\vdash \subseteq \mathcal{S}$, then for any $\overrightarrow{V} : \overrightarrow{\mathbf{A}}$ it holds that $\underline{\underline{P}}[\overrightarrow{V}/\overrightarrow{x}] \; \mathcal{S} \; \underline{\underline{R}}[\overrightarrow{V}/\overrightarrow{x}]$, and we can conclude that $\overrightarrow{x} : \overrightarrow{\mathbf{A}} \vdash \underline{M} \; \mathcal{S}^\circ \; \underline{N}$. □

Theorem 4.5.2. *If $\mathcal{O}$ is a decomposable set of Scott open modalities, then the open extension of the relation of applicative $\mathcal{O}$-similarity is compatible.*

Proof. We write $\sqsubseteq_s$ for the relation of $\mathcal{O}$-similarity. Since $\sqsubseteq_s$ is an $\mathcal{O}$-simulation, we know by Proposition 4.4.19 that $\sqsubseteq_s^{\bullet\,\vdash}$ is an $\mathcal{O}$-simulation, and hence is contained in $\mathcal{O}$-similarity $\sqsubseteq_s$. By Lemma 4.4.2 it holds that $\sqsubseteq_s^{\bullet}$ is compatible, and by Lemma 4.5.1 it is contained in the open extension $\sqsubseteq_s^{\circ}$. By Lemma 4.4.2, we also know that $\sqsubseteq_s^{\circ}$ is contained in $\sqsubseteq_s^{\bullet}$. We can conclude that $\sqsubseteq_s^{\circ}$ is equal to the Howe closure $\sqsubseteq_s^{\bullet}$, which is compatible. □

To prove that the open extension of applicative $\mathcal{O}$-bisimilarity is compatible, we need two other results from the literature (e.g., from Lassen [41]). This particular proof uses what is called *the transitive closure trick*, and only works if the syntax of the language is finitary. See [44] for an approach to applying Howe's method to a language with infinitary syntax.

Lemma 4.5.3. *The transitive closure of a compatible relation is compatible.*

Proof. Suppose $\mathcal{R}$ is compatible, we want to prove that $\widehat{\mathcal{R}^*} \subseteq \mathcal{R}^*$. For illustration, we only prove this for compatible refinement rule **C15**, and note that the proof can be adapted for the other rules.

Assume $\underline{M}\,\mathcal{R}^*_{\mathbf{A}\to\underline{\mathbf{C}}}\,\underline{M'}$ and $V\,\mathcal{R}^*_{\mathbf{A}}\,V'$, we prove that $(\underline{M}\;V)\,\mathcal{R}^*_{\underline{\mathbf{C}}}\,(\underline{M'}\;V')$. There must be sequences $\underline{N}_1,\ldots,\underline{N}_n : \mathbf{A}\to\underline{\mathbf{C}}$ and $W_1,\ldots,W_n : \mathbf{A}$ such that $\underline{M} = \underline{N}_1\,\mathcal{R}\,\underline{N}_2\,\mathcal{R}\ldots\mathcal{R}\,\underline{N}_n = \underline{M'}$ and $V = W_1\,\mathcal{R}\,W_2\,\mathcal{R}\ldots\mathcal{R}\,W_m = V'$. We may assume without loss of generality that $n = m$ because, since $\mathcal{R}$ is compatible and hence reflexive (so $\underline{M'}\,\mathcal{R}\,\underline{M'}$ and $V'\,\mathcal{R}\,V'$). Since $\mathcal{R}$ is compatible, $(\underline{N}_1\;W_1)\,\mathcal{R}\,(\underline{N}_2\;W_2)\,\mathcal{R}\ldots\mathcal{R}\,(\underline{N}_n\;W_n)$, hence we can conclude that $(\underline{M}\;V)\,\mathcal{R}^*_{\underline{\mathbf{C}}}\,(\underline{M'}\;V')$. □

Lemma 4.5.4. *If $\mathcal{R}^\circ$ is symmetric and reflexive, then $\mathcal{R}^{\bullet *}$ is symmetric.*

Proof. This proof is taken from [41, Lemma 3.8.2(4)]. Looking at the compatible refinement rules, it is not difficult to see $\widehat{\mathcal{S}^{op}} = \widehat{\mathcal{S}}^{op}$ for any relation $\mathcal{S}$. From Lemma 4.4.2 we know that $\mathcal{R}^\circ \subseteq \mathcal{R}^\bullet$, and $\mathcal{R}^\bullet$ is compatible hence by Lemma 4.5.3, $\widehat{\mathcal{R}^{\bullet *}} \subseteq \mathcal{R}^{\bullet *}$. So:

$$\widehat{\mathcal{R}^{\bullet *op}}\mathcal{R}^\circ = \widehat{\mathcal{R}^{\bullet *op}}\mathcal{R}^{\circ op} = \widehat{\mathcal{R}^{\bullet *}}^{op}\mathcal{R}^{\circ op} \subseteq$$

$$\mathcal{R}^{\bullet *op}\mathcal{R}^{\circ op} \subseteq \mathcal{R}^{\bullet *op}\mathcal{R}^{\bullet op} = \mathcal{R}^{\bullet op *}\mathcal{R}^{\bullet op} \subseteq \mathcal{R}^{\bullet op *} = \mathcal{R}^{\bullet *op}\ .$$

Hence $\mathcal{R}^{\bullet *op}$ is a solution for $\mathcal{S}$ to the inclusion $\widehat{\mathcal{S}}\;\mathcal{R}^\circ \subseteq \mathcal{S}$. Since $\mathcal{R}^\bullet$ is the least solution, it holds that $\mathcal{R}^\bullet \subseteq \mathcal{R}^{\bullet *op}$. So if $A\,\mathcal{R}^{\bullet *}\,B$, then $A = C_0\,\mathcal{R}^\bullet\,C_1\,\mathcal{R}^\bullet\ldots\mathcal{R}^\bullet\,C_{n-1} = B$ for some choice of sequence $\{C_i\}_{i\in I}$, so we can derive that $A = C_0\,\mathcal{R}^{\bullet *op}\,C_1\,\mathcal{R}^{\bullet *op}\ldots\mathcal{R}^{\bullet *op}\,C_{n-1} = B$ meaning $A\,\mathcal{R}^{\bullet *op}\,B$. We conclude that $B\,\mathcal{R}^{\bullet *}\,A$, so $\mathcal{R}^{\bullet *}$ is symmetric. □

Given these facts, we can derive the following.

Theorem 4.5.5. *If $\mathcal{O}$ is a decomposable set of Scott open modalities, then the open extension of the relation of applicative $\mathcal{O}$-bisimilarity is compatible.*

Proof. We write $\mathcal{O}$-bisimilarity as $\sqsupseteq_b$. From Proposition 4.4.19 we know that $\sqsupseteq_b^\bullet$ on closed terms is an $\mathcal{O}$-simulation. The transitive closure of an $\mathcal{O}$-simulation is an $\mathcal{O}$-simulation, since all the conditions for a relation to be an $\mathcal{O}$-simulation are preserved over transitive closure. In particular, the **F A** clause 6 for being a simulation is preserved because of point 2 of Lemma 4.1.2. So we know that $(\sqsupseteq_b^\bullet)^{*\vdash} = (\sqsupseteq_b^\bullet)^{\vdash *}$ is an $\mathcal{O}$-simulation. Since $\sqsupseteq_b$ is reflexive and symmetric, we know by the Lemma 4.5.4 that $\sqsupseteq_b^{\bullet *}$ is symmetric, hence $(\sqsupseteq_b^\bullet)^{*\vdash}$ is an $\mathcal{O}$-bisimulation. By Lemma 4.5.3, $\sqsupseteq_b^{\bullet *}$ is compatible, and by Lemma 4.5.1 it holds that $(\sqsupseteq_b^{\bullet *}) \subseteq (\sqsupseteq_b^\circ)$. Finally, by Lemma 4.4.2, it holds that $(\sqsupseteq_b^\circ) \subseteq (\sqsupseteq_b^\bullet) \subseteq (\sqsupseteq_b^{\bullet *})$, and hence $(\sqsupseteq_b^{\bullet *}) = (\sqsupseteq_b^\circ)$. We can conclude that $\sqsupseteq_b^\circ$ is compatible. □

In conclusion, we finish the proof of Theorem 3.3.8, the Compatibility Theorem.

Proof of Theorem 3.3.8. By Theorem 4.2.7, we know that the positive behavioural preorder $\sqsupseteq^+$ is equal to applicative $\mathcal{O}$-similarity, which is compatible by Theorem 4.5.2.

By Theorem 4.2.8, it holds that the behavioural equivalence $\equiv$ is equal to applicative $\mathcal{O}$-bisimilarity, which is compatible by Theorem 4.5.5. □

5

Logic variations

The formulas of the general logic $\mathcal{V}$ were chosen in such a way that the resulting logical equivalence, the behavioural equivalence, is compatible (see Theorem 3.3.8). We closed the formulas under countable disjunctions and conjunctions, not only to allow for more expressibility, but also to prove that the behavioural equivalence coincides with applicative bisimilarity. It turns out however, that the set of chosen formulas can be greatly reduced without changing the resulting logical equivalence. In this chapter we explore such different ways of simplifying the syntax of the logic as much as possible while maintaining its distinguishing power.

Other changes can also be made to the logic, like avoiding reference to program terms in formulas of function type, or expressing Hoare logic style formulas in the logic for global store. This chapter studies such variations of the logic, together with several combinations of effects for which we can find a suitable set of modalities. Since the behavioural equivalence coincides with applicative bisimilarity, the results in this chapter also provide different logical characterisations for applicative bisimilarity.

5.1 Eliminating computation formula connectives

In this section, we prove that excluding connectives for computation formulas does not change the induced logical equivalence. To prove this, we first establish a standard result from infinitary propositional logic.

A formula $\underset{\cdot\cdot}{\phi} \in \mathit{Form}(\underset{\cdot\cdot}{\mathbf{E}})$ is a series of disjunctions, conjunctions and negations of basic formulas. Let $B_{\underset{\cdot\cdot}{\phi}} \subseteq \mathit{Form}(\underset{\cdot\cdot}{\mathbf{E}})$ be the underlying set of basic formulas of $\underset{\cdot\cdot}{\phi}$. Specifically, $B_{\underset{\cdot\cdot}{\phi}} := \{\underset{\cdot\cdot}{\phi}\}$ if $\underset{\cdot\cdot}{\phi}$ is a basic formula, $B_{\neg(\underset{\cdot\cdot}{\phi})} := B_{\underset{\cdot\cdot}{\phi}}$, and both $B_{\bigvee X}$ and $B_{\bigwedge X}$ are equal to $\bigcup_{\underset{\cdot\cdot}{\phi} \in X} B_{\underset{\cdot\cdot}{\phi}}$. For any set of formulas X, we define $\neg X$ as the set of negations of formulas from X, specifically $\neg X := \{\neg(\underset{\cdot\cdot}{\phi}) \mid \underset{\cdot\cdot}{\phi} \in X\}$.

Lemma 5.1.1. *Any formula $\underset{\cdot\cdot}{\phi} \in \mathit{Form}(\underset{\cdot\cdot}{\mathbf{E}})$ can be written in disjunctive normal form $\bigvee_{i\in I} \bigwedge_{j\in J_i} \underset{\cdot\cdot}{\phi}_{i,j}$ where each $\underset{\cdot\cdot}{\phi}_{i,j} \in B_{\underset{\cdot\cdot}{\phi}} \cup \neg B_{\underset{\cdot\cdot}{\phi}}$, and in conjunctive normal form $\bigwedge_{i\in I} \bigvee_{j\in J_i} \underset{\cdot\cdot}{\phi}_{i,j}$ where each $\underset{\cdot\cdot}{\phi}_{i,j} \in B_{\underset{\cdot\cdot}{\phi}} \cup \neg B_{\underset{\cdot\cdot}{\phi}}$. If $\underset{\cdot\cdot}{\phi} \in \mathit{Form}(\underset{\cdot\cdot}{\mathbf{E}})_{\mathcal{V}^+}$, the above statements hold where each $\underset{\cdot\cdot}{\phi}_{i,j} \in B_{\underset{\cdot\cdot}{\phi}}$.*

Proof. We first prove the result for $\underline{\phi} \in \mathit{Form}(\underline{\mathbf{E}})_{\mathcal{V}^+}$. Let X be the set of terms $\underline{P}$ such that $\underline{P} \models \underline{\phi}$, and Y the set of terms $\underline{R}$ such that $\underline{P} \not\models \underline{\phi}$. Let $\underline{P} \in X$ and $\underline{R} \in Y$. If for all $\underline{\psi} \in B_{\underline{\phi}}$, $(\underline{P} \models \underline{\psi}) \Rightarrow (\underline{R} \models \underline{\psi})$, then by induction on $\underline{\phi}$, $\underline{R} \models \underline{\phi}$, which is a contradiction. So there is a formula $\underline{\psi}_{\underline{P},\underline{R}} \in B_{\underline{\phi}}$ such that $\underline{P} \models \underline{\psi}_{\underline{P},\underline{R}}$ and $\underline{R} \not\models \underline{\psi}_{\underline{P},\underline{R}}$. For such a choice of formulas, it holds that $\underline{\phi} \equiv \bigvee_{\underline{P} \in X} \bigwedge_{\underline{R} \in Y} \underline{\psi}_{\underline{P},\underline{R}} \equiv \bigwedge_{\underline{R} \in Y} \bigvee_{\underline{P} \in X} \underline{\psi}_{\underline{P},\underline{R}}$.

For $\underline{\phi} \in \mathit{Form}(\underline{\mathbf{E}})$, we take the same X and Y as above. Let $\underline{P} \in X$ and $\underline{R} \in Y$. If for all $\underline{\psi} \in B_{\underline{\phi}}$, $(\underline{P} \models \underline{\psi}) \Leftrightarrow (\underline{R} \models \underline{\psi})$, then by induction on $\underline{\phi}$, $\underline{R} \models \underline{\phi}$, which is a contradiction. So there is a formula $\underline{\psi}_{\underline{P},\underline{R}} \in B_{\underline{\phi}} \vee \neg B_{\underline{\phi}}$ such that $\underline{P} \models \underline{\psi}_{\underline{P},\underline{R}}$ and $\underline{R} \not\models \underline{\psi}_{\underline{P},\underline{R}}$. For such a choice of formulas, it holds that $\underline{\phi} \equiv \bigvee_{\underline{P} \in X} \bigwedge_{\underline{R} \in Y} \underline{\psi}_{\underline{P},\underline{R}} \equiv \bigwedge_{\underline{R} \in Y} \bigvee_{\underline{P} \in X} \underline{\psi}_{\underline{P},\underline{R}}$.

□

The above result is a special case of conjunctive normal form and disjunctive normal form results in infinitary propositional logic, which can be proven using a transfinite induction on the height of formulas.

This lemma allows us to put formulas in disjunctive or conjunctive normal form, without changing the used basic subformulas. The main use of this is during an induction on formulas, where the induction hypothesis gives us information about such basic subformulas. Inductions on formulas will be prevalent throughout this chapter.

Let us return to the main aim of this section, eliminating connectives from computation formulas.

Definition 5.1.2. We define the logic $\mathcal{L}^*$ as the fragment of the logic $\mathcal{L}$ which does not use connectives (disjunctions, conjunctions and negations) for computation formulas.

For example, basic formulas in the logic $\mathcal{L}^*$ of a function type $\mathbf{A} \to \mathbf{F}\,\mathbf{B}$ will always be of the shape $(V \mapsto o(\psi))$.

The following result exploits the disjunctive-conjunctive normal form result from Lemma 5.1.1 to eliminate conjunctions, disjunctions and negations from computation formulas.

Lemma 5.1.3. *Any $\underline{\phi} \in \mathcal{V}$ is equivalent to some formula of the form $\bigvee_{i \in I} \bigwedge_{j \in J_i} \underline{\psi}_{i,j}$ where each $\underline{\psi}_{i,j} \in \mathcal{V}^* \cup \neg\mathcal{V}^*$. Any $\underline{\phi} \in \mathcal{V}^+$ is equivalent to some formula of the form $\bigvee_{i \in I} \bigwedge_{j \in J_i} \underline{\psi}_{i,j}$ where each $\underline{\psi}_{i,j} \in (\mathcal{V}^+)^*$.*

Proof. We firstly prove the second statement by induction on formulas $\underline{\phi} \in \mathcal{V}^+$. Importantly, if $\underline{\phi}$ is a formula of a value type, then the desired $\bigvee_{i \in I} \bigwedge_{j \in J_i} \underline{\psi}_{i,j}$ is a formula from $(\mathcal{V}^+)^*$, so we can say that there is some $\underline{\phi}^* \in (\mathcal{V}^+)^*$ s.t. $\underline{\phi} \equiv \underline{\phi}^*$.

1. If $\underline{\phi} = \{n\}$, then $\underline{\phi} \in (\mathcal{V}^+)^*$, and note that $\underline{\phi} \equiv \bigvee \bigwedge \{\underline{\phi}\}$.

2. If $\underline{\phi} = (V \mapsto \underline{\phi}')$, then by induction hypothesis, $\underline{\phi}' = \bigvee_{i \in I} \bigwedge_{j \in J_i} \underline{\psi}'_{i,j}$ where $\underline{\psi}'_{i,j} \in (\mathcal{V}^+)^*$. So $\underline{\phi} \equiv (V \mapsto \bigvee_{i \in I} \bigwedge_{j \in J_i} \underline{\psi}'_{i,j}) \equiv \bigvee_{i \in I} \bigwedge_{j \in J_i} (V \mapsto \underline{\psi}'_{i,j})$, where $\underline{\psi}'_{i,j} \in (\mathcal{V}^+)^*$.

   The proof of the cases where $\underline{\phi} = (i \mapsto \underline{\psi})$ and $\underline{\phi} = \langle \underline{\phi}' \rangle$ goes similarly.

3. For $\underline{\phi} = \pi_0(\phi)$, then by the induction hypothesis, $\phi \equiv \psi$ where $\psi \in (\mathcal{V}^+)^*$. So $\underline{\phi} \equiv \pi_0(\psi) \in (\mathcal{V}^+)^*$.

   The proof of the case where $\underline{\phi} = \pi_1(\phi)$, $\underline{\phi} = (i, \phi)$ and $\underline{\phi} = o(\phi)$ goes similarly.

4. If $\underline{\phi} = \bigvee_{k\in K} \underline{\phi}_k$ or $\underline{\phi} = \bigwedge_{k\in K} \underline{\phi}_k$, apply the induction hypothesis on each $\underline{\phi}_k$ and use Lemma 5.1.1 to put the formula in disjunctive normal form, without changing the set of basic subformulas (which are from $(\mathcal{V}^+)^*$).

In the case of the full logic, we can repeat the proof used above with an induction on formulas $\underline{\phi} \in \mathcal{V}$, with two modifications. Firstly, in the first point, we may need to use the equivalence $(V \mapsto \neg(\underline{\psi}'')) \equiv \neg(V \mapsto \underline{\psi}'')$, and similarly for $(i \mapsto \underline{\psi})$ and $\langle \underline{\phi}' \rangle$, when necessary. Secondly, we have to add one more case:

5. If $\underline{\phi} = \neg(\underline{\phi}')$, then by the induction hypothesis, $\underline{\phi}' \equiv \bigvee_{i\in I} \bigwedge_{j\in J_i} \underline{\psi}'_{i,j}$ where $\underline{\psi}'_{i,j} \in \mathcal{V}^* \cup \neg\mathcal{V}^*$. So $\underline{\phi} \equiv \neg(\bigvee_{i\in I} \bigwedge_{j\in J_i} \underline{\psi}'_{i,j}) \equiv \bigwedge_{i\in I} \neg(\bigwedge_{j\in J_i} \underline{\psi}'_{i,j}) \equiv \bigwedge_{i\in I} \bigvee_{j\in J_i} \neg(\underline{\psi}'_{i,j})$. If $\underline{\psi}'_{i,j} \in \mathcal{V}^*$, then $\neg(\underline{\psi}'_{i,j}) \in \neg\mathcal{V}$. If $\underline{\psi}'_{i,j} \in \neg\mathcal{V}^*$, then $\underline{\psi}'_{i,j} = \neg(\underline{\psi}''_{i,j})$ where $\underline{\psi}''_{i,j} \in \mathcal{V}^*$, so $\neg(\underline{\psi}'_{i,j}) = \neg(\neg(\underline{\psi}''_{i,j})) \equiv \underline{\psi}''_{i,j}$. So we can use Lemma 5.1.1 to get the desired conclusion.

□

We can now establish the main result of this subsection. We have to be a bit careful though, since a logic without negation for computation formulas does not generally give a symmetric logical preorder. In particular, we prove in the next statement that $(\equiv_{\mathcal{V}}) = (\equiv_{\mathcal{V}^*})$, where moreover we know that $(\equiv_{\mathcal{V}}) = (\sqsubseteq_{\mathcal{V}})$, whereas $(\equiv_{\mathcal{V}^*})$ is not equal to $(\sqsubseteq_{\mathcal{V}^*})$ because of the lack of negation for computation formulas.

Corollary 5.1.4. *It holds that* $(\equiv_{\mathcal{V}}) = (\equiv_{\mathcal{V}^*})$ *and* $(\sqsubseteq_{\mathcal{V}^+}) = (\sqsubseteq_{(\mathcal{V}^+)^*})$.

Proof. We prove the result for $\mathcal{V}$, the proofs for the other case is similar.

If $\underline{P} \equiv_{\mathcal{V}^*} \underline{R}$ and $\underline{P} \models \underline{\phi}$ with $\underline{\phi} \in \mathcal{V}$, then by Lemma 5.1.3 $\underline{\phi} \equiv \bigvee_{i\in I} \bigwedge_{j\in J_i} \underline{\psi}_{i,j}$ where each $\underline{\psi}_{i,j} \in \mathcal{V}^* \cup \neg\mathcal{V}^*$. Since $\underline{P} \equiv_{\mathcal{V}^*} \underline{R}$, it holds that $\underline{P} \models \underline{\psi}_{i,j} \iff \underline{R} \models \underline{\psi}_{i,j}$. Now, there is an $i \in I$ such that for all $j \in J_i$, $\underline{P} \models \underline{\psi}_{i,j}$ hence $\underline{R} \models \underline{\psi}_{i,j}$. We conclude that $\underline{R} \models \underline{\phi}$, so $\underline{P} \sqsubseteq_{\mathcal{V}} \underline{R}$, and hence $\underline{P} \equiv_{\mathcal{V}} \underline{R}$.

The reverse implication holds because $\mathcal{V}^* \subseteq \mathcal{V}$. □

5.2 Infinitary vs finitary value formula connectives

In the logic $\mathcal{V}^*$, like in the full logic $\mathcal{V}$, we have infinitary conjunctions, infinitary disjunctions and negations for value formulas. In this section, we study what other even simpler logics still induce the behavioural preorders $\sqsubseteq^+$ and $\equiv$. In particular, we study in which cases we can use finitary instead of infinitary disjunctions and conjunctions, and in which cases we may remove disjunctions, conjunctions and negations entirely. To better keep track of all the variations of the logic, we define a logic with a quadruple $(\mathcal{O}, a, b, c)$, consisting of the following four elements designating the inclusion of certain formula constructors. These are in order:

1. A set of modalities $\mathcal{O}$.
2. A symbol a for the maximum size of disjunction for value formulas, with $\bigvee$ for countable, $\vee$ for finite and $\bot$ for empty only.
3. A symbol b for the maximum size of conjunction for value formulas, with $\bigwedge$ for countable, $\wedge$ for finite and $\top$ for empty only.
4. A symbol c given by $\neg$ or $+$, respectively for including or excluding negations for value formulas.

More formally, we define $(\mathcal{O}, a, b, c)$ as follows:

Definition 5.2.1. The logic $(\mathcal{O}, a, b, c)$ is the logic inductively defined using the following rules of Figure 3.1:

1. Rules (1) to (6), and rule (8).
2. Rule (7) with modalities from $\mathcal{O}$.
3. Rule (9) for value types over all countable sets X if $a = \bigvee$, only over finite sets X if $a = \vee$, and only over the empty set $X = \emptyset$ only if $a = \bot$.
4. Rule (10) for value types over all countable sets X if $b = \bigwedge$, only over finite sets X if $b = \wedge$, and only over the empty set $X = \emptyset$ only if $b = \top$.
5. Rule (11) over value types only if $c = \neg$.

We will exclude connectives for computation formulas, since their exclusion does not change the logical equivalences according to Corollary 5.1.4. If we define $(\mathcal{L})^*$ as in Definition 5.1.2, then $(\mathcal{O}, \bigvee, \bigwedge, \neg)$ corresponds to $(\mathcal{V})^*$, and $(\mathcal{O}, \bigvee, \bigwedge, +)$ corresponds to $(\mathcal{V}^+)^*$.

5.2.1 Conditions for simplifying value formula connectives

We will further reduce the logic, given sufficient conditions. The goal of this subsection is to establish properties of modalities which can be used to simplify the logic in several ways. Mainly, we aim to replace arbitrary conjunctions and disjunctions with finite or even empty conjunctions and disjunctions. However, which simplifications are possible depend greatly on the effects and their modalities.

Besides Scott openness, we consider three more properties of modalities which can be used to simplify the logic.

Definition 5.2.2. A modality $o \in \mathcal{O}$ *distributes over non-empty disjunctions* if for any non-empty countable set of formulas $X \subseteq \mathit{Form}(\mathbf{A})$, $o(\bigvee X) \equiv \bigvee\{o(\phi) \mid \phi \in X\}$.

Definition 5.2.3. A modality $o \in \mathcal{O}$ *distributes over non-empty conjunctions* if for any non-empty countable set of formulas $X \subseteq \mathit{Form}(\mathbf{A})$, $o(\bigwedge X) \equiv \bigwedge\{o(\phi) \mid \phi \in X\}$.

Definition 5.2.4. A set of modalities $\mathcal{O}$ *allows for negation propagation* if for any $o \in \mathcal{O}$ and any formula $\phi \in Form(\mathbf{A})$, $o(\neg(\phi))$ is equivalent to a disjunction of conjunctions of elements from the set:

$$\{o'(\phi),\ o'(\bot),\ o'(\top),\ \neg(o'(\phi)),\ \neg(o'(\bot)),\ \neg(o'(\top)) \quad \mid\ o' \in \mathcal{O}\}.$$

We look at the modalities for our examples, and check which properties they satisfy.

Effect-free: The termination modality $\downarrow$distributes over both non-empty disjunctions and non-empty conjunctions, since $\underline{M} \models\downarrow(\phi)$ holds if and only if $|\underline{M}| = \langle V\rangle$ and $V \models \phi$. So with $|\underline{M}| = \langle V\rangle$, $\underline{M} \models\downarrow(\bigwedge X) \Leftrightarrow V \models \bigwedge X \Leftrightarrow \forall\phi \in X.V \models \phi \Leftrightarrow \underline{M} \models \bigwedge\{\downarrow(\phi) \mid \phi \in X\}$, and similarly for $\bigvee X$. Moreover, $\mathcal{O}_\emptyset = \{\downarrow\}$ allows for negation propagation since $\downarrow(\neg(\phi)) \quad \equiv \quad \neg(\downarrow(\phi)) \wedge \downarrow(\top)$.

Error: The error modalities E_e trivially distribute over non-empty disjunctions and conjunctions since $\mathsf{E}_e(\phi) \equiv \mathsf{E}_e(\bot)$, and hence $\mathcal{O}_{\text{er}}$ also allows for negation propagation since $\mathsf{E}_e(\neg(\phi)) \quad \equiv \quad \mathsf{E}_e(\bot)$.

Nondeterminism: The may modality $\Diamond$ distributes over non-empty disjunctions, since $\underline{M} \models \Diamond(\bigvee X)$ holds if and only if $|\underline{M}|$ has a leaf $\langle V\rangle$ such that there is a formula $\phi \in X$ where $V \models \phi$, which holds precisely if $\underline{M} \models \bigvee\{\Diamond(\phi) \mid \phi \in X\}$. The must modality $\Box$ distributes over non-empty conjunctions, since $\underline{M} \models \Box(\bigwedge X)$ holds if and only if $|\underline{M}|$ if finite and all leaves are values V where for any formula $\phi \in X$ it holds that $V \models \phi$. This holds precisely if $\underline{M} \models \bigwedge\{\Box(\phi) \mid \phi \in X\}$. None of the sets of modalities for nondeterminism allow for negation propagation though.

Probability: The set of modalities $\mathcal{O}_{\text{pr}}$ allows for negation propagation since $\underline{M} \models \mathsf{P}_{>q}(\neg(\phi)) \equiv \bigvee\{\mathsf{P}_{>p}(\top) \wedge \neg(\mathsf{P}_{>(p-q)}(\phi)) \mid p \geq q\}$. But most modalities neither distribute over non-empty disjunctions nor over non-empty conjunctions.

Global store: The modalities $(s \rightarrowtail s')$ distribute over non-empty disjunctions and conjunctions since $\underline{M} \models (s \rightarrowtail s')(\phi)$ holds if and only if $exec(|\underline{M}|, s) = (V, s')$ where $V \models \phi$. So we can unfold the conjunctions and disjunctions at V. Moreover, $\mathcal{O}_{\text{gs}}$ allows for negation propagation since $(s \rightarrowtail r)(\neg(\phi)) \quad \equiv \quad \neg((s \rightarrowtail r)(\phi)) \wedge (s \rightarrowtail r)(\top)$.

Input/output: The modalities $\langle w\rangle\downarrow$ distribute over non-empty disjunctions and conjunctions since like in global store, $\underline{M} \models \langle w\rangle\downarrow(\phi)$ depends on the satisfaction $V \models \phi$ for a particular leaf V of $|\underline{M}|$. The modalities $\langle w\rangle_{...}$ trivially distribute over non-empty disjunctions and conjunctions since $\langle w\rangle_{...}(\phi) \equiv \langle w\rangle_{...}(\bot)$. Lastly, the set $\mathcal{O}_{\text{io}}$ allows for negation propagation since $\langle w\rangle\downarrow(\neg(\phi)) \quad \equiv \quad \neg(\langle w\rangle\downarrow(\phi)) \wedge \langle w\rangle\downarrow(\top)$ and $\langle w\rangle_{...}(\neg(\phi)) \quad \equiv \quad \langle w\rangle_{...}(\bot)$.

Timer: All the modalities for timer distribute over non-empty disjunction and conjunctions for similar reasons as the modalities of input/output, and any of the sets of modalities for timer allow for negation propagation since:

- $\mathsf{C}_{\leq c}(\neg(\phi)) \equiv \mathsf{C}_{\leq c}(\top) \wedge \neg(\mathsf{C}_{\leq c}(\phi))$.
- $\mathsf{C}_{\geq c}(\neg(\phi)) \equiv \mathsf{C}_{\geq c}(\top) \wedge \neg(\mathsf{C}_{\geq c}(\phi))$.
- $\mathsf{C}^{\uparrow}_{>c}(\neg(\phi)) \equiv \mathsf{C}^{\uparrow}_{>c}(\bot)$.

5.2.2 Connective elimination results

Using the properties defined in the previous subsection, we prove how the logics $(\mathcal{V})^*$ and $(\mathcal{V}^+)^*$ can be reduced without changing the induced logical equivalences.

Lemma 5.2.5. *Suppose $\mathcal{O}$ is a set of modalities such that all $o \in \mathcal{O}$ distribute over non-empty disjunctions, then $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,\neg)}) = (\sqsubseteq_{(\mathcal{O},\bot,\bigwedge,\neg)})$ and $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,+)}) = (\sqsubseteq_{(\mathcal{O},\bot,\bigwedge,+)})$.*

Proof. Let $c \in \{\neg, +\}$, and take $\mathcal{L} = (\mathcal{O}, \bigvee, \bigwedge, c)$ and $\mathcal{K} = (\mathcal{O}, \bot, \bigwedge, c)$. We prove that each formula $\underset{\sim}{\phi} \in \mathcal{L}$ (value or computation) is equivalent to a non-empty disjunction $\bigvee_{i\in I} \underset{\sim}{\psi}_i$ over formulas $\underset{\sim}{\psi}_i$ from $\mathcal{K}$. We prove this by induction on $\underset{\sim}{\phi}$.

1. If $\underset{\sim}{\phi} = \{n\}$, then $\underset{\sim}{\phi} \in \mathcal{K}$ and $\underset{\sim}{\phi} \equiv \bigvee\{\underset{\sim}{\phi}\}$, so we are finished.
2. If $\underset{\sim}{\phi} = \langle\underset{\sim}{\phi}\rangle$, then by induction hypothesis, $\underset{\sim}{\phi} \equiv \bigvee_{i\in I} \underset{\sim}{\phi}_i$ where all $\underset{\sim}{\phi}_i \in \mathcal{K}$. Since $\langle\bigvee_{i\in I} \underset{\sim}{\phi}_i\rangle \equiv \bigvee_{i\in I}\langle\underset{\sim}{\phi}_i\rangle$, we are finished. For $\underset{\sim}{\phi} \in \{\pi_0(\psi), \pi_1(\phi), (V \mapsto \underset{\sim}{\phi}), (j \mapsto \underset{\sim}{\phi}), o(\psi)\}$, the proof goes similarly.
3. If $\underset{\sim}{\phi} = \bot$ or $\underset{\sim}{\phi} = \top$, note that $\underset{\sim}{\phi} \equiv \bigvee\{\underset{\sim}{\phi}\}$, which is of the correct shape.
4. If $\underset{\sim}{\phi} = \bigvee_{j\in J} \underset{\sim}{\phi}_j$ with J non-empty, we use the induction hypothesis and merge the two disjunctions.
5. If $\underset{\sim}{\phi} = \bigwedge_{j\in J} \underset{\sim}{\phi}_j$ with J non-empty, we use the induction hypothesis to see that $\underset{\sim}{\phi} \equiv \bigwedge_{j\in J} \bigvee_{i\in I_j} \underset{\sim}{\psi}_{i,j}$. We use Lemma 5.1.1 to put the formula in disjunctive normal form, and note that countable conjunctions over $\mathcal{K}$-formulas are still a $\mathcal{K}$-formulas.
6. If $\underset{\sim}{\phi} = \neg(\underset{\sim}{\psi})$, in which case $c = \neg$, we use the induction hypothesis and note that $\neg(\bigvee_{i\in I} \underset{\sim}{\phi}_i) \equiv \bigwedge_{i\in I} \neg(\underset{\sim}{\phi}_i) \equiv \bigvee\{\bigwedge_{i\in I} \neg(\underset{\sim}{\phi}_i)\}$, where $\bigwedge_{i\in I} \neg(\underset{\sim}{\phi}_i) \in \mathcal{K}$ (since all $\underset{\sim}{\phi}_i \in \mathcal{K}$).

Satisfaction of disjunctions is completely determined by satisfaction of the subformulas of the disjunction. So since moreover $\mathcal{K} \subseteq \mathcal{L}$, we conclude that $(\sqsubseteq_{\mathcal{L}}) = (\sqsubseteq_{\mathcal{K}})$.

□

Not all modalities distribute over disjunctions though. However, we can also reduce the used size of disjunctions given that all the modalities are Scott open.

Lemma 5.2.6. *Suppose $\mathcal{O}$ is a set of Scott-open modalities.*
Then $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,\neg)}) = (\sqsubseteq_{(\mathcal{O},\vee,\bigwedge,\neg)})$ and $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,+)}) = (\sqsubseteq_{(\mathcal{O},\vee,\bigwedge,+)})$.

Proof. A similar proof to that of Lemma 5.2.5, only changing the case where $\underline{\phi} = o(\psi)$. Because of Scott continuity, $o(\bigvee_{n\in\mathbb{N}} \psi_n) \equiv \bigvee_{m\in\mathbb{N}} o(\bigvee_{n\in\mathbb{N},n<m} \psi_n)$, where $\bigvee_{n\in\mathbb{N},n<m} \psi_n$ is a finite disjunction, and hence the case goes through. □

We can dualize Lemma 5.2.5, adapting the proof accordingly.

Lemma 5.2.7. *Suppose $\mathcal{O}$ is a set of modalities such that all $o \in \mathcal{O}$ distribute over non-empty conjunctions, then $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,\neg)}) = (\sqsubseteq_{(\mathcal{O},\bigvee,\top,\neg)})$ and similarly $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,+)}) = (\sqsubseteq_{(\mathcal{O},\bigvee,\top,+)})$.*

Lastly, we can combine the two results to get rid of both connectives, if possible.

Lemma 5.2.8. *Suppose $\mathcal{O}$ is a set of modalities such that all $o \in \mathcal{O}$ we distribute over non-empty disjunctions and conjunctions, then $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,\neg)}) = (\sqsubseteq_{(\mathcal{O},\bot,\top,\neg)})$ and similarly $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,+)}) = (\sqsubseteq_{(\mathcal{O},\bot,\top,+)})$.*

Proof. Let $c \in \{\neg, +\}$, and take $\mathcal{L} = (\mathcal{O}, \bigvee, \bigwedge, c)$ and $\mathcal{K} = (\mathcal{O}, \bot, \top, c)$. We prove by induction that each formula $\underline{\phi} \in \mathcal{L}$ (value or computation) is equivalent to a non-empty disjunction of non-empty conjunctions $\bigvee_{i\in I} \bigwedge_{j\in J_i} \underline{\psi}_i$ over formulas $\underline{\psi}_i$ from $\mathcal{K}$. Note that $\underline{\phi} \equiv \bigvee\{\bigwedge\{\underline{\phi}\}\}$ to deal with the cases where $\underline{\phi} \in \{\top, \bot, \{n\}\}$. The rest of the proof is similar to the proof of Lemma 5.2.5, requiring further transformations of formulas into their disjunctive normal form where appropriate, using Lemma 5.1.1. □

In case that moreover, the set of modalities allows for negation propagation, we can also get rid of negation:

Lemma 5.2.9. *Suppose $\mathcal{O}$ is a set of modalities which allows for negation propagation, and such that all $o \in \mathcal{O}$ distribute over non-empty disjunctions and conjunctions, then $(\equiv_{(\mathcal{O},\bigvee,\bigwedge,\neg)}) = (\equiv_{(\mathcal{O},\top,\bot,+)})$.*

Proof. Take $\mathcal{L} = (\mathcal{O}, \top, \bot, \neg)$ and $\mathcal{K} = (\mathcal{O}, \bigvee, \bigwedge, +)$. We prove that each formula $\underline{\phi} \in \mathcal{L}$ (value or computation) is equivalent to $\bigvee_{i\in I} \bigwedge_{j\in J_i} \underline{\psi}_i$ (all index sets non-empty), such that each $\underline{\psi}_i$ is from $\mathcal{K} \cup \neg\mathcal{K}$. This will be sufficient to derive the equality of logical equivalences. We prove the statement by induction on $\underline{\phi}$.

1. If $\underline{\phi} = \{n\}$, then $\underline{\phi} \in \mathcal{K}$ and $\underline{\phi} \equiv \bigvee\{\bigwedge\{\underline{\phi}\}\}$, so we are finished.

2. If $\underline{\phi} = \langle\underline{\phi}\rangle$, then by induction hypothesis, $\underline{\phi} \equiv \bigvee_{i\in I} \bigwedge_{j\in J_i} \underline{\phi}_{i,j}$ with $\underline{\phi}_{i,j} \in \mathcal{K} \cup \neg\mathcal{K}$. Since $\langle \bigvee_{i\in I} \bigwedge_{j\in J_i} \underline{\phi}_{i,j}\rangle \equiv \bigvee_{i\in I} \bigwedge_{j\in J_i} \langle\underline{\phi}_{i,j}\rangle$, and for any $\underline{\psi}$; $\langle\neg(\underline{\psi})\rangle \equiv \neg(\langle\underline{\psi}\rangle)$, we get the desired result. For $\underline{\phi} \in \{\pi_0(\phi), \pi_1(\phi), (V \mapsto \underline{\phi}), (j \mapsto \underline{\phi})\}$, the proof is similar.

3. If $\underline{\phi} = o(\psi)$, use the induction hypothesis to find $\psi \equiv \bigvee_{i\in I} \bigwedge_{j\in J_i} \psi_{i,j}$. So $o(\bigvee_{i\in I} \bigwedge_{j\in J_i} \psi_{i,j}) \equiv \bigvee_{i\in I} \bigwedge_{j\in J_i} o(\psi_{i,j})$ where $\psi_{i,j} \in \mathcal{K} \cup \neg\mathcal{K}$. For $\psi \in \mathcal{K}$, by assumption $o(\neg(\psi))$ is equivalent to some disjunction of conjunctions of elements of $\mathcal{K}$. So (using Lemma 5.1.1) $\underline{\phi}$ is equivalent to a formula of the right shape.

4. If $\underline{\phi} \in \{\top, \bot\}$, then $\underline{\phi} \equiv \bigvee\{\bigwedge\{\underline{\phi}\}\}$.

5. If $\underset{\cdot}{\phi} = \neg(\underset{\cdot}{\psi})$, we can use the usual connective manipulation to get the right result.

□

There is a different way to remove negations, even if the set of modalities does not allow for negation propagation (e.g., in case of nondeterminism). This can be done by replacing them with negative modalities. Given $o \in \mathcal{O}$, we define $o_\neg$ to be the modality where $[\![o_\neg]\!] = (T(\mathbf{1}) - [\![o]\!])$. Hence $\underline{M} \models o_\neg(\phi) \iff \underline{M} \not\models o(\phi)$. Let $\mathcal{O}_\neg = \{o_\neg \mid o \in \mathcal{O}\}$. Note that the new modalities $o_\neg$ are not generally monotone.

Lemma 5.2.10. *The relations* $(\equiv_{(\mathcal{O},\bigvee,\bigwedge,\neg)})$ *and* $(\equiv_{(\mathcal{O}\cup\mathcal{O}_\neg,\bigvee,\bigwedge,+)})$ *are the same. Moreover, if we add* $\neg(\{n\})$ *as basic formulas to* $Form(\mathbf{N})$*, then:*

$$(\equiv_{(\mathcal{O},\vee,\wedge,\neg)}) = (\equiv_{(\mathcal{O}\cup\mathcal{O}_\neg,\vee,\wedge,+)}) \quad \textit{and} \quad (\equiv_{(\mathcal{O},\bot,\top,\neg)}) = (\equiv_{(\mathcal{O}\cup\mathcal{O}_\neg,\bot,\top,+)}) \ .$$

Note that in all cases of the formulation of the lemma, the arity of disjunctions and conjunctions match. This is due to the seventh item in the proof.

Proof. Let $(a, b) \in \{(\bigvee, \bigwedge), (\vee, \wedge), (\bot, \top)\}$, $\mathcal{L} = (\mathcal{O}, a, b, \neg)$, and $\mathcal{K} = (\mathcal{O} \cup \mathcal{O}_\neg, a, b, +)$, where we add $\neg(\{n\})$ as basic formulas to $\mathcal{K}$ if $a \neq \bigvee$. We do two separate inductions on formulas.

First we prove that for each formula $\underset{\cdot}{\phi} \in \mathcal{K}$, there is a formula $\underset{\cdot}{\psi} \in \mathcal{K}$ such that $\neg(\underset{\cdot}{\phi}) \equiv \underset{\cdot}{\psi}$.

1. If $\underset{\cdot}{\phi} = \{n\}$, then $\neg(\underset{\cdot}{\phi}) \equiv \bigvee_{m\in\mathbb{N},m\neq n}\{m\}$, which is in $\mathcal{K}$ in the case that we have countable disjunction. In the other cases, with at most finite or empty disjunctions and conjunctions, we added $\neg(\{n\})$ as a basic formula to $\mathcal{K}$.

2. If $\underset{\cdot}{\phi} = \neg(\{n\})$ as basic formula, then $\neg(\underset{\cdot}{\phi}) \equiv \{n\} \in \mathcal{K}$.

3. If $\underset{\cdot}{\phi} = \langle\underline{\phi}\rangle$, then by induction hypothesis there is a $\underline{\psi} \in \mathcal{K}$ such that $\neg(\underline{\phi}) \equiv \underline{\psi}$. Hence $\neg(\underset{\cdot}{\phi}) = \neg(\langle\underline{\phi}\rangle) \equiv \langle\neg(\underline{\phi})\rangle \equiv \langle\underline{\psi}\rangle \in \mathcal{K}$.

4. For $\underset{\cdot}{\phi} = \pi_0(\psi)$, $\underset{\cdot}{\phi} = \pi_1(\psi)$, $\underset{\cdot}{\phi} = (i, \underset{\cdot}{\psi})$, $\underset{\cdot}{\phi} = (V \mapsto \underline{\phi})$ and $\underset{\cdot}{\phi} = (i \mapsto \underline{\phi})$, the proof goes similarly as in the previous case.

5. If $\underset{\cdot}{\phi} = o(\psi)$ where $o \in \mathcal{O}$, then $\neg(\underset{\cdot}{\phi}) \equiv o_\neg(\psi) \in \mathcal{K}$.

6. If $\underset{\cdot}{\phi} = o_\neg(\psi)$ where $o \in \mathcal{O}$, then $\neg(\underset{\cdot}{\phi}) \equiv o(\psi) \in \mathcal{K}$.

7. If $\underset{\cdot}{\phi} = \bigvee_{i\in I} \underset{\cdot}{\phi}_i$, then by induction hypothesis there is for each $i \in I$ a formula $\underset{\cdot}{\psi}_i \in \mathcal{K}$ such that $\neg(\underset{\cdot}{\phi}_i) \equiv \underset{\cdot}{\psi}_i$. Hence $\neg(\underset{\cdot}{\phi}) = \neg(\bigvee_{i\in I} \underset{\cdot}{\phi}_i) \equiv \bigwedge_{i\in I} \neg(\underset{\cdot}{\phi}_i) \equiv \bigwedge_{i\in I} \underset{\cdot}{\psi}_i \in \mathcal{K}$. The case where $\underset{\cdot}{\phi} = \bigwedge_{i\in I} \underset{\cdot}{\phi}_i$ has a dual proof. The above derivation works even if I is empty.

8. If $\underset{\cdot}{\phi} = \neg(\underset{\cdot}{\psi})$, then $\neg(\underset{\cdot}{\phi}) = \neg(\neg(\underset{\cdot}{\psi})) \equiv \underset{\cdot}{\psi} \in \mathcal{K}$.

We now do an induction on $\underset{\cdot}{\phi} \in \mathcal{L}$, to prove that for each such $\underset{\cdot}{\phi}$ there is an equivalent formula $\underset{\cdot}{\psi} \in \mathcal{K}$. In this induction, all cases except the $\underset{\cdot}{\phi} = \neg(\underset{\cdot}{\psi})$ case for $\underset{\cdot}{\phi}$ follow directly, so we only need to prove that case explicitly. If $\underset{\cdot}{\phi} = \neg(\underset{\cdot}{\psi})$, then by induction hypothesis, there is a $\underset{\cdot}{\psi}' \in \mathcal{K}$ such that $\underset{\cdot}{\phi} \equiv \neg(\underset{\cdot}{\psi}) \equiv \neg(\underset{\cdot}{\psi}')$. By the previous induction, there is a formula $\underset{\cdot}{\phi}' \in \mathcal{K}$ such that $\neg(\underset{\cdot}{\psi}') \equiv \underset{\cdot}{\phi}'$, hence $\underset{\cdot}{\phi} \equiv \underset{\cdot}{\phi}'$, and we are finished.

For the other direction, note that $o_\neg(\phi) \equiv \neg(o(\phi))$. So any formula $\underset{\cdot}{\phi} \in \mathcal{K}$ is equivalent to a formula or the negation of a formula in $\mathcal{L}$ (remember that there are no connectives at computation formulas). We can conclude that each formula from $\mathcal{L} \cup \neg\mathcal{L}$ has an equivalent formula from $\mathcal{K}$, and vice versa. So they induce the same logical equivalence on the terms. □

5.2.3 Logic characterisations for effect examples

Combining the properties of the modalities for our effect examples, and the results from the previous subsection, we can reduce the logic in various ways without changing the induced behavioural equivalence.

Effect-free, error, global store, input/output and timer
Firstly, most examples of effects have a set of modalities which both allow for negation propagation, and contain modalities which distribute both over non-empty disjunctions and over non-empty conjunctions.

Proposition 5.2.11. *For effect-free computation, error effect, global store effect, input/output effect, and timer effect, the general behavioural equivalence $\equiv$ (and hence applicative $\mathcal{O}$-bisimilarity) is equal to $\equiv_{(\mathcal{O},\bot,\top,+)}$, where $\mathcal{O}$ is the chosen set of modalities for each effect.*

Proof. As observed before, the modalities allow for negation propagation and distribute over non-empty conjunctions and disjunctions. So we can apply Lemma 5.2.9 to get the desired result. □

Since applicative bisimilarity is equal to $\equiv$, and mutual applicative similarity is equal to $\equiv^+$, we have the following consequence.

Corollary 5.2.12. *For effect-free computation, error effect, global store effect, input/output effect, and timer effect, with the given set of modalities $\mathcal{O}$, applicative $\mathcal{O}$-bisimilarity is equal to mutual applicative $\mathcal{O}$-similarity.*

The same can be said about some combinations of effects. In Section 5.3, we will study which combinations of effects have an appropriate decomposable set of Scott open modalities. We conjecture that if it is a combination containing only effects from the three mentioned in Proposition 5.2.11, the logic can likewise be reduced to $(\mathcal{O}, \bot, \top, +)$ without changing the logical equivalence.

Nondeterminism

Unfortunately, the modalities for nondeterminism do not have many properties, except for the fact that $\Diamond$ distributes over non-empty disjunctions and $\Box$ distributes over non-empty conjunctions. So we can use Lemmas 5.2.5, 5.2.7 and 5.2.10 to derive that:

$$\equiv_{(\{\Diamond\},\bigvee,\bigwedge,\neg)} \;=\; \equiv_{(\{\Diamond\},\bot,\bigwedge,\neg)} \;=\; \equiv_{(\{\Diamond,\Diamond_\neg\},\bigvee,\bigwedge,+)} \quad ,$$

$$\equiv_{(\{\Box\},\bigvee,\bigwedge,\neg)} \;=\; \equiv_{(\{\Box\},\bigvee,\top,\neg)} \;=\; \equiv_{(\{\Box,\Box_\neg\},\bigvee,\bigwedge,+)} \quad ,$$

$$\equiv_{(\{\Diamond,\Box\},\bigvee,\bigwedge,\neg)} \;=\; \equiv_{(\{\Diamond,\Diamond_\neg,\Box,\Box_\neg\},\bigvee,\bigwedge,+)} \quad .$$

Probability

The probabilistic logic can be greatly simplified. The proofs however are a bit more involved, since the modalities do not simply distribute over the connectives. As such, this result is not a consequence of previously established lemmas.

Proposition 5.2.13. $(\equiv_{(\mathcal{O}_{pr},\bigvee,\bigwedge,\neg)}) = (\equiv_{(\mathcal{O}_{pr},\bot,\wedge,+)})$ *holds.*

Proof. Let $\mathcal{L} = (\mathcal{O}_{\mathrm{pr}}, \bigvee, \bigwedge, \neg)$ and $\mathcal{K} = (\mathcal{O}_{\mathrm{pr}}, \bot, \wedge, +)$. We prove by induction that each $\phi \in \mathcal{L}$ is equivalent to $\bigvee_{i\in I}\bigwedge_{j\in J_i}\phi_{i,j}$ (with non-empty indexing sets) where $\phi_{i,j} \in \mathcal{K} \cup \neg\mathcal{K}$. The proof is similar to the proof of Lemma 5.2.9. There is only one non-trivial case.

Assume $\phi = \mathsf{P}_{>q}(\psi)$, then by induction hypothesis, $\psi \equiv \bigvee_{i\in I}\bigwedge_{j\in J_i}\phi_{i,j}$ where $\phi_{i,j} \in \mathcal{K} \cup \neg\mathcal{K}$. In five steps, we will show that ϕ is equivalent to a series of conjunctions, disjunctions and negations of formulas of the form $\mathsf{P}_{>q}(\psi)$ (which we will call a *combination* of formulas $\mathsf{P}_{>q}(\psi)$), where ψ is of an increasingly simpler shape. At this stage, $\psi = \bigvee_{i\in I}\bigwedge_{j\in J_i}\phi_{i,j}$ where $\phi_{i,j} \in \mathcal{K} \cup \neg\mathcal{K}$, but each step we will attempt to simplify the shape of ψ by 'unfolding' the connectives out of the modalities $\mathsf{P}_{>q}$.

(I) We prove that ψ can be of the shape $\vee_{i\in I}\bigwedge_{j\in J_i}\phi_{i,j}$ with I finite and $\phi_{i,j} \in \mathcal{K}\cup\neg\mathcal{K}$. Since $\mathsf{P}(|\underline{M}|[\models \bigvee_{n\in\mathbb{N}}\phi_n]) = \lim_{m\to\infty}\mathsf{P}(|\underline{M}|[\models \bigvee_{n\in\mathbb{N},n<m}\phi_n])$ we observe the equivalence $\mathsf{P}_{>q}(\bigvee_{n\in\mathbb{N}}\psi_n) \equiv \bigvee_{m\in\mathbb{N}}\mathsf{P}_{>q}(\bigvee_{n\in\mathbb{N},n<m}\psi_n)$, where $\bigvee_{n\in\mathbb{N},n<m}$ is a finite disjunction. So we can show that ϕ is equivalent to a combination of formulas of the form $\mathsf{P}_{>q}(\vee_{i\in I}\bigwedge_{j\in J_i}\phi_{i,j})$ with $\phi_{i,j} \in \mathcal{K} \cup \neg\mathcal{K}$.

(II) We prove that ψ can be of the shape $\bigwedge_{j\in J}\phi_j$ where $\phi_j \in \mathcal{K} \cup \neg\mathcal{K}$.
Since $\mathsf{P}(|\underline{M}|[\models \phi\vee\psi]) = \mathsf{P}(|\underline{M}|[\models \phi]) + \mathsf{P}(|\underline{M}|[\models \psi]) - \mathsf{P}(|\underline{M}|[\models \phi\wedge\psi])$, we observe $\mathsf{P}_{>q}(\psi \vee \psi') \equiv \bigvee_{a,b,c\in\mathbb{Q},a+b\geq q+c}\mathsf{P}_{>a}(\psi) \wedge \mathsf{P}_{>b}(\psi') \wedge \neg(\mathsf{P}_{>c}(\psi \wedge \psi'))$. Using that $\psi\wedge(\psi'\vee\psi'') \equiv (\psi\wedge\psi')\vee(\psi\wedge\psi'')$, we can unfold the finite disjunction by applying the above rule a finite number of times. So ϕ is equivalent to a combination of formulas $\mathsf{P}_{>q}(\bigwedge_{j\in J}\phi_j)$ with $\phi_j \in \mathcal{K} \cup \neg\mathcal{K}$.

(III) We prove that ψ can be of the shape $\wedge_{j\in J}\phi_j$ with J finite and $\phi_j \in \mathcal{K} \cup \neg\mathcal{K}$.
Since $\mathsf{P}(|\underline{M}|[\models \bigwedge_{n\in\mathbb{N}}\phi_n]) = \lim_{m\to\infty}\mathsf{P}(|\underline{M}|[\models \bigwedge_{n\in\mathbb{N},n<m}\phi_n])$, we observe the fact that $\mathsf{P}_{>q}(\bigwedge_{n\in\mathbb{N}}\psi_n) \equiv \bigvee_{a\in\mathbb{Q},a>q}\bigwedge_{m\in\mathbb{N}}\mathsf{P}_{>a}(\bigwedge_{n\in\mathbb{N},n<m}\psi_n)$, where $\bigwedge_{n\in\mathbb{N},n<m}$ is a finite disjunction (we use disjunction over $a > q$ to avoid satisfying $\mathsf{P}_{\geq q}$ instead

of $\mathsf{P}_{>q}$). So $\underset{\sim}{\phi}$ is equivalent to a combination of formulas $\mathsf{P}_{>q}(\wedge_{j\in J}\phi_j)$ where J is finite and $\phi_j \in \mathcal{K} \cup \neg\mathcal{K}$.

(IV) We prove that ψ can be $\vee_{i\in I}\wedge_{j\in J_i}\phi_{i,j}$ with I and J_i finite and $\phi_{i,j} \in \mathcal{K}$.
The formula $\wedge_{j\in J}\phi_j$ with $\phi_j \in \mathcal{K} \cup \neg\mathcal{K}$ can be written in the form of $\phi_1 \wedge \cdots \wedge \psi_n \wedge \neg\psi'_1 \wedge \cdots \wedge \neg\psi'_m$ with all formula ϕ_i and ψ'_j from $\mathcal{K}$. This is equivalent to $\alpha \wedge \neg\beta$ where $\alpha = (\psi_1 \wedge \cdots \wedge \psi_n)$ and $\beta = (\psi'_1 \vee \cdots \vee \psi'_m)$. Since $\mathsf{P}(|\underline{M}|[\models \alpha \wedge \neg(\beta)] = \mathsf{P}(|\underline{M}|[\models \alpha] - \mathsf{P}(|\underline{M}|[\models \alpha \wedge \beta]$, we observe that $\mathsf{P}_{>q}(\alpha \wedge \neg(\beta)) \equiv \bigvee_{a,b\in\mathbb{Q},a\geq q+b} \mathsf{P}_{>a}(\alpha) \wedge \neg\mathsf{P}_{>b}(\alpha \wedge \beta)$. Now, $\alpha = (\psi_1 \wedge \cdots \wedge \psi_n)$, and $\alpha \wedge \beta$ is by distributivity equivalent to a finite disjunction of finite conjunction of formulas from $\mathcal{K}$. So $\underset{\sim}{\phi}$ is equivalent to a combination of formulas $\mathsf{P}_{>q}(\vee_{i\in I}\wedge_{j\in J_i}\phi_j)$ where I and J_i is finite and $\phi_{i,j} \in \mathcal{K}$.

(V) Lastly, we prove that ψ can be of the form $\wedge_{j\in J}\phi_j$ with J finite and $\phi_j \in \mathcal{K}$.
This can be done by simply applying step (II) again. So $\underset{\sim}{\phi}$ is equivalent to a combination of formulas $\mathsf{P}_{>q}(\wedge_{j\in J}\phi_j)$ where J is finite and $\phi_j \in \mathcal{K}$.

We can finally conclude that $\underset{\sim}{\phi}$ is equivalent to some series of disjunctions, conjunctions, and negations of formulas from $\mathcal{K}$ (since if all $\phi_j \in \mathcal{K}$, then $\mathsf{P}_{>q}(\wedge_{j\in J}\phi_j) \in \mathcal{K}$). Using Lemma 5.1.1, we can find an equivalence $\underset{\sim}{\phi} \equiv \bigvee_{i\in I}\bigwedge_{j\in J_i}\underset{\sim}{\psi}'_{i,j}$ where $\underset{\sim}{\psi}'_{i,j} \in \mathcal{K} \cup \neg\mathcal{K}$, and we are done with this case.

This completes the induction, so we now know that each $\underset{\sim}{\phi} \in \mathcal{L}$ is equivalent to $\bigvee_{i\in I}\bigwedge_{j\in J_i}\underset{\sim}{\phi}_{i,j}$ where $\underset{\sim}{\phi}_{i,j} \in \mathcal{K} \cup \neg\mathcal{K}$. Since also $(\mathcal{O}_{\mathrm{pr}}, \bot, \wedge, +) \subseteq (\mathcal{O}_{\mathrm{pr}}, \bigvee, \bigwedge, \neg)$ we can conclude that $(\equiv_{(\mathcal{O}_{\mathrm{pr}},\bigvee,\bigwedge,\neg)}) = (\equiv_{(\mathcal{O}_{\mathrm{pr}},\bot,\wedge,+)})$. □

In particular, this means:

Corollary 5.2.14. *For the effect of probability,* $(\sqsubseteq^+) \cap (\sqsubseteq^+)^{\mathsf{op}} = (\equiv)$. *In particular, this means that mutual applicative $\mathcal{O}$-similarity is equal to applicative $\mathcal{O}$-bisimilarity.*

Similar results as the ones given above are featured in [12], where it is shown that a simple *testing logic* is sufficient for characterising applicative bisimilarity for a probabilistic lambda calculus. This testing logic is very similar to the logic $(\mathcal{O}_{\mathrm{pr}}, \top, \wedge, +)$.

5.2.4 Contextual preorder

The contextual preorder is an important relation, and is used as the default notion of program equivalence in most places in the literature. In this section we look at the circumstances under which the positive behavioural preorder coincides with the contextual preorder. In such cases, the general behavioural preorder, which is equal to the general behavioural equivalence, coincides with contextual equivalence. These coincidences can be proven using the reduced logical characterisations of the positive behavioural preorder established in this section. In particular, in the case that connectives can be removed completely from the logic, the coincidence can be shown as long as modalities can be suitably formulated in terms of contexts.

Definition 5.2.15. The contextual preorder $\Subset$ is the largest compatible relation satisfying basic observations on ground type, in particular:

$$\forall \underline{M}, \underline{N} \in \mathit{Terms}(\mathbf{F}\,\mathbf{1}).\ \ \underline{M} \Subset \underline{N} \iff |\underline{M}| \preccurlyeq |\underline{N}| \ .$$

Note the use of the preorder $\preccurlyeq$ on $T\mathbf{1}$ from Definition 3.3.13, which can be alternatively characterised by $t \preccurlyeq r \iff (\forall o \in \mathcal{O}, t \in [\![o]\!] \Rightarrow r \in [\![o]\!])$ from (3.2) for all of our examples. Importantly, if $\sqsubseteq^+$ is compatible and $\underline{\underline{P}} \sqsubseteq^+ \underline{\underline{R}}$ holds, then $\underline{\underline{P}} \Subset \underline{\underline{R}}$ since $\Subset$ is the largest compatible relation.

In the next result we use the notion of *context*, which is a program $C[-]$ of some type $\underline{\underline{\mathbf{E}}}$ with a hole of some type $\underline{\underline{\mathbf{F}}}$ such that for any $\underline{\underline{P}} \in \mathit{Terms}(\underline{\underline{\mathbf{F}}})$, $C[\underline{\underline{P}}]$ is the program of type $\underline{\underline{\mathbf{E}}}$ resulting from plugging $\underline{\underline{P}}$ into each hole $(-)$ of C. We use the fact resulting from compatibility of $\Subset$, that for any type $\underline{\underline{\mathbf{E}}}$, and any context $C[-]$ of type $\mathbf{F}\,\mathbf{1}$ with a hole of type $\underline{\underline{\mathbf{E}}}$, that $\forall \underline{\underline{P}}, \underline{\underline{R}} \in \mathit{Terms}(\underline{\underline{\mathbf{E}}}).\ \ \underline{\underline{P}} \Subset \underline{\underline{R}} \implies C[\underline{\underline{P}}] \Subset C[\underline{\underline{P}}] \implies |C[\underline{\underline{P}}]| \preccurlyeq |\underline{\underline{R}}|$.

In order to prove that the positive behavioural preorder is equal to the contextual preorder, we show that satisfaction of any formula can be represented as a set of *tests* consisting of pairs $(o, C[-])$ of modalities and contexts. However, to be able to do this, the set $\mathcal{O}$ needs to be sufficiently expressive. Though some properties of modalities which imply the coincidence between these two preorders can be identified, it is difficult to find properties general enough to work for all examples (the most problematic one being global store). Identifying the general requirement for proving the coincidence may be an interesting endeavour for the future.

Proposition 5.2.16. *For the effects of error, input/output and global store with finite store locations;* $\underline{\underline{P}} \Subset \underline{\underline{R}}$ *iff* $\underline{\underline{P}} \sqsubseteq_{(\mathcal{O},\bot,\top,+)} \underline{\underline{R}}$.

Proof. Firstly, since $\sqsubseteq_{(\mathcal{O},\bot,\top,+)}$ is compatible, $\underline{\underline{P}} \sqsubseteq_{(\mathcal{O},\bot,\top,+)} \underline{\underline{R}} \implies \underline{\underline{P}} \Subset \underline{\underline{R}}$.

The other implication is established using pairs $(o, C[-])$ consisting of a modality and a context in which to check the modality. For each formula $\underline{\underline{\phi}}$, we will define a set $G(\underline{\underline{\phi}})$ of such pairs, such that:

$$\underline{\underline{P}} \models \underline{\underline{\phi}} \iff \forall (o, C[-]) \in G(\underline{\underline{\phi}}), |C[\underline{\underline{P}}]| \in [\![o]\!].$$

Because formulas are well-founded, and every constructor of the logic $(\mathcal{O}, \bot, \top, +)$ is unary, each formula can be seen as a finite unary tree, a list. So with induction on $\underline{\underline{\phi}}$, we will construct $G(\underline{\underline{\phi}})$ such that the above statement holds. To start with, we need a base modality $\beta \in \mathcal{O}$ such that $\langle * \rangle \in [\![\beta]\!]$ and $\bot \notin [\![\beta]\!]$. This is used to check atomic formulas. In general, we can define the following:

1. $G(\{0\}) := \{(\beta, \mathsf{case}\ (-)\ \mathsf{of}\ \{\mathsf{return}(*), \mathsf{S}(x) \Rightarrow \Omega\})\}$.
2. $G(\{n+1\}) := \{(\beta, \mathsf{case}\ (-)\ \mathsf{of}\ \{\Omega, \mathsf{S}(x) \Rightarrow C[x]\}) \mid (o, C[-]) \in G(\{n\})\}$.
3. $G(\langle \underline{\underline{\phi}} \rangle) := \{(o, C[\mathsf{force}(-)]) \mid (o, C[-]) \in G(\underline{\underline{\phi}})\}$.
4. $G((i, \phi)) := \{(o, \mathsf{pm}\ (-)\ \mathsf{as}\ \{\Omega, \ldots, \Omega, (i.x).C[x], \Omega, \ldots, \Omega\}) \mid (o, C[-]) \in G(\phi)\}$.

5. $G(\pi_0(\phi)) := \{(o, \mathsf{pm}\ (-)\ \mathsf{as}\ (x,y).C[x]) \mid (o, C[-]) \in G(\phi)\}$.
6. $G(\pi_1(\phi)) := \{(o, \mathsf{pm}\ (-)\ \mathsf{as}\ (x,y).C[y]) \mid (o, C[-]) \in G(\phi)\}$.
7. $G((V \mapsto \underline{\phi})) = \{(o, C[(-)\ V]) \mid (o, C[-]) \in G(\underline{\phi})\}$.
8. $G((i \mapsto \underline{\phi})) := \{(o, C[(-)\ i]) \mid (o, C[-]) \in G(\underline{\phi})\}$.
9. $G(\bot) := \{(\beta, \Omega)\}$.
10. $G(\top) := \{(\beta, \mathsf{return}(*))\}$.

For the definition of $G(o(\phi))$, we need to look at the specific effects.

Error: Here we take $\beta := \downarrow$ as the neutral modality.

11. $G(\mathsf{E}_e(\phi)) := \{(\mathsf{E}_e, (-)\ \mathsf{to}\ x.\, \mathsf{return}(*))\}$.
12. $G(\downarrow(\phi)) := \{(\downarrow, (-)\ \mathsf{to}\ x.\, \mathsf{return}(*))\} \cup \{(o, (-)\ \mathsf{to}\ x.\, C[x]) \mid (o, C[-]) \in G(\phi)\}$.

I/O: Here we take $\beta := \langle\varepsilon\rangle\downarrow$.

11. $G(\langle v\rangle_{\dots}(\phi)) := \{(\langle v\rangle_{\dots}, (-)\ \mathsf{to}\ x.\, \mathsf{return}(*))\}$.
12. $G(\langle v\rangle\downarrow(\phi)) := \{(\langle v\rangle\downarrow, (-)\ \mathsf{to}\ x.\, \mathsf{return}(*))\}$
$\cup\ \{(\langle vw\rangle i, (-)\ \mathsf{to}\ x.\, C[x]) \mid (\langle w\rangle i, C[-]) \in G(\phi)\}$.

Global store: Here we take $\beta := (o \rightarrowtail o)$ as the neutral modality, where $o \in \mathsf{State}$ is the always zero state (we could have used any state). Assume that there are only finitely many locations for storing numbers. For any state s we define S_s for the program which updates the global store to s and returns $*$. These programs can only exist if there are finitely many locations.

$$G((s \rightarrowtail r)(\phi)) := \{((s \rightarrowtail r), (-)\ \mathsf{to}\ x.\, \mathsf{return}(*))\} \cup$$
$$\{((s \rightarrowtail b), (-)\ \mathsf{to}\ x.\, S_a; C[x]) \mid ((a \rightarrowtail b), C[-]) \in G(\phi)\}$$

□

We can conclude that for the following examples of effects, it holds that the behavioural preorder is identical to the $\mathcal{O}$-contextual preorder, where $\mathcal{O}$ is the appropriate set of modalities.

Pure: $(\Sigma_\emptyset, \mathcal{O}_\emptyset)$	Error: $(\Sigma_{\mathrm{er}}, \mathcal{O}_{\mathrm{er}})$
Input/Output: $(\Sigma_{\mathrm{io}}, \mathcal{O}_{\mathrm{io}})$	Global store: $(\Sigma_{\mathrm{gs}}, \mathcal{O}_{\mathrm{gs}})$, with finite Loc.

In the above cases of effect examples, it also holds that the behavioural equivalence coincides with the contextual equivalence. This is because the contextual equivalence is equal to the mutual contextual preorder in these cases. It should also be noted that in [12], it has been proven that for the effect of probability, applicative bisimilarity coincides with contextual equivalence, though the programming language studied there is slightly different.

5.3 Combining effects

In this section we look at examples of possible combinations of effects for which we can find a decomposable set of Scott open modalities, which adequately characterises the behavioural properties of such combinations of effects. Combining effects will however turn out to be a lot easier in the quantitative logic of Chapter 6, so we will not explore all possible combinations here.

Given some combination of effects, we need to choose a set of modalities which describe the behaviour of such effects. In most cases, we add a new effect to an already existing set of effects with modalities, and modify those existing modalities to also incorporate the behaviour of the newly added effect. Such modifications do not always result in modalities with the right properties, and can only be done in specific instances. As such, we do not propose a uniform method for combining effects. In fact, as seen in Subsection 3.5.4, some combinations are not possible in the Boolean logic.

For several combinations of effects, we will define a set of modalities $\mathcal{O}$, and we will prove that $\mathcal{O}$ has the right properties: Is it a decomposable set of Scott open modalities? The Scott open property for modalities is usually easily established. It is establishing the property of decomposability which creates the most problems, and cannot always be done. To this end, we first establish some general properties sufficient to prove that certain combinations of effects have decomposable sets of modalities.

Definition 5.3.1. $\mathcal{O}$ is *unidecomposable* if for any $t \in T(T(\mathbf{1}))$ and $o \in \mathcal{O}$ with $\mu t \in [\![o]\!]$, there are $o', o'' \in \mathcal{O}$ such that $t \in o'(o''(\{*\}))$ and for all $r \in T(T(\mathbf{1}))$, $r \in o'(o''(\{*\})) \implies \mu r \in [\![o]\!]$.

This is a stronger property than strong decomposability (Definition 3.3.25). For example, the set of modalities $\mathcal{O}_{\mathrm{pr}}$ for the effect of probability is strongly decomposable, but not unidecomposable. Error, input/output, global store, angelic/demonic/neutral nondeterminism, and timer all have unidecomposable sets of modalities, which is directly established by the proofs of Subsection 3.3.3.

Definition 5.3.2. A *branch-tree* is a tree $t \in T(X)$ such that for any pair of smaller trees $t_0 \leq t$ and $t_1 \leq t$, either $t_0 \leq t_1$ or $t_1 \leq t_0$.

This means the tree never has more than one continuation not labelled $\bot$, it only has one unique branch when excluding $\bot$ leaves. In other words, each node has at most one non-$\bot$ child. Moreover, if t is a branch-tree and $r \leq t$, then r is a branch-tree. See Figure **??** for an example of a branch tree.

Definition 5.3.3. A modality o is *single-branched* if for any $t \in [\![o]\!]$ there is a branch-tree $t' \leq t$ below t such that $t' \in [\![o]\!]$.

Note that if o is also Scott open, the branch-tree t' can be chosen to be finite. Examples of effects where all modalities are single-branched (considering the standard choice of modalities) include: Error, Angelic nondeterminism, Global Store and Input/Output. The modalities for probability and demonic nondeterminism are not single-branched.

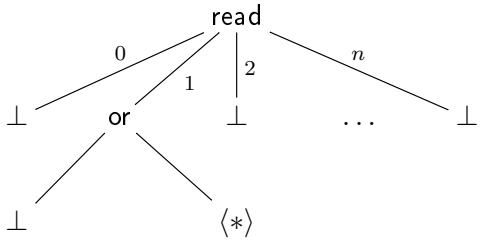


Figure 5.1: Example of a branch-tree.

We first look at combining sets of modalities over the same set of effect operators. Recall Corollary 3.3.24, which shows that decomposability of sets of modalities is preserved by union. This has some interesting applications. For example, if we can consider the $\Box$ modality of must termination for the probabilistic binary choice operator (which is not expressible in terms of the probabilistic modalities $\mathcal{O}_{\mathrm{pr}}$[1]). The extended set $\mathcal{O}_{\mathrm{pr}} \cup \{\Box\}$ is also a decomposable set of Scott open modalities, which gives a different notion of behavioural equivalence for probabilistic languages then $\mathcal{O}_{\mathrm{pr}}$.

Another application of this proposition is that to any decomposable set of Scott open modalities, we can add the termination modality $\downarrow$, with denotation $[\![\downarrow]\!] := \{\langle * \rangle\}$, which observes that a computation has terminated without encountering any effectful behaviour. The resulting set of modalities is still decomposable. We may similarly add a modality $\pm$, with denotation $[\![\pm]\!] := T(\mathbf{1}) - \{\bot\}$, which checks that if a computation diverges, it has at least encountered some effectful behaviour. The resulting set of modalities is still decomposable since: $\forall t \in T(T(\mathbf{1}))$, $\mu t \in \pm(\{*\}) \iff t \in \pm(\pm\{*\})$.

5.3.1 Combinations with error messages

Error messages can be added to any effects. Let $(\Sigma, \mathcal{O})$ be the signature and modalities for one or more effects. Take Err to be a set of error messages we want to add, and let $\Sigma^{\mathsf{Err}} := \Sigma \cup \{\mathsf{raise}_e() \mid e \in \mathsf{Err}\}$ (disjoint union). A modality on Σ^{Err} is specified by a subset $G \subseteq \mathsf{Err}$ of observed error messages. Such a subset gives us a transformation $\overline{G} : T_{\Sigma^{\mathsf{Err}}}(\mathbf{1}) \to T_\Sigma(\mathbf{1})$ defined recursively as follows:

$\overline{G}(\bot) := \bot$
$\overline{G}(\langle * \rangle) := \langle * \rangle$.
$\overline{G}(\mathsf{raise}_e()) := \langle * \rangle$, if $e \in G$, else $\bot$.
$\overline{G}(\mathsf{op}_{l_1,\ldots,l_n}\langle t_0, t_1, \ldots \rangle) := \mathsf{op}_{l_1,\ldots,l_n}\langle \overline{G}(t_0), \overline{G}(t_1), \ldots \rangle$, for $\mathsf{op} \in \Sigma$.

We define a new set of modalities by:

$$\mathcal{O}^{\mathsf{Err}} := \{(o)_{\in G} \mid G \subseteq \mathsf{Err}, o \in \mathcal{O}\}.$$

where $\underline{M} \models (o)_{\in G}(\phi) \iff \overline{G}(|\underline{M}|[\models \phi]) \in [\![o]\!]$. The denotation is given by $[\![(o)_{\in G}]\!] = \overline{G}^{-1}([\![o]\!])$. Note that $t \in (o)_{\in G}(\emptyset) \iff t \in (o)_{\in \emptyset}(\emptyset)$.

[1] $[\![\Box]\!]$ is not equal to $\bigcap_{q \in \mathbb{Q}, q<1} [\![\mathsf{P}_{>q}]\!]$, as the former tests *sure* termination while the latter tests *almost-sure* termination.

For each of the modalities o, $(o)_{\in G}$ treats error messages of G as if they are terminating computations. E.g. $\underline{M} \models (\Box)_{\in G}(\phi)$ holds if the computation $\underline{M}$ must either return a value satisfying ϕ or raise an error $e \in G$. In particular, $\underline{M} \models (\Box)_{\in G}(\bot)$ if it must raise an error from G.

Lemma 5.3.4. *If o is Scott open, then $(o)_{\in G}$ is Scott open.*

Proof. This is a consequence of the continuity of the function $\overline{G}$. □

Lemma 5.3.5. *If $\mathcal{O}$ is a decomposable set of upwards closed modalities, then $\mathcal{O}^{\mathsf{Err}}$ is decomposable.*

Proof. Since we need to deal with double trees, we define a variation of $\overline{G}$, a function $\widehat{G} : T_{\Sigma^{\mathsf{Err}}}(T_{\Sigma^{\mathsf{Err}}}(\mathbf{1})) \to T_{\Sigma}(T_{\Sigma}(\mathbf{1}))$ defined by:

$\widehat{G}(\bot) := \bot$,
$\widehat{G}(\langle t \rangle) := \langle \overline{G}(t) \rangle$,
$\widehat{G}(\mathsf{raise}_e()) := \langle\langle * \rangle\rangle = \eta(\eta(*))$ if $e \in G$, otherwise $\bot$,
$\widehat{G}(\mathsf{op}(t_0, t_1, \dots)) := \mathsf{op}(\widehat{G}(t_0), \widehat{G}(t_1), \dots)$ for $\mathsf{op} \in \Sigma$.

The function is chosen s.t. for $t \in T_{\Sigma^{\mathsf{Err}}}(T_{\Sigma^{\mathsf{Err}}}(\mathbf{1}))$ we have $\overline{G}(\mu t) = \mu(\widehat{G}(t))$. We distinguish between $\preccurlyeq$ and $\preccurlyeq^{\mathsf{Err}}$, the basic preorders for $(\Sigma, \mathcal{O})$ and $(\Sigma^{\mathsf{Err}}, \mathcal{O}^{\mathsf{Err}})$ respectively. Note that $a \preccurlyeq^{\mathsf{Err}} b \iff \forall G \subset \mathsf{Err}, \overline{G}(a) \preccurlyeq \overline{G}(b)$. As a consequence, we have $(\preccurlyeq^{\mathsf{Err}\uparrow}[(\overline{G}^{-1}(A))]) \subseteq \overline{G}^{-1}(\preccurlyeq^{\uparrow}[A])$. We prove the statement of decomposability given by Lemma 3.3.19.

Assume the following, for any $t, r \in T_{\Sigma^{\mathsf{Err}}}(T_{\Sigma^{\mathsf{Err}}}(\mathbf{1}))$, $\forall o' \in \mathcal{O}^{\mathsf{Err}}$ and $\forall A \subseteq T_{\Sigma^{\mathsf{Err}}}(\mathbf{1})$:

(I) If $t \in o'(A)$ then $r \in o'(\preccurlyeq^{\mathsf{Err}\uparrow}[A])$.

For $(o)_{\in G} \in \mathcal{O}^{\mathsf{Err}}$, assume $\mu t \in [\![(o)_{\in G}]\!]$, hence $\mu(\widehat{G}(t)) \in [\![o]\!]$. We want to prove that $\mu r \in [\![(o)_{\in G}]\!]$ by using the decomposability of $\mathcal{O}$ for the trees $\widehat{G}(t)$ and $\widehat{G}(r)$. We prove that the decomposability precondition $\widehat{G}(t) \preccurlyeq \widehat{G}(r)$ holds:

Let $\delta \in \mathcal{O}$ and $A \subseteq T_{\Sigma}(\mathbf{1})$ such that $\widehat{G}(t) \in \delta(A)$. There are two cases for A, depending on whether $\langle * \rangle$ is in A.

1. If $\langle * \rangle \in A$, then errors from G raised in the 'first' part of t (before returning a tree) are acceptable, so $t \in (\delta)_{\in G}(\overline{G}^{-1}(A))$. Using assumption (I) we get $r \in (\delta)_{\in G}(\preccurlyeq^{\mathsf{Err}\uparrow}[\overline{G}^{-1}(A)])$. Using upwards closure of δ, we establish that $r \in (\delta)_{\in G}(\overline{G}^{-1}(\preccurlyeq^{\uparrow}[A]))$, hence $\overline{G}(r[\in \overline{G}^{-1}(\preccurlyeq^{\uparrow}[A])]) \in [\![\delta]\!]$. We can conclude that $\widehat{G}(r) \in \delta(\preccurlyeq^{\uparrow}[A])$ (since $\langle * \rangle \in (\preccurlyeq^{\uparrow}[A])$).

2. If $\langle * \rangle \notin A$, then errors from G raised in the 'first' part of t (before returning a tree) are unacceptable, so $t \in (\delta)_{\in \emptyset}(\overline{G}^{-1}(A))$. Using assumption (I) we get $r \in (\delta)_{\in \emptyset}(\preccurlyeq^{\mathsf{Err}\uparrow}[\overline{G}^{-1}(A)])$. Using upwards closure of δ, we establish that $r \in (\delta)_{\in \emptyset}(\overline{G}^{-1}(\preccurlyeq^{\uparrow}[A]))$. Regardless of whether $\langle * \rangle \in (\preccurlyeq^{\uparrow}[A])$, we get from upwards closure of δ that $\widehat{G}(r) \in \delta(\preccurlyeq^{\uparrow}[A])$.

Having proven that $\widehat{G}(t) \preccurlyeq \widehat{G}(r)$, so by decomposability we know that $\mu(\widehat{G}(t)) \preccurlyeq \mu(\widehat{G}(r))$. Hence $\mu(\widehat{G}(t)) \in [\![o]\!]$ implies $\mu(\widehat{G}(r)) \in [\![o]\!]$, so $\overline{G}(\mu r) \in [\![o]\!]$ and we conclude that $\mu r \in [\![(o)_{\in G}]\!]$. So we know that $\mathcal{O}^{\mathsf{Err}}$ is decomposable. □

When combining error and global store, the final state when an error is raised remains observable by a behavioural property. For example, the computations $\mathsf{update}_l(\mathsf{return}(\overline{0}); \mathsf{raise}_e())$ of type $\mathbf{FN}$ is considered different from $\mathsf{update}_l(\mathsf{return}(\overline{1}); \mathsf{raise}_e())$. The definition of error + global store could however be manually altered to obscure this information, and still create a decomposable set of modalities. In the quantitative logic of Chapter 6, such combinations are more easily tweaked. See for example the discussion in Subsection 6.2.7.

5.3.2 Combinations with angelic nondeterminism

It is possible to combine angelic nondeterminism with any effect which has a unidecomposable set of Scott open modalities. Let $(\Sigma, \mathcal{O})$ be the signature and an accompanying set of modalities. Take $\Sigma^{\mathsf{Aor}} := \Sigma \cup \{\mathsf{or}\}$, with a minor abuse of notation as we should formally take the disjoint union.

A *strategy* is a function which chooses for each or node in a tree $T_{\Sigma^{\mathsf{Aor}}}(X)$ a resolution: do we go left or right? We formalise this by saying a strategy is a function $\rho : \mathbb{N}^* \to \{l, r\}$ from lists of natural numbers to a choice between left l and right r. With the set of strategies $\mathsf{Str} := (\mathbb{N}^* \to \{l, r\})$, we define a function $\widehat{(-)} : \mathsf{Str} \to (T_{\Sigma^{\mathsf{Aor}}}(X) \to T_\Sigma(X))$, which sends ρ to a function $\widehat{\rho} : T_{\Sigma^{\mathsf{Aor}}}(X) \to T_\Sigma(X)$ resolving in a tree $t \in T_{\Sigma^{\mathsf{Aor}}}(X)$ all nondeterministic choices according to the strategy, resulting in a tree $\widehat{\rho}(t) \in T_\Sigma(X)$. This is done according to the following rules (where $\varepsilon \in \mathbb{N}^*$ is the empty list, and $(m)s$ is adding m to the front of list s):

$$\widehat{\rho}(\langle x \rangle) = \langle x \rangle$$

$$\widehat{\rho}(\bot) = \bot$$

$$\widehat{\rho}(\mathsf{or}(t_0, t_1)) = (\lambda v.\widehat{\rho((0)v)})(t_0) \text{ if } \rho(\varepsilon) = l.$$

$$\widehat{\rho}(\mathsf{or}(t_0, t_1)) = (\lambda v.\widehat{\rho((1)v)})(t_1) \text{ if } \rho(\varepsilon) = r.$$

$$\widehat{\rho}(\mathsf{op}(l_1, \ldots, l_n, m \mapsto t_m)) = \mathsf{op}(l_1, \ldots, l_n, m \mapsto (\lambda v.\widehat{\rho((m)v)})(t_m)), \text{ for } \mathsf{op} \in \Sigma.$$

Note that the arity of the operation op above may be finite.

By continuity we have $\widehat{\rho}(\sqcup_i t_i) = \sqcup_i \widehat{\rho}(t_i)$. We define the set of modalities:

$$\mathcal{O}^{\mathsf{Aor}} := \{(o)_\Diamond \mid o \in \mathcal{O}\}, \qquad \text{where } [\![(o)_\Diamond]\!] := \bigcup_{\rho \in \mathsf{Str}} \widehat{\rho}^{-1}([\![o]\!])$$

So $\underline{M} \models ((o)_\Diamond)[\phi] :\iff \exists \rho \in \mathsf{Str}.\widehat{\rho}(|\underline{M}|[\phi]) \in [\![o]\!]$.[2] For the rest of this subsection, we will simply write ρ to mean the function $\widehat{\rho}$.

[2] It holds that $(\Diamond)_{\in\{e\}}$ is isomorphic to $(\mathsf{E}_e)_\Diamond$.

Lemma 5.3.6. *If o is Scott open, then $(o)_\Diamond$ is Scott open.*

Proof. $\sqcup_i t_i \in [\![(o)_\Diamond]\!] \Rightarrow \sqcup_i t_i \in \bigcup_\rho \rho^{-1}([\![o]\!]) \Rightarrow \exists \rho, \rho(\sqcup_i t_i) \in [\![o]\!] \Rightarrow \exists \rho, \sqcup_i \rho(t_i) \in [\![o]\!] \Rightarrow \exists \rho, \exists i, \rho(t_i) \in [\![o]\!] \Rightarrow \exists i, t_i \in [\![(o)_\Diamond]\!]$. □

Lemma 5.3.7. *If $\mathcal{O}$ is a unidecomposable set of upwards closed modalities, then $\mathcal{O}^{\mathsf{Aor}}$ is unidecomposable.*

Proof. Let $t \in T_{\Sigma^{\mathsf{Aor}}}(T_{\Sigma^{\mathsf{Aor}}}(\mathbf{1}))$ such that $\mu t \in [\![(o)_\Diamond]\!]$. Hence there is a ρ such that $\rho(\mu t) \in [\![o]\!]$. There is a ρ' for t (simply given by ρ) and for each leaf x a ρ_x (dependent on the location of x in t) such that $\mu(\rho'(t[x \mapsto \rho_x(x)])) = \rho(\mu t)$. The replacement of x in $t[x \mapsto \rho_x(x)]$ is also dependent on the location of x in t. We use that $\mathcal{O}$ is unidecomposable to get $o', o'' \in \mathcal{O}$ such that:

$$\rho'(t[x \mapsto \rho_x(x)]) \in o'(o''(\{*\})) \text{ and } \forall k \in T_\Sigma(T_\Sigma(\mathbf{1})), k \in o'(o''(\{*\})) \implies \mu k \in [\![o]\!].$$

From $\rho'(t[x \mapsto \rho_x(x)]) \in o'(o''(\{*\}))$ and upwards closure of o' we derive $t \in (o')_\Diamond((o'')_\Diamond(\{*\}))$. If $r \in (o')_\Diamond((o'')_\Diamond(\{*\}))$ then there is a strategy ρ'_r for r and for each leaf y of r a strategy ρ'_y (dependent on the location of y in r) such that $\rho'_r(r[y \mapsto \rho'_y(y)]) \in o'(o''(\{*\}))$. By choice of o' and o'' it holds that $\mu(\rho'_r(r[y \mapsto \rho'_y(y)])) \in [\![o]\!]$, moreover there is a strategy ρ_r such that $\rho_r(\mu r) = \mu(\rho'_r(r[y \mapsto \rho'_y(y)]))$. We can conclude that $\rho_r(\mu r) \in [\![o]\!]$ and hence $\mu r \in [\![(o)_\Diamond]\!]$. □

One can combine angelic nondeterminism with probability in a similar manner, and get behaviourally meaningful modalities. Though as seen in Subsection 3.5.4, this combination of effects is problematic. So the resulting set of modalities is not suitable for describing a behavioural equivalence for this combination of effects.

5.3.3 Combinations with timer effect

We can also add the timer effect to some of the other effects. We will focus on the down-interpretation of timer effects, with $\mathcal{O}^{\downarrow}_{\mathrm{ti}}$, where the modalities $\mathsf{C}_{\leq q}$ are testing whether a computation has terminated in a set amount of time q. The combination with timer works most easily when the other effects have single-branched modalities (Definition 5.3.3), which include error, global store, input/output and angelic nondeterminism.

Let $(\Sigma, \mathcal{O})$ be some other signature with modalities. For the timer effect, we choose a countable set Inc of *rational* delays, constructing a signature (formally disjoint union):

$$\Sigma^{\mathsf{Tim}} := \Sigma \cup \{\mathsf{tick}_c(-) : \alpha \to \alpha \mid c \in \mathsf{Inc}\}.$$

We develop a *time-out pruner* $\nu : T_{\Sigma^{\mathsf{Tim}}}(X) \times \mathbb{Q} \to T_\Sigma(X)$ which given (t, q), goes through the evaluation of the tree t removing tick nodes and pruning subtrees (replacing it with $\bot$) when the sum of time delays given by the removed tick nodes along some evaluation branch exceeds q. We give a formal definition:

$\nu(\bot) := \bot$.

$$\nu(\langle x\rangle, q) := \langle x\rangle.$$
$$\nu(\mathsf{tick}_c(t), q) := \begin{cases} \nu(t,\ q-c) & \text{if } q \geq c \\ \bot & \text{otherwise} \end{cases}.$$
$$\nu(\mathsf{op}_{l_1,\ldots,l_n}\langle t_1, t_2, \ldots\rangle, q) := \mathsf{op}_{l_1,\ldots,l_n}\langle \nu(t_1,q), \nu(t_2,q), \ldots\rangle.$$

Given this map we define for each modality $o \in \mathcal{O}$ and rational time duration $q \in \mathbb{Q}$ a new modality $(o)_{\leq q}$ such that:

$$\underline{M} \models (o)_{\leq q}(\phi) \quad :\Longleftrightarrow \quad \nu(|\underline{M}|, q) \in [\![o]\!]$$

with the denotation $[\![(o)_{\leq q}]\!] := \nu^{-1}([\![o]\!] \times \{q\})$. Note that if $q \leq q'$ holds, then $[\![(o)_{\leq q}]\!] \subseteq [\![(o)_{\leq q'}]\!]$. We define the new set of modalities as $\mathcal{O}^{\mathsf{Tim}} := \{(o)_{\leq q} \mid o \in \mathcal{O}, q \in \mathbb{Q}\}$.

If for example we combine the effect-free language with the timer effect, we get that the resulting set of modalities $\mathcal{O}_\emptyset^{\mathsf{Tim}}$ is isomorphic to the set of modalities $\mathcal{O}_{\mathrm{ti}}^{\downarrow}$ from the down-interpretation of timer.

These new modalities are obviously Scott open if the original modalities were Scott open, because of the continuity of the time-out pruner function. We now look at the proof of decomposability for this set of modalities.

For a double tree $t \in T(T(X))$ we can define a *branch-tree substructure* as a branch-tree $t' \in T(T(X))$, which has at most one non-trivial leaf a given by a branch-tree, such that for some $b \in T(X)$: $t'[a \mapsto b] \leq t$ and $a \leq b$. Equivalently, a branch-tree substructure of t is a double tree $t' \in T(T(X))$ such that $\mu t'$ is a branch tree below μt, and $t'[\in T(X)] \leq t[\in T(X)]$.

Proposition 5.3.8. *If $\mathcal{O}$ is a unidecomposable set of single-branched upwards closed modalities, then $\mathcal{O}^{\mathsf{Tim}}$ is unidecomposable.*

Proof. Let $t \in T_{\Sigma^{\mathsf{Tim}}}(T_{\Sigma^{\mathsf{Tim}}}(X))$ such that $\mu t \in [\![(o)_{\leq q}]\!]$, then $\nu(\mu t, q) \in [\![o]\!]$. Since o is single-branched, there is a branch-tree $b \leq \nu(\mu t, q)$ such that $b \in [\![o]\!]$. Let $r \leq \mu t$ be the branch-tree such that $b = \nu(r, q)$. Let t' be the branch-tree substructure of t such that $\mu t' = r$, hence $\nu(\mu t', q) = b$. Since $\mu t'$ is a branch-tree, its execution follows a single sequence of tick operators. So there must be q' and $q'' \in \mathbb{Q}$, upper bounds to the sum of the ticks before termination and after termination respectively, such that $q' + q'' = q$ and $\mu(\nu(t'[a \mapsto \nu(a, q'')], q')) = b \in [\![o]\!]$.

Since $\mathcal{O}$ is unidecomposable, we can find appropriate o' and $o'' \in \mathcal{O}$ such that $\nu(t'[a \mapsto \nu(a,q'')], q') \in o'(o''(\{*\}))$, and hence $t' \in o'_{\leq q'}(o''_{\leq q''}(\{*\}))$. Now, if it holds that $r \in o'_{\leq q'}(o''_{\leq q''}(\{*\}))$, then $\nu(r[a \mapsto \nu(a,q'')], q') \in o'(o''(\{*\}))$ so as a consequence of uni-strong decomposability, $\mu(\nu(r[a \mapsto \nu(a,q'')], q')) \in [\![o]\!]$. Hence with $\nu(\mu r, q' + q'') \geq \mu(\nu(r[a \mapsto \nu(a,q'')], q'))$ and by upwards closure of o, we get $\nu(\mu r, q) \in [\![o]\!]$, so $\mu r \in [\![(o)_{\leq q}]\!]$. □

This approach to combining effects with timer effects can also be used to add time delays to already existing effect operators. For instance, in combination with angelic nondeterminism, we can let the binary or operator take a set amount of time. One can then adapt the time-out pruner function to modify the set of modalities similar to what is done above.

5.3.4 Combinations with probability

We tie up some loose ends by looking at some combinations with probability. In particular, combinations of probability with global store and input/output are possible, because when testing the satisfaction of a modality for these effects, only one specific branch needs to be checked. We will call such modalities *branch-focussed*, a concept that is formulated in Definition 5.3.9.

Given a signature Σ and a set of modalities $\mathcal{O}$, we define a new set of modalities $\mathcal{O}'$ on $\Sigma' = \Sigma \cup \{\mathsf{pr}\}$ (formally disjoint union). Consider $\{l, r\}^*$, the set of lists of choices between left and right. There is a function $K : \{l, r\}^* \to (T_{\Sigma'}(X) \to T(X))$ resolving the probabilistic choice of a tree $t \in T_{\Sigma'}(X)$ by a series of choices $v \in \{l, r\}^*$ in the following way:

$$
\begin{aligned}
K(v)(\bot) &:= \bot.\\
K(v)(\langle a\rangle) &:= \langle a\rangle.\\
K(\varepsilon)(\mathsf{pr}(t, r)) &:= \bot.\\
K((l)v)(\mathsf{pr}(t, t')) &:= K(v)(t).\\
K((r)v)(\mathsf{pr}(t, t')) &:= K(v)(t').\\
K(v)(\mathsf{op}(l_1, \ldots, l_n, m \mapsto t_m)) &:= \mathsf{op}(l_1, \ldots, l_n, m \mapsto K(v)(t_m)) \text{ for } \mathsf{op} \in \Sigma.
\end{aligned}
$$

Given a Scott open modality $o \in \mathcal{O}$, we can now define the associated probability function $\mathsf{P}^o : T_{\Sigma'}(\{*\}) \to [0, 1]$ by calculating for $t \in T_{\Sigma'}(\{*\})$ the 'probability' of $\{v \in \{l, r\}^* \mid K(v)(t) \in o(\{*\})\}$. Formally, we define for each $n \in \mathbb{N}$ a function $\hat{n} : \{l, r\}^n \to \{l, r\}^*$ sending n-tuples to their respective lists of length n. Then we define $\mathsf{P}^o_n : T_{\Sigma'}(\{*\}) \to [0, 1]$ as $\mathsf{P}^o_n(t) = \#\{s \in \{l, r\}^n \mid K(\hat{n}(s))(t) \in o(\{*\})\}/2^n$. Since for $v \in \{l, r\}^*$ an initial segment of $v' \in \{l, r\}^*$, $K(v)(t) \leq K(v')(t)$, and o is upwards closed, we know that the sequence $\{\mathsf{P}^o_n(t)\}_{n\in\mathbb{N}}$ is monotone increasing. So we can define $\mathsf{P}^o(t)$ as $\lim_{n\in\mathbb{N}} \mathsf{P}^o_n(t)$.

Given a modality $o \in \mathcal{O}$ and a rational $q \in [0, 1]$, we define a modality $o_{>q}$ on Σ' denoted by $[\![o_{>q}]\!] := \{t \in T_{\Sigma'}(\{*\}) \mid \mathsf{P}^o(t) > q\}$. For example, in the case of the effect-free language with $\mathcal{O} = \mathcal{O}_\emptyset = \{\downarrow\}$, the modality $(\downarrow)_{>q}$ is isomorphic to $\mathsf{P}_{>q}$. We now define our new set of modalities on Σ' as $\mathcal{O}' := \{o_{>q} \mid o \in \mathcal{O}, q \in [0, 1] \cap \mathbb{Q}\}$. This set of modalities is well-defined for any set of upwards closed modalities $\mathcal{O}$. However, it does not generally hold that for any decomposable set of Scott open modalities $\mathcal{O}$, the new set of modalities $\mathcal{O}'$ is decomposable.

We identify two properties on the set of modalities $\mathcal{O}$ sufficient for proving that the new set of modalities $\mathcal{O}'$ for the combination of effects with probability is decomposable. These properties may not be exhaustive, though they do enable us to prove that the global store and input/output effects can be combined with probability.

Definition 5.3.9. A modality $o \in \mathcal{O}$ is *branch-focussed* if for any $t \in [\![o]\!]$, there is a *unique* branch-tree $b \leq t$ such that $b \in [\![o]\!]$.

The main function of this property is to establish that satisfaction of a modality depends on one branch only. Examples of sets of branch-focussed modalities are the modalities for global store, input/output and timer. An example of a modality which is

not branch-focussed is the may modality $\Diamond$ for nondeterminism, even though it is single branched.

Definition 5.3.10. $\mathcal{O}$ is *disjointly unidecomposable* if for any $o \in \mathcal{O}$, there is a family of pairs of modalities $\{(o_i, o_i')\}_{i \in I}$ such that:

1. $\forall i \in I, t \in TT\mathbf{1}, \;\; t \in o_i(o_i'(\{*\})) \implies \mu t \in o(\{*\})$.
2. $\forall t \in TT\mathbf{1}, \;\; \mu t \in o(\{*\}) \implies \exists i \in I, t \in o_i(o_i'(\{*\}))$.
3. $\forall i, j \in I, \;\; i \neq j \implies o_i(o_i'(\{*\})) \cap o_j(o_j'(\{*\})) = \emptyset$.

We show that the modalities of two examples of effects satisfy this property.

Global store: For any modality $(s \rightarrowtail s') \in \mathcal{O}_{\text{gs}}$, the family of pairs of modalities given by the set $\{((s \rightarrowtail s''), (s'' \rightarrowtail s')) \mid s'' \in \mathsf{State}\}$ has the desired properties.

Input/output: For the modality $\langle v \rangle\!\downarrow \in \mathcal{O}_{\text{io}}$, the family of pairs of modalities given by the set $\{(\langle w \rangle\!\downarrow, \langle w' \rangle\!\downarrow) \mid w, w' \;\text{ i/o-traces}, ww' = v\}$ has the desired properties, and for the modality $\langle v \rangle_{\dots} \in \mathcal{O}_{\text{io}}$, the set $\{(\langle w \rangle\!\downarrow, \langle w' \rangle_{\dots}) \mid w, w' \text{ i/o-traces}, w' \neq \varepsilon \wedge ww' = v\} \cup \{(\langle v \rangle_{\dots}, \langle \varepsilon \rangle_{\dots})\}$ has the desired properties.

Note that the error effect also satisfies this property, but we do not include it here since we have already shown that we can do combinations with the error effect. It is however interesting to note that $(\mathsf{E}_e)_{>q}$ is isomorphic to $(\mathsf{P}_{>q})_{\in\{e\}}$.

We can now give the main result of this subsection.

Proposition 5.3.11. *If $\mathcal{O}$ is a disjointly unidecomposable set of Scott open and branch-focussed modalities, then $\mathcal{O}'$ is a strongly decomposable set of Scott open modalities.*

Scott openness is relatively straightforward to verify, so we will not go into the proof of that property. We will focus on sketching the proof of the fact that $\mathcal{O}'$ is strongly decomposable given the conditions laid out in the proposition.

Proof sketch. Suppose for $t \in T_{\Sigma'}(T_{\Sigma'}(\{*\}))$ and $o_{>q} \in \mathcal{O}'$, it holds that $t \in [\![o_{>q}]\!]$. Take $\{(o_i, o_i')\}_{i \in I}$ to be the set given to us by disjoint uni-decomposability on $o \in \mathcal{O}$. We prove that the set $\{((o_i)_{>p}, (o_i')_{>p'}) \mid i \in I, p, p' \in [0,1] \cap \mathbb{Q}, t \in (o_i)_{>p}((o_i')_{>p'}(\{*\}))\}$ has the properties required by strong decomposability in Definition 3.3.25 (note that property 1 holds trivially). This proof is similar to the proof that the set of modalities for probability is strongly decomposable.

If $t \in [\![o_{>q}]\!]$, then there is an $n \in \mathbb{N}$ such that $\mathsf{P}_n^o(\mu t) > q$, hence $\#\{s \in \{l,r\}^n \mid K(\hat{n}(s))(\mu t) \in o(\{*\})\} > q \cdot 2^n$. Take one such $s \in \{l,r\}^n$ such that $K(\hat{n}(s))(t) \in o(\{*\})$. Since o is branch focussed, there is a unique branch-tree $r_s \leq K(\hat{n}(s))(\mu t)$ such that $r_s \in o(\{*\})$. We can find a branch-tree substructure t_s of t such that $K(\hat{n}(s))(\mu t_s) = r_s$. Moreover, we can split up $\hat{n}(s)$ into two lists v and w, such that $vw = \hat{n}(s)$ and $\mu(K(v)(K(w)^*(t_s))) = r_s$. Considering

$\mu(K(v)(K(w)^*(t_s))) \in [\![o]\!]$, we can use disjoint uni-decomposability to find a unique $i(s) \in I$ such that $K(v)(K(w)^*(t_s)) \in o_{i(s)}(o'_{i(s)}(\{*\}))$.

In short, for each $s \in \{l,r\}^n$ such that $K(\hat{n}(s))(\mu t) \in o(\{*\})$, we can find an associated unique $i(s) \in I$. So branch-tree substructures of t showing that $\mathsf{P}^o_n(\mu t) > q$ can be partitioned into sets $o_i(o'_i(\{*\}))$. The key is that each $i \in I$ can contribute something to the total probability $\mathsf{P}^o(\mu t)$, and since the sets $o_i(o'_i(\{*\}))$ are disjoint, there is no fear in counting some of the contributions multiples times. For each $i \in I$, we define a real-valued function $f^i_t : [0,1] \to [0,1]$ where for each rational number $q \in [0,1]$ we define $f^i_t(p) := \sup\{p' \in [0,1] \cap \mathbb{Q} \mid t \in (o_i)_{>p}((o'_i)_{>p'}(\{*\}))\}$. Then $\int_0^1 f^i_t(p)dp$ is the contribution of $i \in I$ to $\mathsf{P}^o(\mu t)$. More precisely, $\mathsf{P}^o(\mu t) = \Sigma_{i\in I}(\int_0^1 f^i_t(p)dp)$.

The argument sketched above motivates the equality $\mathsf{P}^o(\mu t) = \Sigma_{i\in I}(\int_0^1 f^i_t(p)dp)$ in different ways. Property 1 of disjoint uni-decomposability implies that $\mathsf{P}^o(\mu t) \geq \Sigma_{i\in I}(\int_0^1 f^i(p)dp)$, whereas property 2 and 3 together imply $\mathsf{P}^o(\mu t) \leq \Sigma_{i\in I}(\int_0^1 f^i(p)dp)$. Lastly, note that the above equality holds for all $t \in T_{\Sigma'}(T_{\Sigma'}(\{*\}))$.

Suppose that $r \in T_{\Sigma'}(T_{\Sigma'}(\{*\}))$ is included in $((o_i)_{>p}((o'_i)_{>p'}(\{*\})$ for any $p, p' \in [0,1] \cap \mathbb{Q}$ such that $t \in (o_i)_{>p}((o'_i)_{>p'}(\{*\}))\}$, then for any p, $f^i_r(p) \geq f^i_t(p)$. Hence $\mathsf{P}^o(\mu r) = \Sigma_{i\in I}(\int_0^1 f^i_r(p)dp) \geq \Sigma_{i\in I}(\int_0^1 f^i_t(p)dp) = \mathsf{P}^o(\mu t) > q$. So we can conclude that $\mu r \in [\![o_{>q}]\!]$, which finishes the proof of strong decomposability. □

We can conclude that the following combinations are possible.

Corollary 5.3.12. *The effects of global store and input/output can be combined with probability. More formally: $\mathcal{O}'_{gs}$ and $\mathcal{O}'_{io}$ as defined above are strongly decomposable sets of Scott open modalities.*

5.3.5 Conclusions on combinations

The diagram given in Figure 5.2 shows which combinations have been proven to be possible in the framework of this dissertation. Start at *Errors*, and follow a choice of arrows to make a correct combination. Any subset of such a combination is also possible. The dotted lines with the crosses show combinations of two effects which are proven to be problematic in Subsection 3.5.4.

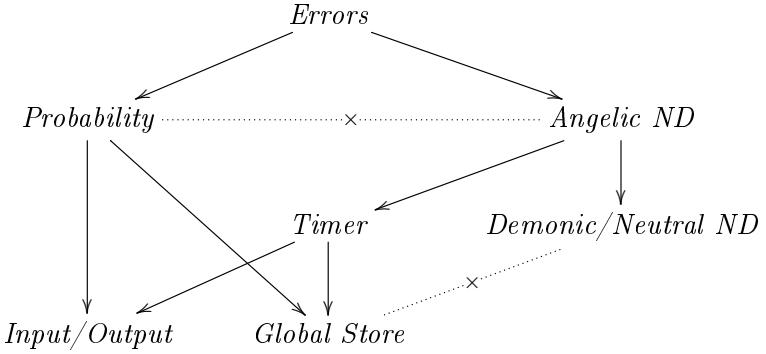


Figure 5.2: Proven combinations and impossibilities.

It may be possible to prove that other combinations of effects can also be interpreted by a decomposable set of modalities, e.g., global store with input/output. The above subsections should give sufficient ideas on how to prove that these combinations have a suitable decomposable set of Scott open modalities. However, since with the quantitative logic of Chapter 6, combinations of effects are more easily modelled, we will not attempt all the proofs here. The diagram in Figure 5.3 contains combinations of effects for which we conjecture to be able to find a decomposable set of Scott open modalities, together with the combinations for which we conjecture that this is impossible.

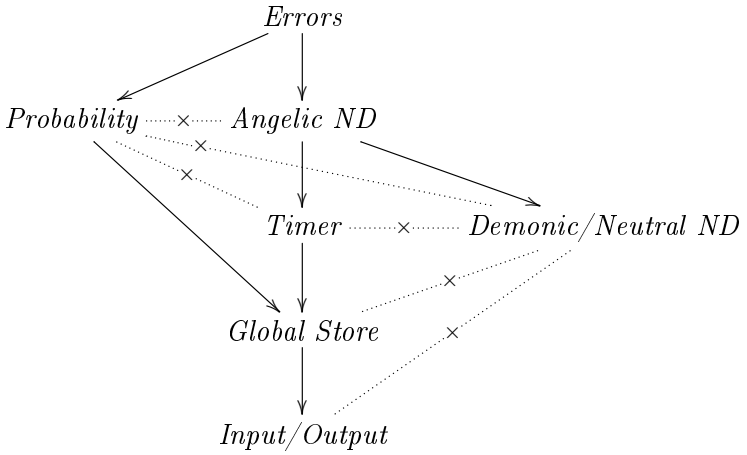


Figure 5.3: Conjectured combinations and impossibilities

5.4 Pure logic

In this section, we explore an alternative formulation of our logic which is independent of the term syntax of the programming language. This has both conceptual and practical motivations. Our very approach to behavioural logic fits into the framework of *endogenous* logics in the sense of Pnueli [87]. Formulas $\underline{\phi}$ express properties of individual programs, through satisfaction relations $\underline{P} \models \underline{\phi}$. Programs are thus considered as 'models' of the logic, with the satisfaction relation being defined via program behaviour.

We explore the possibility to express properties of program behaviour without prior knowledge of the term syntax of the programming language. Under our formulation of the logic $\mathcal{V}$, this idea is violated by the value formula $(V \mapsto \underline{\psi})$ at function type, which mentions the programming language value V.

We replace this basic value formula $(V \mapsto \underline{\psi})$ with the alternative $(\phi \mapsto \underline{\psi})$. Such a change also has a practical motivation. The formula $(\phi \mapsto \underline{\psi})$ declares a precondition and postcondition for function application, supporting a useful specification style.

Definition 5.4.1. The *pure behavioural logic* $\mathcal{F}$ is defined by replacing rule (6) of

Figure 3.1 with the alternative:

$$(6') \ \frac{\phi \in \mathit{Form}(\mathbf{A}) \qquad \underline{\psi} \in \mathit{Form}(\underline{\mathbf{C}})}{(\phi \mapsto \underline{\psi}) \in \mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})}$$

The semantics is specified by defining:

$$\underline{M} \models (\phi \mapsto \underline{\psi}) \quad :\Longleftrightarrow \quad \forall W : \mathbf{A}, (W \models \phi \Rightarrow \underline{M}\ W \models \underline{\psi})$$

We moreover define the *positive pure behavioural logic* $\mathcal{F}^+$ as the subset of $\mathcal{F}$ of formulas not using negation.

The letter 'F' in $\mathcal{F}$ stands for *Formula*, and is named as such because it uses a formula instead of a value in the basic formulas of function types. Two logics are said to be *equi-expressive* if any formula from one of the logics is equivalent to some formula of the other logic, meaning they are satisfied by the same terms. Given sufficient conditions, this new pure logic $\mathcal{F}$ is equi-expressive to $\mathcal{V}$.

Unlike in the proof of Lemma 4.2.6, proofs in this section use an induction on types. At this stage in the dissertation, this is not a problem. However, in Chapter 7, polymorphic and recursive types will be introduced, which will disallow the use of an induction on types. So proofs by induction on types will not carry over to languages containing such types. In the remainder of this chapter, we shall highlight whenever a result uses an induction on types. All other results of this chapter should be adaptable to the setting of Chapter 7, though we shall not include the proofs of these facts in this thesis.

Proposition 5.4.2. *If the open extension of $\sqsubseteq_{\mathcal{V}}$ is compatible then the logics $\mathcal{V}$ and $\mathcal{F}$ are equi-expressive. Similarly, if the open extension of $\sqsubseteq_{\mathcal{V}^+}$ is compatible then the positive fragments $\mathcal{V}^+$ and $\mathcal{F}^+$ are equi-expressive.*

Proof. The formula $(\phi \mapsto \underline{\psi})$ within $\mathcal{V}$, is equivalent to $\bigwedge\{(V \mapsto \underline{\psi}) \mid V \models \phi\}$. This can be used as the basis of an inductive translation from $\mathcal{F}$ to $\mathcal{V}$ (and from $\mathcal{F}^+$ to $\mathcal{V}^+$).

For the reverse translation, whose correctness proof is more interesting, we give a little bit more detail. By an induction on $\underline{\underline{\mathbf{E}}}$, we prove that any formula $\underline{\phi} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})_{\mathcal{V}}$ is equivalent to some formula $\widehat{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})_{\mathcal{F}}$. This is proven with an induction on $\underline{\phi}$. Since the two logics $\mathcal{F}$ and $\mathcal{V}$ only vary in one formula constructor, the one at function type, the only non-trivial case for this induction is when $\underline{\underline{\mathbf{E}}} = \mathbf{A} \to \underline{\mathbf{C}}$ and $\underline{\phi} = (V \mapsto \underline{\psi})$.

We can use Lemma 3.3.6 to find a formula $\chi_V \in \mathcal{V}$ such that: $W \models \chi_V \iff V \sqsubseteq_{\mathcal{V}} W$. Since $\sqsubseteq_{\mathcal{V}}$ is compatible, we know that if $\underline{M}\ V \models \underline{\psi}$ and $V \sqsubseteq_{\mathcal{V}} W$, then $\underline{M}\ W \models \underline{\psi}$. So $\underline{\phi} = (V \mapsto \underline{\psi}) \equiv (\chi_V \mapsto \underline{\psi})$, this is not a formula in either logic, but can be expressed in $\mathcal{V}$.

Using the induction hypothesis on $\mathbf{A}$ and $\underline{\mathbf{C}}$, we can find $\widehat{\chi_V} \in \mathit{Form}(\mathbf{A})_{\mathcal{F}}$ and $\widehat{\underline{\psi}} \in \mathit{Form}(\underline{\mathbf{C}})_{\mathcal{F}}$ such that $\chi_V \equiv \widehat{\chi_V}$ and $\underline{\psi} \equiv \widehat{\underline{\psi}}$. Hence we can derive that:

$$\underline{\phi} \ = \ (V \mapsto \underline{\psi}) \ \equiv \ (\chi_V \mapsto \underline{\psi}) \ \equiv \ (\widehat{\chi_V} \mapsto \widehat{\underline{\psi}}) \ \in \mathcal{F}.$$

So we define $\widehat{\underline{\phi}}$ as $(\widehat{\chi_V} \mapsto \widehat{\underline{\psi}})$. This finishes the inductive translation, and we can conclude that $\mathcal{V}$ and $\mathcal{F}$ are equi-expressive. By the same proof we know that $\mathcal{V}^+$ and $\mathcal{F}^+$ are also equi-expressive. □

Combining the above proposition with Theorem 3.3.8 we obtain the following.

Corollary 5.4.3. *Suppose $\mathcal{O}$ is a decomposable set of Scott-open modalities. Then $\equiv_{\mathcal{F}}$ coincides with $\equiv_{\mathcal{V}}$, and $\sqsubseteq_{\mathcal{F}^+}$ coincides with $\sqsubseteq_{\mathcal{V}^+}$ (applicative $\mathcal{O}$-bisimilarity and similarity respectively). Hence the open extensions of $\equiv_{\mathcal{F}}$ and $\sqsubseteq_{\mathcal{F}^+}$ are compatible.*

Alternatively, we could define the formula $\phi \hookrightarrow \underline{\psi}$ where:

$$\underline{M} \models \phi \hookrightarrow \underline{\psi} \iff \exists V,\ V \models \phi \ \wedge\ \underline{M}\ V \models \underline{\psi}$$

This formula can be defined in the pure logic $\mathcal{F}$ since $(\phi \hookrightarrow \underline{\psi}) \equiv \neg(\phi \mapsto \neg(\underline{\psi}))$. Moreover, it can be expressed in the positive pure logic if the behavioural preorder is compatible, since then $(\phi \hookrightarrow \underline{\psi}) \equiv \bigvee\{(\chi_V \mapsto \underline{\psi}) \mid V \models \phi\}$.

If $\equiv$ is compatible, the logic using $\phi \hookrightarrow \underline{\psi}$ instead of $(V \mapsto \underline{\psi})$ in $\mathcal{V}$ is equi-expressive to $\mathcal{V}$, since $(V \mapsto \underline{\psi}) \equiv (\chi_V \hookrightarrow \underline{\psi})$. This uses the fact that $W \models \chi_V$ holds if and only if $V \equiv W$. However, the same cannot be done for the positive logic, replacing $(V \mapsto \underline{\psi})$ by $\phi \hookrightarrow \underline{\psi}$ in $\mathcal{V}^+$. As an example, look at the type $\mathbf{UF1} \to \mathbf{F1}$ and two of its terms $\lambda x.\,\mathsf{return}(*)$ and $\lambda x.\,\mathsf{force}(x)$. These two terms are obviously not positively equivalent, since on input $\mathsf{thunk}(\Omega)$ the former terminates whereas the latter diverges. However, since $\mathsf{thunk}(\Omega) \sqsubseteq^{+}_{\mathbf{UF1}} \mathsf{thunk}(\mathsf{return}(*))$, the two terms cannot be distinguished in the new positive logic.

5.4.1 Eliminating computation formula connectives in the pure logic

Like for the general logic in Section 5.1, we can remove the connectives for computation formulas in the pure logic without changing the resulting logical equivalences. Recall the formulation of $\mathcal{L}^*$ from Definition 5.1.2.

Lemma 5.4.4. *Any $\underline{\phi} \in \mathcal{F}$ can be written, using an induction on types, as $\bigvee_{i\in I}\bigwedge_{j\in J_i}\underline{\psi}_{i,j}$ where each $\underline{\psi}_{i,j} \in \mathcal{F}^* \cup \neg\mathcal{F}^*$. Any $\underline{\phi} \in \mathcal{F}^+$ can be written, using an induction on types, as $\bigvee_{i\in I}\bigwedge_{j\in J_i}\underline{\psi}_{i,j}$ where each $\underline{\psi}_{i,j} \in (\mathcal{F}^+)^*$.*

Proof. We prove the first statement only, since the proof can be easily adapted for the second statement. We prove with induction on $\underline{\underline{\mathbf{E}}}$, followed by an induction on $\underline{\phi} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})_{\mathcal{F}}$, that $\underline{\phi}$ can be written as $\bigvee_{i\in I}\bigwedge_{j\in J_i}\underline{\psi}_{i,j}$ where $\underline{\psi}_{i,j} \in \mathcal{F}^* \cup \neg\mathcal{F}^*$. Note that in the case of $\underline{\phi}$ being a value formula, this $\bigvee_{i\in I}\bigwedge_{j\in J_i}\underline{\psi}_{i,j}$ is from $\mathcal{F}^*$.

We can use the proof of Lemma 5.1.3 for any case except when $\underline{\underline{\mathbf{E}}} = \mathbf{A} \to \underline{\mathbf{C}}$ and $\underline{\phi} = (\phi \mapsto \underline{\psi})$. Since value formulas are closed under conjunctions, we can find for each closed value term $V : \mathbf{A}$ a formula $\chi_V \in \mathit{Form}(\mathbf{A})$ characterising $V \sqsubseteq_{\mathcal{F}} (-)$ (c.f. Lemma 3.3.6). Since $\sqsubseteq_{\mathcal{F}}$ is compatible, $(V \mapsto \underline{\psi}) \equiv (\chi_V \mapsto \underline{\psi})$ as argued before. Now we use the induction hypothesis on each $\chi_V \in \mathit{Form}(\mathbf{A})$ and on $\underline{\psi} \in \mathit{Form}(\underline{\mathbf{C}})$ to find $\chi_V \equiv \chi'_V \in \mathcal{F}^*$ and $\underline{\psi} \equiv \bigvee_{i\in I}\bigwedge_{j\in J_i}\underline{\psi}_{i,j}$ with each $\underline{\psi}_{i,j} \in \mathcal{F}^* \cup \neg\mathcal{F}^*$. We perform the following derivation:

$$\underline{\phi} \equiv \bigwedge_{V,V\models\phi}(V \mapsto \underline{\psi}) \equiv \bigwedge_{V,V\models\phi}\left(V \mapsto \bigvee_{i\in I}\bigwedge_{j\in J_i}\underline{\psi}_{i,j}\right) \equiv \bigwedge_{V,V\models\phi}\bigvee_{i\in I}\bigwedge_{j\in J_i}(V \mapsto \underline{\psi}_{i,j}).$$

Now apply the equivalences $(V \mapsto \neg(\underline{\psi}')) \equiv \neg(V \mapsto \underline{\psi}')$ and $(V \mapsto \underline{\psi}') \equiv (\chi'_V \mapsto \underline{\psi}')$, and use Lemma 5.1.1 to establish that $\underline{\phi}$ is equivalent to a formula from $\mathcal{F}^*$ of the correct shape. □

Using the proof of Corollary 5.1.4 we can get the following result.

Corollary 5.4.5. *With an induction on types, it holds that* $(\equiv_{\mathcal{F}}) = (\equiv_{\mathcal{F}^*})$ *and* $(\sqsubseteq_{\mathcal{F}^+}) = (\sqsubseteq_{(\mathcal{F}^+)^*})$.

5.4.2 Finitary value formula connectives in the pure logic

How many of the results of Section 5.2 carry over to the pure logic $\mathcal{F}^*$? Not all of them, as we will see. Recall the definition of $(\mathcal{O}, a, b, c)$ from Definition 5.2.1. In this subsection, we similarly define $(\mathcal{O}, a, b, c)_{\mathcal{F}}$ following Definition 5.2.1, using rule (6') from Definition 5.4.1 instead of rule (6). If we take $(\mathcal{L})^*$ as in Definition 5.1.2, then $(\mathcal{O}, \bigvee, \bigwedge, \neg)_{\mathcal{F}}$ corresponds to $(\mathcal{F})^*$, and $(\mathcal{O}, \bigvee, \bigwedge, +)_{\mathcal{F}}$ to $(\mathcal{F}^+)^*$.

Firstly, it is still possible to remove disjunctions once all modalities distribute over non-empty disjunctions. This is due to the following observation:

$$\begin{aligned} & \underline{M} \models \bigvee_{i \in I} \phi_i \ \mapsto \underline{\phi} \\ \iff \quad & \forall V, ((\exists i \in I, V \models \phi_i) \Rightarrow \underline{M}\ V \models \underline{\phi}) \\ \iff \quad & \forall V, \forall i \in I, (V \models \phi_i \Rightarrow \underline{M}\ V \models \underline{\phi}) \\ \iff \quad & \underline{M} \models \bigwedge_{i \in I} (\phi_i \mapsto \underline{\phi}). \end{aligned}$$

Hence $(\bigvee_{i \in I} \phi_i \mapsto \underline{\phi}) \equiv \bigwedge_{i \in I} (\phi_i \mapsto \underline{\phi})$.

Lemma 5.4.6. *Suppose* $\mathcal{O}$ *is a set of modalities such that all* $o \in \mathcal{O}$ *distribute over non-empty disjunctions, then by induction on types it holds that* $(\sqsubseteq_{(\mathcal{O}, \bigvee, \bigwedge, \neg)_{\mathcal{F}}}) = (\sqsubseteq_{(\mathcal{O}, \bot, \bigwedge, \neg)_{\mathcal{F}}})$ *and* $(\sqsubseteq_{(\mathcal{O}, \bigvee, \bigwedge, +)_{\mathcal{F}}}) = (\sqsubseteq_{(\mathcal{O}, \bot, \bigwedge, +)_{\mathcal{F}}})$.

Proof. Let $c \in \{\neg, +\}$, and take $\mathcal{L} = (\mathcal{O}, \bigvee, \bigwedge, c)_{\mathcal{F}}$ and $\mathcal{K} = (\mathcal{O}, \bot, \bigwedge, c)_{\mathcal{F}}$. We prove that each formula $\underline{\phi} \in \mathcal{L}$ (value or computation) is equivalent to a disjunction $\bigvee_{i \in I} \underline{\psi}_i$ over formulas $\underline{\psi}_i$ from $\mathcal{K}$. We prove this by induction on $\underline{\mathbf{E}}$, where of each such type we do an induction on $\underline{\phi} \in \mathit{Form}(\underline{\mathbf{E}})_{\mathcal{L}}$.

The proofs of all cases but $\underline{\mathbf{E}} = \mathbf{A} \to \underline{\mathbf{C}}$ and $\underline{\phi} = (\psi \mapsto \underline{\phi})$ are as in the proof of Lemma 5.2.5. Assume $\underline{\mathbf{E}} = \mathbf{A} \to \underline{\mathbf{C}}$ and $\underline{\phi} = (\psi \mapsto \underline{\phi})$. Since the value formulas of $\mathcal{L}$ are still closed under conjunctions, we can for each $V : \mathbf{A}$ find a characterising formula $\chi_V \in \mathcal{L}$ (c.f. Lemma 3.3.6). By induction hypothesis, $\chi_V \equiv \bigvee_{i \in I_V} \chi_{V,i}$ and $\underline{\phi} \equiv \bigvee_{j \in J} \underline{\phi}_i$ for some selection of formulas $\chi_{V,i} \in \mathcal{K}$ and $\underline{\phi}_j \in \mathcal{K}$.

$$\underline{\phi} = (\psi \mapsto \underline{\phi}) \quad \equiv \bigwedge_{V:\mathbf{A}, V \models \psi} (V \mapsto \underline{\phi}) \qquad\qquad \equiv \bigwedge_{V:\mathbf{A}, V \models \psi} (V \mapsto \bigvee_{j \in J} \underline{\phi}_j)$$

$$\equiv \bigwedge_{V:\mathbf{A},V\models\psi} \bigvee_{j\in J} (V \mapsto \underline{\phi}_j) \qquad \equiv \bigwedge_{V:\mathbf{A},V\models\psi} \bigvee_{j\in J} (\chi_V \mapsto \underline{\phi}_j)$$

$$\equiv \bigwedge_{V:\mathbf{A},V\models\psi} \bigvee_{j\in J} \left(\bigvee_{i\in I_V} \chi_{V,i} \mapsto \underline{\phi}_j \right) \qquad \equiv \bigwedge_{V:\mathbf{A},V\models\psi} \bigvee_{j\in J} \bigwedge_{i\in I_V} (\chi_{V,i} \mapsto \underline{\phi}_j)$$

where each $(\chi_{V,i} \mapsto \underline{\phi}_j) \in \mathcal{K}$. We can swap the first two connectives using Lemma 5.1.1 to get a formula of the desired shape. □

By adapting the proof of Lemma 5.4.6, like Lemma 5.2.6 did with Lemma 5.2.5, we get the following result.

Lemma 5.4.7. *Suppose $\mathcal{O}$ is a set of Scott-open modalities, then by induction on types it holds that $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,\neg)_\mathcal{F}}) = (\sqsubseteq_{(\mathcal{O},\vee,\bigwedge,\neg)_\mathcal{F}})$ and $(\sqsubseteq_{(\mathcal{O},\bigvee,\bigwedge,+)_\mathcal{F}}) = (\sqsubseteq_{(\mathcal{O},\vee,\bigwedge,+)_\mathcal{F}})$.*

Negation can again be absorbed into the modalities in the following way.

Lemma 5.4.8. *By induction on types, $(\equiv_{(\mathcal{O},\bigvee,\bigwedge,\neg)_\mathcal{F}}) = (\equiv_{(\mathcal{O}\cup\mathcal{O}_\neg,\bigvee,\bigwedge,+)_\mathcal{F}})$.*

Proof. Let $\mathcal{L} = (\mathcal{O}, \bigvee, \bigwedge, \neg)_\mathcal{F}$ and $\mathcal{K} = (\mathcal{O} \cup \mathcal{O}_\neg, \bigvee, \bigwedge, +)_\mathcal{F}$. We adapt the proof of Lemma 5.2.10 by doing the two inductions simultaneously. With an induction on $\underline{\underline{\mathbf{E}}}$ we prove that:

1. For any $\underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})_\mathcal{K}$ there is a $\underline{\underline{\psi}} \in \mathcal{K}$ such that $\neg(\underline{\underline{\phi}}) \equiv \underline{\underline{\psi}}$.

2. For any $\underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})_\mathcal{L}$ there is a $\underline{\underline{\psi}} \in \mathcal{K}$ such that $\underline{\underline{\phi}} \equiv \underline{\underline{\psi}}$.

Both these statements are proven with an induction on $\underline{\underline{\phi}}$. Almost all of these cases are identical to their proof in Lemma 5.2.10, where (1) corresponds to the first induction in that proof, and (2) corresponds to the second induction in that proof.

The only non-trivial case (again) is proving (1) for $\underline{\underline{\mathbf{E}}} = \mathbf{A} \to \underline{\mathbf{C}}$ and $\underline{\underline{\phi}} = (\psi \mapsto \underline{\phi})$. By the induction hypothesis, using statement (1), there is a formula $\underline{\psi} \in \mathcal{K}$ such that $\neg(\underline{\phi}) \equiv \underline{\psi}$. For each $V : \mathbf{A}$, there is a characterising formula $\chi_V \in \mathit{Form}(\mathbf{A})_\mathcal{L}$. By the induction hypothesis, using statement (2), there is a formula $\chi'_V \in \mathit{Form}(\mathbf{A})_\mathcal{K}$ equivalent to χ_V. We have all the tools to make the derivation:

$$\neg(\underline{\underline{\phi}}) = \neg(\psi \mapsto \underline{\phi}) \equiv \neg \left(\bigwedge_{V:\mathbf{A},V\models\psi} (V \mapsto \underline{\phi}) \right) \equiv \bigvee_{V:\mathbf{A},V\models\psi} \neg(V \mapsto \underline{\phi})$$

$$\equiv \bigvee_{V:\mathbf{A},V\models\psi} (V \mapsto \neg(\underline{\phi})) \equiv \bigvee_{V:\mathbf{A},V\models\psi} (\chi_V \mapsto \neg(\underline{\phi})) \equiv \bigvee_{V:\mathbf{A},V\models\psi} (\chi'_V \mapsto \underline{\psi}) \in \mathcal{K}$$

So we are done with the inductions, and like for Lemma 5.2.10 we can conclude that $\mathcal{L} \cup \neg\mathcal{L}$ and $\mathcal{K}$ are equi-expressive, hence $(\equiv_{(\mathcal{O},\bigvee,\bigwedge,\neg)_\mathcal{F}}) = (\equiv_{(\mathcal{O}\cup\mathcal{O}_\neg,\bigvee,\bigwedge,+)_\mathcal{F}})$ holds. □

Adapting Lemma 5.4.6, we get a result which works well in combination with Lemma 5.4.8.

Lemma 5.4.9. *If all modalities $o \in \mathcal{O}$ distribute over non-empty disjunctions, then by induction on types* $(\underline{\sqsubseteq}_{(\mathcal{O}\cup\mathcal{O}_\neg,\bigvee,\bigwedge,+)_\mathcal{F}}) = (\underline{\sqsubseteq}_{(\mathcal{O}\cup\mathcal{O}_\neg,\bot,\bigwedge,+)_\mathcal{F}})$.

Proof. Adapt the proof of 5.4.6, finding for each formula $\underline{\phi}$ of $(\mathcal{O}\cup\mathcal{O}_\neg,\bigvee,\bigwedge,+)_\mathcal{F}$, an equivalent disjunction of formulas from $(\mathcal{O}\cup\mathcal{O}_\neg,\bot,\bigwedge,+)_\mathcal{F}$ by induction on types.

The only case not covered is when $\underline{\phi} = o_\neg(\psi) \equiv \neg(o(\psi))$. By induction hypothesis, $\psi \equiv \bigvee_i \psi_i$, so $\underline{\phi} \equiv \neg(o(\bigvee_i \psi_i)) \equiv \neg(\bigvee_i o(\psi_i)) \equiv \bigwedge_i \neg(o(\psi_i))$ and we are finished. □

The above lemma can also be established for the general logic $\mathcal{V}$. However, as we have shown, the logic can be reduced much further for most examples of effects. For the pure logic $\mathcal{F}$, this seems to be the most we can do. Applying the previous results, we get that:

Proposition 5.4.10. *For effect-free, error, input-output, global store and angelic non-deterministic computations, it holds by induction on types that:* $(\equiv_{(\mathcal{O},\bigvee,\bigwedge,\neg)_\mathcal{F}}) = (\equiv_{(\mathcal{O}\cup\mathcal{O}_\neg,\bot,\bigwedge,+)_\mathcal{F}})$.

There is a striking difference between the value logic $\mathcal{V}$ and the pure logic $\mathcal{F}$. Even if the modalities distribute over non-empty conjunctions, it is in general not possible to remove conjunctions from the pure logic. We look in particular to global store, assuming for simplicity that there is only one store location. Modalities will be given by $(n \rightarrowtail m)$ where $n, m \in \mathbb{N}$ give possible values stored in that single store location. Even though all modalities $(n \rightarrowtail m) \in \mathcal{O}_{\text{gs}}$ distribute over non-empty conjunctions, we cannot get rid of the conjunctions. This is in stark contrast with the value logic $\mathcal{V}$, where according to Lemma 5.2.7, this is possible.

Proposition 5.4.11. *The pure logic $(\mathcal{O}_{gs},\bigvee,\top,\neg)_\mathcal{F}$ does not give a compatible logical equivalence for global store.*

We prove this in the case of a single memory location.

Proof. We study the type $\mathbf{U}\,\mathbf{F}\,\mathbf{1} \to \mathbf{F}\,\mathbf{1}$, and two of its terms:

$$\underline{M} := \lambda x.\,\mathsf{update}(0;\ \ \mathsf{force}(\mathsf{return}(x)); \mathsf{update}(1;\ \mathsf{force}(\mathsf{return}(x))))$$

$$\underline{N} := \lambda x.\,\mathsf{update}(0;\ \ \mathsf{force}(\mathsf{return}(x)); \mathsf{update}(2;\ \mathsf{force}(\mathsf{return}(x)))).$$

Note that these terms use sequencing of computations $\underline{P};\underline{Q}$. We prove that $\underline{M} \equiv_{(\mathcal{O}_{\text{gs}},\bigvee,\top,\neg)_\mathcal{F}} \underline{N}$.

Assume that $\underline{M} \models (\phi \mapsto \underline{\phi})$, where since $(\bigvee_{i\in I}\phi_i \mapsto \underline{\phi}) \equiv \bigwedge_{i\in I}(\phi_i \mapsto \underline{\phi})$ and $\neg(\langle\underline{\psi}\rangle) \equiv \langle\neg(\underline{\psi})\rangle$, we can assume without loss of generality that ϕ is either of the form $\langle o(\psi)\rangle$ or $\langle\neg(o(\psi))\rangle$, where $\psi \in \{\bot,\top\}$ (the only two formulas of type $\mathbf{1}$). We use the fact that any computation formula $\underline{\phi}$ satisfied by a diverging computation must be equivalent to $\top$.

- Suppose $\phi = \langle(n \rightarrowtail m)(\top)\rangle$ with $n \neq 0$. Take V to be a term satisfying ϕ, such that $\mathsf{force}(V)$ diverges when initiated with starting state 0. It is possible to find

such a term, because the formula ϕ only checks what happens when $\mathsf{force}(V)$ is imitated with state n. For this V, $\underline{M}\ V$ diverges, and since $\underline{M}\ V \models \underline{\phi}$, $\underline{\phi}$ must be equivalent to $\top$. Hence, $(\phi \mapsto \underline{\phi}) \equiv \top$, so trivially $\underline{N} \models (\phi \mapsto \underline{\phi})$.

- Suppose $\phi = \langle (0 \rightarrowtail m)(\top) \rangle$. Take V to be a term satisfying ϕ, such that $\mathsf{force}(V)$ diverges when initiated with starting state 1. Then $\underline{M}\ V$ diverges, so $\underline{\phi} \equiv \top$, and hence $\underline{N} \models (\phi \mapsto \underline{\phi})$.

- Suppose $\phi = \langle (n \rightarrowtail m)(\bot) \rangle$, then $\phi \equiv \bot$, so $\underline{N}$ (vacuously) satisfies $(\phi \mapsto \underline{\phi})$.

- Suppose $\phi = \neg(\langle (n \rightarrowtail m)(\bot) \rangle)$, then $\phi = \top$, so $\underline{\phi}$ can only be $\top$ (by the same proof as in the first point), so $\underline{N}$ satisfies $(\phi \mapsto \underline{\phi})$.

- Suppose $\phi = \neg(\langle (0 \rightarrowtail m)(\top) \rangle)$, then for any V such that $V \models \phi$, both $\underline{M}\ V$ and $\underline{N}\ V$ diverge, hence $\underline{\phi} \equiv \bot$ and so $\underline{N}$ satisfies $(\phi \mapsto \underline{\phi})$.

- Suppose $\phi = \neg(\langle (n \rightarrowtail m)(\top) \rangle)$ with $n \neq 0$. Take V such that $\mathsf{force}(V)$ always diverges. Then $V \models \phi$ and $\underline{M}\ V$ diverges, so $\underline{\phi} \equiv \top$, and hence $\underline{N} \models (\phi \mapsto \underline{\phi})$.

Whatever the case, $\underline{N} \models (\phi \mapsto \underline{\phi})$ holds. For negated formulas $\neg(\phi \mapsto \underline{\phi})$, just let $\underline{M}$ and $\underline{N}$ swap places in the above analysis. We can conclude that $\underline{M} \equiv_{(\mathcal{O}_{\mathrm{gs}}, \bigvee, \top, \neg)_{\mathcal{F}}} \underline{N}$.

However, let V be the value which, when forced will only diverge when starting with state 2. Else, it simply terminates with the state untouched. Then $\underline{M}\ V$ always terminates, and $\underline{N}\ V$ always diverges. So $\underline{M}\ V \not\equiv_{(\mathcal{O}_{\mathrm{gs}}, \bot, \bigvee, \neg)_{\mathcal{F}}} \underline{N}\ V$. We must conclude that $\equiv_{(\mathcal{O}_{\mathrm{gs}}, \bot, \bigvee, \neg)_{\mathcal{F}}}$ is not compatible.

□

5.5 Logical statements

The propositional logic used in this thesis to specify behavioural equivalence is a low level idealized mathematical logic with infinitary connectives. The formulas are not always very natural for reasoning about programs, but the logic can be seen as a vehicle into which higher level logics can be translated. In this section we will see an example of how we can express more natural statements about effects in the behavioural logic.

Firstly, note that standard logical statements like implications and quantification can be expressed in the infinitary propositional logic. Take for instance the following two examples:

$$\underline{\phi} \Rightarrow \underline{\psi} \quad := \quad \neg(\underline{\phi}) \vee \underline{\psi}$$

$$\exists V : \mathbf{A}, (V \mapsto \underline{\phi}) \quad := \quad \bigvee \{ (V \mapsto \underline{\phi}) \mid V : \mathbf{A} \}$$

As an example of how our behavioural logic can express natural statements about effects, we look at global store and Hoare logic.

5.5.1 Global store and Hoare logic

Hoare logic [28, 88] is given as an axiomatic foundation for imperative languages. An expression in the logic is given by a triple $\{A\}\underline{M}\{B\}$, which states that when the starting state satisfies precondition A, then evaluating $\underline{M}$, if it terminates, will yield an end state satisfying postcondition B. There is a difference between partial correctness and total correctness, where the latter requires $\underline{M}$ to terminate whereas the former does not. We can express both such interpretations of Hoare logic statements in the full logic for global store.

We assume that the store locations are given by a set Loc. Values stored in these locations are natural numbers, and an allocation of numbers to the variables is called a state $s \in \mathsf{State}$. An *assertion* of the global store is a set of such states. We can use arithmetic expressions to describe such assertions, for instance for $x, y, z \in \mathsf{Loc}$, $\{x = 3, z = y^2\} := \{s \in \mathsf{State} \mid s(x) = 3, s(z) = s(y)^2\}$.

Given two state assertions A and B, and a computation $\underline{M} : \mathbf{F}\,\mathbf{1}$, we can express Hoare triple statements in our logic $\mathcal{V}$ with modalities $\mathcal{O}_{\mathrm{gs}}$ as:

$$\{A\}\underline{M}\{B\}_t \quad := \quad \underline{M} \models \bigwedge_{s\in A} \bigvee_{r\in B} (s \rightarrowtail r)(\top)$$

$$\{A\}\underline{M}\{B\}_p := \quad \underline{M} \models \bigwedge_{s\in A} \left(\left(\bigvee_{r\in \mathsf{State}} (s \rightarrowtail r)(\top) \right) \Rightarrow \left(\bigvee_{r\in B} (s \rightarrowtail r)(\top) \right) \right) \ .$$

Note that uncountable conjunctions and disjunctions are used. However, as commented before, since there are only countably many terms, the above statements are equivalent to formulas in the logic $\mathcal{V}$.

The first implements *total correctness*, saying that for any starting state $s \in A$, $\underline{M}$ terminates with a final state satisfying B. The second implements a notion of *partial correctness*, saying that for any starting state satisfying A, if $\underline{M}$ terminates, then it terminates with a final state satisfying B.

Traditionally, Hoare logic has been formulated for an imperative language with while-loops. There is a natural translation of programs of such a language into terms of type $\mathbf{F}\,\mathbf{1}$. Given such a translation, the above formulas accurately reflect the Hoare triples from traditional Hoare logics. More generally, terms of type $\mathbf{F}\,\mathbf{A}$ can return values of type $\mathbf{A}$ whose properties can be examined further. It is natural to incorporate information about the returned values in the Hoare triples, as is done in [66, 67, 88].

It therefore may be nice to generalise Hoare triple expressions in the following way, where given a computation $\underline{M} : \mathbf{F}\,\mathbf{A}$, two assertions A and B on the state and a formula $\phi \in \mathit{Form}(\mathbf{A})$:

$$\{A\}\underline{M}\{B;\phi\}_t \quad := \quad \underline{M} \models \bigwedge_{s\in A} \bigvee_{r\in B} (s \rightarrowtail r)(\phi)$$

$$\{A\}\underline{M}\{B;\phi\}_p \quad := \quad \underline{M} \models \bigwedge_{s\in A} \left(\bigvee_{r\in \mathsf{State}} (s \rightarrowtail r)(\top) \right) \Rightarrow \left(\bigvee_{r\in B} (s \rightarrowtail r)(\phi) \right) \ .$$

These express Hoare triples as explained above, with the added requirement that if $\underline{M}$ produces a value, then that value satisfies ϕ.

5.6 Proof rules

The previous section showed us a way in which a high level logic could be translated into our low level infinitary propositional logic. Such high level logics are more suitable for reasoning about effects. However, that does not mean we cannot reason with our low level logic. We can develop nice compositional proof rules for our modal logic which can be used to verify that certain properties hold. These proof rules are not meant to be useful in practise, but are there to illustrate what proof rules would look like when translated to high level logics.

We begin our study of compositional proof rules by looking at sequencing terms $\underline{M}$ to $x.\,\underline{N}$. We focus on this example since understanding the sequencing of effectful programs is critical for our understanding of effects themselves. Moreover, this sequencing has a simple interaction with modal properties. This interaction is closely related to the properties of decomposability and strong decomposability from Subsection 3.3.2.

If we have a strong decomposable set of modalities, then there are proof rules of a particularly nice form: They are specified by a modality $o \in \mathcal{O}$ together with a family of pairs of modalities: $\{(o_i, o_i')\}_{i\in I}$. This data determines the *sequencing proof rule* below, which makes use of the pure behavioural logic $\mathcal{F}$ from Section 5.4.

$$\frac{\{\, \underline{M} \models o_i(\psi_i) \qquad \lambda x : \mathbf{A}.\, \underline{N} \models (\psi_i \mapsto o_i'(\phi))\,\}_{i\in I}}{\underline{M} \text{ to } x.\, \underline{N} \models o(\phi)} \tag{5.1}$$

In this rule, the modalities o, o_i and o_i' and their index set I have been determined by the specifying data. The other components of the rule may be instantiated arbitrarily, thus the rule is parametric with respect to terms $\underline{M}$ and $\underline{N}$ and the formulas ϕ, ϕ_i and ψ_i. The rule is compositional in the sense that its premises require properties to be established for $\underline{M}$ and $\underline{N}$ separately.

Definition 5.6.1. A sequencing proof rule of the form (5.1) is said to be *sound* if, for all types $\mathbf{A}, \mathbf{B}$, all computation terms $\vdash \underline{M} : \mathbf{F}\,\mathbf{A}$ and $x : \mathbf{A} \vdash \underline{N} : \mathbf{F}\,\mathbf{B}$, and all value formulas $\phi \in \mathit{Form}(\mathbf{B})$ and $\{\psi_i \in \mathit{Form}(\mathbf{A})\}_{i\in I}$, the following implication holds:

$$\forall i \in I.\, (\, \underline{M} \models o_i(\psi_i) \;\wedge\; \lambda x : \mathbf{A}.\, \underline{N} \models (\psi_i \mapsto o_i'(\phi))\,) \quad \Longrightarrow \quad \underline{M} \text{ to } x.\, \underline{N} \models o(\phi)\ .$$

There is a precise connection between the above proof rules, and strong decomposability. Whenever $\mathcal{O}$ is a strongly decomposable set of Scott open modalities, there is a set of sound sequencing proof rules, which is complete in the sense of the proposition below, which informally states: Any true modal property of a sequencing term $\underline{M}$ to $x.\,\underline{N}$ can be proven from properties of its subterms $\underline{M}$ and $\underline{N}$ using some sound sequencing proof rule.

Proposition 5.6.2. *Suppose that $\mathcal{O}$ is a strongly decomposable set of Scott open modalities. Suppose also that $\underline{M'}$ to $x.\,\underline{N'} \models o\,\phi'$ holds (where $\vdash \underline{M'} : \mathbf{F}\,\mathbf{A'}$ and $x : \mathbf{A'} \vdash \underline{N'} : \mathbf{F}\,\mathbf{B'}$). Then there is a sound sequencing proof rule of the form (5.1) for o, with premise modalities $\{(o_i, o_i')\}_{i\in I}$, such that*

$$\forall i \in I.\, (\, \underline{M'} \models o_i\,\psi_i' \;\wedge\; \lambda x : \mathbf{A'}.\, \underline{N'} \models \psi_i' \mapsto o_i'\,\phi'\,)\ ,$$

for some choice of value formulas $\{\psi'_i \in \mathit{Form}(\mathbf{A}')\}_{i\in I}$.

The role of Scott openness in the formulation of this proposition is purely to ensure that the compatibility property holds, which is used in the proof.

Proof. Assume that $\mathcal{O}$ is strongly decomposable and $\underline{M'}$ to $x.\,\underline{N'} \models o\,\phi'$. By Corollary 2.2.10, $|\underline{M'} \text{ to } x.\,\underline{N'}| = \mu(|\underline{M'}|[V \mapsto |\underline{N'}[V/x]|])$, so

$$\mu(|\underline{M'}|[V \mapsto |\underline{N'}[V/x]|[\models \phi']]) \;=\; \mu(|\underline{M'}|[V \mapsto |\underline{N'}[V/x]|])[\models \phi'] \;\in\; o(\{*\}) \;.$$

By strong decomposability, there is a collection $\{(o_i, o'_i)\}_{i\in I}$ of pairs of modalities s.t.:

$$(1)\;\; \forall i \in I, |\underline{M'}|[V \mapsto |\underline{N'}[V/x]|[\models \phi']] \in o_i(o'_i(\{*\})),$$

$$(2)\;\; \forall r \in TT\mathbf{1},\; \text{if } \forall i \in I, r \in o_i(o'_i(\{*\})) \text{ then } \mu r \in o(\{*\}).$$

We prove that $\{(o_i, o'_i)\}_{i\in I}$ specifies the desired sequencing proof rule for o.

Soundness: To prove that the proof rule is sound, consider types $\mathbf{A}$ and $\mathbf{B}$, terms $\vdash \underline{M}\colon \mathbf{F}\,\mathbf{A}$ and $x\colon \mathbf{A} \;\vdash\; \underline{N}\colon \mathbf{F}\,\mathbf{B}$, and value formulas $\phi \in \mathit{Form}(\mathbf{B})$ and $\{\psi_i \in \mathit{Form}(\mathbf{A})\}_{i\in I}$ such that:

$$\forall i \in I.\,(\underline{M} \models o_i\,\psi_i \;\text{ and }\; \lambda x\colon \mathbf{A}.\,\underline{N} \models \psi_i \mapsto o'_i\,\phi\,) \;.$$

For any $i \in I$ and V such that $V \models \psi_i$, we have $|\underline{N}[V/x]|[\models \phi] \in o'_i(\{*\})$. Defining $r = |\underline{M}|[V \mapsto |\underline{N}[V/x]|[\models \phi]]$, we have by monotonicity and because $|\underline{M}|[\models \psi_i] \in o_i(\{*\})$, that $r \in o_i(o'_i(\{*\}))$. This holds for all $i \in I$, so by property (2) of strong decomposability (given above), $\mu r \in o(\{*\})$, hence $|\underline{M} \text{ to } x.\,\underline{N}|[\models \phi] \in o(\{*\})$ and we conclude that $\underline{M}$ to $x.\,\underline{N} \models o\,\phi$.

Completeness: We now prove that suitable value formulas $\{\psi'_i \in \mathit{Form}(\mathbf{A}')\}_{i\in I}$ exist, allowing us to use the rule as a method for proving that $\underline{M'}$ to $x.\,\underline{N'} \models o\,\phi'$. For any $i \in I$, define

$$\psi'_i := \bigvee\{\chi_V \mid V : \mathbf{A}',\; \underline{N'}[V/x] \models o'_i\,\phi'\} \;,$$

where χ_V is the characteristic function for V, as given by Lemma 3.3.6. So if $W \models \chi_V$ then $V \sqsubseteq W$, and since $\underline{N'}[V/x] \models o'_i\,\phi'$ and $\sqsubseteq$ is compatible, $\underline{N'}[W/x] \models o'_i\,\phi'$. Hence, whenever $W \models \psi'_i$, we have that $\underline{N'}[W/x] \models o'_i\,\phi'$, so $\lambda x\colon \mathbf{A}'.\,\underline{N} \models \psi'_i \mapsto o'_i\,\phi'$.

By the definition of ψ'_i and the observation above, it holds that, for any closed value term $W\colon \mathbf{A}'$, $W \models \psi$ if and only if $|\underline{N'}[W/x]|[\models \phi'] \in o'_i(\{*\})$. By property (1) above of strong decomposability, $|\underline{M'}|[V \mapsto |\underline{N'}[V/x]|[\models \phi']] \in o_i(o'_i(\{*\}))$, so $|\underline{M'}|[\models \psi'] \in o_i(\{*\})$, and hence $\underline{M'} \models o_i(\psi')$. □

The completeness argument relies on establishing a weakest precondition in the form of ψ'_i, following completeness arguments established by [11].

We remark that Proposition 5.6.2 also works if we restrict to the positive logics. We give examples of complete collections of sound sequencing proof rules in the case of our effect examples.

For error, we have the following sequence proof rules:

$$\frac{\underline{M} \models \downarrow(\psi) \qquad \lambda x : \mathbf{A}.\, \underline{N} \models (\psi \mapsto \mathsf{E}_e(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models \mathsf{E}_e(\phi)} \qquad \frac{\underline{M} \models \mathsf{E}_e(\bot)}{\underline{M} \text{ to } x.\, \underline{N} \models \mathsf{E}_e(\phi)}.$$

The second rule has been simplified by instantiating $\bot$ for ψ, which can be done in any situation where the rule is applicable. For that rule, the second premise as given in (5.1) drops out, since any formula of the form $(\bot \mapsto \underline{\psi})$ is always satisfied. Note that since E_e is a nullary modality, the first statement can be equivalently formulated by substituting $\bot$ for ϕ.

For nondeterministic choice, the collection of rules is given by:

$$\frac{\underline{M} \models \Diamond(\psi) \qquad \lambda x : \mathbf{A}.\, \underline{N} \models (\psi \mapsto \Diamond(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models \Diamond(\phi)} \qquad \frac{\underline{M} \models \Box(\psi) \qquad \lambda x : \mathbf{A}.\, \underline{N} \models (\psi \mapsto \Box(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models \Box(\phi)}.$$

For probabilistic choice, such a collection of rules is given by:

$$\frac{\{\, \underline{M} \models \mathsf{P}_{>a_i}\, \psi_i \qquad \lambda x\colon \mathbf{A}.\, \underline{N} \models (\psi_i \mapsto \mathsf{P}_{>b_i}(\phi))\}_{i=1}^{n}}{\underline{M} \text{ to } x.\, \underline{N} \models \mathsf{P}_{>q}(\phi)}\ .$$

where each rule in the collection is specified by some natural number $n \geq 1$ and sequences of rationals $0 < a_0 < \cdots < a_n < 1$ and $1 > b_0 > \cdots > b_n > 0$ satisfying the following inequation: $a_0 b_0 + \sum_{i=1}^{n} (a_i - a_{i-1}) b_i \geq q$. Completeness can be derived from Proposition 5.6.2, using the fact that such collections $\{(\mathsf{P}_{>a_i}, \mathsf{P}_{>b_i})\}_{i=1}^{n}$ arise in the proof of strong decomposability for the probability modalities in Subsection 3.3.3.

For global store, such a collection is given by sequencing proof rules of the form:

$$\frac{\underline{M} \models (s \rightarrowtail s'')(\psi) \qquad \lambda x\colon \mathbf{A}.\, \underline{N} \models (\psi \mapsto (s'' \rightarrowtail s')(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models (s \rightarrowtail s')(\phi)}\ .$$

For input/output:

$$\frac{\underline{M} \models \langle v \rangle\!\downarrow(\psi) \qquad \lambda x\colon \mathbf{A}.\, \underline{N} \models (\psi \mapsto \langle w \rangle\!\downarrow(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models \langle vw \rangle\!\downarrow(\phi)}$$

$$\frac{\underline{M} \models \langle v \rangle\!\downarrow(\psi) \qquad \lambda x\colon \mathbf{A}.\, \underline{N} \models (\psi \mapsto \langle w \rangle_{\ldots}(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models \langle vw \rangle_{\ldots}(\phi)} \qquad \frac{\underline{M} \models \langle w \rangle_{\ldots}(\bot)}{\underline{M} \text{ to } x.\, \underline{N} \models \langle w \rangle_{\ldots}(\phi)}\ .$$

Like in the case of error, the second rule has been simplified.

Last but not least, the timer effect.

$$\frac{\underline{M} \models \mathsf{C}_{\leq a}(\psi) \qquad \lambda x : \mathbf{A}.\, \underline{N} \models (\psi \mapsto \mathsf{C}_{\leq b}(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models \mathsf{C}_{\leq a+b}(\phi)}$$

$$\frac{\underline{M} \models \mathsf{C}_{\geq a}(\psi) \qquad \lambda x : \mathbf{A}.\, \underline{N} \models (\psi \mapsto \mathsf{C}_{\geq b}(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models \mathsf{C}_{\geq a+b}(\phi)}$$

$$\frac{\underline{M} \models \mathsf{C}_{\geq a}(\psi) \qquad \lambda x : \mathbf{A}.\, \underline{N} \models (\psi \mapsto \mathsf{C}^{\uparrow}_{>b}(\phi))}{\underline{M} \text{ to } x.\, \underline{N} \models \mathsf{C}^{\uparrow}_{>a+b}(\phi)} \qquad \frac{\underline{M} \models \mathsf{C}^{\uparrow}_{>a}(\bot)}{\underline{M} \text{ to } x.\, \underline{N} \models \mathsf{C}^{\uparrow}_{>a}(\phi)}\ .$$

5.6.1 Other proof rules

Proof rules for other constructors of the programming language also exist. We illustrate this for those constructors that concern themselvers directly with effects, since these have interesting interactions with modalities.

We first give a list of rules for the $\mathsf{return}(-)$ constructor:

$$\frac{V \models \phi}{\mathsf{return}(V) \models \downarrow(\phi)} \qquad \frac{V \models \phi}{\mathsf{return}(V) \models \Diamond(\phi)} \qquad \frac{V \models \phi}{\mathsf{return}(V) \models \Box(\phi)}$$

$$\frac{V \models \phi \qquad q < 1}{\mathsf{return}(V) \models \mathsf{P}_{>q}(\phi)} \qquad \frac{V \models \phi \qquad s \in \mathsf{State}}{\mathsf{return}(V) \models (s \rightarrowtail s)(\phi)}$$

$$\frac{V \models \phi}{\mathsf{return}(V) \models \langle\varepsilon\rangle\downarrow(\phi)} \qquad \frac{}{\mathsf{return}(V) \models \langle\varepsilon\rangle_{\dots}(\phi)}$$

$$\frac{V \models \phi}{\mathsf{return}(V) \models \mathsf{C}_{\leq q}(\phi)} \qquad \frac{V \models \phi}{\mathsf{return}(V) \models \mathsf{C}_{\geq 0}(\phi)}$$

These rules are complete for $\mathsf{return}(-)$, so $\mathsf{return}(V)$ will never satisfy $\mathsf{E}_e(\phi)$, $\mathsf{C}^{\uparrow}_{>q}(\phi)$, or $(s \rightarrowtail s')(\phi)$ if $s \neq s'$.

Possibly more interesting are the rules for the effect operators:

$$\frac{}{\mathsf{raise}_e() \models \mathsf{E}_e(\phi)}$$

$$\frac{\underline{M} \models \Diamond(\phi)}{\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)} \qquad \frac{\underline{N} \models \Diamond(\phi)}{\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)} \qquad \frac{\underline{M} \models \Box(\phi) \qquad \underline{N} \models \Box(\phi)}{\mathsf{or}(\underline{M}, \underline{N}) \models \Box(\phi)}$$

$$\frac{\underline{M} \models \mathsf{P}_{>a}(\phi) \qquad \underline{N} \models \mathsf{P}_{>b}(\phi) \qquad p \leq (a+b)/2}{\mathsf{pr}(\underline{M}, \underline{N}) \models \mathsf{P}_{>p}(\phi)}$$

$$\frac{\lambda x\colon \mathbf{N}.\, \underline{M} \models (\{s(l)\} \mapsto (s \rightarrowtail s')(\phi))}{\mathsf{lookup}_l(x \mapsto \underline{M}) \models (s \rightarrowtail s')(\phi)} \qquad \frac{V \models \{n\} \qquad \underline{M} \models (s[l := n] \rightarrowtail s')(\phi)}{\mathsf{update}_l(V; \underline{M}) \models (s \rightarrowtail s')(\phi)}$$

$$\frac{\lambda x\colon \mathbf{N}.\, \underline{M} \models (\{n\} \mapsto \langle w\rangle\downarrow(\phi))}{\mathsf{read}(x \mapsto \underline{M}) \models \langle (n?)w\rangle\downarrow(\phi)} \qquad \frac{V \models \{n\} \qquad \underline{M} \models \langle w\rangle\downarrow(\phi)}{\mathsf{write}(V; \underline{M}) \models \langle (n!)w\rangle\downarrow(\phi)}$$

$$\frac{\lambda x\colon \mathbf{N}.\, \underline{M} \models (\{n\} \mapsto \langle w\rangle_{\dots}(\phi))}{\mathsf{read}(x \mapsto \underline{M}) \models \langle (n?)w\rangle_{\dots}(\phi)} \qquad \frac{V \models \{n\} \qquad \underline{M} \models \langle w\rangle_{\dots}(\phi)}{\mathsf{write}(V; \underline{M}) \models \langle (n!)w\rangle_{\dots}(\phi)} \qquad \frac{}{\underline{M} \models \langle\varepsilon\rangle_{\dots}(\phi)}$$

$$\frac{\underline{M} \models \mathsf{C}_{\leq q}(\phi) \qquad p \geq q + c}{\mathsf{tick}_c(\underline{M}) \models \mathsf{C}_{\leq p}(\phi)} \qquad \frac{\underline{M} \models \mathsf{C}_{\geq q}(\phi) \qquad p \leq q + c}{\mathsf{tick}_c(\underline{M}) \models \mathsf{C}_{\geq p}(\phi)}$$

$$\frac{\underline{M} \models \mathsf{C}^{\uparrow}_{>q}(\phi) \qquad p \leq q + c}{\mathsf{tick}_c(\underline{M}) \models \mathsf{C}^{\uparrow}_{>p}(\phi)} \qquad \frac{q < c}{\mathsf{tick}_c(\underline{M}) \models \mathsf{C}^{\uparrow}_{>c}(\phi)}$$

The above rules are again complete for the effect operations. So for instance, $\mathsf{raise}_e()$ will never satisfy a formula of the form $\downarrow(\phi)$, nor will $\mathsf{write}(V; \underline{M})$ satisfy a formula of the form $\langle\varepsilon\rangle\downarrow(\phi)$. Moreover, if $\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)$, then either $\underline{M} \models \Diamond(\phi)$ or $\underline{M} \models \Diamond(\phi)$.

Most of the above proof rules can be derived from the proof rules for sequencing. As an example, consider the proof rules for $\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)$. Since both

$\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)$ and $\mathsf{op}(\mathsf{return}(\overline{0}), \mathsf{return}(\overline{1}))$ to $z.\,(\mathsf{case}\ z\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\})$ generate the same effect tree $\mathsf{or}(|\underline{M}|, |\underline{N}|)$, we know by Lemma 3.4.1 that $(\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)) \Leftrightarrow (\mathsf{op}(\mathsf{return}(\overline{0}), \mathsf{return}(\overline{1}))$ to $z.\,(\mathsf{case}\ z\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\}) \models \Diamond(\phi))$. So we can instead apply the sequencing proof rule for $\Diamond$ on the second statement to get:

$$\frac{\mathsf{op}(\mathsf{return}(\overline{0}), \mathsf{return}(\overline{1})) \models \Diamond(\psi) \qquad \lambda x : \mathbf{A}.\,\mathsf{case}\ z\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} \models (\psi \mapsto \Diamond(\phi))}{\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)}$$

Since $\mathsf{op}(\mathsf{return}(\overline{0}), \mathsf{return}(\overline{1})) \models \Diamond(\psi)$ holds if and only if $\overline{0} \models \psi$ or $\overline{1} \models \psi$, we can substitute $\{0\}$ and $\{1\}$ for ψ and remove the first clause. Note that $\underline{K} \models (\{n\} \mapsto \underline{\psi})$ holds precisely when $\underline{K}\ \overline{n} \models \underline{\psi}$. So we can do the following derivations using the operational semantics on the case constructor:

- If $\underline{M} \models \Diamond(\phi)$ then $\lambda x : \mathbf{A}.\,\mathsf{case}\ z\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} \models (\{0\} \mapsto \Diamond(\phi))$, and hence by the above proof rule with $\psi := \{0\}$, it holds that $\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)$.
- If $\underline{N} \models \Diamond(\phi)$ then since $\underline{N}$ does not mention x it holds that $\lambda x : \mathbf{A}.\,\mathsf{case}\ z\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} \models (\{1\} \mapsto \Diamond(\phi))$ and hence with $\psi := \{0\}$ we can derive $\mathsf{or}(\underline{M}, \underline{N}) \models \Diamond(\phi)$.

So we have derived the proof rules for $\mathsf{op}(\underline{M}, \underline{N}) \models \Diamond(\phi)$ from the sequencing proof rule.

This finishes our study of Boolean logics for algebraic effects and their combinations. Next chapter, we study a generalisation of the Boolean logic to a quantitative logic.

6

Quantitative logic

In this chapter, we will focus on generalising our logic to include quantitative behavioural properties. Here we do not express truth with the two element set of the Booleans $\mathbb{B}$, but with a generalised truth space $\mathbb{A}$.

The necessity of such a generalisation arises when combining certain effects, as well as when studying particular examples of effects which are not entirely 'algebraic' in nature. For instance, when one combines nondeterminism with probability in the Boolean logic, the obvious modalities one would define do not form a decomposable set. Effects which allow program execution to jump back to earlier places in the computation, are equally problematic.

In this chapter we will give examples of quantitative logics for effects, including but not limited to: probability, global store, cost, and combinations with different versions of nondeterminism and error.

6.1 Quantitative predicates

We desire a generalised truth space $\mathbb{A}$, which should have a preorder relation $\trianglelefteq$. The preorder gives us a generalised notion of implication, where $a \trianglelefteq b$ if b is at least as true as a. One should keep the standard example of the Boolean space $\mathbb{B}$ with implication $\Rightarrow$ in mind to see how the quantitative logic generalises the definitions and results from earlier chapters.

The space $\mathbb{A}$ should be a countably complete lattice, meaning for each countable subset $X \subseteq \mathbb{A}$ there should be a *least upper bound* (supremum) $\bigvee X$ and a *greatest lower bound* (infimum) $\bigwedge X$. In particular, we have $\boldsymbol{T} := \bigvee \mathbb{A} = \bigwedge \emptyset$ to denote absolute truth and $\boldsymbol{F} := \bigwedge \mathbb{A} = \bigvee \emptyset$ for absolute falsehood. The examples of truth spaces given in this chapter will all be complete lattices, though for the results in this chapter it is enough to only have countable suprema and infima. This is mainly because there are only countably many terms.

Lastly, we desire a notion of *negation*, also called an *involution* for the complete lattice. It is given by a unary map $\neg : \mathbb{A} \to \mathbb{A}$ such that $\forall a, b \in \mathbb{A}, a \trianglelefteq b \iff \neg b \trianglelefteq \neg a$ and $\forall a, \neg\neg a = a$. This notion of negation is however not necessary when defining and proving results about the positive fragment of the quantitative logic.

Examples of generalised truth spaces $\mathbb{A}$ are:

1. The Booleans $\mathbb{A} = \mathbb{B} = \{\boldsymbol{T}, \boldsymbol{F}\}$ with $\boldsymbol{F} \trianglelefteq \boldsymbol{T}$ and $\boldsymbol{T} \not\trianglelefteq \boldsymbol{F}$. Suprema $\bigvee$ and infima $\bigwedge$ are given by disjunction and conjunction as expected, and negation $\neg$ is the standard negation (swap function) on the Booleans.

2. The real number interval $[0, 1]$, which is used to describe probabilities as truth values. The order $\trianglelefteq$ is given by the standard 'less than or equal to' order on real numbers. Suprema $\bigvee$ and infima $\bigwedge$ are as expected for the real numbers, and negation $\neg$ is given by $\neg x = 1 - x$.

3. The positive reals with infinity $[0, \infty]$, which is used to describe expectations of truth. Order is the standard '$\leq$', and negation $\neg$ is given by $\neg x = 1/x$.

4. For any quantitative truth space $\mathbb{A}$, we can flip the order to create a new truth space $\mathbb{A}_{\geq}$. In particular, we can use $[0, \infty]_{\geq}$ to describe costs as quantitative truth values.

5. For any set A we can have a truth space $\mathbb{A} = \mathcal{P}(A)$ given by the powerset of that set. One can see A as a set of possibilities, and a value $a \in \mathbb{A}$ as denoting a precondition for truth. The order $\trianglelefteq$ is given by inclusion $\subseteq$, supremum by union, infimum by intersection and negation by complement $\neg a = \{x \in A \mid x \notin a\}$.

6. For any countably complete lattice $\mathbb{A}$ and set X, we can construct a countably complete lattice with involution $(X \to \mathbb{A})$, consisting of functions with the order and involution defined point-wise.

7. For any two countably complete lattices $\mathbb{A}$ and $\mathbb{X}$ with involutions, we can construct another countably complete lattice with involution, the space of *monotone* functions $(\mathbb{A} \to_{\trianglelefteq} \mathbb{X})$. Hence for $f \in (\mathbb{A} \to_{\trianglelefteq} \mathbb{X})$ and $a, b \in \mathbb{A}$, $\trianglelefteq_{\mathbb{A}} b \implies f(a) \trianglelefteq_{\mathbb{X}} f(b)$. The order is again taken point-wise. The involution is defined as $(\neg f)(a) = \neg(f(\neg a))$.

Using such generalised truth spaces, we will generalise the satisfaction relation of formulas to quantitative relations. Hence, a term will now satisfy a formula to a certain degree, where that degree is given by an element of the quantitative truth space $\mathbb{A}$. Hence the satisfaction relation $\models$ gives a map $\mathit{Terms}(\underline{\underline{\mathbf{E}}}) \times \mathit{Form}(\underline{\underline{\mathbf{E}}}) \to \mathbb{A}$. More generally, we consider quantitative predicates on X which, like formulas, now give maps $D : X \to \mathbb{A}$, and for $t \in T(X)$ we define $t[\in D] = D^*(t) = t[\langle x \rangle \mapsto \langle D(x) \rangle]$. We will give a definition of the quantitative logic in the next subsection.

6.1.1 The logic

In most aspects, the definition of the quantitative logic remains the same as the Boolean logic from Figure 3.1 at the end of Section 3.2, albeit with a few modifications. Conjunctions and disjunctions of formulas are replaced with infima and suprema respectively,

$$\frac{n \in \mathbb{N}}{\{n\} \in \mathit{Form}(\mathbf{N})} \qquad \frac{\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})}{\langle \underline{\phi} \rangle \in \mathit{Form}(\mathbf{U}\,\underline{\mathbf{C}})} \qquad \frac{\phi \in \mathit{Form}(\mathbf{A}_j)}{(j, \phi) \in \mathit{Form}(\mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i)}$$

$$\frac{\phi \in \mathit{Form}(\mathbf{A})}{\pi_0(\phi) \in \mathit{Form}(\mathbf{A} \times \mathbf{B})} \qquad \frac{\phi \in \mathit{Form}(\mathbf{B})}{\pi_1(\phi) \in \mathit{Form}(\mathbf{A} \times \mathbf{B})}$$

$$\frac{V \in \mathit{Terms}(\mathbf{A}) \qquad \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})}{(V \mapsto \underline{\phi}) \in \mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})} \qquad \frac{j \in I \qquad \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}_j)}{(j \mapsto \underline{\phi}) \in \mathit{Form}(\mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i)}$$

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

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

Figure 6.1: Quantitative formula constructors

and instead of negation we use involution. We also add two completely new features, *constant formulas* κ_a and *threshold formulas* $\underline{\underline{\phi}}_{\trianglerighteq a}$. A constant formula κ_a is always satisfied to the same degree a, by any term. A threshold formula $\underline{\underline{\phi}}_{\trianglerighteq a}$ checks whether satisfaction of $\underline{\underline{\phi}}$ is at least as high as a, and yields $\boldsymbol{T}$ when that is the case and $\boldsymbol{F}$ when it is not. So the constant formulas are satisfied to the same degree by all terms of its type. They already inherently existed in the Boolean logic, as $\bigvee \emptyset$ and $\bigwedge \emptyset$, but since there are now intermediate values, constant formulas are explicitly added. The threshold formula, or step construction, checks whether satisfaction of a formula passes a certain threshold. Both such operations are used frequently (albeit implicitly) in practical examples of quantitative verification, e.g., in [56].

To interpret the effects, we will make use of *quantitative modalities* $\mathcal{Q}$. A quantitative modality $q \in \mathcal{Q}$ lifts a quantitative predicate $P : X \to \mathbb{A}$ on any set X, to a quantitative predicate $q(P) : T(X) \to \mathbb{A}$ on the trees over that set. The quantitative predicate P can be used to change the leaves $x \in X$ of a tree $t \in T(X)$ to values $P(x) \in \mathbb{A}$. The quantitative modality then looks at the resulting tree of $T(\mathbb{A})$ and assigns to it a truth value of $\mathbb{A}$. As such, they are specified by a function $[\![q]\!] : T(\mathbb{A}) \to \mathbb{A}$, with which the lifted quantitative predicate can be explicitly given by $q(P)(t) = [\![q]\!](t[\langle x \rangle \mapsto \langle P(x) \rangle])$.

We give the precise formulation of the logic in Figure 6.1, which is very similar to the Boolean logic in Section 3.3, except for the addition of the threshold and constant formulas.

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

$$V \models \{n\} \quad := \quad \begin{cases} \boldsymbol{T} & \text{if } V = \overline{n} \\ \boldsymbol{F} & \text{otherwise.} \end{cases}$$

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

$$V \models \langle \underline{\phi} \rangle \quad := \quad \mathsf{force}(V) \models \underline{\phi}.$$

$$(V_0, V_1) \models \pi_0(\phi) \quad := \quad V_0 \models \phi. \qquad \underline{M} \models (V \mapsto \underline{\phi}) \quad := \quad \underline{M}\ V \models \underline{\phi}.$$

$$(V_0, V_1) \models \pi_1(\phi) \quad := \quad V_1 \models \phi. \qquad \underline{M} \models (j \mapsto \underline{\phi}) \quad := \quad \underline{M}\ j \models \underline{\phi}.$$

The basic quantitative formulas at $\mathbf{F}\,\mathbf{A}$ are particularly important, as they express how the quantitative modalities are used to observe effects:

$$\underline{M} \models q(\phi) \quad := \quad [\![q]\!](|\underline{M}|[\models \phi]) \quad := \quad [\![q]\!](|\underline{M}|[\langle V \rangle \mapsto \langle V \models \phi \rangle]) \ .$$

The satisfaction relation for non-basic formulas is defined as follows.

$$\underset{\sim}{P} \models \bigvee X \quad := \quad \bigvee \{ \underset{\sim}{P} \models \underset{\sim}{\phi} \mid \underset{\sim}{\phi} \in X \}.$$

$$\underset{\sim}{P} \models \bigwedge X \quad := \quad \bigwedge \{ \underset{\sim}{P} \models \underset{\sim}{\phi} \mid \underset{\sim}{\phi} \in X \}.$$

$$\underset{\sim}{P} \models \underset{\sim}{\phi}_{\rhd a} \quad := \quad \begin{cases} \boldsymbol{T} & \text{if } \exists b \in \mathbb{A},\ b = (\underset{\sim}{P} \models \underset{\sim}{\phi}) \wedge a \trianglelefteq b, \\ \boldsymbol{F} & \text{otherwise.} \end{cases}$$

$$\underset{\sim}{P} \models \kappa_a \quad := \quad a.$$

$$\underset{\sim}{P} \models \neg(\underset{\sim}{\phi}) \quad := \quad \neg(\underset{\sim}{P} \models \underset{\sim}{\phi}).$$

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

6.2 Examples

We will look at some examples of effects for which there is a natural description of behaviour using quantitative logics. Some of these examples coincide with examples for the Boolean logic, in which case the resulting behavioural equivalence will be the same. One of the main benefits of the quantitative logics is that it will be easy to build on each of the quantitative descriptions of effects to combine those with the effect of nondeterminism. This is in contrast with the Boolean logic, for which it was shown in Subsection 3.5.4 that it could not properly describe the combinations of probability and global store with nondeterminism. Moreover, other combinations of effects, like probability and global store, are also more easily described compared to for instance Subsection 5.3.4.

In the following subsections, we look at examples of effects and effect combinations which can be described by the quantitative logic. Such (combinations of) effects are represented by an effect signature Σ of effect operations, for which we need to choose a relevant truth space $\mathbb{A}$ and one or more relevant quantitative modalities. In order to define the modalities, we shall always proceed in the following steps. The modalities of the examples are constructed inductively. We define $[\![q]\!]_{(-)}(-) : \mathbb{N} \times T(\mathbb{A}) \to \mathbb{A}$ and take the actual denotation to be $[\![q]\!](t) := \bigvee_{n\in\mathbb{N}}([\![q]\!]_n(t))$. There are three standard cases in the definition:

1. $[\![q]\!]_0(t) := \boldsymbol{F}$.
2. $[\![q]\!]_{n+1}(\bot) := \boldsymbol{F}$.
3. $[\![q]\!]_{n+1}(\eta(a)) := a$.

The remaining cases concerning the nodes of effect operations, are specific to the effects in question. By defining the modalities in such a way, it will be easy to prove that they satisfy the correct properties (of continuity and decomposability) from Subsection 6.3.1 to prove that the resulting behavioural equivalence is a congruence.

6.2.1 Probability

As in Subsection 2.3.4, we have the singleton signature $\Sigma_{\mathrm{pr}} := \{\mathsf{pr}(-,-) : \alpha^2 \to \alpha\}$.

The complete lattice of truth values is the space of real-valued probabilities $\mathbb{A} := [0,1]$, with the standard ordering $\trianglelefteq := \leq$, so $\bigvee = \sup$ and $\bigwedge = \inf$. The minimal element is 0 and maximal element is 1.

We have a single quantitative modality E where

$$[\![\mathsf{E}]\!]_{n+1}(\mathsf{pr}(t,r)) := ([\![\mathsf{E}]\!]_n(t) + [\![\mathsf{E}]\!]_n(r))/2 \ ,$$

the average of two continuations.

Given a computation $\underline{M} : \mathbf{F}\,\mathbf{A}$, and a formula $\phi \in \mathit{Form}(\mathbf{A})$, the element $\underline{M} \models \mathsf{E}(\phi) \in [0,1]$ gives the expectation of $V \models \phi$ when V is sampled/obtained from $\underline{M}$. In the special case that ϕ is Boolean, meaning it gives only $\boldsymbol{T}$ or $\boldsymbol{F}$, $\underline{M} \models \mathsf{E}(\phi)$ can be seen as the probability that ϕ is satisfied by whatever $\underline{M}$ returns. Note that the original modality formulas $\mathsf{P}_{>q}\psi$ for the Boolean description of probability can be defined as $\bigvee\{\mathsf{E}(\psi)_{\trianglerighteq p} \mid p \in \mathbb{Q} \cap [0,1], p > q\}$, given a suitable translation for ψ.

6.2.2 Probability with score

Alternatively to interpreting probability with a truth space of probabilities $[0,1]$, we could interpret it with a truth space of expectations $\mathbb{A} := [0,\infty]$, the non-negative real numbers with infinity and $\leq$ order. This interpretation also lends well to the inclusion of score operators, which can assign value multipliers to certain evaluation branches. These give a way of reweighting the distribution of results by a non-negative factor, as done in [103].

We take as signature the set $\Sigma := \Sigma_{\text{pr}} \cup \{\mathsf{score}_q(-) : \alpha \to \alpha \mid q \in \mathbb{Q}_{>0}\}$. We again have a single quantitative quantitative modality ES where

$$[\![\mathsf{ES}]\!]_{n+1}(\mathsf{pr}(t,r)) := ([\![\mathsf{ES}]\!]_n(t) + [\![\mathsf{ES}]\!]_n(r))/2 \ ,$$

$$[\![\mathsf{ES}]\!]_{n+1}(\mathsf{score}_q(t)) := q \cdot [\![\mathsf{ES}]\!]_n(t) \ .$$

Now for a bit of a tangent. In the case of probability (optionally with score) interpreted using truth space $[0,\infty]$, we may replace the threshold operator with products of formulas $\underline{\phi} \cdot \underline{\psi}$ with the interpretation $\underline{P} \models \underline{\phi} \cdot \underline{\psi} := (\underline{P} \models \underline{\phi}) \cdot (\underline{P} \models \underline{\psi})$. Taking $\underline{\phi}^n$ to be the n-th product of $\underline{\phi}$, the threshold formula $\underline{\phi}_{\trianglerighteq a}$ is equivalent to

$$\bigwedge \{\kappa_a \cdot (\kappa_1 \wedge \bigwedge_{n \in \mathbb{N}} (\underline{\phi} \cdot \kappa_q)^n) \mid q \in \mathbb{Q}_{>0}, 1/q \leq a\} \ .$$

So the inclusion of the threshold operator can be seen as a consequence of the fact that a computation could be sampled any number of times.

6.2.3 Global Store

As in Subsection 2.3.5, we have the signature consisting of two operators per store location: $\Sigma_{\text{gs}} := \{\mathsf{lookup}_l(-) : \alpha^{\mathbb{N}} \to \alpha, \mathsf{update}_l(-;-) : \mathbb{N} \times \alpha \to \alpha \ \mid \ l \in \mathsf{Loc}\}$.

The countably complete lattice of truth values is $\mathbb{A} := \mathcal{P}(\mathsf{State})$, where State is the set of states $(\mathsf{Loc} \to \mathbb{N})$. The order $\trianglelefteq$ is given by inclusion $\subseteq$, where $\bigvee := \bigcup$ and $\bigwedge := \bigcap$. So the minimal element is $\emptyset$ and maximal element is State.

We have a single quantitative modality G where

$$[\![\mathsf{G}]\!]_{n+1}(\mathsf{lookup}_l(t_0, t_1, \dots)) := \{s \in \mathsf{State} \mid s \in \mathsf{State} \in [\![\mathsf{G}]\!]_{\max(0,n-s(l))}(t_{s(l)})\} \ ,$$

$$[\![\mathsf{G}]\!]_{n+1}(\mathsf{update}_l(m;t)) := \{s \mid s[l := m] \in [\![G]\!]_n(t)\} \ .$$

Given a computation $\underline{M} : \mathbf{F}\,\mathbf{A}$, and a formula $\phi \in \mathit{Form}(\mathbf{A})$, the element $\underline{M} \models \mathsf{G}(\phi) \subseteq \mathsf{State}$ is the set of beginning states for which the evaluation of $\underline{M}$ terminates, with some value V and end state s such that $s \in (V \models \phi)$. Hence we can see $\underline{M} \models \mathsf{G}(\phi)$ as the weakest precondition on global states for which, when $\underline{M}$ is run, ϕ is satisfied.

6.2.4 Probability and global store

As an example of how easy it is to combine effects using a quantitative logic, compared to the Boolean logic as done in 5.3.4, we turn towards the combination of probability with global store. This combination has as effect signature the disjoint union $\Sigma := \Sigma_{\text{pr}} \cup \Sigma_{\text{gs}}$. For this combination of effects, we take as truth space the functions $\mathbb{A} := [0,1]^{\mathsf{State}}$ with point-wise order, where State is the set of global states $\mathbb{N}^{\mathsf{Loc}}$ and $[0,1]$ the lattice of probabilities with standard order. Intuitively, this space assigns to each starting state a probability that a property is satisfied. We define a single modality EG which, for each state $s \in \mathsf{State}$, is given by the following rules:

$$[\![\mathsf{EG}]\!]_{n+1}(\mathsf{por}(t,r))(s) := ([\![\mathsf{EG}]\!]_n(t)(s) + [\![\mathsf{EG}]\!]_n(r)(s))/2 \ ,$$

$$[\![\mathsf{EG}]\!]_{n+1}(\mathsf{lookup}_l(t_0, t_1, \dots))(s) = [\![\mathsf{EG}]\!]_{\max(0,n-s(l))}(t_{s(l)})(s) \ ,$$
$$[\![\mathsf{EG}]\!]_{n+1}(\mathsf{update}_{l,m}(t))(s) = [\![\mathsf{EG}]\!]_n(t)(s[l := m]) \ .$$

The statement $\underset{\sim}{P} \models \mathsf{EG}(\phi)$ gives the function sending each state $s \in \mathsf{State}$ to the expected result of the following computation: Run $\underset{\sim}{P}$ with starting state s. If the evaluation diverges, the result will be zero. If the evaluation terminates with ending state s' and returns a term V, then the result will be $(V \models \phi)(s')$.

This quantitative logic for this combination of effects induces a behavioural preorder satisfying the equational theory of the combination of probability and global store with all distributivity laws, given by the tensor of effects in [33].

6.2.5 Timer

As in Subsection 2.3.7, given a countable set of real-valued units of time increments Inc, we define the signature $\Sigma_{\mathrm{ti}} := \{\mathsf{tick}_c(-) : \alpha \to \alpha \mid c \in \mathsf{Inc}\}$.

The up-interpretation for the effect uses as generalised truth space the non-negative real numbers with infinity $\mathbb{A} := [0, \infty]$, with the standard ordering of 'smaller or equal than'. This space forms a complete lattice, where in particular the supremum of a diverging sequence of reals is given by the maximal element $\boldsymbol{T} := \infty$. The element 0 is the minimum element $\boldsymbol{F}$.

There is but one quantitative modality $\mathsf{C}^{\uparrow}$, defined with the rule:

$$[\![\mathsf{C}^{\uparrow}]\!]_{n+1}(\mathsf{tick}_c(t)) := c + [\![\mathsf{C}^{\uparrow}]\!]_n(t) \ .$$

Addition is defined as normal, where in particular $c + \infty = \infty$.

If $\underline{M}$ terminates, then the statement $\underline{M} \models \mathsf{C}^{\uparrow}(\phi)$ gives the total time of termination of $\underline{M}$ plus the time given by $V \models \phi$ where V is the term produced by $\underline{M}$. Note however, that this modality detects 'ticks' even if the computation eventually diverges, since $[\![\mathsf{C}^{\uparrow}]\!](\mathsf{tick}_c(\bot)) = (c + 0) = c \neq \boldsymbol{F}$. As an example of a formula which can be constructed in this logic, consider the formula: $\underline{\phi} := \bigwedge_{n \in \mathbb{N}} (\mathsf{C}^{\uparrow}(\kappa_n))_{\geq c+n}$, which yields $\boldsymbol{T}$ if a computation terminates after delaying the computation for at least $c \in \mathbb{A}$ time.

For the down-interpretation, we use $\mathbb{A} := [0, \infty]_{\geq}$, with the reversed order $\geq$: so $a \trianglelefteq b$ if a is, as a real number, larger or equal than b. We have $\bigvee := \inf$ and $\bigwedge := \sup$, so the minimum element is ∞ and the maximum element is 0.

There is still one quantitative modality $\mathsf{C}^{\downarrow}$, defined with the same rule:

$$[\![\mathsf{C}^{\downarrow}]\!]_{n+1}(\mathsf{tick}_c(t)) := c + [\![\mathsf{C}^{\downarrow}]\!]_n(t)$$

If $\underline{M}$ terminates, then the statement $\underline{M} \models \mathsf{C}^{\downarrow}(\phi)$ has the same interpretation as in the up-interpretation. However, since $c + \infty = \infty$, this modality cannot detect 'ticks' if the computation eventually diverges. E.g. $[\![\mathsf{C}^{\downarrow}]\!](\mathsf{tick}_c(\bot)) = (c + \infty) = \boldsymbol{F}$.

It is possible to assign a quantitative interpretation of the general (combined) interpretation of the timer effect given in Subsection 3.2.7, though we will not include it here. If the weights of all ticks are given by natural numbers, we could alternatively use as quantitative truth space $\mathbb{A} := \mathbb{N} \cup \{\infty\}$. Though this space does not have an involution.

6.2.6 Combination with nondeterminism

We discuss nondeterminism in the sense of Subsection 2.3.3, and show how we can add this effect to any of the examples given above in a uniform way. As was said before, the definitions of the quantitative modalities as constructed in the previous examples are perfectly suited for combinations with nondeterminism. The concepts of supremum and infimum of the coutanbly complete lattice aligns itself neatly with angelic and demonic nondeterminism. In this part, we will present the combination of nondeterminism with other effects.

In the cases of global store and timer, this combination with nondeterminism is in line with the tensor combination of effects given in [33]. More precisely, in such cases, the operations of global store and timer freely distribute over the nondeterministic choice operation, and vice versa. However, in the case of probability with nondeterminism, the combination is neither the tensor nor the sum from [33], since the combination has one distributivity law, but not all. It is like the natural combination of nondeterminism and probability, specified in a similar way as in [49].

We take a signature of effects Σ, a countably complete lattice of truth value $\mathbb{A}$ and a set of quantitative modalities $\mathcal{Q}$ as in any of the previous examples. We extend the signature to include binary nondeterminism $\Sigma' := \Sigma \cup \{\mathsf{or}(-,-) : \alpha^2 \to \alpha\}$. The only thing we need to do is extend the inductive definition of $[\![\mathsf{q}]\!]_{(-)}$ of each $\mathsf{q} \in \mathcal{Q}$, to include its treatment of the nondeterministic choice operator. There are two ways to deal with this, either by taking the optimistic (angelic) approach or by taking the pessimistic (demonic) approach. The two methods result in the new quantitative modalities $\mathsf{q}_\Diamond$ and $\mathsf{q}_\Box$ respectively, using the following definitions:

$$[\![\mathsf{q}_\Diamond]\!]_{n+1}(\mathsf{or}(t,r)) := [\![\mathsf{q}_\Diamond]\!]_n(t) \vee [\![\mathsf{q}_\Diamond]\!]_n(r) \ ,$$

$$[\![\mathsf{q}_\Box]\!]_{n+1}(\mathsf{or}(t,r)) := [\![\mathsf{q}_\Box]\!]_n(t) \wedge [\![\mathsf{q}_\Box]\!]_n(r) \ ,$$

where $\vee$ and $\wedge$ are defined according to the supremum and infimum from $\mathbb{A}$.

The choice of the modalities depends on which type of nondeterminism one wants to consider, either $\mathsf{q}_\Diamond$ for angelic or $\mathsf{q}_\Box$ for demonic. If one wants to consider neutral nondeterminism, the setof modalities should contain both these quantitative modalities.

Take for instance the example of probability and nondeterminism. In this case, we can see the nondetermistic choice as being resolved by a scheduler, whereas the probabilistic choice is resolved by a coin toss. The scheduler will however not know the results of coin tosses in the future[1]. See for instance [49] for various different descriptions for this combination of effects, including the scheduler interpretation. The point of nondeterminism is that we do not know how the scheduler behaves, however we can do a best case and a worst case analysis. We can see $\underline{M} \models \mathsf{E}_\Diamond(\phi)$ as the expectation of satisfaction of ϕ, given that the nondeterministic choice is controlled by some scheduler optimizing (either by intention or by accident) the expectation of satisfaction of ϕ. Similarly, $\underline{M} \models \mathsf{E}_\Box(\phi)$ gives the expectation assuming the agent or

[1] This is reflected in the equational theory by nondeterministic choice not distributing over probabilistic choice.

scheduler controlling the choice is minimizing the expectation of satisfaction. In the case where we do not know what the scheduler will choose, which is the case of neutral nondeterminism, we cannot necessarily do a concrete prediction of the expectation. However, if the nondeterministic choice is resolved by some scheduler, we do know that the expectation of satisfaction lies somewhere between $\underline{M} \models \mathsf{E}_{\Diamond}(\phi)$ and $\underline{M} \models \mathsf{E}_{\Box}(\phi)$.

6.2.7 Combinations with Errors

We may also extend the language with error messages in the sense of Subsection 2.3.2, though there are two different methods which possibly give different interpretations of effectful behaviour. Take Σ, $\mathbb{A}$ and $\mathcal{Q}$, and let Err be a set of error messages, then we can extend the signature (with disjoint union) to $\Sigma' := \Sigma \cup \{\mathsf{raise}_e() : \mathbf{1} \to \alpha \mid e \in \mathsf{Err}\}$.

There are two methods for combining effects with Error. We will start with the most general one, whose result induces an equational theory which is in most cases (e.g., global store and timer) in line with the 'sum' of equational theories from [33]. For each function $D : \mathsf{Err} \to \mathbb{A}$ and modality $\mathsf{q} \in \mathcal{Q}$, assigning a choice of observation value to each error message, we construct q_D using the same rules as for q but treating the new effect operators as follows:

$$[\![\mathsf{q}_D]\!]_{n+1}(\mathsf{raise}_e) := D(e) \ .$$

We get a new set of modalities $\mathcal{Q}^+ := \{\mathsf{q}_D \mid q \in \mathcal{Q}, D : \mathsf{Err} \to \mathbb{A}\}$. This new set is rather large in size, which is in most cases actually unnecessary. We could limit the functions D used to only those whose image is $\{\boldsymbol{T}, \boldsymbol{F}\}$, giving the alternative set of modalities $\mathcal{Q}^{\otimes} := \{\mathsf{q}_D \mid q \in \mathcal{Q}, D : \mathsf{Err} \to \{\boldsymbol{T}, \boldsymbol{F}\}\}$. The equational theory of the result of this method is in most cases (e.g., global store and timer) in line with the 'tensor' of equational theories from [33]. When considering probability or nondeterminism with errors, the two methods yield the same induced behavioural equivalence.

For global store however, $\mathcal{Q}^{\otimes}$ gives a different interpretation then $\mathcal{Q}^+$. This results in the following phenomenon: 'When an error message is reached, the state of the global store cannot be observed'. This is a perfectly valid alternative interpretation, and coincides with the situation where either (1) the state cannot be retrieved after an error has been raised, or (2) the state resets after an error is raised. Concretely, for any modality $(s \rightarrowtail s')_D \in \mathcal{Q}^{\otimes}$,

$$[\![(s \rightarrowtail s')_D]\!](\mathsf{update}_l(\overline{n}; \mathsf{raise}_e())) \quad = \quad [\![(s \rightarrowtail s')_D]\!](\mathsf{raise}_e()).$$

However, there is a modality in $\mathcal{Q}^+$ which distinguishes the two trees.

6.2.8 Input/Output

For input/output, as in Subsection 2.3.6, we have the signature consisting of two operators: $\Sigma_{\mathrm{io}} := \{\mathsf{read}(-) : \alpha^{\mathbb{N}} \to \alpha, \mathsf{write}(-;-) : \mathbb{N} \times \alpha \to \alpha\}$.

Like in the Boolean logic, we classify a set of *io-traces* W' recursively as follows:

$$w ::= \quad \varepsilon \quad | \quad (!m)w \quad | \quad (?m)w$$

where $m \in \mathbb{N}$. Let W be the set $\{w_t, w_o \mid w \in W'\}$ which we equip with the order $\leq$ defined by the following two rules:

1. For any $w \in W'$, $w_o \leq w_t$.
2. For $w \in W'$ and $m \in \mathbb{N}$, then $w(!m)_o \leq w_o$ and $w(?m)_o \leq w_o$.

The countably complete lattice of truth values is given by $\mathbb{A} := \mathcal{P}^{\downarrow}_{\neq\emptyset}(W)$, the set of non-empty down-closed subsets of W. So for $a \subseteq W$,

$$a \in \mathbb{A} \quad \iff \quad a \neq \emptyset \ \wedge \ \forall v_i, w_i \in W.\ (w_i \in a \wedge v_i \leq w_i) \Rightarrow v_i \in a.$$

A truth value gives a set of execution traces which can be produced by a computation. Here, w_t denotes an execution trace followed by termination, whereas w_o denotes an execution trace without the necessity of termination, an open-ended trace. In particular, if an execution trace w_t or w_o can be produced, then any smaller execution trace v_o can be produced, which motivates us to only use down-closed sets as truth values. Moreover, the execution trace ε_o will always be produced, so we moreover only use non-empty sets of execution traces as truth values. Note too that ε_o is the smallest execution trace, so it is included in any truth value.

The order $\trianglelefteq$ of $\mathbb{A}$ is given by the inclusion ordering, with $\bigvee$ and $\bigwedge$ the union and intersection respectively. This space does not have an involution.

We define a quantitative modality IO, checking which traces can be produced by a computation. We define it with the following rules:

$$[\![\mathsf{IO}]\!]_{n+1}(\mathsf{write}(m;t)) := \{((!m)w)_i \mid i \in \{o,t\}, w_i \in [\![\mathsf{IO}]\!]_n(t)\}$$

$$[\![\mathsf{IO}]\!]_{n+1}(\mathsf{read}(t_0,t_1,\dots)) := \{((?m)w)_i \mid i \in \{o,t\}, m \in \mathbb{N}, w_i \in [\![\mathsf{IO}]\!]_{\max(0,n-m)}(t_m)\}.$$

Specifically, the truth value given by $\underline{M} \models \mathsf{IO}(\phi)$ returns the set of execution traces containing (1) traces w_o produced by $\underline{M}$, and (2) traces $(wv)_i$ where $\underline{M}$ produces trace w_t and terminates with a value V such that $v_i \in (V \models \phi)$.

Though this quantitative formulation of input/output combines well with angelic nondeterminism, the same cannot be said about the combination with demonic nondeterminism. The combined modality $\mathsf{IO}_\Box$ will only give execution traces that are guaranteed to occur. As such, the modality does not distinguish between $\mathsf{or}(\mathsf{write}(0;x),\mathsf{write}(1;y))$ and $\bot$. This problem may be solved by defining a larger albeit more complicated 'continuation style' truth space. Such a definition is not as intuitive as the one given above, and as such we will not give it here.

6.2.9 Jumps

For our last example, we represent the effect of jumping to earlier places in the computation as an algebraic effect. This is a simplified version of the jump effect from [20], where here we do not use dynamically created variables to carry additional information with us while jumping. This limited version of jumping acts like a form of exception

catching. As we wil show below, if one tries to model this effect with the Boolean logic, the equational theory trivializes.

The effect resembles exception catching; given a set of jumps Jum which, if reached, will make the computation jump back to an appropriate 'catch' earlier in the computation. The signature is comprised of:

$$\Sigma_{\mathrm{ju}} \quad := \quad \{\mathsf{jump}_j() : \mathbf{1} \to \alpha,\ \mathsf{catch}_j(-,-) : \alpha \times \alpha \to \alpha \ \mid\ j \in \mathsf{Jum}\} \ .$$

The computation $\mathsf{catch}_j(\underline{M}, \underline{N})$ will compute $\underline{M}$ and return its result, unless it reduces to a jump $\mathsf{jump}_j()$, in which case it continues evaluation with $\underline{M}$. The computation $\mathsf{jump}_k()$ will jump back to an earlier point in the computation, the most recent catch $\mathsf{catch}_k(\underline{M}, \underline{N})$, and will then commence computing $\underline{N}$.

The quantitative logic is defined over a space of tokens $\mathbb{A} := (\mathsf{Jum} \cup \{\boldsymbol{T}, \boldsymbol{F}\})$, containing jump tokens $j \in \mathsf{Jum}$ and maximal and minimal elements $\boldsymbol{T}$ and $\boldsymbol{F}$. So for all $i, j \in \mathsf{Jum}$, $\boldsymbol{F} \trianglelefteq i \trianglelefteq \boldsymbol{T}$ and $(i \trianglelefteq j) \Leftrightarrow (i = j)$. A formula in this logic gives back a token, designating whether the computation terminates ($\boldsymbol{T}$), diverges ($\boldsymbol{F}$), or whether we have jumped out of the computation ($j \in \mathsf{Jum}$). Note that this is an example of a non-distributive complete lattice if Jum has more than two elements. Involution can be defined by taking $\forall j \in \mathsf{Jum}.\neg j = j$, and the usual swapping of $\boldsymbol{T}$ and $\boldsymbol{F}$.

We define only one quantitative modality J, which we recursively define as follows:

$$[\![\mathsf{J}]\!]_{n+1}(\mathsf{jump}_j()) := j \ ,$$

$$[\![\mathsf{J}]\!]_{n+1}(\mathsf{catch}_j(t, r)) := \begin{cases} [\![\mathsf{J}]\!]_n(r) & \text{if } [\![\mathsf{J}]\!]_n(t) = j \\ [\![\mathsf{J}]\!]_n(t) & \text{otherwise} \end{cases} .$$

Note that if $[\![\mathsf{J}]\!](t) \trianglelefteq [\![\mathsf{J}]\!](t')$, then $[\![\mathsf{J}]\!](\mathsf{catch}_j(t, r)) \trianglelefteq [\![\mathsf{J}]\!](\mathsf{catch}_j(t', r))$, which is necessary for proving that the modality satisfies the congruence properties from Subsection 6.3.1, yet unlike the previous examples is not immediately obvious from the formulation.

This definition of a quantitative logic for jumps is not directly amendable to a combination with nondeterminism, at least not without having $\mathsf{or}(\mathsf{jump}_i(), \mathsf{jump}_j()) \equiv \bot$ when $i \neq j$ in the case of demonic nondeterminism. It is possible to avoid this problem by complicating the definition of the truth space and the modalities.

Equational theory of jump: To give more intuition on the behaviour of this effect, we give the appropriate equations that will be satisfied by the behavioural equivalence, whose definition is forthcoming.

$$\begin{aligned} \mathsf{catch}_j(\mathsf{jump}_j(), x) &= x = \mathsf{catch}_j(x, \mathsf{jump}_j()) \\ \mathsf{catch}_j(\mathsf{jump}_k(), x) &= \mathsf{jump}_k() \quad \text{if } j \neq k \\ \mathsf{catch}_j(\mathsf{catch}_j(x, y), z) &= \mathsf{catch}_j(x, \mathsf{catch}_j(y, z)) \\ \mathsf{catch}_j(\bot, x) &= \bot \\ \mathsf{catch}_j(x, x) &= x \ . \end{aligned}$$

If we attempt to model the above equations with Boolean modalities as in Chapter 3, we get into trouble. From Section 3.5 we know that if we have a decomposable set of Boolean modalities, then according to Proposition 3.5.7, any equation holds if and only if it holds for any substitution of variables for $\{x, \bot\}$. However, if we assume that the above equations hold, then we can prove (with Proposition 3.5.7) that the following equation holds $\mathsf{catch}_j(x, \mathsf{catch}_j(y, z)) = \mathsf{catch}_j(x, \mathsf{catch}_j(z, y))$. Hence, with $x := \mathsf{jump}_j()$ and $y := \bot$, we can prove that the equation $z = \bot$ holds for any z. As such, the entire equational theory trivializes, which is not a particularly healthy consequence.

6.3 Behavioural preorders

In this section, we will define the behavioural preorders and equivalences, and the properties sufficient for proving that these are compatible. This generalises Section 3.3. We can define a behavioural preorder for any fragment of formulas $\mathcal{L}$ as follows.

Definition 6.3.1. For any subset of the quantitative logic $\mathcal{L} \subseteq \mathcal{U}$, the *logical preorder* $\underline{\sqsubseteq}_{\mathcal{L}}$ is defined by:

$$\forall \underline{P}, \underline{R} \in \mathit{Terms}(\underline{\mathbf{E}}), \quad \underline{P} \mathrel{\underline{\sqsubseteq}_{\mathcal{L}}} \underline{R} \quad \iff \quad \forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{E}})_{\mathcal{L}}, \ \ \underline{P} \models \underline{\phi} \ \trianglelefteq \ \underline{R} \models \underline{\phi} \ .$$

The *logical equivalence* $\equiv_{\mathcal{L}}$ is given by $\underline{\sqsubseteq}_{\mathcal{L}} \cap \underline{\sqsupseteq}_{\mathcal{L}}$.

The *general behavioural preorder* $\underline{\sqsubseteq}$ is the logical preorder $\underline{\sqsubseteq}_{\mathcal{U}}$, whereas the *positive behavioural preorder* $\underline{\sqsubseteq}^+$ is the logical preorder $\underline{\sqsubseteq}_{\mathcal{U}^+}$. We denote by $\equiv$ and $\equiv^+$ the logical equivalences over $\mathcal{U}$ and $\mathcal{U}^+$ respectively (the behavioural equivalences). These closed relations can be extended to relations on open terms by using the *open extension* from Definition 3.3.7.

Remark: The behavioural equivalences for probability, global store, timer, and input/output as defined above using the truth spaces and quantitative modalities from Section 6.2 coincide with the behavioural equivalences as induced by the Boolean logics from Chapter 3 using the corresponding modalities for those effects from Section 3.2.

A *basic* formula is a formula (not necessarily atomic) which on the top level does not have a supremum $\bigwedge$, an infimum $\bigvee$, an involution $\neg$, a constant formula κ_a or a threshold formula $(-)_{\trianglerighteq a}$. It is not difficult to see that both $\underline{\sqsubseteq}$ and $\underline{\sqsubseteq}^+$ are completely determined by basic formulas, similar to what is observed in Lemma 3.3.5. Also note that for any $(\mathcal{L}) \subseteq (\mathcal{U})$, we always have that $(\underline{\sqsubseteq}) \subseteq (\underline{\sqsubseteq}_{\mathcal{L}})$ simply because fewer properties are tested by $\mathcal{L}$ than by $\mathcal{U}$. We give some other general results.

Lemma 6.3.2. *The general behavioural preorder $\underline{\sqsubseteq}$ is symmetric, so $(\underline{\sqsubseteq}) = (\equiv)$.*

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

□

Henceforth, we will use $\equiv$ instead of $\underline{\sqsubseteq}$.

Lemma 6.3.3. *For any fragment $\mathcal{L}$ of the logic $\mathcal{U}$ closed under countable infima, it holds that for any term $\underset{\cdot\cdot}{P} : \underset{\cdot\cdot}{\mathbf{E}}$ there is a formulas $\chi_{\underset{\cdot\cdot}{P}}$ s.t.:*

$$(\underset{\cdot\cdot}{R} \models \chi_{\underset{\cdot\cdot}{P}}) = \begin{cases} \boldsymbol{T} & \text{if } \underset{\cdot\cdot}{P} \equiv_{\mathcal{L}} \underset{\cdot\cdot}{R} \\ \boldsymbol{F} & \text{otherwise.} \end{cases}$$

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

With a similar proof to Lemma 3.4.1, we have the following sufficient condition for behavioural equivalence:

Lemma 6.3.4. *If $|\underline{M}| = |\underline{N}|$ then $\underline{M} \equiv \underline{N}$.*

6.3.1 Congruence properties

We introduce the properties that we will use in order to establish that the (open extensions) of the behavioural preorders are compatible, hence precongruences. These are generalisations of the properties needed for the Boolean logic, established in Subsection 3.3.2.

We follow the structure of Subsection 3.3.2, generalising properties on Boolean modalities used for proving the Compatibility Theorem (Theorem 3.3.8), to properties on quantitative modalities. Firstly, we will generalise the notion of Scott opennes, and later on we generalise the notion of decomposability.

Scott continuity

There are three natural preorders on $T\mathbb{A}$. One is the *tree order* $\leq$ which, as introduced just after Definition 2.2.1, is the natural domain-order for any set of trees TX. Second is the *leaf-order* $\trianglelefteq_{T\mathbb{A}}$, which is the lifting of the order $\trianglelefteq$ on $\mathbb{A}$ to an order on $T\mathbb{A}$, using the mono preservation property of the endofunctor T. Concretely, $t \trianglelefteq_{T\mathbb{A}} r$ if r can be obtained from t by replacing some of the leaves $a \in \mathbb{A}$ of t by leaves $b \in \mathbb{A}$ of a higher degree $a \trianglelefteq b$.

It is natural to combine the two orders into a third general order $\leq_{T\mathbb{A}}$ where $t \leq_{T\mathbb{A}} t'$ if and only if there is a $t'' \in T\mathbb{A}$ such that $t \leq t''$ and $t'' \trianglelefteq_{T\mathbb{A}} t$. We omit the proof of the following lemma since it is straightforward.

Lemma 6.3.5. *For any two trees $t, t' \in T\mathbb{A}$, the following two statements are equivalent:*

1. $t \leq_{T\mathbb{A}} t'$.
2. $\exists t'' \in T(\mathbb{A}),\ t \trianglelefteq_{T\mathbb{A}} t'' \wedge t'' \leq t'$.

More generally, given a domain X, we can construct a domain $T'X$ whose underlying set is TX. The order of this new domain is defined as $\leq_{T\mathbb{A}}$, using the order on X instead of $\trianglelefteq$. This definition gives us an endofunctor T' on the category of domains, and so we can see the order $\leq_{T\mathbb{A}}$ simply as the order of $T'(\mathbb{A}, \trianglelefteq)$.

For each of the three orders defined above, we will define a notion of monotonicity and a notion of continuity for quantitative modalities.

Definition 6.3.6. A quantitative modality $q \in \mathcal{Q}$ can have the following properties:

- q is *tree-monotone* if for any two $t, r \in T\mathbb{A}$, $(t \leq r) \Rightarrow (\llbracket q \rrbracket(t) \trianglelefteq \llbracket q \rrbracket(r))$.
- q is *leaf-monotone* if for any two $t, r \in T\mathbb{A}$, $(t \trianglelefteq_{T\mathbb{A}} r) \Rightarrow (\llbracket q \rrbracket(t) \trianglelefteq \llbracket q \rrbracket(r))$.
- q is *monotone* if for any two $t, r \in T\mathbb{A}$, $(t \leq_{T\mathbb{A}} r) \Rightarrow (\llbracket q \rrbracket(t) \trianglelefteq \llbracket q \rrbracket(r))$.

- q is *Scott tree-continuous* is for any sequence $t_0 \leq t_1 \leq t_2 \leq \ldots$ of trees and its limit $\sqcup_n t_n$, $\llbracket q \rrbracket(\sqcup_n t_n) = \bigvee_n \llbracket q \rrbracket(t_n)$.
- q is *Scott leaf-continuous* is for any sequence $t_0 \trianglelefteq_{T\mathbb{A}} t_1 \trianglelefteq_{T\mathbb{A}} t_2 \trianglelefteq_{T\mathbb{A}} \ldots$ of trees and its limit $\sqcup_n^{\trianglelefteq_{T\mathbb{A}}}$, $\llbracket q \rrbracket(\sqcup_n^{\trianglelefteq_{T\mathbb{A}}} t_n) = \bigvee_n \llbracket q \rrbracket(t_n)$.
- q is *Scott continuous* is for any sequence $t_0 \leq_{T\mathbb{A}} t_1 \leq_{T\mathbb{A}} t_2 \leq_{T\mathbb{A}} \ldots$ of trees and its limit $\sqcup_n^{\leq_{T\mathbb{A}}} t_n$, $\llbracket q \rrbracket(\sqcup_n^{\leq_{T\mathbb{A}}} t_n) = \bigvee_n \llbracket q \rrbracket(t_n)$.

Note that Scott tree-continuity implies tree-monotonicity, Scott leaf-continuity implies leaf-monotonicity, and Scott continuity implies monotonicity. Moreover, q is both Scott tree-continuous and Scott leaf-continuous, if and only if q is Scott continuous, and similarly for monotonicity.

We see the notion of Scott continuity as the quantitative analogue to Scott openness, and we will see that all of the examples of quantitative modalities given in this chapter are Scott continuous. However, in order to prove that the induced behavioural preorders are compatible, we only need leaf-monotonicity together with Scott tree-continuity. The property of leaf-monotonicity has a similar role as the property of leaf-upwards closure in Chapter 3, being used in a lot of auxiliary results.

Decomposability

We are now going to find an analogue to the notion of decomposability, from Definition 3.3.20, in the quantitative setting. This analogue asserts a property of quantitative modalities with respect to the monad multiplication map $\mu : TTX \to TX$. We first redefine the behavioural preorder $\preccurlyeq$ from Definition 3.3.13 as a preorder on $T\mathbb{A}$, and then redefine $\preccurlyeq$ from Definition 3.3.17 as a preorder on $TT\mathbb{A}$.

We write $h : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$ to say that h is a *monotone* function from $\mathbb{A}$ to $\mathbb{A}$, so $a \trianglelefteq b \Rightarrow h(a) \trianglelefteq h(b)$. Remember that for a function $h : X \to Y$ we write $h^* : T(X) \to T(Y)$ for its *lifting* defined by $h^*(t) := t[\langle x \rangle \to \langle h(x) \rangle]$. For a quantitative predicate $h : X \to \mathbb{A}$ and a modality $q \in \mathcal{Q}$, we define a map $q(h) : TX \to \mathbb{A}$, where for $t \in TX$

we write $t \Leftarrow q(h)$ for $[\![q]\!](h^*(t)) \in \mathbb{A}$ (generalising the notation $t \in o(X)$ to quantitative modalities).

We now generalise the relation $\preccurlyeq$ from Definition 3.3.13 to the quantitative setting.

Definition 6.3.7. For any two trees $t, t' \in T\mathbb{A}$:

$$t \preccurlyeq t' \quad \iff \quad \forall q \in \mathcal{Q},\ \forall h : \mathbb{A} \to_{\trianglelefteq} \mathbb{A},\ \ (t \Leftarrow q(h)) \ \trianglelefteq\ (t' \Leftarrow q(h)) \ .$$

For any relation $\mathcal{R} \subseteq X \times Y$, and quantitative predicate $h : X \to \mathbb{A}$, we define $(\mathcal{R}^{\uparrow}[h]) : Y \to \mathbb{A}$ to be the function such that $(\mathcal{R}^{\uparrow}[h])(b) := \sup_{a \in X, a\mathcal{R}b}(h(a))$. This is a quantitative generalisation of the notion of right-set defined in Subsection 3.3.2 (just before Lemma 3.3.19). The following result generalises Proposition 3.3.14 to the quantitative setting.

Lemma 6.3.8. *If all $q \in \mathcal{Q}$ are leaf-monotone, then for any two trees $t, t' \in T\mathbb{A}$, the following three statements are equivalent:*

1. $t \preccurlyeq t'$.
2. $\forall h : \mathbb{A} \to_{\trianglelefteq} \mathbb{A},\ \ h^*(t) \ \preccurlyeq\ h^*(t')$.
3. $\forall q \in \mathcal{Q},\ \forall f : \mathbb{A} \to \mathbb{A},\ \ (t \Leftarrow q(f)) \ \trianglelefteq\ (t' \Leftarrow q(\trianglelefteq^{\uparrow}[f]))$.

Note that in point three, we use all quantitative predicates $f : \mathbb{A} \to \mathbb{A}$, not just the monotone ones.

Proof. The equivalence (1) $\Leftrightarrow$ (2) follows from the fact that the identity function on $\mathbb{A}$ is monotone, and the composition of two monotone functions is monotone.

For (1) $\Rightarrow$ (3), assume $t \preccurlyeq t'$ and take $f : \mathbb{A} \to \mathbb{A}$ and $q \in \mathcal{Q}$. Since for any $a \in \mathbb{A}$, $f(a) \trianglelefteq (\trianglelefteq^{\uparrow}[f])(a)$ and q is leaf-monotone, it holds that $(t \Leftarrow q(f)) \trianglelefteq (t \Leftarrow q(\trianglelefteq^{\uparrow}[f]))$. For $a \trianglelefteq b$, $(\trianglelefteq^{\uparrow}[f])(a) = \bigvee\{f(c) \mid c \in \mathbb{A}, c \trianglelefteq a\} \trianglelefteq \bigvee\{f(c) \mid c \in \mathbb{A}, c \trianglelefteq b\} = (\trianglelefteq^{\uparrow}[f])(b)$, so $(\trianglelefteq^{\uparrow}[f])$ is monotone. Hence by (1), $(t \Leftarrow q(\trianglelefteq^{\uparrow}[f])) \trianglelefteq (t' \Leftarrow q(\trianglelefteq^{\uparrow}[f]))$.

For (3) $\Rightarrow$ (1), note that for any monotone map $h : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$, $(\trianglelefteq^{\uparrow}[h])(a) = \bigvee\{h(b) \mid b \in \mathbb{A}, b \trianglelefteq a\} = h(a)$. □

We classify abstract quantitative behavioural properties on $T\mathbb{A}$, which act as the quantitative alternative to tree formulas defined in Subsection 3.3.2. A quantitative predicate $H : T\mathbb{A} \to \mathbb{A}$ is called *quantitative behaviourally saturated* if for any two trees $t, t' \in T\mathbb{A}$ such that $t \preccurlyeq t'$, it holds that $H(t) \trianglelefteq H(t')$. We write $QBS(T\mathbb{A})$ for the set of quantitative behaviourally saturated functions. Note that for $H \in QBS(T\mathbb{A})$, by definition $H = (\preccurlyeq^{\uparrow}[H])$. Moreover, for any function $F : T\mathbb{A} \to \mathbb{A}$, the function $(\preccurlyeq^{\uparrow}[F])$ is in $QBS(T\mathbb{A})$. So we can conclude that for a function $H : T\mathbb{A} \to \mathbb{A}$, $H \in QBS(T\mathbb{A})$ if and only if there is a function $F : T\mathbb{A} \to \mathbb{A}$ such that $H = (\preccurlyeq^{\uparrow}[F])$. Moreover, for $q \in \mathcal{Q}$, $[\![q]\!] \in QBS(T\mathbb{A})$.

We redefine the relation $\preccurlyeq$ from Definition 3.3.17 as a relation on *quantitative double trees* $TT\mathbb{A}$.

Definition 6.3.9. We define the preorder $\preccurlyeq$ on $TT\mathbb{A}$ by: for any two quantitative double trees $r, r' \in TT\mathbb{A}$,

$$r \preccurlyeq r' \quad :\iff \quad \forall q \in \mathcal{Q}, \forall H \in QBS(T\mathbb{A}), \;\; r \leftarrow q(H) \;\trianglelefteq\; r' \leftarrow q(H).$$

The following result generalises Lemma 3.3.19.

Lemma 6.3.10. *If all modalities $q \in \mathcal{Q}$ are leaf-monotone, then for any two $r, r' \in TT\mathbb{A}$, the following three statements are equivalent:*

1. $r \preccurlyeq r'$.
2. $\forall H \in QBS(T\mathbb{A}), \;\; H^*(t) \;\preccurlyeq\; H^*(t')$.
3. $\forall F : T\mathbb{A} \to \mathbb{A}, \forall q \in \mathcal{Q}, \;\; r \leftarrow q(F) \;\trianglelefteq\; r' \leftarrow q(\preccurlyeq^{\uparrow}[F])$.

Proof. For (1) $\Rightarrow$ (2), note that for $H \in QBS(T\mathbb{A})$ and $h : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$, $(h \circ H) \in QBS(T\mathbb{A})$.

For (2) $\Rightarrow$ (1), use that the identity function $id : \mathbb{A} \to \mathbb{A}$ is monotone.

For (1) $\Rightarrow$ (3), note that for any $t \in T\mathbb{A}$, $F(t) \trianglelefteq (\preccurlyeq^{\uparrow}[F])(t)$ so the result follows from leaf-monotonicity and the fact that $(\preccurlyeq^{\uparrow}[F]) \in QBS(T\mathbb{A})$.

For (3) $\Rightarrow$ (1), use that for $H \in QBS(T\mathbb{A})$, $(\preccurlyeq^{\uparrow}[H]) = H$. □

Given the generalisations of $\preccurlyeq$ and $\preccurlyeq$ given above, the quantitative analogue to the notion of decomposability can be defined in the same way as done in Definition 3.3.20:

Definition 6.3.11. $\mathcal{Q}$ is *decomposable* if for any two double trees $r, r' \in TT\mathbb{A}$:

$$r \preccurlyeq r' \quad \Longrightarrow \quad \mu r \preccurlyeq \mu r'.$$

We can generalise Lemma 3.3.22 to the quantitative setting, which gives an equivalent formulation of decomposability assuming that all modalities are leaf-monotone.

Proposition 6.3.12. *If all $q \in \mathcal{Q}$ are leaf-monotone, then $\mathcal{Q}$ is decomposable if and only if:*

$$\forall r, r' \in TT\mathbb{A}, \;\; r \preccurlyeq r' \quad \Longrightarrow \quad \forall q \in \mathcal{Q}, \;\; [\![q]\!](\mu r) \trianglelefteq [\![q]\!](\mu r').$$

Proof. The implication from left to right is immediate, when considering the (monotone) identity function on $\mathbb{A}$.

For the other direction, assume $r \preccurlyeq r'$, and take some $h : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$. We prove that $h^{**}(r) \preccurlyeq h^{**}(r')$. For $H \in QBS(T\mathbb{A})$ and $q \in \mathcal{Q}$, $(h^{**}(r) \leftarrow q(H)) = [\![q]\!](H^*(h^{**}(r))) = [\![q]\!]((H \circ h^*)^*(r)) = (r \leftarrow q(H \circ h^*))$. By equivalence (1) $\Leftrightarrow$ (2) from Lemma 6.3.8, $(H \circ h^*) \in QBS(T\mathbb{A})$, so since $r \preccurlyeq r'$, $(r \leftarrow q(H \circ h^*)) \trianglelefteq (r' \leftarrow q(H \circ h^*))$. By the same series of equalities as given for r, it holds that $(r' \leftarrow q(H \circ h^*)) = (h^{**}(r') \leftarrow q(H))$, so we have proven that $h^{**}(r) \preccurlyeq h^{**}(r')$. By assumption, and since $h^*(\mu r) = \mu(h^{**}(r))$ and $h^*(\mu r') = \mu(h^{**}(r'))$, it holds that for any $q \in \mathcal{Q}$, $\mu r \leftarrow q(h) = [\![q]\!](\mu(h^{**}(r))) \trianglelefteq [\![q]\!](\mu(h^{**}(r'))) = (\mu r' \leftarrow q(h))$. This is for all $q \in \mathcal{Q}$ and $h : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$, so $\mu r \preccurlyeq \mu r'$. We conclude that $\mathcal{Q}$ is decomposable. □

At this stage it would be possible to formulate a quantitative analogue to strong decomposability as formulated in Definition 3.3.25. It turns out however, that all the examples formulated in this chapter satisfy and even stronger property, which only considers modalities individually.

Definition 6.3.13. A modality $q \in \mathcal{Q}$ is *sequential* if for any double tree $r \in TT\mathbb{A}$:

$$[\![q]\!](\mu r) \quad = \quad [\![q]\!]([\![q]\!]^*(r)).$$

The readers may recognise the above property as one of the conditions for $[\![q]\!]$ to be an Eilenberg-Moore algebra. We will compare the notions further in Subsection 6.3.3.

Proposition 6.3.14. *If all $q \in \mathcal{Q}$ are leaf-monotone and sequential, then $\mathcal{Q}$ is decomposable.*

Proof. We prove the equivalent characterisation of decomposability given in Proposition 6.3.12. Assume $r \preccurlyeq r'$, and let $q \in \mathcal{Q}$. Then since $[\![q]\!] \in QBS(T\mathbb{A})$ and q is sequential, $[\![q]\!](\mu r) = [\![q]\!]([\![q]\!]^*(r)) = (r \Leftarrow q([\![q]\!])) \trianglelefteq (r' \Leftarrow q([\![q]\!])) = [\![q]\!]([\![q]\!]^*(r')) = [\![q]\!](\mu r')$. So $\mathcal{Q}$ is decomposable. □

We can now formulate the central theorem.

Theorem 6.3.15 (The Generalised Compatibility Theorem)**.** *If $\mathcal{Q}$ is a decomposable set of leaf-monotone and Scott tree-continuous modalities, then the open extensions of $\equiv$ and $\sqsubseteq^+$ are compatible, hence precongruences.*

This theorem is proven in the same way as Theorem 3.3.8, following the proof given in Chapter 4. First, in Section 6.4, we define a relator specified by our set of quantitative modalities $\mathcal{Q}$. We then define applicative similarity and bisimilarity as in Section 4.2, and generalise the proofs of the Coincidence Theorems to establish that the general behavioural equivalence and the positive behavioural preorder coincide with applicative bisimilarity and applicative similarity respectively. We will then prove that this relator satisfies the properties established in Section 4.3, given that the modalities satisfy the properties of Scott tree-continuity, leaf-monotonicity and decomposability. With those properties, the proof by Howe's method from Chapter 4 can be reused to conclude that the open extensions of applicative similarity and bisimilarity are compatible, hence proving the Generalised Compatibility Theorem.

We end this subsection with two helpful lemmas.

Lemma 6.3.16. *If all modalities of $\mathcal{Q}$ are leaf-monotone, and $f, g : T(X) \to \mathbb{A}$ are functions such that for all $x \in X$, $f(x) \trianglelefteq g(x)$, then $\forall t \in T(X)$, $f^*(t) \preccurlyeq g^*(r)$.*

Proof. Let $v : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$, then since v is monotone it holds that for all $x \in X$, $v(f(x)) \trianglelefteq v(g(y))$. So for $q \in \mathcal{Q}$, since its leaf-monotone, we get $[\![q]\!](v^*(f^*(t))) = [\![q]\!]((v \circ f)^*(t)) \trianglelefteq [\![q]\!]((v \circ g)^*(t)) \trianglelefteq [\![q]\!]((v \circ g)^*(r)) = [\![q]\!](v^*(g^*(r)))$. □

Lemma 6.3.17. *For each quantitative predicate $D : Terms(\underline{\underline{\mathbf{E}}}) \to \mathbb{A}$ there is a formula $\underline{\underline{\phi}}_D \in Form(\underline{\underline{\mathbf{E}}})$ such that for all $\underline{\underline{P}} \in Terms(\underline{\underline{\mathbf{E}}})$, $(\underline{\underline{P}} \models \underline{\underline{\phi}}_D) = (\sqsubseteq^{\uparrow}[D])(V)$.*

Proof. Just take $\bigvee\{\kappa_{D(\underline{P})} \wedge \chi_{\underline{P}} \mid \underline{P} \in \mathit{Terms}(\underline{\mathbf{E}})\}$, using characteristic formulas from Lemma 6.3.3, which is a supremum over a countable set. □

In the next subsection, we will see that the examples of quantitative modalities given in this chapter satisfy the properties such that the Generalised Compatibility Theorem can be applied.

6.3.2 The examples satisfy Scott continuity and sequentiality

If a quantitative modality is constructed using the recipe laid out in Section 6.2, it is easier to prove that it satisfies properties sufficient for Theorem 6.3.15 to hold. Let q be one such modality from Section 6.2. There we defined the denotation $[\![q]\!] : T\mathbb{A} \to \mathbb{A}$ of q as the supremum of approximations $\bigvee_n [\![q]\!]_n$. In this subsection, we will prove that the modality q is both Scott continuous and sequential.

Firstly, we note that q satisfies the following property.

Definition 6.3.18. A modality q constructed using the recipe of Section 6.2 *preserves limits of ascending sequences* if: For any effect operator $\mathsf{op} : \mathbb{N}^k \times \underline{\alpha}^I \to \underline{\alpha} \in \Sigma$, a k-tuple of natural numbers $\bar{l}$, and an I-indexed family of ascending sequences $\{a_i^0 \trianglelefteq a_i^1 \trianglelefteq \dots\}_{i \in I}$ from $\mathbb{A}$, and $n \in \mathbb{N}$, $[\![q]\!]_n(\mathsf{op}(\bar{l}, i \mapsto \langle \bigvee_{m \in \mathbb{N}} a_i^m \rangle)) = \bigvee_{m \in \mathbb{N}} [\![q]\!]_n(\mathsf{op}(\bar{l}, i \mapsto \langle a_i^m \rangle))$.

Note in particular the use of 'max' in the definitions of modalities over effect operators with countable arities. This is done to make sure that the above property holds.

Lemma 6.3.19. *If q is a modality constructed using the recipe of Section 6.2, and q preserves limits of ascending sequences, then q is Scott continuous.*

Proof. We prove by induction on $n \in \mathbb{N}$ that for any sequence of trees $t_0 \leq_{T\mathbb{A}} t_1 \leq_{T\mathbb{A}} t_2 \leq_{T\mathbb{A}} \dots$, $[\![q]\!]_n(\sqcup_m^{\leq_{T\mathbb{A}}} t_m) = \bigvee_m [\![q]\!]_n(t_m)$.

If $n = 0$, then $[\![q]\!]_n(\sqcup_m^{\leq_{T\mathbb{A}}} t_m) = \boldsymbol{F} = \bigvee_m [\![q]\!]_n(t_m)$ for any sequence of trees.

For $n > 0$, assume that the statement holds for all $k < n$. Take an ascending sequence of trees $t_0 \leq_{T\mathbb{A}} t_1 \leq_{T\mathbb{A}} t_2 \leq_{T\mathbb{A}} \dots$. If all t_m are equal to $\bot$, the statement holds trivially. Otherwise, let m be the lowest number such that $t_m \neq \bot$. We do a case analysis on the root node of t_m remarking that all subsequent trees will have the same type of root node.

- If $t_m = \langle a_m \rangle$ where $a_m \in \mathbb{A}$, then all subsequent trees t_k with $k > m$ must be equal to some leaf $\langle a_k \rangle$, and hence $\sqcup_m^{\leq_{T\mathbb{A}}} t_m = \langle \bigvee_{k \geq m} a_k \rangle$. Given this, $[\![q]\!]_n(\sqcup_m^{\leq_{T\mathbb{A}}} t_m) = \bigvee_{k \geq m} a_k = \bigvee_m [\![q]\!]_n(t_m)$.
- If $t_m = \mathsf{op}(\bar{l}, i \to t_m^i)$ with $\mathsf{op} \in \Sigma$, then all subsequent trees t_k with $k > m$ must be equal to a tree of the form $\mathsf{op}(\bar{l}, i \to t_k^i)$, and $\sqcup_m^{\leq_{T\mathbb{A}}} t_m = \mathsf{op}(\bar{l}, i \to \sqcup_m^{\leq_{T\mathbb{A}}} t_m^i)$. Now by construction of q, since for any $r \in T\mathbb{A}$, $[\![q]\!]_n(r) = [\![q]\!]_n(\langle [\![q]\!]_n(r) \rangle)$, it holds that for any sequence of trees r^i, $[\![q]\!]_n(\mathsf{op}(\bar{l}, i \to r^i) = [\![q]\!]_n(\mathsf{op}(\bar{l}, i \to \langle [\![q]\!]_{n_i}(r^i) \rangle)$ where each $n_i < n$ (depending on the definition of q,

either $n_i := n-1$ or $n_i := \max(n-1-i, 0)$). So in particular, $[\![q]\!]_n(\sqcup_m^{\leq_{T\mathbb{A}}} t_m) = [\![q]\!]_n(\mathsf{op}(\bar{l}, i \to \langle [\![q]\!]_{n_i}(\sqcup_m^{\leq_{T\mathbb{A}}} t_m^i)\rangle))$.

Using the induction hypothesis followed by the assumption that q preserves limits of ascending sequences, we can conclude that $[\![q]\!]_n(\sqcup_m^{\leq_{T\mathbb{A}}} t_m) = [\![q]\!]_n(\mathsf{op}(\bar{l}, i \to \langle \bigvee_m [\![q]\!]_{n_i}(t_m^i)\rangle)) = \bigvee_m [\![q]\!]_n(\mathsf{op}(\bar{l}, i \to \langle [\![q]\!]_{n_i}(t_m^i)\rangle)) = \bigvee_m [\![q]\!]_n(\mathsf{op}(\bar{l}, i \to t_m^i)) = \bigvee_m [\![q]\!]_n(t_m)$.

This finishes the induction, so we can conclude that for any sequence of trees $t_0 \leq_{T\mathbb{A}} t_1 \leq_{T\mathbb{A}} t_2 \leq_{T\mathbb{A}} \ldots$, $[\![q]\!](\sqcup_m^{\leq_{T\mathbb{A}}} t_m) = \bigvee_n [\![q]\!]_n(\sqcup_m^{\leq_{T\mathbb{A}}} t_m) = \bigvee_n \bigvee_m [\![q]\!]_n(t_m) = \bigvee_m [\![q]\!](t_m)$. □

Lemma 6.3.20. *If q is a modality constructed using the recipe of Section 6.2, and q preserves limits of ascending sequences, then q is sequential.*

Proof. The previous lemma implies that q is Scott leaf-continuous. By construction of q it holds that for any $t \in TT\mathbb{A}$ and any two numbers $n, m \in \mathbb{N}$, it holds that:

- $[\![q]\!]_n(\mu t) \trianglelefteq [\![q]\!]_n([\![q]\!]_n^*(t))$.
- $[\![q]\!]_n([\![q]\!]_m^*(t)) \trianglelefteq [\![q]\!]_{n+m}(\mu t)$

Hence $\bigvee_n [\![q]\!]_n(\mu t) = \bigvee_{n,m} [\![q]\!]_n([\![q]\!]_m^*(t))$.

Now observe that for any $t \in TT\mathbb{A}$, we get a sequence of trees ascending in the '$\trianglelefteq_{T\mathbb{A}}$' order $[\![q]\!]_0^*(t) \trianglelefteq_{T\mathbb{A}} [\![q]\!]_1^*(t) \trianglelefteq_{T\mathbb{A}} [\![q]\!]_2^*(t) \trianglelefteq_{T\mathbb{A}} \ldots$. So by Scott leaf-continuity it holds that: $[\![q]\!](\mu t) = \bigvee_n [\![q]\!]_n(\mu t) = \bigvee_{n,m} [\![q]\!]_n([\![q]\!]_m^*(t)) = \bigvee_m [\![q]\!]([\![q]\!]_m^*(t)) = [\![q]\!](\bigvee_m [\![q]\!]_m^*(t)) = [\![q]\!]([\![q]\!]^*(t))$, hence t is sequential. □

We conclude that all the given examples of quantitative modalities are sequential and Scott continuous.

Even though all examples from this chapter are sequential, there are lots of decomposable sets of modalities which contain non-sequential modalities. Take for instance some of the Boolean modalities from Chapter 3, which can be translated to quantitative modalities on the truth space $\mathbb{A} = \mathbb{B}$. This translation will be discussed in more detail in Subsection 6.5.1. Several of the modalities from Chapter 3 are not sequential when translated into quantitative modalities. In the quantitative logic however, we have an extra degree of freedom with our ability to choose a truth space. This flexibility allows us to more easily find a sequential modality for specifying the effect behaviour.

6.3.3 Eilenberg-Moore algebras

In [26], effects are interpreted using monads, on which Eilenberg-Moore algebras are defined. Such algebras have a similar function as the modalities used in this thesis. It turns out that most of the examples of quantitative modalities defined in this chapter actually are specified by EM-algebras $[\![q]\!]$ on $\mathbb{A}$. In this subsection, we will study the connection between quantitative modalities and EM-algebras further. One principal

difference with [26] however, is that in this thesis it is sufficient to define modalities as algebras on the tree monad only.

In a category, an Eilenberg-Moore algebra on some monad structure (T, η, μ) is a morphism $a : TX \to X$ such that the following diagrams commute:

For any of the quantitative modalities q of the given examples in Section 6.2 it holds that $[\![q]\!] : T(\mathbb{A}) \to \mathbb{A}$ is an Eilenberg-Moore algebra in the category of sets, with respect to the monad structure $(T(-), \eta, \mu)$. There is a tight connection between the property of sequentiality and Eilenberg-Moore algebras.

Lemma 6.3.21. *For $q \in \mathcal{Q}$, if $[\![q]\!] : T(\mathbb{A}) \to \mathbb{A}$ is an EM-algebra, then q is sequential.*

Proof. The second diagram for EM-algebras is precisely the equation required for sequentiality. □

As said before, the quantitative modalities from our examples in this chapter do naturally form EM-algebras. Informally, the μ-diagram naturally corresponds to the notion of preservation over sequencing, whereas the η-diagram corresponds to preservation over $\mathsf{return}(-)$. It is however not necessary for our quantitative modalities to denote an algebra satisfying the η-diagram. A similar property which does hold for our modalities q, which seem to be sufficient for our purposes, is that $\forall a, b \in \mathbb{A},\ a \trianglelefteq b \Rightarrow [\![q]\!](\langle a \rangle) \trianglelefteq [\![q]\!](\langle b \rangle)$.

It is however interesting to note that the η-diagram is *almost* a consequence of the μ-diagram because of the following observation.

Lemma 6.3.22. *Suppose $a : TX \to X$ is a surjective morphism in the category of sets. If the μ-diagram commutes, then the η-diagram commutes.*

Proof. Let $x \in X$, then by surjectivity there is a $t \in TX$ such that $a(t) = x$. Since $\eta(t) \in TTX$, we can use the μ-diagram to derive that:

$$a(\eta(x)) = a(\eta(a(t))) = a(Ta(\eta(t))) = a(\mu_X(\eta(t))) = a(t) = x \ .$$

□

The same proof applies when $a : TX \to X$ is a regular epimorphism and T is a monad on a regular category.

Recall that there is an endofunctor T' in the category of domains, where $T'\mathbb{A}$ has as underlying set $T\mathbb{A}$ and as order $\leq_{T\mathbb{A}}$. Given this, a modality q is Scott continuous, if and only if its denotation $[\![q]\!] : T(\mathbb{A}) \to \mathbb{A}$ specifies a morphism from $T'\mathbb{A}$ to $\mathbb{A}$ in the category of domains. Moreover, this endofunctor T' forms a monad $(T'(-), \eta, \mu)$ in the category of domains. As such, we can state the following sufficient condition for compatibility.

Proposition 6.3.23. *If for any $q \in \mathbb{A}$, $[\![q]\!] : T'\mathbb{A} \to \mathbb{A}$ is an Eilenberg-Moore algebra on $\mathbb{A}$ w.r.t. $(T'(-), \eta, \mu)$ in the category of domains, then $\mathcal{Q}$ satisfies the compatibility properties, and hence the open extensions of the positive behavioural preorder and the behavioural equivalence are both compatible.*

We have not found an example of an effect which can only be described using a decomposable set of Scott open modalities which are not EM-algebras. However, by weakening the conditions for 'correct' sets of modalities, there is more freedom in the choice of logic for any particular effect. For example, the Boolean logics for probability and global store have modalities which do not form EM-algebras. Similarly, the combination of probability and global store could also be modelled with truth spaces $\mathbb{B}$, $[0,1]$, and $\mathcal{P}(\mathbb{N}^{\mathsf{Loc}})$, for which the 'correct' modalities also do not denote EM-algebras.

As noted before, the possibility to choose a truth space $\mathbb{A}$, on top of choosing modalities, gives us a flexibility which allows us to more easily find an Eilenberg-Moore algebra. In most cases, the truth space is used to combine multiple Boolean modalities into one quantitative modality denoting an EM-algebra. Take for instance the effect of probability, which has a Boolean modality $\mathsf{P}_{>r}$ for each rational probability $0 \leq r \leq 1$ (see Subsection 3.2.4), all of which can be combined into one quantitative modality E by taking a truth space of all probabilities $[0,1]$ (see Subsection 6.2.1).

6.3.4 Pure quantitative logic

As in Section 5.4, we can define a pure variation of the quantitative logic. We replace the formula constructor $(V \mapsto \underline{\phi})$ with another formula constructor,

$$\frac{\phi \in \mathit{Form}(\mathbf{A}) \qquad \underline{\psi} \in \mathit{Form}(\underline{\mathbf{C}})}{(\phi \mapsto \underline{\psi}) \in \mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})}$$

for which the satisfaction relation is defined by

$$\underline{M} \models (\phi \mapsto \underline{\psi}) \;:=\; \inf\{\, \max\{\, \neg(V \models \phi),\;\; \underline{M}\; V \models \underline{\psi}\} \;\mid\; V \in \mathit{Terms}(\mathbf{A})\} \;.$$

The expression is inspired by the standard equality for implication $(a \Rightarrow b) \equiv (b \vee (\neg a))$. Alternatively, we can give a simpler different definition, not using involution:

$$\underline{M} \models [\phi \mapsto \underline{\psi}] \;:=\; \inf\{\, \underline{M}\; V \models \underline{\psi} \;\mid\; V \in \mathit{Terms}(\mathbf{A}),\; (V \models \phi) = \boldsymbol{T}\} \;.$$

Firstly note that if $\mathbb{A}$ is equal to the Booleans, then the two formulas $(\phi \mapsto \underline{\psi})$ and $[\phi \mapsto \underline{\psi}]$ are equivalent. Moreover, the two logics formulated by replacing the $(V \mapsto \underline{\phi})$ constructor with $(\phi \mapsto \underline{\psi})$ and $[\phi \mapsto \underline{\psi}]$ respectively, yield equi-expressive logics. This is because each formula can be expressed in terms of the other using the other non-basic constructors in the logic:

$$[\phi \mapsto \underline{\psi}] \;\equiv\; (\phi_{\trianglerighteq \boldsymbol{T}} \mapsto \underline{\psi}) \;,$$

$$(\phi \mapsto \underline{\psi}) \;\equiv\; \bigwedge\{\, [\neg((\neg\phi)_{\trianglerighteq a}) \mapsto \underline{\psi}] \vee \kappa_a \;\mid\; a \in \mathbb{A}\} \;.$$

From here on out, we will only consider using $(\phi \mapsto \underline{\psi})$.

Any formula of the form $(\phi \mapsto \underline{\psi})$ can be expressed in $\mathcal{U}$, because of the equivalence

$$(\phi \mapsto \underline{\psi}) \quad \equiv \quad \bigwedge\{\bigvee\{\kappa_{\neg(V\models\phi)}, (V \mapsto \underline{\psi})\} \quad | \quad V \in \mathit{Terms}(\mathbf{A})\} \tag{6.1}$$

Note that $\kappa_{\neg(V\models\phi)}$ is the constant formula for the value $\neg(V \models \phi)$, and does not actually use $\neg$ as a formula constructor. So the constructor can also be expressed in the positive logic $\mathcal{U}^+$.

Let $\mathcal{W}$ and $\mathcal{W}^+$ be the variants of $\mathcal{U}$ and $\mathcal{U}^+$ respectively, for which the definition of formulas uses $(\phi \mapsto \underline{\psi})$ instead of $(V \mapsto \underline{\psi})$. We call $\sqsubseteq_{\mathcal{W}}$ the *pure behavioural preorder* and $\sqsubseteq_{\mathcal{W}^+}$ the *pure positive behavioural preorder* respectively. As argued in Section 5.4, a conceptual advantage of the logic $\mathcal{W}$ is that its syntax is independent of the term syntax of the programming language, a property which $\mathcal{U}$ clearly does not enjoy. Note that we can still see $\mathcal{W}$ and $\mathcal{W}^+$ as fragments of $\mathcal{U}$ by translating each $(\phi \mapsto \underline{\psi})$ in terms of formulas from $\mathcal{U}$ using equivalence (6.1).

Proposition 6.3.24. *If $\sqsubseteq^{\circ}_{\mathcal{U}}$ is compatible, then $\sqsubseteq_{\mathcal{U}} = \sqsubseteq_{\mathcal{W}}$. If $\sqsubseteq^{\circ}_{\mathcal{U}^+}$ is compatible, then $\sqsubseteq_{\mathcal{U}^+} = \sqsubseteq_{\mathcal{W}^+}$.*

Proof. We prove that any formula $\underset{\sim}{\phi} \in \mathcal{U}$ has an equivalent formula $\underset{\sim}{\phi}^* \in \mathcal{W}$. This is done by an induction on types, and for each type and induction on its formulas. There is only one non-trivial case, for function types $\mathbf{A} \to \underline{\mathbf{C}}$.

Let $(V \mapsto \underline{\phi}) \in \mathcal{U}$. We use Lemma 6.3.3 on $\mathcal{W}$ and V to find $\chi^V \in \mathcal{W}$, and induction hypothesis on $\underline{\phi} \in \mathcal{U}$ to find $\underline{\phi}^* \in \mathcal{W}$. We define $(V \mapsto \underline{\phi})^* \in \mathcal{W}$ by $(\chi^V \mapsto \underline{\phi}^*)$. It holds that $M \models (V \mapsto \underline{\phi})^* := \bigwedge\{\boldsymbol{T} \mid W : \mathbf{A}, V \not\sqsubseteq_{\mathcal{W}} W\} \wedge \bigwedge\{\underline{M}\ W \models \underline{\phi} \mid W : \mathbf{A}, V \sqsubseteq_{\mathcal{W}} W\} = \bigwedge\{\underline{M}\ W \models \underline{\phi} \mid W : \mathbf{A}, V \sqsubseteq_{\mathcal{W}} W\} \trianglelefteq (\underline{M}\ V \models \underline{\phi})$ since $V \sqsubseteq_{\mathcal{W}} V$.

By Induction Hypothesis on $\mathbf{A}$ we have $V \sqsubseteq_{\mathcal{W}} W \Rightarrow V \sqsubseteq_{\mathcal{U}} W$, and by compatibility we have $V \sqsubseteq_{\mathcal{U}} W \Rightarrow \underline{M}\ V \sqsubseteq_{\mathcal{U}} \underline{M}\ W$. Hence $\underline{M} \models (V \mapsto \underline{\phi})^* \trianglerighteq (\underline{M}\ V \models \underline{\phi})$. We can conclude that $M \models (V \mapsto \underline{\phi})^* = (\underline{M}\ V \models \underline{\phi})$ and hence $(V \mapsto \underline{\phi}) \equiv (V \mapsto \underline{\phi})^* \in \mathcal{W}$. □

Note that the above proof works as well if we used the other formula constructor $[\phi \mapsto \underline{\psi}]$ instead. This is mainly because $(\chi_V \mapsto \underline{\psi}) \ \equiv \ [\chi_V \mapsto \underline{\psi}]$.

6.4 Applicative $\mathcal{Q}$-simulations

In this section, we illustrate the connection between behavioural equivalence and applicative bisimilarity, generalising the results from Chapter 4. We start with the notion of a $\mathcal{Q}$-relator.

Definition 6.4.1. Given a set of quantitative modalities $\mathcal{Q}$, we define an operator $\mathcal{Q}(-) : \mathcal{P}(X \times Y) \to \mathcal{P}(TX \times TY)$ for any two sets X and Y, given by:

$$t\ \mathcal{Q}(\mathcal{R})\ t' \quad \iff \quad \forall h : X \to \mathbb{A},\ \forall q \in \mathcal{Q},\ \ t \Subset q(h) \ \trianglelefteq\ t' \Subset q(\mathcal{R}^{\uparrow}[h])$$

We repeat the development from Chapter 4, generalising the results from the Boolean to the quantitative setting. The proofs of the generalised results are very similar to the proofs of the original results. We start with the result that the above operator is in fact a relator in the sense of Definition 4.1.1, which generalises Lemma 4.1.2.

Lemma 6.4.2. *If all $q \in \mathcal{Q}$ are leaf-monotone, then $\mathcal{Q}(-)$ is a* relator, *meaning it has the following properties:*

1. *For any set X, $=_{T(X)} \subseteq \mathcal{Q}(=_X)$.*
2. *For $\mathcal{R} \subseteq X \times Y$ and $\mathcal{S} \subseteq Y \times Z$, $\mathcal{Q}(\mathcal{R})\mathcal{Q}(\mathcal{S}) \subseteq \mathcal{Q}(\mathcal{R}\mathcal{S})$.*
3. *If $\mathcal{R} \subseteq \mathcal{S} \subseteq X \times Y$, then $\mathcal{Q}(\mathcal{R}) \subseteq \mathcal{Q}(\mathcal{S})$.*
4. *For $f : X \to Z$, $g : Y \to W$, and $\mathcal{R} \subseteq Z \times W$, $\mathcal{Q}(\{(x,y) \in X \times Y \mid f(x)\mathcal{R}g(y)\}) = \{(t,r) \in TX \times TY \mid f^*(t)\mathcal{Q}(\mathcal{R})g^*(r)\}$.*

Proof.

1. For any $x \in X$, $({=_X}^{\uparrow}[h])(x) = h(x)$, so $t \Leftarrow q(h) \trianglelefteq t \Leftarrow q(\mathcal{R}^{\uparrow}[h])$.
2. $\forall z \in Z$, $(\mathcal{R}\mathcal{S}^{\uparrow}[h])(z) = \sup\{h(z) \mid x \in X, x\mathcal{R}\mathcal{S}z\} =$
   $\sup\{h(z) \mid x \in X, y \in Y, x\mathcal{R}y, y\mathcal{S}z\} = (\mathcal{S}^{\uparrow}[(\mathcal{R}^{\uparrow}[h])])(z)$.
3. If $\mathcal{R} \subseteq \mathcal{S}$, then $\forall x \in X$, $(\mathcal{R}^{\uparrow}[h])(x) \trianglelefteq (\mathcal{S}^{\uparrow}[h])(x)$, so $t' \Leftarrow q(\mathcal{R}^{\uparrow}[h]) \trianglelefteq t' \Leftarrow q(\mathcal{S}^{\uparrow}[h])$.
4. Assume $t\ \mathcal{Q}(\{(x,y) \in X \times Y \mid f(x)\mathcal{R}f(y)\})\ r$, we prove $f^*(t)\mathcal{Q}(\mathcal{R})g^*(r)$. Let $h : Z \to \mathbb{A}$ and $q \in \mathcal{Q}$, then $(f^*(t) \Leftarrow q(h)) = (t \Leftarrow q(h \circ f)) \trianglelefteq$
   $(r \Leftarrow q(\lambda y.\, \sup\{h(f(x)) \mid x \in X, f(x)\mathcal{R}g(y)\})) \trianglelefteq$
   $(r \Leftarrow q(\lambda y.\, \sup\{h(z) \mid z \in Z, z\mathcal{R}g(y)\})) \trianglelefteq$
   $(r \Leftarrow q((\mathcal{R}^{\uparrow}[h]) \circ g)) = (g^*(r) \Leftarrow q(\mathcal{R}^{\uparrow}[h]))$.

   Assume $f^*(t)\mathcal{Q}(\mathcal{R})g^*(r)$, we prove $t\ \mathcal{Q}(\{(x,y) \in X \times Y \mid f(x)\mathcal{R}g(y)\})\ r$. Let $h : X \to \mathbb{A}$ and $q \in \mathcal{Q}$, then $(t \Leftarrow q(h)) \trianglelefteq$
   $(f^*(t) \Leftarrow q(\lambda z.\, \sup\{h(x) \mid x \in X, f(x) = z\})) \trianglelefteq$
   $(g^*(r) \Leftarrow q(\mathcal{R}^{\uparrow}[(\lambda z.\, \sup\{h(x) \mid x \in X, f(x) = z\})])) \trianglelefteq$
   $(g^*(r) \Leftarrow q(\lambda w.\, \sup\{\sup\{h(x) \mid x \in X, f(x) = z\} \mid z \in Z, z\mathcal{R}w\}) \trianglelefteq$
   $(g^*(r) \Leftarrow q(\lambda w.\, \sup\{h(x) \mid x \in X, f(x)\mathcal{R}w\}) =$
   $(r \Leftarrow q(\lambda y.\, \sup\{h(x) \mid x \in X, f(x)\mathcal{R}g(y)\})) =$
   $(r \Leftarrow q(\{(x,y) \in X \times Y \mid f(x)\mathcal{R}g(y)\}^{\uparrow}[h]))$.

□

We will call $\mathcal{Q}(-)$ the *$\mathcal{Q}$-relator*. The following lemma gives an alternative definition of the $\mathcal{Q}$-relator. This uses the quantitative generalisation of right-set $(\mathcal{R}^{\uparrow}[h])$ as defined just above Lemma 6.3.8.

Lemma 6.4.3. *If all modalities $q \in \mathcal{Q}$ are leaf-monotone, then for any $\mathcal{R} \subseteq X \times Y$:*

$$t\,\mathcal{Q}(\mathcal{R})\,r \quad \Longleftrightarrow \quad \forall h : X \to \mathbb{A},\ h^*(t) \preccurlyeq (\mathcal{R}^{\uparrow}[h])^*(r)$$

Proof. The '$\Leftarrow$' proof is done by unfolding the definition of $\preccurlyeq$ with the identity function $id : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$.

For the '$\Rightarrow$' proof, assume $t\,\mathcal{Q}(\mathcal{R})\,r$, take $h : X \to \mathbb{A}$, we need to prove $h^*(t) \preccurlyeq (\mathcal{R}^{\uparrow}[h])^*(r)$. Let $v : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$ and $q \in \mathcal{Q}$, then $h^*(t) \Subset q(v) = t \Subset q(v \circ h) \trianglelefteq r \Subset q(\mathcal{R}^{\uparrow}[(v \circ h)])$. Now, for any $y \in Y$, $(\mathcal{R}^{\uparrow}[(v \circ h)])(y) = \sup\{v(h(x)) \mid x\,\mathcal{R}\,y\} \trianglelefteq v(\sup\{h(x) \mid x\,\mathcal{R}\,y\})$ since v is monotone. Hence, since q is leaf-monotone, $h^*(t) \Subset q(v) \trianglelefteq (\mathcal{R}^{\uparrow}[h])^*(r) \Subset q(v)$. We conclude that $h^*(t) \preccurlyeq (\mathcal{R}^{\uparrow}[h])^*(r)$. □

An applicative $\mathcal{Q}$-simulation is an applicative $\mathcal{O}$-simulation from Definition 4.2.1, using the relator $\mathcal{Q}(-)$ in place of $\mathcal{O}(-)$.

Using Lemma 6.3.17, we can generalise the proof from Lemma 4.2.5 to establish:

Lemma 6.4.4. *If all $q \in \mathcal{Q}$ are leaf-monotone, then $\sqsubseteq^+$ is an applicative $\mathcal{Q}$-simulation.*

Proof. We establish that $\sqsubseteq^+$ is a $\mathcal{Q}$-simulation by proving it satisfies all the simulation rules from Definition 4.2.1.

1. **N**. If $\overline{n} \sqsubseteq^+ \overline{m}$ then $\boldsymbol{T} = (\overline{m} \models \{m\}) \trianglelefteq (\overline{n} \models \{m\})$, hence $n = m$.

2. **U** $\underline{\mathbf{C}}$. If $\mathsf{thunk}(\underline{M}) \sqsubseteq^+ \mathsf{thunk}(\underline{N})$ then $(\mathsf{force}(\mathsf{thunk}(\underline{M})) \models \underline{\phi}) = (\mathsf{thunk}(\underline{M}) \models \langle\underline{\phi}\rangle) \trianglelefteq (\mathsf{thunk}(\underline{N}) \models \langle\underline{\phi}\rangle) = (\mathsf{force}(\mathsf{thunk}(\underline{M})) \models \underline{\phi})$, hence $\mathsf{force}(\mathsf{thunk}(\underline{M})) \sqsubseteq^+ \mathsf{force}(\mathsf{thunk}(\underline{M}))$. By Lemma 6.3.4, since $|\mathsf{force}(\mathsf{thunk}(\underline{M}))| = |\underline{M}|$ and $|\mathsf{force}(\mathsf{thunk}(\underline{N}))| = |\underline{N}|$ we conclude that $\underline{M} \sqsubseteq^+ \underline{N}$.

3. $\boldsymbol{\Sigma}_{i\in I}\,\mathbf{A}_i$. Assume $(j, V) \sqsubseteq^+ (k, W)$ then $((j, V) \models (j, \kappa_{\boldsymbol{T}})) = \boldsymbol{T}$ hence $((k, W) \models (j, \kappa_{\boldsymbol{T}})) = \boldsymbol{T}$ so $k = j$. For any $\phi \in \mathit{Form}(\mathbf{A}_j)$ we now have $(V \models \phi) = ((j, V) \models (j, \phi)) \trianglelefteq ((j, W) \models (j, \phi)) = (W \models \phi)$, so we conclude that $V \sqsubseteq^+ W$.

4. $\mathbf{A} \times \mathbf{B}$. Assume $(V, V') \sqsubseteq^+ (W, W')$. For any $\phi \in \mathit{Form}(\mathbf{A})$ we have $(V \models \phi) = ((V, V') \models \pi_0(\phi)) \trianglelefteq ((W, W') \models \pi_0(\phi)) = (W \models \phi)$. We can conclude that $V \sqsubseteq^+ W$ and with a similar proof $V' \sqsubseteq^+ W'$.

5. $\mathbf{A} \to \underline{\mathbf{C}}$. Assume $\underline{M} \sqsubseteq^+ \underline{N}$ and $V : \mathbf{A}$, then $(\underline{M}\ V \models \underline{\phi}) \trianglelefteq (\underline{M} \models (V \mapsto \underline{\phi})) \trianglelefteq (\underline{N} \models (V \mapsto \underline{\phi})) \trianglelefteq (\underline{N}\ V \models \underline{\phi})$. So $\underline{M}\ V \sqsubseteq^+ \underline{N}\ V$ for all $V : \mathbf{A}$.

6. $\mathbf{F}\,\mathbf{A}$. Assume $\underline{M} \sqsubseteq^+ \underline{N}$, $q \in \mathcal{Q}$ and $D : \mathit{Terms}(\mathbf{A}) \to \mathbb{A}$. We use Lemma 6.3.17 to find a formula ϕ^D such that $(V \models \phi^D) = (\sqsubseteq^{+\uparrow}[D])(V)$. By reflexivity of $\sqsubseteq^+$, we have $D(V) \trianglelefteq (\sqsubseteq^{+\uparrow}[D])(V)$, so by leaf-monotonicity and $\underline{M} \sqsubseteq^+ \underline{N}$ it holds that:

$$|\underline{M}| \Subset q(D) \;\trianglelefteq\; (\underline{M} \models q(\phi^D)) \;\trianglelefteq\; (\underline{N} \models q(\phi^D)) \;=\; |\underline{N}| \Subset q(\sqsubseteq^{+\uparrow}[D]).$$

   We can conclude that $|\underline{M}|\,\mathcal{Q}(\sqsubseteq^+_{\mathbf{A}})\,|\underline{N}|$.

7. $\boldsymbol{\Pi}_{i\in I}\,\underline{\mathbf{C}}_i$. Assume $\underline{M} \sqsubseteq^+ \underline{N}$. Take some $j \in I$, then for any $\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}_j)$ we have $(\underline{M}\ j \models \underline{\phi}) = (\underline{M} \models (j \mapsto \underline{\phi})) \trianglelefteq (\underline{N} \models (j \mapsto \underline{\phi})) = (\underline{N}\ j \models \underline{\phi})$. Hence $\underline{M}\ j \sqsubseteq^+ \underline{N}\ j$.

□

Note that the above result can be expanded in a straightforward way to show that the characterisations of Lemmas 3.4.3 and 4.1.4 also hold for the quantitative logics with leaf-monotone modalities. The above proof works for the general logic as well. Hence we get the following result:

Lemma 6.4.5. *If all $q \in \mathcal{Q}$ are leaf-monotone, then $\equiv$ is an applicative $\mathcal{Q}$-bisimulation.*

Like in Lemma 4.2.6, we prove that any simulation is a subset of $\sqsubseteq^+$ using an induction on formulas. We say a well-typed relation $\mathcal{R}$ preserves a formula $\underline{\underline{\phi}} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$ if, for any $\underline{\underline{P}}, \underline{\underline{R}} \in \mathit{Terms}(\underline{\underline{\mathbf{E}}})$, $\underline{\underline{P}}\,\mathcal{R}_{\underline{\underline{\mathbf{E}}}}\,\underline{\underline{R}} \implies (\underline{\underline{P}} \models \underline{\underline{\phi}}) \trianglelefteq (\underline{\underline{R}} \models \underline{\underline{\phi}})$.

Lemma 6.4.6. *If all $q \in \mathcal{Q}$ are leaf-monotone, then any applicative $\mathcal{Q}$-simulation is a subset of $\sqsubseteq^+$.*

Proof. Let $\mathcal{R}$ be an applicative $\mathcal{O}$-simulation. We prove by induction on formulas $\underline{\underline{\phi}} \in \mathcal{U}^+$ that $\mathcal{R}$ preserves $\underline{\underline{\phi}}$. Given this, the result is immediate. We look at the cases for $\underline{\underline{\phi}}$, and assume as induction hypothesis that $\mathcal{R}$ preserves any subformula of $\underline{\underline{\phi}}$.

1. $\{n\} \in \mathit{Form}(\mathbf{N})$, $V\,\mathcal{R}_{\mathbf{N}}\,W \implies (V = W) \implies (V \models \{n\}) \trianglelefteq (W \models \{n\})$.

2. $\langle \underline{\underline{\phi}} \rangle \in \mathit{Form}(\mathbf{U}\,\underline{\mathbf{C}})$, $\mathsf{thunk}(\underline{M})\,\mathcal{R}_{\mathbf{U}\,\underline{\mathbf{C}}}\,\mathsf{thunk}(\underline{N}) \implies \underline{M}\,\mathcal{R}_{\underline{\mathbf{C}}}\,\underline{N} \implies (\mathsf{thunk}(\underline{M}) \models \langle \underline{\underline{\phi}} \rangle) = (\underline{M} \models \underline{\underline{\phi}}) \trianglelefteq (\underline{N} \models \underline{\underline{\phi}}) = (\mathsf{thunk}(\underline{N}) \models \langle \underline{\underline{\phi}} \rangle)$, using Lemma 6.3.4.

3. $(j, \phi) \in \mathit{Form}(\mathbf{\Sigma}_{i \in I}\,\mathbf{A}_i)$, assume $(k, V)\,\mathcal{R}_{\mathbf{\Sigma}_{i \in I}\,\mathbf{A}_i}\,(h, W) \implies (k = h = j)$. If $(k, V) \models (j, \phi) = \boldsymbol{F}$ we are finished. If $(k, V) \models (j, \phi) \neq \boldsymbol{F}$ then $k = j$, so we get $(k = h = j)$ and $V\,\mathcal{R}_{\mathbf{A}_j}\,W$ and hence $((k, V) \models (j, \phi)) = (V \models \phi) \trianglelefteq (W \models \phi) = ((h, W) \models (j, \phi))$.

4. $\pi_0(\phi) \in \mathit{Form}(\mathbf{A} \times \mathbf{B})$, then $(V, V')\,\mathcal{R}_{\mathbf{A} \times \mathbf{B}}\,(W, W') \implies V\,\mathcal{R}_{\mathbf{A}}\,W \implies ((V, V') \models \pi_0(\phi)) = (V \models \phi) \trianglelefteq (W \models \phi) = ((W, W') \models \pi_0(\phi))$. Similarly for $\pi_1(\phi)$.

5. $(V \mapsto \underline{\underline{\phi}}) \in \mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})$, $\underline{M}\,\mathcal{R}_{\mathbf{A} \to \underline{\mathbf{C}}}\,\underline{N} \implies \underline{M}\,V\,\mathcal{R}_{\underline{\mathbf{C}}}\,\underline{N}\,V \implies (\underline{M} \models (V \mapsto \underline{\underline{\phi}})) = (\underline{M}\,V \models \underline{\underline{\phi}}) \trianglelefteq (\underline{N}\,V \models \underline{\underline{\phi}}) = (\underline{N} \models (V \mapsto \underline{\underline{\phi}}))$.

6. $q(\phi) \in \mathit{Form}(\mathbf{F}\,\mathbf{A})$, by induction hypothesis it holds that $V\,\mathcal{R}_{\mathbf{A}}\,W \implies (V \models \phi) \trianglelefteq (W \models \phi)$, so $(\mathcal{R}^{\uparrow}[\phi])(W) = \sup\{V \models \phi \mid V : \mathbf{A}, V\,\mathcal{R}_{\mathbf{A}}\,W\} \trianglelefteq (W \models \phi)$. Since $\underline{M}\,\mathcal{R}_{\mathbf{F}\,\mathbf{A}}\,\underline{N}$, it holds that $|\underline{M}|\,\mathcal{Q}(\mathcal{R}_{\mathbf{A}})\,|\underline{N}|$, and hence $(\underline{M} \models q(\phi)) = [\![q]\!](|\underline{M}|[\models \phi]) \trianglelefteq |\underline{N}| \lessdot [\![q]\!](\mathcal{R}^{\uparrow}[\phi]) \trianglelefteq [\![q]\!](|\underline{N}|[\models \phi]) = (\underline{N} \models q(\phi))$.

7. $(j \mapsto \underline{\underline{\phi}}) \in \mathit{Form}(\mathbf{\Pi}_{i \in I}\,\underline{\mathbf{C}}_i)$, $\underline{M}\,\mathcal{R}_{\mathbf{\Pi}_{i \in I}\,\underline{\mathbf{C}}_i}\,\underline{N} \implies \underline{M}\,j\,\mathcal{R}_{\underline{\mathbf{C}}_j}\,\underline{N}\,j \implies (\underline{M} \models (j \mapsto \underline{\underline{\phi}})) = (\underline{M}\,j \models \underline{\underline{\phi}}) \trianglelefteq (\underline{N}\,j \models \underline{\underline{\phi}}) = (\underline{N} \models (j \mapsto \underline{\underline{\phi}}))$.

8. $\bigvee X, \bigwedge X \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$, let $\underline{\underline{P}}\,\mathcal{R}_{\underline{\underline{\mathbf{E}}}}\,\underline{\underline{P}}$, then by induction hypothesis it holds that. for all $\underline{\underline{\phi}} \in X$, $(\underline{\underline{P}} \models \underline{\underline{\phi}}) \trianglelefteq (\underline{\underline{R}} \models \underline{\underline{\phi}})$. So $(\underline{\underline{P}} \models \bigvee X) \trianglelefteq (\underline{\underline{R}} \models \bigvee X)$ and $(\underline{\underline{P}} \models \bigwedge X) \trianglelefteq (\underline{\underline{R}} \models \bigwedge X)$.

9. $\kappa_a \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$, then $(V \models \kappa_a) = a \trianglelefteq a = (W \models \kappa_a)$.

10. $\underline{\underline{\phi}}_{\trianglerighteq a} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$, let $\underline{\underline{P}}\,\mathcal{R}_{\underline{\underline{\mathbf{E}}}}\,\underline{\underline{R}}$, then by induction hypothesis $(\underline{\underline{P}} \models \underline{\underline{\phi}}) \trianglelefteq (\underline{\underline{R}} \models \underline{\underline{\phi}})$, so $(\underline{\underline{P}} \models \underline{\underline{\phi}}_{\trianglerighteq a}) \neq \boldsymbol{F} \implies (\underline{\underline{P}} \models \underline{\underline{\phi}}) \trianglerighteq a \implies (\underline{\underline{R}} \models \underline{\underline{\phi}}) \trianglerighteq a \implies (\underline{\underline{R}} \models \underline{\underline{\phi}}_{\trianglerighteq a}) = \boldsymbol{T}$.

□

Lemma 6.4.7. *If all $q \in \mathcal{Q}$ are leaf-monotone, then any applicative $\mathcal{O}$-bisimulation is a subset of $\equiv$.*

Proof. Assume $\mathcal{R}$ is an applicative $\mathcal{O}$-bisimulation. We prove by induction on $\underline{\underline{\phi}} \in \mathcal{U}$ that $\mathcal{R}$ preserves $\underline{\underline{\phi}}$. Since $\mathcal{R}$ is symmetric, the fact that $\mathcal{R}$ preserves $\underline{\underline{\phi}}$ implies that $\underline{M}\,\mathcal{R}\,\underline{N} \implies (\underline{M} \models \underline{\underline{\phi}}) = (\underline{N} \models \underline{\underline{\phi}})$. We can reuse the proof of the previous lemma. There is only one extra case in the induction of formulas:

12. $\neg(\underline{\underline{\phi}}) \in \mathit{Form}(\underline{\underline{\mathbf{E}}})$, if $\underline{\underline{P}}\,\mathcal{R}_{\underline{\underline{\mathbf{E}}}}\,\underline{\underline{R}}$ then $\underline{\underline{R}}\,\mathcal{R}_{\underline{\underline{\mathbf{E}}}}\,\underline{\underline{P}}$ so by induction hypothesis $(\underline{\underline{R}} \models \underline{\underline{\phi}}) \trianglelefteq (\underline{\underline{P}} \models \underline{\underline{\phi}})$. Hence $(\underline{\underline{P}} \models \neg(\underline{\underline{\phi}})) = \neg(\underline{\underline{P}} \models \underline{\underline{\phi}}) \trianglelefteq \neg(\underline{\underline{R}} \models \underline{\underline{\phi}}) = (\underline{\underline{R}} \models \neg(\underline{\underline{\phi}}))$.

□

The following theorem is a consequence of the previous four lemmas.

Theorem 6.4.8 (The Generalised Coincidence Theorem)**.** *If all modalities $q \in \mathcal{Q}$ are leaf-monotone, then the positive behavioural preorder $\sqsubseteq^+$ is $\mathcal{Q}$-similarity, and the general behavioural equivalence $\equiv$ is $\mathcal{Q}$-bisimilarity.*

6.4.1 $\mathcal{Q}$-relator properties

In order to prove the Generalised Compatibility Theorem (Theorem 6.3.15), we are going to use the proof by Howe's method from Sections 4.4 and 4.5. This proof can be directly applied, as long as the $\mathcal{Q}$-relator satisfies the relator properties as given in Section 4.3. So we will generalise the lemmas of that section from the Boolean setting to the quantitative setting.

Lemma 6.4.9. *Suppose all modalities $q \in \mathcal{Q}$ are tree-monotone (implied by Scott tree-continuity). If $t\,\mathcal{Q}(\mathcal{R})\,r$, $t' \leq t$, and $r \leq r'$, then $t'\,\mathcal{Q}(\mathcal{R})\,r'$.*

Proof. Suppose $t\,\mathcal{Q}(\mathcal{R})\,r$, $t' \leq t$, and $r \leq r'$. Let $q \in \mathcal{Q}$, and $h : X \to \mathbb{A}$, then $h^*(t') \leq h^*(t)$. So by tree-monotonicity, $[\![q]\!](h^*(t')) \trianglelefteq [\![q]\!](h^*(t))$, which since $t\,\mathcal{Q}(\mathcal{R})\,r$ holds is below $[\![q]\!]((\mathcal{R}^\uparrow[h])^*(r))$. Since $(\mathcal{R}^\uparrow[h])^*(r) \leq (\mathcal{R}^\uparrow[h])^*(r')$ we can conclude, again using tree-monotonicity, that $[\![q]\!](h^*(t')) \trianglelefteq [\![q]\!]((\mathcal{R}^\uparrow[h])^*(r'))$. □

Lemma 6.4.10. *Given Scott tree-continuity for all modalities, then for $t_0 \leq t_1 \leq \ldots$ and $r_0 \leq r_1 \leq \ldots$: $\quad \forall n.(t_n\,\mathcal{Q}(\mathcal{R})\,r_n) \quad\implies\quad (\sqcup_n t_n)\,\mathcal{Q}(\mathcal{R})\,(\sqcup_n r_n)$.*

Proof. Assume for all $n \in \mathbb{N}$, $t_n\,\mathcal{Q}(\mathcal{R})\,r_n$. Let r be $\sqcup_n r_n$, then by Lemma 6.4.9 we can see that $\forall n \in \mathbb{N}.t_n\,\mathcal{Q}(\mathcal{R})\,r$ since $\forall n \in \mathbb{N}.(r_n \leq r)$. Take an arbitrary $h : X \to \mathbb{A}$ and $q \in \mathcal{Q}$. By Scott tree-continuity we have $[\![q]\!](h^*(\sqcup_n t_n)) = [\![q]\!](\sqcup_n h^*(t_n)) \trianglelefteq \bigvee_n [\![q]\!](h^*(t_n))$. For

any n, since $t_n \mathcal{Q}(\mathcal{R}) r$ we know that $[\![q]\!](\mathcal{R}^\uparrow[h])^*(r) \trianglerighteq [\![q]\!](h^*(t_n))$. Hence $(\mathcal{R}^\uparrow[h])^*(r)$ is an upper bound to the sequence $\{[\![q]\!](h^*(t_n))\}_{n\in\mathbb{N}}$, so $[\![q]\!](\mathcal{R}^\uparrow[h])^*(r) \trianglerighteq \bigvee_n [\![q]\!](h^*(t_n))$. We can conclude that $[\![q]\!](h^*(\sqcup_n t_n)) \trianglelefteq [\![q]\!]((\mathcal{R}^\uparrow[h])^*(r))$, for arbitrary $h : X \to \mathbb{A}$ and $q \in \mathcal{Q}$, hence $(\sqcup_n t_n)\mathcal{Q}(\mathcal{R})r$. □

Lemma 6.4.11. *Assume all modalities from $\mathcal{Q}$ are leaf-monotone. Then given two relations $\mathcal{R} \subseteq X \times Y$, $\mathcal{S} \subseteq Z \times W$ and functions $f : X \to Z$, $g : Y \to W$ such that $\forall (x, y) \in X \times Y.\ x\,\mathcal{R}\,y \Rightarrow f(x)\,\mathcal{S}\,g(y)$. Then it holds that $t\,\mathcal{Q}(\mathcal{R})\,r \Rightarrow f^*(t)\,\mathcal{Q}(\mathcal{S})\,g^*(r)$.*

Proof. Assume $\mathcal{R}, \mathcal{S}, f$ and g as above, and let t, r be such that $t\,\mathcal{Q}(\mathcal{R})\,r$. Take $h : Z \to \mathbb{A}$, then $h \circ f : X \to \mathbb{A}$, so by $t\,\mathcal{Q}(\mathcal{R})\,r$ it holds that for all $q \in \mathcal{Q}$, $(f^*(t) \Leftarrow q(h)) = [\![q]\!](h^*(f^*(t))) = [\![q]\!]((h \circ f)^*(t)) \trianglelefteq [\![q]\!]((\mathcal{R}^\uparrow[(h \circ f)])^*(r))$.

Now we have that: $(\mathcal{R}^\uparrow[(h \circ f)])(y) = \sup\{h(f(x)) \mid x \in X, x\,\mathcal{R}\,y\} \trianglelefteq \sup\{h(f(x)) \mid x \in X, f(x)\,\mathcal{S}\,g(y)\} \trianglelefteq \sup\{h(z) \mid z \in Z, z\,\mathcal{S}\,g(y)\} = (\mathcal{S}^\uparrow[h])(g(y))$. We use leaf-monotonicity of q to conclude that $(f^*(t) \Leftarrow q(h)) \trianglelefteq [\![q]\!]((\mathcal{R}^\uparrow[(h \circ f)])^*(r)) \trianglelefteq [\![q]\!]((\mathcal{S}^\uparrow[h])^*(g^*(r))) = (g^*(r) \Leftarrow q(\mathcal{S}^\uparrow[h]))$. □

Lemma 6.4.12. *Given a decomposable set of leaf-monotone modalities $\mathcal{Q}$, then:*

1. *For all $\mathcal{R} \subseteq X \times Y$, $x \in X$, $y \in Y$: $x\,\mathcal{R}\,y \implies \langle x \rangle\,\mathcal{Q}(\mathcal{R})\,\langle y \rangle$.*
2. *For all $\mathcal{R} \subseteq X \times Y$, $a \in T(T(X))$, $b \in T(T(Y))$: $a\,\mathcal{Q}(\mathcal{Q}(\mathcal{R}))\,b \Rightarrow \mu a\,\mathcal{Q}(\mathcal{R})\,\mu b$.*

Proof. The first property follows from the fact that if $x\,\mathcal{R}\,y$ then $(\mathcal{R}^\uparrow[h])(y) \trianglerighteq h(x)$. So for any $q \in \mathcal{Q}$, by leaf-monotonicity, $[\![q]\!](h^*(\langle x \rangle)) = [\![q]\!](\langle h(x) \rangle) \trianglelefteq [\![q]\!](\langle (\mathcal{R}^\uparrow[h])(y) \rangle) = [\![q]\!]((\mathcal{R}^\uparrow[h])^*(\langle y \rangle))$.

For the second property, assume $a\,\mathcal{Q}(\mathcal{Q}(\mathcal{R}))\,b$ and let $h : X \to \mathbb{A}$ and $q \in \mathcal{Q}$. We want to prove that $[\![q]\!](h^*(\mu a)) \trianglelefteq [\![q]\!]((\mathcal{Q}(\mathcal{R})^\uparrow[h])^*(\mu b))$. Note that $h^*(\mu a) = \mu(h^{**}(a))$ and $(\mathcal{Q}(\mathcal{R})^\uparrow[h])^*(\mu b) = \mu((\mathcal{Q}(\mathcal{R})^\uparrow[h])^{**}(b))$, so by decomposability it is sufficient to prove that $h^{**}(a) \preccurlyeq (\mathcal{Q}(\mathcal{R})^\uparrow[h])^{**}(b)$.

Let $H \in QBS(T\mathbb{A})$ and $q' \in \mathcal{Q}$, we want to prove that $(h^{**}(a) \Leftarrow q(H)) \trianglelefteq ((\mathcal{Q}(\mathcal{R})^\uparrow[h])^{**}(b) \Leftarrow q(H))$. Since $H^*(h^{**}(a)) = (H \circ h^*)^*(a)$, $H \circ h^* : T(X) \to \mathbb{A}$, and $a\,\mathcal{Q}(\mathcal{Q}(\mathcal{R}))\,b$, we know that $(h^{**}(a) \Leftarrow q(H)) = (a \Leftarrow q(H \circ h^*)) \trianglelefteq (b \Leftarrow q(\mathcal{Q}(\mathcal{R})^\uparrow[(H \circ h^*)]))$.

Now, it holds that $(\mathcal{Q}(\mathcal{R})^\uparrow[(H \circ h^*)])(r) = \sup\{H(h^*(t)) \mid t \in T(X), t\,\mathcal{Q}(\mathcal{R})\,r\} \trianglelefteq \sup\{H(h^*(t)) \mid t \in T(X), h^*(t) \preccurlyeq (\mathcal{R}^\uparrow[h])^*(r)\} \trianglelefteq$
$\sup\{H(t) \mid t \in T(\mathbb{A}), t \preccurlyeq (\mathcal{R}^\uparrow[h])^*(r)\} \trianglelefteq H((\mathcal{R}^\uparrow[h])^*(r))$, using Lemma 6.4.3 and the fact that $H \in QBS(T\mathbb{A})$. So by leaf-monotonicity, and Lemma 6.3.16, $(b \Leftarrow q(\mathcal{Q}(\mathcal{R})^\uparrow[(H \circ h^*)])) \trianglelefteq (b \Leftarrow q(H \circ (\mathcal{R}^\uparrow[h])^*)) = ((\mathcal{Q}(\mathcal{R})^\uparrow[h])^{**}(b) \Leftarrow q(H))$.

We have proven that $h^{**}(a) \preccurlyeq (\mathcal{Q}(\mathcal{R})^\uparrow[h])^{**}(b)$, so by decomposability it holds that $\mu(h^{**}(a)) \preccurlyeq \mu((\mathcal{Q}(\mathcal{R})^\uparrow[h])^{**}(b))$. We get the desired result by unfolding the definition of $\preccurlyeq$ using q and the identity function $id : \mathbb{A} \to_{\trianglelefteq} \mathbb{A}$. □

Note that $\mathcal{Q}(\trianglelefteq) = \preccurlyeq$ and $\mathcal{Q}(\trianglelefteq) = \preccurlyeq$. So from Lemma 6.4.12 we can conclude that:

Corollary 6.4.13. *If all modalities $q \in \mathcal{Q}$ are leaf-monotone, then $\mathcal{Q}$ is decomposable if and only if $\forall \mathcal{R} \subseteq X \times Y, \forall r, r' \in TT\mathbb{A}$, $r\,\mathcal{Q}(\mathcal{Q}(\mathcal{R}))\,r' \Rightarrow \mu r\,\mathcal{Q}(\mathcal{R})\,\mu r'$.*

We have all the sufficient properties for the relator $\mathcal{Q}(-)$ to prove the following theorem, using the same proofs by Howe's method as for Theorems 4.5.2 and 4.5.5

Theorem 6.4.14. *The open extensions of applicative $\mathcal{Q}$-similarity and applicative $\mathcal{Q}$-bisimilarity are compatible if $\mathcal{Q}$ is a decomposable set of Scott tree-continuous and leaf-monotone modalities.*

6.5 Variations

We end this chapter with some thoughts on possible variations of the quantitative logic.

6.5.1 The Boolean logic revisited

The quantitative logic defined in this chapter is a generalisation of the Boolean logic defined in Chapter 3. In other words, any logic for programs with effects defined in that chapter can be recast as a quantitative logic in the sense of this chapter. As such, Theorems 4.5.2 and 4.5.5 can be seen as instances of Theorem 6.4.14 given above.

Note first that the Booleans $\mathbb{B}$ form a complete lattice with involution. Let Σ be a signature and $\mathcal{O}$ a set of modalities on that signature. The main difference between the Boolean and quantitative formulations of the logics is that a Boolean modality o is given by a subset $[\![o]\!] \subseteq T\mathbf{1}$, not by function from $T\mathbb{B}$ to $\mathbb{B}$, or subset of $T\mathbb{B}$. The set $T\mathbb{B}$ distinguishes between divergence $\bot$ and formula dissatisfaction $\boldsymbol{F}$. In our translation from Boolean modalities to quantitative modalities, we must cope with this distinction.

For $o \in \mathcal{O}$, we define q_o as the quantitative modality, whose denotation gives a function $[\![q_o]\!] : T\mathbb{B} \to \mathbb{B}$, such that:

$$[\![q_o]\!](t) = \boldsymbol{T} \quad \Longleftrightarrow \quad t \in o(\{\boldsymbol{T}\}).$$

We take $\mathcal{Q}_{\mathcal{O}} := \{q_o \mid o \in \mathcal{O}\}$ as our set of quantitative modalities over Σ using the truth space $\mathbb{B}$. We check how the conditions for compatibility on $\mathcal{O}$ carry over to the generalised conditions on $\mathcal{Q}_{\mathcal{O}}$.

Lemma 6.5.1. *If o is leaf-upwards closed, then q_o is leaf-monotone.*

Proof. For $t, t' \in T\mathbb{B}$ such that $t \trianglelefteq_{T\mathbb{A}} t'$, it holds that $t[\in \{\boldsymbol{T}\}] \leq t'[\in \{\boldsymbol{T}\}] \leq t[\bot \mapsto \langle * \rangle] = t[\in \{\boldsymbol{T}\}][\bot \mapsto \langle * \rangle]$. So since o is leaf upwards closed, if $t \in o(\{\boldsymbol{T}\})$, then $t' \in o(\{\boldsymbol{T}\})$. □

Lemma 6.5.2. *If o is Scott open, then q_o is Scott tree-continuous.*

Proof. For $t_i \in T\mathbb{B}$, $t_0 \leq t_1 \leq t_2 \leq \ldots$, it holds that $t_0[\in \{\boldsymbol{T}\}] \leq t_1[\in \{\boldsymbol{T}\}] \leq \ldots$.
If $[\![q_o]\!](\bigsqcup_n t_n) = \boldsymbol{T}$, then $(\bigsqcup_n t_n)[\in \{\boldsymbol{T}\}] \in [\![o]\!]$ hence $\bigsqcup_n (t_n[\in \{\boldsymbol{T}\}]) \in [\![o]\!]$, so since o is Scott open there is an $n \in \mathbb{N}$ such that $t_n[\in \{\boldsymbol{T}\}] \in [\![o]\!]$ and hence $[\![q_o]\!](t_n) = \boldsymbol{T}$. We can conclude that $\bigvee_m [\![q_o]\!](t_n) = \boldsymbol{T}$, hence $[\![q_o]\!](\bigsqcup_n t_n) \trianglelefteq \bigvee_m [\![q_o]\!](t_n)$. The other direction follows by leaf-monotonicity of q_o from the previous lemma. □

Assume now that all $o \in \mathcal{O}$ are leaf upwards closed. For $h : \mathbb{B} \to_{\trianglelefteq} \mathbb{B}$ monotone, $t \Leftarrow q_o(h)$ holds if and only if $t \in o(h^{-1}\{\boldsymbol{T}\})$. Considering that $h^{-1}\{\boldsymbol{T}\}$ can only be $\{\boldsymbol{T}\}$, $\{\boldsymbol{F}, \boldsymbol{T}\}$, or $\emptyset$ we can see that:

$$t \preccurlyeq_{\mathcal{Q}_\mathcal{O}} t' \quad \iff \quad t[\in \{\boldsymbol{T}\}] \preccurlyeq_\mathcal{O} t'[\in \{\boldsymbol{T}\}] \ \wedge \ t[\in \{\boldsymbol{F}, \boldsymbol{T}\}] \preccurlyeq_\mathcal{O} t'[\in \{\boldsymbol{F}, \boldsymbol{T}\}]$$

It seems difficult to compare the Boolean notion of decomposability with the quantitative notion of decomposability via their definition using the $\preccurlyeq$ and $\preccurlyeq$ relations. However, using their equivalent formulations in terms of relators from Corollary 4.3.3 and 6.4.13, we can see that the two notions of decomposability coincide.

Lemma 6.5.3. *The relators $\mathcal{O}(-)$ and $\mathcal{Q}_\mathcal{O}(-)$ are identical.*

Proof. Let $\mathcal{R} \subseteq X \times Y$, $t \in TX$ and $t' \in TY$.

Assume $t \,\mathcal{O}(\mathcal{R})\, t'$ and for some $h : X \to \mathbb{B}$ and $q_o \in \mathcal{Q}_\mathcal{O}$ it holds that $(t \Leftarrow q_o(h)) = \boldsymbol{T}$. Then $t \in o(h^{-1}(\{\boldsymbol{T}\}))$, so $t' \in o(\mathcal{R}^\uparrow[(h^{-1}(\{\boldsymbol{T}\}))])$. Now, $y \in (\mathcal{R}^\uparrow[h^{-1}(\{\boldsymbol{T}\})])$ holds if and only if there is an $x \in X$ such that $h(x) = \boldsymbol{T}$ and $x \,\mathcal{R}\, y$, which holds precisely when $(\mathcal{R}^\uparrow[h])(y) = \boldsymbol{T}$. So $t' \in o((\mathcal{R}^\uparrow[h])^{-1}(\{\boldsymbol{T}\}))$, hence $(t' \Leftarrow q_o(\mathcal{R}^\uparrow[h])) = \boldsymbol{T}$.

Assume $t \,\mathcal{Q}_\mathcal{O}(\mathcal{R})\, t'$ and for some $A \subseteq X$ and $o \in \mathcal{O}$ it holds that $t \in o(A)$. Let $h : X \to \mathbb{B}$ be such that $h(x) = \boldsymbol{T}$ if and only if $x \in A$. Then $t \in o(h^{-1}(\{\boldsymbol{T}\}))$ and hence $(t \Leftarrow q_o(h)) = \boldsymbol{T}$. So $(t' \Leftarrow q_o(\mathcal{R}^\uparrow[h])) = \boldsymbol{T}$ which, by the same reasoning as above, means $t' \in o(\mathcal{R}^\uparrow[(h^{-1}(\{\boldsymbol{T}\}))])$ and hence $t' \in o(\mathcal{R}^\uparrow[A])$. □

Using the equivalent formulations of decomposability using the relators, given in Corollary 4.3.3 and 6.4.13, we can derive the following corollary.

Corollary 6.5.4. *If all $o \in \mathcal{O}$ are leaf-upwards closed, then $\mathcal{O}$ is decomposable if and only if $\mathcal{Q}_\mathcal{O}$ is decomposable.*

6.5.2 Infinitary vs finitary quantitative formula connectives

In Sections 5.1 and 5.2 we have seen different cases in which the size of the cardinality of Boolean connectives for the Boolean logics could be reduced without altering the induced logical preorder. In this subsection, we will attempt to prove similar results for the quantitative logic, trying to reduce the size of the cardinalities of suprema and infima without changing the induced behavioural preorder.

Section 5.1 for example shows us how we may remove disjunctions, conjunctions and negations for computation formulas without changing the resulting equivalences. We adapt this result to the quantitative logic, where moreover, we also remove constant formulas and threshold formulas from the value logic. We adapt Definition 5.1.2.

Definition 6.5.5. For each quantitative logic $\mathcal{L}$, we define the logic $\mathcal{L}^*$ as the largest fragment of $\mathcal{L}$ such that all computation formulas are basic formulas.

In other words, $\mathcal{L}^*$ does not have any suprema, infima, involutions, constant formulas, or threshold formulas at computation types.

Proposition 6.5.6. *The fragment $\mathcal{U}^*$ of $\mathcal{U}$ induces the same logical equivalence as $\mathcal{U}$. Similarly, the fragment $\mathcal{U}^{+*}$ of $\mathcal{U}^+$ induces the same logical preorder as $\mathcal{U}$.*

Proof. The proof is similar to the proof of the Boolean case, using the induction on formulas from Lemma 5.1.3. The only alteration to that proof is the addition of new quantitative formula constructors. Such cases can be dealt with by noting that formula constructors like $\langle - \rangle$ and $(V \mapsto (-))$ commute with formula constructor $(-)_{\geq a}$, and formulas like $\langle \kappa_a \rangle$ and $(V \mapsto \kappa_a)$ are equivalent to κ_a (of the appropriate type). □

The above result can alternatively be proven by establishing that both logical equivalences $\equiv$ and $\equiv_{\mathcal{U}^*}$ are equal to $\mathcal{Q}$-bisimilarity. This method is followed in the proof of the following result, which looks at what may be a more interesting reduction of the logic, involving the use of Scott leaf-continuity from Definition 6.3.6. Recall that all of the given examples of quantitative modalities satisfy this property.

We borrow the notation $(\mathcal{Q}, a, b, c)$ for logics from Section 5.2 as defined in Definition 5.2.1, but adapt it to the quantitative logic. In particular, $a \in \{\bigvee, \vee, \bot\}$ denotes the possible sizes for suprema, $b \in \{\bigwedge, \wedge, \top\}$ the possible sizes for infima, and $c \in \{\neg, +\}$ denotes whether or not involutions are included in the logic. In particular, $(\mathcal{Q}, \bigvee, \bigwedge, \neg) = \mathcal{U}^*$ and $(\mathcal{Q}, \bigvee, \bigwedge, +) = \mathcal{U}^{+*}$.

Proposition 6.5.7. *If all $q \in \mathcal{Q}$ are Scott leaf-continuous, then $(\mathcal{Q}, \vee, \bigwedge, \neg)$ induces the same logical equivalence as $\mathcal{U}$, and $(\mathcal{Q}, \vee, \bigwedge, +)$ induces the same logical preorder as $\mathcal{U}^+$.*

Proof. Instead of doing an induction on the structure of formulas, we instead just adapt the proof of Theorem 6.4.8 to see that the resulting logical preorder still gives $\mathcal{Q}$-bisimilarity (and $\mathcal{Q}$-similarity). Let $\mathcal{L}$ be either $(\mathcal{Q}, \vee, \bigwedge, \neg)$ or $(\mathcal{Q}, \vee, \bigwedge, +)$. The proofs for the two logics are similar.

Most of the proof of Theorem 6.4.8 goes through for $\mathcal{L}$. The only problem is case 6 of the proof of Lemma 6.4.4, where we need to show that $\underline{M} \sqsubseteq_{\mathcal{L}} \underline{N}$ implies $|\underline{M}|\mathcal{Q}(\sqsubseteq_{\mathcal{L}})|\underline{N}|$. The proof makes use of Lemma 6.3.17, which found for each quantitative predicate $D : \mathit{Terms}(\mathbf{A}) \to \mathbb{A}$ a formula $\dot{\phi}_D \in \mathit{Form}(\mathbf{A})$ such that for all $V \in \mathit{Terms}(\mathbf{A})$, $(V \models \dot{\phi}_D) = (\sqsubseteq^{\uparrow}[D])(V)$. This formula was defined as a countable supremum $\phi_D := \bigvee\{\kappa_{D(V)} \wedge \chi_V \mid V \in \mathit{Terms}(\mathbf{A})\}$. Note that Lemma 6.3.3 still works in $\mathcal{L}$, so characteristic formulas χ_V exist in the logic. However, $\dot{\phi}_D \notin \mathcal{L}$.

We can however do an approximation of $\dot{\phi}_D$, using an enumeration on terms, specifying an injective function $\#(-) : \mathit{Terms}(\mathbf{A}) \to \mathbb{N}$. For a quantitative predicate $D : \mathit{Terms}(\mathbf{A}) \to \mathbb{A}$ we take a sequence of formulas $\phi_D^n \in \mathit{Form}(\mathbf{A})$, defined in the following way:

$$\phi_D^n \;:=\; \bigvee\{\kappa_{D(V)} \wedge \chi_V \;\mid\; V \in \mathit{Terms}(\mathbf{A}), \#V < n\},$$

a finite supremum over the first n terms of type $\mathbf{A}$. So $(\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])(V) = \lim_{n\to\infty} V \models \phi_D^n$.

We use this to adapt case 6 of the proof of Lemma 6.4.4. If $\underline{M} \sqsubseteq_{\mathcal{L}} \underline{N} : \mathbf{F}\,\mathbf{A}$, then for any $q \in \mathcal{Q}$ and $D : \mathit{Terms}(\mathbf{A}) \to \mathbb{A}$ it holds that $\forall n, [\![q]\!](|\underline{M}|[\models \phi_D^n]) \trianglelefteq [\![q]\!](|\underline{N}|[\models \phi_D^n])$,

which by Scott leaf-continuity means that $|\underline{M}| \Lhd q(\sqsubseteq_{\mathcal{L}}^{\uparrow}[D]) \trianglelefteq |\underline{N}| \Lhd q(\sqsubseteq_{\mathcal{L}}^{\uparrow}[D])$. With leaf-monotonicity we have $|\underline{M}| \Lhd q(D) \trianglelefteq |\underline{M}| \Lhd q(\sqsubseteq_{\mathcal{L}}^{\uparrow}[D]))$, so we have checked the relevant case for proving that $\sqsubseteq_{\mathcal{L}}$ is a $\mathcal{Q}$-simulation.

We conclude that $\sqsubseteq_{\mathcal{L}}$ is $\mathcal{Q}$-bisimilarity (or similarity), and hence equal to $\equiv$ (or $\sqsubseteq^{+}$). □

When we talk about Scott leaf-continuous modalities, we consider limits of increasing sequence of trees. We could instead consider *lower leaf-continuous* modalities, which considers descending sequences.

Definition 6.5.8. A modality q is *lower leaf-continuous* if for any descending chain of trees in the leaf-order $t_0 \trianglerighteq_{T\mathbb{A}} t_1 \trianglerighteq_{T\mathbb{A}} t_2 \trianglerighteq_{T\mathbb{A}} \dots$, it holds that $[\![q]\!](\sqcap_n^{T(\trianglelefteq)} t_n) = \bigwedge_n [\![q]\!](t_n)$.

The examples of probability, probability with score, global store, probability with global store, and timer all have quantitative modalities which are lower leaf-continuous. Moreover, any combination of the effects listed above with error according to Subsection 6.2.7 also yield lower leaf-continuous modalities. However, modalities resulting from combinations with angelic nondeterminism, like $\mathsf{E}_{\Diamond}$, are not lower leaf-continuous.

Proposition 6.5.9. *If all modalities are leaf-monotone and lower leaf-continuous, then* $(\mathcal{Q}, \bigvee, \wedge, \neg)$ *induces the same logical equivalence as* $\mathcal{U}$*, and* $(\mathcal{Q}, \bigvee, \wedge, +)$ *induces the same logical preorder as* $\mathcal{U}^{+}$*.*

Note that the first condition is leaf-monotonicity, not leaf-continuity.

Proof. Let $\mathcal{L}$ be the logic $(\mathcal{Q}, \bigvee, \wedge, \neg)$ (or the logic $(\mathcal{Q}, \bigvee, \wedge, +)$). This proof follows the same pattern as the proof of Proposition 6.5.7. However, here we have the problem that characteristic formulas χ_V as defined in Lemma 6.3.3 use countable infima. As such, $\phi_D := \bigvee\{\kappa_{D(V)} \wedge \chi_V \mid V \in \mathit{Terms}(\mathbf{A})\}$ is not in $\mathcal{L}$, and we need to approximate these formulas in a different way, this time from above.

First we define for each V a formula χ_V^n, the finite approximation of the characteristic formula from Lemma 6.3.3 given by the finite conjunction $\chi_V^n := \bigwedge\{\psi^W_{\trianglerighteq V \models \psi^W} \mid V \not\sqsubseteq_{\mathcal{L}} W, \#W \leq n\}$, where for each W such that $V \not\sqsubseteq_{\mathcal{L}} W$, we choose a formula ψ^W such that $(V \models \psi^W) \not\trianglelefteq (W \models \psi^W)$. We get that:

$$(W \models \chi_V^n) = \begin{cases} \boldsymbol{T} & \text{if } V \sqsubseteq_{\mathcal{L}} W \\ \boldsymbol{F} & \text{if } V \not\sqsubseteq_{\mathcal{L}} W \text{ and } \#W \leq n \\ ? & \text{otherwise.} \end{cases}$$

The result is not specified when $V \not\sqsubseteq_{\mathcal{L}} W$ and $\#W > n$. However, we do know that for any $n \in \mathbb{N}$ it holds that $(W \models \chi_V^{n+1}) \trianglelefteq (W \models \chi_V^n)$. So $V \sqsubseteq_{\mathcal{L}} W \iff \forall n.(W \models \chi_V^n) = \boldsymbol{T} \iff (W \models \chi_V^{\#W}) = \boldsymbol{T}$. For a quantitative predicate $D : \mathit{Terms}(\mathbf{A}) \to \mathbb{A}$, let $\chi_D^n := \bigvee\{\kappa_{D(W)} \wedge \chi_W^n \mid W \in \mathit{Terms}(\mathbf{A})\}$. From the formulation, we know that for any $n \in \mathbb{N}$, $(V \models \chi_D^{n+1}) \trianglelefteq (V \models \chi_D^n)$. We will prove that $\bigwedge_n \chi_D^n = (\sqsubseteq_{\mathcal{L}}^{\uparrow}[D])$.

Fix some value term V, then for any W and $m \geq \#V$ we have $(V \models \chi^m_W) = \boldsymbol{T} \iff W \sqsubseteq_{\mathcal{L}} V$. We can derive that:

$$(\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])(V) = \sup\{D(W) \mid W \sqsubseteq_{\mathcal{L}} V\} = \sup\{D(W) \mid (V \models \chi^m_W) = \boldsymbol{T}\} = (V \models \chi^m_D)$$

Since for $k \leq n < m$ we still have that $(V \models \chi^m_D) \trianglelefteq (V \models \chi^k_D)$, we can derive that $(\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])(V) = (V \models \bigwedge_n \chi^n_D)$.

We conclude the proof by observing that for a lower leaf-continuous modality q it holds that:

$$\begin{array}{rrcll} \forall n \in \mathbb{N}. & [\![q]\!](|\underline{M}|[\models \phi^n_D]) & \trianglelefteq & [\![q]\!](|\underline{N}|[\models \phi^n_D]) & \\ \Leftrightarrow & ([\![q]\!](|\underline{M}|[\models \bigwedge_n \phi^n_D]) & \trianglelefteq & [\![q]\!](|\underline{N}|[\models \bigwedge_n \phi^n_D])) & \text{by lower leaf-continuity} \\ \Leftrightarrow & ([\![q]\!](|\underline{M}|[\models (\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])]) & \trianglelefteq & [\![q]\!](|\underline{N}|[\models (\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])])) & \text{by the above equality.} \end{array}$$

so we can adapt the proof of Proposition 6.5.7. □

Last but not least, we can combine the proofs of the above to Proposition.

Proposition 6.5.10. *If all modalities are Scott leaf-continuous and lower leaf-continuous, then* $(\mathcal{Q}, \vee, \wedge, \neg)$ *induces the same logical equivalence as* $\mathcal{U}$*, and* $(\mathcal{Q}, \vee, \wedge, +)$ *induces the same logical preorder as* $\mathcal{U}^+$*.*

Proof. Let $\mathcal{L} = (\mathcal{Q}, \vee, \wedge, \neg)$ (or $\mathcal{L} = (\mathcal{Q}, \vee, \wedge, +)$), and following the proof of Proposition 6.5.7 we need to prove that $\sqsubseteq_{\mathcal{L}}$ is an applicative $\mathcal{Q}$-simulation, specifically by proving the sixth condition for applicative $\mathcal{Q}$-simulations.

This can be done by combining the methods of the proofs from the previous two propositions. In both proofs we defined for each quantitative predicate $D : \mathit{Terms}(\mathbf{A}) \to \mathbb{A}$ a sequence of approximations of $(\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])$ defined with formulas from $\mathcal{L}$. In the general logic, $(\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])$ can be defined with the formula $\bigvee\{\kappa_{D(V)} \wedge \chi_V \mid V \in \mathit{Terms}(\mathbf{A})\}$, with $\chi_V := \bigwedge\{\psi^W_{\trianglerighteq(W \models \psi^W)} \mid V \not\sqsubseteq W\}$, where for each W such that $V \not\sqsubseteq W$ we chose a formula ψ^W such that $(V \models \psi^V) \not\trianglelefteq (V \models \psi^W)$. However, this formula both has an infinite supremum and infinite infima, and hence is not in $\mathcal{L}$. In the proof of Proposition 6.5.7, we did an approximation of the outermost supremum, and in the proof of Proposition 6.5.9, we did an approximation of the infima used by the characteristic formulas χ_V. In this proof, we need to combine these two approximations.

Remember that we have an injective function $\# : \mathit{Terms}(\mathbf{A}) \to \mathbb{N}$. Firstly, we define for each V a formula $\chi^n_V \in \mathcal{L}$, the finite approximation of the characteristic formula as defined in the proof of Proposition 6.5.9 using the function $\#$ (but taking the formulas ψ^W from $\mathcal{L}$ instead). So $V \sqsubseteq_{\mathcal{L}} W \iff \forall n.(W \models \chi^n_V) = \boldsymbol{T} \iff (W \models \chi^{\#W}_V) = \boldsymbol{T}$. Hence, it holds that $(W \models \bigwedge_n \chi^n_V) = \boldsymbol{T} \Leftrightarrow (V \sqsubseteq_{\mathcal{L}} W)$.

Secondly, take some quantitative predicate $D : \mathit{Terms}(\mathbf{A}) \to \mathbb{A}$, and for each $n \in \mathbb{N}$ we define the approximation $D^n : \mathit{Terms}(\mathbf{A}) \to \mathbb{A}$ sending V to $D(V)$ if $\#(V) \leq n$, otherwise to $\boldsymbol{F}$. For each $n, m \in \mathbb{N}$ we define the formula $\phi^{n,m}_D \in \mathit{Form}(\mathbf{A})_{\mathcal{L}}$ as follows:

$$\phi^{n,m}_D \quad := \quad \bigvee\{\kappa_{D(V)} \wedge \chi^n_V \;\mid\; V \in \mathit{Terms}(\mathbf{A}),\ \#V \leq m\} \;\in\; \mathcal{L}.$$

Note that this is a finite suprema over formulas from $\mathcal{L}$, hence it is itself in $\mathcal{L}$.

Let $V \in \mathit{Terms}(\mathbf{A})$ such that $\#V \leq m$. Then $(V \models \bigwedge_n \phi_D^{n,m}) =$
$\bigwedge_n \bigvee\{(V \models (\kappa_{D(W)} \wedge \chi_W^n)) \mid W \in \mathit{Terms}(\mathbf{A}),\ \#W \leq m\} =$
$\bigwedge_n \bigvee\{D(W) \mid W \in \mathit{Terms}(\mathbf{A}),\ \#W \leq m,\ (V \models \chi_W^n) = \boldsymbol{T}\} =$
$\bigwedge_{n \leq \#V} \bigvee\{D(W) \mid W \in \mathit{Terms}(\mathbf{A}),\ \#W \leq m,\ (V \models \chi_W^n) = \boldsymbol{T}\} =$
$\bigvee\{D(W) \mid W \in \mathit{Terms}(\mathbf{A}),\ \#W \leq m,\ (V \models \chi_W^{\#V}) = \boldsymbol{T}\} =$
$\bigvee\{D(W) \mid W \in \mathit{Terms}(\mathbf{A}),\ \#W \leq m,\ W \sqsubseteq_{\mathcal{L}} V\} =$
$\bigvee\{D^m(W) \mid W \in \mathit{Terms}(\mathbf{A}),\ W \sqsubseteq_{\mathcal{L}} V\}$.

Hence $(V \models \bigwedge_n \phi_D^{n,m}) = (\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D^m])(V)$, so we can derive that $(V \models \bigvee_m \bigwedge_n \phi_D^{n,m}) = (\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])(V)$. Note that for each $m \in \mathbb{N}$, $\{\phi_D^{n,m}\}_{n \in \mathbb{N}}$ is a decreasing sequence of formulas (their satisfaction never increases), and that $\{\bigwedge_n \phi_D^{n,m}\}_{m \in \mathbb{N}}$ is an increasing sequence of formulas. We finish the proof by observing that, for a Scott leaf-continuous and lower leaf-continuous modality q it holds that:

$$
\begin{array}{rcll}
\forall n, m,\ [\![q]\!](|\underline{M}|[\models \phi_D^{n,m}]) & \trianglelefteq & [\![q]\!](|\underline{N}|[\models \phi_D^{n,m}]) & \\
\Leftrightarrow \forall m,\ [\![q]\!](|\underline{M}|[\models \bigwedge_n \phi_D^{n,m}]) & \trianglelefteq & [\![q]\!](|\underline{N}|[\models \bigwedge_n \phi_D^{n,m}]) & \text{by Scott leaf-continuity} \\
\Leftrightarrow [\![q]\!](|\underline{M}|[\models \bigvee_m \bigwedge_n \phi_D^{n,m}]) & \trianglelefteq & [\![q]\!](|\underline{N}|[\models \bigvee_m \bigwedge_n \phi_D^{n,m}]) & \text{by lower leaf-continuity} \\
\Leftrightarrow [\![q]\!](|\underline{M}|[\models (\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])]) & \trianglelefteq & [\![q]\!](|\underline{N}|[\models (\sqsubseteq_{\mathcal{L}}{}^{\uparrow}[D])]) & \text{by the observed equality}
\end{array}
$$

so we can adapt the proof of Proposition 6.5.7. □

6.5.3 A short note on involutions

We end this chapter with a final remark. We required the inclusion of an involution for our truth space in our formulation of the general logic. However, upon closer inspection, it is not strictly necessary to include an involution. It is enough to add a simpler version of negation $\neg$ where $\neg \boldsymbol{F} = \boldsymbol{T}$ and for any $a \in \mathbb{A}$ where $a \neq \boldsymbol{F}$, $\neg a = \boldsymbol{F}$. So we can still formulate the general behavioural equivalence for examples of effects with truth spaces without involutions, like timer and input/output as given in Subsections 6.2.5 and 6.2.8.

As an example of how the proofs in this chapter can be adapted for this new notion of negation, we look at Lemma 6.3.2, which says that $\sqsubseteq$ is symmetric. We give an alternative proof to this Lemma, which only assumes that $\neg \boldsymbol{F} = \boldsymbol{T}$ and $\neg \boldsymbol{T} = \boldsymbol{F}$.

Alternative proof of Lemma 6.3.2. Suppose $\underline{P} \sqsubseteq \underline{R}$. If $(\underline{R} \models \underline{\phi}) \trianglerighteq a$ then $(\underline{R} \models \underline{\phi}_{\trianglerighteq a}) = \boldsymbol{T}$ and $(\underline{R} \models \neg(\underline{\phi}_{\trianglerighteq a})) = \boldsymbol{F}$. If moreover $(\underline{P} \models \underline{\phi}) \not\trianglerighteq a$, then $(\underline{P} \models \underline{\phi}_{\trianglerighteq a}) = \boldsymbol{F}$, so $(\underline{R} \models \underline{\phi}_{\trianglerighteq a}) = \boldsymbol{F}$ and $(\underline{R} \models \neg(\underline{\phi}_{\trianglerighteq a})) = \boldsymbol{T}$, so by $\underline{P} \sqsubseteq \underline{R}$ it holds that $\boldsymbol{F} = (\underline{R} \models \neg(\underline{\phi}_{\trianglerighteq a})) \trianglerighteq \boldsymbol{T}$. This gives us a contradiction, hence $(\underline{R} \models \underline{\phi}) \trianglerighteq a$ implies $(\underline{P} \models \underline{\phi}) \trianglerighteq a$. We conclude that $(\underline{R} \models \underline{\phi}) \trianglelefteq (\underline{P} \models \underline{\phi})$, and hence $\underline{R} \sqsubseteq \underline{P}$. □

7

Polymorphic and recursive types

The particular language we used to study behavioural equivalence for effectful programs was a call-by-push-value language with general recursion. It is however not difficult to change or add to the language, adapting the definition of the logic and behavioural equivalence appropriately.

In this chapter, we focus on extending the language in two significant ways, by adding new type constructors and terms for universal polymorphic types and recursive types. When extending the language in such a way, the logic needs to be extended as well in order to specify a behavioural equivalence on the types created by the new type constructors. This can be done in a natural way, reflecting the behaviour of terms of such types. For simplicity, we will only focus on extending the quantitative logic from Chapter 6, since it is more general then the Boolean logic from Chapter 3. The Boolean logics could however be extended in a similar way.

The proof for the Generalised Compatibility Theorem, Theorem 6.3.15, can be extended, establishing that the logic for the extended language induces a compatible program equivalence. To extend the proof of that theorem for such extended languages, it is necessary to break into some of the quite technical proofs from Chapter 4 in particular. We will try to do this in a modular way for universal polymorphic and recursive types. These examples of extensions are indicative of how the language can be extended in different ways.

So we extend the language in two main ways, adding universal polymorphic and recursive types, both of which are powerful constructions. With recursive types for instance, it is possible to express the untyped lambda calculus, which is another language for which effects and applicative bisimilarity have been widely studied (e.g., in [14]).

7.1 Adding type constructors

We first lay out the general scheme for extending the language and the logic $\mathcal{U}$, giving a roadmap for proving that the Generalised Compatibility Theorem still holds. This is done in four different steps, which require looking at three different chapters in this thesis.

(I) Firstly, we add the new type constructors, with their terms, typing rules, term reductions and possibly new stacks and stack reductions. This will expand the operational semantics from Chapter 2.

(II) We then define the appropriate basic logical formulas for the new types, for instance using the new term introduction rules to lift formulas. This extends the definition of the logic in Subsection 6.1.1.

(III) Next, we define clauses for the new types (new simulation rules) to the definition of applicative $\mathcal{Q}$-simulation (Definition 4.2.1). Moreover, we add appropriate cases to Lemmas 6.4.4 and 6.4.6, in order to prove the Generalised Coincidence Theorems, Theorems 6.4.8, for the extended language.

(IV) Lastly, we want to prove that applicative $\mathcal{Q}$-similarity and $\mathcal{Q}$-bisimilarity are compatible. To do this, we adapt the proof by Howe's method from Section 4.4 in three steps, in order to re-establish Proposition 4.4.19:

a) Prove that the Howe closure of an applicative $\mathcal{Q}$-simulation satisfies the new simulation rule from step (III).

b) If there is a new stack constructor formulated in step (I), add it as a frame constructor to Definition 4.4.10, and extend Definition 4.4.13 appropriately. Verify that lemmas related to frames still hold, mainly Lemma 4.4.15 as the other lemmas tend to follow directly.

c) For any new computation term not constructed by a frame, prove its case in the Lemma 4.4.18 in order to verify the Key Lemma.

This proves that the Howe closure of an applicative $\mathcal{Q}$-simulation is an applicative $\mathcal{Q}$-simulation, so we can conclude the proof by Howe's method as done in Section 4.5.

These four steps enable us to appropriately extend the language, and give it a compatible behavioural equivalence. We will now focus on the particular extensions of the language involving universal polymorphic types and recursive types. We start by adding type variables, which establishes a basis upon which we can define the new type constructors and terms.

7.1.1 Type variables

We introduce type variables $\alpha, \beta, \ldots$ for value types and $\underline{\alpha}, \underline{\beta}, \ldots$ for computation types, to enrich our language, and add type judgements for constructing them. *Type Contexts* are recursively defined as follows:

$$\Delta \;:=\; \varepsilon \;\mid\; \Delta, \alpha \;\mid\; \Delta, \underline{\beta}$$

We write $\Delta \vdash \mathbf{A}$ to say $\mathbf{A}$ is a value type in context Δ, and $\Delta \underline{\vdash} \underline{\mathbf{C}}$ to say $\underline{\mathbf{C}}$ is a computation type in context Δ. We have new judgements for types:

$$\frac{}{\Delta, \alpha, \Delta' \vdash \alpha} \qquad \frac{}{\Delta, \underline{\beta}, \Delta' \vdash \underline{\beta}} \qquad \frac{}{\Delta \vdash \mathbf{1}} \qquad \frac{}{\Delta \vdash \mathbf{N}}$$

All other type judgements follow the type introduction rules, e.g.:

$$\frac{\Delta \underline{\vdash} \underline{\mathbf{C}}}{\Delta \vdash \mathbf{U}\,\underline{\mathbf{C}}} \qquad \frac{\Delta \vdash \mathbf{A} \qquad \Delta \underline{\vdash} \underline{\mathbf{C}}}{\Delta \underline{\vdash} \mathbf{A} \to \underline{\mathbf{C}}}$$

Given some type $\underline{\underline{\mathbf{E}}}$ and some value type $\mathbf{A}$, we write $\underline{\underline{\mathbf{E}}}[\mathbf{A}/\alpha]$ for the substitution of $\mathbf{A}$ for each instance of α in $\underline{\underline{\mathbf{E}}}$. Similarly, we write $\underline{\underline{\mathbf{E}}}[\underline{\mathbf{C}}/\underline{\beta}]$ for the substitution of computation type $\underline{\mathbf{C}}$ for $\underline{\beta}$ in $\underline{\underline{\mathbf{E}}}$. In the same way, we define $\underline{\underline{P}}[\mathbf{A}/\alpha]$ and $\underline{\underline{P}}[\underline{\mathbf{C}}/\underline{\beta}]$ for such substitutions in some term $\underline{\underline{P}}$. By construction, we have the following lemma for type substitution:

Lemma 7.1.1. *If* $\Delta, \alpha \vdash \underline{\underline{\mathbf{E}}}$ *and* $\Delta \vdash \mathbf{A}$*, then* $\Delta \vdash \underline{\underline{\mathbf{E}}}[\mathbf{A}/\alpha]$.

We call a type $\underline{\underline{\mathbf{E}}}$ *closed* if it is of the empty context $\varepsilon \underline{\underline{\vdash}} \underline{\underline{\mathbf{E}}}$. We introduce the convention that a type is closed unless mentioned otherwise. We will only define a logic and equivalence for closed types, and their closed terms, similar to how we only have a logic for closed terms in this thesis. Type variables are used to create type constructors for universal polymorphic types and recursive types, both giving a wide supply of interesting programs.

In the specification of behavioural equivalence for these new types, one has to keep in mind that the formulas cannot be constructed by induction on types, they are constructed mutually inductively over all types simultaneously, as an induction on formulas.

7.2 Universal polymorphic types

By allowing us to build terms using type variables, we can express more general phenomena in the programming language. Take for instance the return function which, in a Church-style simply typed system (as used in this thesis), can only be the return function on a specific closed type (e.g., $\mathbf{N} \to \mathbf{F}\,\mathbf{N}$). However, there is a more general concept of return function, typed polymorphically by $\forall\alpha.\, \alpha \to \mathbf{F}\,\alpha$. In order to explore the behaviour of such general concepts, we introduce the universal polymorphic types.

We add universal polymorphic types over both value type variables and computation type variables. We consider both such polymorphic types as computation types, following the precedent set by function- and Π-types.

$$\frac{\Delta, \alpha \underline{\vdash} \underline{\mathbf{C}}}{\Delta \underline{\vdash} \forall\alpha.\, \underline{\mathbf{C}}} \qquad \frac{\Delta, \underline{\beta} \underline{\vdash} \underline{\mathbf{C}}}{\Delta \underline{\vdash} \forall\underline{\beta}.\, \underline{\mathbf{C}}}$$

Here, α is bound in $\forall\alpha.\, \underline{\mathbf{C}}$, and $\underline{\beta}$ is bound in $\forall\underline{\beta}.\, \underline{\mathbf{C}}$. We write $TV(\underline{\underline{\mathbf{E}}})$ and $TV(\Gamma)$ for the set of free type variables occurring in $\underline{\underline{\mathbf{E}}}$ and Γ respectively. We define new terms;

giving constructors and deconstructors for the new types, together with their typing judgements, which are added to Figure 2.1:

$$\frac{\Gamma \vdash \underline{M} : \underline{\mathbf{C}} \qquad \alpha \notin TV(\Gamma)}{\Gamma \vdash \Lambda\alpha.\, \underline{M} : \forall\alpha.\, \underline{\mathbf{C}}} \qquad \frac{\Gamma \vdash \underline{M} : \forall\alpha.\, \underline{\mathbf{C}} \qquad \alpha \notin TV(\Gamma) \cup TV(\mathbf{A})}{\Gamma \vdash \underline{M}_{\mathbf{A}} : \underline{\mathbf{C}}[\mathbf{A}/\alpha]}$$

$$\frac{\Gamma \vdash \underline{M} : \underline{\mathbf{C}} \qquad \underline{\beta} \notin TV(\Gamma)}{\Gamma \vdash \Lambda\underline{\beta}.\, \underline{M} : \forall\underline{\beta}.\, \underline{\mathbf{C}}} \qquad \frac{\Gamma \vdash \underline{M} : \forall\underline{\beta}.\, \underline{\mathbf{C}} \qquad \underline{\beta} \notin TV(\Gamma) \cup TV(\underline{\mathbf{D}})}{\Gamma \vdash \underline{M}_{\underline{\mathbf{D}}} : \underline{\mathbf{C}}[\underline{\mathbf{D}}/\underline{\beta}]}$$

Here, α is bound in $\Lambda\alpha.\, \underline{M}$, and $\underline{\beta}$ is bound in $\forall\underline{\beta}.\, \underline{\mathbf{C}}$.

As an example of a new term, the polymorphic identity function can be given by the return function $\Lambda\alpha.\, (\lambda x : \alpha.\, \mathsf{return}(x)) : \forall\alpha.\, (\alpha \to \mathbf{F}\,\alpha)$ and the force function $\Lambda\underline{\beta}.\, (\lambda x : \mathbf{U}\,\underline{\beta}.\, \mathsf{force}(x)) : \forall\underline{\beta}.\, (\mathbf{U}\,\underline{\beta} \to \underline{\beta})$, for call-by-value and call-by-name style respectively.

We define the operational semantics on closed terms of closed types by adding new rules to the already defined reduction relation. Since both $\forall\alpha.\, \underline{\mathbf{C}}$ and $\forall\underline{\beta}.\, \underline{\mathbf{C}}$ are computation terms, they contain term that may be reduced. So we first identify what they can reduce to, their *terminal terms*; expanding Definition 2.1.3 with the following two terms:

$$\Lambda\alpha.\, \underline{M} \quad | \quad \Lambda\underline{\beta}.\, \underline{M} \ .$$

In order to evaluate terms $\underline{M}_{\mathbf{A}}$ and $\underline{M}_{\underline{\mathbf{C}}}$, the term $\underline{M}$ needs to be evaluated first. As such, we need to add two new *stack* constructors:

$$S ::= \cdots \mid S \circ -_{\mathbf{A}} \mid S \circ -_{\underline{\mathbf{C}}} \ , \qquad \text{where } (S \circ -_{\underline{\mathbf{E}}})\{\underline{M}\} = S\{\underline{M}_{\underline{\mathbf{E}}}\} \ .$$

The stack reduction relation $\rightarrowtail$ from Definition 2.1.7 is extended to include these terms in the following way:

8. $(S, \underline{M}_{\mathbf{A}}) \rightarrowtail (S \circ -_{\mathbf{A}}, \underline{M})$.
9. $(S \circ -_{\mathbf{A}}, \Lambda\alpha.\, \underline{M}) \rightarrowtail (S, \underline{M}[\mathbf{A}/\alpha])$.
8'. $(S, \underline{M}_{\underline{\mathbf{C}}}) \rightarrowtail (S \circ -_{\underline{\mathbf{C}}}, \underline{M})$.
9'. $(S \circ -_{\underline{\mathbf{D}}}, \Lambda\underline{\beta}.\, \underline{M}) \rightarrowtail (S, \underline{M}[\underline{\mathbf{D}}/\underline{\beta}])$.

These new rules induce a new operational semantics $|-| : \mathit{Terms}(\underline{\mathbf{C}}) \to T(\mathit{Tct}(\underline{\mathbf{C}}))$. This finishes step (I) as laid out in Section 7.1. We will now turn towards step (II), extending the quantitative logic.

7.2.1 Logic for universal polymorphic types

Considering the similarity with function types, it is not hard to identify what should be considered a behavioural property of terms of universal polymorphic types. As such, we define the basic quantitative formulas for the new types as follows, adding two new rules to Figure 6.1:

$$\frac{\alpha \vdash \underline{\mathbf{C}} \qquad \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}[\mathbf{A}/\alpha])}{\mathbf{A} \mapsto \underline{\phi} \in \mathit{Form}(\forall\alpha.\, \underline{\mathbf{C}})} \qquad \frac{\underline{\beta} \vdash \underline{\mathbf{C}} \qquad \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}[\underline{\mathbf{D}}/\underline{\beta}])}{\underline{\mathbf{D}} \mapsto \underline{\phi} \in \mathit{Form}(\forall\underline{\beta}.\, \underline{\mathbf{C}})}$$

Note that formulas are only defined over closed types. Since we are extending the quantitative logic, satisfaction of such formulas is given by the following rules, extending the definition given at the end of Subsection 6.1.1.

$$\underline{M} \models \mathbf{A} \mapsto \underline{\phi} \;:=\; \underline{M}_{\mathbf{A}} \models \underline{\phi} \qquad \underline{M} \models \underline{\mathbf{D}} \mapsto \underline{\phi} \;:=\; \underline{M}_{\underline{\mathbf{D}}} \models \underline{\phi} \;.$$

With the extended quantitative logic, we can define the behavioural equivalence $\equiv$ and positive behavioural preorder $\sqsubseteq^+$ for the extended language, as defined in Definition 6.3.1 and the paragraph below it.

Firstly, note that the proof of Lemma 3.4.1 can be easily adapted to work for the new polymorphic computation types, hence for $\underline{M}, \underline{N} : \forall\alpha.\underline{\mathbf{C}}$ we have that $|\underline{M}| = |\underline{N}|$ implies $\underline{M} \equiv \underline{N}$. We also get the following immediate classification result for the new types (c.f. extending Lemma 3.4.3):

Lemma 7.2.1. *For $\mathcal{R}$ either $\sqsubseteq^+$ or $\equiv$, it holds that:*

8. $\underline{M}\,\mathcal{R}_{\forall\alpha.\underline{\mathbf{C}}}\,\underline{N} \iff \forall\mathbf{A},\ \underline{M}_{\mathbf{A}}\,\mathcal{R}_{\underline{\mathbf{C}}[\mathbf{A}/\alpha]}\,\underline{N}_{\mathbf{A}}$.

8'. $\underline{M}\,\mathcal{R}_{\forall\underline{\beta}.\underline{\mathbf{C}}}\,\underline{N} \iff \forall\underline{\mathbf{D}},\ \underline{M}_{\mathbf{A}}\,\mathcal{R}_{\underline{\mathbf{C}}[\underline{\mathbf{D}}/\underline{\beta}]}\,\underline{N}_{\mathbf{A}}$.

Proof. Suppose $\underline{M} \sqsubseteq^+ \underline{N} : \forall\alpha.\underline{\mathbf{C}}$, then $(\underline{M} \models \mathbf{A} \mapsto \underline{\phi}) \trianglelefteq (\underline{N} \models \mathbf{A} \mapsto \underline{\phi})$, hence $(\underline{M}_{\mathbf{A}} \models \underline{\phi}) \trianglelefteq (\underline{N}_{\mathbf{A}} \models \underline{\phi})$, implying that $\underline{M}_{\mathbf{A}} \sqsubseteq^+ \underline{N}_{\mathbf{A}}$. The converse is equally straightforward, and the proof for $\equiv$ and $\forall\underline{\beta}.\underline{\mathbf{C}}$ goes similarly . □

7.2.2 Applicative bisimilarity for polymorphic types

Following step (III) from Section 7.1, we extend the notion of applicative bisimilarity by adding relevant clauses to the definition of applicative $\mathcal{Q}$-simulation for the types created by by the new type constructors. Since these should reflect the behaviour of the terms, these are inspired by the simulation rule for function types.

Definition 7.2.2. (Addition to Definition 4.2.1) A well-typed relation $\mathcal{R}$ is an *applicative $\mathcal{Q}$-simulation* if moreover:

8. $\underline{M}\,\mathcal{R}_{\forall\alpha.\underline{\mathbf{C}}}\,\underline{N} \implies \forall\mathbf{A},\ \underline{M}_{\mathbf{A}}\,\mathcal{R}_{\underline{\mathbf{C}}[\mathbf{A}/\alpha]}\,\underline{N}_{\mathbf{A}}$

8'. $\underline{M}\,\mathcal{R}_{\forall\underline{\beta}.\underline{\mathbf{C}}}\,\underline{N} \implies \forall\underline{\mathbf{D}},\ \underline{M}_{\underline{\mathbf{D}}}\,\mathcal{R}_{\underline{\mathbf{C}}[\underline{\mathbf{D}}/\underline{\beta}]}\,\underline{N}_{\underline{\mathbf{D}}}$

With this extended definition, we can define the notion of applicative $\mathcal{Q}$-similarity and $\mathcal{Q}$-bisimilarity. We want to establish the Generalised Coincidence Theorem, Theorem 6.4.8, establishing that the behavioural equivalence is equal to $\mathcal{Q}$-bisimilarity.

The proofs in this section will only be carried out for $\forall\alpha.\underline{\mathbf{C}}$ types, since the proofs for $\forall\underline{\beta}.\underline{\mathbf{C}}$ types are practically identical.

Theorem 7.2.3. *(Theorem 6.4.8 in the presence of polymorphic types) If all modalities are leaf-monotone then the positive behavioural preorder $\sqsubseteq^+$ is applicative $\mathcal{Q}$-similarity and the general behavioural preorder $\equiv$ is applicative $\mathcal{Q}$-bisimilarity.*

Proof. Part 1: By Lemma 7.2.1, we can extend the proof of Lemma 6.4.4 to prove that the positive behavioural preorder is an applicative $\mathcal{Q}$-simulation.

Part 2: We add one case to the proof of Lemma 6.4.6, that any applicative $\mathcal{Q}$-simulation $\mathcal{R}$ preserves the new formula $\mathbf{A} \mapsto \underline{\phi}$ given that it preserves $\underline{\phi}$. Assume $\underline{M}\,\mathcal{R}_{\forall\alpha.\underline{\mathbf{C}}}\,\underline{N}$, then by the simulation rule it holds that $\underline{M}_{\mathbf{A}}\,\mathcal{R}_{\underline{\mathbf{C}}[\mathbf{A}/\alpha]}\,\underline{N}_{\mathbf{A}}$. Hence since $\mathcal{R}$ preserves $\underline{\phi}$, it holds that $(\underline{M} \models \mathbf{A} \mapsto \underline{\phi}) = (\underline{M}_{\mathbf{A}} \models \underline{\phi}) \trianglelefteq (\underline{N}_{\mathbf{A}} \models \underline{\phi}) = (\underline{M} \models \mathbf{A} \mapsto \underline{\phi})$.

We can conclude that $\sqsubseteq^+$ is the largest $\mathcal{Q}$-simulation, and by adapting the proofs slightly (c.f. Theorem 6.4.8 and Lemma 6.4.7), we can also conclude that $\equiv$ is the largest symmetric $\mathcal{Q}$-simulation. □

7.2.3 The Compatibility Theorem for polymorphic types

In order to prove compatibility of the open extension of the behavioural equivalence for the extended language, we first need to establish what it means to be compatible. We have four more compatible refinement rules, for each of the term introduction rules, extending Figure 3.2.

$$\frac{\Gamma \vdash \underline{M}\,\mathcal{R}\,\underline{N} : \underline{\mathbf{C}}}{\Gamma \vdash \Lambda\alpha.\,\underline{M}\,\widehat{\mathcal{R}}\,\Lambda\alpha.\,\underline{N} : \forall\alpha.\,\underline{\mathbf{C}}}\mathbf{P1} \qquad \frac{\Gamma \vdash \underline{M}\,\mathcal{R}\,\underline{N} : \forall\alpha.\,\underline{\mathbf{C}}}{\Gamma \vdash \underline{M}_{\mathbf{A}}\,\widehat{\mathcal{R}}\,\underline{N}_{\mathbf{A}} : \underline{\mathbf{C}}[\mathbf{A}/\alpha]}\mathbf{P2}$$

$$\frac{\Gamma \vdash \underline{M}\,\mathcal{R}\,\underline{N} : \underline{\mathbf{C}}}{\Gamma \vdash \Lambda\underline{\beta}.\,\underline{M}\,\widehat{\mathcal{R}}\,\Lambda\underline{\beta}.\,\underline{N} : \forall\underline{\beta}.\,\underline{\mathbf{C}}}\mathbf{P3} \qquad \frac{\Gamma \vdash \underline{M}\,\mathcal{R}\,\underline{N} : \forall\underline{\beta}.\,\underline{\mathbf{C}}}{\Gamma \vdash \underline{M}_{\underline{\mathbf{D}}}\,\widehat{\mathcal{R}}\,\underline{N}_{\underline{\mathbf{D}}} : \underline{\mathbf{C}}[\underline{\mathbf{D}}/\underline{\beta}]}\mathbf{P4}$$

Remember that a well-typed relation $\mathcal{R}$ is compatible if $\widehat{\mathcal{R}} \subseteq \mathcal{R}$. We write $\overline{\gamma}$ and $\overline{\underline{\mathbf{E}}}$ for a tuple of type variables and types respectively. We change the definition of *open extension*, given in Definition 3.3.7, by adding substitution of types, where for $\overline{\gamma} \vdash \underline{\mathbf{E}}$:

$$\overline{x} : \Gamma \vdash \underline{P}\,\mathcal{R}^{\circ}\,\underline{R} : \underline{\mathbf{E}} \Leftrightarrow \forall\overline{\underline{\mathbf{F}}}, \forall\overline{V} : \Gamma[\overline{\underline{\mathbf{F}}}/\overline{\gamma}], \varepsilon \vdash \underline{P}[\overline{\underline{\mathbf{F}}}/\overline{\gamma}, \overline{V}/\overline{x}]\ \mathcal{R}\ \underline{R}[\overline{\underline{\mathbf{F}}}/\overline{\gamma}, \overline{V}/\overline{x}] : \underline{\mathbf{E}}[\overline{\underline{\mathbf{F}}}/\overline{\gamma}].$$

All that remains is to adapt the Howe's method proof, following step (IV) from Section 7.1. Since we added type variables, the notion of *substitutivity* from Lemma 4.4.3 needs to be extended to include substitution of types, as in [75]. So we start with an auxiliary result, that the Howe closure is subsitutive in this extended sense. This has a similar proof as Lemma 4.4.3. Recall that the Howe closure is defined with rule (**H**) in Definition 4.4.1.

Lemma 7.2.4 (Type Substitutivity)**.** *Suppose* $\Delta, \gamma \vdash \underline{\mathbf{E}}$ *and* $\Gamma \vdash \underline{P}\mathcal{R}^{\bullet}\underline{R} : \underline{\mathbf{E}}$*. then for all* $\Delta \vdash \underline{\mathbf{F}}$*,* $\Gamma[\underline{\mathbf{F}}/\gamma] \vdash \underline{P}[\underline{\mathbf{F}}/\gamma]\ \mathcal{R}^{\bullet}\ \underline{R}[\underline{\mathbf{F}}/\gamma] : \underline{\mathbf{E}}[\underline{\mathbf{F}}/\gamma]$.

Proof. This requires an induction on the shape (or typing derivation) of $\underline{P}$ (which may be a value or a computation type). If $\Delta, \gamma \vdash \underline{\mathbf{E}}$, $\Delta \vdash \underline{\mathbf{F}}$ and $\Gamma \vdash \underline{P}\mathcal{R}^{\bullet}\underline{R} : \underline{\mathbf{E}}$ then by **H** it holds that $\Gamma \vdash \underline{P}\widehat{\mathcal{R}^{\bullet}}\underline{Q}$ and $\Gamma \vdash \underline{Q}\,\mathcal{R}^{\circ}\,\underline{R}$ for some $\underline{Q}$. So we know that $\Gamma \vdash \underline{Q}[\underline{\mathbf{F}}/\gamma]\,\mathcal{R}^{\circ}\,\underline{R}[\underline{\mathbf{F}}/x]$, by the extended definition of open extension. We need to prove that $\Gamma \vdash \underline{P}[\underline{\mathbf{F}}/\gamma]\,\widehat{\mathcal{R}^{\bullet}}\,\underline{Q}[\underline{\mathbf{F}}/\gamma])$. In each of the cases of $\underline{P}$, $\Gamma \vdash \underline{P}\widehat{\mathcal{R}^{\bullet}}\underline{Q}$ is derived from rule **Cn** or **Pn** for some number n. This rule has as its premise some sequence of relations $\underline{P}_i\,\mathcal{R}^{\bullet}\,\underline{Q}_i$. By induction hypothesis it holds that $\underline{P}_i[\underline{\mathbf{F}}/\gamma]\,\mathcal{R}^{\bullet}\,\underline{Q}_i[\underline{\mathbf{F}}/\gamma]$, this is also trivially

true in the base cases $n \in \{1, 2\}$ since then the sequence is empty. Using **Cn** or **Pn** we can then derive that $\Gamma \vdash \underline{\underline{P}}[\underline{\underline{\mathbf{F}}}/\underline{\underline{\gamma}}]\widehat{\mathcal{R}^\bullet}\underline{\underline{Q}}[\underline{\underline{\mathbf{F}}}/\underline{\underline{\gamma}}]$. One can verify that this argument works for each of the cases of **Cn**. So $\Gamma \vdash \underline{\underline{P}}[\underline{\underline{\mathbf{F}}}/\underline{\underline{\gamma}}]\,\widehat{\mathcal{R}^\bullet}\,\underline{\underline{Q}}[\underline{\underline{\mathbf{F}}}/\underline{\underline{\gamma}}]$ and $\Gamma \vdash \underline{\underline{Q}}[\underline{\underline{\mathbf{F}}}/\underline{\underline{\gamma}}]\,\mathcal{R}^\circ\,\underline{\underline{R}}[\underline{\underline{\mathbf{F}}}/\underline{\underline{\gamma}}]$, hence $\Gamma \vdash \underline{\underline{P}}[\underline{\underline{\mathbf{F}}}/\underline{\underline{\gamma}}]\,\mathcal{R}^\bullet\,\underline{\underline{R}}[\underline{\underline{\mathbf{F}}}/\underline{\underline{\gamma}}]$. □

We assume that $\mathcal{Q}$ is a decomposable set of leaf-monotone and Scott-tree continuous modalities. Let $\subseteq$ be some applicative $\mathcal{Q}$-simulation.

Lemma 7.2.5. *The Howe closure $\subseteq^\bullet$ satisfies rule 8 and 8' from Definition 7.2.2.*

Proof. Assume $\underline{M} \subseteq^\bullet_{\forall\alpha.\,\underline{\mathbf{C}}} \underline{N}$ for two closed terms. Take some value type $\mathbf{A}$, then by compatibility of $\subseteq^\bullet$ (Lemma 4.4.2) we get $\underline{M}_{\mathbf{A}} \subseteq^\bullet_{\underline{\mathbf{C}}[\mathbf{A}/\alpha]} \underline{N}_{\mathbf{A}}$. □

Since the definition of a stack has been extended, so has the definition of a frame in Definition 4.4.10. Extending Definition 4.4.13; the frame $Z \circ -_{\mathbf{B}}$ *dominates* $S \circ -_{\mathbf{A}}$ if $\mathbf{A} = \mathbf{B}$, and Z dominates S. It is not difficult to check that Lemma 4.4.12, 4.4.11, 4.4.15, and 4.4.14 still hold, since the proofs are similar to the proofs for the Π-types. We manually check the least trivial lemma.

Lemma 7.2.6 (Lemma 4.4.15 in the presence of polymorphic types)**.** *If $S\{\underline{M'}\} \subseteq^\bullet \underline{N}$ then there is a frame Z and a term $\underline{N'}$ s.t.; Z dominates S, $\underline{M'} \,\widehat{\subseteq^\bullet}\, \underline{N'}$ and $Z\{\underline{N'}\} \subseteq \underline{N}$.*

Proof. We add a case to the proof of Lemma 4.4.15 by induction on S. Assume the statement holds for S'.

3. If $S = -_{\mathbf{A}} \circ S'$, then $S\{\underline{M'}\} = S'\{\underline{M'}\}_{\mathbf{A}}$. Now, there is a term $\underline{K}$ such that $S\{\underline{M'}\} \,\widehat{\subseteq^\bullet}\, \underline{K} \subseteq \underline{N}$. The statement $S'\{\underline{M'}\}_{\mathbf{A}} \,\widehat{\subseteq^\bullet}\, \underline{K}$ could only have been derived from rule **P2**, so we know there is a $\underline{K'}$ such that $\underline{K} = \underline{K'}_{\mathbf{A}}$ and $S'\{\underline{M'}\} \subseteq^\bullet \underline{K'}$.

   We use the induction hypothesis on $S'\{\underline{M'}\} \subseteq^\bullet \underline{K'}$ to find a term $\underline{N'}$ and frame Z' dominating S' such that $\underline{M'} \,\widehat{\subseteq^\bullet}\, \underline{N'}$ and $Z'\{\underline{N'}\} \subseteq \underline{K'}$. Let $Z := -_{\mathbf{A}} \circ Z'$, then Z dominates S. From $Z'\{\underline{N'}\} \subseteq \underline{K'}$ and the new simulation rule we have $Z\{\underline{N'}\} = Z'\{\underline{N'}\}_{\mathbf{A}} \subseteq \underline{K'}_{\mathbf{A}} = \underline{K}$. With $\underline{K} \subseteq \underline{N}$ we can conclude that $Z\{\underline{N'}\} \subseteq \underline{N}$ and from earlier $\underline{M'} \,\widehat{\subseteq^\bullet}\, \underline{N'}$. So Z and $\underline{N'}$ have the desired properties.

□

Lastly, we need to adapt the Key Lemma, Lemma 4.4.17, by adding a new case to Lemma 4.4.18. There is one more type of *informative* computation term, $\Lambda\alpha.\,\underline{M}$. We add the case here, as if we are in the proof of the Key Lemma.

Lemma 7.2.7. *(Lemma 4.4.17 (Key Lemma) in the presence of polymorphic types) Given two closed terms $\underline{M}, \underline{N} : \mathbf{F}\,\mathbf{A}$ such that $\underline{M} \subseteq^\bullet \underline{N}$, then $\forall n, |\underline{M}|_n \,\mathcal{O}(\subseteq^\bullet)\, |\underline{N}|$.*

Proof. It is sufficient to add the case of $\underline{M} = \Lambda\alpha.\,\underline{P}$ to the proof of Lemma 4.4.18.

14. If $\underline{M} = \Lambda\alpha.\,\underline{P}$, then for $\underline{M}$ to fit in S, S must be of the form $S' \circ -_{\mathbf{A}}$ for some value type $\mathbf{A}$. Since Z dominates S, $Z = Z' \circ -_{\mathbf{A}}$ where Z' dominates S'. The statement $\underline{M}\,\widehat{\subseteq^{\bullet}}\,\underline{N}$ could only have been derived via the new compatibility rule **P1**, so $\underline{N} = \Lambda\alpha.\,\underline{P}'$ for some $\underline{P}'$ such that $\underline{P} \subseteq^{\bullet} \underline{P}'$, hence by Lemma 7.2.4, $\underline{P}[\mathbf{A}/\alpha] \subseteq^{\bullet} \underline{P}'[\mathbf{A}/\alpha]$. Using Lemma 4.4.16 and (IH), we can do the following derivation:

$|S\{\underline{M}\}|_{n+1} = |S'\{\Lambda\alpha.\,\underline{P}_{\mathbf{A}}\}|_{n+1} = |S'\{\underline{P}[\mathbf{A}/\alpha]\}|_{n-1}\ \mathcal{O}(\subseteq^{\bullet})\ |Z'\{\underline{P}'[\mathbf{A}/\alpha]\}| = |Z'\{\Lambda\alpha.\,\underline{P}'_{\mathbf{A}}\}| = |Z\{\underline{N}\}|$.

□

We can conclude that the extended logics for the extended language gives us behavioural preorders with compatible open extensions. As such, we have verified the Generalised Compatibility Theorem, Theorem 6.3.15, for the extended language:

Theorem 7.2.8. *If $\mathcal{Q}$ is a decomposable set of leaf-monotone and Scott-tree continuous modalities, then the behavioural preorders for the language extended with universal polymorphic types, are compatible.*

7.2.4 Constructions using polymorphic types

There are some interesting examples of polymorphic terms besides the polymorphic return and force functions discussed before. Most notably, we can consider the effect operators as polymorphic terms. Before, when we said $\mathsf{or}(-,-) : \underline{\alpha} \times \underline{\alpha} \to \underline{\alpha}$, we meant that the effect operator $\mathsf{or}(-,-)$ exists for any instantiation of the meta-variable $\underline{\alpha}$. Now, we can consider $\underline{\alpha}$ as a type variable instead, considering the effect operator polymorphic.

So with the introduction of polymorphic types, we can construct polymorphic effect operators, in the sense of [57, 58]. Take for instance the following term:

$$\mathsf{OR} := \Lambda\underline{\beta}.\,(\lambda p : \mathbf{U}\,\underline{\beta} \times \mathbf{U}\,\underline{\beta}.\,\mathsf{pm}\ p\ \mathsf{as}\ (x,y).\mathsf{or}(\mathsf{force}(x), \mathsf{force}(y))) : \forall\underline{\beta}.\,\mathbf{U}\,\underline{\beta} \times \mathbf{U}\,\underline{\beta} \to \underline{\beta}$$

The call-by-push-value setting necessitates the use of the thunk type to construct such a polymorphic effect operator. This term takes as arguments a computation type $\underline{\mathbf{C}}$, and then a pair of thunked computation terms of type $\underline{\mathbf{C}}$, and combines them in the nondeterministic choice program of type $\underline{\mathbf{C}}$ which chooses between the two given computation terms.

Similarly, we can define in the case of global store, the polymorphic effect operator

$$\mathsf{LU} := \Lambda\underline{\beta}.\,(\lambda x : \mathbf{U}\,(\mathbf{N} \to \underline{\beta}).\,\mathsf{lookup}_l(y \mapsto \mathsf{force}(x)\ y)) : \forall\underline{\beta}.\,(\mathbf{U}\,(\mathbf{N} \to \underline{\beta})) \to \underline{\beta}$$

for the polymorphic version of the lookup operator.

Let us look at an example of a Boolean formula we can construct in the logic for polymorphic types. We call a type $\mathbf{A}$ *rich* if it has two terms $V, W : \mathbf{A}$ such that $V \not\equiv W$. For a nondeterministic language, we define the following formula of type $\forall\alpha.\,\alpha \to \alpha \to \alpha \to \mathbf{F}\,\alpha$, as a conjunction over rich types:

$$\bigwedge_{\mathbf{A},\ \text{rich}} \mathbf{A} \mapsto \bigvee\{(V \mapsto (W \mapsto (L \mapsto \Diamond(\phi) \wedge \Diamond(\psi)))) \mid V, W, L : \mathbf{A}, \phi, \psi \in \mathit{Form}(\mathbf{A}), \phi \wedge \psi \equiv \bot\}$$

This formula checks whether there are three arguments such that two disjoint formulas may be satisfied according to the may deterministic $\Diamond$ modality. This is equivalent to saying that the terms carries information from at least two of its arguments, meaning at least two different arguments may possibly be returned. It is satisfied for example by terms $\Lambda\alpha.\,\lambda x.\,\lambda y.\,\lambda z.\,\mathsf{or}(\mathsf{return}(x),\mathsf{return}(y))$, $\Lambda\alpha.\,\lambda x.\,\lambda y.\,\lambda z.\,\mathsf{or}(\mathsf{return}(y),\mathsf{return}(z))$, and $\Lambda\alpha.\,\lambda x.\,\lambda y.\,\lambda z.\,\mathsf{or}(\mathsf{or}(\mathsf{return}(x),\Omega),\mathsf{or}(\mathsf{return}(y),\mathsf{return}(z)))$, but not by the term $\Lambda\alpha.\,\lambda x.\,\lambda y.\,\lambda z.\,\mathsf{return}(y)$.

7.3 Recursive types

In order to greatly increase the expressibility of the language, we add recursive types. This is a very powerful programming mechanism, allowing us to for example model the untyped lambda calculus, recreate the fixpoint operator, and create recursive data types like lists. For full generality, we consider adding recursive types on top of universal polymorphic types, but we could of course alternatively add the types independently.

We add both a value recursive type and a computation recursive type constructor.

$$\frac{\Delta, \alpha \vdash \mathbf{A}}{\Delta \vdash \mu\alpha.\,\mathbf{A}} \qquad\qquad \frac{\Delta, \underline{\beta} \vdash \underline{\mathbf{C}}}{\Delta \vdash \underline{\mu\beta}.\,\underline{\mathbf{C}}}$$

Here, α is bound in $\mu\alpha.\,\mathbf{A}$, and $\underline{\beta}$ is bound in $\underline{\mu\beta}.\,\underline{\mathbf{C}}$. We define both term constructors and term deconstructors for the new type, together with their typing judgements added to Figure 2.1:

$$\frac{\Gamma \vdash V : \mathbf{A}[\mu\alpha.\,\mathbf{A}/\alpha]}{\Gamma \vdash \mathsf{fold}(V) : \mu\alpha.\,\mathbf{A}} \qquad \frac{\Gamma \vdash V : \mu\alpha.\,\mathbf{A} \quad \Gamma, x : \mathbf{A}[\mu\alpha.\,\mathbf{A}/\alpha] \;\vdash\; \underline{M} : \underline{\mathbf{C}}}{\Gamma \;\vdash\; \mathsf{pm}\, V\, \mathsf{as}\, \mathsf{fold}(x).\, \underline{M} : \underline{\mathbf{C}}}$$

$$\frac{\Gamma \;\vdash\; \underline{M} : \underline{\mathbf{C}}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}]}{\Gamma \;\vdash\; \underline{\mathsf{fold}}(\underline{M}) : \underline{\mu\beta}.\,\underline{\mathbf{C}}} \qquad \frac{\Gamma \;\vdash\; \underline{M} : \underline{\mu\beta}.\,\underline{\mathbf{C}}}{\Gamma \;\vdash\; \underline{\mathsf{unfold}}(\underline{M}) : \underline{\mathbf{C}}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}]}$$

We expand the operational semantics. The new computation term created for the value recursive type $\mu\alpha.\,\mathbf{A}$ can be directly reduced with the following rule, added to Definition 2.1.4:

8. $\mathsf{pm}\,(\mathsf{fold}(V))\,\mathsf{as}\,\mathsf{fold}(x).\,\underline{M} \;\rightsquigarrow\; \underline{M}[V/x]$.

In order to evaluate the computation term $\underline{\mathsf{unfold}}(\underline{M})$ created for the computation recursive type $\underline{\mu\beta}.\,\underline{\mathbf{C}}$, we need to first evaluate $\underline{M}$. Hence, we need to add the unfold operation to the stacks:

$$S ::= \cdots \mid S \circ \underline{\mathsf{unfold}}(-) \qquad\qquad \text{where} \quad (S \circ \underline{\mathsf{unfold}}(-))\{\underline{M}\} := S\{\underline{\mathsf{unfold}}(\underline{M})\} \;.$$

To finish of the operational semantics, we add the two appropriate stack reduction rules from Definition 2.1.7:

10. $(S, \underline{\mathsf{unfold}}(\underline{M})) \rightarrowtail (S \circ \underline{\mathsf{unfold}}(-), \underline{M})$.

11. $(S \circ \underline{\mathsf{unfold}}(-), \underline{\mathsf{fold}}(\underline{M})) \rightarrowtail (S, \underline{M})$.

This finishes the extension of the language and its operational semantics.

7.3.1 Logic for recursive types

We now need to establish what is considered a behavioural property of terms of these new types, to appropriately extend the behavioural logic. Both types have terms introduced by a $\mathsf{fold}(-)$ operator, hence we define a formula by lifting a property along the term introduction rules of the $\mathsf{fold}(-)$ operator, adding two new rules to Figure 6.1:

$$\frac{\phi \in \mathit{Form}(\mathbf{A}[\mu\alpha.\,\mathbf{A}/\alpha])}{f(\phi) \in \mathit{Form}(\mu\alpha.\,\mathbf{A})} \qquad \frac{\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}])}{\underline{f}(\underline{\phi}) \in \mathit{Form}(\underline{\mu\beta}.\,\underline{\mathbf{C}})}$$

We define the satisfaction of the new formulas as follows, extending the definition given at the end of Subsection 6.1.1.

$$(\mathsf{fold}(V) \models f(\phi)) \;:=\; (V \models \phi) \qquad\qquad (\underline{M} \models \underline{f}(\underline{\phi})) \;:=\; (\underline{\mathsf{unfold}}(\underline{M}) \models \underline{\phi})\ .$$

Note that the difference between the two definitions is similar to the difference between basic formulas of pair types and basic formulas of Π-types.

With the extended quantitative logic, we can define the behavioural equivalence $\equiv$ and positive behavioural preorder $\sqsubseteq^+$ for the extended language, as defined in Definition 6.3.1 and the paragraph below it. We can include the following two classifications (c.f. Lemma 3.4.3) of those relations:

Lemma 7.3.1. *For $\mathcal{R}$ either $\sqsubseteq^+$ or $\equiv$, it holds that:*

9. $\mathsf{fold}(V)\,\mathcal{R}_{\mu\alpha.\,\mathbf{A}}\,\mathsf{fold}(V) \iff V\,\mathcal{R}_{\mathbf{A}[\mu\alpha.\,\mathbf{A}/\alpha]}\,W$.

9'. $\underline{M}\ \mathcal{R}_{\underline{\mu\beta}.\,\underline{\mathbf{C}}}\ \underline{N} \iff \underline{\mathsf{unfold}}(\underline{M})\ \mathcal{R}_{\underline{\mathbf{C}}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}]}\ \underline{\mathsf{unfold}}(\underline{N})$.

Proof. $\mathsf{fold}(V) \sqsubseteq^+_{\mu\alpha.\,\mathbf{A}} \mathsf{fold}(W)$ holds if and only if $\forall \phi \in \mathit{Form}(\mathbf{A}[\mu\alpha.\,\mathbf{A}])$, $(\mathsf{fold}(V) \models f(\phi)) \trianglelefteq (\mathsf{fold}(W) \models f(\phi))$, meaning $(V \models \phi) \trianglelefteq (W \models \phi)$. This precisely holds when $V \sqsubseteq^+_{\mathbf{A}[\mu\alpha.\,\mathbf{A}]} W$.

$\underline{M} \sqsubseteq^+_{\underline{\mu\beta}.\,\underline{\mathbf{C}}} \underline{N}$ holds if and only if $\forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}])$, $(\underline{\mathsf{unfold}}(\underline{M}) \models \underline{\phi}) = (\underline{M} \models \underline{f}(\underline{\phi})) \trianglelefteq (\underline{N} \models \underline{f}(\underline{\phi})) = (\underline{\mathsf{unfold}}(\underline{N}) \models \underline{\phi})$. This is precisely when $\underline{\mathsf{unfold}}(\underline{M}) \sqsubseteq^+_{\underline{\mathbf{C}}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}]} \underline{\mathsf{unfold}}(\underline{N})$. □

7.3.2 Applicative bisimilarity for recursive types

We extend the definition of applicative $\mathcal{Q}$-simulation by adding simulation rules for the new types. This simultaneously defines applicative $\mathcal{Q}$-similarity and $\mathcal{Q}$-bisimilarity.

Definition 7.3.2. (Addition to Definition 7.2.2) A well-typed relation $\mathcal{R}$ is an *applicative $\mathcal{Q}$-simulation* if moreover:

9. $\mathsf{fold}(V)\,\mathcal{R}_{\mu\alpha.\,\mathbf{A}}\,\mathsf{fold}(W) \implies V\,\mathcal{R}_{\mathbf{A}[\mu\alpha.\,\mathbf{A}/\alpha]}\,W$.

9'. $\underline{M}\mathcal{R}_{\underline{\mu\beta}.\,\underline{\mathbf{C}}}\underline{N} \implies (\underline{\mathsf{unfold}}(\underline{M})\,\mathcal{R}_{\underline{\mathbf{C}}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}]}\,\underline{\mathsf{unfold}}(\underline{N}))$.

Just as we did in Subsection 7.2.2, we adapt the proof of the Generalised Coincidence Theorem appropriately to be suitable for this extended notion of applicative $\mathcal{Q}$-bisimilarity.

Theorem 7.3.3 (Theorem 6.4.8 with universal polymorphic and recursive types)**.** *If all modalities are leaf-monotone, then the positive behavioural preorder $\sqsubseteq^+$ is applicative $\mathcal{Q}$-similarity and the general behavioural preorder $\equiv$ is applicative $\mathcal{Q}$-bisimilarity.*

Proof. We add two cases to Part 1 and Part 2 of the proof of Theorem 7.2.3.

Part 1: With Lemma 7.3.1 we know that $\sqsubseteq^+$ satisfies the new simulation rules 9 and 9', so we can add the two cases to prove that the positive behavioural preorder is an applicative $\mathcal{Q}$-simulation.

Part 2, we add two more cases to the proof of Lemma 6.4.6, showing that any applicative $\mathcal{Q}$-simulation $\mathcal{R}$ preserves the new formulas $f(\phi)$ and $\underline{f}(\underline{\phi})$ given that it preserves ϕ and $\underline{\phi}$.

Assume $\mathsf{fold}(V)\,\mathcal{R}_{\mu\alpha.\,\mathbf{A}}\,\mathsf{fold}(W)$, then by the simulation rule it holds that $V\,\mathcal{R}_{\mathbf{A}[\mu\alpha.\,\mathbf{A}/\alpha]}\,W$. Since $\mathcal{R}$ preserves ϕ, we derive that: $(\mathsf{fold}(V) \models f(\phi)) = (V \models \phi) \trianglelefteq (W \models \phi) = (\mathsf{fold}(W) \models f(\phi))$. Hence $\mathcal{R}$ preserves $f(\phi)$.

Assume $\underline{M}\,\mathcal{R}_{\underline{\mu\beta}.\,\mathbf{C}}\,\underline{N}$, then by the simulation rule it holds that $\underline{\mathsf{unfold}}(\underline{M})\,\mathcal{R}_{\mathbf{A}[\underline{\mu\beta}.\,\mathbf{C}/\underline{\beta}]}\,\underline{\mathsf{unfold}}(\underline{N})$. Since $\mathcal{R}$ preserves $\underline{\phi}$, we derive that: $(\underline{M} \models \underline{f}(\underline{\phi})) = (\underline{\mathsf{unfold}}(\underline{M}) \models \underline{\phi}) \trianglelefteq (\underline{\mathsf{unfold}}(\underline{N}) \models \underline{\phi}) = (\underline{N} \models \underline{f}(\underline{\phi}))$. Hence $\mathcal{R}$ preserves $\underline{f}(\underline{\phi})$.

We can conclude that $\sqsubseteq^+$ is the largest $\mathcal{Q}$-simulation, and by adapting the proofs slightly (c.f. Theorem 6.4.8 and Lemma 6.4.7) that $\equiv$ is the largest symmetric $\mathcal{Q}$-simulation. □

7.3.3 The Compatibility Theorem for recursive types

We will prove that open extensions of the behavioural preorders induced by the logic are compatible. Let us first consider how the definition of compatible refinement can be broadened to appropriately adapt the definition of compatibility for this extended language. We extend Figure 3.2 with the following four rules:

$$\frac{\Gamma \vdash V\,\mathcal{R}\,W : \mathbf{A}[\mu\alpha.\,\mathbf{A}/\alpha]}{\Gamma \vdash \mathsf{fold}(V)\,\widehat{\mathcal{R}}\,\mathsf{fold}(W) : \mu\alpha.\,\mathbf{A}}\mathbf{R1}$$

$$\frac{\Gamma \vdash V\,\mathcal{R}\,W : \mu\alpha.\,\mathbf{A} \qquad \Gamma, x : \mathbf{A}[\mu\alpha.\,\mathbf{A}/\alpha] \vdash \underline{M}\,\mathcal{R}\,\underline{N} : \underline{\mathbf{C}}}{\Gamma \vdash (\mathsf{pm}\,V\,\mathsf{as}\,\mathsf{fold}(x).\,\underline{M})\,\widehat{\mathcal{R}}\,(\mathsf{pm}\,W\,\mathsf{as}\,\mathsf{fold}(x).\,\underline{N}) : \underline{\mathbf{C}}}\mathbf{R2}$$

$$\frac{\Gamma \vdash \underline{M}\,\mathcal{R}\,\underline{N} : \mathbf{A}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}]}{\Gamma \vdash \underline{\mathsf{fold}}(\underline{M})\,\widehat{\mathcal{R}}\,\underline{\mathsf{fold}}(\underline{N}) : \underline{\mu\beta}.\,\underline{\mathbf{C}}}\mathbf{R3} \qquad \frac{\Gamma \vdash \underline{M}\,\mathcal{R}\,\underline{N} : \underline{\mu\beta}.\,\underline{\mathbf{C}}}{\Gamma \vdash \underline{\mathsf{unfold}}(\underline{M})\,\widehat{\mathcal{R}}\,\underline{\mathsf{unfold}}(\underline{N}) : \underline{\mathbf{C}}[\underline{\mu\beta}.\,\underline{\mathbf{C}}/\underline{\beta}]}\mathbf{R4}$$

Now we extend the proofs necessary for establishing compatibility, following step (IV). Note the usual lemmas for the Howe closure hold, including type substitutivity by Lemma 7.2.4 from the previous section. Assume $\mathcal{Q}$ is a decomposable set of leaf-monotone and Scott-tree continuous modalities, and let $\subseteq$ be a $\mathcal{Q}$-simulation.

Lemma 7.3.4. *The Howe closure of a simulation $\subseteq$ satisfies the rules 9 and 9' from Definition 7.3.2.*

Proof. First the simulation rule for $\mu\alpha.\,\mathbf{A}$. If $\mathsf{fold}(V) \subseteq^\bullet \mathsf{fold}(W)$ then $\mathsf{fold}(V)\,\widehat{\subseteq^\bullet}\,\mathsf{fold}(L) \subseteq^\circ \mathsf{fold}(W)$ for some L. By the simulation property for $\subseteq$, and the

fact that L and W are closed, we know that $L \subseteq^{\circ} W$. Moreover, $\mathsf{fold}(V) \mathrel{\widehat{\subseteq^{\bullet}}} \mathsf{fold}(L)$ could have only been derived from the new compatible closure rule **R1**, hence $V \subseteq^{\bullet} L$. We can conclude that $V \subseteq^{\bullet} W$ by Lemma 4.4.4.

The simulation rule for $\underline{\mu\beta.\, \mathbf{C}}$ is muh easier to prove, since it follows from rule **R4** and the fact that $\subseteq^{\bullet}$ is compatible. □

There is one more constructor for stacks, so one more constructor for frames as well. Extending Definition 4.4.13, the frame $Z \circ \underline{\mathsf{unfold}}(-)$ *dominates* $S \circ \underline{\mathsf{unfold}}(-)$ if and only if Z dominates S. The usual lemmas about frames hold. Again, we prove the least trivial one:

Lemma 7.3.5. *(Lemma 4.4.15 with universal polymorphic and recursive computation types) If $S\{\underline{M'}\} \subseteq^{\bullet} \underline{N}$ then there is a frame Z and a term $\underline{N'}$ such that; Z dominates S, $\underline{M'} \mathrel{\widehat{\subseteq^{\bullet}}} \underline{N'}$ and $Z\{\underline{N'}\} \subseteq \underline{N}$.*

Proof. We add a case to the proof of Lemma 7.2.6 by induction on S. Assume the statement holds for S'.

4. $S = \underline{\mathsf{unfold}}(-) \circ S'$, then $S\{\underline{M'}\} = \underline{\mathsf{unfold}}(S'\{\underline{M'}\})$. Now, there is a term $\underline{K}$ such that $S\{\underline{M'}\} \mathrel{\widehat{\subseteq^{\bullet}}} \underline{K} \subseteq \underline{N}$. The statement $\underline{\mathsf{unfold}}(S'\{\underline{M'}\}) \mathrel{\widehat{\subseteq^{\bullet}}} \underline{K}$ could only have been derived from rule **R4**, so we know there is a $\underline{K'}$ such that $\underline{K} = \underline{\mathsf{unfold}}(\underline{K'})$ and $S'\{\underline{M'}\} \subseteq^{\bullet} \underline{K'}$.

   We use the induction hypothesis on $S'\{\underline{M'}\} \subseteq^{\bullet} \underline{K'}$ to find a term $\underline{N'}$ and frame Z' dominating S' such that $\underline{M'} \mathrel{\widehat{\subseteq^{\bullet}}} \underline{N'}$ and $Z'\{\underline{N'}\} \subseteq \underline{K'}$. Let $Z := \underline{\mathsf{unfold}}(-) \circ Z'$, then Z dominates S. From $Z'\{\underline{N'}\} \subseteq \underline{K'}$ and a simulation rule we have $Z\{\underline{N'}\} = \underline{\mathsf{unfold}}(Z'\{\underline{N'}\}) \subseteq \underline{\mathsf{unfold}}(\underline{K'}) = \underline{K}$. With $\underline{K} \subseteq \underline{N}$ we can conclude that $Z\{\underline{N'}\} \subseteq \underline{N}$ and from earlier $\underline{M'} \mathrel{\widehat{\subseteq^{\bullet}}} \underline{N'}$. So Z and $\underline{N'}$ have the desired properties.

□

We finish the adaptation of the proof by Howe's method by adding two new cases of computation terms to the Key Lemma (Lemma 4.4.17).

Lemma 7.3.6 (Lemma 4.4.17 with universal polymorphic and recursive types)**.** *Given two closed terms $\underline{M}, \underline{N} : \mathbf{F}\,\mathbf{A}$ such that $\underline{M} \subseteq^{\bullet} \underline{N}$, then $\forall n, |\underline{M}|_n \, \mathcal{O}(\subseteq^{\bullet}) \, |\underline{N}|$.*

Proof. It is sufficient to add the cases of $\underline{M} = (\mathsf{pm}\, V \,\mathsf{as}\, \mathsf{fold}(x).\, \underline{P})$ and $\underline{M} = \underline{\mathsf{unfold}}(\underline{N})$ to Lemma 4.4.18. Note that $\underline{\mathsf{unfold}}(\underline{P})$ is not an informative term, so need not be added (similar to how $\underline{P}\ V$ is not a case in the original lemma).

15. If $\underline{M} = (\mathsf{pm}\, V \,\mathsf{as}\, \mathsf{fold}(x).\, \underline{P})$, then it holds from **R2** that $\underline{N} = (\mathsf{pm}\, W \,\mathsf{as}\, \mathsf{fold}(x).\, \underline{Q})$ for some W and $\underline{Q}$ such that $V \subseteq^{\bullet} W$ and $\underline{P} \subseteq^{\bullet} \underline{Q}$. Now we have that $V = \mathsf{fold}(V')$ and $W = \mathsf{fold}(W')$ for some V' and W' since the terms are closed, from which by the new simulation rule we get $V' \subseteq^{\bullet} W'$. From $\underline{P} \subseteq^{\bullet} \underline{Q}$ and Lemma 4.4.3 it holds that $\underline{P}[V'/x] \subseteq^{\bullet} \underline{Q}[W'/x]$. We conclude with (IH) and Lemma 4.4.16 that:

   $|S\{\underline{M}\}|_{n+1} = |S\{\mathsf{pm}\, \mathsf{fold}(V') \,\mathsf{as}\, \mathsf{fold}(x).\, \underline{P}\}|_{n+1} =$
   $|S\{\underline{P}[V'/x]\}|_n \, \mathcal{Q}(\subseteq^{\bullet}) \, |S'\{\underline{Q}[W'/x]\}| = |S'\{\underline{N}\}|.$

16. If $\underline{M} = \underline{\mathsf{fold}}(\underline{P})$, then $\underline{M} \mathrel{\widehat{\subseteq^\bullet}} \underline{N}$ is from **R3**, so $\underline{N} = \underline{\mathsf{fold}}(\underline{Q})$ for some $\underline{Q}$ such that $\underline{P} \subseteq^\bullet \underline{Q}$. For $\underline{M}$ to fit in S (and $S\{\underline{M}\} : \mathbf{F}\,\mathbf{A}$), S must be of the form $S' \circ \underline{\mathsf{unfold}}(-)$. Since Z dominates S, $Z = Z' \circ \underline{\mathsf{unfold}}(-)$ such that Z' dominates S'. Hence $S\{\underline{P}\} \subseteq^\bullet Z\{\underline{Q}\}$ by Lemma 4.4.14. By (IH) and Lemma 4.4.16, we can derive that:

$$|S\{\underline{M}\}|_{n+1} = |S' \circ \underline{\mathsf{unfold}}(-)\{\underline{\mathsf{fold}}(\underline{P})\}|_{n+1} = |S'\{\underline{P}\}|_{n-1} \mathcal{Q}(\subseteq^\bullet) |Z'\{\underline{Q}\}| = |\underline{N}|.$$

□

We can conclude that the Generalised Compatibility Theorem, Theorem 6.3.15, holds for the language with universal polymorphic types and recursive types:

Theorem 7.3.7. *If $\mathcal{Q}$ is a decomposable set of leaf-monotone and Scott-tree continuous modalities, then the behavioural preorders for the language extended with universal polymorphic types and recursive types, are compatible.*

7.3.4 Constructions using recursive types

Using recursive types, we can define many interesting types and terms. For instance, it is possible to redefine the fixpoint operator using recursive types. In this subsection, we look at another construction. For simplicity, we write $\mathbf{A} + \mathbf{A}'$ for the binary sumtype $\mathbf{\Sigma}_{i \in I}\, \mathbf{B}_i$ where $I = \{0, 1\}$, $\mathbf{B}_0 = \mathbf{A}$, and $\mathbf{B}_1 = \mathbf{A}'$.

The natural numbers type $\mathbf{N}$ can be defined as the recursive type $\mu\alpha.\, \mathbf{1} + \alpha$ and hence need not be taken as primitive. The usual constructors and deconstructors of the natural numbers type can be redefined to:

$$\mathsf{Z} \;:=\; \mathsf{fold}((0, *)), \qquad \mathsf{S}(V) \;:=\; \mathsf{fold}((1, V)),$$

$$\mathsf{case}\ V\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(z) \Rightarrow \underline{N}\} \;:=\; \mathsf{pm}\, V\ \mathsf{as}\ \mathsf{fold}(x).\, (\mathsf{pm}\ x\ \mathsf{as}\ \{(0, y).\underline{M}, (1, z).\underline{N}\})\ .$$

The basic formulas for the natural numbers can also be retrieved, with $\{0\} := f((0, \top))$ and $\{n+1\} := f((1, \{n\}))$.

For any value type $\mathbf{A}$, we can create a type of lists $\mathbf{list}(\mathbf{A})$ of terms of that type, as $\mathbf{list}(\mathbf{A}) := \mu\alpha.\, \mathbf{1} + (\mathbf{A} \times \alpha)$. Basic formulas of $\mathbf{list}(\mathbf{A})$, as given by the formulas for recursive types, are of the following three forms:

- A formula $f((0, \top))$ which checks whether a list is empty. Here $V \models f((0, \top))$ yields $\boldsymbol{T}$ if V gives an empty list, and $\boldsymbol{F}$ otherwise.
- A formula $f((1, \pi_0(\phi)))$ where $\phi \in \mathit{Form}(\mathbf{A})$, checking the head of the list. $V \models f((1, \pi_0(\phi)))$ yields $\boldsymbol{F}$ if V gives an empty list, and $W \models \phi$ if V is a non-empty list whose head is given by W.
- A formula $f((1, \pi_1(\phi)))$ where $\phi \in \mathit{Form}(\mathbf{list}(\mathbf{A}))$, checking the tail of the list. $V \models f((1, \pi_1(\phi)))$ yields $\boldsymbol{F}$ if V gives an empty list, and $W \models \phi$ if V is a non-empty list whose tail is given by W.

The resulting behavioural equivalence will state that two lists are equivalent if they are of the same length, and elements at matching locations in the list are equivalent.

7.4 Thoughts on language extensions

With the addition of universal polymorphic types or recursive types, we have shifted away from a simply-typed system. One consequence of this is that we are unable to perform proofs regarding the pure logics from Section 5.4 and Subsection 6.3.4. In particular, we used inductions on types in Propositions 5.4.2 and 6.3.24 to prove that the pure logic induces the same equivalence as the general logic. However, because formulas of universal polymorphic types and recursive types may contain sub-formulas of 'higher' types, such inductions on types are not well-founded. For instance, we would need to find a characteristic function χ_V for some value of a polymorphic type $\forall\alpha.\,\mathbf{A}$. But such a formula may contain formulas for a higher type like $\mathbf{A}[\forall\alpha.\,\mathbf{A}/\alpha]$, breaking the induction. Hence, the results on pure logics do not directly apply to the language extended with universal polymorphic types and/or recursive types.

It may however be possible to prove Propositions 5.4.2 and 6.3.24 in another way. In [10], it was proven that properties of universal polymorphic types could be tested at a particular instantiation of the type. If such a thing would be true for the language studied in this thesis, behavioural equivalence could be determined at such a type instantiation. This may enable us to fix the necessary induction on types. It is however unclear whether the results from [10] could be extended to apply to the call-by-push-value language with algebraic effects used in this thesis.

Another issue with the current framework is the inability to incorporate existential types, e.g., as done in [74]. The behavioural equivalence formulated in this thesis coincides with a notion of applicative bisimilarity. However, existential types seem to be incompatible with applicative bisimilarity techniques. This problem is discussed in [99], where, to solve this issue, they consider a notion of bisimulation on program tuples in order to properly investigate languages with existential types.

In some languages, it is possible to model existential types as universal polymorphic types. E.g [57] defines an existential type as $\exists X.\mathbf{B}$ as $\forall Y.\,\forall X.\,\mathbf{B} \to Y \to Y$. However, there does not seem to be an appropriate translation of this type into the particular call-by-push-value language used in this thesis. None of the choices for translating this type seem to implement the right interpretation for existential types.

8

Conclusions

In this final chapter of the dissertation, we will discuss the topics of this thesis in relation to other work, and talk about potential topics for future research. This will be done in the form of five big questions.

What happened? We have defined a relation of equality between call-by-push-value programs with effects by specifying behavioural properties. The resulting behavioural equivalence was proven to be compatible for certain algebraic effects, including error, nondeterministic choice, probabilistic choice, global store, input/output, timer and certain combinations thereof.

The primary tool used for interpreting the behaviour of effectful programs is the notion of modality, which was adapted from the notion of observation in [35]. There, observations were defined on trees of type $\mathbf{N}$, whereas modalities are defined on unit-type effect trees, or quantity-valued trees in the case of the quantitative logic. Modalities in this dissertation are used to lift properties on values to properties on computations, analogous to predicate lifting in coalgebra [34]. The paper [35] features properties of Scott-openness and decomposability with a similar function to the notions in this thesis with the same name, though decomposability for modalities is more subtle than the corresponding notion for observations in [35].

The behavioural equivalence has been proven to be compatible by equating it to a notion of applicative bisimilarity [2], which interprets effects using relators (relation lifting devices). Such applicative bisimilarities for effectful programs were defined in [14] for an untyped lambda calculus with effects, where furthermore Howe's method [30, 31] was used to prove compatibility of this notion of bisimilarity. This proof was adapted in this thesis to work for typed call-by-push-value language with algebraic effects, general recursion, universal polymorphic types and recursive types.

Is it reasonable? In the literature, logics for effects have primarily been considered for the purpose of reasoning about programs with effects. Some such logics are sound with respect to notions of program equivalence, e.g., bisimilarity, mutual similarity and contextual equivalence. In general, programming logics are not used to determine a notion of compatible program equivalence, as is the purpose of the logic in this disser-

tation. However, it is possible for logics sound with respect to behavioural equivalence to be expressed within our logic. We review some of the alternative logic descriptions.

Pitts' *evaluation logic* was an early logic for general computational effects [73]. Evaluation logic has built-in $\Box$ and $\Diamond$ modalities, which are used to interpret any effect through the lens of demonic/must and angelic/may nondeterminism. This interpretation can be awkward for some examples of effects, for instance for probability. It would therefore be natural to consider replacing the $\Box$ and $\Diamond$ modalities in the evaluation logic with the effect-specific modalities considered in this dissertation, defined in Section 3.2.

A more general description of effectful properties is given in a logic for algebraic effects, from Plotkin and Pretnar [85]. There, equations are taken as the primitive interpretation of effectful behaviour, and are used to axiomatise an equational theory over the effect operators. This is in line with the algebraic interpretation of effects advocated by Plotkin and Power [83]. The chosen equational axioms from [85] are typically sound with respect to the program equivalence, and can thus be used to soundly reason about program equivalence. In this dissertation, modalities are chosen as primitives, and the equations are induced as a consequence, as seen in Section 3.5. We can see modalities as some kind of dual to equations, because they are better suited for establishing non-equivalence between two terms (by finding a formula that distinguishes the two). Potentially, the two theories could be used to complement each other.

The logic of [85] does use modalities, each determined by an individual effect operator from Σ. In the case of the input-output effect, such operator specific modalities could be expressed within the logic of this dissertation. Perhaps more usefully, the logic from [85] could be altered by replacing the 'local' operator modalities with the 'global' behavioural modalities used in this dissertation. This may provide an alternative more usable framework for specification and verification of programs with algebraic effects. For instance, as we have seen in Section 5.6, the notion of decomposability provides us with proof rules which are potentially helpful in verification. This leads us to the next question.

Is it practical? The main function of the logic in this dissertation is to formulate and define a notion of program equivalence. Since the base logics are infinitary, they cannot directly be used for practical applications such as specification and verification. However, as we have seen in Chapter 5 and Section 6.5, the logic necessary for specifying the behavioural equivalence can in many cases be reduced to a finitary logic. It would be interesting to see if this might be useful for designing algorithmic approaches to establishing non-equivalence between programs.

The low-level infinitary logic has merit too, as it serves as a base logic into which more practical finitary logics can be translated. Such a translation will prove that the logic is sound with respect to applicative bisimilarity from Chapter 4. For this purpose, the closure of the logics under infinitary propositional logic is crucial, as it allows for the standard translations of quantifiers and least and greatest fixed points into the logic. As another example, the translation of Hoare logic into the logic of this disserta-

tion, as illustrated in Subsection 5.5.1, makes crucial use of infinitary disjunctions and conjunctions.

As an alternative to the infinitary propositional logic, one might explore logics with finitary syntax, including for example fixpoint formulas and quantifiers, for expressing interesting properties in the logic (e.g., with modal μ-calculus [39]). The compositional proof rules established in Section 5.6 could potentially be expressed in such finitary logics, providing a framework for formal proofs that programs satisfy certain properties. This is one possible direction for developing practical applications from this thesis.

The previous paragraph discusses using the logic to prove *properties* of programs. It could also be interesting to verify *equivalence* between programs. In this respect, as mentioned before, the main potential of this logic seems to be establishing *non-equivalence*: two terms $\underset{\sim}{P}$ and $\underset{\sim}{R}$ are non-equivalent if there is a formula of their type distinguishing them. Bisimilarity, on the other hand, is a useful tool for proving equivalence, two terms are equivalent if there is a simulation relating the two. This proof technique becomes even more practical if 'up-to' methods [13, 90, 92] are available. Hence, it may be useful to consider the logic from this dissertation as a useful complement to bisimilarity, since the resulting behavioural equivalence coincides with a notion of applicative bisimilarity.

We may also relate the logic with other useful methods for establishing equivalence, such as the method of *logical relations* [9, 29, 35, 37, 75, 76] often used for establishing contextual equivalence. We have seen in Subsection 5.2.4 that for some examples of effects the positive behavioural preorder coincides with the contextual preorder. These examples include error, input/output, and global store with finite locations. A notable exception is nondeterminism, for which it has been proven, e.g., in [40, 41], that the bisimilarity is distinct from contextual equivalence. For some probabilistic languages, bisimilarity does coincide with contextual equivalence, as seen in [12]. As such, a similar result may hold in the context of this thesis.

Is it relevant? We compare the approach taken in this dissertation with some similar approaches to defining behavioural equivalence for effectful languages. There are for instance similarities between the Boolean logic in this thesis and the logic developed in Abramsky's domain theory in logical form [3]. There, a finite endogenous logic is used to characterise denotational equivalence. In contrast, in this dissertation we construct a logic on the operational semantics (with effect trees) to define behavioural equivalence.

We look at a plethora of relatively recent developments related to this thesis. For instance, another approach to logics for effects has been proposed by Goncharov, Mossakowski and Schröder [23, 65], who define the logic semantically within a pure fragment of the programming language itself. In the case of global store, this approach derives Hoare logic. However, the approach taken appears not to be applicable to as many examples as the approach used in this dissertation.

The quantitative logic also has similarities to other approaches. The denotations $[\![q]\!] : T\mathbb{A} \to \mathbb{A}$ of quantitative modalities are, in the case of the running examples, Eilenberg-Moore algebras (see Subsection 6.3.3). As such, our examples of quantitative modalities potentially fit into the framework of Hasuo [26]. It should be noted that

modalities need not be EM-algebras for the resulting equivalence to be compatible. For instance, the Scott open Boolean modalities denote functions from $T\mathbb{B} \to \mathbb{B}$ (see Subsection 6.5.1), but most of the example modalities in the Boolean logic do not denote EM-algebras. However, there might be a uniform way to combine all modalities from a decomposable set of Scott open modalities into a single EM-algebra, by choosing an appropriate truth space. This is in a way what happens when going from the Boolean modalities to the quantitative modality in the individual cases of probabilistic choice and global store.

Another approach to reasoning about effects is done in the F^* project [100], where an effectful programming language is described within a theorem-prover with a dependent type structure. At this stage, that framework supports many of the effect examples given in this thesis, described by *Dijkstra monads* [5, 52, 100, 101]. The formulation is based on identifying the right notion of precondition for a given postcondition, a concept which is intuitively clear in the case of global store, but becomes more subtle for other effect examples. In the case of global store, the precondition-postcondition style arises naturally in the framework of this thesis, as the $(s \rightarrowtail s')$ modality and the quantitative G modality both check correctness of beginning states for getting a certain result. For other examples of effects, the connection becomes less clear. Because of the potential practical properties of the logic of this thesis, further comparisons to Dijkstra monads could provide good reasoning tools for the latter. Moreover, for some effects the formulation using (quantitative) modalities may be considered more natural, for instance in the case of input/output where effects may be observed in the middle of execution (see a formulation of input/output with Dijkstra monads in [51, 72]).

In short, there are many current approaches to reasoning about and establishing equivalence for programs with effects. Each has its own merits, focussing at a specific level of generality. So it is that we arrive at the final question.

Is it general? The notion of program equivalence in this thesis has been defined for a call-by-push-value language with general recursion, finite sums and products, universal polymorphic types, recursive types, and a wide range of algebraic effects. As such, equivalences could be extracted for any language which can be embedded within it, for instance typed call-by-name, call-by-value and lazy PCF [42, 43], and untyped call-by-value lambda-calculus with algebraic effects [14]. Similarly, as discussed before, other logics and reasoning principles could be embedded within the infinitary propositional logic. Let us discuss the ways the language and logic can be made even more general.

In this thesis, we consider program properties (or observations) as the primary way of describing program behaviour. According to this philosophy, the generalisation to quantitative properties given in Chapter 6 is natural. Alternatively, one could consider relations (or comparisons) as primary, and instead generalise to quantitative relations. One approach in this direction is using metrics, along the lines of [6, 17, 53]. Relating the quantitative logic of this thesis, or a variation thereof, to metrics (e.g., like the ones in [21]) is a topic for future research. One technical difficulty in using quantitative relations is that sometimes certain properties related to metrics (such as non-expansivity) are

needed, and sometimes this necessitates using restrictive calculi such as affine calculi. It seems possible that there may be other approaches to quantitative relations that avoid such problems.

The quantitative logic does not however naturally induce a metric on terms. This is mainly because of the inclusion of threshold-formulas $\phi_{\trianglerighteq a}$, which takes the quantitative information from ϕ and collapses it to a binary value. These threshold-formulas are necessary for relating the behavioural equivalence to applicative bisimilarity. Their necessity can be seen as a natural consequence of the non-linearity of the language, created in part by general recursion.

The quantitative logic is very expressive, allowing one to deal with some awkward combinations of effects that are not amenable to a Boolean treatment. These combinations of effects are treated in a-case-by-case basis. A possible uniform theory for combining effects and their quantitative modalities is currently under development. Another potential future direction is developing a system of subtyping, where each program can be (dynamically) associated to a set of effects it might produce.

The algebraic operators used to model effects in this thesis are limited to arities formed by products of $\underline{\alpha}$, $\mathbf{N} \to \underline{\alpha}$ and $\mathbf{N}$. The theory makes use of the fact that terms of type $\mathbf{N}$ are equivalent if and only if they are syntactically the same. For terms of more complex types, this is not necessarily the case, which makes the definition of modalities over effect trees using operators with more general arities challenging. However, more general arities are needed to include effects such as local store, higher-order store, and jumps with dynamically created variables as in [20]. For some of these effects, the appropriate notion of bisimilarity is *environmental bisimilarity* [38, 91]. The complexity of this notion hints at the degree of complexity required for formulating a program logic for such effects.

Last but not least, we could consider another way of programming with algebraic effects using *handlers* [8, 86]. In general, handlers violate the standard equations associated with algebraic effects, and would hence break the behavioural equivalence. However, there are different options for implementing handlers to the current framework. Firstly, one can extend the language with effect handlers. To this extend, one might want to try and identify 'safe' handlers, the ones that do not break the equivalence. An investigation along these lines is done in [50], where safety of handlers is studied with respect to equational theories. Secondly, one could use handlers to implement modalities into the language itself, e.g., various operators from concurrency can be defined using handlers [1, 22]. Considering these two directions, adding handlers may be a fruitful pursuit for the future.

Bibliography

[1] Martin Abadi and Gordon Plotkin. A model of cooperative threads. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, pages 29–40, 2009.

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

[3] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51(1–2):1–77, 1991.

[4] Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 334–344, 1998.

[5] Danel Ahman, Catalin Hritcu, Kenji Maillard, Guido Martínez, Gordon Plotkin, Jonathan Protzenko, Aseem Rastogi, and Nikhil Swamy. Dijkstra monads for free. In *44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 515–529, 2017.

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

[7] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in logic and the foundations of mathematics 103. Amsterdam : North-Holland, 1984.

[8] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *Journal of Logical and Algebraic Methods in Programming*, 84:108–123, 2015.

[9] Nick Benton, Martin Hofmann, and Vivek Nigam. Abstract effects and proof-relevant logical relations. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14*, pages 619–632, 2014.

[10] Jean-Philippe Bernardy, Patrik Jansson, and Koen Claessen. Testing polymorphic properties. *Lecture Notes in Computer Science*, 6012, 2010. In: Gordon A.D. (eds) Programming Languages and Systems. ESOP 2010.

[11] Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal of Computing*, 1978.

[12] Raphaëlle Crubillé and Ugo Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi. *Lecture Notes in Computer Science*, 8410, 2014.

[13] Ugo Dal Lago and Francesco Gavazzo. Effectful normal form bisimulation. In Luís Caires, editor, *Programming Languages and Systems*, pages 263–292, 2019.

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

[15] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs (long version). *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, 49, 2013.

[16] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. *ACM SIGPLAN Notices*, 49(1):297–308, 2014.

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

[18] Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the [1]-calculus. *Formal Description of Programming Concepts*, pages 193–217, 1986.

[19] Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103:235–271, 1992.

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

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

[22] Rob van Glabbeek and Gordon Plotkin. *On CSP and the Algebraic Theory of Effects*, pages 333–369. Springer London, London, 2010.

[23] Sergey Goncharov and Lutz Schröder. A relatively complete generic Hoare logic for order-enriched effects. In *Proceedings of the 28th Annual Symposium on Logic in Computer Science (LICS 2013)*, pages 273–282. IEEE, 2013.

[24] Andrew D. Gordon. *Functional Programming and Input/Output*. PhD thesis, University of Cambridge, 1994.

[25] Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. Edinburgh LCF: A mechanized logic of computation. *Lecture Notes in Computer Science*, 1979.

[26] Ichiro Hasuo. Generic weakest precondition semantics from monads enriched with order. *Theoretical Computer Science*, 604(C):2–29, 2015.

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

[28] Charles A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[29] Martin Hofmann. Logical relations and nondeterminism. In Rocco De Nicola and Rolf Hennicker, editors, *Software, Services, and Systems. Lecture Notes in Computer Science*, volume 8950, pages 62–74. Springer International Publishing, Cham, 2015.

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

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

[32] Martin Hyland and Luke Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163:285–408, 2000.

[33] Martin Hyland, Gordon Plotkin, and John Power. Combining effects: Sum and tensor. *Theoretical Computer Science*, 357(1):70 – 99, 2006.

[34] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge University Press, 2016.

[35] Patricia Johann, Alex Simpson, and Janis Voigtländer. A generic operational metatheory for algebraic effects. In *Proceedings of Logic in Computer Science (LICS'10)*, pages 209–218, 2010.

[36] Clair Jones. *Probabilistic Non-determinism*. PhD thesis, University of Edinburgh, 1989.

[37] Shin-ya Katsumata. Relating computational effects by $\top\top$-lifting. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 174–185, 2011.

[38] Vasileios Koutavas, Paul Blain Levy, and Eijiro Sumii. From applicative to environmental bisimulation. *Electronic Notes in Theoretical Computer Science*, 276:215 – 235, 2011. Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics.

[39] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333 – 354, 1983. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

[40] Søren B. Lassen. Action semantics reasoning about functional programs. *Mathematical Structures in Computer Science*, 7:557–589, 1997. Special issue dedicated to the Workshop on Logic, Domains, and Programming Languages, 1995.

[41] Søren B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, University of Aarhus, 1998.

[42] Paul B. Levy. *Call-By-Push-Value*. PhD thesis, Queen Mary and Westfield College, University of London, 2001.

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

[44] Paul B. Levy. Infinitary howe's method. *Electronic Notes in Theoretical Computer Science*, 164(1):85–104, 2006.

[45] Paul B. Levy. Amb breaks well-pointedness, ground amb doesn't. *Electronic Notes in Theoretical Computer Science*, 173:221 – 239, 2007. Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIII).

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

[47] Paul B. Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Information and Computation*, 185(2):182–210, 2003.

[48] John R. Longley. *Realizability Toposes and Language Semantics*. PhD thesis, University of Endinburgh, 1994.

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

[50] Žiga Lukšič and Matija Pretnar. Local algebraic effect theories, 2019. Submitted to the Journal of Functional Programming.

[51] Kenji Maillard, Danel Ahman, Robert Atkey, Guido Martínez, Catalin Hritcu, Exequiel Rivas, and Éric Tanter. Dijkstra monads for all. In *24th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, 2019.

[52] Gregory Malecha, Greg Morrisett, and Ryan Wisnesky. Trace-based verification of imperative programs with I/O. *Journal of Symbolic Computation*, 46(2):95–118, 2011.

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

[54] Johannes Marti and Yde Venema. Lax extensions of coalgebra functors and their logic. *J. Comput. Syst. Sci.*, 81(5):880–900, 2015.

[55] John McCarthy. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, pages 33–70, 1963.

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

[57] Rasmus E. Møgelberg and Alex Simpson. A logic for parametric polymorphism with effects. In *Types*, volume 4941, pages 142–156, 2007.

[58] Rasmus E. Møgelberg and Alex Simpson. Relational Parametricity for Computational Effects. *Logical Methods in Computer Science*, Volume 5, Issue 3, 2009.

[59] Robin Milner. A formal notion of simulation between programs, 1970. Memo 14, Computers and Logic Research Croup.

[60] Robin Milner. An algebraic definition of simulation between programs. In *International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 481–489, 1971.

[61] Robin Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[62] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, Heidelberg, 1982.

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

[64] James H. Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, Massachusetts Institute of Technology, 1969.

[65] Till Mossakowski, Lutz Schröder, and Sergey Goncharov. A generic complete dynamic logic for reasoning about purity and effects. *Formal Aspects of Computing*, 22(3–4):363–384, 2010.

[66] Aleksandar Nanevski and Gregory Morrisett. Dependent type theory of stateful higher-order functions. *Technical Report TR-24-05, Harvard University*, 2005.

[67] Aleksandar Nanevski, Gregory Morrisett, and Lars Birkedal. Hoare type theory, polymorphism and separation. *Journal of Functional Programming*, 18:865–911, 2008.

[68] Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In Laurent Fribourg, editor, *Computer Science Logic*, pages 1–19, 2001.

[69] Peter W. O'Hearn, Hongseok Yang, and John C. Reynolds. Separation and information hiding. *ACM SIGPLAN Notices*, 39(1):268–280, 2004.

[70] Luke Ong. Non-determinism in a functional setting. *Symposium on Logic in Computer Science, Montreal.*, 8:275–286, 1993.

[71] David Park. Concurrency and automata on infinite sequences. *Lecture Notes in Computer Science*, 154:561–572, 1981.

[72] Willem Penninckx, Bart Jacobs, and Frank Piessens. Sound, modular and compositional verification of the Input/Output behavior of programs. In Jan Vitek, editor, *Programming Languages and Systems*, pages 158–182, 2015.

[73] Andrew M. Pitts. Evaluation logic. In *Proceedings of 4th Higher Order Workshop*, pages

162–189, 1991.

[74] Andrew M. Pitts. Existential types: Logical relations and operational equivalence. In KG Larsen, S Skyum, and G Winskel, editors, *Proceedings of Automata, Languages and Programming, 25th International Colloquium, ICALP'98*, volume 1443, page 309–326, 1998.

[75] Andrew M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10:321–359, 2000.

[76] Andrew M. Pitts. Typed operational reasoning. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7, pages 245–289. The MIT Press, London, UK, 2005.

[77] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, New York, NY, USA, 2013.

[78] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[79] Gordon D. Plotkin. A powerdomain construction. *Siam J. Comput.*, 5(3):452–487, 1976.

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

[81] Gordon D. Plotkin. Domains, 1983. Course notes.

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

[83] Gordon D. Plotkin and John Power. Notions of computation determine monads. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, pages 342–356, 2002.

[84] Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11:69–94, 2003.

[85] Gordon D. Plotkin and Matija Pretnar. A logic for algebraic effects. In *Proceedings of Logic in Computer Science*, pages 118–129, 2008.

[86] Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4):1–36, 2013.

[87] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57, 1977.

[88] Yann Régis-Gianas and François Pottier. A Hoare logic for call-by-value functional programs. In *Proceedings of the 9th International Conference on Mathematics of Program Construction*, MPC '08, pages 305–335, 2008.

[89] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, LICS '02, pages 55–74, 2002.

[90] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, Cambridge, UK, 2011.

[91] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5:1–5:69, 2011.

[92] Davide Sangiorgi and Jan Rutten, editors. *Advanced Topics in Bisimulation and Coinduction*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 2011.

[93] Dana Scott. Data types as lattices. *SIAM Journal on Computing*, 5:522–587, 1976.

[94] Dana Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121, 1993.

[95] Alex Simpson and Niels Voorneveld. Behavioural equivalence via modalities for algebraic effects. In *Programming Languages and Systems (ESOP 2018)*, pages 300–326, 2018.

[96] Alex Simpson and Niels Voorneveld. Behavioural equivalence via modalities for algebraic effects. *ACM Trans. Program. Lang. Syst.*, 42, 2020. 45 pages.

[97] Michael B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16:23–36, 1978.

[98] Thomas Streicher. *Domain-theoretic Foundations of Functional Programming.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2006.

[99] Eijiro Sumii and Benjamin C. Pierce. A bisimulation for type abstraction and recursion. *ACM SIGPLAN Notices*, 40(1):63–74, 2005.

[100] Nikhil Swamy, Catalin Hritcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoué, and Santiago Zanella-Béguelin. Dependent types and multi-monadic effects in F*. In *43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 256–270, 2016.

[101] Nikhil Swamy, Joel Weinberger, Cole Schlesinger, Juan Chen, and Benjamin Livshits. Verifying higher-order programs with the Dijkstra monad. *ACM SIGPLAN Notices*, 48(6):387–398, 2013.

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

[103] Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *Proc. ACM Program. Lang.*, 3(POPL):36:1–36:29, 2019.

[104] Niels Voorneveld. Non-deterministic effects in a realizability model. *Electronic Notes in Theoretical Computer Science*, 336:299 – 314, 2018. The Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII).

[105] Niels Voorneveld. Quantitative logics for equivalence of effectful programs. In *Proc. of MFPS XXXV (Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics)*, ENTCS. Elsevier, 2019. To appear.

[106] Philip Wadler. Monads for functional programming. *Program Design Calculi*, 118:233–264, 1993.

[107] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38 – 94, 1994.

Razširjeni povzetek v slovenskem jeziku

Disertacija preučuje pojem enakovrednosti programov za funkcijski programski jezik z algebrajskimi učinki in splošno rekurzijo, ki uporablja klic po naloženi vrednosti (call-by-push-value oz. na kratko cbpv). Osredotočamo se predvsem na *vedenjsko ekvivalenco*, pri čemer je obnašanje programa določeno z zbirko formul, ki so odvisne od učinkov. Dva programa istega tipa imamo za enakovredna, če zadoščata istim formulam, ki izražajo vedenjske lastnosti programov. Vpliv učinkov na izračun programa opišemo z modalnostmi. Da bi učinke lažje kombinirali, logiko posplošimo v kvantitativno logiko s kvantitativnimi modalnostmi.

Eden glavnih prispevkov disertacije je določitev pogojev na modalnostih, pri katerih je vedenjska ekvivalenca, ki jo inducira logika, kongruenca. To pomeni, da programi ne ločijo med enakovrednimi izrazi. Pogoji vključujejo relevantne lastnosti zveznosti, ki se tičejo izračunov programov, ki se potencialno ne končajo, in pojem *razcepnosti*, ki obravnava zaporedje izračunov (zaporedno izvajanje programov). Pod temi pogoji vedenjska ekvivalenca sovpada s še enim pojmom ekvivalence programov, aplikativno bipodobnostjo, za katero je kongruenca dokazana preko Howejeve metode.

Disertacija prispeva tudi načina, kako lahko rezultate uporabimo na nekaterih primerih kombinacij učinkov in kako modalnosti inducirajo prave enačbe med programi z učinki. Preučimo tudi nekatere različice logike in raziščemo situacije, v katerih te različice logike inducirajo isto ekvivalenco.

Poglavje 2: Jezik in operacijska semantika

Ogledamo si funkcijski programski jezik [42, 43] z algebrajskimi učinki in splošno rekurzijo, ki uporablja pristop cbpv. Programski jezik ima *tipe vrednosti* in *tipe izračunov*, ki so rekurzivno podani z:

$$\mathbf{A}, \mathbf{B} ::= \mathbf{U}\,\underline{\mathbf{C}} \mid \mathbf{1} \mid \mathbf{N} \mid \mathbf{\Sigma}_{i\in I}\,\mathbf{A}_i \mid \mathbf{A} \times \mathbf{B},$$
$$\underline{\mathbf{C}}, \underline{\mathbf{D}} ::= \mathbf{F}\,\mathbf{A} \mid \mathbf{A} \to \underline{\mathbf{C}} \mid \mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}}_i.$$

Sintaksa izrazov in pravila za tipe so podani na sliki 2.1, kjer je Γ kontekst, ki vsebuje spremenljivke tipov vrednosti. Pišemo $\mathit{Terms}(\underline{\underline{\mathbf{E}}})$ za zaprte izraze tipa $\underline{\underline{\mathbf{E}}}$.

Izraz lahko evalviramo in tako reduciramo na *končni izraz* oblike $\mathsf{return}(V)$, $\lambda x.\,\underline{M}$ ali $\langle \underline{M_i} \mid i \in I \rangle$. Takšna evalvacija lahko traja večno ali kaže sprožanje učinkov.

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

$$\frac{\Gamma \vdash V : \mathbf{N} \qquad \Gamma \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}} \qquad \Gamma, x : \mathbf{N} \mathrel{\underline{\vdash}} \underline{N} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{case}\ V\ \mathsf{of}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} : \underline{\mathbf{C}}}$$

$$\frac{}{\Gamma, x : \mathbf{A}, \Gamma' \vdash x : \mathbf{A}} \qquad \frac{\Gamma \vdash V : \mathbf{A} \qquad \Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{let}\ x\ \mathsf{be}\ V.\underline{M} : \underline{\mathbf{C}}} \qquad \frac{\Gamma \vdash V : \mathbf{A}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{return}(V) : \mathbf{F}\,\mathbf{A}}$$

$$\frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} : \mathbf{F}\,\mathbf{A} \qquad \Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{N} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \underline{M}\ \mathsf{to}\ x.\underline{N} : \underline{\mathbf{C}}} \qquad \frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \vdash \mathsf{thunk}(\underline{M}) : \mathbf{U}\,\underline{\mathbf{C}}} \qquad \frac{\Gamma \vdash V : \mathbf{U}\,\underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{force}(V) : \underline{\mathbf{C}}}$$

$$\frac{\Gamma, x : \mathbf{A} \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \lambda x.\,\underline{M} : \mathbf{A} \to \underline{\mathbf{C}}} \qquad \frac{\Gamma \vdash V : \mathbf{A} \qquad \Gamma \mathrel{\underline{\vdash}} \underline{M} : \mathbf{A} \to \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \underline{M}\ V : \underline{\mathbf{C}}}$$

$$\frac{\Gamma \vdash V : \mathbf{A}_j}{\Gamma \vdash (j, V) : \mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i} \qquad \frac{\Gamma \vdash V : \mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i \qquad \Gamma, x : \mathbf{A}_i \mathrel{\underline{\vdash}} \underline{M}_i : \underline{\mathbf{C}} \ \text{ za vsako } \ i \in I}{\Gamma \mathrel{\underline{\vdash}} \mathsf{pm}\ V\ \mathsf{as}\ \{\ldots, (i.x).\underline{M}_i, \ldots\} : \underline{\mathbf{C}}}$$

$$\frac{\Gamma \vdash V : \mathbf{A} \qquad \Gamma \vdash W : \mathbf{B}}{\Gamma \vdash (V, W) : \mathbf{A} \times \mathbf{B}} \qquad \frac{\Gamma \vdash V : \mathbf{A} \times \mathbf{B} \qquad \Gamma, x : \mathbf{A}, y : \mathbf{B} \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{pm}\ V\ \mathsf{as}\ (x, y).\underline{M} : \underline{\mathbf{C}}}$$

$$\frac{\Gamma \mathrel{\underline{\vdash}} \underline{M}_i : \underline{\mathbf{C}}_i \ \text{ za vsak } \ i \in I}{\Gamma \mathrel{\underline{\vdash}} \langle \underline{M}_i \mid i \in I \rangle : \mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i} \qquad \frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} : \mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i \qquad j \in I}{\Gamma \mathrel{\underline{\vdash}} \underline{M}\ j : \underline{\mathbf{C}}_j} \qquad \frac{\Gamma \mathrel{\underline{\vdash}} \underline{M} : \mathbf{U}\,\underline{\mathbf{C}} \to \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{fix}(\underline{M}) : \underline{\mathbf{C}}}$$

Slika 2.1: Pravila za tipe.

$$\frac{\Gamma \vdash V_i : \mathbf{N} \text{ za vsak } 1 \le i \le n \quad \Gamma \mathrel{\underline{\vdash}} \underline{M}_i : \underline{\mathbf{C}} \text{ za vsak } 1 \le i \le m}{\Gamma \mathrel{\underline{\vdash}} \mathsf{op}(V_1, \ldots, V_n, \underline{M}_1, \ldots, \underline{M}_m) : \underline{\mathbf{C}}}\ (\mathsf{op} : \mathbf{N}^n \times \underline{\alpha}^m \to \underline{\alpha}) \in \Sigma$$

$$\frac{\Gamma \vdash V_i : \mathbf{N} \ \text{ za vsak } \ 1 \le i \le n \qquad \Gamma, x : \mathbf{N} \mathrel{\underline{\vdash}} \underline{M} : \underline{\mathbf{C}}}{\Gamma \mathrel{\underline{\vdash}} \mathsf{op}(V_1, \ldots, V_n, x \mapsto \underline{M}) : \underline{\mathbf{C}}}\ (\mathsf{op} : \mathbf{N}^n \times \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}) \in \Sigma$$

Slika 2.2: Pravila za tipe za učinke.

V jezik vnesemo *algebrajske učinke* tako, da ga razširimo z množico učinkovnih operacij, ki so zbrane v *signaturi učinkov* Σ. Pravila za tipe tovrstne operacije so podana na sliki 2.2.

Operacijska semantika programskih izrazov je podana v razdelkih 2.1 in 2.2, izrazi tipov izračunov se evalvirajo v *drevesa učinkov*. Neformalno izračun, ki je izraz tipa izračuna, evalviramo v drevo, čigar vozlišča opisujejo možne pojavitve učinkovne operacije.

Definicija 2.2.1. *Drevo učinkov (effect tree)* (v nadaljevanju *drevo*) TX nad množico X, določeno s signaturo učinkov Σ, je drevo morda neskončne globine, čigar vozlišča so označena z učinkovnimi operacijami in listi bodisi z rezultatom $\langle x \rangle$, kjer je $x \in X$, ali s simbolom $\bot$ za divergenco.

Množico TX opremimo z ω-polno urejenostjo takoj po definiciji 2.2.1 v osrednjem delu disertacije. Še več, T dodamo strukturo monade, vključno z enotami $\eta : X \to TX$ in preslikavami množenja $\mu : TTX \to TX$ za vsako množico X.

Redukcijo na drevo nato definiramo kot preslikavo $|-| : \mathit{Terms}(\underline{\mathbf{C}}) \to T(\mathit{Terms}(\underline{\mathbf{C}}))$, ki slika izračune v drevesa učinkov. Pri tem uporabimo funkcijo za aproksimacijo dreves iz definicije 2.2.3.

Neformalno je ta aproksimativna funkcija definirana na naslednji način. Izračun evalviramo, dokler se evalvacijski proces, ki je deterministično, ne konča (če se to ne zgodi, je drevo enako $\bot$). Če pa se evalvacijski proces konča v konfiguraciji, ki vsebuje učinkovno operacijo $\mathsf{op}(\dots)$, v drevesu ustvarimo notranje vozlišče, ki ga označimo op. Nato nadaljujemo z izgradnjo dreves za podizračune operacije $\mathsf{op}(\dots)$ kot otroke danega vozlišča tako, da proces ponovimo.

Primeri učinkov

V razdelku 2.3 obravnavamo mnoge primere učinkov, ki so v literaturi morda standardni, a bodo služili kot tipični primeri v disertaciji.

Napaka: Vzemimo množico obvestil o napaki Err in za vsako obvestilo $e \in \mathsf{Err}$ dodajmo učinkovno operacijo $\mathsf{raise}_e() : \mathbf{1} \to \underline{\alpha}$. Torej je naša signatura enaka $\Sigma_{\mathrm{er}} := \{\mathsf{raise}_e() : \mathbf{1} \to \underline{\alpha} \mid e \in \mathsf{Err}\}$, kjer izračun $\mathsf{raise}_e()$ prekine evalvacijo in prikaže e kot obvestilo o napaki.

Nedeterminizem: Naj bo $\mathsf{or}(-,-) : \underline{\alpha}^2 \to \underline{\alpha}$ binarna operacija izbire, ki poda dve možnosti nadaljevanja izračuna. Če podamo dva izračuna $\underline{M}$ in $\underline{N}$, je $\mathsf{or}(\underline{M}, \underline{N})$ izračun, ki nadaljuje evalvacijo bodisi kot $\underline{M}$ bodisi kot $\underline{N}$. Katera od obeh možnosti bo obveljala, ne vemo.

Verjetnost: Verjetnostno izbiro implementiramo z uporabo binarne operacije izbire $\mathsf{pr}(-,-) : \underline{\alpha}^2 \to \underline{\alpha}$, ki vrne dve možnosti za nadaljevanje izračuna. Naša signatura učinkov je $\Sigma_{\mathrm{pr}} := \{\mathsf{pr}(-,-) : \underline{\alpha}^2 \to \underline{\alpha}\}$. V tem primeru je izbira nadaljevanja verjetnostna, saj ima $\mathsf{pr}(\underline{M}, \underline{N})$ enako verjetnost za nadaljevanje z izračunom $\underline{M}$ ali $\underline{N}$.

Globalni pomnilnik: Oglejmo si primer, ko imajo programi lahko interakcije z nekim globalnim pomnilnikom ter lahko iz njega berejo in vanj pišejo. Naj bo Loc množica lokacij za shranjevanje naravnih števil. Za vsako lokacijo $l \in \mathsf{Loc}$ imamo dve učinkovni operaciji $\mathsf{lookup}_l(-) : \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$ in $\mathsf{update}_l(-;-) : \mathbf{N} \times \underline{\alpha} \to \underline{\alpha}$, $\Sigma_{\mathrm{gs}} := \{\mathsf{lookup}_l(-), \mathsf{update}_l(-;-) \mid l \in \mathsf{Loc}\}$. Izračun $\mathsf{lookup}_l(x \mapsto \underline{M})$ prebere število na lokaciji l in ga zamenja za x v $\underline{M}$, izračun $\mathsf{update}_l(\overline{n}; \underline{M})$ pa shrani n na lokaciji l in nadaljuje z izračunom $\underline{M}$.

Vhod/Izhod: Vzemimo operaciji $\mathsf{read}(-) : \underline{\alpha}^{\mathbf{N}} \to \underline{\alpha}$, ki prebere število z vhoda in ga poda kot argument nekemu izračunu, in $\mathsf{write}(-;-) : \mathbf{N} \times \underline{\alpha} \to \underline{\alpha}$, ki izpiše število (prvi argument) na zaslon in nato nadaljuje z izračunom iz drugega argumenta. Skupaj tvorita signaturo $\Sigma_{\mathrm{io}} := \{\mathsf{read}(-) : \underline{\alpha}^{\mathbf{N}}, \mathsf{write}(-;-) : \mathbf{N} \to \underline{\alpha} \times \underline{\alpha} \to \underline{\alpha}\}$.

$$\frac{n \in \mathbb{N}}{\{n\} \in \mathit{Form}(\mathbf{N})} \qquad \frac{\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})}{\langle \underline{\phi} \rangle \in \mathit{Form}(\mathbf{U}\,\underline{\mathbf{C}})} \qquad \frac{j \in I \qquad \phi \in \mathit{Form}(\mathbf{A}_j)}{(j, \phi) \in \mathit{Form}(\mathbf{\Sigma}_{i \in I}\, \mathbf{A}_i)}$$

$$\frac{\phi \in \mathit{Form}(\mathbf{A})}{\pi_0(\phi) \in \mathit{Form}(\mathbf{A} \times \mathbf{B})} \qquad \frac{\phi \in \mathit{Form}(\mathbf{B})}{\pi_1(\phi) \in \mathit{Form}(\mathbf{A} \times \mathbf{B})}$$

$$\frac{V \in \mathit{Terms}(\mathbf{A}) \qquad \underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}})}{(V \mapsto \underline{\phi}) \in \mathit{Form}(\mathbf{A} \to \underline{\mathbf{C}})} \qquad \frac{o \in \mathcal{O} \qquad \phi \in \mathit{Form}(\mathbf{A})}{o(\phi) \in \mathit{Form}(\mathbf{F}\,\mathbf{A})}$$

$$\frac{\underline{\phi} \in \mathit{Form}(\underline{\mathbf{C}}_j)}{(j \mapsto \underline{\phi}) \in \mathit{Form}(\mathbf{\Pi}_{i \in I}\, \underline{\mathbf{C}}_i)} \qquad \frac{X \subseteq_{\text{countable}} \mathit{Form}(\underline{\underline{\mathbf{E}}})}{\bigvee X \in \mathit{Form}(\underline{\underline{\mathbf{E}}})}$$

$$\frac{X \subseteq_{\text{countable}} \mathit{Form}(\underline{\underline{\mathbf{E}}})}{\bigwedge X \in \mathit{Form}(\underline{\underline{\mathbf{E}}})} \qquad \frac{\underline{\phi} \in \mathit{Form}(\underline{\underline{\mathbf{E}}})}{\neg(\underline{\phi}) \in \mathit{Form}(\underline{\underline{\mathbf{E}}})}$$

Slika 3.1: Konstruktorji formul.

Časomer: Definiramo izračunljivo množico racionalnih časovnih korakov $\mathsf{Inc} \subset \mathbb{Q}$, kjer imamo za vsak $c \in \mathsf{Inc}$ operacijo $\mathsf{tick}_c(-) : \underline{\alpha} \to \underline{\alpha}$. Naša signatura učinkov je $\Sigma_{\text{ti}} := \{\mathsf{tick}_c(-) : \underline{\alpha} \to \underline{\alpha} \mid c \in \mathsf{Inc}\}$, kjer je $\mathsf{tick}_c(\underline{M})$ izračun, pri katerem evalvacijo odložimo za čas c, nato pa nadaljujemo z izračunom $\underline{M}$.

Poglavje 3: Vedenjska ekvivalenca

V tem poglavju definiramo pojem ekvivalence med programi, ki nas zanima, tj. *vedenjsko ekvivalenco*. Najprej induktivno podamo logiko vedenjskih lastnosti jezika tako, da za vsak tip $\underline{\underline{\mathbf{E}}}$ definiramo množico formul $\mathit{Form}(\underline{\underline{\mathbf{E}}})$. Pravila za to definicijo so podana na sliki 3.1. Z $\mathcal{V}$ označimo množico vseh formul in z $\mathcal{V}^+$ množico tistih formul, ki ne vsebujejo negacij '$\neg$'.

Naj bo $\underline{\underline{\phi}}$ formula in $\underline{P}$ izraz. Dejstvo, da ima $\underline{P}$ lastnost $\underline{\underline{\phi}}$, pišemo kot $\underline{P} \models \underline{\underline{\phi}}$. Za definicijo logike na sliki 3.1 je ključnega pomena, da smo določili množico modalnosti $\mathcal{O}$, kar je tudi temeljni prispevek te disertacije.

Modalnosti za učinke

Množico modalnosti $\mathcal{O}$ uporabljamo, da določimo obnašanje z učinki. Vsaka modalnost $o \in \mathcal{O}$ poda podmnožico $[\![o]\!]$ dreves učinkov z enotskim tipom $T(\{*\})$. Naj bo ϕ formula in $|\underline{M}|[\models \phi]$ drevo, ki ga dobimo, ko zamenjamo vse liste $\langle \mathsf{return}(V) \rangle$ drevesa $|\underline{M}|$ z $\langle * \rangle$, če velja $V \models \phi$, in z $\bot$, če velja $V \not\models \phi$. Potem velja $\underline{M} \models o(\phi)$, če je $|\underline{M}|[\models \phi]$ v množici $[\![o]\!]$. Oglejmo si nekaj neformalnih opisov primerov modalnosti.

Napaka, zaznava obvestila: $\mathcal{O}_{\text{er}} = \{\downarrow\} \cup \{\mathsf{E}_e \mid e \in \mathsf{Err}\}$, kjer je $[\![\downarrow]\!] := \{\langle * \rangle\}$ in $[\![\mathsf{E}_e]\!] = \{\mathsf{raise}_e\}$. Če $\underline{M}$ vrne vrednost, ki zadošča ϕ, velja $\underline{M} \models\!\downarrow (\phi)$, in če $\underline{M}$ sproži obvestilo o napaki e, velja $\underline{M} \models \mathsf{raise}_e(\phi)$.

Nedeterminizem, lahkó in móra: Definirajmo $\mathcal{O}_{\text{nd}} = \{\Diamond, \Box\}$, kjer je $[\![\Diamond]\!] = \{t \mid t$ ima neki list z oznako $\langle * \rangle\}$ in $[\![\Box]\!] = \{t \mid t$ je končno drevo in vsak list je označen z $\langle * \rangle\}$. Če $\underline{M}$ lahko vrne vrednost, ki zadošča formuli ϕ, velja $\underline{M} \models \Diamond(\phi)$, in če $\underline{M}$ mora vrniti vrednost, ki zadošča formuli ϕ, velja $\underline{M} \models \Box(\phi)$. Za $\mathcal{O}$ imamo tri možnosti: $\{\Diamond, \Box\}$ za nevtralni nedeterminizem, $\{\Diamond\}$ za angelski nedeterminizem in $\{\Box\}$ za demonski nedeterminizem.

Verjetnost, pričakovana zadostitev formul: Definiramo $\mathcal{O}_{\text{pr}} = \{\mathsf{P}_{>q} \mid q \in \mathbb{Q}, 0 \leq q \leq 1\}$, kjer $[\![\mathsf{P}_{>q}]\!]$ poda množico dreves $t \in T(\{*\})$, za katera je verjetnost, da pridemo do lista, označenega z $\langle * \rangle$, večja od q. Če je verjetnost, da $\underline{M}$ vrne vrednost, ki zadošča ϕ, večja od q, velja $\underline{M} \models \mathsf{P}_{>q}(\phi)$.

Globalni pomnilnik, modalnosti za prehod stanja: Definiramo množico globalnih stanj $\mathsf{State} = \mathbb{N}^{\mathsf{Loc}}$, kjer za $s \in \mathsf{State}$, $s(l) = m$ pomeni, da je število m shranjeno na lokaciji l. Modalnosti podamo z množico $\mathcal{O}_{\text{gs}} = \{(s \rightarrowtail r) \mid s, r \in \mathsf{State}\}$, kjer velja $\underline{M} \models (s \rightarrowtail r)(\phi)$, če bo, ko evalviramo $\underline{M}$ z globalnim pomnilnikom v začetnem stanju s, izraz $\underline{M}$ vrnil vrednost, ki zadošča ϕ, in bo nato globalni pomnilnik v stanju r.

Vhod/izhod, sledi: Definiramo *vhodno/izhodno sled* kot besedo w nad abecedo $\{?n \mid n \in \mathbb{N}\} \cup \{!n \mid n \in \mathbb{N}\}$. Pri tem $?n$ opisuje dejstvo, da je program kot vhodni podatek od uporabnika dobil n, in $!n$ dejstvo, da je program uporabniku vrnil n kot izhodni podatek. Kot množico modalnosti vzamemo $\mathcal{O}_{\text{io}} = \{\langle w \rangle\!\downarrow, \langle w \rangle_{\dots} \mid w$ vhodno/izhodna sled$\}$. Če je w možna sled za izraz $\underline{M}$, velja $\underline{M} \models \langle w \rangle_{\dots}(\phi)$. Če je w možna sled za izraz $\underline{M}$, ki vrne vrednost, ki zadošča formuli ϕ, potem velja $\underline{M} \models \langle w \rangle\!\downarrow(\phi)$.

Časomer, potrebni čas: Naj bo $\mathcal{O}^{\downarrow}_{\text{ti}} = \{\mathsf{C}_{\leq q} \mid q \in \mathbb{Q}_{\geq 0}\}$ množica modalnosti, kjer velja $\underline{M} \models \mathsf{C}_{\leq q}(\phi)$, če se evalvacija izraza $\underline{M}$ konča, vrne vrednost, ki zadošča formuli ϕ, in je bila odložena za največ q časa.

Formalno definirajmo, kaj pomeni, da programi zadoščajo formulam. Za vsak tip $\underline{\underline{\mathbf{E}}}$ imamo zadostitveno relacijo $\models \subseteq \mathit{Terms}(\underline{\underline{\mathbf{E}}}) \times \mathit{Form}(\underline{\underline{\mathbf{E}}})$, ki jo definiramo z naslednjimi pravili:

$$
\begin{array}{rcl@{\qquad}rcl}
V \models \{n\} & \iff & V = \overline{n}, & V \models \langle \underline{\underline{\phi}} \rangle & \iff & \mathsf{force}(V) \models \underline{\underline{\phi}}. \\
(i, V) \models (j, \phi) & \iff & i = j \wedge V \models \phi, & (V, W) \models \pi_0(\phi) & \iff & V \models \phi. \\
(V, W) \models \pi_1(\phi) & \iff & W \models \phi, & \underline{M} \models (V \mapsto \underline{\underline{\phi}}) & \iff & \underline{M}\ V \models \underline{\underline{\phi}}. \\
\underline{M} \models o(\phi) & \iff & |\underline{M}|[\models \phi] \in [\![o]\!], & \underline{M} \models (j \mapsto \underline{\underline{\phi}}) & \iff & \underline{M}\ j \models \underline{\underline{\phi}}. \\
\underline{\underline{P}} \models \bigvee X & \iff & \exists \underline{\underline{\phi}} \in X.\ \underline{\underline{P}} \models \underline{\underline{\phi}}, & \underline{\underline{P}} \models \bigwedge X & \iff & \forall \underline{\underline{\phi}} \in X.\ \underline{\underline{P}} \models \underline{\underline{\phi}}. \\
\underline{\underline{P}} \models \neg(\underline{\underline{\phi}}) & \iff & \neg(\underline{\underline{P}} \models \underline{\underline{\phi}}). & & & \\
\end{array}
$$

Logične šibke urejenosti

Logiko vpeljemo z namenom, da določimo šibko urejenost na izrazih. Naj bosta $\mathcal{L}$ podmnožica $\mathcal{L} \subseteq \mathcal{V}$ in $\underline{\underline{\mathbf{E}}}$ tip. Pišemo $Form(\underline{\underline{\mathbf{E}}})_{\mathcal{L}}$ za $Form(\underline{\underline{\mathbf{E}}}) \cap \mathcal{L}$. Vsaka podmnožica logike definira naravno ekvivalenco, kjer sta dva izraza v relaciji, če zadoščata istim formulam. To ekvivalenco lahko podamo z logično šibko urejenostjo tako, da je izraz $\underline{\underline{R}}$ nad izrazom $\underline{\underline{P}}$, če $\underline{\underline{R}}$ zadošča vsem formulam, ki jim zadošča tudi izraz $\underline{\underline{P}}$.

Definicija 3.3.2. Za vsako podmnožico logike $\mathcal{L} \subseteq \mathcal{V}$ definiramo *logično šibko urejenost* $\sqsubseteq_{\mathcal{L}}$, tako da velja:

$$\forall \underline{\underline{P}}, \underline{\underline{R}} : \underline{\underline{\mathbf{E}}}, \quad \underline{\underline{P}} \sqsubseteq_{\mathcal{L}} \underline{\underline{R}} \iff (\forall \underline{\underline{\phi}} \in Form(\underline{\underline{\mathbf{E}}})_{\mathcal{L}},\ \underline{\underline{P}} \models \underline{\underline{\phi}} \ \Rightarrow\ \underline{\underline{R}} \models \underline{\underline{\phi}}).$$

Splošna vedenjska šibka urejenost $\sqsubseteq$ je logična šibka urejenost na $\mathcal{V}$, medtem ko je *pozitivna vedenjska šibka urejenost* $\sqsubseteq^{+}$ logična šibka urejenost na $\mathcal{V}^{+}$. Za ekvivalence, ki jih inducirajo šibke urejenosti $\equiv_{\mathcal{L}}$, $\equiv$ in $\equiv^{+}$, pišemo $\sqsubseteq_{\mathcal{L}}$, $\sqsubseteq$, in $\sqsubseteq^{+}$.

Splošna vedenjska šibka urejenost $\sqsubseteq$ je simetrična, torej je enaka splošni vedenjski ekvivalenci $\equiv$.

Naslednji rezultat je eden glavnih prispevkov te disertacije. Lastnosti *razcepnost* in *Scottova odprtost* sta definirani kasneje.

Izrek 3.3.8 (Izrek o združljivosti)**.** Naj bo $\mathcal{O}$ razcepna množica Scottovo odprtih modalnosti. Tedaj sta odprti razširitvi polne vedenjske ekvivalence $\equiv$ in pozitivne vedenjske šibke urejenosti $\sqsubseteq^{+}$ združljivi.

Intuitivno to pomeni, da so ekvivalence kongruence in da so šibke urejenosti šibke kongruence. Pojasnimo še lastnosti, ki nastopata v izreku.

Definicija 3.3.10. Pravimo, da je modalnost $o \in \mathcal{O}$ *Scottovo odprta*, če je $[\![o]\!]$ odprta podmnožica v Scottovi topologiji na $T(\{*\})$, tj. če je $t_1 \leq t_2 \leq \ldots$ naraščajoča veriga v $T(\{*\})$, potem velja $\bigsqcup_i t_i \in [\![o]\!] \iff \exists n \in \mathbb{N}. t_n \in [\![o]\!]$.

Razcepnost je lastnost preslikave množenja $\mu : T(T(\{*\})) \to T(\{*\})$, ki pripada strukturi monade. Neformalno je vloga razcepnosti zagotoviti, da preslikava množenja ohranja pozitivno šibko urejenost $\sqsubseteq^{+}$. V formalni definiciji uporabimo abstraktni relaciji $\preccurlyeq \subseteq T(\{*\}) \times T(\{*\})$ in $\preccurlyeq \subseteq T(T(\{*\})) \times T(T(\{*\}))$, ki ju definiramo preko modalnosti iz množice $\mathcal{O}$. Formalni definiciji teh relacij sta podani v razdelku 3.3.2.

Definicija 3.3.20 (Razcepnost)**.** Pravimo, da je množica modalnosti $\mathcal{O}$ *razcepna*, če za vsa dvojna drevesa $r, r' \in T(T(\{*\}))$ $r \preccurlyeq r'$ implicira $\mu r \preccurlyeq \mu r'$.

V razdelku 3.3.3 pokažemo, da je za vse primere učinkov, ki smo jih podali zgoraj, njihova množica modalnosti $\mathcal{O}$ razcepna množica Scottovo odprtih modalnosti.

Algebrajske teorije

V literaturi je enakost med programi mnogokrat opredeljena preko aksiomov enakosti, ki jim morajo učinki zadostiti. V tej disertaciji pa kot osnovo uporabimo pojem modalnosti in so nato enačbe inducirane glede na to, katere modalnosti izberemo. Da bi to natančno povedali, oblikujemo abstrakten pojem enačbe, inducirane z modalnostmi.

Označimo spremenljivke z naravnimi števili in naj bodo $e \in T(\mathbb{N})$ posplošeni algebrajski izrazi kot v [49]. Modalnosti iz množice $\mathcal{O}$ inducirajo relaciji $\widehat{\leq}$ in $\widehat{=}$ na $T(\mathbb{N})$ (definicija 3.5.3). Ker je $\mathcal{O}$ razcepna množica Scottovo odprtih modalnosti, je $\widehat{\leq}$ *dopustna*, $\widehat{\leq}$ in $\widehat{=}$ pa sta *kompozicijski* (glej [35]). Še več, $\widehat{\leq}$ in $\widehat{=}$ služita kot abstrakciji (algebrajskima izrazoma) $\sqsubseteq^+$ in $\equiv$.

V disertaciji smo pokazali, da za vsak primer učinka s signaturo učinkov Σ in izbrano množico modalnosti $\mathcal{O}_\Sigma$ nastala relacija $\widehat{\leq}$ zadošča običajnim neenakostim, ki so specifične za učinke (na primer tistim iz [83, 84]). To smo izvedli v lemi 3.5.13.

Ne moremo pa vseh algebrajskih teorij inducirati z razcepno množico Scottovo odprtih modalnosti. Še posebej to velja za nekatere algebrajske teorije za kombinacije učinkov. Z uporabo zgornjih orodij lahko pokažemo naslednji dejstvi:

Lema 3.5.17. Vsaka razcepna množica Scottovo odprtih modalnosti, ki je definirana za kombinacijo učinkov globalnega pomnilnika in demonskega nedeterminizma in ki zadošča želenim enačbam za takšne učinke, tudi inducira $x \mathrel{\widehat{=}} \bot$ (vsak izraz je enak $\bot$).

Podoben rezultat pokažemo v lemi 3.5.19, kjer dokažemo, da ne moremo najti razcepne množice Scottovo odprtih modalnosti za opis kombinacije verjetnosti in angelskega nedeterminizma. Zaključimo lahko, da kombinacije globalnega pomnilnika in nedeterminizma ne moremo pravilno opisati s takšno Boolovo logiko, kot smo jo predstavili v poglavju 3. To nam služi kot motivacija, da v poglavju 6 logiko posplošimo na kvantitativno logiko.

Poglavje 4: Aplikativna bipodobnost

Oglejmo si alternativni pojem enakosti med programi, aplikativno bipodobnost [2], ki ga uporabimo v dokazu izreka o združljivosti 3.3.8.

Naj bo $t \in T(X)$ drevo in $A \subseteq X$. Definiramo $t[A] \in T(\{*\})$ kot drevo, ki ga dobimo tako, da v t vse liste z oznako $x \in A$ zamenjamo z $*$ in vse liste z oznako $x \in (X - A)$ zamenjamo z $\bot$. Za dano signaturo Σ in množico modalnosti $\mathcal{O}$ (za Σ) definiramo operacijo $\mathcal{O}(-) : \mathcal{P}(X \times Y) \to \mathcal{P}(T(X) \times T(Y))$ kot[1]:

$$t\ \mathcal{O}(\mathcal{R})\ r \quad \iff \quad \forall D \subseteq X, o \in \mathcal{O},\ (t[D] \in [\![o]\!] \ \Rightarrow\ r[\{y \in Y \mid \exists x \in D, x\,\mathcal{R}\,y\}] \in [\![o]\!].$$

V razdelku 4.1 pokažemo, da ta operacija zadošča lastnostim relatorja, ki je definiran v definiciji 4.1.1.

[1] Za relacijo $\mathcal{R} \subseteq X \times Y$ pišemo $x\ \mathcal{R}\ y$ namesto $(x, y) \in \mathcal{R}$

Aplikativne simulacije

Relator uporabimo, da definiramo različice Abramskyjeve *aplikativne podobnosti (applicative similarity)* in *aplikativne bipodobnosti (applicative bisimilarity)* [2, 14], ki ustrezajo našemu jeziku.

Definicija 4.2.1. Dobro tipizirana relacija $\mathcal{R}$ na zaprtih izrazih je *aplikativna $\mathcal{O}$-simulacija*, če velja:

1. $V\,\mathcal{R}_{\mathbf{N}}\,W \implies V = W$.
2. $\mathsf{thunk}(\underline{M})\,\mathcal{R}_{\mathbf{U}\,\underline{\mathbf{C}}}\,\mathsf{thunk}(\underline{N}) \implies \underline{M}\,\mathcal{R}_{\underline{\mathbf{C}}}\,\underline{N}$.
3. $(j, V)\,\mathcal{R}_{\mathbf{\Sigma}_{i\in I}\,\mathbf{A}_i}\,(k, W) \implies (j = k) \wedge V\,\mathcal{R}_{\mathbf{A}_j}\,W$.
4. $(V, V')\,\mathcal{R}_{\mathbf{A}\times\mathbf{B}}\,(W, W') \implies V\,\mathcal{R}_{\mathbf{A}}\,W \wedge V'\,\mathcal{R}_{\mathbf{B}}\,W'$.
5. $\underline{M}\,\mathcal{R}_{\mathbf{A}\to\underline{\mathbf{C}}}\,\underline{N} \implies \forall V \in \mathit{Terms}(\mathbf{A}), \underline{M}\;V\,\mathcal{R}_{\underline{\mathbf{C}}}\,\underline{N}\;V$.
6. $\underline{M}\,\mathcal{R}_{\mathbf{F}\,\mathbf{A}}\,\underline{N} \implies |\underline{M}|\;\mathcal{O}(\mathcal{R}_{\mathbf{A}})\;|\underline{N}|$.
7. $\underline{M}\,\mathcal{R}_{\mathbf{\Pi}_{i\in I}\,\underline{\mathbf{C}}_i}\,\underline{N} \implies \forall j \in I, \underline{M}\;j\,\mathcal{R}_{\underline{M}_j}\,\underline{N}\;j$.

Definiramo še dve relaciji: *Aplikativna $\mathcal{O}$-podobnost (Applicative $\mathcal{O}$-similarity)* je največja $\mathcal{O}$-simulacija in *Aplikativna $\mathcal{O}$-bipodobnost (Applicative $\mathcal{O}$-bisimilarity)* je največja simetrična $\mathcal{O}$-simulacija.

Izrek 4.2.7 in 4.2.8 (Izreka o sovpadanju)**.** Če so vse modalnosti Scottovo odprte, potem je pozitivna vedenjska šibka urejenost $\sqsubseteq^+$ aplikativna $\mathcal{O}$-podobnost, polna vedenjska ekvivalenca $\equiv$ je aplikativna $\mathcal{O}$-bipodobnost.

Z uporabo Howejeve metode [14, 31] v razdelkih 4.4 in 4.5 pokažemo naslednja izreka:

Izrek 4.5.2 in 4.5.5. Če je $\mathcal{O}$ razcepna množica Scottovo odprtih modalnosti, potem sta relaciji aplikativne $\mathcal{O}$-podobnosti in aplikativne $\mathcal{O}$-bipodobnosti združljivi.

Zaključimo lahko, da izrek o združljivosti 3.3.8 velja.

Poglavje 5: Različice logike

Splošna vedenjska ekvivalenca $\equiv$ in pozitivna vedenjska šibka urejenost $\sqsubseteq^+$ sta inducirani z logikama $\mathcal{V}$ in $\mathcal{V}^+$. Logiki pa lahko na različne načine preoblikujemo, ne da bi pri tem spremenili inducirani logični šibki urejenosti. Zamenjamo lahko osnovno formulo vrednosti $(V \mapsto \underline{\psi})$ z alternativno formulo $(\phi \mapsto \underline{\psi})$, katere semantiko podamo z:

$$\underline{M} \models (\phi \mapsto \underline{\psi}) \quad :\Longleftrightarrow \quad \forall W : \mathbf{A}, (W \models \phi \Rightarrow \underline{M}\;W \models \underline{\psi})\ .$$

S $\mathcal{F}$ označimo dobljeno logiko z negacijo in s $\mathcal{F}^+$ logiko brez negacije. Menjava logike ima konceptualno prednost v tem, da se formule ne sklicujejo na sintakso izrazov programskega jezika.

Posledica 5.4.3. Če je $\equiv_{\mathcal{V}}$ kongruenca, potem velja $(\equiv_{\mathcal{V}}) = (\equiv_{\mathcal{F}})$, in če je $\sqsubseteq_{\mathcal{V}^+}$ šibka kongruenca, potem velja $(\sqsubseteq_{\mathcal{V}^+}) = (\sqsubseteq_{\mathcal{F}^+})$.

Ne da bi spremenili inducirano logično šibko urejenost, lahko logiko preoblikujemo tudi na naslednje načine:

- Definiramo logiko $\mathcal{V}^*$, pri kateri formule izračunov ne vsebujejo konjunkcij, disjunkcij in negacij. V posledici 5.1.4 pojasnimo, da $\equiv_{\mathcal{V}}$ sovpada z $\equiv_{\mathcal{V}^*}$.
- Namesto števne disjunkcije uporabimo končno disjunkcijo in tako definiramo logiko $(\mathcal{O}, \vee, \bigwedge, \neg)$. Z lemo 5.2.6 zagotovimo, da se $\equiv_{\mathcal{V}}$ sklada z $\equiv_{(\mathcal{O},\vee,\bigwedge,\neg)}$, če so vse modalnosti Scottovo odprte.
- Brez uporabe disjunkcij, konjunkcij ali negacij definiramo logiko $(\mathcal{O}, \bot, \top, +)$. V trditvi 5.2.11 povemo, da $\equiv_{\mathcal{V}}$ sovpada z $\equiv_{(\mathcal{O},\bot,\top,+)}$, če so učinki napaka, globalni pomnilnik, vhod/izhod ali časomer.

V razdelku 5.3 obravnavamo kombinacije učinkov in definiramo pripadajoče Boolove modalnosti, ki opisujejo obnašanje teh kombinacij. Določimo tudi okoliščine, v katerih Boolove modalnosti tvorijo množico Scottovo odprtih modalnosti. Nekaj pozitivnih rezultatov se nahaja v lemah 5.3.4, 5.3.7 in drugih. Vendar pa, kot smo že videli, dobimo tudi nekaj negativnih rezultatov, ki motivirajo posplošitev na kvantitativno logiko.

Poglavje 6: Kvantitativna logika

Ker želimo opisati vedenjsko ekvivalenco za jezike z določenimi kombinacijami učinkov ali pa podati bolj naraven način za opisovanje vedenjskih lastnosti programov z učinki, posplošimo Boolovo logiko na kvantitativno logiko.

Kot prostor resničnostnih vrednosti uporabimo mrežo $\mathbb{A}$ z urejenostjo $\trianglelefteq$, ki je števno polna. To pomeni, da imamo za vsako števno podmnožico $X \subseteq \mathbb{A}$ *najmanjšo zgornjo mejo* (supremum) $\bigvee X$ in *največjo spodnjo mejo* (infimum) $\bigwedge X$. Naj bosta $\boldsymbol{T}$ in $\boldsymbol{F}$ največji ter najmanjši element množice $\mathbb{A}$. Dodatno zahtevamo obstoj *involucije* na množici $\mathbb{A}$, saj jo bomo uporabili za modeliranje *negacije*. Involucijo podamo s preslikavo $\neg : \mathbb{A} \to \mathbb{A}$, pri čemer velja $\forall a, b \in \mathbb{A}, a \trianglelefteq b \iff \neg b \trianglelefteq \neg a$ in $\forall a, \neg\neg a = a$.

Relacije zadostitve '$\models$' naredimo kvantitativne, tako da so to zdaj funkcije $\models : \mathit{Terms}(\underline{\underline{\mathbf{E}}}) \times \mathit{Form}(\underline{\underline{\mathbf{E}}}) \to \mathbb{A}$ za vsak tip $\underline{\underline{\mathbf{E}}}$. Logiko na sliki 3.1 razširimo z dvema novima konstruktorjema formul:

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

kjer definiramo $\underline{\underline{P}} \models \underline{\underline{\phi}}_{\trianglerighteq a} := \begin{cases} \boldsymbol{T} & \text{če } (\underline{\underline{P}} \models \underline{\underline{\phi}}) \trianglerighteq a. \\ \boldsymbol{F} & \text{sicer.} \end{cases}$ $\qquad \underline{\underline{P}} \models \kappa_a := a.$

Za vse ostale relacije zadostitve v definiciji Boolove logike zamenjamo '$\Leftrightarrow$' z '$=$'. V razdelku 6.1.1 se nahaja formalna definicija kvantitativne relacije zadostitve.

Učinke interpretiramo z uporabo kvantitativnih modalnosti $\mathcal{Q}$, kjer vsaka operacija $q \in \mathcal{Q}$ podaja funkcijo $[\![q]\!] : T(\mathbb{A}) \to \mathbb{A}$. Definiramo $\underline{M} \models q(\phi)$ tako, da v drevesu $|\underline{M}|$ zamenjamo vse liste oblike $\langle \mathsf{return}(V) \rangle$ z $\langle V \models \phi \rangle$ in rezultat podamo funkciji $[\![q]\!]$. Naj bo $\mathcal{U}$ polna logika in $\mathcal{U}^+$ logika brez negacij.

Definicija 6.3.1. Naj bo $\mathcal{L} \subseteq \mathcal{U}$ podmnožica logike. *Logično šibko urejenost* $\sqsubseteq_{\mathcal{L}}$ definiramo z ekvivalenco:

$$\forall \underline{P}, \underline{R} \in \mathit{Terms}(\underline{\mathbf{E}}), \quad \underline{P} \sqsubseteq_{\mathcal{L}} \underline{R} \iff \forall \underline{\phi} \in \mathit{Form}(\underline{\mathbf{E}})_{\mathcal{L}},\ \underline{P} \models \underline{\phi} \trianglelefteq \underline{R} \models \underline{\phi}$$

Logično ekvivalenco $\equiv_{\mathcal{L}}$ dobimo kot presek $\sqsubseteq_{\mathcal{L}} \cap \sqsupseteq_{\mathcal{L}}$.

Primeri

V splošnem velja $[\![q]\!](\langle a \rangle) = a$ in $[\![q]\!](\bot) = \boldsymbol{F}$. Oglejmo si nekaj primerov prostorov resničnostnih vrednosti in kvantitativnih modalnosti za učinke in njihove kombinacije.

Verjetnost: Kot polno mrežo resničnostnih vrednosti uporabimo prostor realnih verjetnosti $\mathbb{A} := [0, 1]$, ki vsebuje verjetnosti resnice. Urejenost $\trianglelefteq$ je kar '$\leq$' in za negacijo $\neg$ vzamemo $\neg x = 1 - x$. Definiramo zgolj eno kvantitativno modalnost E, ki označuje pričakovano vrednost in za katero velja $[\![\mathsf{E}]\!](\mathsf{pr}(t, r)) := ([\![\mathsf{E}]\!]_n(t) + [\![\mathsf{E}]\!]_n(r))/2$.

Globalni pomnilnik: Kot prostor resničnostnih vrednosti uporabimo množico stanj $\mathbb{A} := \mathcal{P}(\mathbb{N}^{\mathsf{Loc}})$ in jo uredimo z inkluzijo. Zopet podamo zgolj eno kvantitativno modalnost G, za katero velja: $[\![\mathsf{G}]\!](\mathsf{lookup}_l(t_0, t_1, \dots)) := \{s \in \mathbb{N}^{\mathsf{Loc}} \mid s \in [\![\mathsf{G}]\!](t_{s(l)})\}$ in $[\![\mathsf{G}]\!](\mathsf{update}_l(m; t)) := \{s \in \mathbb{N}^{\mathsf{Loc}} \mid s[l := m] \in [\![G]\!](t)\}$.

Časomer: Prostor resničnostnih vrednosti je tokrat interval časovnih zamikov $\mathbb{A} := [0, \infty]$, ki ga uredimo z obratno urejenostjo, in definiramo kvantitativno modalnost C, tako da velja $[\![\mathsf{C}]\!](\mathsf{tick}_c(t)) := c + [\![\mathsf{C}]\!](t)$.

Dodajanje nedeterminizma: Zgornjim primerom lahko dodamo nedeterminizem na naslednji način. Za prostor resničnostnih vrednosti obdržimo $\mathbb{A}$ in za vsako modalnost $q \in \mathcal{Q}$ definiramo dve različici $q_\Diamond$ in $q_\Box$. Zanju veljata naslednji pravili: $[\![\mathsf{q}_\Diamond]\!](\mathsf{or}(t, r)) := [\![\mathsf{q}_\Diamond]\!](t) \vee [\![\mathsf{q}_\Diamond]\!](r)$ in $[\![\mathsf{q}_\Box]\!](\mathsf{or}(t, r)) := [\![\mathsf{q}_\Box]\!](t) \wedge [\![\mathsf{q}_\Box]\!](r)$, kjer sta operaciji $\vee$ in $\wedge$ definirani s strukturo mreže na $\mathbb{A}$.

Zgornji primeri in tudi druge kombinacije učinkov, kot na primer kombinacija verjetnosti in globalnega pomnilnika ali kombinacije z napako, so pojasnjeni v razdelku 6.2.

Združljivost

Vse zgornje rezultate, ki se nanašajo na Boolovo logiko, lahko razširimo na kvantitativno logiko, če posplošimo pojma razcepnosti in Scottove odprtosti. Dokažemo lahko, da se $\equiv_{\mathcal{U}}$ in aplikativna $\mathcal{Q}$-bipodobnost skladata in da sta kongruenci. Podrobnosti so v razdelkih 6.3 in 6.4, podamo pa kratek povzetek.

Naj bosta $h : X \to Y$ in $t \in T(X)$. Naj bo $t[h] \in T(Y)$ drevo, ki ga dobimo, ko zamenjamo liste, označene z $x \in X$, z listi z oznako $h(x) \in Y$.

Definicija 6.4.1. Za dano množico kvantitativnih modalnosti $\mathcal{Q}$ imamo operacijo $\mathcal{Q}(-) : \mathcal{P}(X \times Y) \to \mathcal{P}(TX \times TY)$, ki je podana z ekvivalenco

$$t\,\mathcal{Q}(\mathcal{R})\,t' \quad \iff \quad \forall h : X \to \mathbb{A}, \forall q \in \mathcal{Q},\ [\![q]\!](t[h]) \trianglelefteq [\![q]\!](t'[\lambda y. \bigvee\{h(x) \mid x \in X, x\mathcal{R}y\}].$$

Pojem Scottove odprtosti za množico Boolovih modalnosti zamenjamo z listno monotonostjo in drevesno Scottovo zveznostjo, ki sta definirani v razdelku 6.3.1. Razcepnost posplošimo v definicijah 6.3.7, 6.3.9 in 6.3.11, lahko pa jo karakteriziramo tudi z naslednjo posledico.

Posledica 6.4.13. Če so vse modalnosti $q \in \mathcal{Q}$ listno monotone, potem je $\mathcal{Q}$ razcepna množica natanko tedaj, ko velja $\forall \mathcal{R} \subseteq X \times Y, \forall r, r' \in TT\mathbb{A}$, $r\,\mathcal{Q}(\mathcal{Q}(\mathcal{R}))\,r' \Rightarrow \mu r\,\mathcal{Q}(\mathcal{R})\,\mu r'$.

Izrek 6.3.15 (Posplošeni izrek o združljivosti)**.** Naj bo $\mathcal{Q}$ razcepna množica listno monotonih drevesno Scottovo zveznih modalnosti. Tedaj sta odprti razširitvi $\equiv_{\mathcal{U}}$ in $\sqsubseteq_{\mathcal{U}^+}$ združljivi.

Če so vse modalnosti $q \in \mathcal{Q}$ listno monotone, potem je po lemi 6.4.2 $\mathcal{Q}(-)$ relator. *Aplikativno $\mathcal{Q}$-simulacijo* definiramo kot v definiciji 4.2.1, vendar uporabimo relator $\mathcal{Q}(-)$ namesto $\mathcal{O}(-)$. Enako drži tudi za *aplikativno $\mathcal{Q}$-podobnost* in *aplikativno $\mathcal{Q}$-bipodobnost*.

Izrek 6.4.8 (Posplošeni izrek o sovpadanju)**.** Naj $\mathcal{Q}$ vsebuje le listno monotone modalnosti. Tedaj je logična šibka urejenost $\sqsubseteq_{\mathcal{U}^+}$ $\mathcal{Q}$-podobnost in logična ekvivalenca $\sqsubseteq_{\mathcal{U}}$ $\mathcal{Q}$-bipodobnost.

Izrek 6.4.14. Aplikativna $\mathcal{Q}$-podobnost in aplikativna $\mathcal{Q}$-bipodobnost sta kompatibilni, če je $\mathcal{Q}$ razcepna množica listno monotonih drevesno Scottovo zveznih modalnosti.

Izreka sta dokazana v razdelku 6.4.

Poglavje 7: Polimorfni in rekurzivni tipi

V tem poglavju premislimo, kako bi v jezik dodali dodali konstruktorje za univerzalne polimorfne tipe in rekurzivne tipe, ki so zelo močni mehanizmi. Uvedemo spremenljivke tipa vrednosti α in spremenljivke tipa izračuna $\underline{\beta}$.

$\mathbf{A}, \mathbf{B} ::= \cdots \mid \mu\alpha.\, \mathbf{A},$
$\underline{\mathbf{C}}, \underline{\mathbf{D}} ::= \cdots \mid \forall\alpha.\, \underline{\mathbf{C}} \mid \forall\underline{\beta}.\, \underline{\mathbf{C}} \mid \underline{\mu\beta}.\, \underline{\mathbf{C}}.$

Definicije izrazov, njihove operacijske semantike in vedenjske logike lahko primerno razširimo, da dobimo naslednji rezultat.

Izrek 7.2.8 in 7.3.7. Naj bo $\mathcal{Q}$ razcepna množica listno monotonih drevesno Scottovo zveznih modalnosti. Tedaj so odprte razširitve vedenjskih šibkih urejenosti, ki jih dobimo iz logik za jezik, razširjen z univerzalnimi polimorfnimi tipi in rekurzivnimi tipi, združljive.