# Quantitative Program Analysis: From Strongest Postconditions to Hyperproperties

*Linpeng Zhang*

A dissertation submitted in partial fulfillment

of the requirements for the degree of

**Doctor of Philosophy**

of

**University College London**.

Department of Computer Science

University College London

July 21, 2025

I, Linpeng Zhang, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

# Abstract

The main aim of this thesis is to provide a holistic comprehension of strongest-postcondition-style calculus and weakest-precondition-style calculus for quantitative program analysis.

To achieve this goal, the thesis will present novel concepts that deepen the understanding of program correctness and incorrectness, paving the way for the creation of new program logics. For example, it will define new predicate transformers such as the strongest liberal post, along with their quantitative variants, and demonstrate their applications in Information Flow Analysis.

Furthermore, by bridging the gap between quantitative forward and backward transformers, the thesis will progressively elevate the reasoning to a more general setting, making the calculi parametrized to a class of semirings. This parametrization will enable a deeper understanding of the fundamental relationships between forward and backward, correctness and incorrectness, as well as nontermination and unreachability.

Finally, the thesis will delve into the study of hyperproperties and integrate predicate transformers reasoning into them, and beyond. It will demonstrate that their quantitative variants enable reasoning about hyperquantities, thus facilitating analyses involving expected values, variance, and other quantitative metrics. Moreover, the exploration of higher-order predicate transformers will provide insights into their limitations and advantages, allowing for the instantiation of simpler predicate transformers when reasoning about less complex properties.

# Impact Statement

**Impact Outside Academia.**

The increasing complexity of hardware and software systems necessitates more advanced verification methodologies. While testing remains an invaluable tool, it is inherently limited and cannot guarantee the detection of all bugs, as famously noted by Dijkstra [1]. Consequently, traditional formal methods [2; 3; 4; 5] have focused on proving safety properties. However, these approaches often prove impractical for everyday programmers, as they can generate false positives [6; 7], which conflicts with the fundamental static analysis axiom: “Don’t spam the developer” [8].

This thesis tackles these challenges by developing new theoretical frameworks that bridge the gap between tools for correctness and those for incorrectness. Our work introduces novel predicate transformers applicable across a wide range of scenarios, including safety and bug-finding, (probabilistic) nontermination, unreachability, and hyperproperties such as Information Flow Analysis. These frameworks are compositional [4], a key feature that contributes to the scalability of industrial static analyses, as demonstrated at Meta [9; 10; 11] and in broader industry contexts [12]. By enhancing the precision and applicability of these methodologies, this thesis aims to make formal verification more accessible and effective for real-world software development.

**Impact Inside Academia.**

Existing efforts in quantitative program verification have predominantly focused on weakest-precondition-style calculi [13; 14; 15; 16]. This thesis expands the academic conversation by providing a comprehensive understanding of both strongest-postcondition and weakest-precondition calculi in the context of quantitative program analysis. We introduce new concepts, such as the strongest liberal postcondition and its quantitative variants [17], and examine their potential to improve the analysis of program correctness and incorrectness [18].

In a similar vein to how Zilberstein et al. [19, 20]; Clarkson and Schneider [21] extended Hoare Logic to handle hyperproperties [21], we extend this approach to predicate transformers, demonstrating how predicate transformer reasoning can be applied in this context. Our work illustrates that quantitative variants of predicate transformers are powerful tools for reasoning about hyperquantities, such as expected values and variance [18]. Through the examination of more general transformers, we identify conditions that lead to the development of simplified frameworks for analyzing more straightforward properties.

Just as traditional predicate transformers and Hoare-like logics empower programmers to verify program correctness, we contend that our framework provides researchers with a deeper insight into existing logics. Indeed, our calculus reveals novel dualities between forward and backward transformers, correctness and incorrectness, as well as nontermination and unreachability.

**Publications.**

The findings of this thesis have been showcased at several major conferences, including the Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) in both 2022 and 2024 [17; 18], as well as at the workshop "Formal Methods for Incorrectness 2024" co-located at POPL. Additionally, although not included in this thesis, I have explored further applications of Rice's Theorem to control flow graphs with equality guards and affine assignments. This research led to publications at the International Colloquium on Automata, Languages, and Programming (ICALP) in 2021 [22], as well as in the journal Information and Computation [23].

# Acknowledgements

I am deeply grateful to my supervisors, Benjamin and Alexandra, for granting me this opportunity and for their unwavering support throughout my PhD journey, through both its challenges and successes.

Benjamin, thank you for your sharp research instincts, intuition, and for emphasizing the importance of explainability and simplicity. Alexandra, I sincerely appreciate your balanced perspective and thoughtful guidance—without it, I would not have completed this journey.

I am also thankful to Fredrik Dahlqvist for chairing my first-year and transfer vivas, to Kevin, Lena and Tobias for the fruitful discussions at Schloss Dagstuhl that sparked the initial ideas of this thesis, and to my collaborator Noam, whose invaluable feedback and supervision have greatly enriched our joint work.

As part of the Programming Principles, Logic, and Verification group, I had the pleasure of meeting incredible colleagues: Jas, Lachlan, Leo, Louis, Mateo, Robin, Stefan, Tao, Tiago, Todd, Will, Wojciech, and many others who made our group such a unique and inspiring community.

Alongside my PhD, I've been working as a Teaching Assistant for various courses, which has been a truly enjoyable experience—one I would highly recommend to anyone. In particular, teaching Blockchain Technologies was invaluable, as it introduced me to an entirely new field. Thank you Jiahua and Silvia for giving me this opportunity.

Beyond my academic journey, I was fortunate to explore various industry opportunities over the years. I began as a Software Engineer Intern at Meta, working with the Static Analysis (Infer) team. Thank you, Jules, Sungkeun, and Dulma, for your mentorship—this experience was so invaluable that it inspired me to pursue full-time industry roles. Later, I joined Lacework (subsequently acquired by Fortinet), where I worked with the exceptional Code Security team and filed two patents. A special thank you to Peter for bringing together the dream team—Christien, Patrice, Edoardo, Jérémy, Pascal, Ionut, Vaggelis, Ibrahim, Francesco, Hemanth, David, Leanthony, Teodor, and many others—and for your bold vision of pushing beyond the boundaries of academia.

More recently, I returned to Meta as a Production Engineer, where I am grateful to the MonCap team for their mentorship and for deepening my understanding of infrastructure at scale. While my time there was rewarding, I could not resist the opportunity to join Meticulous AI, an exciting hypergrowth startup at the forefront of innovative technologies. I am particularly thankful to my current colleagues—Gabriel, Quentin, Alex, Daniel, and Edoardo (our paths crossing once again)—for your warm welcome and collaborative spirit. These diverse professional experiences across multiple organizations in a short timeframe have significantly expanded my perspective, offering valuable insights into different technical approaches, organizational cultures, and problem-solving methodologies that have complemented my academic training.

I am also deeply grateful to my Italian friends—Lorenzo, Mattia, Teodor, Giulio, Marco, Giuseppe(s), Riccardo, Damiano, Alex, Denis, Francesco, Fiammetta and Anna—for their friendship and support. A heartfelt thank you to my family for their unwavering encouragement throughout this journey. In particular, I dedicate this thesis to my parents, Chunxin and Guanzhen, for their unconditional love, and to my sister, Giulia, for always being there for me.

Finally, to Alice—thank you for taking care of me, for celebrating every high, and for standing by me through every low. You have been an essential part of this journey, and this thesis would not exist without you.

## UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

### Quantitative Strongest Post: A Calculus for Reasoning about the Flow of Quantitative Information, [17]

1. **1. For a research manuscript that has already been published** (if not yet published, please skip to section 2)**:**

   (a) **What is the title of the manuscript?** Quantitative Strongest Post: A Calculus for Reasoning about the Flow of Quantitative Informa- tion.

   (b) **Please include a link to or doi for the work:** https://dl.acm.org/doi/10.1145/3527331.

   (c) **Where was the work published?** Proceedings of the ACM on Programming Languages, Volume 6, Issue OOPSLA1.

   (d) **Who published the work?** Association for Computing Machinery

   (e) **When was the work published?** 29th April 2022.

   (f) **List the manuscript's authors in the order they appear on the publication:** Linpeng Zhang, Benjamin Lucien Kaminski.

   (g) **Was the work peer reviewd?** Yes.

   (h) **Have you retained the copyright?** Yes.

   (i) **Was an earlier form of the manuscript uploaded to a preprint server (e.g. medRxiv)? If 'Yes', please give a link or doi.** Yes: `https://arxiv.org/abs/2202.06765`.
   If 'No', please seek permission from the relevant publisher and check the box next to the below statement:

   ☐ *I acknowledge permission of the publisher named under 1d to include in this thesis portions of the publication named as included in 1c.*

2. **For multi-authored work, please give a statement of contribution covering all authors** (if single-author, please skip to section 4)**:** This work represents the result of equal contributions and close collaboration between the authors. Benjamin Lucien Kaminski initially proposed a quantitative generalisation of the strongest postcondition calculus and its application to information flow analysis, while providing supervision throughout the project. Building on this foundation, Linpeng Zhang performed the technical development that led to original contributions beyond the initial proposal, such as the formulation of slp and the formal foundations for partial incorrectness reasoning.

3. **In which chapter(s) of your thesis can this material be found?** This material can be found in Chapter 3, which substantially reproduces the original paper with minimal modifications to integrate with the overall thesis structure.

## Quantitative Weakest Hyper Pre: Unifying Correctness and Incorrectness Hyperproperties via Predicate Transformers, [18]

1. **1. For a research manuscript that has already been published** (if not yet published, please skip to section 2)**:**

   (a) **What is the title of the manuscript?** Quantitative Weakest Hyper Pre: Unifying Correctness and Incorrectness Hyperproperties via Predicate Transformers.

   (b) **Please include a link to or doi for the work:** `https://dl.acm.org/doi/10.1145/3689740`.

   (c) **Where was the work published?** Proceedings of the ACM on Programming Languages, Volume 6, Issue OOPSLA1.

   (d) **Who published the work?** Association for Computing Machinery

   (e) **When was the work published?** 8th October 2024.

   (f) **List the manuscript's authors in the order they appear on the publication:** Linpeng Zhang, Noam Zilberstein, Benjamin Lucien Kaminski, Alexandra Silva.

   (g) **Was the work peer reviewd?** Yes.

   (h) **Have you retained the copyright?** Yes.

   (i) **Was an earlier form of the manuscript uploaded to a preprint server (e.g. medRxiv)? If 'Yes', please give a link or doi.** Yes: `https://arxiv.org/abs/2404.05097`.

   If 'No', please seek permission from the relevant publisher and check the box next to the below statement:

   ☐ *I acknowledge permission of the publisher named under 1d to include in this thesis portions of the publication named as included in 1c.*

2. **For multi-authored work, please give a statement of contribution covering all authors** (if single-author, please skip to section 4)**:** As first

author, Linpeng Zhang originated the idea of the quantitative weakest hyper pre calculus and conducted the technical development. Benjamin Lucien Kaminski provided supervision, particularly on the writing of the manuscript presented at the "Formal Methods for Incorrectness 2024" workshop co-located at POPL. After the workshop, Noam Zilberstein contributed technical supervision and guidance, especially in generalizing the calculus to arbitrary semi-rings, and also helped significantly with organizing the presentation of the manuscript. Both Benjamin Lucien Kaminski and Alexandra Silva helped to finalise the final version of the manuscript published at OOPSLA 2024.

3. **In which chapter(s) of your thesis can this material be found?** This material can be found in Chapter 4 (specifically Sections 4.2 and 4.4 to 4.6) and in Chapter 5 (specifically Sections 5.1 to 5.3 and 5.6 to 5.10). The material has been revised and expanded to fit the thesis structure.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Reasoning about programs is a fundamental yet profoundly challenging aspect of software engineering. As software systems become increasingly complex and embedded in critical aspects of modern life, ensuring their correctness has never been more crucial. The importance of formal verification—a rigorous methodology for proving the correctness of programs through mathematical means—has been recognized since the early days of computing and remains a critical area of research and practice today.

In the modern world, software systems are ubiquitous, controlling essential functions such as airplane flight systems [25], nuclear power plant emergency protocols [26], and car engines [27]. The increasing complexity of these systems, including ever-expanding machine learning models, increases the likelihood of introducing errors, which can lead to substantial economic losses [28; 29; 30] or, in severe cases, endanger human lives [31; 32].

As software systems become more integrated into critical infrastructure and daily operations, the challenges of ensuring their correctness intensify. Formal verification provides a crucial tool for addressing these challenges by offering a systematic approach to proving the reliability and safety of software through mathematical rigor. This approach is indispensable in mitigating the risks associated with software errors and ensuring the robustness of systems that are integral to modern life.

## 1.1 Historical Context

The quest for program correctness dates back to the early 20th century. In 1941, Konrad Zuse built the Z3, widely regarded as the world's first programmable computer. Zuse's design intentionally avoided loops and conditional branching, opting instead for a straightforward, linear control flow. His choice was driven by concerns over the complexity introduced by such constructs, which he believed could hinder comprehensibility and reasoning about program behavior [33].

This concern about program complexity was echoed by Edsger Dijkstra in 1968. Dijkstra criticized the "goto" statement for its potential to create convoluted code, advocating instead for structured programming constructs like "while" loops. He believed that such constructs would promote clearer, more understandable code, thereby facilitating more effective reasoning Dijkstra [34]. Both Zuse and Dijkstra highlighted the challenges of comprehending and reasoning about program behavior, setting the stage for the development of formal verification techniques.

The evolution of computing has introduced even more complexities. The notion of probabilistic programs, which incorporate random elements and conditional branching based on probabilistic outcomes, adds a new layer of difficulty. Randomization in computing has been explored since the early days, with significant contributions such as Tony Hoare's randomized Quicksort algorithm [35] and Michael Rabin's probabilistic automata [36]. Despite these advances, reasoning about probabilistic programs remains an active area of research and development, reflecting the ongoing challenges of formal verification [37; 16].

The formal study of probabilistic programs gained momentum with Dexter Kozen's foundational work in the late 1970s and early 1980s, which laid the groundwork for understanding the semantics of probabilistic computations [38; 39]. Subsequent contributions by Sergiu Hart, Micha Sharir, and Amir Pnueli, as well as further work by Kozen himself, advanced the verification techniques for probabilistic programs [40; 41; 42; 13]. Despite these seminal advances, the techniques for formal reasoning about probabilistic programs are still less developed compared to those for deterministic programs. As argued by Kaminski, it is likely that part of the reason for this discrepancy is that reasoning about probabilistic programs is harder from a computability perspective [43; 44; 15].

With the rapid advances in machine learning and the abundance of available data, machine-learned software is increasingly important in assisting or even autonomously making decisions that impact our lives. As a result, formal methods is gaining further momentum, when applied to machine learning models [45]. One key property of interest is the robustness of neural networks, ensuring that they behave correctly even when the input is perturbed. This problem is addressed by several established methods, including precise methods based on solvers [46; 47], abstract interpretation [48; 49; 50; 51], or a combination of both [52].

Another important aspect is the fairness of machine learning models [53], which can be seen as a form of verification. Fairness in neural networks, specifically ensuring that certain inputs do not lead to biased outcomes [53], is related to the study of information flow [54; 55; 56].

As we delve into the realm of formal verification, this thesis will investigate the general methodologies and challenges associated with reasoning about deterministic, non-deterministic and probabilistic programs. The goal is to enhance our understanding of verification techniques and address the gaps that exist.

## 1.2 Formal Methods

To address software errors before deployment, traditional methods such as testing are employed. Testing involves running programs with a finite set of inputs and verifying their correctness through assertions. However, as Dijkstra noted, "testing can be quite effective for showing the presence of bugs, but is hopelessly inadequate for showing their absence" [1]. Testing alone cannot guarantee the absence of all bugs, as it is limited by the scope and number of test cases.

Formal methods provide a more rigorous approach to ensuring software correctness and, more recently, incorrectness [6] as well. By creating a precise mathematical model of a program's behavior, formal methods enable the formal proof of properties. However, Rice's undecidability theorem and its intensional generalizations [57; 22; 23] present a significant challenge, asserting that many non-trivial properties of programs are undecidable, meaning no algorithm can decide whether these properties hold for all possible programs. This does not mean that such properties are impossible to solve, but rather that they require trade-offs, because they cannot be solved automatically in *all* cases. As a result, formal methods must balance between completeness (proving all true facts), soundness (ensuring conclusions are correct under specified assumptions), and automation (the extent to which proofs can be automated). To motivate why formal methods work, a more recent characterization of the undecidability of formal methods is provided in [58]. They demonstrate that many existing properties, including those undecidable by Rice's Theorem, are so-called witnessable: although it is impossible to solve these properties precisely, it is always possible to improve any decidable approximation to make it closer to the precise solution, motivating the efforts in this area.

## 1.3 Background

### 1.3.1 Correctness Reasoning

Program analysis and formal verification have been studied for over 70 years, beginning with early foundational work by Alan Turing [59]. One of the pivotal developments in this field is *Hoare Logic* ($\mathsf{HL}$) [2], a proof system used to establish the *partial correctness* of programs. $\mathsf{HL}$ employs logical assertions to reason about the state of a program at specific points in its execution, typically in the form of Hoare triples $\models \{\, \varphi \,\}\ C\ \{\, \psi \,\}$. This triple intuitively expresses that if the precondition $\varphi$ holds before executing the program $C$, then the postcondition $\psi$ will hold after $C$ executes, provided $C$ terminates.

A key technique in this domain is *overapproximation*, used to prove safety specifications. Overapproximation involves considering a broader set of possible program behaviors than the program might actually exhibit. By doing so, we can ensure that if no errors are found within this broad set, the program is *certainly correct* [3; 6]. This is because the actual set of program behaviors, being a subset of the overapproximation, will also be error-free.

### 1.3.2 Incorrectness Reasoning

While correctness logics such as Hoare Logic focus on properties over *all* executions, these *overapproximate* logics can be limiting when it comes to identifying bugs. O'Hearn [6] highlights this limitation, advocating for the development of new program logics that emphasize bug-finding, similar to approaches in symbolic execution literature described by Godefroid et al. [60]. Unlike traditional correctness reasoning, which uses overapproximation, the focus here is on *underapproximation* [6]. This technique is crucial for identifying bugs and proving the presence of errors in programs. By concentrating on a subset of program behaviors, underapproximation ensures that if an error is found within this subset, the program is *definitely incorrect.*

One notable approach in this direction is *Incorrectness Logic* (IL). Proposed by O'Hearn [6] and independently by de Vries and Koutavas [61] (under the name *reverse Hoare logic*), IL provides a formal framework for reasoning about program errors. In IL, a triple $[\,\varphi\,]\ C\ [\,\psi\,]$ expresses that the postcondition $\psi$ is an *underapproximation* of the set of states reachable by executing the program $C$ starting from a state satisfying the precondition $\varphi$. This stands in contrast to Hoare Logic, where the focus is on ensuring that all executions from a given precondition lead to a desired postcondition. IL, on the other hand, is used to demonstrate that specific erroneous states are reachable, thus proving the presence of bugs in the program.

### 1.3.3 Predicate Transformers

The logics developed for correctness [2; 62] and incorrectness [61; 6] introduce sufficient preconditions so that the postconditions are, respectively, an overapproximation and an underapproximation of the reachable states.

A different approach was introduced originally by Dijkstra [63; 1], where he presented the so-called *weakest precondition calculus.* Given a postcondition, this method finds a necessary and sufficient precondition so that, if satisfied, the program will definitely terminate in a state satisfying the postcondition. In other words, once the weakest precondition $\mathsf{wp}$ of a program $C$ with respect to a postcondition $\psi$ is computed, one can easily check the validity of a Hoare triple by checking an implication with the weakest precondition. This is also useful for understanding exactly what conditions need to be true before execution to ensure a desired outcome. Notably, $\mathsf{wp}$ is a backward-moving technique, working from the desired postcondition back to the initial preconditions.

An analogous calculus, but forward-moving, is the strongest postcondition calculus [64]. The strongest postcondition $\mathsf{sp}$ of a program $C$ with respect to a precondition $\varphi$ describes the most precise set of states that can result from executing $C$ starting from any state satisfying $\varphi$. This helps in determining the exact reachable states, which can then be underapproximated to prove incorrectness specifications or overapproximated for correctness reasoning [6].

### 1.3.4 Quantitative Weakest Pre Transformer

Backward moving weakest-precondition-style calculi have been generalized to real-valued-function transformers, first by Kozen [13], to reason about probabilistic programs. Intuitively, classical weakest pre anticipates the truth of a given postcondition - whether it will hold after the execution of the program, or not. Kozen [13] extends the analysis from simply checking whether a postcondition holds (a boolean truth value) to anticipating numerical values, such as the probability that some postcondition will be satisfied after program termination, given an initial state. This approach was further extended by McIver and

Morgan [14]; Hark et al. [15]; Kaminski [16] to reason about expected values of general random variables, after the execution of a program. These generalizations enable the calculation of various quantities, such as the expected value of a given program variable or the program runtime.

### 1.3.5 Program Logics for Hyperproperties

*Hoare Logic* (HL) [2] and *Incorrectness Logic* (IL) [6] are foundational proof systems designed to establish properties of programs based on *individual executions*. While powerful, these logics fall short in addressing properties that inherently involve relationships between multiple executions, such as security properties related to confidentiality, integrity, or authenticity. These properties, known as *hyperproperties*, require reasoning beyond single execution traces, as they may be violated through the analysis of multiple execution traces by an attacker [21].

To bridge this gap, Benton [65] introduced a relational extension of Hoare Logic (RHL), enabling reasoning about pairs of executions and thereby allowing the verification of hyperproperties. This relational approach is crucial for expressing and proving properties that depend on the comparison of different execution traces, even for potentially different programs. RHL has subsequently been extended by many researchers [66; 67; 68; 69] to prove security properties of single programs. On the incorrectness side, the relational extension of IL has been put forward by Murray [70] to prove insecurity in a program, such as disproving non-interference.

### 1.3.6 Unified Program Logics

The aforementioned theories of incorrectness (IL) diverge significantly from theories of correctness (HL), meaning that entirely separate analysis must be done for verification and bug-finding. To overcome this limitation, new theories for *unified* reasoning about both correctness and incorrectness have been proposed [71; 72; 19; 73; 20; 74]. These include logics not only for individual program traces but also on hyperproperties [73]. Outcome Logic

(OL) [60; 19; 20; 74] and Hyper Hoare Logic (HHL) [73]—which advocate that a single logic can be used to prove (or disprove) a wide variety of properties, including hyperproperties.

## 1.4 Main aims

The main aim of this thesis is to provide a holistic comprehension of strongest-postcondition-style calculus and weakest-precondition-style calculus for quantitative program analysis.

To this goal, the thesis will introduce novel concepts that enrich the understanding of program correctness and incorrectness, thereby suggesting the development of new program logics. For instance, new predicate transformers such as strongest liberal post will be defined, along with their quantitative variants (Definitions 3.4.1 and 3.4.2), showcasing their applications in Information Flow Analysis (Section 3.8).

Furthermore, by bridging the gap between quantitative forward and backward transformers, the thesis will progressively elevate the reasoning to a more general setting, making the calculi parametrized to a class of semirings (Definitions 4.3.1 and 4.4.1). This parametrization will enable a deeper understanding of the fundamental relationships between forward and backward reasoning (Theorem 4.7.4), correctness and incorrectness (Section 4.5), as well as nontermination and unreachability (Section 4.6).

Finally, the thesis will delve into the study of hyperproperties and integrate predicate transformer reasoning into them and beyond. It will demonstrate that their quantitative variants enable reasoning about hyperquantities (Definition 5.2.1), thus facilitating analyses involving expected values, variance, and other quantitative metrics (Example 5.2.4). Moreover, the exploration of higher-order predicate transformers (Definitions 5.3.3, 5.4.3 and 5.5.1) will provide insights into their limitations and advantages, allowing for the instantiation of simpler predicate transformers when reasoning about less complex properties (Definition 5.6.1).

## 1.5 Overview

- **Chapter 1:** This chapter offers an introduction to the topics covered in the thesis, setting the stage for the overall objectives and direction of the research.

- **Chapter 2:** This chapter reviews prior fundamental research and presents the mathematical background necessary to understand the main results of the thesis. Topics include Hoare Logic, Incorrectness Logic, Weakest Precondition and Strongest Postcondition calculi, along with a brief overview of their quantitative counterparts.

- **Chapter 3:** Based on the paper by Zhang and Kaminski [17]. This chapter presents a novel strongest-postcondition-style calculus for quantitative reasoning about non-deterministic programs with loops. Whereas existing quantitative weakest pre allows reasoning about the value of a quantity *after* a program terminates on a given *initial state*, quantitative strongest post allows reasoning about the value that a quantity had *before* the program was executed and reached a given *final state.* It demonstrates how strongest post enables reasoning about the *flow of quantitative information through programs.*

  Similarly to weakest *liberal* preconditions, a *quantitative strongest liberal post* is also developed. As a byproduct, the notion of *strongest liberal postconditions* is obtained, and it shows how these foreshadow a potential new program logic — *partial incorrectness logic* — which would be a more liberal version of O'Hearn's incorrectness logic [6].

- **Chapter 4:** Based on the paper (and extensions of) Zhang et al. [18, Sections 4.1, 5.1, 5.2, 5.3]. This chapter presents a novel *strongest post calculus* for *reasoning about quantitative properties* over weighted programs, including both nondeterministic and probabilistic variants. By developing this calculus, we aim to facilitate forward reasoning about optimization problems, formal languages, and other domains where quantitative properties play a crucial role. Additionally, this approach will uncover novel dualities between forward and backward transformers, correctness and incorrectness, as well as nontermination and unreachability, shedding new light on fundamental aspects of program semantics and reasoning.

- **Chapter 5:** Based on the paper (and extensions of) Zhang et al. [18]. This chapter presents a novel *weakest pre calculus* for *reasoning about quantitative hyperproperties* over *nondeterministic and probabilistic* programs. Whereas existing calculi allow reasoning about the expected value that a quantity assumes after program termination from a *single initial state*, it does so for *initial sets of states* or *initial probability distributions.* It thus (i) obtains a weakest pre calculus for hyper Hoare logic and (ii) enables reasoning about so-called *hyperquantities* which include expected values but also quantities (e.g. variance) out of scope of previous work. Furthermore, the chapter will explore a forward variant of whp, denoted as shp, with the goal of offering a comprehensive perspective on quantitative hyperpredicate transformer reasoning and beyond.

- **Chapter 6:** Concludes the thesis by providing a discussion, a survey of related work, and outlining the ideas and directions for future work.

Chapter 2

# Preliminaries

## 2.1 Hoare Logic

Hoare logic (HL) is a formal system used to reason about the correctness of programs. Developed by Hoare [2] and building on earlier work by Floyd [62], Hoare logic uses a notation called *Hoare triples.* A Hoare triple is written as $\models \{\, \varphi \,\}\ C\ \{\, \psi \,\}$, where:

- $\varphi$ is the *precondition* that must be true before executing the program $C$.
- $C$ is the program or command being executed. In this chapter, we consider deterministic programs, meaning that the execution of $C$ will always produce the same output given the same input.
- $\psi$ is the *postcondition* that will be true after execution, assuming the precondition $\varphi$ was true.

For example, consider a simple program $C$ that increments a variable $x$ by 1. A possible Hoare triple could be $\models \{\, x = n \,\}\ C\ \{\, x = n + 1 \,\}$, which means that if $x$ starts with the value $n$, then after executing $C$, $x$ will be $n + 1$. Here, $n$ is a logical variable that does not occur in $C$, meaning it is used to represent a fixed value that $x$ holds before the execution of $C$ and is not modified by $C$. Notice the aspect of overapproximation: the triple $\models \{\, x = n \,\}\ C\ \{\, x \geq n + 1 \,\}$ would also be valid, since starting from $x = n$, we end exactly with $x = n + 1$, which satisfies the postcondition $x \geq n + 1$.

Hoare logic distinguishes between *partial correctness* and *total correctness* (in the sense of [75]).

### 2.1.1 Partial Correctness

A triple in $\mathsf{HL}$ for partial correctness (denoted $\models_{\mathrm{pc}} \{\, \varphi \,\}\; C \;\{\, \psi \,\}$) ensures that if the precondition $\varphi$ is true, the program $C$ will either terminate with the postcondition $\psi$ being true or it will not terminate (diverge). More precisely, the triple $\models_{\mathrm{pc}} \{\, \varphi \,\}\; C \;\{\, \psi \,\}$ is valid for partial correctness if:

Every initial state satisfying $\varphi$, after the execution of program $C$, will end up in a final state satisfying $\psi$ or will diverge.

**Example 2.1.1** (Partial Correctness). *Let $C = \mathtt{while}\,(\, x > 10 \,)\,\{\, x \coloneqq x + 1 \,\}$. The triple $\models_{\mathrm{pc}} \{\, x > 5 \,\}\; C \;\{\, x > 0 \,\}$ is valid because the postcondition $x > 0$ is an overapproximation of the state of $x$ after executing $C$ from an initial state where $x > 5$. In other words, if $x$ starts with a value greater than $5$, then $x$ will still be greater than $0$ after $C$ executes (since any operation within the program $C$ will not decrease $x$), assuming the program terminates. Nonterminating states are simply ignored and treated as acceptable behavior.*

### 2.1.2 Total Correctness

A triple in $\mathsf{HL}$ for total correctness (denoted $\models_{\mathrm{tc}} \{\, \varphi \,\}\; C \;\{\, \psi \,\}$) ensures that if the precondition $\varphi$ is true before executing the program $C$, the program will terminate and the postcondition $\psi$ will be true. Formally, the triple $\models_{\mathrm{tc}} \{\, \varphi \,\}\; C \;\{\, \psi \,\}$ is valid for total correctness if:

Every initial state satisfying $\varphi$, after the execution of program $C$, will end up in a final state satisfying $\psi$.

**Example 2.1.2** (Total Correctness). *Let $C = \mathtt{while}\,(\, x > 10 \,)\,\{\, x \coloneqq x + 1 \,\}$. The triple $\models_{\mathrm{tc}} \{\, x > 5 \,\}\; C \;\{\, x > 0 \,\}$ is not valid because starting from the precondition $x > 5$, the program might not terminate (e.g., if $x = 6$). However, the triple $\models_{\mathrm{tc}} \{\, 10 > x > 5 \,\}\; C \;\{\, x > 0 \,\}$ is valid, as the program is guaranteed to terminate (since the guard of the loop does not hold initially) with $x > 0$.*

Both examples Examples 2.1.1 and 2.1.2 demonstrate overapproximation in Hoare logic. The postcondition $x > 0$ is a broader condition than the exact state of $x$ after executing $C$. We are not concerned with the precise value of $x$, but rather with a condition ($x > 0$) that is certainly true given our preconditions.

## 2.2 Incorrectness Logic

The roots of incorrectness logic can be traced back to the work of de Vries and Koutavas [61], who introduced *reverse Hoare logic* to reason about reachability properties. Independently, O'Hearn [6] developed a closely related logic known as *Incorrectness Logic* (IL), which extends the principles of reverse Hoare logic to create a formal theory for bug-finding. Since then, several similar logics and extensions of IL for heaps and concurrent programs have been proposed [76; 77; 78; 79; 80; 81]. In IL, a triple $\models_{\text{ti}} [\,\varphi\,]\ C\ [\,\psi\,]$ expresses that the postcondition $\psi$ is an *underapproximation* of the set of reachable states when executing the program $C$ on a state satisfying the precondition $\varphi$. More precisely, a triple in IL $\models_{\text{ti}} [\,\varphi\,]\ C\ [\,\psi\,]$ is valid if:

Every final state satisfying $\psi$ is reachable by executing $C$
on some initial state satisfying $\varphi$.

**Example 2.2.1** (Incorrectness Logic). *Let* $C = \texttt{while}\,(\,x > 10\,)\,\{\,x \coloneqq x + 1\,\}$. *The triple* $\models_{\text{ti}} [\,\texttt{true}\,]\ C\ [\,x < 5\,]$ *is valid because all states satisfying the postcondition* $x < 5$ *are reachable from some initial state satisfying* $\texttt{true}$.

Here, observe that one can prove incorrectness by using an undesired property (e.g., a bug) as the postcondition $\psi$. In particular, Example 2.2.1 demonstrates underapproximation: the postcondition $x < 5$ represents a subset of the exact reachable states of $x$ after executing $C$. We are not focused on the precise value of $x$, but rather on a condition ($x < 5$) that is entirely reachable. Unlike Hoare Logic, we do not guarantee that the program will always terminate with $x < 5$, which is clearly not the case in this example.

In this sense, $\mathsf{IL}$ can witness the reachability of specific properties (e.g., bad outcomes) by proving that there exists *some execution* that ends in that specific property. However, unlike $\mathsf{HL}$, $\mathsf{IL}$ cannot be used to prove properties over *all executions*, such as guaranteeing that the program will always terminate with the correct outcome.

In the rest of the thesis we will also refer to $\mathsf{IL}$ as *total incorrectness*, to differentiate with our definition of *partial incorrectness* ( Definition 3.6.2).

## 2.3 Predicate Transformer Semantics

The logics developed for partial correctness [2; 62] and incorrectness [61; 6] introduce sufficient preconditions so that the postconditions are, respectively, an overapproximation and an underapproximation of the reachable states.

### 2.3.1 Weakest Preconditions

A different approach was originally introduced by Dijkstra [63; 1], who presented the *weakest precondition calculus.* Given a postcondition, this calculus allows us to find a necessary and sufficient precondition such that, if satisfied, the program will definitely terminate in a state that satisfies the postcondition. Symbolically, we denote by $\mathsf{wp}\,[\![C]\!]\,(\psi)$ the weakest precondition of a postcondition $\psi$, which is the largest precondition $\varphi$ making the Hoare triple $\models_{\text{tc}} \{\,\varphi\,\}\; C\; \{\,\psi\,\}$ valid. By having the weakest precondition, the validity of Hoare triples can be reduced to an implication check, as for all predicates $\varphi$ and $\psi$ we have:

$$\varphi \implies \mathsf{wp}\,[\![C]\!]\,(\psi) \quad \text{iff} \quad \models_{\text{tc}} \{\,\varphi\,\}\; C\; \{\,\psi\,\} \;\text{ is valid }.$$

We remark the fundamental difference with respect to Hoare triples for total correctness: if an initial state $\sigma$ does not satisfy $\mathsf{wp}\,[\![C]\!]\,(\psi)$, then executing program $C$ from $\sigma$ will definitely not terminate in a state satisfying $\psi$. Instead, if $\models \{\,\varphi\,\}\; C\; \{\,\psi\,\}$ is valid and $\sigma$ does not satisfy $\varphi$, then we have no information about the execution of program $C$ from $\sigma$.

An alternative predicate transformer has also been proposed by Dijkstra [1]:

the so-called *weakest liberal precondition* ($\mathsf{wlp}$), differently from its non-liberal version, includes in the precondition all the states that lead to nontermination, and thus is related to Hoare triples for *partial correctness*. Put formally, we have:

$$\varphi \implies \mathsf{wlp}[\![C]\!]\,(\psi) \quad \text{iff} \quad \models_{\text{pc}} \{\,\varphi\,\}\; C\; \{\,\psi\,\} \;\text{ is valid }.$$

The key difference between $\mathsf{wp}$ and $\mathsf{wlp}$ is in their handling of nontermination. The weakest precondition $\mathsf{wp}\,[\![C]\!]\,(\psi)$ strictly requires termination in a state satisfying $\psi$, making it suitable for reasoning about total correctness. Conversely, $\mathsf{wlp}[\![C]\!]\,(\psi)$ includes states from which $C$ may not terminate, reflecting partial correctness. This means that $\mathsf{wlp}$ is more permissive, allowing us to reason about the correctness of $C$ even if $C$ might not terminate. This distinction is critical when analyzing programs where termination cannot be guaranteed, as $\mathsf{wlp}$ still provides meaningful insights into the program's behavior under partial correctness criteria.

### 2.3.2 Strongest Postconditions

While weakest-precondition-style calculi are backward-moving, in the sense that they start from a postcondition and compositionally work backward through the program's instructions to derive an initial precondition, there also exists a forward-moving counterpart known as the *strongest postcondition* [64, Section 12]. The strongest postcondition calculus takes a precondition and returns the most precise (i.e., smallest) postcondition after the execution of a given program, which is exactly the set of all states that are reachable for the given precondition. Using Dijkstra's terminology [64, Section 11], the strongest postcondition is the *converse* transformer of the *weakest liberal precondition*; more concretely, denoted $\mathsf{sp}\,[\![C]\!]\,(\varphi)$ the strongest postcondition of a precondition $\varphi$, the following duality holds:

$$\varphi \implies \mathsf{wlp}[\![C]\!]\,(\psi) \quad \text{iff} \quad \mathsf{sp}\,[\![C]\!]\,(\varphi) \implies \psi\ .$$

In the literature, there has been a significant focus on quantitative weakest preconditions, which generalize the weakest precondition to reason about probabilistic and quantitative aspects of programs. Seminal work by Kozen [42, 13] on probabilistic programs laid the foundation for this area, and further extensions by McIver and Morgan [14] and others [16] explored expected values and other quantitative measures. Despite these advancements, the study of strongest postconditions has remained relatively underexplored.

In this thesis, we address this gap by introducing the concept of the strongest *liberal* postcondition (slp) in Section 3.4, demonstrating its relationship to unreachable states in a manner analogous to how wlp relates to nonterminating states. Furthermore, we will provide novel quantitative extensions for both sp and slp, which have not been developed previously. These contributions will significantly enhance the theoretical framework for program analysis, offering new tools for reasoning about both deterministic and nondeterministic program behaviors.

For a more detailed introduction to predicate transformer semantics, we recommend [64]. Additionally, [16] offers a comprehensive overview of their quantitative extensions.

## 2.4 Domain Theory

Domain theory is a significant field in mathematics and computer science that focuses on the study of specific types of partially ordered sets (posets) to model the semantics of computation. Originally developed by Dana Scott in the late 1960s [82], domain theory provides a rigorous mathematical foundation for understanding the semantics of programming languages, especially those involving recursion and infinite data structures.

The concepts of partially ordered sets and lattices are fundamental to defining the semantics of programming languages [83; 84]. For instance, these structures are commonly employed in denotational semantics [85], Abstract Interpretation [3] and Morphology [86].

For a more in-depth treatment of domain theory, readers are referred to the introduction provided in the Handbook of Logic in Computer Science [87]. However, to ensure that this thesis remains self-contained, we will briefly review some of the fundamental concepts that are critical for the discussions that follow.

### 2.4.1 Basics

A *relation* $R$ over two sets $X$ and $Y$ is a subset of $X \times Y$, namely $R \in \mathcal{P}(X \times Y)$ where $\mathcal{P}(X \times Y)$ denotes the powerset of $X \times Y$, i.e., the set of all subsets of $X \times Y$. The composition of two relations $R_1 \in \mathcal{P}(X \times Y)$, $R_2 \in \mathcal{P}(Y \times Z)$ is defined as

$$R_1 \circ R_2 \triangleq \{(x, z) \mid \exists (x, y) \in R_1 \ \wedge \ \exists (y, z) \in R_2\}.$$

A relation $R \in \mathcal{P}(X \times Y)$ is *total* if

$$\forall x \in X \colon \exists y \in Y \colon (x, y) \in R.$$

Let $R$ be a relation on $X^2$.

- $R$ is *reflexive* if $\forall x \in X \colon (x, x) \in R$.
- $R$ is *transitive* if $\forall x_1, x_2, x_3 \in X \colon (x_1, x_2) \in R \wedge (x_2, x_3) \in R \implies (x_1, x_3) \in R$.
- $R$ is *symmetric* if $\forall x_1, x_2 \in X \colon (x_1, x_2) \in R \iff (x_2, x_1) \in R$.
- $R$ is *antisymmetric* if $\forall x_1, x_2 \in X \colon (x_1, x_2) \in R \wedge (x_2, x_1) \in R \implies x_1 = x_2$.

If $R$ is reflexive, transitive, and symmetric, we say that $R$ is an *equivalence relation*. If $R$ is reflexive, transitive, and antisymmetric, we say that $R$ is a *partial order*. If $R$ is reflexive and transitive, we say that $R$ is a *quasiorder* (or *preorder*).

A *function* is a relation $R \in \mathcal{P}(X \times Y)$ where any element of $X$ is in relation with at most one element of $Y$, that is

$$\forall x \in X \colon \forall y_1, y_2 \in Y \colon (x, y_1) \in R \ \land \ (x, y_2) \in R \implies y_1 = y_2.$$

If $f \in \mathcal{P}(X \times Y)$ is a function from $X$ to $Y$, we write $f \colon X \to Y$.

### 2.4.2 Orders and Lattices

**Definition 2.4.1** (Partial order)**.** *A partial order on a set $X$ is a relation $\leq \subseteq X \times X$ such that the following properties hold:*

- *Reflexivity:* $\forall x \in X,\ (x, x) \in \leq$
- *Anti-symmetry:* $\forall x, y \in X,\ (x, y) \in \leq \quad \text{and} \quad (y, x) \in \leq \implies x = y$
- *Transitivity:* $\forall x, y, z \in X,\ (x, y) \in \leq \quad \text{and} \quad (y, z) \in \leq \implies (x, z) \in \leq$

Given a partial order $\leq$, we will use $\geq$ to denote the converse relation $\{(y, x) \mid (x, y) \in \leq\}$ and $<$ to denote $\{(x, y) \mid (x, y) \in \leq \text{ and } x \neq y\}$.

From now on we will use the notation $xRy$ to indicate $(x, y) \in R$.

**Definition 2.4.2** (Partially ordered set)**.** *A partially ordered set (or poset) is a pair $(X, \leq)$ in which $\leq$ is a partial order on $X$.*

We will use partially ordered sets to encode collections of program states.

A particularly important structure over posets are adjunctions [87, Definition 3.1.11]. Adjunctions have various equivalent definitions across the literature [87, Proposition 3.1.10] and are commonly referred to as Galois Connections in the context of programming languages [28, Definition 11.1], which is the term we will use throughout this thesis.

**Definition 2.4.3** (Galois connection)**.** *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be two partially ordered sets. The pair $\langle \alpha, \gamma \rangle$ of functions $\alpha \colon D \to A$ (lower adjoint) and $\gamma \colon A \to D$ (upper adjoint) is a Galois connection (GC) if and only if:*

$$\forall c \in C \colon \forall a \in A \colon \alpha(c) \leq a \iff c \sqsubseteq \gamma(a)$$

*In this case, we write* $\langle C, \sqsubseteq \rangle \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} \langle A, \leq \rangle$.

An example of Galois connection is the well-known duality between wlp and sp, see Section 3.6.1 for more.

We should note that our definition of a Galois connection is sometimes referred to as a monotonic Galois Connection [86, Definition 165], in contrast to an Antitone Galois Connection [86, Definition 166]. The terminology arises because, within our framework, we can establish the following lemma.

**Lemma 2.4.1** (Lemma 11.28, [28])**.** *If* $\langle C, \sqsubseteq \rangle \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} \langle A, \leq \rangle$ *then* $\alpha, \gamma$ *are monotonic.*

**Definition 2.4.4** (Meet-semilattice)**.** *A meet-semilattice is a partially ordered set* $(L, \leq)$ *such that for every pair of elements* $a, b \in L$*, there exists an element* $c \in L$ *satisfying the following conditions:*

*1.* $c \leq a$ *and* $c \leq b$

*2.* $\forall d \in L$*, if* $d \leq a$ *and* $d \leq b$*, then* $d \leq c$

*The element* $c$ *is called the* meet *or* greatest lower bound *of* $a$ *and* $b$*, and is denoted by* $a \wedge b$*.*

**Definition 2.4.5** (Join-semilattice)**.** *A join-semilattice is a partially ordered set* $(L, \leq)$ *such that for every pair of elements* $a, b \in L$*, there exists an element* $c \in L$ *satisfying the following conditions:*

*1.* $c \geq a$ *and* $c \geq b$

*2.* $\forall d \in L$*, if* $d \geq a$ *and* $d \geq b$*, then* $d \geq c$

*The element* $c$ *is called the* join *or* least upper bound *of* $a$ *and* $b$*, and is denoted by* $a \vee b$*.*

Both join and meet operations are idempotent, associative, and commutative.

**Definition 2.4.6** (Lattice)**.** *A poset $(L, \leq)$ is a lattice if it is both a join-semilattice and a meet-semilattice.*

**Definition 2.4.7** (Complete lattice)**.** *A partially ordered set $(L, \leq)$ is called a* complete lattice *if for every subset $S \subseteq L$, there exist elements* $\sup S$ *and* $\inf S$ *in L such that:*

1. $\sup S$ *(the supremum or least upper bound of $S$) is an element of $L$ satisfying:*
   - *For all $s \in S$, $s \leq \sup S$.*
   - *For any $u \in L$, if $s \leq u$ for all $s \in S$, then $\sup S \leq u$.*
2. $\inf S$ *(the infimum or greatest lower bound of $S$) is an element of $L$ satisfying:*
   - *For all $s \in S$, $\inf S \leq s$.*
   - *For any $l \in L$, if $l \leq s$ for all $s \in S$, then $l \leq \inf S$.*

Every complete lattice possesses a *least element* (or *bottom*), denoted by $\bot = \inf L$, and a *greatest element* (or *top*), denoted by $\top = \sup L$. Importantly, a complete lattice cannot be empty, as it must include at least the supremum of the empty set, $\sup \emptyset$. Additionally, every ascending or descending sequence within a complete lattice forms a subset, which ensures that these sequences converge, having a least upper bound or greatest lower bound, respectively.

### 2.4.3 Fixed Points

Fixed points play a crucial role in formalizing the semantics of programs and defining predicate transformers, which are key tools in reasoning about program correctness and behavior. The concept of a fixed point provides a foundation for understanding recursive definitions and iterative processes within a mathematical framework. In this section, we introduce the fundamental definitions and theorems related to fixed points that will be utilized throughout this thesis.

**Definition 2.4.8** (Fixed Points)**.** *Given a function $f \colon X \to X$, a fixed point of $f$ is an element $x \in X$ such that $x = f(x)$.*

*The set of all fixed points of $f$ is denoted by* $\mathrm{fix}(f)$*, defined as*

$$\mathrm{fix}(f) = \{x \in X \mid x = f(x)\}.$$

The notions of least and greatest fixed points are particularly important in the context of lattices and monotonic functions, as they allow us to characterize the minimal and maximal solutions to recursive equations.

**Definition 2.4.9** (Least and Greatest Fixed Points)**.** *Given a partially ordered set $(X, \leq)$ and a function $f\colon X \to X$,*

- *The least fixed point of $f$, denoted by* $\mathrm{lfp}(f)$*, is defined as the smallest element $a^* \in \mathrm{fix}(f)$ such that $a^* \leq a$ for all $a \in \mathrm{fix}(f)$.*
- *The greatest fixed point of $f$, denoted by* $\mathrm{gfp}(f)$*, is defined as the largest element $a^* \in \mathrm{fix}(f)$ such that $a^* \geq a$ for all $a \in \mathrm{fix}(f)$.*

The structures we consider in this thesis, particularly the sets on which predicate transformers operate, are typically complete lattices. This structure is essential, as it ensures the existence of least and greatest fixed points for certain classes of functions. Specifically, functions that are Scott-continuous, or simply continuous, are guaranteed to have these fixed points, making them highly relevant in the study of domain theory and fixed-point semantics [87].

**Definition 2.4.10** (Continuity [87])**.** *Let $(D, \leq)$ be a complete lattice and let $\Phi\colon D \to D$. Let $\Phi(S)$ denote the set $\{\Phi(a) \mid a \in S\}$. The function $\Phi$ is called:*

1. *continuous if and only if for every ascending chain $S = \{s_0 \leq s_1 \leq s_2 \leq \ldots\} \subseteq D$,*
$$\Phi(\sup S) = \sup \Phi(S),$$
2. *co-continuous if and only if for every descending chain $S = \{s_0 \geq s_1 \geq s_2 \geq \ldots\} \subseteq D$,*
$$\Phi(\inf S) = \inf \Phi(S).$$

The notion of continuity employed here, known as $\omega$-(co)continuity, requires that the function preserves the supremum (infimum) of chains of countable length ($\omega$). It is worth noting that under this definition, continuity implies monotonicity.

**Lemma 2.4.2** (Monotonicity of continuous and co-continuous functions)**.** *Every continuous or cocontinuous function is monotonic.*

*Proof.* Let $a, b$ such that $a \leq b$. Consider a continuous function $\Phi$. We have:

$$\begin{aligned} \Phi(a) &\leq \sup\ \{\Phi(a), \Phi(b)\} \\ &= \sup\ \Phi(\{a, b\}) \\ &= \Phi(\sup\ \{a, b\}) && \text{(by continuity of } \Phi) \\ &= \Phi(b) \end{aligned}$$

and hence $\Phi$ is monotonic. Similarly, consider now a co-continuous function $\Psi$; we have:

$$\begin{aligned} \Psi(a) &= \Psi(\inf\ \{a, b\}) \\ &= \inf\ \Psi(\{a, b\}) && \text{(by co-continuity of } \Psi) \\ &= \inf\ \{\Psi(a), \Psi(b)\} \\ &\leq \Psi(b) \end{aligned}$$

and hence $\Psi$ is monotonic. □

With these fundamental concepts established, we are now prepared to discuss two important fixed-point theorems extensively utilized in this thesis. The origins of these theorems trace back to Bronisław Knaster's work in 1928, who proved a weaker result in set theory [88; 89]. Subsequent theorems were proved by Tarski [90] for arbitrary suprema-preserving functions and later extended by Scott [91] and Cousot and Cousot [92]. Erroneously attributed to Stephen Cole Kleene [93], it is often referred to as a folk theorem [89].

**Theorem 2.4.3** (Kleene Fixed Point Theorem, [87; 89])**.** *Let $(D, \leq)$ be a complete lattice with a least element $\bot$ and a greatest element $\top$. Moreover, let $\Phi\colon D \to D$ be continuous and $\Psi\colon D \to D$ be co-continuous (therefore, $\Phi, \Psi$ are monotonic). Then $\Phi$ has a least fixed point* $\operatorname{lfp}\Phi$ *and $\Psi$ a greatest fixed point* $\operatorname{gfp}\Psi$*, given by:*

$$\operatorname{lfp}\Phi = \sup_{n\in\mathbb{N}} \Phi^n(\bot) \quad \textit{and} \quad \operatorname{gfp}\Psi = \inf_{n\in\mathbb{N}} \Psi^n(\top).$$

The Kleene Fixed Point Theorem will be employed to constructively demonstrate the existence of least fixed points, which are heavily utilized in our definitions of transformers for while loops. Additionally, we will rely on an induction principle known as Park's Lemma, Park's Theorem, or Park induction—also referred to as Fixpoint Induction [28, Theorem 22.1]—originally attributed to David Park.

**Lemma 2.4.4** (Park's Lemma, [94])**.** *Let $(D, \leq)$ be a complete lattice, let $d \in D$, and let $\Phi\colon D \to D$ be a monotonic function. Then:*

$$\Phi(d) \leq d \quad \textit{implies} \quad \operatorname{lfp}\Phi \leq d,$$

*and dually,*

$$d \leq \Phi(d) \quad \textit{implies} \quad d \leq \operatorname{gfp}\Psi.$$

The correctness of many induction rules for loops presented in this thesis follows directly from Park's Lemma. This concludes our overview of the domain-theoretical concepts necessary for this work.

Chapter 3

# Quantitative Strongest Post

This chapter is essentially a verbatim of the paper by Zhang and Kaminski [17], with minor adaptations to fit the thesis format. We introduce a novel **strongest-postcondition-style calculus for quantitative reasoning** about non-deterministic programs with loops. Whereas existing quantitative weakest pre allows reasoning about the value of a quantity *after* a program terminates on a given *initial state*, quantitative strongest post allows reasoning about the value that a quantity had *before* the program was executed and reached a given *final state*. We show how strongest post enables reasoning about the *flow of quantitative information through programs*.

Similarly to weakest *liberal* preconditions, we also develop a *quantitative strongest liberal post*. As a byproduct, we obtain the notion of ***strongest liberal postconditions*** and show how these foreshadow a potential new program logic — ***partial incorrectness logic*** — which would be a more liberal version of O'Hearn's Incorrectness logic [6].

## 3.1 Introduction

### Partial Correctness

Already in one of the earliest works on program verification, Turing [59] separates reasoning about partial correctness and termination. Partial correctness means that the program is correct, *if* it terminates. Nontermination is in that sense deemed "correct" behavior. *Hoare triples* [2] capture partial correctness formally: Given program $C$ and predicates $G, F$, we say that $\models \{\, G \,\}\; C \;\{\, F \,\}$ is *valid for partial correctness*, if from every state $\sigma$ satisfying precondition $G$, $C$ *either* terminates in some state satisfying postcondition $F$, *or* $C$ does *not* terminate on $\sigma$.

A different approach to partial correctness are the *weakest liberal preconditions* of Dijkstra [63]: Given program $C$ and postcondition $F$, the weakest liberal precondition is the weakest (largest) predicate $\mathsf{wlp}[\![C]\!]\,(F)$, such that starting from any state $\sigma$ satisfying the precondition $\mathsf{wlp}[\![C]\!]\,(F)$, $C$ *either* terminates in some state satisfying the postcondition $F$, *or* $C$ does not terminate on $\sigma$. $\mathsf{wlp}[\![C]\!]\,(\ldots)$ is a called a backward-moving *predicate transformer semantics*, because it transforms a postcondition (a predicate) $F$ into a precondition (another predicate) $\mathsf{wlp}[\![C]\!]\,(F)$.

A different predicate transformer semantics are the forward-moving *strongest postconditions* of Dijkstra and Scholten [64]: they transform a precondition $G$ into the strongest (smallest) predicate $\mathsf{sp}\,[\![C]\!]\,(G)$, such that $\mathsf{sp}\,[\![C]\!]\,(G)$ contains all states that can be reached by executing $C$ on some state satisfying the precondition $G$. Hoare triples, weakest liberal preconditions, and strongest postconditions are strongly related by the following well-known fact:

$$\begin{gathered}\models \{\, G \,\}\; C \;\{\, F \,\} \;\text{ is valid for partial correctness}\\ \text{iff}\\ G \implies \mathsf{wlp}[\![C]\!]\,(F)\\ \text{iff}\\ \mathsf{sp}\,[\![C]\!]\,(G) \implies F~.\end{gathered}$$

Having a choice between $\mathsf{wlp}$ and $\mathsf{sp}$ is beneficial because sometimes the partial correctness proof can be easier in the, say, forward direction than in the backward direction.

## Quantitative Verification

*Backward-moving* predicate transformers have been generalized to *real-valued-function transformers*, first by Kozen [13], in order to reason about probabilistic programs, e.g. about the *probability* that some postcondition will be satisfied after program termination. For the forward direction, Jones [95] presented a counterexample to the existence of probabilistic strongest postconditions. While we will defer reasoning about probabilistic programs in Chapter 4, we will in this chapter develop a *quantitative strongest post transformer for reasoning about nondeterministic programs.*

Intuitively, quantitative predicate-transformer-style calculi lift reasoning

$$\begin{array}{rl} \text{from} & \textit{predicates } F\colon \mathsf{States} \to \{\mathsf{true},\, \mathsf{false}\} \\ \text{to} & \textit{quantities } f\colon \mathsf{States} \to \mathbb{R}^{\pm\infty}\ , \end{array}$$

i.e. functions $f$ that associate a *real number* (or $+\infty$ or $-\infty$) to each state. Given a *post*quantity $f$ associating a number to *final* states, our backward-moving weakest liberal pre transformer $\mathsf{wlp}[\![C]\!]\,(f) : \mathsf{States} \to \mathbb{R}^{\pm\infty}$ associates numbers to *initial* states, so that $\mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)$ *anticipates* what value $f$ will have after $C$ terminates on $\sigma$ (and $\mathsf{wlp}$ *anticipates* $+\infty$ if $C$ does not terminate on $\sigma$).

For example, what is the anticipated value of $2x$ after executing the assignment $x \coloneqq x+1$? Our quantitative weakest liberal pre calculus will push the "assertion" $2x$ backward through the program, obtaining the annotations on the right (read from bottom to top).

$$\begin{array}{l} /\!\!/\!\!/\ \ \boldsymbol{2x+2} \\ x \coloneqq x+1 \\ /\!\!/\!\!/\ \ \boldsymbol{2x} \end{array}$$

Indeed, given an *initial* value $x_\sigma = 5$ for the program variable $x$, the *final* value of the expression $2x$ will be $2x_\sigma + 2 = 2 \cdot 5 + 2 = 12$.

While counterintuitive — since $\mathsf{wlp}$ moves backwards —, $\mathsf{wlp}$ acts like a weather *forecast*: Given the current state $\sigma$ of the global atmosphere, a function $f$ mapping atmosphere state to the temperature in Auckland, and an (algorithmic) description $C$ of how the atmosphere evolves within 24 hours, $\mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)$ anticipates *now* what the temperature in Auckland will be *tomorrow*.

In this chapter, we develop a *quantitative strongest post transformer* $\mathsf{sp}$ with as strong a connection (more precisely: a *Galois connection*) to quantitative

$\mathsf{wlp}$ as in the qualitative case, namely

$$g \preceq \mathsf{wlp}[\![C]\!]\,(f) \qquad \text{iff} \qquad \mathsf{sp}\,[\![C]\!]\,(g) \preceq f\ .$$

Dually to $\mathsf{wlp}$, our forward-moving strongest post transformer acts like a weather *backcast*: Given the current global atmosphere state $\tau$, $\mathsf{sp}\,[\![C]\!]\,(f)\,(\tau)$ *retrocipates* now what the temperature in Auckland was *yesterday*. Speaking in terms of programs and quantities, given a *pre*quantity $f$ associating a number to *initial* states, $\mathsf{sp}\,[\![C]\!]\,(f) : \mathsf{States} \to \mathbb{R}^{\pm\infty}$ associates numbers to *final* states, such that $\mathsf{sp}\,[\![C]\!]\,(f)\,(\tau)$ *retrocipates* what value $f$ had in an initial state before $C$ terminated in $\tau$ (and $\mathsf{sp}$ *retrocipates* $-\infty$ if $\tau$ is not reachable by executing $C$ on some initial state).

For example, what is the retrocipated value of $2x$ before the assignment $x \coloneqq x + 1$? Our quantitative strongest post calculus will push the "assertion" $2x$ forward through the program, obtaining the annotations on the right (read from top to bottom). Indeed, given a *final* value $x_\tau = 5$ for the program variable $x$, the *initial* value of the expression $2x$ must have been $2x_\tau - 2 = 2 \cdot 5 - 2 = 8$.

```
/// $\boldsymbol{2x}$
$x \coloneqq x + 1$
/// $\boldsymbol{2x - 2}$
```

Notably, our quantitative strongest post transformer provides some notion of *flow of quantitative information through the program*: If we start the above program with initial value $x_\sigma = 4$ for $x$, then we have initially $2x_\sigma = 2 \cdot 4 = 8$. After the execution of the program, the final value of $x$ is $x_\tau = 5$. The expression $2x - 2$ evaluated in $x_\tau$ is again $2 \cdot x_\tau = 2 \cdot 5 - 2 = 8$. In that sense, our quantitative $\mathsf{sp}$ takes a quantity — for instance: a secret value — and propagates through the program an expression which *preserves* the value of the initial quantity. Given some final state, we can hence read off what the quantity was initially and so reason about quantitative flow and leakage of information.

## Contributions and Organization

*Not* being our main contribution, we present in Sec. 3.3 quantitative $\mathsf{wp}$ and $\mathsf{wlp}$. Differently from [14; 96; 16], our quantitative transformers act on *signed* unbounded quantities in $\mathbb{R}^{\pm\infty}$, whereas traditional probabilistic $\mathsf{wlp}$ act on $[0,\, 1]$ and $\mathsf{wp}$ on $\mathbb{R}^{\infty}_{\geq 0}$.

In Section 3.4, we present our *main contribution*: a novel quantitative strongest post transformer $\mathsf{sp}$ as described above. Moreover, we provide a quantitative strongest *liberal* post transformer $\mathsf{slp}$, which gives a different value than $\mathsf{sp}$ to *unreachable* states (whereas $\mathsf{wlp}$ gives a different value than $\mathsf{wp}$ to *nonterminating* states). We study essential properties of all our transformers in Section 3.5 and show how they embed reasoning about predicates à la Dijkstra and Scholten [64].

In Section 3.6, we show that $\mathsf{slp}$ has as tight a (Galois) connection to $\mathsf{wp}$ as $\mathsf{sp}$ to $\mathsf{wlp}$, namely

$$\mathsf{wp}\,[\![C]\!]\,(f) \;\preceq\; g \qquad \text{iff} \qquad f \;\preceq\; \mathsf{slp}[\![C]\!]\,(g)\;\;.$$

When restricting to predicates, our $\mathsf{slp}$ transformer yields the novel notion of *strongest liberal postconditions*, later investigated by Verscht et al. [97]; Oda [98] among others. While it is known that strongest postconditions are tightly connected with the recent *incorrectness logic* of O'Hearn [6], we show how $\mathsf{slp}$ foreshadows a new program logic — *partial incorrectness logic.* We also hint at two further new program logics: one of *necessary liberal preconditions* and one of *necessary liberal postconditions.*

In Section 3.7, we present proof rules for loops for all four quantitative transformers. In Section 3.8 we demonstrate efficacy of $\mathsf{sp}$ and $\mathsf{slp}$ for reasoning about the flow of quantitative information.

## 3.2 Nondeterministic Programs

The syntax of the *nondeterministic guarded command language* (nGCL) à la Dijkstra is given by

$$
\begin{array}{rcll}
C & ::= & \texttt{skip} & \text{(effectless statement)} \\
 & \mid & x \coloneqq e & \text{(assignment)} \\
 & \mid & C \fatsemi C & \text{(sequential composition)} \\
 & \mid & \{\, C \,\} \,\square\, \{\, C \,\} & \text{(nondeterministic choice)} \\
 & \mid & \texttt{if}\ (\,\varphi\,)\ \{\, C \,\}\ \texttt{else}\ \{\, C \,\} & \text{(conditional choice)} \\
 & \mid & \texttt{while}\,(\,\varphi\,)\,\{\, C \,\}\ , & \text{(loop)}
\end{array}
$$

where $x \in \mathsf{Vars}$ is a variable, $e$ is an arithmetic expression and $\varphi$ is a predicate. A program state $\sigma$ is a function that assigns an integer to each program variable. The set of program states is given by

$$\Sigma = \{\, \sigma \mid \sigma : \mathsf{Vars} \to \mathbb{Z} \,\} \ .$$

Given a program state $\sigma$, we denote by $\sigma(\xi)$ the evaluation of an arithmetic or Boolean expression $\xi$ in $\sigma$, i.e. the value that is obtained by evaluating $\xi$ after replacing any occurrence of any variable $x$ in $\xi$ by the value $\sigma(x)$. Moreover, we denote by $\sigma\,[x/v]$ a new state that is obtained from $\sigma$ by setting the valuation of the variable $x \in \mathsf{Vars}$ to $v \in \mathbb{Z}$. Formally:

$$\sigma\,[x/v] \quad = \quad \lambda y\colon \begin{cases} v & \text{if } y = x \\ \sigma(y) & \text{otherwise} \end{cases} .$$

We assign meaning to our nondeterministic nGCL-statements in terms of a denotational *collecting* semantics (as is standard in program analysis, see [3; 99; 100]), i.e. we have as input a *set of initial states* and as output the *set of reachable states*.

**Definition 3.2.1** (Collecting Semantics for nGCL Programs)**.** *Let* $\mathtt{Conf} = \mathcal{P}(\Sigma)$ *be the set of* program configurations, *i.e. a single configuration is a* set of program states*; and let* $[\![\varphi]\!]S = \{\sigma \mid \sigma \in S \wedge \sigma \models \varphi\}$ *be a filtering of a program configuration to only those states where the predicate* $\varphi$ *holds.*

*The collecting semantics* $[\![C]\!] \colon \mathtt{Conf} \to \mathtt{Conf}$ *of an* nGCL *program* $C$ *is defined inductively by*

$$\begin{aligned}
[\![\mathtt{skip}]\!]S &= S && \text{(effectless program)}\\
[\![x \coloneqq e]\!]S &= \{\sigma\,[x/\sigma(e)] \mid \sigma \in S\} && \text{(assignment)}\\
[\![C_1 \,⨟\, C_2]\!]S &= ([\![C_2]\!] \circ [\![C_1]\!])S && \text{(sequential composition)}\\
[\![\mathtt{if}\ (\,\varphi\,)\ \{\,C_1\,\}\ \mathtt{else}\ \{\,C_2\,\}]\!]S &= ([\![C_1]\!] \circ [\![\varphi]\!])S \cup ([\![C_2]\!] \circ [\![\neg\varphi]\!])S &&\\
& && \text{(conditional choice)}\\
[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]S &= [\![\neg\varphi]\!]\big(\mathsf{lfp}\ X\colon S \ \cup\ \big([\![C]\!] \circ [\![\varphi]\!]\big)X\big) && \text{(loop)}\\
[\![\{\,C_1\,\} \,\Box\, \{\,C_2\,\}]\!]S &= [\![C_1]\!]S \cup [\![C_2]\!]S\ . && \text{(nondeterministic choice)}
\end{aligned}$$

*By slight abuse of notation, we write* $[\![C]\!](\sigma)$ *for* $[\![C]\!]\{\sigma\}$. ◁

Let us explain the semantics of $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$. Let $S$ again be the set of input states. First, we denote by $F_S$ the function

$$F_S(X) \;=\; S \;\cup\; \big([\![C]\!] \circ [\![\varphi]\!]\big)X\ ,$$

i.e. $F_S$ first applies the filtering with respect to the loop guard $\varphi$ to its input $X$, then applies the semantics of the loop body $C$ to the filtered set, and finally unions that result with the given set of input states $S$. Using $F_S$, the collecting semantics for while loops can be expressed as

$$[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]S \;=\; [\![\neg\varphi]\!]\big(\mathsf{lfp}\ X\colon F_S(X)\big)\ ,$$

where the least fixed point above is understood with respect to the partial order of set inclusion, which renders the structure $\langle \mathtt{Conf},\, \subseteq\rangle$ a complete lattice with least element $\emptyset$. The least fixed point above filtered by $\neg\varphi$ expresses exactly the set $[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]S$ of final states reachable after termination of $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ starting from any initial state in $S$. We remark that to determine the least fixed point of the continuous function $F_S$, it is sufficient to apply Kleene's fixpoint theorem and, as a result, we have that the infinite ascending chain $\emptyset \subseteq F_S^1(\emptyset) \subseteq F_S^2(\emptyset) \subseteq \ldots F_S^\omega(\emptyset)$, where $F_S^{i+1}(X) = F_S(F_S^i(X))$, converges in at most $\omega$ iterations.

**Example 3.2.1** (Collecting Semantics of While Loops). *Assume there is only a single program variable $x$ and consider the configuration $S = \{\{x \mapsto 0\}, \{x \mapsto 8\}\}$. We now want to execute the loop $\texttt{while}\,(\,x > 5\,)\,\{\,x \coloneqq x + 1\,\}$ on this configuration and collect the reachable states. By our definition above, we have*

$$[\![\texttt{while}\,(\,x > 5\,)\,\{\,x \coloneqq x+1\,\}]\!]S \;=\; [\![x \leq 5]\!]\big(\mathsf{lfp}\ X\colon F_S(X)\big)\ , \quad \text{where}$$

$$\begin{aligned} F_S(X) \;&=\; S \,\cup\, \big([\![C]\!] \circ [\![\varphi]\!]\big)X \\ &=\; \{\{x \mapsto 0\}, \{x \mapsto 8\}\} \,\cup\, \{\,\sigma\,[x/x+1] \mid \sigma \in X,\ \sigma(x) > 5\,\}\ . \end{aligned}$$

*The Kleene iterates are:*

$$\begin{aligned} F(\emptyset) \;&=\; \big\{\{x \mapsto 0\}, \{x \mapsto 8\}\big\} \,\cup\, \emptyset \\ F^2(\emptyset) \;&=\; \big\{\{x \mapsto 0\},\, \{x \mapsto 8\}\big\} \,\cup\, \big\{\{x \mapsto 9\}\big\} \\ F^2(\emptyset) \;&=\; \big\{\{x \mapsto 0\},\, \{x \mapsto 8\}\big\} \,\cup\, \big\{\{x \mapsto 9\},\, \{x \mapsto 10\}\big\} \\ &\;\;\vdots \\ F^\omega(\emptyset) \;&=\; \big\{\{x \mapsto 0\}\big\} \,\cup\, \big\{\,\{x \mapsto i\} \;\big|\; i \geq 9\,\big\} \end{aligned}$$

*After filtering $F^\omega(\emptyset)$ by the negation of the loop guard, we obtain the loop's collecting semantics*

$$[\![\texttt{while}\,(\,x > 5\,)\,\{\,x \coloneqq x+1\,\}]\!]S \;=\; [\![x \leq 5]\!]\big(F^\omega(\emptyset)\big) \;=\; \big\{\{x \mapsto 0\}\big\}\ . \quad \triangleleft$$

## 3.3 Weakest Pre

We develop novel weakest (liberal) pre calculi á la Dijkstra [63] for *quantitative reasoning* about nondeterministic programs. While we repeat that the weakest pre calculi are not our main contribution (that being the quantitative strongest post calculi), we believe that weakest pre calculi are easier to understand and provide the necessary intuition for moving from the Boolean to the quantitative realm. We first shortly recap Dijkstra's classical weakest preconditions before we lift them to a quantitative setting. Thereafter, we lift weakest *liberal* preconditions to quantities.

### 3.3.1 Classical Weakest Preconditions

Dijkstra's weakest precondition calculus employs *predicate transformers* of type

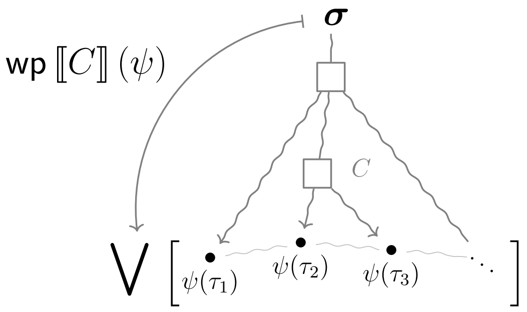
$$\mathsf{wp}[\![C]\!]\colon \quad \mathbb{B} \;\to\; \mathbb{B}\;, \qquad \text{where} \quad \mathbb{B} \;=\; \{0,\,1\}^{\Sigma}\;,$$

which associate to each nondeterministic program $C$ a mapping from predicates (sets of program states) to predicates. Somewhat less common, we consider here an *angelic* setting, where the nondeterminism is resolved to our advantage.[1] Specifically, the *angelic* weakest precondition transformer $\mathsf{wp}[\![C]\!]$ maps a *post*-condition $\psi$ over final states to a *pre*condition $\mathsf{wp}\,[\![C]\!]\,(\psi)$ over initial states, such that executing the program $C$ on an initial state satisfying $\mathsf{wp}\,[\![C]\!]\,(\psi)$ guarantees that $C$ *can*[2] terminate in a final state satisfying $\psi$. More symbolically, recalling that $[\![C]\!](\sigma)$ is the set of all final states reachable after termination of $C$ on $\sigma$,

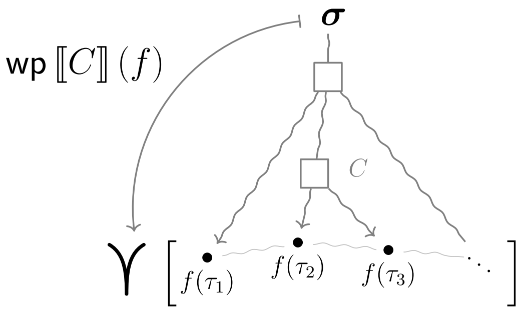
$$\sigma \;\models\; \mathsf{wp}\,[\![C]\!]\,(\psi) \qquad \text{iff} \qquad \exists\,\tau \in [\![C]\!](\sigma)\colon \quad \tau \;\models\; \psi\;.$$

[1]Considering an angelic setting allows us not only to show that our transformers enjoy several properties, but also to provide tight connections between quantitative weakest preconditions and quantitative strongest postconditions.

[2]Recall that $C$ is a *nondeterministic* program. For the (standard) *demonic* setting as well as for deterministic programs, we can *replace "can" by "will"*.



**(a) Weakest preconditions:** Given initial state $\sigma$, $\mathsf{wp}\,[\![C]\!]\,(\psi)$ determines all final states $\tau_i$ reachable from executing $C$ on $\sigma$, evaluates $\psi$ in those states, and returns the disjunction ($\vee$) over all these truth values.



**(b) Quantitative weakest pre:** Given initial state $\sigma$, $\mathsf{wp}\,[\![C]\!]\,(f)$ determines all final states $\tau_i$ reachable from executing $C$ on $\sigma$, evaluates $f$ in those states, and returns the supremum ($\curlyvee$) over all these quantities.

**Figure 3.1:** (Angelic) weakest precondition and quantitative weakest precondition

While the above is a *set perspective* on $\mathsf{wp}$, an equivalent perspective on $\mathsf{wp}$ is a *map perspective*, see Figure 3.1a: The postcondition $\psi\colon \Sigma \to \{0,\, 1\}$ maps program states to truth values. The predicate $\mathsf{wp}\,[\![C]\!]\,(\psi)$ is then a map that takes as input an initial state $\sigma$, determines for each reachable final state $\tau \in [\![C]\!](\sigma)$ the (truth) value $\psi(\tau)$, takes a disjunction over all these truth values, and finally returns the truth value of that disjunction. More symbolically,

$$\mathsf{wp}\,[\![C]\!]\,(\psi)\,(\sigma) \qquad = \qquad \bigvee_{\tau \in [\![C]\!](\sigma)} \psi(\tau)\ .$$

It is this map perspective which we will now gradually lift to a quantitative setting. For that, we first need to leave the realm of Boolean valued predicates and move to *real-valued* functions.

## 3.3.2 Quantities

For our development here, we are interested in *signed* quantities. Such quantities form — just like first-order logic for weakest preconditions — the *assertion “language” of our quantitative calculi.*

**Definition 3.3.1** (Quantities)**.** *The set of all* quantities *is defined by*

$$\mathbb{A} \;=\; \big\{ f \;\big|\; f\colon \Sigma \to \mathbb{R}^{\pm\infty} \big\}$$

*i.e. the set of all functions $f\colon \Sigma \to \mathbb{R}^{\pm\infty}$ associating an* extended real *(i.e. either a proper real number, or $-\infty$, or $+\infty$) to each program state. The point-wise order*

$$f \;\preceq\; g \qquad \text{iff} \qquad \forall\, \sigma \in \Sigma\colon \quad f(\sigma) \;\leq\; g(\sigma)$$

*renders $\langle \mathbb{A},\, \preceq \rangle$ a complete lattice with join $\curlywedge$ and meet $\curlyvee$, given point-wise by*

$$f \curlywedge g \;=\; \lambda\sigma\colon \min\big\{f(\sigma),\, g(\sigma)\big\} \quad \text{and} \quad f \curlyvee g \;=\; \lambda\sigma\colon \max\big\{f(\sigma),\, g(\sigma)\big\} \;.$$

*Joins and meets over arbitrary subsets exist. When we write $a \curlyvee b \curlywedge c$, we assume that $\curlywedge$ binds stronger than $\curlyvee$, so we read that as $a \curlyvee (b \curlywedge c)$.* $\lhd$

**Remark 3.3.1** (Signed Quantities)**.** *Kozen [13] also considers* signed *functions for reasoning about probabilistic programs. However, Kozen's induction rule for while loops only applies to non-negative functions, see [13, page 168]. Kaminski and Katoen [101] have rules for probabilistic loops and signed functions, but their machinery is quite involved and their rule for loops is more involved than simple induction. Our development in this chapter is — on the plus-side — comparatively simple, but — as a trade-off — we cannot handle probabilistic programs.* $\lhd$

### 3.3.3 Quantitative Weakest Pre

We now define a calculus á la Dijkstra for formal reasoning about the value of a quantity $f \in \mathbb{A}$ after execution of a nondeterministic program. For that, we generalize the *map perspective* of weakest preconditions to quantities. Instead of a post*condition*, we now have a post*quantity* $f\colon \Sigma \to \mathbb{R}^{\pm\infty}$ mapping (final) program states to extended reals. $\mathsf{wp}\, [\![C]\!]\, (f) : \Sigma \to \mathbb{R}^{\pm\infty}$ is then a function that takes as input an initial state $\sigma$, determines all final states $\tau$ reachable from executing $C$ on $\sigma$, evaluates the postquantity $f(\tau)$ in each final state $\tau$, and finally returns the supremum over all these so-determined quantities, see Figure 3.1b. If the program is completely *deterministic* and if $C$ terminates on input $\sigma$, then $\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)$ *anticipates* the single possible value that $f$ will have, evaluated in the final state that is reached after executing $C$ on $\sigma$.

One of the main advantages of Dijkstra's calculus is that the weakest preconditions can be defined by induction on the program structure, thus allowing for *compositional reasoning*. Indeed, the same applies to our quantitative setting.

**Definition 3.3.2** (Quantitative Weakest Pre)**.** *The* weakest pre transformer

$$\mathsf{wp}\colon\quad \mathsf{nGCL} \to (\mathbb{A} \to \mathbb{A})$$

*is defined inductively according to the rules in* Table 3.1*. We call the function*

$$\Phi_f(X) \;=\; [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(X)\;\;,$$

*whose* least *fixed point defines the weakest pre* $\mathsf{wp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)$*, the* $\mathsf{wp}$–characteristic function *(of* $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$ *with respect to* $f$*).* ◁

Let us show for some of the rules how the quantitative weakest pre semantics can be developed and understood analogously to Dijkstra's classical weakest preconditions.

**Effectless Program.**

For the effectless program `skip`, what is the anticipated value of $f$ after executing `skip`? It is again just $f$, and so $\mathsf{wp}\,[\![\mathtt{skip}]\!]\,(f) = f$.

**Assignment.**

The weakest pre*condition* of an assignment is given by

$$\mathsf{wp}\,[\![x \coloneqq e]\!]\,(\psi) \;=\; \psi\,[x/e]\;\;,$$

| $C$ | $\mathsf{wp}\,[\![C]\!]\,(f)$ |
|---|---|
| `skip` | $f$ |
| `diverge` | $-\infty$ |
| $x \coloneqq e$ | $f\,[x/e]$ |
| $C_1 \fatsemi C_2$ | $\mathsf{wp}\,[\![C_1]\!]\,\big(\mathsf{wp}\,[\![C_2]\!]\,(f)\big)$ |
| $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$ | $\mathsf{wp}\,[\![C_1]\!]\,(f) \;\curlyvee\; \mathsf{wp}\,[\![C_2]\!]\,(f)$ |
| `if` $(\,\varphi\,)\,\{\, C_1 \,\}$ `else` $\{\, C_2 \,\}$ | $[\varphi] \curlywedge \mathsf{wp}\,[\![C_1]\!]\,(f) \;\curlyvee\; [\neg\varphi] \curlywedge \mathsf{wp}\,[\![C_2]\!]\,(f)$ |
| `while` $(\,\varphi\,)\,\{\, C' \,\}$ | $\mathsf{lfp}\ X\colon \quad [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C']\!]\,(X)$ |

**Table 3.1:** Rules for the weakest pre transformer. Here, $\mathsf{lfp}\ f\colon \Phi(f)$ denotes the least fixed point of $\Phi$.

where $\psi\,[x/e]$ is the replacement of every occurrence of variable $x$ in the postcondition $\psi$ by the expression $e$. For *quantitative* weakest pre, we can do something completely analogous, except that we do not have a syntax like first-order logic for the postquantities at hand.[3] Still, we can define semantically what it means to "syntactically replace" every "occurrence" of $x$ in $f$ by $e$ — and with it the quantitative weakest pre of an assignment — as follows:

$$\mathsf{wp}\,[\![x \coloneqq e]\!]\,(f) \;=\; f\,[x/e] \;\coloneqq\; \lambda\,\sigma\colon\, f\Big(\sigma\,[x \mapsto \sigma(e)]\Big)\;.$$

So what is the value of $f$ in the final state reached after executing the assignment $x \coloneqq e$ on initial state $\sigma$? It is precisely $f$, but evaluated at the final state $\sigma\,[x \mapsto \sigma(e)]$ — the state obtained from $\sigma$ by updating variable $x$ to value $\sigma(e)$.

**Nondeterministic Choice.**

When "executing" the nondeterministic choice $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$ on some initial state $\sigma$, *either* $C_1$ *or* $C_2$ will be executed, chosen nondeterministically. Hence, the execution will reach either a final state in which executing $C_1$ on $\sigma$ terminates or a final state in which executing $C_2$ on $\sigma$ terminates (or no final state if both computations diverge).

[3]For probabilistic programs, an expressive and relatively complete (with respect to taking weakest preexpectations) *syntax* for expressing functions (expectations) of type $\Sigma \to \mathbb{R}_{\geq 0}^{\infty}$ has been presented in [102].

Denotationally, the *angelic* weakest pre*condition* of $\{\, C_1 \,\} \;\Box\; \{\, C_2 \,\}$ is given by

$$\mathsf{wp}\,[\![\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}]\!]\,(\psi) \;\;=\;\; \mathsf{wp}\,[\![C_1]\!]\,(\psi) \;\;\vee\;\; \mathsf{wp}\,[\![C_2]\!]\,(\psi) \;\;.$$

Indeed, whenever an initial state $\sigma$ satisfies the precondition $\mathsf{wp}\,[\![C_1]\!]\,(\psi) \vee \mathsf{wp}\,[\![C_2]\!]\,(\psi)$, then — either by executing $C_1$ or by executing $C_2$ — it is possible that the computation will terminate in some final state satisfying the postcondition $\psi$.

Quantitatively, what is the anticipated value of $f$ after termination of either $C_1$ or $C_2$? Since $C_1$ and $C_2$ could both terminate but very well yield different values for $f$, we need to accommodate for two different numbers. In the maximizing spirit of *angelic* $\mathsf{wp}$, we also maximize and select as *quantitative* weakest pre of $\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}$ the *largest* possible final value of $f$ via the join

$$\mathsf{wp}\,[\![\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}]\!]\,(f) \;\;=\;\; \mathsf{wp}\,[\![C_1]\!]\,(f) \;\;\curlyvee\;\; \mathsf{wp}\,[\![C_2]\!]\,(f) \;\;.$$

**Diverge.**

`diverge` is a shorthand for `while` ( $\mathsf{true}$ ) { `skip` } — the certainly diverging loop. Denotationally, the weakest pre*condition* of `diverge` is given by

$$\mathsf{wp}\,[\![\texttt{diverge}]\!]\,(\psi) \;\;=\;\; \mathsf{false}\;.$$

As there is *no* initial state that satisfies $\mathsf{false}$, this simply tells us that there is no initial state on which `diverge` could possibly terminate in any final state satisfying $\psi$.

Note that the predicate $\mathsf{false}$ is the *least element* in the Boolean lattice. When lifting this to a quantitative setting, we also assign the least element. Hence,

$$\mathsf{wp}\,[\![\texttt{diverge}]\!]\,(f) \;\;=\;\; -\infty\;.$$

Another explanation goes by considering again the *angelic*, i.e. maximizing, aspect of quantitative weakest pre: What is the maximal value that we can anticipate for $f$ *after* `diverge` has terminated? Since `diverge` does not terminate at all (but we are still forced to assign some "number" to this situation), the largest value that we can possibly anticipate is the absolute minimum: $-\infty$.

**Remark 3.3.2** (Quantitative Weakest Pre and Nontermination)**.** *In some sense,* $-\infty$ *is* the value of nontermination *in quantitative* $\mathsf{wp}$*. Note that it is more tedious to detect nontermination by standard weakest preconditions: Consider e.g. the program* `diverge` *and postcondition "*$x$ is odd*". Then*

$$\mathsf{wp}\,[\![\texttt{diverge}]\!]\,(x \text{ is odd}) \;=\; \mathsf{false}\ .$$

*On the other hand, for the* terminating *program* $x \coloneqq 2 \cdot x$*, we also have*

$$\mathsf{wp}\,[\![x \coloneqq 2 \cdot x]\!]\,(x \text{ is odd}) \;=\; 2 \cdot x \text{ is odd} \;=\; \mathsf{false}\ .$$

*Thus,* $\mathsf{wp}\,[\![C]\!]\,(\psi)\,(\sigma) = \mathsf{false}$ *is not a sufficient criterion for detecting nontermination of* $C$ *on* $\sigma$*.* $\mathsf{false}$ *merely tells us that the program* either *does not terminate* or *it fails to establish the postcondition. To distinguish the two cases, one needs to check,* additionally*, whether* $\sigma$ *terminates, i.e., whether* $\mathsf{wp}\,[\![C]\!]\,(\mathsf{true})\,(\sigma)$ *holds.*

*In* our *quantitative* $\mathsf{wp}$ *calculus, given any non-infinite postquantity* $f$*, our wp transformer distinguishes whether the program terminates or not* in one go*. Indeed, if* $-\infty \;\preceq\; f \;\preceq\; +\infty$ *and* $\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \;=\; 0$*, then definitely* $C$ *terminates on* $\sigma$ *and* $f$ *assumes value 0 after termination of* $C$ *on* $\sigma$*. For instance, for postquantity* $x$ *we have*

$$\mathsf{wp}\,[\![\texttt{diverge}]\!]\,(x) \;=\; -\infty \qquad \text{and} \qquad \mathsf{wp}\,[\![x \coloneqq 2 \cdot x]\!]\,(x) \;=\; 2 \cdot x\ ,$$

*and can thus read off that the program* `diverge` *indeed does not terminate, whereas, since* $x > -\infty$*, we can see that* $x \coloneqq 2 \cdot x$ *does always terminate.* $\triangleleft$

**Conditional Choice.**

When executing $\texttt{if}\ (\varphi)\ \{C_1\}\ \texttt{else}\ \{C_2\}$ on some initial state $\sigma$, the branch $C_1$ is executed $\sigma$ satisfies the predicate $\varphi$ and otherwise $C_2$ is executed.

Denotationally, the weakest precondition of $\texttt{if}\ (\varphi)\ \{C_1\}\ \texttt{else}\ \{C_2\}$ is given by

$$\mathsf{wp}\,[\![\texttt{if}\ (\varphi)\ \{C_1\}\ \texttt{else}\ \{C_2\}]\!]\,(\psi) \;=\; \varphi \wedge \mathsf{wp}\,[\![C_1]\!]\,(\varphi) \vee \neg\varphi \wedge \mathsf{wp}\,[\![C_2]\!]\,(\varphi)\ ,$$

where — as usual — $\wedge$ binds stronger than $\vee$. Indeed, whenever an initial state $\sigma$ satisfies the above precondition then *either* $\sigma \models \varphi$ and then — since then $\sigma$ must also satisfy $\mathsf{wp}\,[\![C_1]\!]\,(\psi)$ — executing $C_1$ can terminate in a final state satisfying $\varphi$, *or* $\sigma \not\models \varphi$ and then — since then $\sigma$ must also satisfy $\mathsf{wp}\,[\![C_2]\!]\,(\psi)$ — executing $C_2$ can terminate in a final state satisfying $\varphi$.

In order to mimic the above in a quantitative setting, we make use of so called *Iverson brackets* [103]. Usually, these turn a predicate $\varphi$ into an indicator function $[\varphi]_{\mathsf{std}} \colon \Sigma \to \{0,\,1\}$, which map a state $\sigma$ to 1 or 0, depending on whether $\sigma \models \varphi$ or not. In our extended real setting, however, we need to slightly adapt the Iverson brackets as follows:

**Definition 3.3.3** (Extended Iverson Brackets). *For a predicate $\varphi$, we define the extended Iverson bracket $[\varphi] \colon \Sigma \to \{-\infty,\,+\infty\}$ by*

$$[\varphi]\,(\sigma) \;=\; \begin{cases} +\infty & \text{if } \sigma \models \varphi \\ -\infty & \text{otherwise.} \end{cases} \qquad \triangleleft$$

Intuitively, this choice is motivated by the fact that $-\infty, +\infty$ are respectively the bottom and top element of the lattice, and equipped with $\curlyvee, \curlywedge$, they behave exactly as the boolean values $\mathsf{true}, \mathsf{false}$ with $\vee, \wedge$. Using these Iverson brackets, we define the *quantitative* weakest pre of conditional choice by

$$\mathsf{wp}\,[\![\texttt{if}\ (\varphi)\ \{C_1\}\ \texttt{else}\ \{C_2\}]\!]\,(f) \;=\; [\varphi] \curlywedge \mathsf{wp}\,[\![C_1]\!]\,(f) \curlyvee [\neg\varphi] \curlywedge \mathsf{wp}\,[\![C_2]\!]\,(f)$$

(Recall that $\curlywedge$ binds stronger than $\curlyvee$.) If the current program state $\sigma$ satisfies $\varphi$, then $[\varphi]$ evaluates to $+\infty$ — the *greatest* element of $\mathbb{A}$. Taking a minimum ($\curlywedge$) with $\mathsf{wp}\,[\![C_1]\!]\,(f)$ will thus yield exactly $\mathsf{wp}\,[\![C_1]\!]\,(f)$. $[\neg\varphi]$, on the other hand, then evaluates to $-\infty$ — the *smallest* element of $\mathbb{A}$. Taking a minimum with any other lattice element will again yield $-\infty$. Finally, we then take a maximum ($\curlyvee$) between $\mathsf{wp}\,[\![C_1]\!]\,(f)$ and $-\infty$, yielding $\mathsf{wp}\,[\![C_1]\!]\,(f)$. This is precisely the quantity that we would expect to anticipate for $f$, if $\sigma \models \varphi$, because then $C_1$ is executed and $\mathsf{wp}\,[\![C_1]\!]\,(f)$ anticipates the value of $f$ after execution of $C$. The situation for $\sigma \not\models \varphi$ is completely dual, yielding $\mathsf{wp}\,[\![C_2]\!]\,(f)$. Indeed, depending on whether an initial state satisfies $\varphi$ or not, the quantitative weakest pre anticipates *either* $\mathsf{wp}\,[\![C_1]\!]\,(f)$ *or* $\mathsf{wp}\,[\![C_2]\!]\,(f)$.

**Remark 3.3.3.** *We note that our* $\mathsf{wp}$ *rule for conditional choice is different from e.g. [13; 104; 16], who use standard instead of extended Iverson brackets, multiplication instead of minimum, and summation instead of maximum, i.e.*

$$\mathsf{wp}\,[\![\texttt{if}\ (\,\varphi\,)\ \{\,C_1\,\}\ \texttt{else}\ \{\,C_2\,\}]\!]\,(f) = [\varphi]_{\mathit{std}} \cdot \mathsf{wp}\,[\![C_1]\!]\,(f) + [\neg\varphi]_{\mathit{std}} \cdot \mathsf{wp}\,[\![C_2]\!]\,(f)$$

*This rule, however, would fail in our context of signed quantities because of issues with* $+\infty \cdot -\infty$. $\triangleleft$

**Sequential Composition.**

What is the anticipated value of $f$ after executing $C_1 \fatsemi C_2$, i.e. the value of $f$ after first executing $C_1$ and then $C_2$? To answer this, we first anticipate the value of $f$ after execution of $C_2$ which gives $\mathsf{wp}\,[\![C_2]\!]\,(f)$. Then, we anticipate the value of the intermediate quantity $\mathsf{wp}\,[\![C_2]\!]\,(f)$ after execution of $C_1$, yielding

$$\mathsf{wp}\,[\![C_1 \fatsemi C_2]\!]\,(f) = \mathsf{wp}\,[\![C_1]\!]\,(\mathsf{wp}\,[\![C_2]\!]\,(f))\,.$$

**Looping.**

The quantitative weakest pre of a loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ is defined as a least fixed point of the $\mathsf{wp}$*–characteristic function* $\Phi_f\colon \mathbb{A} \to \mathbb{A}$. This function is chosen in a way so that iterating $\Phi_f$ on the least element of the lattice $-\infty$ essentially yields an ascending chain of loop unrollings

$$\begin{aligned}\Phi_f(-\infty) &= \mathsf{wp}\,[\![\texttt{if}(\varphi)\{\texttt{diverge}\}]\!]\,(f)\\ \Phi_f^2(-\infty) &= \mathsf{wp}\,[\![\texttt{if}(\varphi)\{C \fatsemi \texttt{if}(\varphi)\{\texttt{diverge}\}\}]\!]\,(f)\\ \Phi_f^3(-\infty) &= \mathsf{wp}\,[\![\texttt{if}(\varphi)\{C \fatsemi \texttt{if}(\varphi)\{C \fatsemi \texttt{if}(\varphi)\{\texttt{diverge}\}\}\}]\!]\,(f)\end{aligned}$$

and so on, whose supremum is the least fixed point of $\Phi_f$.

Let us now state in which sense our weakest pre semantics is sound:

**Theorem 3.3.1** (Characterization of $\mathsf{wp}$). *For all programs $C$ and initial states $\sigma$,*

$$\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \quad = \quad \bigcurlyvee_{\tau \in [\![C]\!](\sigma)} f(\tau)~.$$

Intuitively, for a given postquantity $f$ and initial state $\sigma$, $\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)$ is the *supremum* over all the values that $f$ can assume measured in the final states reached after successful termination of the program $C$ on initial state $\sigma$. In case of no terminating state, i.e. $[\![C]\!](\sigma) = \emptyset$, that supremum becomes $-\infty$ — the absolute minimal value. In particular, if $\forall\, \tau\colon f(\tau) > -\infty$, then $\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) = -\infty$ unambiguously indicates *nontermination* of $C$ on input $\sigma$.

### 3.3.4 Weakest Liberal Pre

Besides weakest preconditions, Dijkstra also defines weakest *liberal* preconditions. The *weakest liberal precondition transformer* is again of type

$$\mathsf{wlp}[\![C]\!]\colon \quad \mathbb{B} \to \mathbb{B}~,$$

associating to each nondeterministic program $C$ a mapping from predicates to predicates. For reasons of duality, we now consider a *demonic* setting, where the nondeterminism is resolved to our *dis*advantage. The difference from *non*liberal weakest preconditions, however, is that *nonterminating* behavior is deemed *good behavior* (i.e. as if the program terminated in a state satisfying the postcondition). Specifically, the *demonic* weakest liberal precondition transformer $\mathsf{wlp}[\![C]\!]$ maps a *post*condition $\psi$ over final states to a *pre*condition $\mathsf{wlp}[\![C]\!]\,(\psi)$ over initial states, such that executing $C$ on an initial state satisfying $\mathsf{wlp}[\![C]\!]\,(\psi)$ guarantees that $C$ will either not terminate, or terminate in a final state satisfying $\psi$. More symbolically, recalling that $[\![C]\!](\sigma)$ is the set of all final states reachable after termination of $C$ on $\sigma$,

$$\sigma \;\models\; \mathsf{wlp}[\![C]\!]\,(\psi) \qquad \text{iff} \qquad \forall\, \tau \in [\![C]\!](\sigma)\colon \quad \tau \;\models\; \psi\;,$$

where the right-hand-side of the implication is vacuously true if $[\![C]\!](\sigma) = \emptyset$, i.e. if $C$ does not terminate on $\sigma$. From the *map perspective*, $\mathsf{wlp}[\![C]\!]\,(\psi)$ is a function that takes as input an initial state $\sigma$, determines for each reachable final state $\tau \in [\![C]\!](\sigma)$ the (truth) value $\psi(\tau)$, and returns a *conjunction* over all these truth values. More symbolically,

$$\mathsf{wlp}[\![C]\!]\,(\psi)\,(\sigma) \qquad = \qquad \bigwedge_{\tau \in [\![C]\!](\sigma)} \psi(\tau)\;,$$

where the conjunction over an empty set is — as is standard — given by $\mathsf{true}$.

Just like a conjunction in some sense minimizes truth values, our quantitative weakest liberal pre should also minimize, while at the same time assigning a maximal value to nontermination. This is captured by the following transformer:

**Definition 3.3.4** (Quantitative Weakest Liberal Pre)**.** *The* quantitative weakest liberal pre transformer

$$\mathsf{wlp}\colon \quad \mathsf{nGCL} \to (\mathbb{A} \to \mathbb{A})$$

*is defined inductively according to the rules in* Table 3.2 *(right column). We*

| $C$ | $\mathsf{wlp}\,[\![C]\!]\,(f)$ |
|---|---|
| `skip` | $f$ |
| `diverge` | $+\infty$ |
| $x \coloneqq e$ | $f\,[x/e]$ |
| $C_1 \fatsemi C_2$ | $\mathsf{wlp}[\![C_1]\!]\,\big(\mathsf{wlp}[\![C_2]\!]\,(f)\big)$ |
| $\{\,C_1\,\} \mathrel{\square} \{\,C_2\,\}$ | $\mathsf{wlp}[\![C_1]\!]\,(f) \;\curlywedge\; \mathsf{wlp}[\![C_2]\!]\,(f)$ |
| $\texttt{if}\;(\,\varphi\,)\;\{\,C_1\,\}\;\texttt{else}\;\{\,C_2\,\}$ | $[\varphi] \curlywedge \mathsf{wlp}[\![C_1]\!]\,(f) \;\curlyvee\; [\neg\varphi] \curlywedge \mathsf{wlp}[\![C_2]\!]\,(f)$ |
| $\texttt{while}\,(\,\varphi\,)\,\{\,C'\,\}$ | $\mathsf{gfp}\;X\colon\quad [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C']\!]\,(X)$ |

**Table 3.2:** Rules for the weakest liberal pre transformer. Here, $\mathsf{gfp}\; f\colon \Phi(f)$ denotes the greatest fixed point of $\Phi$.

*call the function*

$$\Phi_f(X) \;=\; [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(X)\;,$$

*whose* greatest *fixed point defines* $\mathsf{wlp}[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)$*, the* $\mathsf{wlp}$–characteristic function *(of* $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ *with respect to* $f$*).* $\lhd$

The rules for assignments, sequential composition, and conditional choice are the same as for $\mathsf{wp}$. This is unsurprisingly so, since those rules pertain neither to nontermination nor to nondeterminism. Let us thus go over the rules for the language constructs, where the rules for $\mathsf{wlp}$ and $\mathsf{wp}$ differ.

**Diverge.**

Since `diverge` is certainly nonterminating and liberal pre*conditions* deem this good behavior, the weakest liberal precondition of `diverge` is given by

$$\mathsf{wlp}[\![\texttt{diverge}]\!]\,(\psi) \;=\; \mathsf{true}\;.$$

Note that $\mathsf{true}$ is the *greatest element* in the Boolean lattice. When moving to quantities, we also assign to nonterminating behavior the greatest element, i.e.

$$\mathsf{wlp}[\![\texttt{diverge}]\!]\,(f) \;=\; +\infty\;.$$

**Remark 3.3.4** (Quantitative Weakest Liberal Pre and Nontermination)**.** *Analogously to $-\infty$ being the* the value of nontermination *in* $\mathsf{wp}$ *(see Remark 3.3.2),* $+\infty$ *is* the value of nontermination *in* $\mathsf{wlp}$. $\lhd$

**Nondeterministic Choice.**

Since weakest *liberal* pre is *demonic*, we ask in $\mathsf{wlp}$ for the *minimal* anticipated value of $f$ after termination of $C_1$ or $C_2$. Hence the rule is dually given by the meet

$$\mathsf{wlp}[\![\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}]\!]\,(f) \;=\; \mathsf{wlp}[\![C_1]\!]\,(f) \;\curlywedge\; \mathsf{wlp}[\![C_2]\!]\,(f) \enspace .$$

Notice that if either $C_1$ or $C_2$ yield $+\infty$ because of nontermination, the $\mathsf{wlp}$ above will select as value the respective other branch if that one terminates.

**Looping.**

The weakest liberal pre of a loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ is defined as a greatest fixed point of the $\mathsf{wlp}$–*characteristic function* $\Phi_f\colon \mathbb{A} \to \mathbb{A}$. This function is chosen in a way so that iterating $\Phi_f$ on the greatest element of the lattice $+\infty$ essentially yields a descending chain of loop unrollings

$$\begin{aligned}
\Phi_f(+\infty) &= \mathsf{wlp}[\![\texttt{if}(\varphi)\{\texttt{diverge}\}]\!]\,(f)\\
\Phi_f^2(+\infty) &= \mathsf{wlp}[\![\texttt{if}(\varphi)\{C\,⨾\;\texttt{if}(\varphi)\{\texttt{diverge}\}\}]\!]\,(f)\\
\Phi_f^3(+\infty) &= \mathsf{wlp}[\![\texttt{if}(\varphi)\{C\,⨾\;\texttt{if}(\varphi)\{C\,⨾\;\texttt{if}(\varphi)\{\texttt{diverge}\}\}\}]\!]\,(f)
\end{aligned}$$

and so on, whose infimum is the greatest fixed point of $\Phi_f$.

After having provided an intuition, let us now state in which sense our weakest liberal pre semantics is sound:

**Theorem 3.3.2** (Characterization of $\mathsf{wlp}$)**.** *For all programs $C$ and states*

$\sigma \in \Sigma$,

$$\mathsf{wlp}[\![C]\!]\,(f)\,(\sigma) \quad = \quad \bigcurlywedge_{\tau \in [\![C]\!](\sigma)} f(\tau)~.$$

Intuitively, for a given postquantity $f$ and initial state $\sigma$, the quantitative weakest liberal pre $\mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)$ is the *infimum* over all values that $f$ can assume measured in the final states after termination of the program $C$ on initial state $\sigma$. In case of no terminating state, i.e. $[\![C]\!](\sigma) = \emptyset$, that infimum automatically becomes $+\infty$ — the absolute maximal value. In particular, if $\forall\, \tau\colon f(\tau) < +\infty$, then $\mathsf{wlp}[\![C]\!]\,(f)\,(\sigma) = +\infty$ unambiguously indicates *non-termination* of $C$ on input $\sigma$.

## 3.4 Strongest Post

We now present our main contribution: A lifting of the strongest postcondition calculus of Dijkstra and Scholten [64] to quantities and a completely novel (quantitative) strongest *liberal* post calculus. To the best of our knowledge, a strongest liberal post(condition) has never been proposed before, not even in the *qualitative* setting.[4] We again start by recapping the classical calculus.

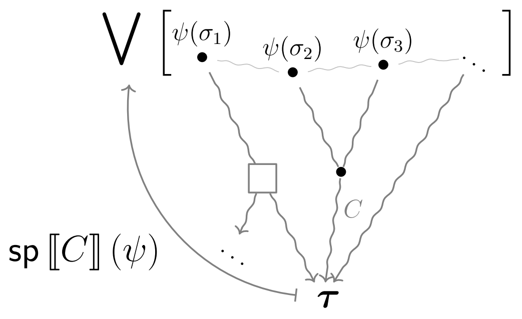
### 3.4.1 Classical Strongest Postconditions

Dijkstra and Scholten's strongest postcondition calculus employs *predicate transformers* of type

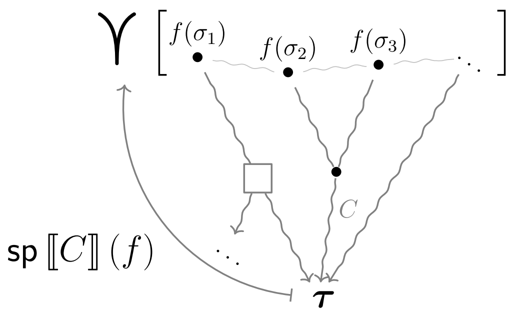
$$\mathsf{sp}[\![C]\!]\colon \quad \mathbb{B} \ \to\ \mathbb{B}~, \qquad \text{where} \quad \mathbb{B} \ =\ \Sigma \to \{0,\, 1\}~,$$

which associate to each nondeterministic program $C$ a mapping from predicates (sets of program states) to predicates. Strongest post transformers, analogously to the collecting semantics, characterize the set states that *can* be reached, so that an *angelic* setting is chosen to resolve nondeterminism to our advantage. Concretely, the *angelic* strongest postcondition transformer $\mathsf{sp}[\![C]\!]$ maps a *pre-*

[4]Although some authors do use the term “strongest liberal postcondition”, see Section 3.9 for a comparison.



**(a) Strongest postconditions:** Given final state $\tau$, $\mathsf{sp}\,[\![C]\!]\,(\psi)$ determines all initial states $\sigma_i$ that can reach $\tau$ by executing $C$, evaluates $\psi$ in those states, and returns the disjunction over all these truth values.



**(b) Quantitative strongest post:** Given final state $\tau$, $\mathsf{sp}\,[\![C]\!]\,(f)$ determines all initial states $\sigma_i$ that can reach $\tau$ by executing $C$, evaluates $f$ in those states, and returns the supremum ($\curlyvee$) over all these quantities.

**Figure 3.2:** Angelic strongest postconditions and quantitative strongest posts.

condition $\psi$ over initial states to a *post*condition $\mathsf{sp}\,[\![C]\!]\,(\psi)$ over final states, such that every state in the postcondition *is reachable* from some initial state satisfying $\psi$. This corresponds exactly with the definition of the collecting semantics $[\![C]\!](\sigma)$: In fact,

$$\tau \models \mathsf{sp}\,[\![C]\!]\,(\psi) \qquad \text{iff} \qquad \exists\,\sigma \text{ with } \tau \in [\![C]\!](\sigma)\colon \quad \sigma \models \psi~.$$

As we did for weakest pre, let us provide a *map perspective* on strongest postconditions, see Figure 3.2a. From this perspective, the precondition $\psi\colon \Sigma \to \{0,\, 1\}$ maps program states to truth values. The predicate $\mathsf{sp}\,[\![C]\!]\,(\psi)$ is then a map that takes as input a *final* state $\tau$, determines for all initial states $\sigma$ that can reach $\tau$ the (truth) value $\psi(\sigma)$, and returns the disjunction ($\vee$) over all these truth values:

$$\mathsf{sp}\,[\![C]\!]\,(\psi)\,(\tau) \qquad = \qquad \bigvee_{\sigma \text{ with } \tau\in[\![C]\!](\sigma)} \psi(\sigma)~.$$

In other words: *Given a final state $\tau$, $\mathsf{sp}\,[\![C]\!]\,(\psi)\,(\tau)$ <u>retro</u>dicts whether before executing $C$ the predicate $\psi$ could have been true.* In the following, we define quantitative strongest post and strongest liberal post calculi which *<u>retrocipate</u>*

values of signed quantities *before* the execution of a nondeterministic program (whereas $\mathsf{wp}$ and $\mathsf{wlp}$ *anticipate* values after the execution).

### 3.4.2 Quantitative Strongest Post

Let us generalize the *map perspective* of strongest postconditions to quantities. Instead of a pre*condition*, we now have a pre*quantity* $f\colon \Sigma \to \mathbb{R}^{\pm\infty}$. $\mathsf{sp}\,[\![C]\!]\,(f) : \Sigma \to \mathbb{R}^{\pm\infty}$ is then a function that takes as input a *final* state $\tau$, determines all initial states $\sigma$ that can reach $\tau$ by executing $C$, evaluates the prequantity $f(\sigma)$ in each of those initial states $\sigma$, and finally returns the supremum over all these so-determined quantities, see Figure 3.2b. As a transformer, we obtain the following:

**Definition 3.4.1** (Quantitative Strongest Post)**.** *The* strongest post transformer

$$\mathsf{sp}\colon \quad \mathsf{nGCL} \to (\mathbb{A} \to \mathbb{A})$$

*is defined inductively according to the rules in* Table 3.3*. We call the function*

$$\Psi_f(X) \;=\; f \;\curlyvee\; \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge X)\;\;,$$

*whose* least *fixed point is used to define* $\mathsf{sp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)$*, the* $\mathsf{sp}$*–characteristic function of* $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ *with respect to* $f$*.* ◁

Again, let us go over some of the rules for quantitative $\mathsf{sp}$ and show how they can be developed and understood analogously to strongest postconditions.

#### Effectless Program

What is the retrocipated value of $f$ before executing `skip`? It is again just $f$. $\mathsf{sp}\,[\![\texttt{skip}]\!]\,(f)$ is hence the identity function.

| $C$ | $\mathsf{sp}\,[\![C]\!]\,(f)$ |
|---|---|
| `skip` | $f$ |
| `diverge` | $-\infty$ |
| $x \coloneqq e$ | $\mathcal{S}\alpha\colon\; [x = e\,[x/\alpha]] \;\curlywedge\; f\,[x/\alpha]$ |
| $C_1 ⨾ C_2$ | $\mathsf{sp}\,[\![C_2]\!]\,\big(\mathsf{sp}\,[\![C_1]\!]\,(f)\big)$ |
| $\{\,C_1\,\} \,\square\, \{\,C_2\,\}$ | $\mathsf{sp}\,[\![C_1]\!]\,(f) \;\curlyvee\; \mathsf{sp}\,[\![C_2]\!]\,(f)$ |
| `if` $(\,\varphi\,)\,\{\,C_1\,\}$ `else` $\{\,C_2\,\}$ | $\mathsf{sp}\,[\![C_1]\!]\,([\varphi] \curlywedge f) \;\curlyvee\; \mathsf{sp}\,[\![C_2]\!]\,([\neg\varphi] \curlywedge f)$ |
| `while` $(\,\varphi\,)\,\{\,C'\,\}$ | $[\neg\varphi] \curlywedge \big(\mathsf{lfp}\, Y\colon f \curlyvee \mathsf{sp}\,[\![C']\!]\,([\varphi] \curlywedge Y)\big)$ |

**Table 3.3:** Rules for the strongest post transformer. Here, $\mathsf{lfp}\; f\colon \Phi(f)$ denotes the least fixed point of $\Phi$ and $\mathcal{S}\alpha\colon f(\alpha)$ denotes a supremum of $f(\alpha)$ ranging over all values of $\alpha$.

**Assignment.**

Dijkstra and Scholten's strongest postcondition of an assignment is given by

$$\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\psi) \qquad = \qquad \exists\,\alpha\colon\quad \underbrace{x = e\,[x/\alpha]}_{(1)} \;\wedge\; \underbrace{\psi\,[x/\alpha]}_{(2)}\,.$$

Intuitively, the quantified $\alpha$ represents an *initial* value that $x$ could have had *before* executing the assignment. (If at all possible), the $\alpha$ is chosen in a way so that

1. $x$ has in the final state the value of expression $e$ but evaluated using $x$'s initial value $\alpha$, and

2. the precondition $\psi$ was true in the initial state where $x$ had value $\alpha$.

For quantities, we note that, regarding (1), there could have been multiple valid initial values $\alpha$ for $x$; for instance, before the execution of $x \coloneqq 10$, *any* initial value $\alpha$ is valid. Our intuition is that, in order to preserve backward compatibility, we substitute the *existential* quantifier with a *supremum* (denoted by the $\mathcal{S}$ "quantifier", cf. [102]), thus obtaining the supremum of $f\,[x/\alpha]$ ranging over all valid initial values $\alpha$ of $x$:

$$\mathsf{sp}\,[\![x \coloneqq e]\!]\,(f) \quad = \quad \text{Ƨ}\alpha\colon \quad [x = e\,[x/\alpha]] \;\curlywedge\; f\,[x/\alpha]~.$$

Let us consider a few examples. First, consider

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq x+1]\!]\,(x) &= \text{Ƨ}\alpha\colon\; [x = \alpha + 1] \;\curlywedge\; \alpha \\
&= \text{Ƨ}\alpha\colon\; [\alpha = x - 1] \;\curlywedge\; \alpha \\
&= x - 1~.
\end{aligned}$$

For a final state $\tau(x) = 10$, this gives us $\tau(x) - 1 = 10 - 1 = 9$ which is indeed *the* initial value that the prequantity $x$ must have had if the final state after executing $x \coloneqq x + 1$ is $\tau(x) = 10$.

As another example, consider

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq 10]\!]\,(x) &= \text{Ƨ}\alpha\colon\; [x = 10] \;\curlywedge\; \alpha \\
&= [x = 10] \;\curlywedge\; \infty \\
&= [x = 10]~.
\end{aligned}$$

For the final state $\tau(x) = 10$, this gives us $[10 = 10] = [\mathsf{true}] = +\infty$ which is indeed the *least upper bound* (angelic!) on the initial value of $x$ if the final state after executing $x \coloneqq 10$ is $\tau$. In other words: by evaluating $[x = 10]$ in $\tau$, we know that $\tau$ was reachable, but we have no information on what maximal value $x$ could have had initially, which is sensible because $x \coloneqq 10$ forgets any initial value of $x$. For final state $\tau'(x) = 9$, on the other hand, we get $[9 = 10] = [\mathsf{false}] = -\infty$ which is the *value of unreachability* in $\mathsf{sp}$ (cf. also the next paragraph on divergence). Indeed, the final state after executing $x \coloneqq 10$ cannot ever be $\tau'$.

**Diverge.**

The strongest postcondition of `diverge` is given by

$$\mathsf{sp}\,[\![\texttt{diverge}]\!]\,(\psi) \quad = \quad \mathsf{false}\ ,$$

the *least element* in the Boolean lattice. Since there is *no* state that satifies $\mathsf{false}$, this simply tells us that there is no final state reachable by executing `diverge`.

For quantities, we also assign the least element and hence get

$$\mathsf{sp}\,[\![\texttt{diverge}]\!]\,(f) \quad = \quad -\infty\ .$$

Another explanation goes by considering again the *angelic*, i.e. maximizing, aspect of strongest post: What is the maximal value that we can retrocipate for $f$ *before* `diverge` has terminated in some final state $\tau$? Since `diverge` does not terminate at all and hence no such $\tau$ could have been reached (but we are still forced to assign some "number" to this situation), the largest value that we can possibly retrocipate is the absolute minimum: $-\infty$.

**Remark 3.4.1** (Quantitative Strongest Post and Unreachability)**.** *Dually to values of nontermination in* $\mathsf{w(l)p}$ *(see Remarks 3.3.2 and 3.3.4),* $-\infty$ *is in that sense* the value of unreachability *in* $\mathsf{sp}$. ◁

**Nondeterministic Choice.**

The *angelic* strongest post*condition* of $\{\,C_1\,\} \,\square\, \{\,C_2\,\}$ is given by

$$\mathsf{sp}\,[\![\{\,C_1\,\} \,\square\, \{\,C_2\,\}]\!]\,(\psi) \quad = \quad \mathsf{sp}\,[\![C_1]\!]\,(\psi) \;\;\vee\;\; \mathsf{sp}\,[\![C_2]\!]\,(\psi)\ .$$

Indeed, the set of reachable states starting from initial states satisfying $\psi$ is the union of the reachable set after executing $C_1$ and the ones after executing $C_2$.

In a quantitative setting, where we want to retrocipate the value of a quantity $f$ before executing either $C_1$ or $C_2$, we *angelically* maximize between the two retrocipated quantities:

$$\mathsf{sp}\,[\![\{\,C_1\,\}\;\Box\;\{\,C_2\,\}]\!]\,(f) \;\;=\;\; \mathsf{sp}\,[\![C_1]\!]\,(f) \;\;\curlyvee\;\; \mathsf{sp}\,[\![C_2]\!]\,(f)\;\;.$$

**Conditional Choice.**

The strongest post*condition* of $\texttt{if}\;(\,\varphi\,)\;\{\,C_1\,\}\;\texttt{else}\;\{\,C_2\,\}$ is given by

$$\mathsf{sp}\,[\![\texttt{if}\;(\,\varphi\,)\;\{\,C_1\,\}\;\texttt{else}\;\{\,C_2\,\}]\!]\,(\psi) \;\;=\;\; \mathsf{sp}\,[\![C_1]\!]\,(\varphi\wedge\psi) \;\;\vee\;\; \mathsf{sp}\,[\![C_2]\!]\,(\neg\varphi\wedge\psi)\;\;,$$

So to determine the set of reachable states starting from precondition $\psi$, we split the precondition into two disjoint ones — $\varphi\wedge\psi$ assumes that the guard is true and we execute $C_1$, whereas $\neg\varphi\wedge\psi$ assumes the guard to be false and we execute $C_2$. Thereafter, we union the so-obtained reachable sets.

Similarly for our quantitative strongest post calculi, we make use of the extended Iverson brackets and thus, the denotational strongest post of the conditional choice is:

$$\mathsf{sp}\,[\![\texttt{if}\;(\,\varphi\,)\;\{\,C_1\,\}\;\texttt{else}\;\{\,C_2\,\}]\!]\,(f) \;\;=\;\; \mathsf{sp}\,[\![C_1]\!]\,([\varphi]\curlywedge f)\curlyvee\mathsf{sp}\,[\![C_2]\!]\,([\neg\varphi]\curlywedge f)\;\;.$$

Intuitively, $\mathsf{sp}\,[\![C_1]\!]\,([\varphi]\curlywedge f)$ is the supremum of $f$ measured in all initial states before the execution of $C_1$ satisfying $\varphi$; and analogously for $\mathsf{sp}\,[\![C_2]\!]\,([\neg\varphi]\curlywedge f)$. By then taking $\curlyvee$, we finally obtain the maximum initial quantity that $f$ could have had before the execution of the conditional choice.

**Sequential Composition.**

What is the retrocipated value of $f$ before executing $C_1\,⨾\;C_2$? For this, we first retrocipate the value of $f$ before executing $C_1$ which gives $\mathsf{sp}\,[\![C_1]\!]\,(f)$. Then, we retrocipate the value $\mathsf{sp}\,[\![C_1]\!]\,(f)$ before executing $C_2$, yielding

$$\mathsf{sp}\,[\![C_1\,⨾\;C_2]\!]\,(f) = \mathsf{sp}\,[\![C_2]\!]\,(\mathsf{sp}\,[\![C_1]\!]\,(f))\;\;.$$

**Looping.**

The strongest post of a loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ is characterized using the least fixed point of the so-called $\mathsf{sp}$–*characteristic function* $\Psi_f\colon \mathbb{A} \to \mathbb{A}$. As for weakest pre, the function is chosen so that by Kleene's fixpoint theorem, the least fixed point corresponds to iterating on the least element of the lattice $-\infty$, which yields an ascending chain of loop unrollings

$$\begin{aligned}
[\neg\varphi] \curlywedge \Psi_f(-\infty) &= \mathsf{sp}\,[\![\texttt{if}(\varphi)\{\texttt{diverge}\}]\!]\,(f)\\
[\neg\varphi] \curlywedge \Psi_f^2(-\infty) &= \mathsf{sp}\,[\![\texttt{if}(\varphi)\{C\,\fatsemi\;\texttt{if}(\varphi)\{\texttt{diverge}\}\}]\!]\,(f)\\
[\neg\varphi] \curlywedge \Psi_f^3(-\infty) &= \mathsf{sp}\,[\![\texttt{if}(\varphi)\{C\,\fatsemi\;\texttt{if}(\varphi)\{C\,\fatsemi\;\texttt{if}(\varphi)\{\texttt{diverge}\}\}\}]\!]\,(f)
\end{aligned}$$

and so on, where the guard is needed to filter only those states that exit the loop; we finally obtain as strongest post

$$\mathsf{sp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C]\!]\,(f) \quad = \quad [\neg\varphi] \;\curlywedge\; \mathsf{lfp}\;\Psi_f\;.$$

After having provided an intuition, let us now state in which sense our quantitative strongest post semantics is sound:

**Theorem 3.4.1** (Characterization of $\mathsf{sp}$)**.** *For all programs $C$ and final states $\tau$,*

$$\mathsf{sp}\,[\![C]\!]\,(f)\,(\tau) \quad = \quad \bigcurlyvee_{\sigma \text{ with } \tau\in[\![C]\!](\sigma)} f(\sigma)\;.$$

Intuitively, for a given prequantity $f$ and final state $\tau$, $\mathsf{sp}[\![f]\!](\tau)$ is the *supremum* over all the values that $f$ can assume in those initial states $\sigma$ from which executing $C$ terminates in $\tau$. In case that the final state $\tau$ is *unreachable*, i.e. $\forall\sigma\colon \tau \notin [\![C]\!](\sigma)$, that supremum automatically becomes $-\infty$ — the absolute minimal value. In particular, if $\forall\,\sigma\colon f(\sigma) > -\infty$, then $\mathsf{sp}\,[\![C]\!]\,(f)\,(\tau) = -\infty$ unambiguously indicates *unreachability* of $\tau$ by executing $C$ on any input $\sigma$.

### 3.4.3 Quantitative Strongest Liberal Post

Although Dijkstra does not define strongest *liberal* postconditions, we believe that a reasonable choice for a quantitative strongest liberal post transformer is to take the *infimum* over all prequantities. Restricting to predicates, we thereby also obtain a novel *strongest liberal postcondition transformer* of type

$$\mathsf{slp}[\![C]\!] \colon \mathbb{B} \to \mathbb{B}$$

associating to each nondeterministic program $C$ a mapping from predicates to predicates. Since $\mathsf{slp}$ is associated with the infimum, we will consider a *demonic* setting, where the nondeterminism is resolved to our *dis*advantage. Whereas weakest liberal pre, in contrast to the *non*-liberal transformers, deems *nontermination* good behavior, strongest liberal post deems *unreachability* good behavior.

Specifically, the *demonic* strongest liberal postcondition transformer $\mathsf{slp}[\![C]\!]$ maps a *pre*condition $\psi$ over initial states to a *post*condition $\mathsf{slp}[\![C]\!]\,(\psi)$ over final states, such that for a given final state $\tau$ satisfying $\mathsf{slp}[\![C]\!]\,(\psi)$, all initial states that can reach $\tau$ satisfy the precondition $\psi$. More symbolically, recalling that $[\![C]\!](\sigma)$ is the set of all final states reachable after termination of $C$ on $\sigma$,

$$\tau \ \models\ \mathsf{slp}[\![C]\!]\,(\psi) \qquad \text{iff} \qquad \forall\, \sigma \text{ with } \tau \in [\![C]\!](\sigma) \colon \quad \sigma \ \models\ \psi\ ,$$

where the right-hand-side of the implication is vacuously true if $\tau$ is unreachable. From a *map perspective* on $\mathsf{slp}$, the predicate $\mathsf{slp}[\![C]\!]\,(\psi)$ is a function that takes as input a final state $\tau$, determines for each initial state $\sigma$ that can reach $\tau$, i.e., $\tau \in [\![C]\!](\sigma)$, the (truth) value $\psi(\sigma)$, takes a *conjunction* over all these truth values, and finally returns the truth value of that conjunction. More symbolically,

$$\mathsf{slp}[\![C]\!]\,(\psi)\,(\tau) \qquad = \qquad \bigwedge_{\sigma \text{ with } \tau \in [\![C]\!](\sigma)} \psi(\sigma)\ ,$$

| $\boldsymbol{C}$ | $\mathsf{slp}\,[\![C]\!]\,(f)$ |
|---|---|
| `skip` | $f$ |
| `diverge` | $+\infty$ |
| $x \coloneqq e$ | $\curlywedge\alpha\colon\ [x \neq e\,[x/\alpha]]\ \curlyvee\ f\,[x/\alpha]$ |
| $C_1 \fatsemi C_2$ | $\mathsf{slp}[\![C_2]\!]\,\big(\mathsf{slp}[\![C_1]\!]\,(f)\big)$ |
| $\{\,C_1\,\}\,\square\,\{\,C_2\,\}$ | $\mathsf{slp}[\![C_1]\!]\,(f)\ \curlywedge\ \mathsf{slp}[\![C_2]\!]\,(f)$ |
| `if` $(\,\varphi\,)\ \{\,C_1\,\}$ `else` $\{\,C_2\,\}$ | $\mathsf{slp}[\![C_1]\!]\,([\neg\varphi] \curlyvee f)\ \curlywedge\ \mathsf{slp}[\![C_2]\!]\,([\varphi] \curlyvee f)$ |
| `while` $(\,\varphi\,)\,\{\,C'\,\}$ | $[\varphi] \curlyvee \big(\mathsf{gfp}\ Y\colon f\ \curlywedge\ \mathsf{slp}[\![C']\!]\,([\neg\varphi] \curlyvee Y)\big)$ |

**Table 3.4:** Rules for the strongest liberal post transformer. Here, $\mathsf{gfp}\ f\colon \Phi(f)$ denotes the greatest fixed point of $\Phi$ and $\curlywedge\alpha\colon f(\alpha)$ denotes an infimum of $f(\alpha)$ ranging over all values of $\alpha$.

where the conjunction over an empty set is defined — as is standard — as $\mathsf{true}$. For quantities, we essentially replace $\wedge$ by $\curlywedge$ and define the following quantitative strongest liberal post transformer:

**Definition 3.4.2** (Quantitative Strongest Liberal Post)**.** *The* quantitative strongest liberal post transformer

$$\mathsf{slp}\colon\quad \mathsf{nGCL} \to (\mathbb{A} \to \mathbb{A})$$

*is defined inductively according to the rules in* Table 3.4 *(right column). We call the function*

$$\Psi_f(X) \;=\; f\ \curlywedge\ \mathsf{sp}\,[\![C]\!]\,([\neg\varphi] \curlyvee X)\ ,$$

*whose* greatest *fixed point is used to define* $\mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)$*, the* $\mathsf{slp}$*–characteristic function of* `while` $(\,\varphi\,)\,\{\,C\,\}$ *with respect to* $f$*.* $\triangleleft$

Let us thus go over the language constructs where the rules for $\mathsf{slp}$ and $\mathsf{sp}$ differ and explain both strongest liberal postconditions and quantitative strongest liberal post.

**Assignment.**

The strongest liberal post*condition* of an assignment is given by

$$\mathsf{slp}[\![x \coloneqq e]\!]\,(\psi) \qquad = \qquad \forall\, \alpha\colon \quad \underbrace{x \neq e\,[x/\alpha]}_{(1)} \;\vee\; \underbrace{\psi\,[x/\alpha]}_{(2)}\;.$$

Intuitively, the quantified $\alpha$ represents *candidates for initial values* of $x$ *before* executing the assignment. For each such candidate $\alpha$, it must be true that

1. $\alpha$ is in fact *not* a valid initial value for $x$, i.e. $x$ does *not* have in the final state the value of expression $e$ evaluated using the candidate value $\alpha$ for $x$, or

2. $\alpha$ *is* valid and the precondition $\psi$ was true in the initial state where $x$ had value $\alpha$.

Intuitively, (1) captures that strongest liberal postconditions deem unreachability good behavior, because if some state is not reachable by executing $x \coloneqq e$, then $x \neq e\,[x/\alpha]$ is true for all $\alpha$ and hence the strongest liberal post evaluates to $\mathsf{true}$.

For quantities, dually to the strongest *non-liberal* post, we now substitute the *universal* quantifier with an *infimum* (denoted by the $\text{\Large ʖ}$ "quantifier" [102]) and the $\vee$ with a $\curlyvee$, thus obtaining

$$\mathsf{slp}[\![x \coloneqq e]\!]\,(f) \qquad = \qquad \text{ʖ}\,\alpha\colon \quad [x \neq e\,[x/\alpha]] \;\;\curlyvee\;\; f\,[x/\alpha]$$

Let us again consider a few examples. First, one can convince oneself that

$$\mathsf{slp}[\![x \coloneqq x+1]\!]\,(x) \;\;=\;\; x-1 \;\;=\;\; \mathsf{sp}\,[\![x \coloneqq x+1]\!]\,(x) \;\;.$$

$\mathsf{slp} = \mathsf{sp}$ is not surprising in this case, because *every* state $\tau(x) = \beta$ is reachable by executing $x \coloneqq x+1$, namely by starting from initial state $\sigma(x) = \beta - 1$. As another example, consider

$$\begin{aligned}\mathsf{slp}[\![x \coloneqq 10]\!]\,(x) \quad &= \quad \text{\Large$\ell$}\,\alpha\colon\; [x \neq 10] \;\curlyvee\; \alpha \\ &= \quad [x \neq 10] \;\curlyvee\; \infty \\ &= \quad [x \neq 10]\ .\end{aligned}$$

For the final state $\tau(x) = 10$, this gives us $[10 \neq 10] = [\mathsf{false}] = -\infty$ which is indeed the *greatest lower bound* (demonic!) on the initial value of $x$ if the final state after executing $x \coloneqq 10$ is $\tau$. In other words: by evaluating $[x \neq 10]$ in $\tau$, we know that $\tau$ was reachable, but we have no information on what minimal value $x$ could have had initially, which is sensible because $x \coloneqq 10$ forgets any initial value of $x$. For final state $\tau'(x) = 9$, on the other hand, we get $[9 \neq 10] = [\mathsf{true}] = +\infty$ which is the *value of unreachability* in $\mathsf{slp}$ (cf. also the next paragraph on divergence). Indeed, the final state after executing $x \coloneqq 10$ cannot ever be $\tau'$.

**Diverge.**

Since `diverge` is certainly nonterminating, i.e. it reaches no final state, and since liberal post deems nonreachability good behavior, the quantitative strongest liberal post assigns the greatest element, i.e.

$$\mathsf{slp}[\![\texttt{diverge}]\!]\,(f) = +\infty\ .$$

**Remark 3.4.2** (Quantitative Strongest Liberal Post and Unreachability)**.** *Analogously to $-\infty$ being the value of unreachability in* $\mathsf{sp}$ *(cf. Remark 3.4.1), $+\infty$ is* the value of unreachability *in* $\mathsf{slp}$. $\triangleleft$

**Nondeterministic Choice.**

The *demonic* strongest liberal postcondition of $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$ is

$$\mathsf{slp}[\![\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}]\!]\,(\psi) \quad = \quad \mathsf{slp}[\![C_1]\!]\,(\psi) \;\wedge\; \mathsf{slp}[\![C_2]\!]\,(\psi)\ .$$

Indeed, $\mathsf{slp}[\![C_i]\!]\,(\psi)$ contains all final states $\tau$ such that all initial states $\sigma$ that can reach $\tau$ by executing $C_i$ satisfy $\psi$. By intersecting $\mathsf{sp}\,[\![C_1]\!]\,(\psi)$ and $\mathsf{sp}\,[\![C_2]\!]\,(\psi)$ we ensure the *stronger* requirement that all initial states $\sigma$ that can reach $\tau$ by executing $C_1$ *or* $C_2$ satisfy $\psi$.

In a quantitative setting, where we want to retrocipate the value of a quantity $f$ before executing $C_1$ or $C_2$, we *demonically* minimize the possible initial value and hence take as strongest post

$$\mathsf{slp}[\![\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}]\!]\,(f) \;=\; \mathsf{slp}[\![C_1]\!]\,(f) \;\curlywedge\; \mathsf{slp}[\![C_2]\!]\,(f) \enspace .$$

**Conditional Choice**

The *demonic* strongest liberal postcondition of $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$ is given by

$$\mathsf{slp}[\![\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}]\!]\,(\psi) \;=\; \mathsf{slp}[\![C_1]\!]\,(\neg\varphi \vee \psi) \;\wedge\; \mathsf{slp}[\![C_2]\!]\,(\varphi \vee \psi) \enspace ,$$

Indeed, since the disjunction can be seen as an implication, $\mathsf{slp}[\![C_1]\!]\,(\neg\varphi \vee \psi)$ contains all final states $\tau$ such that, all initial states that satisfy $\varphi$ (sic!) and that can reach $\tau$ by executing $C_1$ do also satisfy $\psi$. Similarly, $\mathsf{slp}[\![C_2]\!]\,(\varphi \vee \psi)$ contains all final states $\tau$ such that, all initial states that satisfy $\neg\varphi$ (sic!) and that can reach $\tau$ by executing $C_2$ do also satisfy $\psi$. By intersecting the postconditions $\mathsf{slp}[\![C_1]\!]\,(\neg\varphi \vee \psi)$ and $\mathsf{slp}[\![C_2]\!]\,(\varphi \vee \psi)$, we obtain exactly all those final states $\tau$ such that, all initial states that, *either* satisfy $\varphi$ and can reach $\tau$ by executing $C_1$, or satisfy $\neg\varphi$ and can reach $\tau$ by executing $C_2$ do also satisfy the precondition $\psi$.

Similarly for our quantitative strongest post calculi, we make use of the *extended Iverson brackets* and thus, the quantitative strongest liberal post of the conditional choice is

$$\mathsf{slp}[\![\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}]\!]\,(f) \;=\; \mathsf{slp}[\![C_1]\!]\,([\neg\varphi] \curlyvee f) \curlywedge \mathsf{slp}[\![C_2]\!]\,([\varphi] \curlyvee f) \enspace .$$

Intuitively, $\mathsf{slp}[\![C_1]\!]\,([\neg\varphi] \curlyvee f)$ characterizes the infimum of $f$ measured in all initial states before the execution of $C_1$ satisfying $\varphi$; and analogously for $\mathsf{slp}[\![C_2]\!]\,([\varphi] \curlyvee f)$. By taking $\curlywedge$, we obtain exactly the minimum initial quantity that $f$ could have had before executing the conditional choice.

**Looping**

For a loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$, $\mathsf{slp}$ is characterized using the greatest fixed point of the so-called $\mathsf{slp}$–*characteristic function* $\Psi_f \colon \mathbb{A} \to \mathbb{A}$. As for weakest liberal pre, the function is chosen so that by Kleene's fixpoint theorem, the greatest fixed point corresponds to iterating on the top element of the lattice $+\infty$, which yields a descending chain of loop unrollings

$$\begin{aligned}
[\varphi] \curlyvee \Psi_f(+\infty) &= \mathsf{slp}[\![\texttt{if}(\varphi)\{\texttt{diverge}\}]\!]\,(f)\\
[\varphi] \curlyvee \Psi_f^2(+\infty) &= \mathsf{slp}[\![\texttt{if}(\varphi)\{C\,;\ \texttt{if}(\varphi)\{\texttt{diverge}\}\}]\!]\,(f)\\
[\varphi] \curlyvee \Psi_f^3(+\infty) &= \mathsf{slp}[\![\texttt{if}(\varphi)\{C\,;\ \texttt{if}(\varphi)\{C\,;\ \texttt{if}(\varphi)\{\texttt{diverge}\}\}\}]\!]\,(f)
\end{aligned}$$

and so on. Since our strongest liberal postcondition considers unreachability as "good behavior", we join the Kleene's iterates with all the final states where the guard still hold and obtain as strongest liberal post:

$$\mathsf{slp}[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \quad = \quad [\varphi] \ \curlyvee\ \mathsf{gfp}\ \Psi_f\ .$$

After having provided an intuition, let us now state soundness of our strongest liberal post semantics:

**Theorem 3.4.2** (Characterization of $\mathsf{slp}$). *For all programs $C$ and states $\tau \in \Sigma$,*

$$\mathsf{slp}[\![C]\!]\,(f)\,(\tau) \quad = \quad \bigcurlywedge_{\sigma \text{ with } \tau \in [\![C]\!]\sigma} f(\sigma)$$

Intuitively, for a given prequantity $f$ and final state $\tau$, the $\mathsf{slp}[\![C]\!]\,(f)\,(\tau)$ is the *infimum* over all values that $f$ can assume measured in the initial states $\sigma$, so that executing $C$ on $\sigma$ terminates in $\tau$. In case that the final state $\tau$ is *unreachable*, i.e. $\forall \sigma \colon \tau \notin [\![C]\!](\sigma)$, that infimum becomes $+\infty$ — the absolute maximum value. In particular, if $\forall\, \sigma\colon f(\sigma) < +\infty$, then $\mathsf{sp}\,[\![C]\!]\,(f)\,(\tau) = +\infty$ unambiguously indicates *unreachability* of $\tau$ by executing $C$ on any input $\sigma$.

## 3.5 Properties of Quantitative Transformers

In reasoning about quantitative program properties, certain so-called "healthiness conditions" emerge as key properties that ensure the validity and robustness of the underlying mathematical frameworks. These conditions—continuity, strictness, monotonicity, and others—guarantee that transformers maintain well-defined behaviors, such as fixed-point existence for loops. In this section, we formalize these healthiness properties for our quantitative transformers in nondeterministic programming, some of which are analogous to Dijkstra's, Kozen's, or McIver & Morgan's calculi. We furthermore present several dualities between our transformers and how to embed classical into quantitative reasoning.

### 3.5.1 Healthiness Properties

We begin by stating our formal healthiness theorem, followed by a detailed examination of its specific properties.

**Theorem 3.5.1** (Healthiness Properties of Quantitative Transformers)**.** *For all programs $C$, all quantitive transformers are monotonic, i.e.*

$$f ~\preceq~ g \quad \text{implies} \quad \mathsf{ttt}\,[\![C]\!]\,(f) ~\preceq~ \mathsf{ttt}\,[\![C]\!]\,(g)~, \qquad \text{for } \mathsf{ttt} \in \{\mathsf{wp}, \mathsf{wlp}, \mathsf{sp}, \mathsf{slp}\}~.$$

*The non-liberal transformers $\mathsf{wp}[\![C]\!]$ and $\mathsf{sp}[\![C]\!]$ satisfy the following properties:*

1. *Quantitative universal disjunctiveness: For any set of quantities $S \subseteq \mathbb{A}$,*

$$\mathsf{wp}\,[\![C]\!]\,(\curlyvee S) ~=~ \curlyvee~\mathsf{wp}\,[\![C]\!]\,(S) \quad \text{and} \quad \mathsf{sp}\,[\![C]\!]\,(\curlyvee S) ~=~ \curlyvee~\mathsf{sp}\,[\![C]\!]\,(S)~.$$

2. *Strictness:* $\quad \mathsf{wp}\,[\![C]\!]\,(-\infty) ~=~ -\infty \quad \text{and} \quad \mathsf{sp}\,[\![C]\!]\,(-\infty) ~=~ -\infty~.$

*The liberal transformers $\mathsf{wlp}[\![C]\!]$ and $\mathsf{slp}[\![C]\!]$ satisfy the following properties:*

3. *Quantitative universal conjunctiveness:* *For any set of quantities $S \subseteq \mathbb{A}$,*

$$\mathsf{wlp}[\![C]\!]\,(\curlywedge S) \;=\; \curlywedge\, \mathsf{wlp}[\![C]\!]\,(S) \quad \text{and} \quad \mathsf{slp}[\![C]\!]\,(\curlywedge S) \;=\; \curlywedge\, \mathsf{slp}[\![C]\!]\,(S)\ .$$

4. *Costrictness:* $\mathsf{wlp}[\![C]\!]\,(+\infty) \;=\; +\infty \quad \text{and} \quad \mathsf{slp}[\![C]\!]\,(+\infty) \;=\; +\infty\ .$

## Monotonicity

A crucial healthiness property for quantitative transformers is monotonicity.

**Definition 3.5.1** (Monotonicity)**.** *A transformer $T\colon \mathbb{A} \to \mathbb{A}$ is monotonic if, for any two functions $f, g \in \mathbb{A}$, we have:*

$$f \preceq g \implies T(f) \preceq T(g)\ .$$

$\lhd$

Monotonicity guarantees that if one function is pointwise less than or equal to another, the transformer will preserve this order in their respective images. This property can be utilized, for instance, to derive the consequence rule of various logics, such as Hoare Logic and Incorrectness Logic.
All our transformers are monotonic, see Theorem 3.5.1.

## Continuity, Disjunctiveness and Conjunctiveness

Continuity plays a crucial role in ensuring that our quantitative transformers can handle infinite sequences or chains of computations smoothly. Formally, we define continuity as follows.

**Definition 3.5.2** (Continuity and Co-continuity)**.** *A transformer $T\colon \mathbb{A} \to \mathbb{A}$ is:*

- continuous *if it commutes with the suprema of ascending chains, i.e., for any countable chain of functions $f_1 \preceq f_2 \preceq \ldots$ in $\mathbb{A}$, continuity guarantees that* *the transformer preserves their least upper bound:*

$$T\left(\curlyvee_i f_i\right) = \curlyvee_i T(f_i).$$

- co-continuous *if it commutes with the infima of descending chains, i.e., for any countable descending chain of functions $f_1 \succeq f_2 \succeq \ldots$ in $\mathbb{A}$, co-continuity guarantees that the transformer preserves their greatest lower bound:*

$$T\left(\curlywedge_i f_i\right) = \curlywedge_i T(f_i).$$

$\triangleleft$

This property is particularly significant for reasoning about loops and recursive constructs. Programs with loops can often be modeled as limits of successive approximations, where each step corresponds to an iteration of the loop. Continuity ensures that the limit of these iterations exists and converges exactly to particular fixed points. In fact, (co)continuity is related to the Kleene Fixed Point Theorem 2.4.3, which guarantees the existence of a least (greatest) fixed point for continuous functions on a complete lattice, thereby making our transformers well-defined. Monotonicity is a consequence of continuity and co-continuity, and it also ensures the existence of fixed points. However, fixed point iteration may only stabilize at ordinals higher than $\omega$ for non-(co)continuous functions [92].

In Theorem 3.5.1 (3, 1) we show that not only our transformers are (co)continuous, but they preserve arbitrary suprema (infima). More specifically, this property is sometimes referred as *quantitative universal disjunctiveness (conjunctiveness)*, and are quantitative analogues to Dijkstra and Scholten's original calculi, whereas conjunctiveness of $\mathsf{slp}$ is novel (since $\mathsf{slp}$ is novel) and fits well into this picture of duality. Note that quantitative universal disjunctiveness (conjunctiveness) implies (co)continuity.

**Strictness**

In the context of classical $\mathsf{wp}$, strictness (also known as the "Law of the Excluded Miracle" [63]) ensures that no initial state can terminate in a state satisfying "false". Quantitative generalisations of strictness [16, Definition 4.13], defined as $\mathsf{wp}\,[\![C]\!]\,(0) = 0$, mean that the expected value of the constantly 0 random variable after executing a program $C$ is 0.

Formally, in our context employing mixed real values, quantitative strictness and costrictness are defined as follows:

**Definition 3.5.3** (Strictness and Costrictness)**.** *Let* $T\colon \mathbb{A} \to \mathbb{A}$ *be a quantitative transformer. Then:*

1. *$T$ is called* strict*, if* $T(-\infty) = -\infty$.

2. *$T$ is called* costrict*, if* $T(+\infty) = +\infty$.

$\lhd$

Analogously to Dijkstra's predicate transformers, liberal transformers are costrict and their nonliberal versions are strict, see Theorem 3.5.1 (2, 4). Strictness of $\mathsf{wp}$, i.e. $\mathsf{wp}\,[\![C]\!]\,(-\infty) = -\infty$, states that the anticipated value of $-\infty$ after executing $C$ is $-\infty$ if the program terminates, and otherwise yields $\mathsf{wp}$'s value of nontermination: $-\infty$. Strictness of $\mathsf{sp}$, i.e. $\mathsf{sp}\,[\![C]\!]\,(-\infty) = -\infty$, indicates that $-\infty$ retrocipates the value of $-\infty$ if the final state is reachable, and otherwise yields $\mathsf{sp}$'s value of unreachability: $-\infty$. Explanations for costrictness are analogous.

The predicate interpretation of (co)strictness is also preserved: Since $-\infty = [\mathsf{false}]$ and $+\infty = [\mathsf{true}]$ and hence $\mathsf{wp}\,[\![C]\!]\,([\mathsf{false}]) = [\mathsf{false}]$ and $\mathsf{wlp}[\![C]\!]\,([\mathsf{true}]) = [\mathsf{true}]$, strictness of quantitative $\mathsf{wp}[\![C]\!]$ means that $C$ cannot terminate in some $\tau \in \emptyset$; strictness of $\mathsf{sp}[\![C]\!]$ that no $\tau$ is reachable by executing $C$ on any $\sigma \in \emptyset$; costrictness of $\mathsf{wlp}[\![C]\!]$ that on all states $C$ either terminates or not; and costrictness of $\mathsf{slp}[\![C]\!]$ (novelly) that all states are either reachable by executing $C$ or unreachable.

**Linearity**

Another fundamental property of quantitative transformers is linearity, which is fundamental for compositional reasoning. Sub- and superlinearity have been studied by Kozen, McIver & Morgan, and Kaminski for probabilistic $\mathsf{w(l)p}$ transformers. Our transformers also obey to analogous rules.

**Theorem 3.5.2** (Linearity)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ and $\mathsf{sp}[\![C]\!]$ are sublinear, and $\mathsf{wlp}[\![C]\!]$ and $\mathsf{slp}[\![C]\!]$ are superlinear, i.e. for all $f, g \in \mathbb{A}$ and* non-negative *constants $r \in \mathbb{R}_{\geq 0}$,*

$$\begin{aligned}
\mathsf{wp}\,[\![C]\!]\,(r \cdot f + g) &\preceq r \cdot \mathsf{wp}\,[\![C]\!]\,(f) + \mathsf{wp}\,[\![C]\!]\,(g) \enspace, \\
\mathsf{sp}\,[\![C]\!]\,(r \cdot f + g) &\preceq r \cdot \mathsf{sp}\,[\![C]\!]\,(f) + \mathsf{sp}\,[\![C]\!]\,(g) \enspace, \\
r \cdot \mathsf{wlp}[\![C]\!]\,(f) + \mathsf{wlp}[\![C]\!]\,(g) &\preceq \mathsf{wlp}[\![C]\!]\,(r \cdot f + g) \enspace, \quad \text{and} \\
r \cdot \mathsf{slp}[\![C]\!]\,(f) + \mathsf{slp}[\![C]\!]\,(g) &\preceq \mathsf{slp}[\![C]\!]\,(r \cdot f + g) \enspace.
\end{aligned}$$

### 3.5.2 Relationship between Qualitative and Quantitative Transformers

Our calculi subsume both the classical ones of Dijkstra and Scholten [64] and our definition of strongest liberal postcondition for predicates by means of our extended Iverson brackets:

**Theorem 3.5.3** (Embedding Classical into Quantitative Transformers)**.** *For all $\underline{\text{deterministic}}$ programs $C$ and predicates $\psi$, we have*

$$\mathsf{wp}\,[\![C]\!]\,([\psi]) = [\mathsf{wp}\,[\![C]\!]\,(\psi)] \qquad \text{and} \qquad \mathsf{wlp}[\![C]\!]\,([\psi]) = [\mathsf{wlp}\,[\![C]\!]\,(\psi)] \enspace,$$

*and for $\underline{\text{all}}$ programs $C$ and predicates $\psi$, we have*

$$\mathsf{sp}\,[\![C]\!]\,([\psi]) = [\mathsf{sp}\,[\![C]\!]\,(\psi)] \qquad \text{and} \qquad \mathsf{slp}[\![C]\!]\,([\psi]) = [\mathsf{slp}[\![C]\!]\,(\psi)] \enspace.$$

From a predicate perspective, $\mathsf{sp}\,[\![C]\!]\,(\psi)$ contains final states $\tau$ that are reachable from at least one initial state satisfying $\psi$, whereas $\mathsf{slp}[\![C]\!]\,(\psi)$ requires that every initial state that may end in $\tau$ satisfies $\psi$. Hence, we have a fundamentally dual meaning of the word *liberal*:

- $\mathsf{wlp}$, differently from $\mathsf{wp}$, provides *preconditions* containing all *diverging* initial states, but contains no state that can terminate outside the postcondition.
- $\mathsf{slp}$, differently from $\mathsf{sp}$, provides *postconditions* containing all *unreachable* final states, but contains no state that can be reached from outside the precondition.

Let us also consider two other examples: $\mathsf{sp}\,[\![C]\!]\,([\mathsf{true}])$ is the indicator function of the reachable states. If $\mathsf{sp}\,[\![C]\!]\,([\mathsf{true}]) = [\mathsf{false}]$ (i.e. $\mathsf{sp}\,[\![C]\!]\,(+\infty) = -\infty$), no state is reachable and hence $C$ diverges on every input. Similarly, $\mathsf{slp}[\![C]\!]\,([\mathsf{false}])$ is the indicator function of all states that are either reachable from an initial state satisfying $\mathsf{false}$ (of which there are none) or which are unreachable. Thus, if $\mathsf{slp}[\![C]\!]\,([\mathsf{false}]) = [\mathsf{true}]$ (i.e. $\mathsf{slp}[\![C]\!]\,(-\infty) = +\infty$) then all states are unreachable, meaning $C$ diverges on every input. Put shortly,

$$\mathsf{sp}\,[\![C]\!]\,(+\infty) \;=\; -\infty \quad \text{iff} \quad \mathsf{slp}[\![C]\!]\,(-\infty) \;=\; +\infty\ .$$

Finally, we note that the *quantitative weakest pre calculi* of Kaminski [16, Section 2.3], restricted to *deterministic non-probabilistic programs* are even simply subsumed by the fact that we consider a larger lattice, namely quantities of type $f\colon \Sigma \to \mathbb{R}^{\pm\infty}$ instead of $f\colon \Sigma \to \mathbb{R}^{\infty}_{\geq 0}$.

### 3.5.3 Relationship between Liberal and Non-liberal Transformers

**Theorem 3.5.4** (Liberal–Non-liberal Duality). *For any program $C$ and quantity $f$, we have*

$$\mathsf{wp}\,[\![C]\!]\,(f) \;=\; -\,\mathsf{wlp}[\![C]\!]\,(-f) \qquad \text{and} \qquad \mathsf{sp}\,[\![C]\!]\,(f) \;=\; -\,\mathsf{slp}[\![C]\!]\,(-f)\ .$$

The duality for weakest pre is very similar to $\mathsf{wp}\,[\![C]\!]\,(\psi) = \neg\mathsf{wlp}[\![C]\!]\,(\neg\psi)$ in Dijkstra's classical calculus and $\mathsf{wp}\,[\![C]\!]\,(f) = 1 - \mathsf{wlp}[\![C]\!]\,(1-f)$ for 1-bounded functions $f$ in Kozen's and McIver & Morgans development for probabilistic programs.

When considering only *deterministic* programs $C$ (i.e. *syntactically* without nondeterministic choices), then executing $C$ on initial state $\sigma$ will either terminate in a *single* final state (i.e. $[\![C]\!](\sigma) = \{\tau\}$, for some $\tau$), or diverge (i.e. $[\![C]\!](\sigma) = \emptyset$), meaning that $[\![C]\!](\_\_)$ becomes a proper (partial) function. Hence, in case of termination, supremum and infimum of the final values of $f$ coincide:

**Corollary 3.5.4.1.** *If a deterministic program $C$ terminates on an input $\sigma$, then for all quantities $f$,*

$$\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \;=\; \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)\ ,$$

*and otherwise*

$$\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \;=\; -\infty \quad \text{and} \quad \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma) \;=\; -\infty\ .$$

*Proof.* Direct consequence of Theorems 3.3.1 and 3.3.2. □

As a direct consequence of Corollary 3.5.4.1, for postquantities everywhere smaller than $+\infty$ (which is not restrictive since values of program variables are finite), we can precisely detect whether a given initial state has terminated or not. Kaminski [16, Remark 2.12], in contrast, cannot easily distinguish whether a certain initial state does not terminate, or whether the anticipated value is 0.

Note that dual results for $\mathsf{sp}$ and $\mathsf{slp}$ do *not* hold since even for deterministic programs the fiber of the concrete semantics is not a function: multiple initial states can terminate in a single final state $\tau$.

## 3.6 Correctness and Incorrectness Reasoning

### 3.6.1 Galois Connections between Weakest Pre and Strongest Post

The classical strongest postcondition is the left adjoint to the weakest liberal precondition [64, Section 12], i.e. the transformers $\mathsf{wlp}$ and $\mathsf{sp}$ form the Galois connection

$$G \implies \mathsf{wlp}[\![C]\!]\,(F) \qquad \text{iff} \qquad \mathsf{sp}\,[\![C]\!]\,(G) \implies F\,, \tag{$\dagger$}$$

which intuitively is true because $G \implies \mathsf{wlp}\,[\![C]\!]\,(F)$ means that starting from $G$ the program $C$ will either diverge or terminate in a state satisfying $F$, and $\mathsf{sp}\,[\![C]\!]\,(G) \implies F$ means that starting from $G$ any state reachable by executing $C$ satisfies $F$.

The above Galois connection is preserved in our quantitative setting; in fact, by substituting the partial order $\implies$ on predicates with the partial order $\preceq$ on $\mathbb{A}$ we obtain:

**Theorem 3.6.1** (Galois Connection between $\mathsf{wlp}$ and $\mathsf{sp}$). *For all $C \in \mathsf{nGCL}$ and $g, f \in \mathbb{A}$:*

$$g \preceq \mathsf{wlp}[\![C]\!]\,(f) \qquad \text{iff} \qquad \mathsf{sp}\,[\![C]\!]\,(g) \preceq f\,.$$

As $\mathsf{wlp}$ is for partial correctness, Theorem 3.6.1 shows that $\mathsf{sp}$ is also suitable for partial correctness. One may now wonder whether there exists a strongest post transformer that is tightly related to $\mathsf{wp}$, and hence, to total correctness. Unfortunately, Dijkstra and Scholten [64, Section 12] show that there cannot exist a predicate transformer $\mathsf{stp}$ — a *"strongest total postcondition"* — such that

$$G \implies \mathsf{wp}\,[\![C]\!]\,(F) \qquad \text{iff} \qquad \mathsf{stp}\,[\![C]\!]\,(G) \implies F\,.$$

Categorically, that negative result is a consequence of the fact that we are requiring $\mathsf{wp}$ to be a *right adjoint functor*, and a necessary condition for that is to preserve all infima, but this is not true since $\mathsf{wp}$ is not costrict. Despite this negative result, since $\mathsf{wp}$ preserves all suprema (cf. Theorem 3.5.1 (1)), we argue that $\mathsf{wp}$ is instead a *left adjoint functor* and show that its right adjoint is exactly $\mathsf{slp}$:

**Theorem 3.6.2** (Galois Connection between $\mathsf{wp}$ and $\mathsf{slp}$)**.** *For all $C \in \mathsf{nGCL}$ and $g, f \in \mathbb{A}$:*

$$\mathsf{wp}\,[\![C]\!]\,(f) \;\preceq\; g \qquad \text{iff} \qquad f \;\preceq\; \mathsf{slp}[\![C]\!]\,(g)$$

Let us provide an intuition on this connection, for simplicity only with "predicates" $[F]$ and $[G]$: $[F] \preceq \mathsf{slp}[\![C]\!]\,([G])$ means that every final state satisfying $F$ is either reached only by states satisfying $G$ or unreachable. This is equivalent to saying that all initial states terminating in $F$ must satisfy $G$, which is precisely expressed by $\mathsf{wp}\,[\![C]\!]\,([F]) \preceq [G]$.

### 3.6.2 Resolving Nondeterministic Choice: Angelic vs. Demonic

Our choices of how to resolve nondeterminism are motivated by establishing dualities between weakest pre and strongest post presented in Section 3.6.1. The only thing we take for granted is that the standard definition of $\mathsf{sp}$ is angelic, thus characterizing the "set of reachable states". Indeed, if $\mathsf{sp}$ is angelic, then we are (provably) also forced to make $\mathsf{wp}$ angelic, and both $\mathsf{wlp}$ and $\mathsf{slp}$ demonic – otherwise, duality would break. We can also come up with an intuition for these choices: Both, angelic $\mathsf{wp}$ and demonic $\mathsf{wlp}$ transformers try to *avoid nontermination*, if at all possible, whereas angelic $\mathsf{sp}$ and demonic $\mathsf{slp}$ try to *avoid unreachability.*

By dualizing all resolutions of nondeterminism one would obtain the following intuition: Demonic $\mathsf{wp}$ and angelic $\mathsf{wlp}$ transformers try to *drive the execution towards nontermination* (more standard for both $\mathsf{wp}$ and $\mathsf{wlp}$), whereas demonic $\mathsf{sp}$ and angelic $\mathsf{slp}$ try to *establish unreachability* (less standard for $\mathsf{sp}$, whereas $\mathsf{slp}$ is novel anyway). We leave it as future work to study whether this dual situation would also preserve the Galois connections of Section 3.6.1.

### 3.6.3 Strongest Post and Incorrectness Logic

A Hoare triple $\models \{G\}\ C\ \{F\}$ is *valid for partial correctness* iff $G \implies \mathsf{wlp}[\![C]\!]\,(F)$ or (equivalently, see (†) in Section 3.6.1) $\mathsf{sp}\,[\![C]\!]\,(G) \implies F$ holds. Somewhat recently, a different kind of triples have been proposed, first by de Vries and Koutavas [61] under the name *reverse Hoare logic* for studying reachability specifications. A few years ago, O'Hearn [6] rediscovered those triples under the name *incorrectness logic* and used them for explicit error handling. Bruni et al. [71] provide a logic parametrized by an abstract interpretation that, through a notion of local completeness, can prove both correctness and incorrectness.

In this section we show, first, the relationship between our strongest post transformer and incorrectness triples [61; 6]; then, more importantly, we argue that such triples deal with *total incorrectness* and hint at novel *partial incorrectness* triples.

#### (Total) Incorrectness Logic

In the sense of de Vries and Koutavas [61], an *incorrectness triple*

$$[\,G\,]\ C\ [\,F\,] \text{ is valid} \qquad \text{iff} \qquad \forall \tau \models F \quad \exists \sigma \text{ with } \tau \in [\![C]\!](\sigma){:} \quad \sigma \models G\ .$$

In other words, the set of states $F$ is an underapproximation of the set of states reachable by executing $C$ on some state in $G$, i.e., $F \subseteq \mathsf{sp}\,[\![C]\!]\,(G)$ [6, Definition 1]. The term *incorrectness logic* originates from the fact that if $[\,G\,]\ C\ [\,F\,]$ is valid and $F$ contains an error state, then this *error state is guaranteed to be reachable from $G$*. Since our quantitative strongest post transformer subsumes the classical one, we can (re)define incorrectness triples by substituting predicates with extended Iverson brackets and obtain the following equivalent definition:

**Definition 3.6.1** (Incorrectness Triples)**.** *For predicates $G, F$ and program $C$, the* incorrectness triple

$$[\,G\,]\ C\ [\,F\,] \text{ is } \textit{valid for (total) incorrectness} \qquad \text{iff} \qquad [F] \ \preceq\ \mathsf{sp}\,[\![C]\!]\,([G])\ . \qquad \triangleleft$$

**Partial Incorrectness**

We argue that the aforementioned triples deal with *total incorrectness* by providing novel triples for *partial incorrectness.* Recall that a Hoare triple $\models \{\,G\,\}\ C\ \{\,F\,\}$ is valid for total correctness if $G \implies \mathsf{wp}\,[\![C]\!]\,(F)$. By replacing $\mathsf{wp}$ with $\mathsf{wlp}$, we can define partial correctness triples: $\models \{\,G\,\}\ C\ \{\,F\,\}$ is valid for partial correctness if $G \implies \mathsf{wlp}[\![C]\!]\,(F)$. By mimicking the above, we define *partial incorrectness* by replacing $\mathsf{sp}$ with $\mathsf{slp}$ in Definition 3.6.1:

**Definition 3.6.2** (Partial Incorrectness)**.** *For predicates $G, F$ and program $C$, the* incorrectness triple

$$[\,G\,]\ C\ [\,F\,] \text{ is } \textit{valid for partial incorrectness} \qquad \text{iff} \qquad [F] \ \preceq\ \mathsf{slp}[\![C]\!]\,([G])\ . \qquad \triangleleft$$

By definition of $\mathsf{slp}$,

$$\begin{gathered}[\,G\,]\ C\ [\,F\,] \text{ is valid for partial incorrectness} \\ \text{iff} \\ \forall\,\tau \models F \quad \forall\,\sigma \text{ with } \tau \in [\![C]\!](\sigma)\colon \quad \sigma\ \models\ G\ .\end{gathered}$$

In other words, only if the state $\tau$ is *reachable*, then the triple guarantees that $\tau$ is reached only from initial states $\sigma$ that satisfy $G$. Note that this is dual to the relationship between total and partial correctness: with partial incorrectness, to have *full* information on *initial states* we require an additional *proof of reachability* on *final states* (whereas with partial correctness, to obtain full information on *final states* we require an additional *proof of termination* on *initial states*).

We also note that, due to the Galois between wp and slp (Theorem 3.6.2) we have

$$[\,G\,]\ C\ [\,F\,]\ \text{ is valid for } \textit{partial incorrectness} \qquad \text{iff} \qquad \mathsf{wp}\,[\![C]\!]\,([F]) \;\preceq\; [G]\ .$$

This implies that $G$ is an overapproximation of the set of states that end up in $F$, and corresponds to the notion of *necessary preconditions* studied by Cousot et al. [105]. In particular, if an initial state $\sigma \not\models G$, then $\sigma$ is guaranteed to not terminate in $F$ ($\sigma$ could also diverge).

**Other Triples**

| implication | | | defines |
|---:|:---:|:---|:---|
| $G$ | $\Longrightarrow$ | $\mathsf{wp}\,[\![C]\!]\,(F)$ | total correctness |
| $G$ | $\Longrightarrow$ | $\mathsf{wlp}[\![C]\!]\,(F)$ | partial correctness |
| $\mathsf{wp}\,[\![C]\!]\,(F)$ | $\Longrightarrow$ | $G$ | partial incorrectness |
| $\mathsf{wlp}[\![C]\!]\,(F)$ | $\Longrightarrow$ | $G$ | ??? |
| $F$ | $\Longrightarrow$ | $\mathsf{sp}\,[\![C]\!]\,(G)$ | (total) incorrectness |
| $F$ | $\Longrightarrow$ | $\mathsf{slp}[\![C]\!]\,(G)$ | partial incorrectness |
| $\mathsf{sp}\,[\![C]\!]\,(G)$ | $\Longrightarrow$ | $F$ | partial correctness |
| $\mathsf{slp}[\![C]\!]\,(G)$ | $\Longrightarrow$ | $F$ | ¿¿¿ |

We note that the naming conventions *correctness* and *incorrectness* may not necessarily always be appropriate. First of all, we argue that incorrectness triples [61; 6] can be used to prove *good behavior*: for instance, a triple $[\,G\,]\ C\ [\,F\,]$ where $F$ contains good states, ensures that every (*good*) state in $F$ is reachable from precondition $G$. Rather than *correctness* versus *incorrectness*, we believe that the fundamental difference between the triples is that correctness triples provide information on the behavior of *initial* states satisfying *preconditions*, whereas incorrectness triples guarantee reachability properties on *final* states satisfying *postconditions*.

Secondly, note that our transformers can define two additional triples other than total (partial) correctness (incorrectness), for which the current naming conventions are insufficient. So far, we have the picture depicted in the table above. The two blue and the two orange lines define the same notion due to the Galois connections between wlp/sp and wp/slp. For ??? and ¿¿¿, however, there are no appropriate names (let alone program logics) yet. We can say, however, that ??? gives rise to a notion of *necessary liberal preconditions*, in the sense that (1) $G$ contains all initial states $\sigma$ that *diverge*, and (2) whenever $\sigma \not\models G$, then $\sigma$ is guaranteed to *terminate* in a state $\tau \not\models F$. ¿¿¿, on the other hand, provides *necessary liberal postconditions*, meaning that (1) $F$ contains all *unreachable* states, and every final state $\tau \not\models F$ is guaranteed to be *reachable* from some initial state $\sigma \not\models G$.

Following the terminology from above, which is inspired from the naming *necessary preconditions* of Cousot et al. [105], we can state that

- total correctness triples provide *sufficient preconditions*;
- total incorrectness triples provide *sufficient postconditions*;
- partial correctness triples provide *sufficient liberal preconditions* (or *necessary postconditions*);
- partial incorrectness triples provide *sufficient liberal postconditions* (or *necessary preconditions*).

We also note that even the terminology for the predicate transformers, *strongest* post- and *weakest* precondition, might be imprecise. Indeed, as pointed by O'Hearn [6], such terminology is tied with the classical aim of Hoare logic to find either the smallest (strongest) set of necessary (overapproximating) postconditions or the largest (weakest) set of sufficient (underapproximating) preconditions. The strongest postcondition can be seen also as the *weakest sufficient postcondition*, whereas the weakest precondition is the *strongest necessary precondition.* Switching to our liberal predicate transformers, our strongest liberal post computes the *strongest necessary liberal postcondition* or, equivalently, the *weakest sufficient liberal postcondition.* Finally, our weakest liberal pre computes the *weakest sufficient liberal precondition* or the *strongest necessary liberal precondition.*

**Duality**

As a consequence of the liberal–non-liberal duality of Theorem 3.5.4, we have

$$G \implies \mathsf{wp}\,[\![C]\!]\,(F) \qquad \text{iff} \qquad \mathsf{wlp}[\![C]\!]\,(\neg F) \implies \neg G\ .$$

In other words, the triples connected to ??? are the contrapositive of total correctness triples. Similarly, ¿¿¿ is the contrapositive of total incorrectness, whereas partial incorrectness is the contrapositive of partial correctness. This implies (interestingly) that only three kind of triples fundamentally cannot be stated in terms of other triples. Nevertheless, we would argue that it is still useful to work with, e.g. ??? triples, depending on the verification aim, especially in the context of *explainable verification*: For example, if one is interested in inferring necessary preconditions, it would certainly appear easier and more natural to work and think directly with partial incorrectness, instead of complementing both the *sufficient liberal preconditions* obtained via partial correctness and the original postcondition. The resulting proof and annotations, directly in terms of *necessary preconditions*, will be much easier to understand for a working programmer.

## 3.7 Loops Rules

Reasoning about loops is one of the most challenging tasks in verification. We provide novel rules for quantitative forward and backward reasoning about loops.

**Theorem 3.7.1** (Induction Rules for Loops). *For any quantities $i, f, g \in \mathbb{A}$, boolean expression $\varphi$ and program $C$, the following proof rules for loops are valid:*

$$\frac{g \;\preceq\; i \;\preceq\; [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(i)}{g \;\preceq\; \mathsf{wlp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)}\;\text{while}-\mathsf{wlp}$$

$$\frac{g \;\curlyvee\; \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge i) \;\preceq\; i \qquad \text{and} \qquad [\neg\varphi] \curlywedge i \;\preceq\; f}{\mathsf{sp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g) \;\preceq\; f}\;\text{while}-\mathsf{sp}$$

$$\frac{[\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(i) \quad \preceq \quad i \quad \preceq \quad g}{\mathsf{wp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \;\; \preceq \; g}\;\text{while}-\mathsf{wp}$$

$$\frac{i \quad \preceq \quad g \;\curlywedge\; \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee i) \qquad \text{and} \qquad f \quad \preceq \quad [\varphi] \curlyvee i}{f \quad \preceq \quad \mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g)}\;\text{while}-\mathsf{slp}$$

The rule while−sp is novel. The while−wlp rule has already been investigated in [16, Section 5] in a probabilistic setting, but in a more restricted lattice where quantities map to the unit interval. Our definition of wlp is not probabilistic but for a more general lattice of unbounded signed quantities. Notice that while−wlp and while−sp are tightly connected by a Galois connection (cf. Theorem 3.6.1), and by taking $g = [G]$ and $f = \; [F]$ for predicates $G, F$, we conclude for both rules the validity of the Hoare triple $\models \{\, G \,\}\; \mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}\; \{\, F \,\}$ for partial correctness. Indeed, as standard in literature, the rule while−wlp requires to find an invariant that satisfy two conditions:

1. $[G] \preceq [I]$, meaning that whenever precondition $G$ holds, then the invariant $I$ also holds.

2. $[I] \preceq [\neg\varphi] \curlywedge [F] \curlyvee [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,([I])$, meaning that whenever $I$ holds, either the loop guard $\varphi$ does *not* hold, but then postcondition $F$ holds; or $\varphi$ *does* hold, but then $I$ still holds after one iteration of the loop body (or the loop body itself diverges (think: nested loops)).

By induction, (2) ensures that, starting from $I$ and no matter how many loop iterations are executed, $I$ can only terminate in states that again satisfy $I$. Assuming termination, eventually $\neg\varphi$ will hold and thus $I$ implies the postcondition $F$. (1) guarantees that the initial precondition $G$ implies $I$. Hence any state initially satisfying $G$ and on which the loop eventually terminates will do so in a final state satisfying postcondition $F$. The rule while−sp is analogous, but for forward reasoning.

The rule while−wp has also been investigated by Kaminski [16] in a probabilistic setting but again in a more restricted lattice where quantities map to unsigned positive extended reals. The rule while−slp is completely novel (since $\mathsf{slp}$ is novel). Again, by Galois connection and by taking as quantities the Iverson bracket of predicates $G, F$, we obtain for the last two rules as conclusion the validity of the triple $[\,G\,]\ \texttt{while}\,(\,\varphi\,)\,\{\,C\,\}\ [\,F\,]$ for *partial incorrectness* in the sense of Definition 3.6.2. As for an intuition, recall that validity for partial incorrectness means here that $G$ is a *necessary precondition* to end in a final state satisfying $F$ after termination of $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$. For proving this, the rule while$-\mathsf{wp}$ requires to find an invariant $I$, such that:

1. $[I] \preceq [G]$, meaning that whenever invariant $I$ holds, then the precondition $G$ also holds.

2. $[\neg\varphi] \curlywedge [F] \preceq [I]$, meaning that if the loop has terminated in postcondition $F$, then $I$ holds;

3. $[\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,([I]) \preceq [I]$, meaning that if the loop is in some state $\sigma$ in which the loop guard holds (i.e. the loop is about to be executed once more) and one loop iteration will terminate in some state where $I$ holds again, then $I$ holds for $\sigma$.

By induction, (2) and (3), which represent the first premise of while$-\mathsf{wp}$, imply that $I$ is a necessary precondition for the loop to terminate in $F$. Indeed, starting from the base case (2), for the inductive step we assume that $I$ overapproximates those states terminating in $F$ after $n$ loop iterations. By (3), $I$ also contains $[\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,([I])$, i.e., an overapproximation of those states terminating in $F$ after $n+1$ iterations. (1) guarantees that the precondition $G$ contains $I$ and hence $G$ is a necessary precondition for the loop to terminate in $F$. Again, the rule while$-\mathsf{slp}$ is analogous, but forward.

**Example 3.7.1** (Inductive Reasoning)**.** *Consider the program*

$$\texttt{while}\,(\,x < 10\,)\,\{\,x \coloneqq x + 4\,\}\ .$$

*In order to show that $x \,|\, 4$ (read: $x$ is divisible by 4) is a* necessary precondition *to terminate in postcondition $x = 12$, it is sufficient to prove the partial* *incorrectness triple* $[x = 12] \preceq \mathsf{slp}[\![C]\!]([x \mid 4])$. *If we apply the inductive rule we obtain:*

$$\frac{i \preceq [x \mid 4] \curlywedge \mathsf{slp}[\![x \coloneqq x + 4]\!]([x \geq 10] \curlyvee i) \qquad [x = 12] \preceq [x < 10] \curlyvee i}{[x = 12] \quad \preceq \quad \mathsf{slp}[\![\mathtt{while}\,(\, x < 10\,)\,\{\, x \coloneqq x + 4\,\}]\!]([x \mid 4])} \text{ while}-\mathsf{slp}$$

*Now take as invariant* $i = [x \mid 4]$. *As for the right premise, we can easily convince ourselves that* $[x = 12] \preceq [x < 10] \curlyvee [x \mid 4]$ *holds. As for the left premise, we have*

$$\begin{aligned}
& [x \mid 4] \;\curlywedge\; \mathsf{slp}[\![x \coloneqq x + 4]\!]([x \geq 10] \curlyvee [x \mid 4]) \\
= \;& [x \mid 4] \;\curlywedge\; \big([x - 4 \geq 10] \curlyvee [x - 4 \mid 4]\big) \\
= \;& [x \mid 4] \;\curlywedge\; \big([x \geq 14] \curlyvee [x \mid 4]\big) \\
= \;& [x \mid 4] \quad \succeq \quad [x \mid 4] \\
= \;& i\;.
\end{aligned}$$

*Hence we can infer the conclusion of while–*$\mathsf{slp}$ *and we have proven that* $[x \mid 4]$ *is a necessary precondition for the loop to terminate in* $[x = 12]$. $\lhd$

The forward transformers $\mathsf{sp}$ and $\mathsf{slp}$ come with an additional induction rule: under certain premises, it allows to immediately conclude that the fixpoint of the characteristic function for a quantity $f$ is *precisely* $f$ itself, i.e. the *second* Kleene iterate.

**Proposition 3.7.2.** *The following proof rules for loops are valid:*

$$\frac{\mathsf{sp}\,[\![C]\!]\,(f) \quad \preceq \quad f}{\mathsf{sp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \quad = \quad [\neg\varphi] \curlywedge f}$$

$$\frac{f \quad \preceq \quad \mathsf{slp}[\![C]\!]\,(f)}{\mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \quad = \quad [\varphi] \curlyvee f}$$

An intuition of Proposition 3.7.2 for $\mathsf{sp}$ is the following: for a loop $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$, the premise $\mathsf{sp}\,[\![C]\!]\,(f) \preceq f$ means that the value of $f$ *retrocipated for one iteration* is lower than the original value of $f$. By induction, retrocipating $f$ for any number of iterations leads to a decreasing quantity. So what is the *maximum initial value* that $f$ could have had? It is the initial quantity $f$, i.e. $\mathsf{sp}$ "gets away" with not even entering the loop. The guard $[\neg\varphi]$ in the conclusion is needed to ensure reachability. For $\mathsf{slp}$, retrocipating the execution of the loop increases the initial quantity $f$ - and hence the *minimum initial value* of $f$ is again $f$ itself.

**Example 3.7.2.** *Consider the program*

$$C = \texttt{while}\,(\,x < 10\,)\,\{\,\{\,x \coloneqq x+1\,\}\,\square\,\{\,x \coloneqq x+2\,\}\,\}$$

*and the precondition $x \geq 0$. To determine the set of states reachable from precondition $x \geq 0$, i.e. to determine $\mathsf{sp}\,[\![C]\!]\,([x \geq 0])$, we first check the premise*

$$\begin{aligned} &\mathsf{sp}\,[\![\{\,x \coloneqq x+1\,\}\,\square\,\{\,x \coloneqq x+2\,\}]\!]\,([x \geq 0]) \\ =\quad & [x-1 \geq 0] \curlyvee [x-2 \geq 0] \\ =\quad & [x \geq 1] \curlyvee [x \geq 2] \\ =\quad & [x \geq 1] \quad \preceq \quad [x \geq 0] \end{aligned}$$

*and thus conclude by Proposition 3.7.2 that*

$$\mathsf{sp}\,[\![C]\!]\,([x \geq 0]) \quad \preceq \quad [x \geq 10] \curlywedge [x \geq 0] \quad = \quad [x \geq 10]$$

*This allows to include immediately that $x \geq 10$ is the* strongest necessary post-condition *or, equivalently, the* weakest sufficient postcondition. *In particular, this result verifies that* precisely *those final states with $x \geq 10$ are* reachable *from initial states with $x \geq 0$.* ◁

## 3.8 Case Studies

In this section, we demonstrate the efficacy of quantitative strongest (liberal) post reasoning. We use the annotation style on the right to express that $g = \mathsf{sp}\,[\![C]\!]\,(f)$ (or that $g = \mathsf{slp}[\![C]\!]\,(f)$, depending on the context) and furthermore that $g' = g$. Full calculations of strongest posts are provided in Appendix A.6.

```
$/\!\!/\!\!/\ \ \boldsymbol{f}$
$C$
$/\!\!/\!\!/\ \ \boldsymbol{g}$
$^{=}/\!\!/\!\!/\ \ \boldsymbol{g'}$
```

## 3.8.1 Quantitative Information Flow — Loop Free

Consider the program $C_{\mathit{flow}} = \texttt{if}\ (\,hi > 7\,)\ \{\,lo \coloneqq 99\,\}\ \texttt{else}\ \{\,lo \coloneqq 80\,\}$. As usual in quantitative information flow, $hi$ is a secret and we want to ensure that, by observing the variable $lo$, one cannot infer information about $hi$. Below, we show $\mathsf{sp}$ (Figure 3.3) and $\mathsf{slp}$ (Figure 3.4) annotations for prequantity $hi$, i.e. we indeed show how the initial value of $hi$ flows from the top to the bottom of the computation.

```
$/\!\!/\!\!/\ \ \boldsymbol{hi}$
$\texttt{if}\ (\,hi > 7\,)\ \{$
    $/\!\!/\!\!/\ \ [\boldsymbol{hi > 7}]\ \curlywedge\ \boldsymbol{hi}$
    $lo \coloneqq 99$
    $/\!\!/\!\!/\ \ \text{Ƨ}\boldsymbol{\alpha}\colon\ [\boldsymbol{lo = 99}]\ \curlywedge\ [\boldsymbol{hi > 7}]\ \curlywedge\ \boldsymbol{hi}$
    $^{=}/\!\!/\!\!/\ \ [\boldsymbol{lo = 99}]\ \curlywedge\ [\boldsymbol{hi > 7}]\ \curlywedge\ \boldsymbol{hi}$
$\}\,\texttt{else}\,\{$
    $/\!\!/\!\!/\ \ [\boldsymbol{hi \leq 7}]\ \curlywedge\ \boldsymbol{hi}$
    $lo \coloneqq 80$
    $/\!\!/\!\!/\ \ \text{Ƨ}\boldsymbol{\alpha}\colon\ [\boldsymbol{lo = 80}]\ \curlywedge\ [\boldsymbol{hi \leq 7}]\ \curlywedge\ \boldsymbol{hi}$
    $^{=}/\!\!/\!\!/\ \ [\boldsymbol{lo = 80}]\ \curlywedge\ [\boldsymbol{hi \leq 7}]\ \curlywedge\ \boldsymbol{hi}$
$\}$
$/\!\!/\!\!/\ \ ([\boldsymbol{lo = 99}] \curlywedge [\boldsymbol{hi > 7}] \curlywedge \boldsymbol{hi})\ \curlyvee\ ([\boldsymbol{lo = 80}] \curlywedge [\boldsymbol{hi \leq 7}] \curlywedge \boldsymbol{hi})$
```

**Figure 3.3:** Full calculations of $\mathsf{sp}\,[\![C_{\mathit{flow}}]\!]\,(hi)$.

```
/// $\boldsymbol{hi}$
if ( $hi > 7$ ) {
    /// $[\boldsymbol{hi \leq 7}] \curlyvee \boldsymbol{hi}$
    $lo := 99$
    /// $\wr\,\alpha$: $[\boldsymbol{lo \neq 99}] \curlyvee [\boldsymbol{hi \leq 7}] \curlyvee \boldsymbol{hi}$
    $^{=}$/// $[\boldsymbol{lo \neq 99}] \curlyvee [\boldsymbol{hi > 7}] \curlyvee \boldsymbol{hi}$
} else {
    /// $[\boldsymbol{hi > 7}] \curlyvee \boldsymbol{hi}$
    $lo := 80$
    /// $\wr\,\alpha$: $[\boldsymbol{lo \neq 80}] \curlyvee [\boldsymbol{hi > 7}] \curlyvee \boldsymbol{hi}$
    $^{=}$/// $[\boldsymbol{lo \neq 80}] \curlyvee [\boldsymbol{hi > 7}] \curlyvee \boldsymbol{hi}$
}
/// $\big([\boldsymbol{lo \neq 99}] \curlyvee [\boldsymbol{hi \leq 7}] \curlyvee \boldsymbol{hi}\big) \curlywedge \big([\boldsymbol{lo \neq 80}] \curlyvee [\boldsymbol{hi > 7}] \curlyvee \boldsymbol{hi}\big)$
```

**Figure 3.4:** Full calculations of $\mathsf{slp}[\![C_{\mathit{flow}}]\!]\,(hi)$.

Let us first note that we can precisely infer the set of states that are reachable after executing $C_{\mathit{flow}}$ by recalling that for a prequantity $f$ strictly larger than $-\infty$, $\mathsf{sp}\,[\![C]\!]\,(f)\,(\tau) = -\infty$ if and only if $\tau$ is unreachable. When does the (left) expression $\big([lo = 99] \curlywedge [hi > 7] \curlywedge hi\big) \curlyvee \big([lo = 80] \curlywedge [hi \leq 7] \curlywedge hi\big)$ evaluate to something larger than $-\infty$? This is precisely the case if either the final value of $lo$ is 99 and $hi$ is larger than 7, or if $lo$ is 80 and $hi$ smaller or equal 7. The reachable states are thus given by

$$\begin{aligned} &\{\tau \mid \mathsf{sp}\,[\![C]\!]\,(hi)\,(\tau) \neq -\infty\} \\ =\ &\{\tau \mid \tau(lo) = 99 \wedge \tau(hi) > 7 \ \vee\ \tau(lo) = 80 \wedge \tau(hi) \leq 7\}\ . \end{aligned}$$

The same insight could have been achieved with $\mathsf{slp}$ by computing $\{\tau \colon \mathsf{slp}[\![C]\!]\,(hi)\,(\tau) \neq +\infty\}$.

Secondly, we can — in a principled way — construct from the $\mathsf{sp}$ and $\mathsf{slp}$ annotations a function $\xi$ that, given the final value of only the observable variable $lo$ (which we denote $lo'$), returns the set containing an overapproximation of all possible *initial* values of the quantity $hi$, namely:

$$\begin{aligned}\xi(lo') &= \{\, \alpha \mid \tau \in \Sigma, \tau(lo) = lo', \mathsf{slp}[\![C_{\mathit{flow}}]\!]\,(hi)\,(\tau) \leq \alpha \leq \mathsf{sp}\,[\![C_{\mathit{flow}}]\!]\,(hi)\,(\tau)\,\} \\ &= \begin{cases} \{\, \alpha \mid 7 < \alpha \,\}, & \text{if } lo' = 99 \\ \{\, \alpha \mid \alpha \leq 7 \,\}, & \text{if } lo' = 80 \\ \emptyset, & \text{otherwise.} \end{cases}\end{aligned}$$

Now, what can we infer about the secret initial value of $hi$ by observing only the final value $lo'$? If $lo' = 99$, then $hi$ must be larger than 7; if $lo = 90$, then $hi$ must be smaller or equal 7, and otherwise this state was actually unreachable (and hence such a situation could have not been observed in the first place). Hence, observing the final value of $lo$ leaks information about the secret $hi$. In fact, by having used both $\mathsf{sp}$ and $\mathsf{slp}$, the above gave us *precisely* the entire information that is leaked about $hi$ from observing the final value of $lo$.

### 3.8.2 Quantitative Information Flow for Loops

Consider the program $C_{\mathit{while}} = hi \coloneqq hi + 5 \,⨾\, \texttt{while}\,(\, lo < hi \,)\,\{\, lo \coloneqq lo + 1 \,\}$. Again, we show below the $\mathsf{sp}$ (Figure 3.3) and $\mathsf{slp}$ (Figure 3.4) annotations for prequantity $hi$.

```
/// hi
hi := hi + 5
/// hi − 5
while ( lo < hi ) {
    lo := lo + 1
}
/// [lo ≥ hi] ⅄ (hi − 5)
```

**Figure 3.5:** Full calculations of $\mathsf{sp}\,[\![C_{\mathit{while}}]\!]\,(\mathit{hi})$.

```
/// hi
hi := hi + 5
/// hi − 5
while ( lo < hi ) {
    lo := lo + 1
}
/// [lo < hi] ⋎ (hi − 5)
```

**Figure 3.6:** Full calculations of $\mathsf{slp}[\![C_{\mathit{while}}]\!]\,(\mathit{hi})$.

For sp and slp of the loop, the Kleene iteration stabilizes after 2 iterations, see Appendix A.6 for detailed computations. There is no need for invariant, nor reasoning about limits, or anything alike. Even more conveniently, we can alternatively apply Proposition 3.7.2: indeed, for instance for sp we have $\mathsf{sp}\,[\![lo := lo + 1]\!]\,(hi - 5) = hi - 5 \preceq hi - 5$ and thus Proposition 3.7.2 yields that sp of the loop is *precisely* $[lo \geq hi] \curlywedge (hi - 5)$.

We construct (again) the function $\xi$ that, given the final value $lo'$ of the variable $lo$, returns an overapproximation of all possible *initial* values of the quantity $hi$, and obtain $\xi(lo') \;=\; \{\, \alpha \mid \alpha \leq lo' - 5 \,\}$. Hence, by observing only the final value $lo'$ we infer that $hi$ must be at most $lo' - 5$. In fact, any of such value $\alpha \leq lo' - 5$ after being incremented by 5 leads to a value that $\alpha' \leq lo'$, so without entering the loop, $C_{\mathit{while}}$ terminates with the correct final value $lo'$. Again, using both sp and slp, we obtain *precisely* the entire information that is leaked about $hi$ from observing the final value of $lo$.

**Quantitative Information Flow for Loops using wp.**

The set $\xi(lo')$ could have alternatively been determined with classical weakest preconditions: In fact, $\mathsf{wp}\,[\![C]\!]\,([lo = lo'])$ is the set of all initial states that will end with a final state where $lo = lo'$, and by projecting only to the values of the variable $hi$ we obtain all initial values of $hi$. However, aside from a (perhaps subjective) elegance perspective, we point out that the computation of $\mathsf{wp}\,[\![C]\!]\,([lo = lo'])$ is actually more involved: the Kleene's iterates of the loop for $\mathsf{wp}$ stabilize only at $\omega$ – not 2:

$$\begin{aligned}
\Phi(\mathsf{false}) &= [lo \geq hi] \wedge [lo = lo'] \\
\Phi^2(\mathsf{false}) &= [lo \geq hi] \wedge [lo = lo'] \vee [lo' - 1 < hi \leq lo'] \wedge [lo = lo' - 1] \\
\Phi^3(\mathsf{false}) &= [lo \geq hi] \wedge [lo = lo'] \vee [lo' - 1 < hi \leq lo'] \wedge [lo = lo' - 1] \\
&\quad \vee [lo' - 1 < hi \leq lo'] \wedge [lo = lo' - 2] \\
&\vdots \\
\Phi^\omega(\mathsf{false}) &= [hi \leq lo] \wedge [lo = lo'] \vee \left( \bigvee_{n=1}^{\omega} [lo' - 1 < hi \leq lo'] \wedge [lo = lo' - n] \right)
\end{aligned}$$

Reasoning about this requires some form of creativity or advanced technique: either reasoning about the limit, or finding an invariant plus a termination prove. Only after determining $\Phi^\omega(\mathsf{false})$, one can perform the $\mathsf{wp}$ for the assignment, which again results in a huge formula. For $\mathsf{sp}$ and $\mathsf{slp}$, the Kleene's iterates stabilize after 2 iterations (Appendix A.6): no need for invariant nor reasoning about limits nor projections of huge formulas.

### 3.8.3 Automation

Our calculi, in their full generality, cannot be fully automated, which is not surprising since our calculi can express both termination and reachability properties for a Turing-complete computational model – both of which are well known to be undecidable [106; 57]. Nevertheless, we believe that our calculi are at least syntactically mechanizable. For this aim, we plan to investigate an expressive "assertion" language for quantities, such as the one proposed by Batz et al. [102] for quantitative reasoning about probabilistic programs. This would allow showing *relative completeness* in the sense of Cook [107], i.e., decidability modulo checking whether $g \preceq f$ holds, where $g, f$ may contain suprema and infima. Similar problems (decidability modulo checking a logical implication) exist for classical predicate transformers and Hoare logic [107].

We also point out that the main goal of our calculi is to provide a framework, on which future tools for (partially) automating quantitative $\mathsf{wlp}/\mathsf{sp}/\mathsf{slp}$ proofs can ground. For example, it may well be possible to fully automate the transformers for some syntactic (e.g. linear) fragments of $\mathsf{nGCL}$.

### 3.8.4 Partial Incorrectness Reasoning

We now show an application of partial incorrectness triples and, hence, of our strongest liberal postconditions. Consider a program/system $C_{login}$ that takes as input a variable *password*. If *password* contains the correct password, say `"oopsla2022"`, then $C_{login}$ terminates in a final state containing a boolean variable "*access*" storing the value $\mathsf{true}$; otherwise, the program terminates with value $access = \mathsf{false}$. Now, recall that

$$\mathsf{slp}[\![C_{login}]\!]\,([password = \texttt{"oopsla2022"}])$$

is a predicate characterizing those final states which are reached *only* by initial states $\sigma$ with the correct password, i.e. initial states with $\sigma(password) =$ `"oopsla2022"`. If the partial incorrectness triple $[\,access = \mathsf{true}\,]\ C_{login}\ [\,password = \texttt{"oopsla2022"}\,]$, which translates to

$$[access = \mathsf{true}] \quad\Longrightarrow\quad \mathsf{slp}[\![C]\!]\,([password = \texttt{"oopsla2022"}])\ ,$$

holds, then knowing the correct password is a *necessary precondition* to access the system. In other words, validity of the partial incorrectness triple guarantees that no user without knowledge of the correct password can end up in a final state $\tau$ where $\tau(access) = \mathsf{true}$.

We also note that, by the Galois Connection of Theorem 3.6.2, one can check whether the partial incorrectness triple holds also by employing $\mathsf{wp}$:

$$\mathsf{wp}\,[\![C]\!]\,([access = \mathsf{true}]) \implies [password = \texttt{"oopsla2022"}]$$

However, reasoning with $\mathsf{slp}$ may well (1) be more feasible in practice (as demonstrated in Section 3.8.2) as well as (2) more intuitive when reasoning about *necessary preconditions* to access a system.

## 3.9 Related Work

**More General Predicate Transformers.**

Aguirre and Katsumata [108] focus on an abstract theory of $\mathsf{wp}$ for *loop-free* programs. In particular, our $\mathsf{w(l)p}$, *restricted to the fragment of loop-free programs*, can be derived by instantiating their Corollary 4.6. In fact, consider:

- the powerset monad $\mathcal{P}$;
- the lattice of extended reals $\mathbb{R}^{\pm\infty}$;
- the Eilenberg-Moore algebra $\sup \colon \mathcal{P}(\mathbb{R}^{\pm\infty}) \to \mathbb{R}^{\pm\infty}$.

As a consequence of [108, Corollary 4.6], we obtain an abstract operation $\mathsf{awp}\colon (A \to \mathcal{P}(B)) \to (B \to \mathbb{R}^{\pm\infty}) \to (A \to \mathbb{R}^{\pm\infty})$ such that:

$$\mathsf{awp}(C)(f)(a) = \sup_{b \in C(a)} f(b)$$

Note that $\mathsf{awp}$ preserves all joins in the position of $f$. By taking as monad the collecting semantics starting from a single state $[\![C]\!]\colon \Sigma \to \mathcal{P}(\Sigma)$ which maps states into set of states, for all loop-free programs $C$, $f \in \mathbb{A}, \sigma \in \Sigma$ we have:

$$\mathsf{awp}([\![C]\!])(f)(\sigma) = \sup_{\tau \in [\![C]\!](\sigma)} f(\tau) = \mathsf{wp}\,[\![C]\!]\,(f) \ .$$

Similarly, if we consider the Eilenberg-Moore algebra inf, we obtain an abstract operator $\mathsf{awlp}$ such that:

$$\mathsf{awlp}([\![C]\!])(f)(\sigma) = \inf_{\tau \in [\![C]\!](\sigma)} f(\tau) = \mathsf{wlp}[\![C]\!]\,(f) \ .$$

Aguirre and Katsumata [108, Section 4.1] also define an *abstract* strongest post-condition as a left adjoint of their weakest precondition (without constructing it); we believe that, due to our Theorem 3.6.2, an abstract strongest liberal post can be defined dually as a right adjoint of their weakest precondition. On the other hand, our definition of strongest post is explicitly given by induction on the program structure and not implicitly as an adjoint. The difficulties with finding strongest posts for probabilistic programs demonstrate that an explicit definition of a strongest post is more than desirable.

**Strongest Liberal Post**

The term *"strongest liberal postcondition"* is sometimes used in the literature for the original non-liberal strongest postcondition, see e.g. [109, Section 2.2], [110, Section 0], or [111, Definition 8]. In fact, [109, Section 2.2] argues that the strongest postcondition is often denoted also as strongest liberal postcondition due to the relationship between weakest liberal pre. However, since wlp "allows" nontermination whereas wp does not, and analogously slp "allows" unreachability whereas sp does not, we believe that our naming convention of slp and sp is more appropriate and natural.

**Information Flow Analysis**

Some previous work on information flow analysis use type systems [56; 112]. However, these are imprecise and may reject safe programs such as $lo := hi \fatsemi\ lo := 0$ due to a *potential* flow from $hi$ to $lo$ [113]. A Hoare-like logic combined with abstract interpretation has been proposed by Amtoft and Banerjee [113], but fails for simple programs such as [113, Section 9], which instead can be easily detected with our s(l)p analysis. Other abstract interpretation-based techniques focus on the trace semantics [114; 115]. Urban et al. [53] verify dependency fairness of neural networks by applying a backward analysis to compute the set of input values that lead to a certain ouput value; this approach is similar to a wp-based calculus with ghost variables, as shown in Example 3.8.2, and we speculate that sp-based approaches could also be applied and potentially lead to better performances (as shown in Example 3.8.2). The work of Mazzucato et al. [116], which quantifies the impact of input variables on other outputs, also employs a backward analysis approach. In Security Concurrent Separation logic [69] the authors provide an extension of concurrent separation logic [117; 118] by adding sensitivity assertions which, roughly, assigns to a certain variable a certain degree of security; however, their proof system deals only with partial correctness and restricts to conditional statements and loops that cannot use sensitive variables, so that our examples from Section 3.8 cannot be covered by their logic. Differently from the aforementioned works, our framework provides *quantitative details* about the amount of information flow, instead of a single boolean output, see [119] for an overview.

## 3.10 Conclusion & Future Work

We have presented a novel *quantitative strongest post calculus* that subsumes classical strongest postconditions. Moreover, we developed a novel *quantitative strongest <u>liberal</u> post* calculus. Restricted to a Boolean setting, we obtain the – to the best of our knowledge – unexplored notion of *strongest liberal postconditions* which ultimately lead to our definition of *partial incorrectness*. The latter connection is justified by the fundamental Galois connection between $\mathsf{slp}$ and $\mathsf{wp}$, and the strong duality between total and partial correctness, but where we replace *nontermination* with *unreachability*. Finally, we notice that there are three additional Hoare-style triples that can be naturally defined using our transformers, and we identify a precise connection between *partial incorrectness* and the so-called *necessary preconditions* [105].

As future work, we plan to investigate the newly observed Hoare triples and to provide novel proof systems for them. We also plan to extend our quantitative strongest calculi with heap manipulation, similarly to the work of [96] for weakest pre calculi; this could lead to connections with incorrectness separation logic [76].

Finally, we plan to deepen the applications of quantitative strongest post calculi to quantitative information flow, perhaps by establishing connections with abstract interpretation [3]. In fact, we believe that our $\mathsf{s(l)p}$ transformers can be viewed as sound approximations of the fiber of the concrete semantics. Examples 3.8.1, 3.8.2 go into this direction after-all, since the combination of our strongest and strongest liberal post calculi can be viewed as an *interval abstraction* [120] of the possible initial values of a certain pre-quantity.

# Chapter 4

# Quantitative Transformers For Weighted Programs

Sections 4.2, 4.4, 4.5, and 4.6 of this chapter are based on the paper [18]. This chapter builds upon the quantitative strongest postcondition calculus for non-deterministic programs introduced in the previous chapter Section 3.4, lifting it to the realm of weighted programs, including both nondeterministic and probabilistic variants. While backward reasoning via weakest precondition style has been established for weighted programs [24], the goal here is to facilitate forward reasoning in various domains where quantitative properties are crucial, such as optimization problems and formal languages. As stated by Batz et al. [24], forward reasoning is far from trivial:

> An interesting direction for future work is to explicitly construct a *strongest postcondition* transformer for weighted programming, which Aguirre and Katsumata [108] define non-constructively as an adjoint to $\mathsf{wp}$. Problems with defining strongest postexpectations for probabilistic programs, see [95], demonstrate that giving a *concrete* strongest post semantics is far less easy, even if it can be defined abstractly as an adjoint.

Our novel weighted strongest post transformer uncovers new dualities between forward and backward transformers, correctness and incorrectness, as well as nontermination and unreachability. These dualities provide new insights into fundamental aspects of program semantics and reasoning, offering a robust framework for addressing complex quantitative questions in program analysis.

In subsequent sections, we will delve into Weighted Programming, which roughly can be seen as a generalisation of probabilistic programming. We will present a weakest precondition calculus similar to [24], which enables quantitative reasoning about programs in a weighted programming language and more originally we will later present our novel quantitative strongest post.

## 4.1 Introduction

### 4.1.1 Probabilistic Programming

The semantics of structured probabilistic programs were first rigorously studied by Kozen in the late 1970s and early 1980s [38; 39; 42; 13]. In contrast to Dijkstra's Guarded Command Language, which incorporates nondeterministic choice similarly to our earlier discussion in Section 3.2, Kozen replaced nondeterminism with probabilistic choice. Building upon Kozen's foundational work, McIver and Morgan later reintroduced nondeterministic choices into the probabilistic framework [121; 14], developing a programming language that accounts for both randomness and nondeterminism. A comprehensive overview of these concepts can be found in [16]. Here, we adopt their approach and present a variant of their probabilistic Guarded Command Language (pGCL), which effectively models both types of uncertainty: randomness and nondeterminism.

Beyond its role in randomized algorithms for accelerating the solution of computationally intractable problems, probabilistic programming has seen rapidly increasing interest in machine learning over the past decade. In this domain, probabilistic programs serve as intuitive algorithmic descriptions of complex probability distributions. As Gordon et al. [122] aptly stated:

> The goal of probabilistic programming is to enable probabilistic modeling [. . . ] to be accessible to the working programmer, who has sufficient domain expertise, but perhaps not enough expertise in probability theory [. . . ].

The distinction between probabilistic and nondeterministic uncertainty is illustrated using the example of a fair coin. In probabilistic uncertainty, such as flipping a fair coin, we can assign a specific probability (e.g., 50%) to each outcome, enabling quantitative reasoning. For instance, the probability of flipping ten heads in a row is extremely low (about 0.1%), making it a safe bet to take.

In contrast, with nondeterministic uncertainty—where an unknown oracle determines the outcome—we cannot assign a meaningful probability to the events. This type of uncertainty only allows us to state whether an outcome is possible, without quantifying it.

In summary, randomness represents a form of uncertainty that allows for quantitative reasoning by assigning probabilities to outcomes. Nondeterminism, on the other hand, only permits qualitative reasoning, as we can only determine whether an outcome is possible or not, without quantifying it. Combining probabilistic and nondeterministic behaviors within a single computational process introduces significant difficulties and is an ongoing area of research [123; 124; 125; 126; 127; 128]. Similarly to our approach in Chapter 3, we will adopt an *angelic* method, which tackles these difficulties by resolving nondeterminism in the most advantageous manner.

### 4.1.2 Weighted Programming

In this work, we will focus on weighted programming, a generalization of probabilistic programming that relies on weights beyond simple probabilities. Weighted programming is a versatile paradigm designed to specify a broad spectrum of mathematical models. As highlighted by Batz et al. [24], weighted programs are characterized by two distinctive features: (1) nondeterministic branching and (2) the ability to assign weights to execution traces. These weights are not confined to numerical values but can also include words from an alphabet, polynomials, formal power series, or even cardinal numbers. This flexibility allows weighted programming to extend its applicability beyond the realm of probabilistic programming, which primarily deals with probability distributions, enabling the modeling of a wider array of mathematical constructs.

Similar to probabilistic programming, weighted programming is particularly advantageous for individuals with a programming background who may lack extensive expertise in advanced mathematical theory. By providing a structured and accessible framework, weighted programming facilitates the modeling of diverse mathematical phenomena directly through code, making mathematical modeling more intuitive and practical.

It is important to recognize that Weighted Programs have been the focus of prior research. Brunel et al. [129] explored functional languages with weightings and developed a static analysis technique for proving upper bounds on these weights. Aguirre and Katsumata [108] developed an abstract weakest precondition calculus for loop-free programs. Independently, Batz et al. [24] focused on a concrete setting, offering constructive definitions for a weakest precondition calculus that includes loops and demonstrating a wide range of applications. Gaboardi et al. [130] introduced Graded Hoare Logic, which can be viewed as a Hoare logic specifically tailored for weighted programming, although it is restricted to bounded loops.

## Weighted Strongest Postcondition

One of our key advances is to anticipate the strongest postcondition ($\mathsf{sp}$) rather than use a standard operational semantics such as that of Batz et al. [24, Section 3.3]. To achieve this, we developed a novel forward weighted $\mathsf{sp}$ transformer; it is interesting that within our framework (1) we subsume *both* $\mathsf{sp}$ and $\mathsf{slp}$ (arguably the main contributions of Zhang and Kaminski [17]), and (2) the order of factors changes in some rules.

To demonstrate this, we use the $\odot$ operator, which represents multiplication in semirings. Semirings provide the mathematical foundation for reasoning about various program behaviors (further details in Section 4.2.1). In our programming language, these semiring elements are used as weights for traces. For example, Boolean weights can be used to describe which traces are possible in a nondeterministic program, whereas real-valued weights quantify the likelihoods of probabilistic outcomes.

In commutative semirings, the order of multiplication does not matter; that is, $a \odot b = b \odot a$ for any elements $a$ and $b$. However, in non-commutative semirings—which deal with sequences and order-sensitive operations—$a \odot b$ may not equal $b \odot a$. An example is the formal languages semiring in Example 4.7.1, where $\odot$ corresponds to word concatenation, which is clearly order dependent.

Now, we investigate the predicate transformer semantics of these weighting constructs. Below, we see that $\mathsf{sp}$ weights the result in the opposite order as compared to $\mathsf{wp}$.

$$\mathsf{sp}\,[\![\odot\, e]\!]\,(f) = f \odot e \qquad\qquad \mathsf{wp}\,[\![\odot\, e]\!]\,(f) = e \odot f$$

Our loop rule, while similar to those in [17, Table 2], features a slightly different factor order as well. These differences, while subtle, are crucial, and the correctness of our rules is supported by the novel dualities presented in Theorem 4.7.4 and Example 4.7.1. This underscores that previous rules [64; 17] were accurate only because they used commutative semirings. Indeed, in commutative semirings, the order of multiplication does not matter. However, in non-commutative semirings such as the formal languages semiring in Example 4.7.1 where multiplication corresponds to word concatenation, the order of multiplication is relevant.

## 4.2 Syntax and Semantics

We introduce a language of commands $\mathsf{wReg}$, which encompasses nondeterministic imperative constructs similar to those found in the Guarded Command Language [1]. Furthermore, we adopt the weighting assertion as in [24; 74], which enables representation of general weights over states. This includes reasoning of expected values over probability distributions, as studied in [16; 14].

### 4.2.1 Algebraic Preliminaries for Weights

We begin by reviewing some algebraic structures, starting with the weights of computation traces.

**Definition 4.2.1** (Naturally Ordered Semirings). *A* monoid $\langle U, \oplus, \mathbb{0}\rangle$ *consists of a set* $U$*, an associative binary operation* $\oplus\colon U \times U \to U$*, and an identity element* $\mathbb{0} \in U$ *(with* $u \oplus \mathbb{0} = \mathbb{0} \oplus u = u$*). The monoid is* partial *if* $\oplus\colon U \times U \rightharpoonup U$ *is partial, and* commutative *if* $\oplus$ *is commutative (i.e.* $u \oplus v = v \oplus u$*).* ◁

**Definition 4.2.2** (Semirings). *A* semiring $\langle U, \oplus, \odot, \mathbb{0}, \mathbb{1}\rangle$ *is an algebraic structure such that* $\langle U, \oplus, \mathbb{0}\rangle$ *is a commutative monoid,* $\langle U, \odot, \mathbb{1}\rangle$ *is a monoid, and the following additional properties hold:*

1. *Distributivity:*

$$u \odot (v \oplus w) = u \odot v \oplus u \odot w$$

$$(u \oplus v) \odot w = u \odot w \oplus v \odot w$$

2. *Annihilation:*

$$\mathbb{0} \odot u = u \odot \mathbb{0} = \mathbb{0}$$

*The semiring is* partial *if* $\langle U, \oplus, \mathbb{0}\rangle$ *is a partial monoid (but* $\odot$ *is total).* ◁

Unlike [24] which exclusively focused on total semirings, our framework accommodates partial semirings where addition may be undefined for certain elements. We will rely on this extension to model probabilistic computations where probabilities cannot exceed 1, aligning with the approach taken in [19].

**Definition 4.2.3** (Complete semirings [131]). *A (partial) semiring* $\langle U, \oplus, \odot, \mathbb{0}, \mathbb{1}\rangle$ *is complete if there is a sum operator* $\bigoplus_{i\in I}$ *with the following properties:*

1. *If* $I = \{i_1, \ldots, i_n\}$ *is finite, then* $\bigoplus_{i\in I} u_i = u_{i_1} + \cdots + u_{i_n}$.
2. *If* $\bigoplus_{i\in I} x_i$ *is defined, then* $v \odot \bigoplus_{i\in I} u_i = \bigoplus_{i\in I} v \odot u_i$ *and* $(\bigoplus_{i\in I} u_i) \odot v = \bigoplus_{i\in I} u_i \odot v$.

3. *Let $(J_k)_{k\in K}$ be a family of nonempty disjoint subsets of $I$ ($I = \bigcup_{k\in K} J_k$ and $J_k \cap J_l = \emptyset$ if $k \neq l$), then $\bigoplus_{k\in K} \bigoplus_{j\in J_k} u_j = \bigoplus_{i\in I} u_i$.*

**Definition 4.2.4** (Scott Continuity [132]). *A (partial) semiring with order $\leq$ is Scott Continuous if for any directed set $D \subseteq X$ (where all pairs of elements in $D$ have a supremum), the following hold:*

$$\sup_{x\in D} (x \oplus y) = (\sup D) \oplus y$$

$$\sup_{x\in D} (x \odot y) = (\sup D) \odot y$$

$$\sup_{x\in D} (y \odot x) = y \odot \sup D$$

**Definition 4.2.5** (Natural order). *On a (partial) semiring $\langle U, \oplus, \odot, \mathbb{0}, \mathbb{1}\rangle$, we define a relation $\leq$ by*

$$u \leq v \iff \exists w\colon u \oplus w = v.$$

*The semiring is called* naturally ordered *if $\leq$ is a complete partial order.* $\lhd$

As shown later in Figure 4.1, semirings will serve as the structure from which we draw weights of computation traces in our semantics. To this end, we extend the definition of quantities [17, Definition 3.1] to any semiring, similar to Zilberstein [74, Definition 2.3].

**Definition 4.2.6** (Quantities). *Given a partial semiring $\mathcal{A} = \langle U,\, \oplus,\, \odot,\, \mathbb{0},\, \mathbb{1}\rangle$, the set $\mathbb{A}_{\mathcal{A}}(X)$ of all* quantities *is defined as the set of all functions $f\colon X \to U$, i.e.*

$$\mathbb{A}_{\mathcal{A}}(X) \;=\; \{\, f \mid f\colon X \to U \,\} \;.$$

We will write $\mathbb{A}$ instead of $\mathbb{A}_{\mathcal{A}}(X)$ when $\mathcal{A}$ and $X$ are clear from context. Semiring addition, scalar multiplication, and constants are lifted pointwise to quantities as follows:

$$\begin{aligned}(m_1 \oplus m_2)(x) &\triangleq m_1(x) \oplus m_2(x)\\(u \odot m)(x) &\triangleq u \odot m(x)\\u(x) &\triangleq u\end{aligned}$$

For example, by taking $X$ as the set of program states $\Sigma$ and the semiring $\langle \mathbb{R}^{\pm\infty}, \max, \min, -\infty, +\infty \rangle$ one can represent the quantities of Zhang and Kaminski [17, Definition 3.1]. Other instances of semirings encode other computations. For example:

- Nondeterministic computation employs the Boolean semiring $\mathsf{Bool} = \langle \{0,1\}, \vee, \wedge, 0, 1 \rangle$.
- Randomization in $\mathsf{OL}$ [19] adopts probabilities in the partial semiring $\mathsf{Prob} = \langle [0,1], +, \cdot, 0, 1 \rangle$, where $x + y$ is undefined if $x + y > 1$.
- Expectations in quantitative weakest pre [14; 15] adopts non-negative values in the semiring $\mathsf{PosReals} = \langle \mathbb{R}_{\geq 0}, +, \cdot, 0, +\infty \rangle$.
- Optimization problems (e.g., the path with minimum weight) can be encoded via the tropical semiring $\mathsf{Tropical} = \langle [0, +\infty], \min, +, +\infty, 0 \rangle$ which utilises non-negative real-valued weights with minimum and addition operations.

We refer to Batz et al. [24, Table 1] and Zilberstein [74, Section 2] for more examples and details.

### 4.2.2 Program States and Quantities

A state $\sigma$ is a function that assigns a natural-numbered value to each variable. To ensure that the set of states is countable, we restrict to a finite set of program variables $\mathsf{Vars}$ (which is not restrictive as every program in our language admits a finite number of variables).

The set of program states is given by

$$\Sigma = \{\, \sigma \mid \sigma \colon \mathsf{Vars} \to \mathbb{N} \,\}^{1}.$$

The semantics of an arithmetic, boolean or weight expression $e$ is denoted by $[\![e]\!] \colon \Sigma \to \mathbb{N} \cup U$ and is obtained in a state $\sigma$, by evaluating $e$ after replacing all occurrences of variables $x$ by $\sigma(x)$. We occasionally write $\sigma(e) \triangleq [\![e]\!](\sigma)$ following existing conventions [16; 17; 24], to denote the evaluation of an expression in a state $\sigma$. Moreover, we denote by $\sigma\,[x/v]$ a new state obtained from $\sigma$ by setting the valuation of $x \in \mathsf{Vars}$ to $v \in \mathbb{N}$. Formally:

$$\sigma\,[x/v] \quad = \quad \lambda y\colon \begin{cases} v & \text{if } y = x \\ \sigma(y) & \text{otherwise .} \end{cases}$$

A particular useful quantity is the Iverson bracket [103]: denoted as $[\varphi]$ for a given predicate $\varphi$, it takes as input a state $\sigma$ and evaluates to 1 if the statement is true and 0 if the statement is false. We generalise it to arbitrary semirings, subsuming other quantitative generalisations such as [17, Definition 3.5].

**Definition 4.2.7** (Iverson Brackets). *For any semiring $\mathcal{A} = \langle U, \oplus, \odot, \mathbb{0}, \mathbb{1}\rangle$ and a predicate $\varphi$ over program states $\Sigma$, the Iverson bracket $[\varphi] : \Sigma \to U$ is defined as*

$$[\varphi]\,(\sigma) \;\triangleq\; \begin{cases} \mathbb{1} & \text{if } \sigma \models \varphi \\ \mathbb{0} & \text{otherwise .} \end{cases} \qquad \lhd$$

When the context is clear, for a constant $\varphi$ we will occasionally write

$$[\varphi] \quad = \quad \begin{cases} \mathbb{1} & \text{if } \varphi \text{ holds} \\ \mathbb{0} & \text{otherwise .} \end{cases}$$

Every quantity $f \colon \Sigma \to U$ can be seen as a set of states via its support

[1]Following [24], we restrict $\Sigma$ a priori to avoid technical issues.

$\mathsf{supp}\,(f) = \{\sigma\colon f(\sigma) \neq \mathbb{0}\}$, whereas every set can be converted into a quantity via Iverson brackets. For example, the set of reachable states starting from $\phi \subseteq \Sigma$ is given by $\mathsf{supp}\,(\mathsf{sp}\,[\![C]\!]\,([\phi]))$.

**Remark 4.2.1** (Quantities as sets)**.** *In quantitative literature [16; 14] every set $P$ can be uniquely identified via its Iverson brackets. Here, multiple quantities may be associated with the same set of states, as the semiring may contain more than just $\mathbb{0}$ and $\mathbb{1}$. This is because in our framework we weight the states in a set, rather than just considering whether they are in the set or not.*

### 4.2.3 Weighted Programs

Throughout the thesis, we denote $\mathcal{A} = \langle U, \oplus, \odot, \mathbb{0}, \mathbb{1}\rangle$ as a naturally ordered, complete, Scott continuous, partial semiring with a top element $\top \in U$ such that $\top \geq u$ for all $u \in U$. We assign meaning to wReg-statements in terms of a denotational semantics, taking as input an *initial state* $\sigma$ and a *final state* $\tau$, and returning the sum of the weights of all paths starting from $\sigma$ and terminating in $\tau$ after the execution of $C$. The syntax of the *weighted regular command language* (wReg) is below:

$$
\begin{aligned}
C ::=\ & x \coloneqq e && \text{(assignment)}\\
\mid\ & x \coloneqq \texttt{nondet()} && \text{(nondeterministic assignment)}\\
\mid\ & \odot\, e && \text{(weighting)}\\
\mid\ & C \fatsemi C && \text{(sequencing)}\\
\mid\ & \{\, C \,\} \,\Box\, \{\, C \,\} && \text{(nondeterministic choice)}\\
\mid\ & C^{\langle e, e' \rangle} && \text{(iteration)}
\end{aligned}
$$

where $\odot\, e$ weights the current computation branch. Similarly to [17; 24], we do not provide an explicit syntax for weights because we focus on semantic assertions. Our weighting construct is more expressive than Batz et al. [24]; Zilberstein [74]: not only we can represent values $u \in U$ and Boolean tests (via Iverson brackets), but we also reason about intensional properties of the computation. The iteration construct $C^{\langle e,e' \rangle}$, introduced in [74], represents a fundamental weighted control flow structure that either terminates with weight $e'$ or executes the body $C$ with weight $e$ before continuing the iteration. This construct elegantly unifies several common programming patterns: while loops can be expressed as $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\} = C^{\langle \varphi, \neg\varphi \rangle}$, probabilistic iterations as $C^{\langle p, 1-p \rangle}$ (where $p$ represents execution probability), and Kleene's star operation as $C^{\langle \mathbb{1}, \mathbb{1} \rangle}$ (allowing arbitrary repetition). The construct is particularly valuable in the context of partial semirings, where traditional approaches to defining loops through Kleene star operations become problematic due to the inherently nondeterministic nature of such operations, which are not well-defined in general [74, Footnote 1].

We show below a list of common constructs defined as syntactic sugar.

$$\begin{array}{rcl}
\texttt{assume}\ \varphi & \triangleq & \odot\,\varphi \qquad\qquad \texttt{diverge} \ \triangleq\ \odot\,\mathbb{0} \\
\texttt{if}\ (\,\varphi\,)\ \{\,C_1\,\}\ \texttt{else}\ \{\,C_2\,\} & \triangleq & \{\,\texttt{assume}\ \varphi\,\fatsemi\ C_1\,\} \mathbin{\square} \{\,\texttt{assume}\ \neg\varphi\,\fatsemi\ C_2\,\} \\
\{\,C_1\,\}\ [\,p\,]\ \{\,C_2\,\} & \triangleq & \{\,\odot\,p\,\fatsemi\ C_1\,\} \mathbin{\square} \{\,\odot\,1-p\,\fatsemi\ C_1\,\} \\
\texttt{while}\,(\,\varphi\,)\,\{\,C\,\} & \triangleq & C^{\langle \varphi, \neg\varphi \rangle} \qquad C^{\star} \ \triangleq\ C^{\langle \mathbb{1}, \mathbb{1} \rangle}
\end{array}$$

The semantics is shown in Figure 4.1 and is described below.

**Assignment:**

The semantics for assignment asserts that the weight of transitioning from $\sigma$ to $\tau$ after executing $x \coloneqq e$ is $\mathbb{1}$ if $\tau$ is equal to $\sigma$ with the value of $x$ updated to $\sigma(e)$, or $\mathbb{0}$ otherwise.

**Nondeterministic Assignment:**

The denotational semantics for $x \coloneqq \texttt{nondet()}$,

$$[\![ x \coloneqq \texttt{nondet()} ]\!](\sigma, \tau) = \bigoplus_{\alpha \in \mathbb{N}} [\sigma\,[x/\alpha] = \tau]\ ,$$

$$\begin{aligned}
[\![x \coloneqq e]\!](\sigma,\tau) &\triangleq [\sigma\,[x/\sigma(e)] = \tau] && \text{(assignment)}\\
[\![x \coloneqq \texttt{nondet()}]\!](\sigma,\tau) &\triangleq \bigoplus_{\alpha\in\mathbb{N}} [\sigma\,[x/\alpha] = \tau] && \text{(nondeterministic assignment)}\\
[\![\odot\, e]\!](\sigma,\tau) &\triangleq [\![e]\!](\sigma) \odot [\sigma = \tau] && \text{(weighting)}\\
[\![C_1 \fatsemi C_2]\!](\sigma,\tau) &\triangleq \bigoplus_{\iota\in\Sigma} [\![C_1]\!](\sigma,\iota) \odot [\![C_2]\!](\iota,\tau) && \\
& && \text{(sequential composition)}\\
[\![\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}]\!](\sigma,\tau) &\triangleq [\![C_1]\!](\sigma,\tau) \;\oplus\; [\![C_2]\!](\sigma,\tau) && \text{(nondeterministic choice)}\\
[\![C^{\langle e,e'\rangle}]\!](\sigma,\tau) &\triangleq (\mathsf{lfp}\ X\colon\quad \Phi_{C,e,e'}(X))(\sigma,\tau) && \text{(iteration)}
\end{aligned}$$

where

$$\Phi_{C,e,e'}(X)(\sigma,\tau) \;=\; [\![e]\!](\sigma) \odot \left(\bigoplus_{\iota\in\Sigma} [\![C]\!](\sigma,\iota) \odot X(\iota,\tau)\right) \oplus [\![e']\!](\sigma) \odot [\sigma = \tau]$$

$\lhd$

**Figure 4.1:** Denotational semantics $[\![C]\!]\colon (\Sigma \times \Sigma) \to U$ of $\mathsf{wReg}$ programs, where $\mathcal{A} = \langle U,\, \oplus,\, \odot,\, \mathbb{0},\, \mathbb{1}\rangle$ is a semiring and the least fixed point is defined via point-wise extension of the natural order $\leq$ such that $f \leq f'$ iff $f(\sigma_1,\sigma_2) \leq f'(\sigma_1,\sigma_2)$ for all $\sigma,\sigma' \in \Sigma$.

indicates that the weight of transitioning from initial state $\sigma$ to final state $\tau$ after executing $x \coloneqq$ `nondet()` is $\mathbb{1}$ if $\sigma$ and $\tau$ differ only in the value of $x$, and $\mathbb{0}$ otherwise. This is achieved by treating $\bigoplus$ akin to an existential quantifier. Specifically, given $\sigma$, we consider all possible values that $x$ may take after the execution of $x \coloneqq$ `nondet()`.

**Assume/Weighting:**

The semantics for `assume` $\varphi$ indicates that the weight of transitioning from $\sigma$ to $\sigma$ is determined by the evaluation of $\varphi$ in $\sigma$: an initial state $\sigma$ is unchanged and has weight $\mathbb{1}$ if the guard holds, and $\mathbb{0}$ otherwise. If $\tau \neq \sigma$, then the weight of the transition is $\mathbb{0}$.

The intuition of the weighting statement in Batz et al. [24] is to weight arbitrary constant values $u \in U$, which does not generalize `assume` $\varphi$ (but only `assume true` and `assume false`). In our setting, weight can be any expression, so $\odot\, e$ is a proper generalization of the assume rule and is defined as

$$[\![\odot\, e]\!](\sigma,\tau) \;=\; [\![e]\!](\sigma) \odot [\sigma = \tau]\,.$$

Here, the weighting rule expresses that the weight of transitioning from $\sigma$ to itself after a weighting operation is determined by the weight $[\![e]\!](\sigma)$.

**Sequential Composition:**

The semantics for $C_1 \fatsemi C_2$ calculates the weight of transitioning from $\sigma$ to $\tau$ after executing a sequence of $C_1$ followed by $C_2$, considering all possible intermediate states $\sigma'$.

**Nondeterministic Choice:**

The semantics for $\{\, C_1 \,\} \square \{\, C_2 \,\}$ captures the weight of transitioning from $\sigma$ to $\tau$ after executing either $C_1$ or $C_2$, with the weight being the sum of the individual weights.

**Iteration:**

The recursive semantics of $C^{\langle e,e'\rangle}$ is captured by the equation $C^{\langle e,e'\rangle} = \left\{\, \odot\, e \fatsemi C \fatsemi C^{\langle e,e'\rangle} \,\right\} \square \{\, \odot\, e' \,\}$, which expresses the intuition that a loop either executes its body $C$ (weighted by condition $e$) and then recurses, or terminates (weighted by condition $e'$). By introducing a variable $X$ to stand for the recursive occurrence, we obtain the functional $\Phi_{C,e,e'}(X) = \{\, \odot\, e \fatsemi C \fatsemi X \,\} \square \{\, \odot\, e' \,\}$. This functional is Scott-continuous when $\Phi_{C,e,e'}$ is a total function (see Section 4.2.4 for well-definedness conditions). By Kleene's fixpoint theorem, the loop's semantics is then given by the least fixed point of $\Phi_{C,e,e'}$, computed as the supremum of the ascending chain beginning with the bottom element $\mathbb{0}$ (representing divergence). Each step in this chain corresponds to a finite unrolling of the loop, gradually approximating its full behavior as demonstrated in the following sequence:

$$\begin{aligned}
\Phi_{C,e,e'}(\mathbb{0})(\sigma,\tau) &= [\![\{\, \odot\, e \fatsemi \texttt{diverge} \,\} \square \{\, \odot\, e' \,\}]\!](\sigma,\tau)\\
\Phi^2_{C,e,e'}(\mathbb{0})(\sigma,\tau) &= [\![\{\, \odot\, e \fatsemi C \fatsemi \{\, \odot\, e \fatsemi \texttt{diverge} \,\} \square \{\, \odot\, e' \,\} \,\} \square \{\, \odot\, e' \,\}]\!](\sigma,\tau)
\end{aligned}$$

$$\Phi^3_{C,e,e'}(\mathbb{0})(\sigma,\tau) \;=\;$$
$$[\![\{\, \odot\, e \,⨾\, C \,⨾\, \{\, \odot\, e \,⨾\, C \,⨾\, \{\, \odot\, e \,⨾\, \texttt{diverge} \,\} \,\square\, \{\, \odot\, e' \,\} \,\} \,\square\, \{\, \odot\, e' \,\} \,\} \,\square\, \{\, \odot\, e' \,\}]\!](\sigma,\tau)$$

and so on, whose supremum is the least fixed point of $\Phi_{C,e,e'}$.

### 4.2.4 Well-definedness of the Denotational Semantics

We argue that the denotational semantics defined in Figure 4.1 is well-defined when $\Phi_{C,e,e'}(X)$ is a total function. This totality condition holds for any total semirings (such as Bool,Tropical), rendering our semantics more general than several others [73; 24; 17]. For partial semirings, extra caution is necessary as $\oplus$ may not always be well-defined. Hence:

1. We restrict the assignment $x \coloneqq \texttt{nondet()}$, Kleene's star $C^\star$ and nondeterministic choices $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$ to total semi-rings only.

2. We allow only nondeterministic choices of the form $\{\, \odot\, e \,⨾\, C_1 \,\} \,\square\, \{\, \odot\, e' \,⨾\, C_2 \,\}$ and loops $C^{\langle e,e' \rangle}$ where the expressions are compatible [74, Section A.3], that is, $[\![e]\!](\sigma) \oplus [\![e']\!](\sigma)$ is defined for any $\sigma \in \Sigma$.

Restricting to compatible expressions allows the use of syntactic constructs $\texttt{if}\ (\varphi)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}$ and the guarded loop $\texttt{while}\,(\,\varphi\,)\,\{\, C \,\}$ for *every* semiring. Additionally, the probabilistic choice $\{\, C_1 \,\}\ [\,p\,]\ \{\, C_2 \,\}$ remains well-defined for the partial semiring Prob. For the remainder of the chapter, we assume that programs are constructed in this manner, ensuring they are always well-defined. Proofs of well-definedness are in Appendix A.7.

## 4.3 Quantitative Weakest Pre for Weighted Programs

Although this is not our primary contribution, we will adapt the weakest preweighting calculus presented by Batz et al. [24, Section 4.3] to our framework, which allows for nondeterministic assignments and more generalized loops. The resulting calculus is a Dijkstra-style approach for formal reasoning about the value of a quantity $f \in \mathbb{A}$ after the execution of a *weighted* program. For simplicity, we will refer to this as the *quantitative weakest pre*, which serves as a direct generalization of the calculus we introduced in Section 3.3.

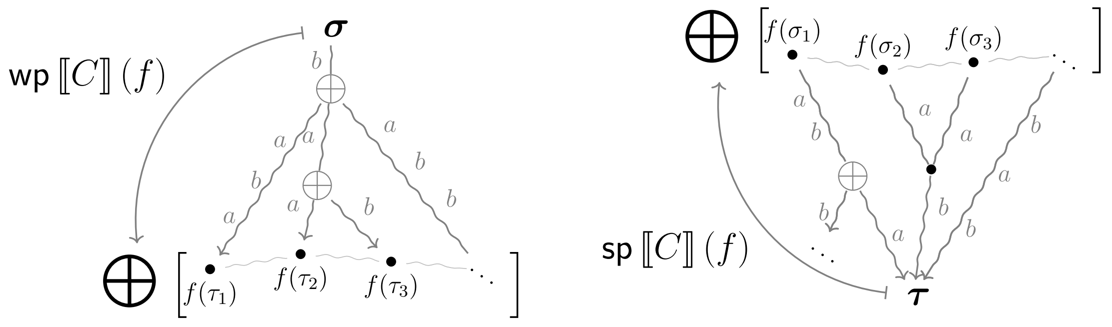
To achieve this, we generalize the *map perspective* of weakest preconditions to quantities. Recall that in Section 3.3, we extended Dijkstra's original calculus,

$$\begin{aligned} \text{from} \qquad \mathsf{wp}[\![C]\!]\colon \quad & (\Sigma \to \{0,1\}) \;\to\; (\Sigma \to \{0,1\}) \ , \\ \text{to} \qquad \mathsf{wp}[\![C]\!]\colon \quad & (\Sigma \to \mathbb{R}^{\pm\infty}) \;\to\; (\Sigma \to \mathbb{R}^{\pm\infty}) \ . \end{aligned}$$

i.e., from traditional pre and post*conditions* mapping states to Boolean values $0, 1$, to more general pre and post*quantities* mapping program states to extended real values in $\mathbb{R}^{\pm\infty}$. This generalization enables quantitative reasoning about program behavior beyond simple truth values. In our *weighted* setting, we employ a more general definition of quantities, as given in Definition 4.2.6, where $\mathsf{wp}\,[\![C]\!]\,(f) : \mathbb{A} \to \mathbb{A}$, with $\mathbb{A} = \Sigma \to U$, and $U$ being the set of values from an arbitrarily chosen semiring $\mathcal{A}$. Intuitively, we obtain a function $\mathsf{wp}[\![C]\!]$ that takes an initial state $\sigma$ as input, and for each path starting in $\sigma$ and terminating in some final state $\tau_i$, it computes the semiring product of all weights along the path, including the additional postweight $f(\tau_i)$ at the end. Finally, it returns the semiring sum of these values; see Figure 4.2a for a graphical representation.

If the program has no weights, and if $C$ terminates on input $\sigma$, then $\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)$ $\underline{\textit{anticipates}}$ the possible values that $f$ will take, evaluated in the final state that is reached after executing $C$ on $\sigma$. Subsequently, $\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)$ performs the semiring sum over these values, which could represent, for example, the supremum or infimum. This process intuitively subsumes both our quantitative weakest precondition and its liberal version, as defined in Section 3.3.

As pointed in Section 3.3.3, one of the main advantages of Dijkstra's calculus is that the weakest preconditions can be defined inductively based on the program structure, which allows for *compositional reasoning*. We will demonstrate how the same principles apply to our weighted setting.



**(a) Weighted Weakest Pre:** Given an initial state $\sigma$, $\mathsf{wp}\,[\![C]\!]\,(f)$ computes the semiring product of all weights along each path starting in $\sigma$ and terminating in some final state $\tau_i$, including the postquantity $f(\tau_i)$, and returns the semiring sum of these values.

**(b) Weighted Strongest Post:** Given a final state $\tau$, $\mathsf{sp}\,[\![C]\!]\,(f)$ computes the semiring product of all weights along each path starting from an initial state $\sigma_i$ that can reach $\tau$, including the prequantity $f(\sigma_i)$, and returns the semiring sum of these values.

**Definition 4.3.1** (Quantitative Weakest Pre for Weighted Programs)**.** *The* weakest pre transformer

$$\mathsf{wp}\colon\quad \mathsf{nGCL} \to (\mathbb{A} \to \mathbb{A})$$

*is defined inductively according to the rules in* Table 4.1*. We call the function*

$$\Phi_f(X) \;=\; [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X) \;\;,$$

*whose* least *fixed point defines the weakest pre* $\mathsf{wp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f)$*, the* $\mathsf{wp}$–characteristic function *(of* $C^{\langle e,e'\rangle}$ *with respect to* $f$*).* $\triangleleft$

Let us show for some of the rules how the quantitative weakest pre semantics can be developed and understood as a generalization or our calculus in Section 3.3.

**Assignment:**

The weighted weakest pre*condition* of an assignment is given by

$$\mathsf{wp}\,[\![x \coloneqq e]\!]\,(f) \;=\; f\,[x/e] \;\;,$$

| $C$ | $\mathsf{wp}\,[\![C]\!]\,(f)$ |
|---|---|
| $x \coloneqq e$ | $f\,[x/e]$ |
| $x \coloneqq \texttt{nondet()}$ | $\bigoplus_\alpha f\,[x/\alpha]$ |
| $\odot\, w$ | $w \odot f$ |
| $C_1 \fatsemi C_2$ | $\mathsf{wp}\,[\![C_1]\!]\,\big(\mathsf{wp}\,[\![C_2]\!]\,(f)\big)$ |
| $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$ | $\mathsf{wp}\,[\![C_1]\!]\,(f) \oplus \mathsf{wp}\,[\![C_2]\!]\,(f)$ |
| $C^{\langle e,e'\rangle}$ | $\mathsf{lfp}\ X\colon [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X)$ |

**Table 4.1:** Rules for the weakest pre transformer. Here, $\mathsf{lfp}\ f\colon \Phi(f)$ denotes the least fixed point of $\Phi$. While the rules for $x \coloneqq \texttt{nondet()}$ and $C^{\langle e,e'\rangle}$ are newly introduced, the remaining rules have been previously presented in [24, Table 2].

where $f\,[x/e]$ is the replacement of every occurrence of variable $x$ in the postquantity $f$ by the expression $e$. This rule is analogous to our quantitative weakest pre, as no weights are involved at all, so the only weight we have is given by the postquantity $f$ evaluated at the final state $\sigma\,[x \mapsto \sigma(e)]$ — the state obtained from $\sigma$ by updating variable $x$ to value $\sigma(e)$.

**Nondeterministic Assignment:**

The statement $x \coloneqq \texttt{nondet()}$ is analogous to $x \coloneqq e$, but without any restriction on the final value of $x$. Since the assignment is entirely nondeterministic, we cannot deterministically anticipate the final value of $x$. The weakest precondition is thus given by:

$$\mathsf{wp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(f) = \bigoplus_\alpha f\,[x/\alpha]\,.$$

Here, for each possible value $\alpha$ of $x$, we have a different anticipated quantity $f\,[x/\alpha]$. These quantities are aggregated via the semiring sum, reflecting the nondeterministic nature of the assignment.

**Assume/Weighting:**

In the assume statement $\texttt{assume}\ \varphi$, the weakest pre is given by:

$$\mathsf{wp}\,[\![\texttt{assume}\ \varphi]\!]\,(f) = [\varphi] \cdot f.$$

For predicates, this indicates that if the initial state satisfies $\varphi \wedge f$, then executing `assume` $\varphi$ ensures that the state will satisfy $f$. Quantitatively, `assume` $\varphi$ acts as a filter, preserving $f$ only if the initial state satisfies $\varphi$. This behavior is achieved using our semiring-parametrized Iverson bracket Definition 4.2.7.

In weighted programming, we generalize this concept to more general weighting functions $w$, yielding the rule:

$$\mathsf{wp}\,[\![\odot\, w]\!]\,(f) = w \odot f.$$

Here, $a \odot f$ represents the scaling of $f$ by the weight $w$. This generalizes the notion of filtering to accommodate various use cases, such as probabilities.

The apparent inversion between program syntax notation $\odot w$ and semantic denotation $w\odot$ serves a precise mathematical purpose, especially in non-commutative settings. When programs execute, they process states sequentially from program entry to exit—encountering a weight $w$ during execution effectively appends this value to the right end of the accumulated trace of weights, constituting a natural right-multiplication operation.

Conversely, weakest precondition calculus are backward-moving: they traverse programs in reverse, from postconditions toward preconditions. As our backward analysis encounters a weighting instruction, we must incorporate this weight at the beginning of our postquantity $f$, which may anticipate the effects of subsequent computations. We thus prepend $w$ to the current postquantity $f$, yielding a left-multiplication $w \odot f$ in the denotations.

**Sequential Composition:**

For sequential composition $C_1 \fatsemi C_2$, the quantitative weakest pre is computed via:

$$\mathsf{wp}\,[\![C_1 \fatsemi C_2]\!]\,(f) = \mathsf{wp}\,[\![C_1]\!]\,(\mathsf{wp}\,[\![C_2]\!]\,(f))\,.$$

This reflects the process where we first compute the weakest pre for $C_2$ with respect to $f$, resulting in $\mathsf{wp}\,[\![C_2]\!]\,(f)$. We then use this result as the input to compute the weakest pre for $C_1$, thereby anticipating the overall weight.

**Nondeterministic Choice:**

For the nondeterministic choice $\{\, C_1 \,\} \mathbin{\square} \{\, C_2 \,\}$, where either $C_1$ or $C_2$ may be executed, the weakest pre is given by the semiring sum of the weakest pre for $C_1$ and $C_2$:

$$\mathsf{wp}\,[\![\{\, C_1 \,\} \mathbin{\square} \{\, C_2 \,\}]\!]\,(f) = \mathsf{wp}\,[\![C_1]\!]\,(f) \oplus \mathsf{wp}\,[\![C_2]\!]\,(f)\,.$$

This captures the need to account for the possibility of either $C_1$ or $C_2$ being executed. The semiring sum combines the results, allowing us to compute suprema or infima, thereby encompassing our previous quantitative calculi defined in Section 3.3.

**Iteration:**

The weakest pre for the iteration $C^{\langle e,e'\rangle}$ is an extension to the one in Section 3.3, but generalised to arbitrary weights $e, e'$. The intended meaning of $C^{\langle e,e'\rangle}$ is to be equal to $\left\{\, \odot\, e \fatsemi\, C \fatsemi\, C^{\langle e,e'\rangle} \,\right\} \mathbin{\square} \{\, \odot\, e' \,\}$. Replacing the recursive instance of $C^{\langle e,e'\rangle}$ with $X$, we get $\Phi_{C,e,e'}(X)$, and so by Kleene's fixpoint theorem, the least fixed point corresponds to iterating on the least element $\mathbb{0}$, which yields an ascending chain of loop unrollings:

$$\begin{aligned}
\Phi_f(\mathbb{0}) &= \mathsf{wp}\,[\![\{\, \odot\, e \fatsemi\, \texttt{diverge} \,\} \mathbin{\square} \{\, \odot\, e' \,\}]\!]\,(f) \\
\Phi_f^2(\mathbb{0}) &= \mathsf{wp}\,[\![\{\, \odot\, e \fatsemi\, C \fatsemi\, \{\, \odot\, e \fatsemi\, \texttt{diverge} \,\} \mathbin{\square} \{\, \odot\, e' \,\} \,\} \mathbin{\square} \{\, \odot\, e' \,\}]\!]\,(f) \\
\Phi_f^3(\mathbb{0}) &= \\
&\mathsf{wp}\,[\![\{\, \odot\, e \fatsemi\, C \fatsemi\, \{\, \odot\, e \fatsemi\, C \fatsemi\, \{\, \odot\, e \fatsemi\, \texttt{diverge} \,\} \mathbin{\square} \{\, \odot\, e' \,\} \,\} \mathbin{\square} \{\, \odot\, e' \,\} \,\} \mathbin{\square} \{\, \odot\, e' \,\}]\!]\,(f)
\end{aligned}$$

which converge to the least fixed point of $\Phi_f(X) = [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X)$, yielding the rule $\mathsf{wp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \;=\; \big(\mathsf{lfp}\, X\colon [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X)\,\big)$

Let us show what $\mathsf{wp}$ computes semantically.

**Theorem 4.3.1** (Characterization of wp)**.** *For all programs $C \in$ wReg and final states $\tau \in \Sigma$, the following equality holds:*

$$\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \quad = \quad \bigoplus_{\tau \in \Sigma} \; [\![C]\!](\sigma, \tau) \odot f(\tau)\ .$$

The result above is a generalization of the characterization result for the quantitative weakest pre in Theorem 3.3.1 and Theorem 3.3.2, which can be obtained by choosing the semirings $\langle \mathbb{R}^{\pm\infty}$, max, min, $-\infty$, $+\infty\rangle$ respectively. In the context of probability distributions, the well known duality theorem of Kozen [13] can be seen as instantiation of Theorem 4.3.1 as well, by taking the semiring Prob. Indeed, the following can be proved as a corollary of Theorem 4.3.1.

**Corollary 4.3.1.1** (Kozen [13] Duality)**.** *For all programs $C$, probability distributions $\mu\colon \Sigma \to [0,1]$, and all functions $f \in \mathbb{A}$, we have*

$$\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) = \sum_{\tau \in \Sigma} \; [\![C]\!](\sigma, \tau) \cdot f(\tau).$$

To demonstrate the generality of our calculus, we illustrate in Table 4.2 that by applying different semirings, our calculus encompasses several existing weakest precondition calculi, including those presented in Section 3.3.

| **Calculus** | **Semiring** |
|---|---|
| Quantitative Weakest Preexpectation [14; 16][2] | $\langle \mathbb{R}^{\infty}_{\geq 0},\ +,\ \cdot,\ 0,\ 1\rangle$ |
| Weakest Precondition [64] | $\langle \{0,1\}, \vee, \wedge, 0, 1\rangle$ |
| Weakest Liberal Precondition [64] | $\langle \{0,1\}, \wedge, \vee, 1, 0\rangle$ |
| (Angelic) Quantitative Weakest Pre [17] | $\langle \mathbb{R}^{\pm\infty}, \max, \min, -\infty, +\infty\rangle$ |
| (Demonic) Quantitative Weakest Liberal Pre [17] | $\langle \mathbb{R}^{\pm\infty}, \min, \max, +\infty, -\infty\rangle$ |

**Table 4.2:** Existing weakest pre calculi subsumed via our quantitative weighted pre.

## 4.4 Quantitative Strongest Post for Weighted Programs

We now present our main contribution: A lifting of our quantitative strongest post calculus to weighted programs, which we will subsume both our quantitative strongest post and its liberal version.

To achieve this, we generalize the *map perspective* of strongest postconditions to quantities. Recall that in Section 3.4, we extended Dijkstra's original calculus,

$$\begin{aligned} \text{from} \qquad \mathsf{sp}[\![C]\!]\colon \quad & (\Sigma \to \{0,1\}) \;\to\; (\Sigma \to \{0,1\}) \;, \\ \text{to} \qquad \mathsf{sp}[\![C]\!]\colon \quad & (\Sigma \to \mathbb{R}^{\pm\infty}) \;\to\; (\Sigma \to \mathbb{R}^{\pm\infty}) \;. \end{aligned}$$

Similarly to how we did for weighted $\mathsf{wp}$, we now generalize the strongest post transformer to work with quantities over arbitrary semirings. Given a pre*quantity* $f\colon \Sigma \to U$, the weighted strongest post $\mathsf{sp}\,[\![C]\!]\,(f) : \Sigma \to U$ is a function that takes as input a *final* state $\tau$, determines all initial states $\sigma$ that can reach $\tau$ by executing $C$, evaluates the prequantity $f(\sigma)$ in each of these initial states, combines it with the weight of the execution path from $\sigma$ to $\tau$ using the semiring product $\odot$, and finally aggregates all these values using the semiring sum $\oplus$. See Figure 4.2b for a graphical representation. As a transformer, we obtain the following:

We define a novel *quantitative strongest post* transformer for $\mathsf{wReg}$.

**Definition 4.4.1** (Quantitative Strongest Post)**.** *The* strongest post transformer

$$\mathsf{sp}\colon \quad \mathsf{wReg} \to (\mathbb{A} \to \mathbb{A})$$

| $\boldsymbol{C}$ | $\mathsf{sp}\,[\![C]\!]\,(f)$ |
|---|---|
| $x \coloneqq e$ | $\bigoplus_\alpha f\,[x/\alpha] \odot [x = e\,[x/\alpha]]$ |
| $x \coloneqq \texttt{nondet()}$ | $\bigoplus_\alpha f\,[x/\alpha]$ |
| $\odot\, w$ | $f \odot w$ |
| $C_1 \fatsemi C_2$ | $\mathsf{sp}\,[\![C_2]\!]\,(\mathsf{sp}\,[\![C_1]\!]\,(f))$ |
| $\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}$ | $\mathsf{sp}\,[\![C_1]\!]\,(f) \oplus \mathsf{sp}\,[\![C_2]\!]\,(f)$ |
| $C^{\langle e, e' \rangle}$ | $\big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(f)$ |

**Table 4.3:** Rules for the strongest post transformer. Here, $\mathsf{lfp}\ f\colon \Phi(f)$ denotes the least fixed point of $\Phi$.

*is defined inductively according to the rules in* Table 4.3*. We call the function*

$$\Phi(\mathit{trnsf}) \;=\; \lambda f\colon \mathit{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!]\ ,$$

*whose* least *fixed point is used to define* $\mathsf{sp}\,[\![C^{\langle e, e' \rangle}]\!]\,(f)$*, the* $\mathsf{sp}$–characteristic function *(of* $C^{\langle e, e' \rangle}$*).* ◁

Let us show for some of the rules how the weighted strongest post semantics can be developed and understood analogously to our previously defined quantitative strongest post in Section 3.4.

**Assignment:**

The quantitative strongest post for an assignment $x \coloneqq e$ is given by:

$$\mathsf{sp}\,[\![x \coloneqq e]\!]\,(f) = \bigoplus_\alpha f\,[x/\alpha] \odot [x = e\,[x/\alpha]]\,.$$

This rule reflects that we need to consider all possible values $\alpha$ that $x$ could have had before the assignment and summing all evaluations of quantity $f$ under those possible $\alpha$.

**Nondeterministic Assignment:**

The nondeterministic assignment $x \coloneqq \texttt{nondet()}$ allows $x$ to take any possible value, without constraints on its initial state. Since the assignment does not depend on a specific initial value of $x$, any initial value is considered valid.

Importantly, the semantics of $x \coloneqq \texttt{nondet()}$ remain consistent whether the assignment is evaluated forward or backward. Consequently, the rule for the strongest postcondition is identical to that of the weakest precondition:

$$\mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(f) = \bigoplus_{\alpha} f\,[x/\alpha]\,.$$

**Assume/Weighting:**

In the assume statement, the strongest post is given by $[\varphi] \cdot f$, where $[\varphi]$ acts as a filter, nullifying states for which the predicate does not hold.

The weighting statement $\odot\, w$ extends the assume rule by allowing any weighting function $w$.

The strongest post for weighting involves scaling the initial quantity $f$ by the weight $w$, obtaining:

$$\mathsf{sp}\,[\![\odot\, w]\!]\,(f) = f \odot w.$$

Here, $f \odot w$ indicates that the initial quantity $f$ is scaled by the weight $w$, reflecting how the weighting modifies the state.

The order of the rule is particularly important in non-commutative semirings, where the sequence of operations cannot be interchanged without affecting the result. In strongest post calculi, the order of multiplication is reversed compared to the backward-moving weakest pre calculi. Specifically, in strongest post calculi, $f$ represents a prequantity that abstracts or summarizes the effects of preceding computations. As we move from the front to the back of a program, encountering a weighting by $w$ requires us to append $w$ to the current prequantity $f$, resulting in a right-multiplication $f \odot w$ in the denotations. This right-multiplication correctly captures the cumulative impact of the weighting on the initial quantity, preserving the intended semantics in a non-commutative context.

**Sequential Composition:**

For sequential composition, the quantitative strongest postcondition is given by:

$$\mathsf{sp}\,[\![C_1 \,⨾\, C_2]\!]\,(f) = \mathsf{sp}\,[\![C_2]\!]\,(\mathsf{sp}\,[\![C_1]\!]\,(f))\,.$$

This process begins by computing the strongest postcondition for the initial command $C_1$, and then uses the resulting postcondition as the input for evaluating the second command $C_2$. The expression $\mathsf{sp}\,[\![C_1]\!]\,(f)$ represents the possible states and associated weights after executing $C_1$. These states serve as the starting point for $C_2$, ensuring that the final weights are the combined effect of executing both commands in sequence.

**Nondeterministic Choice:**

For the nondeterministic choice $\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}$, where either $C_1$ or $C_2$ may be executed, the strongest post is given by the semiring sum of the strongest post for $C_1$ and $C_2$:

$$\mathsf{sp}\,[\![\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}]\!]\,(f) = \mathsf{sp}\,[\![C_1]\!]\,(f) \oplus \mathsf{sp}\,[\![C_2]\!]\,(f)\,.$$

This reflects the fact that either command might be executed, and therefore the overall strongest post must account for the outcomes of both possibilities. The semiring sum aggregates the weights in different ways, including the suprema or infima of the results. This approach thus generalizes and subsumes the methods used in our previous calculi in Section 3.4.

**Iteration:**

The strongest post for the iteration $C^{\langle e,e'\rangle}$ is an extension to the one in Section 3.4, but generalised to arbitrary weights $e, e'$ instead of predicates. It is thus obtained via loop unrollings

$$\begin{aligned}
\Psi_f(\mathbb{0}) \odot [\![e']\!] &= \mathsf{sp}\,[\![\{\, \odot\, e \,⨾\, \texttt{diverge} \,\} \,\Box\, \{\, \odot\, e' \,\}]\!]\,(f) \\
\Psi_f^2(\mathbb{0}) \odot [\![e']\!] &= \mathsf{sp}\,[\![\{\, \odot\, e \,⨾\, C \,⨾\, \{\, \odot\, e \,⨾\, \texttt{diverge} \,\} \,\Box\, \{\, \odot\, e' \,\} \,\} \,\Box\, \{\, \odot\, e' \,\}]\!]\,(f)
\end{aligned}$$

$$\Psi_f^3(\mathbb{0}) \odot [\![e']\!] = .$$
$$\mathsf{sp}\,[\![\{\,\odot e\,⨟\, C\,⨟\,\{\,\odot e\,⨟\, C\,⨟\,\{\,\odot e\,⨟\,\mathtt{diverge}\,\}\,\square\,\{\,\odot e'\,\}\,\}\,\square\,\{\,\odot e'\,\}\,\}\,\square\,\{\,\odot e'\,\}]\!]\,(f)$$

Similarly to our previous strongest post calculus defined in Section 3.4, it may be tempting to conclude that the loop unrollings converge to the least fixed point of $S_f(X) = f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])$, yielding the rule $\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) = \big(\mathsf{lfp}\ X\colon f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])\big) \odot [\![e']\!]$. However, this would be wrong in the context of partial semirings. In fact, for partial semirings, the operation $\oplus$ is not always defined for arbitrary elements, which means the fixed point computation may encounter undefined additions during intermediate steps, that we would not encounter if we were to weight by $e'$ each addend.

Let us illustrate this with an example.

**Example 4.4.1.** *Let us consider the semiring* $\mathsf{Prob} = \langle [0,1], +, \cdot, 0, 1\rangle$ *of probabilities. The addition operation is only defined for values in* $[0,1]$. *Consider the program* $C_{\mathit{skiploop}} = \mathtt{skip}^{\langle 0.5,0.5\rangle}$. *If we were to compute* $\mathsf{sp}\,[\![C_{\mathit{skiploop}}]\!]\,(\mathbf{1}_{x=0})$ *via* $(\mathsf{lfp}\ X\colon \mathbf{1}_{x=0} \oplus \mathsf{sp}\,[\![\mathtt{skip}]\!]\,(X \odot [\![e]\!])) \odot [\![e']\!]$, *we would rely on Kleene's Fixed Point Theorem, obtaining the following iterates:*

$$\begin{aligned} S_{\mathbf{1}_{x=0}}(0) &= \mathbf{1}_{x=0} \\ S^2_{\mathbf{1}_{x=0}}(0) &= \mathbf{1}_{x=0} \oplus \mathbf{1}_{x=0} \odot 0.5 = \textit{undefined} \end{aligned}$$

◁

The previous example demonstrates that a naive approach to defining the loop rule would not work. However, this does not preclude the possibility of defining the strongest post in a meaningful way. Indeed, we would expect that the strongest post of $C_{\text{skiploop}}$ should coincide with that of $\mathtt{skip}$, which is the prequantity $\mathbf{1}_{x=0}$. To achieve this, we reformulate the loop rule as the least solution of the following recursive equation:

$$\mathsf{sp}[\![C^{\langle e,e'\rangle}]\!]$$

$$
\begin{aligned}
&= \mathsf{sp}[\![\left\{\, \odot\, e \fatsemi\, C \fatsemi\, C^{\langle e,e'\rangle} \right\} \,\square\, \{\, \odot\, e' \,\}]\!] \\
&= \lambda f\colon \mathsf{sp}\,[\![\odot\, e \fatsemi\, C \fatsemi\, C^{\langle e,e'\rangle}]\!]\,(f) \oplus \mathsf{sp}\,[\![\odot\, e']\!]\,(f) \\
&= \lambda f\colon \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!]\ ,
\end{aligned}
$$

which corresponds to the least fixpoint of the higher order function

$$\Psi(\mathsf{trnsf}) \;=\; \lambda f\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!]\ .$$

We thus obtain the rule:

$$\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \;=\; \big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(f)\ .$$

**Example 4.4.2.** *Let us consider again the semiring* $\mathit{Prob} = \langle [0,1], +, \cdot, 0, 1\rangle$ *of probabilities, and the program* $C_{\mathit{skiploop}} = \mathtt{skip}^{\langle 0.5, 0.5\rangle}$. *We compute* $\mathsf{sp}\,[\![C_{\mathit{skiploop}}]\!]\,(\mathbf{1}_{x=0}) = \big(\mathsf{lfp}\ \mathit{trnsf}\colon \Psi(\mathit{trnsf})\big)(\mathbf{1}_{x=0})$, *and relying on Kleene's Fixed Point Theorem, obtain the following iterates:*

$$
\begin{aligned}
\Psi(\lambda g\colon 0) &= \lambda f\colon f \odot 0.5 \\
\Psi^2(\lambda g\colon 0) &= \lambda f\colon f \odot 0.5 \odot 0.5 \oplus f \odot 0.5 \\
&\ \ \vdots \\
\Psi^n(\lambda g\colon 0) &= \lambda f\colon \bigoplus_{i=1}^{n} f \odot 0.5^i
\end{aligned}
$$

*This allows us to conclude that:*

$$
\begin{aligned}
\mathsf{sp}\,[\![C_{\mathit{skiploop}}]\!]\,(\mathbf{1}_{x=0}) &= \big(\mathsf{lfp}\ \mathit{trnsf}\colon \Psi(\mathit{trnsf})\big)(\mathbf{1}_{x=0}) \\
&= \bigoplus_{i=1}^{\infty} \mathbf{1}_{x=0} \odot 0.5^i \\
&= \mathbf{1}_{x=0}\ .
\end{aligned}
$$

$\triangleleft$

In general, for total semirings, the simplified loop rule is also correct, as shown in the following theorem.

**Theorem 4.4.1** (Loop rule for total semirings)**.** *For all programs $C \in \mathsf{wReg}$, if the ambient semiring is a total semiring, the simplified loop rule:*

$$\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \quad = \quad \big(\mathsf{lfp}\ X\colon f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])\,\big) \odot [\![e']\!]$$

*holds for all $f \in \mathbb{A}$.*

The order $X \odot [\![e]\!]$ rather than $[\![e]\!] \odot X$ in the fixed point equation reflects the operational semantics of the loop: we first compute the accumulated effect $X$ from previous iterations, and then scale it by the guard condition $[\![e]\!]$ to determine which states continue in the iteration. This contrasts with $\mathsf{wp}$, where the order is reversed ($[\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X)$) because weakest pre are backward moving from postquantities, whereas our strongest post pushes prequantities forward.

Let us show what $\mathsf{sp}$ computes semantically.

**Theorem 4.4.2** (Characterization of $\mathsf{sp}$)**.** *For all programs $C \in \mathsf{wReg}$ and final states $\tau \in \Sigma$,*

$$\mathsf{sp}\,[\![C]\!]\,(\mu)\,(\tau) \quad = \quad \bigoplus_{\sigma \in \Sigma} \mu(\sigma) \odot [\![C]\!](\sigma, \tau)\ .$$

Theorem 4.4.2 guarantees the correct behavior of $\mathsf{sp}$ by asserting that it appropriately maps initial weighted quantities to final weighted quantities, including probability distributions and program sets of states. Our characterization of $\mathsf{sp}$ is different compared to the one disproven by [95, p. 135]. The latter focuses on identifying the most precise assertion for the triples defined in [95, p. 124].

In particular, Table 4.4 shows that by instantiating our calculus with different semirings we subsume several existing strongest post calculi, such as those introduced in Section 3.4. Additionally, similarly to [24, Table 1], weighted strongest post can handle optimization and combinatorial problems as well, with the main difference to be our calculus moving forward instead of backward.

| **Calculus** | **Semiring** |
|---|---|
| Strongest Postcondition [64] | $\langle \{0,1\}, \vee, \wedge, 0, 1 \rangle$ |
| Strongest Liberal Postcondition [17] | $\langle \{0,1\}, \wedge, \vee, 1, 0 \rangle$ |
| Quantitative Strongest Post [17] | $\langle \mathbb{R}^{\pm\infty}, \max, \min, -\infty, +\infty \rangle$ |
| Quantitative Strongest Liberal Post [17] | $\langle \mathbb{R}^{\pm\infty}, \min, \max, +\infty, -\infty \rangle$ |

**Table 4.4:** Existing strongest post calculi subsumed via our quantitative strongest post.

## 4.5 Expressivity

In the preceding sections, we characterized our quantitative weighted transformers: wp and sp. In this section, we aim to illustrate the expressive capabilities of the calculus by demonstrating that it subsumes several other logics and calculi.

### 4.5.1 An Overview of Several Hoare-Like Logics

We illustrate how several program logics, namely partial correctness, angelic total correctness, partial incorrectness, and total incorrectness (according to the terminology in [17]), can be represented via classical predicate transformers, see Table 4.5. As a byproduct, they are subsumed by our weighted transformers as well.

### 4.5.2 Disproving Hoare-Like Triples

Reasoning about predicate transformers is useful to define new triples that are not expressible in classical Hoare-like logics. For example, we can semantically define new triples by falsifying the triples of Table 4.5, see Table 4.6.

Let us provide an intuition for the falsifying triples in Table 4.6.

- $\not\models_{\mathrm{pc}} \{ P \}\ C\ \{ Q \}$: there is some state in $P$ that can terminate in $\neg Q$, and

| **Logic** | **Syntax** | **Semantics** |
|---|---|---|
| Hoare Logic (partial correctness) | $\models_{\text{pc}} \{\, P \,\}\; C \;\{\, Q \,\}$ | $P \subseteq \mathsf{wlp}[\![C]\!]\,(Q)$ |
| Lisbon Logic (angelic total correctness) | $\models_{\text{atc}} \{\, P \,\}\; C \;\{\, Q \,\}$ | $P \subseteq \mathsf{wp}\,[\![C]\!]\,(Q)$ |
| Partial Incorrectness Logic | $\models_{\text{pi}} [\, P \,]\; C \;[\, Q \,]$ | $Q \subseteq \mathsf{slp}[\![C]\!]\,(P)$ |
| Incorrectness Logic/Reverse Hoare Logic | $\models_{\text{ti}} [\, P \,]\; C \;[\, Q \,]$ | $Q \subseteq \mathsf{sp}\,[\![C]\!]\,(P)$ |

**Table 4.5:** Partial and total (in)correctness using classical predicate transformers, according to the terminology of [17]. Our weighted wp and sp subsume these logics as well.

| **Syntax** | **Semantics** |
|---|---|
| $\not\models_{\text{pc}} \{\, P \,\}\; C \;\{\, Q \,\}$ | $P \cap \mathsf{wp}\,[\![C]\!]\,(\neg Q) \neq \emptyset$ |
| $\not\models_{\text{atc}} \{\, P \,\}\; C \;\{\, Q \,\}$ | $P \cap \mathsf{wlp}[\![C]\!]\,(\neg Q) \neq \emptyset$ |
| $\not\models_{\text{pi}} [\, P \,]\; C \;[\, Q \,]$ | $Q \cap \mathsf{sp}\,[\![C]\!]\,(\neg P) \neq \emptyset$ |
| $\not\models_{\text{ti}} [\, P \,]\; C \;[\, Q \,]$ | $Q \cap \mathsf{slp}[\![C]\!]\,(\neg P) \neq \emptyset$ |

**Table 4.6:** Disproving partial and total (in)correctness using classical predicate transformers. Non-liberal transformers can be expressed via liberal transformers and vice versa by duality [17, Section 5.3]

hence it is false that every state in $P$ terminates only in $Q$ (if it terminates at all)

- $\not\models_{\text{atc}} \{\, P \,\}\; C \;\{\, Q \,\}$: there is some state in $P$ that terminates only in $\neg Q$ (if it terminates at all), and hence it is false that every state in $P$ can terminate in $Q$
- $\not\models_{\text{pi}} [\, P \,]\; C \;[\, Q \,]$: there is some state in $Q$ that is reachable from $\neg P$, and hence it is false that every state in $Q$ is reachable only from $P$
- $\not\models_{\text{ti}} [\, P \,]\; C \;[\, Q \,]$: there is some state in $Q$ that is reachable only from $\neg P$ (if it is reachable at all), and hence it is false that every state in $Q$ is reachable from $P$

It remains to define program logics for the newly defined falsifying triples. To this end, one can prove that the existing program logics are actually falsifying program logics for other triples. More precisely:

**Theorem 4.5.1** (Falsifying correctness triples via correctness triples)**.**

$$\begin{aligned}
\models_{\text{pc}} \{\,P\,\}\ C\ \{\,Q\,\} \quad &\text{iff} \quad \forall \sigma \in P\colon\ \not\models_{\text{atc}} \{\,\{\sigma\}\,\}\ C\ \{\,\neg Q\,\} \\
\models_{\text{atc}} \{\,P\,\}\ C\ \{\,Q\,\} \quad &\text{iff} \quad \forall \sigma \in P\colon\ \not\models_{\text{pc}} \{\,\{\sigma\}\,\}\ C\ \{\,\neg Q\,\} \\
\models_{\text{pi}} [\,P\,]\ C\ [\,Q\,] \quad &\text{iff} \quad \forall \sigma \in Q\colon\ \not\models_{\text{ti}} [\,\neg P\,]\ C\ [\,\{\sigma\}\,] \\
\models_{\text{ti}} [\,P\,]\ C\ [\,Q\,] \quad &\text{iff} \quad \forall \sigma \in Q\colon\ \not\models_{\text{pi}} [\,\neg P\,]\ C\ [\,\{\sigma\}\,]
\end{aligned}$$

An intuition for Theorem 4.5.1 is as follows.

- $\models_{\text{pc}} \{\,P\,\}\ C\ \{\,Q\,\}$: every state in $P$ can only terminate in $Q$ (if it terminates at all), and hence by starting on any of those state it is false that it can terminate in $\neg Q$
- $\models_{\text{atc}} \{\,P\,\}\ C\ \{\,Q\,\}$: every state in $P$ can terminate in $Q$, and hence by starting on any of those states it is false that it can terminates only in $\neg Q$ (if it terminates at all)
- $\models_{\text{pi}} [\,P\,]\ C\ [\,Q\,]$: every state in $Q$ is reachable only from $P$, and hence from any of those states it is false that it is reachable from $\neg P$
- $\models_{\text{ti}} [\,P\,]\ C\ [\,Q\,]$: every state in $Q$ is reachable from $P$, and hence from any of those states it is false that it is reachable only from $\neg P$

Theorem 4.5.1 not only demonstrates that existing program logics can generate proofs to falsify other triples but also establishes a crucial "if and only if" relationship. This indicates that not only the current logics are sound, but they are *complete* as well: the existence of an invalid triple implies the presence of a corresponding valid triple that renders the original one invalid. Restating Theorem 4.5.1 from a negative perspective as below might make it more clear how to practically falsify triples.

**Corollary 4.5.1.1.**

$$\begin{aligned}
\not\models_{\text{pc}} \{\, P \,\}\; C \;\{\, Q \,\} \quad &\text{iff} \quad \exists \sigma \in P\colon\; \models_{\text{atc}} \{\, \{\sigma\} \,\}\; C \;\{\, \neg Q \,\} \\
\not\models_{\text{atc}} \{\, P \,\}\; C \;\{\, Q \,\} \quad &\text{iff} \quad \exists \sigma \in P\colon\; \models_{\text{pc}} \{\, \{\sigma\} \,\}\; C \;\{\, \neg Q \,\} \\
\not\models_{\text{pi}} [\, P \,]\; C \;[\, Q \,] \quad &\text{iff} \quad \exists \sigma \in Q\colon\; \models_{\text{ti}} [\, \neg P \,]\; C \;[\, \{\sigma\} \,] \\
\not\models_{\text{ti}} [\, P \,]\; C \;[\, Q \,] \quad &\text{iff} \quad \exists \sigma \in Q\colon\; \models_{\text{pi}} [\, \neg P \,]\; C \;[\, \{\sigma\} \,]
\end{aligned}$$

As highlighted by Zhang and Kaminski [17, p. 20, “Other Triples”], the use of the terms “correctness” and “incorrectness” in naming conventions may be imprecise. Correctness triples can be seen as ∀-properties over preconditions, whereas incorrectness triples exhibit characteristics of ∀-properties over post-conditions. Furthermore, it is noteworthy that the falsification of such ∀-triples can be interpreted as ∃-triples, a result that aligns with the expectation that disproving these properties involves finding at least one counterexample. This perspective concurs with the observation made by Cousot [93, Logic 23] that Incorrectness Logic provides sufficient (though not necessary) conditions to falsify partial correctness triples, thereby demonstrating its greater-than-needed power. Let us show how to practically falsify triples.

**Example 4.5.1** (Backward-Moving Assignment Rule for (Total) Incorrectness Logic)**.** *Consider the triple $\models_{\text{ti}} [\, y = 42 \,]\; x \coloneqq 42 \;[\, y = x \,]$, obtained by taking as precondition the syntactic replacement of $x = 42$ from the post. As shown in [6] with a counterexample, this is not valid. We can prove it by computing a partial incorrectness triple with precondition $y \neq 42$.*

*Using the rules defined in [17, Table 2, Column 2], we have:*

$$\models_{\text{pi}} [\, y \neq 42 \,]\; x \coloneqq 42 \;[\, y \neq 42 \lor x \neq 42 \,]$$

*This post clearly contains at least one state with $y = x$ (e.g., take a state where $\sigma(x) = \sigma(y) = 0$), which implies $\not\models_{\text{ti}} [\, y = 42 \,]\; x \coloneqq 42 \;[\, y = x \,]$ (by Corollary 4.5.1.1).* ◁

## 4.6 Semantics of Nontermination and Unreachability

As discussed in Ascari et al. [78, Section 5.4], we also show how existing triples capture properties such as must-nontermination, may-termination, unreachability, and reachability, but within our setting. Our initial focus is on illustrating ∀-properties, see Table 4.7. It is noteworthy that the transition from partial

| **Triple** | **Semantics** | **Property** |
|---|---|---|
| $\models_{\text{pc}} \{\, P \,\}\ C\ \{\, \mathsf{false} \,\}$ | $\forall \sigma \in P\colon\ \nexists \tau\colon \tau \in [\![C]\!](\sigma)$ | Must-Nontermination |
| $\models_{\text{atc}} \{\, P \,\}\ C\ \{\, \mathsf{true} \,\}$ | $\forall \sigma \in P\colon \exists \tau\colon \tau \in [\![C]\!](\sigma)$ | May-Termination |
| $\models_{\text{pi}} [\, \mathsf{false} \,]\ C\ [\, Q \,]$ | $\forall \tau \in Q\colon\ \nexists \sigma\colon \tau \in [\![C]\!](\sigma)$ | Unreachability |
| $\models_{\text{ti}} [\, \mathsf{true} \,]\ C\ [\, Q \,]$ | $\forall \tau \in Q\colon \exists \sigma\colon \tau \in [\![C]\!](\sigma)$ | Reachability |

**Table 4.7:** ∀-properties on nontermination and unreachability.

(in)correctness to total (in)correctness involves the negation of the properties under consideration. Specifically, the negation of may-termination corresponds to must-nontermination, and unreachability is the negation of reachability.

Another useful perspective that has been observed by many [78; 93; 133], is to view reachability as the may-termination of the reverse semantics, while unreachability can be conceptualized as its must-nontermination.

By examining their falsification, we derive their dual counterparts, characterized as ∃-properties, see Table 4.8.

| **Triple** | **Semantics** | **Property** |
|---|---|---|
| $\not\models_{\text{pc}} \{\, P \,\}\ C\ \{\, \mathsf{false} \,\}$ | $\exists \sigma \in P\colon \exists \tau\colon \tau \in [\![C]\!](\sigma)$ | May-Termination |
| $\not\models_{\text{atc}} \{\, P \,\}\ C\ \{\, \mathsf{true} \,\}$ | $\exists \sigma \in P\colon\ \nexists \tau\colon \tau \in [\![C]\!](\sigma)$ | Must-Nontermination |
| $\not\models_{\text{pi}} [\, \mathsf{false} \,]\ C\ [\, Q \,]$ | $\exists \tau \in Q\colon \exists \sigma\colon \tau \in [\![C]\!](\sigma)$ | Reachability |
| $\not\models_{\text{ti}} [\, \mathsf{true} \,]\ C\ [\, Q \,]$ | $\exists \tau \in Q\colon\ \nexists \sigma\colon \tau \in [\![C]\!](\sigma)$ | Unreachability |

**Table 4.8:** ∃-properties on nontermination and unreachability.

In the preceding sections, we have demonstrated the capabilities and strengths of our framework, showcasing its effectiveness for proving a variety of program properties. Despite these achievements, our framework has certain limitations that should be acknowledged. Notably, we are unable to prove must-termination properties—a concept closely related to demonic total correctness, where a program is guaranteed to terminate for all possible non-deterministic choices. Beyond must-termination, our framework faces challenges with several other significant properties of interest in program verification. These properties, which remain beyond the scope of our current approach, are systematically categorized and presented in Table 4.9. The table provides a comprehensive overview of these limitations, clarifying the boundaries of our framework's applicability while identifying existing individual logics that can successfully prove each of these properties that lie beyond our approach.

| **Triple** | **Property** |
|---|---|
| $\models_{\text{apc}} \{\, P \,\}\ C\ \{\, \mathsf{false} \,\}$ | May-Nontermination |
| $\models_{\text{dtc}} \{\, P \,\}\ C\ \{\, \mathsf{true} \,\}$ | Must-Termination |
| $\not\models_{\text{apc}} \{\, P \,\}\ C\ \{\, \mathsf{false} \,\}$ | Must-Termination |
| $\not\models_{\text{dtc}} \{\, P \,\}\ C\ \{\, \mathsf{true} \,\}$ | May-Nontermination |

**Table 4.9:** Nontermination and unreachability.

The reason our logics are unable to express scenarios where a program might not terminate or must terminate stems from the underlying program semantics, where looping is defined via a least fixed point and only represents an accumulation of the terminating traces. As a result, we cannot identify whether additional nonterminating traces exist, since, e.g., $[\![\{\, \texttt{while}\,(\,\mathsf{true}\,)\,\{\, \texttt{skip} \,\}\,\} \,\square\, \{\, \texttt{skip} \,\}]\!](\sigma) = [\![\texttt{skip}]\!](\sigma)$ for all $\sigma$. The semantics could be adjusted to also indicate whether nontermination is possible (e.g., see Cousot and Cousot [134]; Cousot [93]), but we leave a complete exploration for future work. However, in the following section, we will briefly investigate how nontermination analysis fits into our framework.

### 4.6.1 Proving may nontermination

Raad et al. [81] defined a sound and complete proof system to prove may nontermination, by combining (angelic) total correctness and total incorrectness logics. We start by comparing the nontermination program logics developed in Raad et al. [81] to show how our framework could not only subsume and prove every rule, but we also define several novel interpretations of nontermination.

Raad et al. [81] introduce a triple notation $\models_{\text{unter}} \{\, P \,\}\ C^{\star}\ \{\, \infty \,\}$ to formalize may-divergence: for any state satisfying precondition $P$, there exists at least one diverging execution trace of $C^{\star}$. We will refer this to may nontermination, since here we do not require all paths to diverge. We observe that this definition can be expressed in our framework as an angelic partial correctness triple $\models_{\text{apc}} \{\, P \,\}\ C^{\star}\ \{\, \mathsf{false} \,\}$. Using our terminology, their fundamental rule can be formulated as:

$$\frac{\models_{\text{atc}} \{\, P \,\}\ C\ \{\, P \,\}}{\forall \sigma \in P \colon [\![ C^{\star} ]\!](\sigma) \text{ may diverge}}$$

Whilst [81, Section 1, "Formal Interpretation of Divergent Triples"] focuses on a stronger interpretation of triples where $\models_{\text{atc}} \{\, P \,\}\ C\ \{\, \infty \,\}$ means *every* state $\sigma \in P$ have *at least* a diverging trace, our framework allows to express a novel interpretation as well, namely *must divergence*. Unlike *may divergence*, *must divergence* asserts that all traces originating from a given initial state must diverge. We highlight the inadequacy of $C^{\star}$ due to its semantics implicitly assuming that must divergence never happen. Consequently, our subsequent exploration will revolve around `while` $(\, \varphi \,)\, \{\, C \,\}$, and we will present rules for both interpretations, see Figure 4.3.

$$\frac{\models_{\text{atc}} \{\, P \,\}\; C \;\{\, P \,\} \qquad P \subseteq \varphi}{\forall \sigma \in P \colon [\![\texttt{while}\,(\,\varphi\,)\,\{\, C \,\}]\!](\sigma) \text{ may diverge}}$$

$$\frac{\models_{\text{pc}} \{\, P \,\}\; C \;\{\, P \,\} \qquad P \subseteq \varphi}{\forall \sigma \in P \colon [\![\texttt{while}\,(\,\varphi\,)\,\{\, C \,\}]\!](\sigma) \text{ must diverge}}$$

**Figure 4.3:** Nontermination rules for the $\texttt{while}\,(\,\varphi\,)\,\{\, C \,\}$ construct using existing program logics.

The duality between angelic total correctness and partial correctness is influenced by the choices made in our interpretation of nondeterminism, which bears resemblance to the one highlighted in [17].

### 4.6.2 More on angelic partial correctness and demonic total correctness

As pointed in Table 4.9, angelic partial correctness and demonic total correctness have a key role in proving may-nontermination and must-termination. It is thus surprising that [81] chose to combine (angelic) total correctness and total incorrectness logics for their sound and complete proof system that allows to prove may-nontermination.

In this section, we show how a standard angelic partial correctness proof system relates with the rules in [81]. We consider guarded imperative languages with nondeterministic choices (i.e., with while constructs instead of Kleene star), and the rules for angelic partial correctness as analogous to those for standard partial correctness, except for the nondeterministic choice [16, Definition 4.5]. In particular, it is well known that by coinduction, the following rule holds:

$$\frac{\models_{\text{apc}} \{\, P \wedge \varphi \,\}\; C \;\{\, P \,\}}{\models_{\text{apc}} \{\, P \,\}\; \texttt{while}\,(\,\varphi\,)\,\{\, C \,\}\; \{\, \neg\varphi \wedge P \,\}}$$

We shall observe that angelic partial correctness is a complete proof system (for guarded imperative languages), and this already means that every may-nontermination triple can be proved. However, let us show how we can derive simpler rules (analogous to those in [81]) without the need to add explicit rules for may-nontermination.

**Theorem 4.6.1.** *The following rules are valid in angelic partial correctness logic:*

$$\frac{\models_{\text{apc}} \{\, P \,\}\; C_1 \;\{\, \mathsf{false} \,\}}{\models_{\text{apc}} \{\, P \,\}\; C_1 \fatsemi C_2 \;\{\, \mathsf{false} \,\}}$$

$$\frac{\models_{\text{apc}} \{\, P \,\}\; C_1 \;\{\, Q \,\} \qquad \models_{\text{apc}} \{\, Q \,\}\; C_2 \;\{\, \mathsf{false} \,\}}{\models_{\text{apc}} \{\, P \,\}\; C_1 \fatsemi C_2 \;\{\, \mathsf{false} \,\}}$$

$$\frac{\models_{\text{apc}} \{\, P \,\}\; C_i \;\{\, \mathsf{false} \,\} \;\; \textit{for some } i \in \{1,2\}}{\models_{\text{apc}} \{\, P \,\}\; \{\, C_1 \,\} \,\square\, \{\, C_2 \,\} \;\{\, \mathsf{false} \,\}}$$

$$\frac{\models_{\text{apc}} \{\, P \wedge \varphi \,\}\; C \;\{\, P \wedge \varphi \,\}}{\models_{\text{apc}} \{\, P \wedge \varphi \,\}\; \mathtt{while}\,(\,\varphi\,)\,\{\, C \,\} \;\{\, \mathsf{false} \,\}}$$

The rules above resemble to those in [81], but again we stress that here we are not developing a new complex logic. It is also easy to show that the loop rule for while loops in [81] can be very easily proved:

$$\frac{\dfrac{\models_{\text{atc}} \{\, P \wedge \varphi \,\}\; C \;\{\, P \wedge \varphi \,\}}{\models_{\text{apc}} \{\, P \wedge \varphi \,\}\; C \;\{\, P \wedge \varphi \,\}}}{\models_{\text{apc}} \{\, P \wedge \varphi \,\}\; \mathtt{while}\,(\,\varphi\,)\,\{\, C \,\} \;\{\, \mathsf{false} \,\}}$$

### 4.6.3 Nontermination and Unreachability

It is worth noting that in all rules mentioned in the previous section, we were concerned with correctness triples rather than incorrectness ones. This emphasis is due to our focus on the termination of the forward semantics. An analogous rule for total incorrectness triples would facilitate the identification of nonterminating states in the backward semantics:

$$\frac{\models_{\text{ti}} [\, P \,]\; C \;[\, P \,]}{\exists \sigma \in P \colon [\![ C^{\star} ]\!]^{-1}(\sigma) \text{ may diverge}}$$

The premise $\models_{\text{ti}} [\,P\,]\; C\; [\,P\,]$ ensures that every final state in $P$ is reachable by executing $C$ on some initial state in $P$. In other words, for any state we pick in $P$, we can find a backward trace that leads to another state in $P$, by reversing $C$. It is straightforward to show by induction that we can continue the backward trace from $P$ to $P$ as many times as we wish (i.e., we can reach $P$ from $P$ in $0, 1, 2, 3, \ldots$ steps) by just executing $C$ backwards (i.e., $[\![C^\star]\!]^{-1}(\sigma)$). The rule can be used in the context of program inversion to assess whether one could compute the pre-image by simply executing the inverted program.

The correlation between nontermination and unreachability, as highlighted in [17], may lead one to question whether proving states as unreachable is related to demonstrating nontermination. However, when considering backward semantics, a single nonterminating trace does not provide enough information to establish unreachability. It is essential for all backward traces to be nonterminating, aligning with the concept of must-termination in backward semantics, precisely corresponding to what is conventionally meant by unreachability. This insight strengthens the connection described in [17], where their dualities between nontermination and unreachability arise from the resolution of nondeterministic choices. In other words, when [17] refers to nontermination, they essentially mean must-nontermination.

We conclude by examining whether backward must-nontermination rules would be useful. For $C^\star$, we previously excluded this from forward must-nontermination analysis because arbitrary iterations always allow terminating executions. The same limitation applies to backward reasoning: since $C^\star$ can execute zero times, there is always a terminating execution path, making backward must-nontermination impossible. For while loops $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$, the analysis is equally trivial: a potential final state $\tau$ where $\tau \models \varphi$ cannot be reached by normal termination, while a state where $\tau \models \neg\varphi$ is reached in zero iterations without executing the loop body. In both cases, backward must-nontermination analysis is trivial.

## 4.7 Properties of Quantitative Weighted Transformers

Similar to the transformers discussed in Section 3.5, which satisfy various healthiness conditions such as continuity, linearity and monotonicity, aiding in concrete reasoning about nondeterministic programs and forming a foundation for compositional reasoning, we will explore such properties in our weighted setting.

More in details, we demonstrate that our weighted transformers possess analogous healthiness properties. We will show how they encompass existing properties discussed in Section 3.5 and highlight novel dualities.

### 4.7.1 Healthiness Properties

We start by presenting the healthiness properties of $\mathsf{wp}$, which have been explored by Batz et al. [24, Theorem 4.8], and novel healthiness properties of $\mathsf{sp}$.

**Theorem 4.7.1** (Healthiness Properties of Weighted Transformers)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ and $\mathsf{sp}$ satisfy the following properties:*

1. *Monotonicity:*

$$f \preceq g \qquad \text{implies} \qquad \mathsf{ttt}\,[\![C]\!]\,(f) \;\preceq\; \mathsf{ttt}\,[\![C]\!]\,(g)~, \quad \text{for } \mathsf{ttt} \in \{\mathsf{wp}, \mathsf{sp}\}~.$$

2. *Quantitative universal disjunctiveness: For any set of quantities $S \subseteq \mathbb{A}$,*

$$\mathsf{wp}\,[\![C]\!]\,(\curlyvee S) \;=\; \curlyvee\, \mathsf{wp}\,[\![C]\!]\,(S) \quad \text{and} \quad \mathsf{sp}\,[\![C]\!]\,(\curlyvee S) \;=\; \curlyvee\, \mathsf{sp}\,[\![C]\!]\,(S)~.$$

3. *Strictness:*

$$\mathsf{wp}\,[\![C]\!]\,(\mathbb{0}) \;=\; \mathbb{0} \quad \text{and} \quad \mathsf{sp}\,[\![C]\!]\,(\mathbb{0}) \;=\; \mathbb{0}~.$$

### Monotonicity

Monotonicity of weighted $\mathsf{wp}$ and $\mathsf{sp}$, defined in Theorem 4.7.1 (1), is a fundamental property that is subsumed by continuity. It aids compositional reasoning and ensures that larger input quantities result in larger output quantities.

### Continuity and Disjunctiveness

Similar to the quantitative $\mathsf{wp}$ and $\mathsf{sp}$ defined in Sections 3.3 and 3.4, our weighted transformers exhibit quantitative universal disjunctiveness, see Theorem 4.7.1 (2). This property ensures that our transformers preserve arbitrary suprema of quantities, which in turn implies their continuity and well-definedness. It is important to note that disjunctiveness in the weighted setting subsumes both quantitative universal disjunctiveness and conjunctiveness as described in Theorem 3.5.1 (3, 1), by simply considering the semirings $\langle \mathbb{R}^{\pm\infty}, \max, \min, -\infty, +\infty \rangle$ and $\langle \mathbb{R}^{\pm\infty}, \min, \max, +\infty, -\infty \rangle$ respectively.

### Strictness

Strictness of $\mathsf{wp}$ and $\mathsf{sp}$, defined in Theorem 4.7.1 (3) generalise the notion of strictness and costrictness as defined in Section 3.5, ensuring that if the input is the bottom element $\mathbb{0}$, the output is also the bottom element. In fact, recall that $\mathbb{0} \triangleq +\infty$ in the semirings used by $\mathsf{wlp}$ and $\mathsf{slp}$ (Tables 4.2 and 4.4), which leads to co-strictness.

### Linearity

Sub- and superlinearity, extensively studied by Kozen, McIver & Morgan, and Kaminski for probabilistic $\mathsf{w(l)p}$ transformers, are subsumed in our weighted setting as well, if the chosen semiring is commutative, as shown in Batz et al. [24, Theorem 4.8] for $\mathsf{wp}$. More in general, we show a more general result for non-commutative semirings, which is dual to the one for $\mathsf{sp}$. We start by splitting linearity into two more fine-grained properties, namely additivity and homogeneity.

**Definition 4.7.1** (Additivity)**.** *A transformer $T\colon \mathbb{A} \to \mathbb{A}$ is additive if, for any two quantities $f, g \in \mathbb{A}$,*

$$T(f \oplus g) = T(f) \oplus T(g).$$

$\triangleleft$

**Definition 4.7.2** (Homogeneity)**.** *For any scalar $a \in U$ and any quantity $f \in \mathbb{A}$, a transformer $T\colon \mathbb{A} \to \mathbb{A}$ is:*

- left homogeneous *if*
$$T(a \odot f) = a \odot T(f),$$
- right homogeneous *if*
$$T(f \odot a) = T(f) \odot a,$$
- homogeneous *if it is both left and right homogeneous.*

$\triangleleft$

Both additivity and homogeneity allow a complex property to be broken down into simpler ones, which can be proved separately. The results can then be soundly recombined to complete the proof of the original complex property. Left and right homogeneity are particularly useful in the context of non-commutative semirings, where the order of multiplication matters. We can prove the following.

**Theorem 4.7.2** (Extended Healthiness Properties on Weighted Transformers)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ and $\mathsf{sp}$ satisfy the following properties:*

*1. Additivity:* *For all $f, g \in \mathbb{A}$, we have*

$$\begin{aligned}\mathsf{wp}\,[\![C]\!]\,(f \oplus g) \quad &= \quad \mathsf{wp}\,[\![C]\!]\,(f) \oplus \mathsf{wp}\,[\![C]\!]\,(g) \quad \text{and} \\ \mathsf{sp}\,[\![C]\!]\,(f \oplus g) \quad &= \quad \mathsf{sp}\,[\![C]\!]\,(f) \oplus \mathsf{sp}\,[\![C]\!]\,(g)\ .\end{aligned}$$

2. *Right-homogeneity: For all $a \in U, f \in \mathbb{A}$, we have*

$$\mathsf{wp}\, [\![C]\!]\, (f \odot a) \quad = \quad \mathsf{wp}\, [\![C]\!]\, (f) \odot a \ .$$

3. *Left-homogeneity: For all $a \in U, f \in \mathbb{A}$, we have*

$$\mathsf{sp}\, [\![C]\!]\, (a \odot f) \quad = \quad a \odot \mathsf{sp}\, [\![C]\!]\, (f) \ .$$

While additivity has been proved in Batz et al. [24, Theorem 4.8] (only for $\mathsf{wp}$), we remark that right-homogeneity and left-homogenity are novel and essential to understand the behaviour of the two transformers. An intuition is the following:

- $\mathsf{sp}$ prepends the prequantity, so that $\mathsf{sp}\, [\![C]\!]\, (a \odot f)$ corresponds to prepending $a \odot f$ before the weights collected during the execution of $C$. This is equivalent to $a \odot \mathsf{sp}\, [\![C]\!]\, (f)$ since $\mathsf{sp}\, [\![C]\!]\, (f)$ will prepend $f$ before all the weights of the program $C$, resulting in $a \odot f$ followed by all other weights.
- $\mathsf{wp}$ appends the postquantity, so that $\mathsf{wp}\, [\![C]\!]\, (f \odot a)$ corresponds to appending $f \odot a$ after the weights collected during the execution of $C$. This is equivalent to $\mathsf{wp}\, [\![C]\!]\, (f) \odot a$ since $\mathsf{wp}\, [\![C]\!]\, (f)$ will append $f$ after all the weights of the program $C$, resulting in all other weights followed by $f \odot a$.

The properties above allow us to reason about more general linearity properties for $\mathsf{wp}$ and $\mathsf{sp}$, which we present next.

**Theorem 4.7.3** (Linearity). *For all programs $C$, $\mathsf{wp}[\![C]\!]$ is right-linear $\mathsf{sp}[\![C]\!]$ is left-linear. That is, for all $f, g \in \mathbb{A}$ and $a \in U$, we have:*

$$\begin{aligned} \mathsf{wp}\, [\![C]\!]\, (f \odot a \oplus g) &= \mathsf{wp}\, [\![C]\!]\, (f) \odot a \oplus \mathsf{wp}\, [\![C]\!]\, (g) \ , \\ \mathsf{sp}\, [\![C]\!]\, (a \odot f \oplus g) &= a \odot \mathsf{sp}\, [\![C]\!]\, (f) \oplus \mathsf{sp}\, [\![C]\!]\, (g) \ . \end{aligned}$$

Right-linearity and left-linearity show, again, the differences between backward and forward reasoning, respectively, for non-commutative semiring.

In the simpler setting of commutative semirings, we can prove the following result, subsuming the one for $\mathsf{wp}$ in Batz et al. [24, Theorem 4.8].

**Corollary 4.7.3.1** (Linearity)**.** *For all programs $C$, if $\odot$ is commutative, both $\mathsf{wp}[\![C]\!]$ and $\mathsf{sp}[\![C]\!]$ are linear. That is, for all $f, g \in \mathbb{A}$ and $a \in U$, we have:*

$$\begin{aligned}\mathsf{wp}\,[\![C]\!]\,(a \odot f \oplus g) &= a \odot \mathsf{wp}\,[\![C]\!]\,(f) \oplus \mathsf{wp}\,[\![C]\!]\,(g) \ ,\\ \mathsf{sp}\,[\![C]\!]\,(a \odot f \oplus g) &= a \odot \mathsf{sp}\,[\![C]\!]\,(f) \oplus \mathsf{sp}\,[\![C]\!]\,(g) \ .\end{aligned}$$

## 4.7.2 Dualities

We contend that our definition of $\mathsf{sp}$ is inherently intuitive, extending the classical concept of "reachable sets" to final distributions where the binary notion of reachability is substituted with real values. This inherent intuitiveness is additionally justified by the close connection between weakest pre and strongest post in our framework. We can also prove that the following more symmetrical duality between our $\mathsf{sp}$ and $\mathsf{wp}$ holds:

**Theorem 4.7.4** (Weighted $\mathsf{sp}$-$\mathsf{wp}$ Duality)**.** *For all programs $C$ and all functions $\mu, g \in \mathbb{A}$, we have*

$$\bigoplus_{\tau \in \Sigma} \mathsf{sp}\,[\![C]\!]\,(\mu)\,(\tau) \odot g(\tau) \quad = \quad \bigoplus_{\sigma \in \Sigma} \mu(\sigma) \odot \mathsf{wp}\,[\![C]\!]\,(g)\,(\sigma) \ .$$

In essence, Theorem 4.7.4 establishes a novel equivalence between forward and backward transformers. An intuition for the probabilistic semiring $\mathsf{Prob}$ is that computing the expectation of a quantity $g$ after the program execution—captured in the final distribution $\mathsf{sp}\,[\![C]\!]\,(\mu)$—is analogous to calculating the expected value through $\mathsf{wp}\,[\![C]\!]\,(g)\,(\sigma)$ but with the added nuance of being weighted by the initial distribution $\mu$. In the case of other semirings, the idea is that on the left-hand side all terminating traces originating from $\mu$ are aggregated and then $g$ appended. Conversely, on the right-hand side, the process is reversed: we initiate from $g$ and move backward until we reach $\mu$.

**Example 4.7.1.** *Consider the semiring of formal languages* $\mathcal{A} = \langle \mathcal{P}(\{a,b\}^*), \cup, \odot, \emptyset, \{\epsilon\}\rangle$ *and the program* $C = \{\odot\{a\}\} \;\square\; \{\odot\{b\}\}$. *Let* $\mu = \lambda\sigma\colon \{a\}$ *and* $g = \lambda\sigma\colon \{b\}$ *represent the prequantity we aim to prepend and the postquantity we intend to append at the end of the execution, respectively. This results in the following language:*

$$\begin{aligned}
&\bigoplus_{\sigma\in\Sigma} \mu(\sigma) \odot \mathsf{wp}\,[\![C]\!]\,(g)\,(\sigma)\\
&= \bigoplus_{\sigma\in\Sigma} \{a\} \odot (\mathsf{wp}\,[\![\odot\{a\}]\!]\,(g)\,(\sigma) \oplus \mathsf{wp}\,[\![\odot\{b\}]\!]\,(g)\,(\sigma))\\
&= \{a\} \odot (\{ab\} \oplus \{bb\}) \;=\; \{aab, abb\}
\end{aligned}$$

*which is exactly*

$$\begin{aligned}
&\bigoplus_{\tau\in\Sigma} \mathsf{sp}\,[\![C]\!]\,(\mu)\,(\tau) \odot g(\tau)\\
&= \bigoplus_{\tau\in\Sigma} (\mathsf{sp}\,[\![\odot\{a\}]\!]\,(\mu)\,(\sigma) \oplus \mathsf{sp}\,[\![\odot\{b\}]\!]\,(\mu)\,(\sigma)) \odot \{b\}\\
&= (\{aa\} \oplus \{ab\}) \odot \{b\} \;=\; \{aab, abb\} \qquad \triangleleft
\end{aligned}$$

We conclude the section by observing that we have the following connection as well.

**Proposition 4.7.5** (wp / sp Connection)**.** *For every predicate* $P, Q$ *and program* $C$*, we have:*

$$P \cap \mathsf{wp}\,[\![C]\!]\,(Q) \neq \emptyset \quad \text{iff} \quad Q \cap \mathsf{sp}\,[\![C]\!]\,(P) \neq \emptyset\ .$$

A simple consequence of the above is the duality $\not\models_{\text{pc}} \{\,P\,\}\ C\ \{\,\neg Q\,\}$ iff $\not\models_{\text{pi}} [\,\neg P\,]\ C\ [\,Q\,]$, which is not surprising, as the duality $\models_{\text{pc}} \{\,P\,\}\ C\ \{\,Q\,\}$ iff $\models_{\text{pi}} [\,\neg P\,]\ C\ [\,\neg Q\,]$ has already been explored in [17, p.22, "Duality"] and again in [78].

### 4.7.3 Loop Rules

Similarly to Section 3.7, we can derive inductive invariant based rules for loops in our weighted setting. We start by the induction rule for $\mathsf{wp}$ presented in [24, Theorem 5.1], but adapted to our more general setting.

**Theorem 4.7.6** (Quantitative Inductive Reasoning for $\mathsf{wp}$, Batz et al. [24])**.** *For any program $C$ and any quantities $i, f \in \mathbb{A}$, we have:*

$$\Phi_f(i) \preceq i \implies \mathsf{wp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \preceq i,$$

*where $\Phi_f(X) = [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X)$ is the characteristic function of $C^{\langle e,e'\rangle}$ w.r.t. $f$.* $\lhd$

The inductive rule for $\mathsf{sp}$ is similar to the one for $\mathsf{wp}$, but works with a higher-order invariant since the loop semantics is defined as a least fixed point of a higher-order function.

**Theorem 4.7.7** (Quantitative Inductive Reasoning for $\mathsf{sp}$)**.** *For any program $C$ and any quantities $i, f \in \mathbb{A}$, we have:*

$$\Psi(i) \preceq i \implies \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \preceq i(f),$$

*where $\Phi(\mathit{trnsf}) = \lambda f\colon \mathit{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!]))$ is the characteristic function of $C^{\langle e,e'\rangle}$.* $\lhd$

For total semirings, we can also prove a simpler version of the above theorem, which provides more direct reasoning similar to the $\mathsf{wp}$ rule in Theorem 4.7.6.

**Theorem 4.7.8** (Quantitative Inductive Reasoning for $\mathsf{sp}$ (total semirings))**.** *For any program $C$ and any quantities $i, f \in \mathbb{A}$, if the ambient semiring is a total semiring, we have:*

$$\Psi_f(i) \preceq i \implies \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \preceq i \odot [\![e']\!],$$

*where* $\Psi_f(X) \;=\; f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])$. $\lhd$

Depending on the choice of the semiring, we subsume all rules presented in Theorem 3.7.1.

## 4.8 A Case Study: The Ski Rental Problem

Weighted Programming, as demonstrated by Batz et al. [24], is a versatile formalism that can be applied to a wide range of problems. In this section, we revisit the well-known *Ski Rental Problem* [135], a classic problem in competitive analysis: Imagine embarking on a ski trip for an *unknown* number of days $n \geq 1$ without owning a pair of skis. At the start of each day, you must decide whether to rent skis for one day (cost: 1 Euro) or to buy a pair of skis (cost: $y$ Euros). The goal is to minimize the total cost over the entire trip.

If we knew the duration $n$ of the trip a priori, the optimal solution would be rather obvious: If $n \geq y$, we *buy* the skis. Otherwise, we are better off renting every day. This situation corresponds to an *offline* setting, with both $n$ and $y$ known, allowing for an optimal solution. Conversely, if the trip duration $n$ is *unknown* and only the cost $y$ of the skis is known, we are in an online setting of the Ski Rental Problem. Lacking knowledge about the entire input a priori often comes at the cost of *non-optimality.*

Specifically, we will model both the *optimal solution* to the *Ski Rental Problem* and the *optimal deterministic online algorithm*, using our forward transformer sp, to illustrate how we can model optimization problems and reason about the competitive ratios of online algorithms [136; 137]. A backward approach using wp has already been explored in [24]. In fact, as highlighted by Batz et al. [24], weighted programs using the tropical semiring Tropical provide an effective framework for analyzing the competitive performance of *infinite-state* online algorithms. This is because (1) nondeterministic programs naturally represent algorithmic problems and solutions, and (2) reasoning at the source code level allows for the analysis of *infinite-state* models. Modeling online algorithms as weighted programs builds on the work of [138; 139], who use finite-state weighted automata for the automated competitive analysis of *finite-state* online algorithms. Our approach is more general, as it removes the finite-state restriction, though this comes at the cost of limiting the full automation of verification.

## 4.8.1 Optimal Algorithm

```
/// $[\boldsymbol{n = 0 \wedge c \neq 0 \wedge y > 0}]$
while ( $c \neq 0$ ) {
    { $\odot\, y$ ⨾ $n \coloneqq$ nondet() ⨾ assume $n \geq 0$ ⨾ $c \coloneqq 0$ }
    $\square$ { $\odot\, 1$ ⨾ $n \coloneqq n + 1$ }
    $\square$ { $c \coloneqq 0$ }
}
/// $[\boldsymbol{y > 0 \wedge n \geq 0 \wedge c = 0}] + (\boldsymbol{n \curlywedge y})$
```

**Figure 4.4:** Program $C_{opt}$ modelling the optimal solution of the offline version of the *Ski Rental Problem.*

The program $C_{opt}$ depicted in Figure 4.4 models the optimal solution to the Ski Rental Problem. The loop in $C_{opt}$ iterates as long as the trip continues ($c \neq 0$). We begin with the assumptions: we start from day $n = 0$, have $c = 0$ (to enter the loop), and assume that the cost of buying skis is positive ($y > 0$). At each iteration, we have the following three options:

1. **Buy the skis**: In this case, we record the cost ($\odot\, y$) and no further decisions are needed. We can continue skiing for as many days as we wish ($n \coloneqq$ `nondet()` ⨾ `assume` $n \geq 0$), and the loop terminates ($c \coloneqq 0$).

2. **Rent the skis**: Here, we increase the cost ($\odot\, 1$) and increment the day counter ($n \coloneqq n + 1$) to proceed to the next day.

3. **End the trip**: If we decide to end the trip, no additional costs are incurred, and the loop terminates ($c \coloneqq 0$).

This model captures the essence of the Ski Rental Problem by providing a structured way to evaluate the costs associated with each decision. Each day, we non-deterministically explore all options, allowing us to determine the path with the lowest weight and thereby identify the optimal strategy to minimize the total cost over the duration of the trip. Operationally, we rely on the tropical semiring $\mathsf{Tropical} = \langle [0, +\infty], \min, +, +\infty, 0 \rangle$, where the weight of a terminating computation path is the sum of the weights along the path, representing the cost of a possible strategy. By taking the minimum of all costs, we can solve the problem. This can be achieved by computing the strongest postcondition of $C_{opt}$ with respect to the prequantity $[n = 0 \wedge c \neq 0 \wedge y > 0]$, which corresponds to our initial reasonable assumptions.

In fact, Theorem 4.4.2 instantiated to the tropical semiring $\mathsf{Tropical}$ guarantees that the strongest postcondition of $C_{opt}$ with respect to $[n = 0 \wedge c \neq 0 \wedge y > 0]$ is precisely:

$$
\begin{aligned}
&\mathsf{sp}\,[\![C_{opt}]\!]\,([n = 0 \wedge c \neq 0 \wedge y > 0]) \\
&= \bigoplus_{\sigma \in \Sigma} \mu(\sigma) \odot [\![C]\!](\sigma, \tau) \\
&= \bigcurlywedge_{\sigma \in \Sigma} \; [n = 0 \wedge c \neq 0 \wedge y > 0]\,(\sigma) + [\![C]\!](\sigma, \tau)\ ,
\end{aligned}
$$

which is exactly the optimal cost of the Ski Rental Problem. By taking the minimum of all $[\![C]\!](\sigma, \tau)$ (i.e., the cost of every possible strategy) and enforcing our initial assumptions $[n = 0 \wedge c \neq 0 \wedge y > 0]\,(\sigma)$, we can determine the optimal strategy.

Before we proceed with the analysis, we shall remark a *fundamental* difference compared to the program studied by Batz et al. [24], depicted below:

```
while ( $n > 0$ ) {
    $n \coloneqq n - 1$ ⨾
    $\{\odot\, 1\} \,\square\, \{\odot\, y \,⨾\, n \coloneqq 0\}$
}
```

Our program directly models the Ski Rental Problem by starting from day 0 and non-deterministically generating strategies for every possible number of days. We choose to solve our program using sp, which allows us to map states *after* the execution of the program to costs. This ensures that by the end of the execution, our program has accounted for any possible valid number of days.

In contrast, the approach taken by [24] is reversed: it starts from any number of days $n$ and iterates down to 0, effectively defining an algorithm to solve the problem rather than modeling the problem itself. This method aligns with wp, as it maps initial states (any number of days) to costs.

While we argue that both programs and approaches can effectively solve the Ski Rental Problem, the choice between forward and backward reasoning might depend on the specific problem at hand. For example, in Section 3.8.2, sp proved to be simpler for the problem at hand. The complexity of different transformers remains an open problem, as acknowledged by Verscht and Kaminski [140, Section 7]. Therefore, it is advantageous to have both tools at our disposal, allowing us to choose the most appropriate method for each specific scenario.

Going back to the proper analysis, we can now compute the optimal cost of the Ski Rental Problem by evaluating the strongest postcondition of $C_{opt}$ with respect to the prequantity $[n = 0 \land c \neq 0 \land y > 0]$. Since the tropical semiring Tropical is a total semiring, we can apply the simplified rule for total semirings from Theorem 4.4.1. Therefore, we define the loop characteristic function:

$$\Psi(X) \;=\; [n = 0 \land c \neq 0 \land y > 0] \curlywedge \mathsf{sp}\,[\![C]\!]\,(X + [c \neq 0])$$

and compute the fixed point $\mathsf{lfp}\ X\colon \Psi(X)$. The Kleene's iterates are as follows:

$$
\begin{aligned}
\Psi(+\infty) &= [n = 0 \land c \neq 0 \land y > 0] \curlywedge \mathsf{sp}\,[\![C]\!]\,(+\infty + [c \neq 0]) \\
&= [n = 0 \land c \neq 0 \land y > 0] \\
\Psi^2(+\infty) &= [n = 0 \land c \neq 0 \land y > 0] \\
&\curlywedge \mathsf{sp}\,[\![C]\!]\,([n = 0 \land c \neq 0 \land y > 0] + [c \neq 0]) \\
&= [n = 0 \land c \neq 0 \land y > 0] \curlywedge \mathsf{sp}\,[\![C]\!]\,([n = 0 \land c \neq 0 \land y > 0]) \\
&= [n = 0 \land c \neq 0 \land y > 0] \curlywedge ([n \geq 0 \land c = 0 \land y > 0] + y) \\
&\curlywedge ([n = 1 \land c \neq 0 \land y > 0] + 1) \curlywedge ([n = 0 \land y > 0 \land c = 0]) \\
\Psi^3(+\infty) &= [n = 0 \land c \neq 0 \land y > 0] \curlywedge \mathsf{sp}\,[\![C]\!]\,\big(\Psi^2(+\infty) + [c \neq 0]\big) \\
&= [n = 0 \land c \neq 0 \land y > 0] \curlywedge ([n \geq 0 \land c = 0 \land y > 0] + y) \\
&\curlywedge ([n = 1 \land c \neq 0 \land y > 0] + 1) \curlywedge ([n = 0 \land y > 0 \land c = 0]) \\
&\curlywedge ([n \geq 0 \land c = 0 \land y > 0] + 1 + y) \curlywedge ([n = 2 \land c \neq 0 \land y > 0] + 2) \\
&\curlywedge ([n = 1 \land c = 0 \land y > 0] + 1) \\
&= [n = 0 \land c \neq 0 \land y > 0] \curlywedge ([n \geq 0 \land c = 0 \land y > 0] + y) \\
&\curlywedge ([n = 1 \land c \neq 0 \land y > 0] + 1) \curlywedge ([n = 0 \land y > 0 \land c = 0]) \\
&\curlywedge ([n = 2 \land c \neq 0 \land y > 0] + 2) \curlywedge ([n = 1 \land c = 0 \land y > 0] + 1) \\
\Psi^m(+\infty) &= [y > 0] + \\
&\big(([n \geq 0 \land c = 0] + y) \curlywedge ([n = m-1 \land c \neq 0] + m - 1) \\
&\bigcurlywedge_{i=0}^{m-2} ([n = i \land c \neq 0] + i) \curlywedge ([n = i \land c = 0] + i)\big) \\
&\vdots \\
\Psi^\omega(+\infty) &= [y > 0] + \big(([n \geq 0 \land c = 0] + y) \\
&\curlywedge ([n \geq 0 \land c \neq 0] + n) \curlywedge ([n \geq 0 \land c = 0] + n)\big) \\
&= [y > 0 \land n \geq 0] + \big(([c = 0] + (n \curlywedge y)) \curlywedge ([c \neq 0] + n)\big)\ .
\end{aligned}
$$

We finally obtain:

$$\begin{aligned}
&\mathsf{sp}\,[\![C_{opt}]\!]\,([n = 0 \land c \neq 0 \land y > 0]) \\
&\quad = \big(\mathsf{lfp}\ X\colon \Psi(X)\big) + [c = 0] \\
&\quad = [y > 0 \land n \geq 0] + \big(([c = 0] + (n \curlywedge y)) \curlywedge ([c \neq 0] + n)\big) + [c = 0] \\
&\quad = [y > 0 \land n \geq 0 \land c = 0] + (n \curlywedge y)
\end{aligned}$$

In other words, $[y > 0 \land n \geq 0 \land c = 0]$ ensures that the assumptions hold, and the optimal cost of the Ski Rental Problem is $n \curlywedge y$, which is the expected result: we either rent every day, leading to a cost of $n$, if that is lower than buying the skis directly; otherwise, we buy the skis at the beginning.

### 4.8.2 Optimal Online Algorithm

Online algorithms perform their computation without knowing the entire input a priori. Rather, parts of the input are revealed to the online algorithm during the course of the computation. An online algorithm typically performs worse than the optimal offline algorithm.

```
/// $[n = 0 \land c \neq 0 \land y > 0]$
while ( $c \neq 0$ ) {
    { $c \coloneqq 0$ } $\Box$ {
        if ( $n + 1 < y$ ) {
            $\odot 1 ⨟ n \coloneqq n + 1$
          } else {
            $\odot y ⨟ n \coloneqq$ nondet() ⨟ assume $n \geq 0 ⨟ c \coloneqq 0$
        }
    }
}
/// $[n \geq 0 \land c = 0] + \big(([y > n] + n) \curlywedge ([y > 0] + 2y - 1)\big)$
```

**Figure 4.5:** Program $C_{onl}$ modelling the optimal solution of the online version of the *Ski Rental Problem.*

The program $C_{onl}$ depicted in Figure 4.5 models the optimal solution to the online version of the Ski Rental Problem. Unlike in Figure 4.4, here we cannot non-deterministically rent and buy the skis at the same time. Instead, as long as the duration of the trip is expected to be less than the cost of the skis ($n + 1 < y$), we continue to rent. Otherwise, we buy the skis. The assumptions are similar: we start from day $n = 0$, have $c \neq 0$ (to enter the loop), and assume that the cost of buying skis is positive ($y > 0$).

To evaluate the strongest postcondition of $C_{onl}$ with respect to the pre-quantity $[n = 0 \wedge c \neq 0 \wedge y > 0]$, we start by defining the loop characteristic function for total semirings (Theorem 4.4.1):

$$\Psi(X) \;=\; [n = 0 \wedge c \neq 0 \wedge y > 0] \curlywedge \mathsf{sp}\,[\![C]\!]\,(X + [c \neq 0])$$

and compute the fixed point $\mathsf{lfp}\ X\colon \Psi(X)$. The Kleene's iterates are as follows:

$$\begin{aligned}
\Psi(+\infty) &= [n = 0 \wedge c \neq 0 \wedge y > 0] \curlywedge \mathsf{sp}\,[\![C]\!]\,(+\infty + [c \neq 0]) \\
&= [n = 0 \wedge c \neq 0 \wedge y > 0] \\
\Psi^2(+\infty) &= [n = 0 \wedge c \neq 0 \wedge y > 0] \\
&\quad \curlywedge \mathsf{sp}\,[\![C]\!]\,([n = 0 \wedge c \neq 0 \wedge y > 0] + [c \neq 0]) \\
&= [n = 0 \wedge c \neq 0 \wedge y > 0] \curlywedge \mathsf{sp}\,[\![C]\!]\,([n = 0 \wedge c \neq 0 \wedge y > 0]) \\
&= [n = 0 \wedge c \neq 0 \wedge y > 0] \curlywedge ([n = 0 \wedge c = 0 \wedge y > 0]) \\
&\curlywedge ([n = 1 \wedge c \neq 0 \wedge y > 1] + 1) \curlywedge ([n \geq 0 \wedge c = 0 \wedge y = 1] + y) \\
\Psi^3(+\infty) &= [n = 0 \wedge c \neq 0 \wedge y > 0] \curlywedge \mathsf{sp}\,[\![C]\!]\,\big(\Psi^2(+\infty) + [c \neq 0]\big) \\
&= [n = 0 \wedge c \neq 0 \wedge y > 0] \curlywedge ([n = 0 \wedge c = 0 \wedge y > 0]) \\
&\curlywedge ([n = 1 \wedge c \neq 0 \wedge y > 1] + 1) \curlywedge ([n \geq 0 \wedge c = 0 \wedge y = 1] + y)
\end{aligned}$$

$$
\begin{aligned}
& \curlywedge ([n = 1 \wedge c = 0 \wedge y > 1] + 1) \curlywedge ([n = 2 \wedge c \neq 0 \wedge y > 2] + 2) \\
& \curlywedge ([n \geq 0 \wedge c = 0 \wedge y = 2] + 1 + y) \\
\Psi^m(+\infty) \;=\; & ([n = m-1 \wedge c \neq 0 \wedge y > m-1] + m - 1) \curlywedge \\
& \bigcurlywedge_{i=0}^{m-2} \big(([n = i \wedge c \neq 0 \wedge y > i] + i) \curlywedge ([n = i \wedge c = 0 \wedge y > i] + i) \\
& \qquad \curlywedge ([n \geq 0 \wedge c = 0 \wedge y = i+1] + i + y)\big) \\
\Psi^\omega(+\infty) \;=\; & ([n \geq 0 \wedge c \neq 0 \wedge y > n] + n) \curlywedge ([n \geq 0 \wedge c = 0 \wedge y > n] + n) \\
& \curlywedge ([n \geq 0 \wedge c = 0 \wedge y > 0] + 2y - 1) \\
& = \; [n \geq 0] + \big(([c \neq 0 \wedge y > n] + n) \\
& \curlywedge ([c = 0 \wedge y > n] + n) \curlywedge ([c = 0 \wedge y > 0] + 2y - 1)\big)
\end{aligned}
$$

We finally obtain:

$$
\begin{aligned}
& \mathsf{sp}\, [\![C_{onl}]\!] \, ([n = 0 \wedge c \neq 0 \wedge y > 0]) \\
& = \; \big(\mathsf{lfp}\ X\colon \Psi(X)\big) + [c = 0] \\
& = \; [n \geq 0] + \big(([c \neq 0 \wedge y > n] + n) \curlywedge ([c = 0 \wedge y > n] + n) \\
& \curlywedge ([c = 0 \wedge y > 0] + 2y - 1)\big) + [c = 0] \\
& = \; [n \geq 0 \wedge c = 0] + \big(([y > n] + n) \curlywedge ([y > 0] + 2y - 1)\big)
\end{aligned}
$$

In summary, $[n \geq 0 \wedge c = 0]$ ensures that the assumptions hold and the program terminates. The optimal cost of the Ski Rental Problem is $n$ (renting every day) if $y > n$, and otherwise it is $2y - 1$. This cost corresponds to renting for $y - 1$ days (as long as the number of days does not exceed the cost of the skis) and buying the skis on the following day. This is the expected result: the online algorithm performs worse than the optimal offline algorithm, but it remains competitive.

### 4.8.3 Competitive Analysis

An online algorithm generally does not perform as well as the optimal offline algorithm. *Competitive analysis* [136] is a method used to evaluate how close an online algorithm is to the optimal solution. The key concept in this analysis is the *competitive ratio* of an online algorithm. Using the notation of Batz et al. [24, Section 6.1.3], for a given problem instance $\rho$, let $\mathsf{ONL}(\rho)$ represent the cost incurred by an online algorithm $\mathsf{ONL}$, and let $\mathsf{OPT}(\rho)$ represent the cost incurred by its optimal offline counterpart $\mathsf{OPT}$. The competitive ratio of $\mathsf{ONL}$ is defined as

$$c_{\mathsf{ONL}} \triangleq \sup_{\rho} \frac{\mathsf{ONL}(\rho)}{\mathsf{OPT}(\rho)}$$

i.e., the smallest constant that upper-bounds the ratio between the cost of $\mathsf{ONL}$ and $\mathsf{OPT}$ for all problem instances $\rho$. We determine such competitive ratios by $\mathsf{sp}$-reasoning on weighted programs as follows below. For simplicity, let us assume both $n > 0$ and $y > 0$. This assumption is reasonable since the problem becomes trivial if the trip ends immediately or the skis are free. We conclude that the competitive ratio is 2 since:

$$\begin{aligned}
& c_{\mathsf{ONL}} \\
= \;& \sup_{\rho} \frac{\mathsf{ONL}(\rho)}{\mathsf{OPT}(\rho)} \\
= \;& \sup_{\tau \in \Sigma\colon \tau(c)=0} \frac{\mathsf{sp}\,[\![C_{onl}]\!]\,([n = 0 \wedge c \neq 0 \wedge y > 0])\,(\tau)}{\mathsf{sp}\,[\![C_{opt}]\!]\,([n = 0 \wedge c \neq 0 \wedge y > 0])\,(\tau)} \\
= \;& \sup_{\tau \in \Sigma\colon \tau(c)=0} \left( \frac{[n \geq 0 \wedge c = 0] + \big(([y > n] + n) \curlywedge ([y > 0] + 2y - 1)\big)}{[y > 0 \wedge n \geq 0 \wedge c = 0] + (n \curlywedge y)} \right)(\tau) \\
= \;& \sup_{\tau \in \Sigma\colon \tau(c)=0} \left( \frac{([y > n] + n) \curlywedge (2y - 1)}{n \curlywedge y} \right)(\tau) \\
= \;& \sup_{\tau \in \Sigma\colon \tau(c)=0} \left( \Big( [n \geq y] + 2 - \frac{1}{y} \Big) \curlywedge ([n < y] + 1) \right)(\tau) \\
= \;& 2\,.
\end{aligned}$$

## 4.9 Related Work

### Generalized Predicate Transformers and Hoare Logics

The closest work to ours is [24], which introduced wp and wlp for weighted programming. Our work, in contrast, focuses on sp and its connections with wp. Generalizations of predicate transformers, such as potential functions $\Phi : \Sigma \to \mathbb{R}_{\geq 0}$, have been used in amortized complexity analysis [141] and in resource bound verification systems [142; 143]. These approaches, excluding procedures and recursion, can be seen as instances of our framework.

Aguirre and Katsumata [108] introduced a general monad-parametrized strongest post, but they do not provide a constructive definition. Gaboardi et al. [130] introduced graded categories over partially ordered monoids, allowing the modeling of probabilities or weights, but their framework is limited to bounded loops.

Outcome Logic and its variations [19; 20; 74; 144; 145] are more general as they allow relating multiple program executions, but they are more complex and do not support weakest-precondition-style or strongest-postcondition-style reasoning.

Other triples and predicate transformers, incorporating different forms of non-determinism, have been explored recently by Verscht and Kaminski [146, 140]. Their work delves into a broader range of dualities, connections, and asymmetries across various logics but does not address weighted programs.

### Algebras and Semirings

O'Conner [147] and Dolan [148] showed that computational problems such as shortest paths can be reduced to linear algebra over a suitable semiring. In particular, they compute the so-called *star* or *closure* $x^* = 1 + x + x^2 + \ldots$, where $x$ is a matrix over the semiring. They demonstrate that $x^*$ is the least solution of the equation $x^* = 1 + x \cdot x^*$, which is closely related to the fixed points we use in our wp and sp. However, our framework extends this to infinite state spaces and allows reasoning in a symbolic fashion. The techniques by

O'Conner [147] and Dolan [148] are limited to finite-state problems and shortest paths in finite graphs. Laird et al. [149] and Brunel et al. [129] considered functional languages parameterized by a semiring, allowing reasoning about resource consumption, reachability probabilities, or expected values. However, Brunel et al. [129] do not deal with infinite computations.

Kleene Algebras with Tests (KAT) and their variations [150; 151; 152] can model imperative programs in an abstract fashion by identifying them with objects from a *Kleene algebra* that includes an embedded Boolean subalgebra. ProbGKAT extends KAT to allow probabilistic transitions [153], while Graded KAT [154] replaces the Boolean subalgebra with a more general object that can be viewed as a semiring with additional operations and axioms. The elements of this semiring constitute the graded (or weighted) outcomes of the tests. However, (Graded) KAT are not concrete programming languages; their main purpose is to prove general results about imperative languages with loops and conditionals in an abstract fashion. Investigating which of our transformers and invariant-based proof rules can be derived in Graded KAT is an appealing direction for future work.

## 4.10 Conclusions

We developed a strongest postcondition-style calculus for reasoning about weighted programs, complementing the weakest precondition-style calculus developed by [24] and showcasing novel dualities and healthiness properties.

Future work includes exploring several existing techniques within our weighting setting, such as $k$-induction [155; 156], program synthesis [157; 158], and separation logics [118; 5; 96]. Additionally, we aim to further investigate rules for termination and nontermination proving [159; 81; 160], potentially by considering more general program semantics that can handle weighted non-terminating traces [93; 144]. Similar to [146; 140; 145; 161], we also want to explore our transformers in the context of demonic non-determinism. Finally, a promising direction for future work is to generalize the verification infrastructure for probabilistic programs introduced by Schröer et al. [162] to weighted programs.

Chapter 5

# Quantitative Hyper Transformers

Sections 5.1 to 5.3 and 5.6 to 5.10 of this chapter are based on the paper [18], but extended to include our novel forward transformers. This chapter introduces three novel quantitative transformers for hyperproperties: the *weakest hyper pre* (whp), the *strongest hyper post* (shp), and the *strongest liberal hyper post* (slhp). These transformers enable reasoning about quantitative hyperproperties over both nondeterministic and probabilistic programs, extending well-established program verification techniques to more complex domains.

Whereas existing calculi allow reasoning about the expected value that a quantity assumes after program termination from a *single initial state*, our transformers operate on *initial sets of states* or *initial probability distributions*. Through these transformers, we (i) obtain comprehensive predicate transformers for hyper Hoare logic and (ii) enable reasoning about so-called *hyperquantities*—including not only expected values but also quantities like variance that were previously out of scope. Our framework reveals novel dualities between forward and backward transformers, correctness and incorrectness, as well as nontermination and unreachability.

## 5.1 Introduction

Program logic has evolved from Hoare Logic [2] through relational extensions for hyperproperties [65] and Incorrectness Logic for bug-finding [6], ultimately leading to unified frameworks that can reason about both correctness and incorrectness across single and multiple program traces. We build on two such developments—Outcome Logic (OL) [19; 20; 74] and Hyper Hoare Logic (HHL) [73]—which advocate that a single logic can be used to prove (or disprove) a wide variety of properties, including hyperproperties, and we present a novel *(quantitative) weakest pre calculus* perspective. Weakest precondition calculi date back to the 1970's when Dijkstra [63, 1] introduced them as predicate transformer semantics for imperative programs. Given a command $C$ and a postcondition $Q$, the *weakest liberal precondition* is the weakest assertion $P$ such that running $C$ in any state satisfying $P$ will terminate in a state satisfying $Q$ or not terminate at all. Pratt [163] observed that these calculi have a close connection to Hoare Logic and they were later used in a completeness proof for Hoare Logic [164].[1]

Weakest liberal preconditions have been generalized to probabilistic programs to allow for reasoning about expected values of random variables in a program that terminates from a *single initial state*. The core idea in these quantitative calculi [13; 14; 16; 17] is that one can replace predicates over states by real-valued functions. All these calculi, classical and quantitative, offer predicate transformers that have two key benefits over program logics: First, they discover the *most precise* assertions to make a triple valid. Second, they provide a calculus with a clear path towards mechanizability.

In this chapter, we present a novel *weakest pre calculus* (whp) for *reasoning about quantitative hyperproperties* over *programs with effects* that cause the program execution to branch such as nondeterminism or probabilistic choice, in the style of weighted programming [24] or OL [74] (Section 4.2). We generalize

[1]Although the original relative completeness proof of Cook [107] used the *strongest postcondition*, a later, simplified proof by Clarke [164] used the weakest liberal precondition.

existing work on quantitative weakest pre calculi [17] by considering program termination from *initial sets of states* or *initial probability distributions* rather than single initial states. We thus obtain weakest preconditions for HHL and enable reasoning about so-called *hyperquantities* (Section 5.3), which include expected values (considered in previous work), but also more general quantities that were not supported before, e.g. variance. Unlike HHL, our whp supports quantitative probabilistic reasoning, employing *hyperquantities* evaluated in probability distributions. Moreover, we show that many existing logics are subsumed by whp (Section 5.7), and how to prove (and disprove) properties in those logics. whp is hence a single calculus for correctness and incorrectness analysis, which enjoys expected healthiness and duality properties (Section 5.6). whp can be applied in a variety of settings, which we illustrate through a range of examples (Section 5.8).

Similarly to how predicate transformers and Hoare-like logics empower programmers to demonstrate correctness, we contend that our framework offers researchers a deeper comprehension of existing logics. Our calculus reveals novel dualities between forward and backward transformers, correctness and incorrectness, as well as nontermination and unreachability.

### 5.1.1 Main challenges

While we observe parallels with existing wp calculi [16; 121], HHL, and OL, extending these frameworks to our setting of (quantitative) hyperproperties involves several non-trivial steps, including lifting the calculus from initial states to hyperproperties and weighted sets of states, and completely revisiting the rules to handle our more expressive assertion language with hyperquantitites. For example, we will show that our loop rule involves a fixpoint over a higher-order function (Proposition 5.3.1), which is not considered in previous works. A summary of these key technical insights follows.

**Quantitative Reasoning over Hyperproperties**

Defining the meaning of quantitative reasoning in a hyperproperty setting was another challenge. We observed the similarity between hyperproperties and weighted distributions, which necessitated the development of new rules and interpretations to handle this complexity. Each of the whp rules are different compared to those of Zhang and Kaminski [17]. In addition, the rule for nondeterministic choice is different from those of Hyper Hoare Logic (HHL) and Outcome Logic (OL), since we aim for completeness in a predicate transformer semantics, whereas HHL and OL both require additional *infinitary* rules.

**Restrictions of Hyperquantities**

We investigated why reasoning over quantities—as in Kozen [13]; McIver and Morgan [14]; Kaminski [16]; Batz et al. [24]; Zhang and Kaminski [17]—is simpler, and studied the restrictions of hyperquantities to derive simpler rules similar to the existing ones (Section 5.6.3). This involved identifying and formalizing conditions under which our more general framework could simplify, bridging the complexity gap between hyperproperties and traditional properties while maintaining greater expressivity. In fact, even in restricted settings (e.g., the expected value hyperquantity), we can reason about initial probability distributions rather than single initial states.

## 5.2 Overview: Strategies for Reasoning about Hyperproperties

We begin our discussion by focusing on noninterference [54]—a hyperproperty commonly used in information security applications. More precisely, noninterference stipulates that any two executions of a program with the same *public* inputs (but potentially different *secret* inputs) must have the same public outputs. This guarantees that the program does not *leak* any secret information to unprivileged observers. As a demonstration, consider the following program, where the variable $\ell$ (for low) is publicly visible, but $h$ (for high) is secret.

$$C_{\mathrm{ni}} \;=\; \texttt{assume}\ h > 0 \mathbin{⨟} \ell \coloneqq \ell + h$$

While we will show shortly that $C_{\mathrm{ni}}$ does not satisfy noninterference, we first show how one might attempt to prove it does. Following the approach of logics such as Hyper Hoare Logic (HHL), one can define $\mathrm{low}(\ell)$ to mean that the value of $\ell$ is equal in any pair of executions, and then attempt to establish the validity of $\models_{\mathrm{hh}} \{\, \mathrm{low}(\ell) \,\}\ C_{\mathrm{ni}}\ \{\, \mathrm{low}(\ell) \,\}$, meaning that if $C_{\mathrm{ni}}$ is executed twice with the same initial $\ell$, then $\ell$ will also have the same value in both executions when (and if) the program finishes—hence, the initial values of $h$ cannot influence $\ell$.

HHL is sound and complete, meaning that any true triples can be proven in it. However, doing so is not always straightforward. For example, although the specification of the triple above does not mention $h$, intermediary assertions required to complete the proof *must* mention $h$, and introducing this information cannot be done in a mechanical way, but rather requires inventiveness.

Furthermore, whereas HHL (analogously to OL) can disprove any of its triples [73, Theorem 4], deriving either a positive or negative result—i.e., proving that a program is secure or not—requires one to know a priori which spec they wish to prove, or trying both.

The predicate transformer approach we advocate in this chapter proves highly advantageous as it only requires a *single hyperpostcondition* to determine the most precise *hyperprecondition* that validates (or invalidates) a triple. In that sense, it solves the two aforementioned issues by mechanically working backward from the postcondition, *discovering* intermediary assertions along the way, and finding the *most precise* precondition with respect to the desired spec.

In this chapter, we define a novel whp calculus, and the validity of $\mathrm{low}(\ell) \subseteq \mathsf{whp}\ [\![C_{\mathrm{ni}}]\!]\ (\mathrm{low}(\ell))$ is the answer to the noninterference problem, without the risk of attempting to prove an invalid triple. In the case of the above example, our calculus leads us to a simple counterexample; if we have $\ell = 0$ and $h = 1$ in the first execution and $\ell = 0$ and $h = 2$ in the second execution, then clearly $\mathrm{low}(\ell)$ holds, but the values of $\ell$ will be distinguishable at the end. This means that the program is insecure. In the remainder of this section, we will give an overview of the technical ideas underlying our $\mathsf{whp}$ calculus.

### 5.2.1 Classical Weakest Pre

Dijkstra's original weakest precondition calculus employs *predicate transformers* of type

$$\mathsf{wp}[\![C]\!]\colon \quad \mathbb{B} \;\to\; \mathbb{B}\;, \qquad \text{where} \quad \mathbb{B} \;=\; \Sigma \to \{0,1\}\;.$$

The set $\mathbb{B}$ of maps from program states ($\Sigma$) to Booleans ($\{0,1\}$) can also be thought of as predicates or assertions over program states. The *angelic* weakest precondition transformer $\mathsf{wp}[\![C]\!]$ maps a postcondition $\psi$ to a precondition $\mathsf{wp}\,[\![C]\!]\,(\psi)$ such that executing $C$ on an initial state in $\mathsf{wp}\,[\![C]\!]\,(\psi)$ guarantees that $C$ *can*[2] terminate in a final state in $\psi$. Given a semantics function $[\![C]\!]$ such that $[\![C]\!](\sigma,\,\tau) = 1$ iff executing $C$ on initial state $\sigma$ can terminate in $\tau$, the angelic $\mathsf{wp}$ is so defined:

$$\mathsf{wp}\,[\![C]\!]\,(\psi) \quad = \quad \{\sigma \in \Sigma \mid \exists\tau\colon [\![C]\!](\sigma,\tau) = 1 \;\wedge\; \tau \in \psi\}$$

This allows to check if an angelic total correctness triple holds via the well-known fact

$$\models_{\text{atc}} \{\, G \,\}\; C \;\{\, F \,\} \;\text{ is valid} \qquad \text{iff} \qquad G \implies \mathsf{wp}\,[\![C]\!]\,(F)\;\;.$$

While the above is a set perspective on $\mathsf{wp}$, an equivalent perspective on $\mathsf{wp}$ is a map perspective: the predicate $\mathsf{wp}\,[\![C]\!]\,(\psi)$ is a map that takes as input an initial state $\sigma$, determines for each reachable final state $\tau$ the (truth) value

[2]$C$ is a *nondeterministic* program. For the *demonic* setting and for deterministic programs, we can replace "can" by "will".

$\psi(\tau)$, takes a disjunction over all these truth values, and finally returns the truth value of that disjunction. More symbolically, we have

$$\mathsf{wp}\,[\![C]\!]\,(\psi)\,(\sigma) \qquad = \qquad \bigvee_{\tau\colon\, [\![C]\!](\sigma,\tau)=1} \psi(\tau)\ .$$

### 5.2.2 Weakest Pre over Hyperproperties

To reason about hyperproperties [21], we lift our domain of discourse from *sets of states* to *sets of sets of states*, i.e. we go

$$\text{from} \qquad \mathsf{wp}[\![C]\!]\colon \quad \mathbb{B} \to \mathbb{B} \qquad \text{to} \qquad \mathsf{whp}\,[\![C]\!]\colon \quad \mathbb{BB} \to \mathbb{BB}\ ,$$

where $\mathbb{B} = \Sigma \to \{0,1\}$, as before, and $\mathbb{BB} = \mathcal{P}(\Sigma) \to \{0,1\}$.

Given a postcondition $\psi \in \mathbb{B}$ (i.e. a predicate ranging over states), classical angelic $\mathsf{wp}\,[\![C]\!]\,(\psi)$ anticipates for a *single* initial state $\sigma$ whether running $C$ on $\sigma$ can reach $\psi$. Given a hyperpostcondition $\psi\!\!\psi \in \mathbb{BB}$ (a predicate ranging over *sets* of states), the weakest hyperprecondition $\mathsf{whp}\,[\![C]\!]\,(\psi\!\!\psi)$ anticipates for a given *set* of initial states $\phi$ (a precondition), whether the set of states reachable from executing $C$ on every state in $\phi$ satisfies $\psi\!\!\psi$. To define $\mathsf{whp}$, we will rely on $\mathsf{sp}\,[\![C]\!]\,(\phi)$, the classical *strongest postcondition* [64] of $C$ with respect to precondition $\phi$; in other words: the set of all final states *reachable* by executing $C$ on any initial state in $\phi$.

From a set perspective, we have:

$$\mathsf{whp}\,[\![C]\!]\,(\psi\!\!\psi) \quad = \quad \{\phi \in \mathcal{P}(\Sigma) \mid \mathsf{sp}\,[\![C]\!]\,(\phi) \in \psi\!\!\psi\}\ ,$$

i.e., $\mathsf{whp}\,[\![C]\!]\,(\psi\!\!\psi)$ is defined as the set of all preconditions $\phi$ such that their strongest postconditions $\mathsf{sp}\,[\![C]\!]\,(\phi)$ satisfy the hyperpostcondition $\psi\!\!\psi$.

From a map perspective, $\mathsf{whp}\,[\![C]\!]\,(\psi\!\!\psi)$ maps a hyperproperty $\psi\!\!\psi$ over postconditions to a hyperproperty $\mathsf{whp}\,[\![C]\!]\,(\psi\!\!\psi)$ over preconditions. In other words, we are anticipating whether the strongest postcondition of $\phi$ satisfies the hyperpostcondition $\not\psi$:

$$\mathsf{whp}\ [\![C]\!]\,(\not\psi)\,(\phi) \quad = \quad \not\psi(\mathsf{sp}\,[\![C]\!]\,(\phi))\ .$$

In particular, executing $C$ on a *precondition* $\phi$ satisfying $\mathsf{whp}\ [\![C]\!]\,(\not\psi)$ guarantees that the set of reachable states $\mathsf{sp}\,[\![C]\!]\,(\phi)$ will satisfy $\not\psi$. Reasoning about hyperproperties is strictly more expressive as it relates multiple executions. We showcase this in the following examples.

**Example 5.2.1** (Weakest Hyperpreconditions)**.** *Given some precondition $\phi$, if $\phi$ satisfies*

1. $\mathsf{whp}\ [\![C]\!]\,(\lambda\rho.\ |\rho| = 2)$*, then the number of states reachable from $\phi$ by executing $C$ is 2.*

2. $\mathsf{whp}\ [\![C]\!]\,(\lambda\rho.\ \mathit{Bugs} \subseteq \rho)$*, where $\mathit{Bugs} \subseteq \Sigma$, then* all *states in the set Bugs are reachable by running $C$ on* some *state in $\phi$ (this amounts to Incorrectness Logic [6]).*

3. $\mathsf{whp}\ [\![C]\!]\,(\lambda\rho.\ \rho \subseteq \mathit{Good})$*, where $\mathit{Good} \subseteq \Sigma$, then starting from $\phi$ only Good can be reached or $C$ does not terminate (this amounts to partial correctness [2]).*

*We refer to Clarkson and Schneider [21] for more examples of hyperproperties.* $\lhd$

**Remark 5.2.1.** *Outcome Logic [19] and Hyper Hoare Logic [73] can handle all of Example 5.2.1 via $\models \{\not\phi\}\ C\ \{\not\psi\}$ triples, but are agnostic of preconditions not satisfying $\not\phi$ since $\phi \notin \not\phi$ does not imply $\mathsf{sp}\,[\![C]\!]\,(\phi) \notin \not\psi$. Predicate transformers, on the other hand, yield the most precise assertions in the sense that $\phi \in \mathsf{whp}\ [\![F]\!]\,(\not\psi)$ iff $\mathsf{sp}\,[\![C]\!]\,(\phi) \in \not\psi$.* $\lhd$

**Remark 5.2.2.** *Note that* $\mathsf{sp}$ *is deterministic, rendering the consideration of angelic/demonic* $\mathsf{whp}$ *unnecessary.*

### 5.2.3 Strongest Post over Hyperproperties

We can reason about hyperproperties in a forward manner as well. Similarly to $\mathsf{whp}$, we lift our domain of discourse from *sets of states* to *sets of sets of states*, i.e. we go

$$\text{from} \qquad \mathsf{sp}[\![C]\!]\colon \quad \mathbb{B} \to \mathbb{B} \qquad \text{to} \qquad \mathsf{shp}\,[\![C]\!]\colon \quad \mathbb{B}\!\mathbb{B} \to \mathbb{B}\!\mathbb{B}\ ,$$

where $\mathbb{B} \;=\; \Sigma \to \{0,1\}$ and $\mathbb{B}\!\mathbb{B} \;=\; \mathcal{P}(\Sigma) \to \{0,1\}$, as before.

Given a precondition $\phi \in \mathbb{B}$ (i.e., a predicate ranging over states), the classical strongest postcondition $\mathsf{sp}\,[\![C]\!]\,(\phi)$ determines, for a *single* final state $\tau$, whether there exists any initial state satisfying the precondition $\phi$ that terminates in $\tau$.

Given a hyperprecondition $\phi\!\!\!\phi \in \mathbb{B}\!\mathbb{B}$ (a predicate ranging over *sets* of states), the strongest hyperpostcondition $\mathsf{shp}\,[\![C]\!]\,(\phi\!\!\!\phi)$ determines, for a given *set* of final states $\psi$ (a postcondition), whether there exists any initial *set of states* satisfying the hyperprecondition $\phi\!\!\!\phi$ such that executing $C$ yields the set of final states $\psi$.

From a set perspective, we have:

$$\mathsf{shp}\,[\![C]\!]\,(\phi\!\!\!\phi) \quad = \quad \{\mathsf{sp}\,[\![C]\!]\,(\phi) \mid \phi \in \phi\!\!\!\phi\}\ ,$$

i.e., is defined as the set of all postconditions $\psi$ that we can obtain by computing the strongest postconditions $\mathsf{sp}\,[\![C]\!]\,(\phi)$ from any precondition satisfying the hyperprecondition $\phi\!\!\!\phi$.

From a map perspective, $\mathsf{shp}\,[\![C]\!]\,(\phi\!\!\!\phi)$ transforms a hyperproperty $\phi\!\!\!\phi$ over preconditions into a hyperproperty $\mathsf{shp}\,[\![C]\!]\,(\phi\!\!\!\phi)$ over postconditions. Specifically, $\mathsf{shp}\,[\![C]\!]\,(\phi\!\!\!\phi)\,(\psi)$ identifies all initial preconditions $\phi$ that result in the postcondition $\psi$ after executing $C$ (i.e., $\mathsf{sp}\,[\![C]\!]\,(\phi) \;=\; \psi$). It then checks if these preconditions satisfy the hyperprecondition $\phi\!\!\!\phi$, effectively determining whether the hyperpredicate $\psi$ could have been true before executing $C$, thereby

<u>retro</u>cipating the truth of $\phi\!\!\!/$. Formally, we have:

$$\mathsf{shp}\, [\![C]\!]\, (\phi\!\!\!/)\, (\psi) \quad = \bigvee_{\phi \text{ with } \mathsf{sp}\, [\![C]\!](\phi)=\psi} \phi\!\!\!/(\phi) \ ,$$

where $\curlyvee$ represents angelic nondeterminism (similarly to $\mathsf{sp}$). We showcase its expressivity this in the following examples.

**Example 5.2.2** (Strongest Hyperpostconditions)**.** *Given some precondition $\phi$, if*

1. $\mathsf{shp}\, [\![C]\!]\, (\{\phi\}) \subseteq \lambda\rho.\ |\rho| = 2$*, then the number of states reachable from $\phi$ by executing $C$ is 2.*
2. $\mathsf{shp}\, [\![C]\!]\, (\{\phi\}) \subseteq \lambda\rho.\ \mathit{Bugs} \subseteq \rho$*, where $\mathit{Bugs} \subseteq \Sigma$, then* all *states in the set Bugs are reachable by running $C$ on* some *state in $\phi$ (this amounts to Incorrectness Logic [6]).*
3. $\mathsf{shp}\, [\![C]\!]\, (\{\phi\}) \subseteq \lambda\rho.\ \rho \subseteq \mathit{Good}$*, where $\mathit{Good} \subseteq \Sigma$, then starting from $\phi$ only Good can be reached or $C$ does not terminate (this amounts to partial correctness [2]).*

$\lhd$

### 5.2.4 Quantitative Reasoning over Hyperproperties

As shown in [13; 14; 16], one can replace predicates over states by real-valued functions, also known as quantities [17, Section 3]. These quantitative calculi subsume the classical ones by mimicking predicates through the use of *Iverson brackets* [103]. To design a calculus for quantitative reasoning over hyperproperties, we lift quantities in $\mathbb{A} = \{\, f \mid f\colon \Sigma \to \mathbb{R}^{\pm\infty} \,\}$, i.e. functions of type $\Sigma \to \mathbb{R}^{\pm\infty}$, to hyperquantities.

**Definition 5.2.1** (Hyperquantities)**.** *The set of all* hyper*quantities is defined by*

$$\mathbb{A}\mathbb{A} \ = \ \{\, f\!\!f \mid f\!\!f\colon (\Sigma \to \mathbb{R}^{\pm\infty}) \to \mathbb{R}^{\pm\infty} \,\} \ ,$$

*$\mathbb{AA}$ is the set of all functions $ff\colon \mathbb{A} \to \mathbb{R}^{\pm\infty}$ associating an* extended real *(i.e. either a non-negative real number or $+\infty$) to each quantity in $\mathbb{A}$. The point-wise order*

$$ff \;\preceq\; gg \qquad \text{iff} \qquad \forall\, f \in \mathbb{A}\colon \quad ff(f) \;\leq\; gg(f)$$

*renders $\langle \mathbb{AA},\, \preceq \rangle$ a complete lattice with join $\curlyvee$ and meet $\curlywedge$, given point-wise by*

$$\begin{aligned} ff \curlyvee gg \;&=\; \lambda f\colon\, \max\bigl\{ ff(f),\, gg(f) \bigr\} \\ ff \curlywedge gg \;&=\; \lambda f\colon\, \min\bigl\{ ff(f),\, gg(f) \bigr\}\ . \end{aligned}$$

*Joins and meets over arbitrary subsets exist (because suprema and maxima over arbitrary sets of extended reals exist). For $a \curlyvee b \curlywedge c$, we assume that $\curlywedge$ binds stronger than $\curlyvee$, so we read that as $a \curlyvee (b \curlywedge c)$.* ◁

To distinguish between quantities Definition 4.2.6 and hyperquantities, we show some examples of both.

**Example 5.2.3** (Quantities)**.** *Examples of quantities (not hyperquantities) include:*

- *(sub)probability distributions over program states;*
- *extended measures over states;*
- *sets of states $P \subseteq \Sigma$, represented via the Iverson bracket:*

$$[P]\,(\sigma) = \begin{cases} 1 & \text{if } \sigma \in P \\ 0 & \text{otherwise} \end{cases}$$

◁

Hyperquantities enable *quantitative* reasoning, e.g., measures over probability distributions.

**Example 5.2.4** (Hyperquantities over Distributions)**.** *Given a quantity $f\colon \Sigma \to \mathbb{R}_{\geq 0}^{\infty} \in \mathbb{A}$ (think: random variable $f$), we define hyperquantities*

$$
\begin{aligned}
\mathbb{E}[f] &\triangleq \lambda\mu.\ \sum_{\sigma} f(\sigma)\cdot\mu(\sigma)\\
\mathit{Var}[f] &\triangleq \lambda\mu.\ \mathbb{E}[f^2](\mu) - \big(\mathbb{E}[f](\mu)\big)^2\\
\mathit{Cov}[f,g] &\triangleq \lambda\mu.\ \mathbb{E}[fg](\mu) - \mathbb{E}[f](\mu)\cdot\mathbb{E}[g](\mu)\ ,
\end{aligned}
$$

*that take as input quantities (interpreted as probability distributions) $\mu\colon \Sigma \to \mathbb{R}_{\geq 0}^{\infty}$. The above hyperquantities are then respectively* expected value*,* variance *and* covariance *of $f$ (and $g$) over $\mu$.* $\lhd$

We now present as an example an adaptation of [73, Example 3] – showcasing how Boolean Hyper Hoare Logic (HHL) would deal with statistical properties.

**Example 5.2.5** (Mean Number of Requests)**.** *Consider a program $C_{db}$ where, after termination, the variable $n$ represents the number of database requests performed. For a final set of states $\rho \subseteq \Sigma$, we define its mean number of requests by $mean_n(\rho) = \sum_{\sigma\in\rho} \frac{\sigma(n)}{|\rho|}$.*

HHL *allows to bound $mean_n$ by a* specific number*, say 2, by taking as hyperpostcondition $Q = \lambda\rho.mean_n(\rho) \leq 2$. Proving the* HHL *triple $\models_{\mathrm{hh}} \{\,\mathsf{true}\,\}\ C_{db}\ \{\,Q\,\}$ then ensures that for every initial set of states, the* mean number *of performed requests after the execution of $C_{db}$ is at most 2.* $\lhd$

**Example 5.2.6** (Quantitative Information Flow)**.** *Consider a program, $C_{qif}$ containing lowly and highly sensitive variables. As outlined in [17, Section 8.1], we will demonstrate in Section 5.8.3, how our framework also enables to determine, for instance, the maximum initial value allowable for the secret variable $h$ based on observing a specific final value for $l$.* HHL *allows reasoning only about the existence of some information flow or about a bound over $h$.* $\lhd$

Using instead quantitative weakest hyper pre has two main advantages over using HHL:

#### Beyond Decision Problems

While $\mathsf{HHL}$ and Outcome Logic ($\mathsf{OL}$) are capable of statistical reasoning, our quantitative calculus can directly *measure* quantities of interest, such as the information flow. For example, $\mathsf{whp}\, [\![C_{\mathrm{db}}]\!]\, (\mathrm{mean}_n)$ gives us the mean number of requests after executing $C_{\mathrm{db}}$ rather than just bounding it.

#### Probability Distributions

Reasoning about means is restrictive, especially for infinite sets. As shown in Example 5.2.4, hyperquantities assign numerical values such as expected values to distributions. For example, $\mathsf{whp}\, [\![C_{\mathrm{db}}]\!]\, (\mathbb{E}[n])\, (\mu)$ maps *every distribution* $\mu$ to the *expected number of requests* after executing $C_{\mathrm{db}}$ on some initial state drawn from $\mu$.

### 5.2.5 Limitations

#### Hyperproperties over probability distributions

We can only reason about properties over probability distributions *or* hyperproperties over single states (i.e., properties over sets of states) in our framework. In other words, we cannot reason about hyperproperties over probability distributions, such as probabilistic non-interference [165]. To illustrate what such reasoning would entail, consider how we might formulate *probabilistic non-interference* for *observational* programs (unlike Definition 3 from O'Neill et al. [165], which addresses interactive programs). Intuitively, probabilistic non-interference requires that for any two initial states with identical low-security inputs $l$, the probability distribution of final low-security values must also be identical. This yields the following formal definition:

$$\begin{aligned}\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l) \implies& \\ &\forall v\colon \mathsf{sp}\, [\![C]\!]\, (\mathbf{1}_{\sigma_1})\, ([l = v]) = \mathsf{sp}\, [\![C]\!]\, (\mathbf{1}_{\sigma_2})\, ([l = v]).\end{aligned}$$

Now consider the program in Figure 5.1, which is the non-interactive analog of Program 4 from O'Neill et al. [165].

```
if ( $h$ is even ) {
    { $l \coloneqq 0$ } [ $0.99$ ] { $l \coloneqq 1$ }
} else {
    { $l \coloneqq 0$ } [ $0.01$ ] { $l \coloneqq 1$ }
}
```

**Figure 5.1:** A program that does not satisfy probabilistic non-interference.

If the probabilistic choices were replaced with non-deterministic ones, then the program would satisfy generalized non-interference, since we cannot infer the value of $h$ by observing the value of $l$. However, with *probabilistic* choices, the situation changes; observing $l = 0$ means that it is more *likely* that the first path has been chosen, i.e., that $h$ is even. We can address this situation with the above definition, and show that the program above does not satisfy probabilistic non-interference. Unfortunately, such property is a hyperproperty over probability distributions, and goes beyond our framework. Extending whp to support probabilistic non-interference is an interesting future direction.

### Demonic total correctness & angelic partial correctness

Similarly to Zhang and Kaminski [17]; Ascari et al. [78]; Zilberstein et al. [19]; Dardinier and Müller [73], we subsume neither demonic Hoare logic for total correctness, nor angelic Hoare logic for partial correctness, which are subsumed respectively by existing demonic wp and angelic wlp [16]. This limitation is due to how our whp anticipates an angelic sp (as usual in literature), which only considers terminating states, and not the existence of divergent ones. We stress that this limitation holds for Hyper Hoare Logic and Outcome Logic, and that our initial objective was to establish a weakest precondition calculus for them.

## 5.3 Quantitative Weakest Hyper Pre

In the rest of the chapter we will employ a more general definition of hyperquantities that:

- is parametrised to arbitrary semi-rings;
- allows for *mixed-sign hyperquantities*, as opposed to our previous setting [18];

**Definition 5.3.1** ((Weighted) Hyperquantities). *Given a partial semiring* $\mathcal{A} = \langle U, \oplus, \odot, \mathbb{0}, \mathbb{1} \rangle$*, the set* $\mathbb{AA}_{\mathcal{A}}$ *of all* hyperquantities *is defined as the set of all functions* $f\!\!f \colon (\Sigma \to U) \to \mathbb{R}^{\pm\infty}$*, i.e.*

$$\mathbb{AA}_{\mathcal{A}} \;=\; \{ f\!\!f \colon (\Sigma \to U) \to \mathbb{R}^{\pm\infty} \}$$

$\triangleleft$

As with quantities, we will use the simplified notation $\mathbb{AA}$ when the underlying semiring $\mathcal{A}$ is clear from context. Our formulation deliberately maps quantities of type $\Sigma \to U$ to hyperquantities of type $(\Sigma \to U) \to \mathbb{R}^{\pm\infty}$. While we could have adopted a more general approach where hyperquantities have type $(\Sigma \to U) \to U'$ with $U$ and $U'$ potentially drawn from distinct semirings, we have intentionally restricted ourselves to the codomain of real numbers $\mathbb{R}^{\pm\infty}$. This design choice follows the approach of [16], allowing us to leverage the well-established theory of real-valued quantitative reasoning. The resulting framework achieves a balance—extending beyond traditional hyperproperties while maintaining sufficient concreteness to serve as a foundation for practical verification tools.

First of all, we show in which sense we can represent hyperproperties via functions. We have already seen that predicates can be encoded via Iverson brackets (Definition 4.2.7), and decoded by the support set, since every quantity $f \colon \Sigma \to U$ can be seen as a set of states via $\mathsf{supp}\,(f) = \{\sigma \colon f(\sigma) \neq \mathbb{0}\}$. For example, the set of reachable states starting from $\phi \subseteq \Sigma$ is $\mathsf{supp}\,(\mathsf{sp}\,[\![C]\!]\,([\phi]))$.

| $C$ | $\mathsf{whp}\,\llbracket C \rrbracket\,(\mathit{ff})$ |
|---|---|
| $x \coloneqq e$ | $\mathit{ff}\,[x/e]$ |
| $x \coloneqq \texttt{nondet()}$ | $\lambda f\colon \mathit{ff}(\bigoplus_\alpha f\,[x/\alpha])$ |
| $\odot\, w$ | $\mathit{ff} \odot w$ |
| $C_1 \fatsemi C_2$ | $\mathsf{whp}\,\llbracket C_1 \rrbracket\,\big(\mathsf{whp}\,\llbracket C_2 \rrbracket\,(\mathit{ff})\big)$ |
| $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$ | $\text{Ƨ}\,\nu_1, \nu_2\colon\ \mathit{ff}(\nu_1 \oplus \nu_2) \curlywedge \mathsf{whp}\,\llbracket C_1 \rrbracket\,([\nu_1]) \curlywedge \mathsf{whp}\,\llbracket C_2 \rrbracket\,([\nu_2])$ |
| $C^{\langle e, e' \rangle}$ | $\lambda f\colon \mathit{ff}\big(\big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X : \mathsf{trnsf}(\mathsf{sp}\,\llbracket C \rrbracket\,(X \odot \llbracket e \rrbracket)) \oplus X \odot \llbracket e' \rrbracket\big)(f)\big)$ |

**Table 5.1:** Rules for defining the quantitative weakest hyper pre transformer.

To encode and decode hyperpredicates, we need to introduce hyper Iverson brackets.

**Definition 5.3.2** (Hyper Iverson Brackets)**.** *Given a semiring $\mathcal{A} = \langle U, \oplus, \odot, \mathbb{0}, \mathbb{1} \rangle$, for a hyperpredicate $\phi\colon \mathcal{P}(\mathcal{P}(\Sigma))$ we define the hyper Iverson bracket $[\phi] : (\Sigma \to U) \to \mathbb{R}^{\pm\infty}$ by*

$$[\phi]\,(f) \;=\; \begin{cases} +\infty & \textit{if}\ \mathsf{supp}\,(f) \in \phi \\ -\infty & \textit{otherwise}\ . \end{cases}$$

◁

For a hyperquantity $\mathit{ff}$, its corresponding hyperpredicate is defined by $\mathsf{supp}\,(\mathit{ff}) = \{f\colon \mathit{ff}(f) > 0\}$. We shall remark that hyperpredicates in our setting can represent predicates over quantities, including hyperproperties and predicates over probability distributions.

**Definition 5.3.3** (Quantitative Weakest Hyper Pre)**.** *The* quantitative weakest hyper pre transformer

$$\mathsf{whp}\colon \quad \mathsf{wReg} \to (\mathbb{AA} \to \mathbb{AA})$$

*is defined inductively according to the rules in Table 5.1.* ◁

Let us show for some of the rules how the quantitative weakest hyper pre semantics can be developed and understood analogously to Dijkstra's classical weakest preconditions.

**Assignment.**

The weakest pre*condition* of an assignment is given by

$$\mathsf{wp}\,[\![x \coloneqq e]\!]\,(\psi) \;=\; \psi\,[x/e]$$

where $\psi\,[x/e]$ denotes the substitution of the variable $x$ in $\psi$ with the expression $e$. From a semantic perspective, this replacement can be expressed as

$$\psi\,[x/e] \coloneqq \lambda\sigma\colon \psi\Big(\sigma\,[x \mapsto \sigma(e)]\Big).$$

In simpler terms, the weakest precondition operates by predicting the operational semantics: it examines whether, given an initial state $\sigma$, the final state $\sigma\,[x \mapsto \sigma(e)]$ adheres to the condition $\psi$.

For *quantitative weakest hyper pre*, a similar approach is taken, but we anticipate the strongest post rather than the operational semantics. Therefore, the value of $f\!f$ in the resulting distribution (or set of states) after the execution of $x \coloneqq e$ on the initial distribution (or set) $f$ corresponds to $f\!f$, but evaluated at the final distribution $\mathsf{sp}\,[\![x \coloneqq e]\!]\,(f) = \bigoplus_\alpha f\,[x/\alpha] \odot [x = e\,[x/\alpha]]$. We thus define the syntactic replacement of the variable $x$ in a hyperquantity $f\!f$ by $f\!f\,[x/e] \coloneqq \lambda f\colon f\!f(\mathsf{sp}\,[\![x \coloneqq e]\!]\,(f))$, yielding the rule:

$$\mathsf{whp}\;[\![x \coloneqq e]\!]\,(f\!f) \;=\; f\!f\,[x/e]\;\;.$$

In fact, when we replace $x$ with $e$ in formula $f\!f$ syntactically, we are essentially modifying a function over a quantity (e.g., a distribution) $f$ to an updated function that treats $x$ as a new variable with the evaluation $e$. We provide a semantics definition, to avoid handling variable scoping, similarly to other works on weakest preconditions (e.g. [16]).

**Example 5.3.1.** *Consider the assignment $x \coloneqq x - 1$ and the hyperquantity*

$\mathbb{E}[x]$. *We have:*

$$\begin{aligned}
&\mathsf{whp}\ [\![x \coloneqq x-1]\!]\,(\mathbb{E}[x]) \\
&= \mathsf{whp}\ [\![x \coloneqq x-1]\!]\left(\lambda\mu.\ \sum_{\sigma} \sigma(x)\cdot\mu(\sigma)\right) \\
&= \lambda\mu.\ \sum_{\sigma} \sigma(x-1)\cdot\mu(\sigma) \\
&= \mathbb{E}[x-1]\ ,
\end{aligned}$$

*i.e., the anticipated expected value of $x$ after the assignment is exactly the expected value of $x-1$ before the assignment.*

**Nondeterministic Assignment:**

The nondeterministic assignment is analogous to the standard assignment, but now with $x$ ranging over any possible value. Therefore, the weakest hyper pre of a nondeterministic assignment is given by:

$$\mathsf{whp}\ [\![x \coloneqq \mathtt{nondet()}]\!]\,(\mathit{ff}) \;=\; \lambda f\colon \mathit{ff}\left(\bigoplus_{\alpha} f\,[x/\alpha]\right)$$

**Assume/Weighting.**

For the classical weakest precondition, we have $\mathsf{wp}\,[\![\mathtt{assume}\ \varphi]\!]\,(\psi) \;=\; \varphi\wedge\psi$. Indeed, if the initial state $\sigma$ satisfies the combined precondition $\varphi\wedge\psi$, the execution of $\mathtt{assume}\ \varphi$ entails progression through the assumption of $\varphi$. Since the assumption itself does not alter the program state, the process concludes in state $\sigma$, which also satisfies the post $\psi$. Conversely, if $\sigma$ fails to meet $\varphi\wedge\psi$, the execution of $\mathtt{assume}\ \varphi$ results in either not progressing through the assumption of $\varphi$ or passing through the assumption but $\sigma$ not satisfying the post $\psi$. The *quantitative weakest hyper pre* on an initial distribution (set) $f$ anticipates the strongest post, yielding the rule $\mathsf{whp}\ [\![\mathtt{assume}\ \varphi]\!]\,(\mathit{ff}) = \lambda f\colon \mathit{ff}([\varphi]\odot f)$.

To simplify the notation, we introduce the product $\odot$ between quantities and hyperquantities as:

$$ff \odot w \;=\; \lambda f\colon ff(f \odot w) \qquad w \odot ff \;=\; \lambda f\colon ff(w \odot f)\;,$$

leading to the syntactically simpler rule $\mathsf{whp}\,[\![\texttt{assume}\ \varphi]\!]\,(ff) \;=\; ff \odot [\varphi]$. We point out that the order of the arguments is reversed compared to weighted $\mathsf{wp}$ ( [24]). In our setting, $\mathsf{whp}$ anticipates the strongest post, which computes the prequantity $f$ before the weight $w$.

For the more general weighting statement, we obtain the rule:

$$\mathsf{whp}\,[\![\odot\, w]\!]\,(ff) \;=\; ff \odot w\;,$$

where $w$ can be any quantity.

**Nondeterministic Choice.**

When executing nondeterministic choice $\{\,C_1\,\} \,\Box\, \{\,C_2\,\}$ on some initial state $\sigma$, operationally *either* $C_1$ *or* $C_2$ will be executed. Hence, the execution will reach either a final state in which executing $C_1$ on $\sigma$ terminates or a final state in which executing $C_2$ on $\sigma$ terminates (or no final state if both computations diverge).

The *angelic* weakest pre*condition* of $\{\,C_1\,\} \;\Box\; \{\,C_2\,\}$ is given by $\mathsf{wp}\,[\![\{\,C_1\,\} \;\Box\; \{\,C_2\,\}]\!]\,(\psi) \;=\; \mathsf{wp}\,[\![C_1]\!]\,(\psi) \;\vee\; \mathsf{wp}\,[\![C_2]\!]\,(\psi)$. Indeed, whenever an initial state $\sigma$ satisfies the precondition $\mathsf{wp}\,[\![C_1]\!]\,(\psi)$ or $\mathsf{wp}\,[\![C_2]\!]\,(\psi)$, then — either by executing $C_1$ or $C_2$ — it is possible that the computation will terminate in some final state satisfying the postcondition $\psi$.

Moving to hyperquantities, a key insight emerges: nondeterminism is effectively eliminated when computing the strongest post because $\mathsf{sp}[\![\{\,C_1\,\} \,\Box\, \{\,C_2\,\}]\!]$ operates deterministically. While operationally the nondeterministic choice represents executing either $C_1$ or $C_2$, semantically $\mathsf{sp}[\![\{\,C_1\,\} \,\Box\, \{\,C_2\,\}]\!]$ aggregates the results from both branches into a single combined distribution. This semantic determinism means that given the same initial distribution (or set of states), $\mathsf{sp}[\![\{\,C_1\,\} \,\Box\, \{\,C_2\,\}]\!]$ will always produce the same final distribution.

Consequently, the value of $\mathit{ff}$ in the resulting distribution (or set of states) after executing either $C_1$ or $C_2$ on the initial distribution (or set) $f$ is

$$\begin{aligned}&\mathsf{whp}\ \llbracket \{\, C_1 \,\} \square \{\, C_2 \,\} \rrbracket\, (\mathit{ff}) \ = \\ &\reflectbox{$\mathsf{S}$}\, \nu_1, \nu_2 \colon \Sigma \to U \colon\ \ \mathit{ff}(\nu_1 \oplus \nu_2) \curlywedge \mathsf{whp}\ \llbracket C_1 \rrbracket\, ([\nu_1]) \curlywedge \mathsf{whp}\ \llbracket C_2 \rrbracket\, ([\nu_2])\ .\end{aligned}$$

Recalling that the final distribution is the combination of $\mathsf{sp}\,\llbracket C_1 \rrbracket\,(f)$ and $\mathsf{sp}\,\llbracket C_2 \rrbracket\,(f)$, identifying $\nu_i$ such that $\nu_i = \mathsf{sp}\,\llbracket C_i \rrbracket\,(f)$ makes computing $\mathit{ff}(\nu_1 \oplus \nu_2)$ sufficient. By aggregating over every $\nu_i$ for which $\mathsf{whp}\ \llbracket C_i \rrbracket\,([\nu_1])\,(f)$ holds, we ensure that only those $\nu_i$ where $\nu_i = \mathsf{sp}\,\llbracket C_i \rrbracket\,(f)$ will contribute. Consequently, $\mathit{ff}(\nu_1 \oplus \nu_2)$ precisely equals $\mathit{ff}(\mathsf{sp}\,\llbracket \{\, C_1 \,\} \square \{\, C_2 \,\} \rrbracket\,(f))$.

**Remark 5.3.1.** *In the case of* $\{\, C_1 \,\} \square \{\, C_2 \,\}$*,* OL *and* HHL *exhibit forward-style rules that are simpler but not complete, if taken separately. In fact, while such rules maintain soundness, completeness necessitates the inclusion of an additional existential rule. As our approach adopts a weakest pre style calculus aiming for both soundness and completeness, the introduction of the suprema quantification becomes imperative. This quantification mirrors the existential rule utilized in* OL *and* HHL*, encompassing all relevant cases. Our rule shares similarities with den Hartog [166, Definition 6.5.2], although they provide multiple rules depending on the structure of the hyperquantity. Since in our thesis we focus on semantic assertions, we refrain from analyzing the syntactic structure of hyperquantities. However, we later introduce simpler rules for the class of additive hyperquantities, as outlined in Definition 5.6.1.*

**Sequential Composition.**

What is the anticipated value of $\mathit{ff}$ after executing $C_1 \fatsemi C_2$, i.e. the value of $\mathit{ff}$ after first executing $C_1$ and then $C_2$? To answer this, we first anticipate the value of $\mathit{ff}$ after execution of $C_2$ which gives $\mathsf{whp}\ \llbracket C_2 \rrbracket\,(\mathit{ff})$. Then, we anticipate the value of the intermediate quantity $\mathsf{whp}\ \llbracket C_2 \rrbracket\,(\mathit{ff})$ after execution of $C_1$, yielding:

$$\mathsf{whp}\;[\![C_1 \fatsemi C_2]\!]\,(ff) = \mathsf{whp}\;[\![C_1]\!]\,(\mathsf{whp}\;[\![C_2]\!]\,(ff))\enspace .$$

**Iteration**

The rule for $C^{\langle e,e'\rangle}$ is obtained by anticipating the execution of $C^{\langle e,e'\rangle}$. More precisely, for a given initial quantity $f$ we want to anticipate the value of $ff$ on the final quantity after executing $C^{\langle e,e'\rangle}$. This is possible by computing $ff$ on $\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f)$, yielding the rule:

$$\begin{aligned}\mathsf{whp}\;\left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](ff) \;&=\\ &\lambda f\colon ff\Big(\big(\mathsf{lfp}\;\mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(f)\Big)\enspace .\end{aligned}$$

At first, it might seem counterintuitive that the recursive variable $X$ is computed before evaluating the weight $e$, especially since this differs from $\mathsf{wp}$ where the weight is evaluated before the postquantity. However, this approach directly aligns with the definition of $\mathsf{whp}$, which anticipates $\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f)$. In the strongest postcondition semantics (see Section 4.4), the weight $e$ is applied after computing the recursive quantity $X$, and our transformer faithfully preserves this sequence of operations.

Our rule is consistent in the sense that it is a solution of the equation:

$$\begin{aligned}&\mathsf{whp}\left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right]\\ &=\;\mathsf{whp}\left[\!\!\left[\Big\{\odot e \fatsemi C \fatsemi C^{\langle e,e'\rangle}\Big\} \,\Box\, \{\odot e'\}\right]\!\!\right]\\ &=\;\lambda hh\,\lambda f\colon\\ &\qquad\quad \mathcal{S}\nu_1,\nu_2\colon\; hh(\nu_1 \oplus \nu_2)\cdot \mathsf{whp}\left[\!\!\left[\odot e \fatsemi C \fatsemi C^{\langle e,e'\rangle}\right]\!\!\right]([\nu_1])\,(f)\cdot \mathsf{whp}\;[\![\odot e']\!]\,([\nu_2])\,(f)\\ &=\;\lambda hh\,\lambda f\colon \mathcal{S}\nu_1,\nu_2\colon\; hh(\nu_1 \oplus \nu_2)\cdot \mathsf{whp}\left[\!\!\left[C \fatsemi C^{\langle e,e'\rangle}\right]\!\!\right]([\nu_1])\,(f \odot [\![e]\!])\cdot [\nu_2]\,(f \odot [\![e']\!])\\ &=\;\lambda hh\,\lambda f\colon \mathcal{S}\nu\colon\; hh(\nu \oplus f \odot [\![e']\!])\cdot \mathsf{whp}\;[\![C]\!]\left(\mathsf{whp}\left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right]([\nu])\right)(f \odot [\![e]\!])\end{aligned}$$

Indeed one can show the following.

**Proposition 5.3.1** (Consistency of iteration rule). *Let*

$$\Phi(\mathit{trnsf}) = \lambda \mathit{hh}\, \lambda f \colon \mathcal{S}\nu\colon\ \mathit{hh}(\nu \oplus f \odot [\![e']\!]) \cdot \mathsf{whp}\ [\![C]\!]\ (\mathit{trnsf}([\nu]))\ (f \odot [\![e]\!])$$

*Then,* $\mathsf{whp}\ [\![C^{\langle e,e'\rangle}]\!]$ *is a fixpoint of the higher order function* $\Phi(\mathit{trnsf})$*, that is:*

$$\Phi(\lambda \mathit{ff}\ \lambda\mu\colon \mathit{ff}(\mathsf{sp}\ [\![C^{\langle e,e'\rangle}]\!]\ (\mu))) = \lambda \mathit{ff}\ \lambda\mu\colon \mathit{ff}(\mathsf{sp}\ [\![C^{\langle e,e'\rangle}]\!]\ (\mu))$$

**Remark 5.3.2.** *One might attempt a rule for* $C^{\langle e,e'\rangle}$ *by defining* $F(X) = \lambda f\colon X(f \oplus \mathsf{sp}\ [\![C]\!]\ (f \odot [\![e]\!]))$*. Intuitively,* $F$ *takes as input a hyperquantity* $X$*, but instead of applying it on a distribution* $f$*, it computes one iteration of the loop* $\mathsf{sp}\ [\![C]\!]\ (f \odot [\![e]\!])$ *and then pass all as argument of* $X$*. Recalling that* $\Psi_f(X) = f \oplus \mathsf{sp}\ [\![C]\!]\ (X \odot [\![e]\!])$*, one can then observe that for every* $n \in \mathbb{N}$*:*

$$\begin{aligned}
\lambda f\colon \mathit{ff}(f \odot [\![e']\!]) &= \lambda f\colon \mathit{ff}(\Psi_f(\mathbb{0}) \odot [\![e']\!]) \\
F(\lambda f\colon \mathit{ff}(f \odot [\![e']\!])) &= \lambda f\colon \mathit{ff}(\Psi_f^2(\mathbb{0}) \odot [\![e']\!]) \\
&\vdots \\
F^n(\lambda f\colon \mathit{ff}(f \odot [\![e']\!])) &= \lambda f\colon \mathit{ff}(\Psi_f^{n+1}(\mathbb{0}) \odot [\![e']\!])
\end{aligned}$$

*However, it is important to note that in general,* $F^n(\lambda f\colon \mathit{ff}(f \odot [\![e']\!]))$ *does not form an ascending or descending chain. For example, take* $\mathit{ff} = \mathbf{1}_\nu$*, where* $\nu$ *is a probability distribution. it is very well possible that* $\mathbf{1}_\nu(\Psi_f^k(\mathbb{0}) \odot [\![e']\!]) = \mathbb{1}$ *for some* $k, \mu$*: that is, we anticipate an incomplete proability distribution and find out that it is equal* $\nu$*. However, at the* $k+1$ *iteration, the anticipated probability distribution is refined, so that it could be* $\Psi_\mu^{k+1}(\mathbb{0}) \odot [\![e']\!] \neq \nu$*, leading to a decreasing iterate. Additionally, it is not always desirable to stop at the first fixpoint - as multiple extra iterations might be needed to compute the correct anticipated probability distribution. That said, it is entirely possible that simpler rules exist when restricting* $\mathit{ff}$*, see e.g. Table 5.4.* ◁

We also argue why we chose to use the least fixed point in the loop rule, which aligns naturally with the definition of strongest postcondition

($\mathsf{sp}$). Although a greatest fixed point formulation would theoretically be viable—potentially leading to a novel transformer that anticipates the strongest liberal postcondition ($\mathsf{slp}$)—the practical utility of such an approach remains questionable.

After having provided an intuition on the rules, let us show that $\mathsf{whp}$ does actually anticipate $\mathsf{sp}$.

**Theorem 5.3.2** (Characterization of $\mathsf{whp}$)**.** *For all programs $C$, hyperquantities $\mathit{ff} \in \mathbb{AA}$, and quantities $f \in \mathbb{A}$*

$$\mathsf{whp}\,[\![C]\!]\,(\mathit{ff})\,(f) \quad = \quad \mathit{ff}(\mathsf{sp}\,[\![C]\!]\,(f)) \ .$$

For a given hyperquantity $\mathit{ff}$ and initial quantity $\mu$, $\mathsf{whp}\,[\![C]\!]\,(\mathit{ff})\,(\mu)$ represents the value assumed by $\mathit{ff}$ in the final quantity reached after the termination of $C$ on $\mu$. Unlike standard $\mathsf{wp}$, which distinguishes between terminating and nonterminating states, $\mathsf{whp}$ does not make this distinction. When there are no terminating states, i.e., $\mathsf{sp}\,[\![C]\!]\,(\mu) = \mathbb{0}$, the value of $\mathsf{whp}\,[\![C]\!]\,(\mathit{ff})\,(\mu)$ is determined by $\mathit{ff}(\mathbb{0})$. The assignment of any desired value to the empty set of states 0 by the hyperquantity $\mathit{ff}$ allows us to express both weakest preconditions and weakest liberal ones.

## 5.4 Quantitative Strongest Hyper Post

In this section, we define a forward transformer that complements the backward moving weakest pre transformer introduced in Section 5.3. Before doing so, we introduce a new construct, which can be viewed as an instantiation of the separating conjunction ($*$) from O'Hearn's and Pym's [167] logic of bunched implications (BI). The separating conjunction has been extensively used in formal verification across various domains, involving so-called resource semantics [168]: in heaps in (Incorrectness) Separation Logic [5; 118; 76], where it allows reasoning about disjoint memory regions; in its quantitative analogs [96; 169], where it handles quantitative properties over heap-manipulating programs; in concurrent programming [170], where it helps in reasoning about parallel processes that operate on shared memory; in probabilistic scenarios [171], where it is used to reason about probabilistic independence. Perhaps more relevantly to our framework, the separating conjunction is employed in both Hyper Hoare Logic [73, Definition 6] and Outcome Logic and its variants [19; 74; 172, Outcome Conjunction] to reason about hyperproperties. Unsurprisingly, we will also utilize this construct, extending its definition to our quantitative setting in a manner similar to the quantitative lifting described in [96] for Separation Logic. We will refer to it as the *quantitative outcome conjunction*, denoted by $\boxtimes$.

**Definition 5.4.1** (Quantitative Outcome Conjunction)**.** *The quantitative outcome conjunction $\boxtimes$ is defined as follows:*

$$(\mathit{ff} \boxtimes \mathit{gg})(\nu) \triangleq \bigcurlyvee_{\mu_1,\mu_2 \in \mathbb{A}\colon\, \mu_1 \oplus \mu_2 = \nu} \mathit{ff}(\mu_1) \curlywedge \mathit{gg}(\mu_2) \ ,$$

*where $\oplus$ represents the pointwise lifting of the semiring's addition operation to quantities, meaning $(f \oplus g)(x) = f(x) \oplus g(x)$ for all inputs $x$.*

Intuitively, each hyperquantity $\mathit{ff}, \mathit{gg}$ is evaluated separately in a split of the quantity $\nu$. The conjunction $(\mathit{ff} \boxtimes \mathit{gg})(\nu)$ finds the optimal way to partition $\nu$ into two parts—$\mu_1$ and $\mu_2$—such that the minimum of $\mathit{ff}(\mu_1)$ and $\mathit{gg}(\mu_2)$ is maximized. This separation allows us to reason about different properties holding in different parts of our program state.

When reasoning about hyperproperties, the quantitative outcome conjunction subsumes the Outcome Conjunction of Zilberstein et al. [19], and in turn, the Hyper Conjunction of Dardinier and Müller [73, Definition 6] (which is an instantiation of the former with the powerset monad). In fact, we can prove the following.

**Definition 5.4.2** (Outcome Conjunction for Powerset Monad [19])**.** *Let $m_1, m_2, m \in \mathcal{P}(\Sigma)$ be sets of states and $P, Q \in \mathcal{P}(\mathcal{P}(\Sigma))$ be hypersets of* *states. The Outcome Conjunction $\otimes$ is defined as follows:*

$$m \models P \otimes Q \iff \exists m_1, m_2 \in \mathcal{P}(\Sigma)\colon\ m = m_1 \oplus m_2 \textit{ and } m_1 \models P \textit{ and } m_2 \models Q$$

$\lhd$

**Proposition 5.4.1** (Subsumption of Outcome Conjunction)**.** *Let $\otimes$ be the Outcome Conjunction [19] instantiated to the powerset monad. Then, for any $P, Q \in \mathcal{P}(\mathcal{P}(\Sigma))$ and $m \in \mathcal{P}(\Sigma)$, we have:*

$$m \models P \otimes Q \quad \text{iff} \quad m \in \mathsf{supp}\left([P] \boxtimes [Q]\right)$$

*Proof.*

$$\begin{aligned} & m \models P \otimes Q \\ & \quad \text{iff} \quad \exists m_1, m_2 \in \mathcal{P}(\Sigma).\ m = m_1 \oplus m_2 \text{ and } m_1 \models P \text{ and } m_2 \models Q \\ & \quad \text{iff} \quad \exists m_1, m_2 \in \mathcal{P}(\Sigma).\ m = m_1 \oplus m_2 \text{ and } \ [P]\,(m_1) > 0 \text{ and } \ [Q]\,(m_2) > 0 \\ & \quad \text{iff} \quad ([P] \boxtimes [Q])(m) > 0 \\ & \quad \text{iff} \quad m \in \mathsf{supp}\left([P] \boxtimes [Q]\right) \end{aligned}$$

$\square$

The proposition above holds for the powerset monad and the probability distribution monads, as defined in [19]. However, it could be very easily extended to any monad, by considering a more general definition of Iverson brackets. We decide to not do so, as we are only interested in the powerset and probability distribution monads. Intuitively, $S \in \mathsf{supp}\left([\not\!\not\phi] \boxtimes [\not\!\not\psi]\right)$ asserts that the hyperproperties $\not\!\not\phi$ and $\not\!\not\psi$ each hold in reachable executions: $S$ can be partitioned into two parts, $S_1$ and $S_2$, such that $\not\!\not\phi$ holds in $S_1$ and $\not\!\not\psi$ holds in $S_2$. Our encoding, similar to Outcome Logic, extends beyond simple hyperproperties. For example, given two properties $\phi$ and $\psi$ over probability distributions, we have that $\mu \in \mathsf{supp}\left([\phi] \boxtimes [\psi]\right)$ if $\mu$ can be split into two

| $\boldsymbol{C}$ | $\mathsf{\textbf{shp}}\,\llbracket \boldsymbol{C} \rrbracket\,(\boldsymbol{\mathit{ff}})$ |
|---|---|
| $x \coloneqq e$ | $\lambda f \colon \reflectbox{S}\mu\colon\ \mathit{ff}(\mu) \curlywedge [\bigoplus_\alpha \mu\,[x/\alpha] \odot [x = e\,[x/\alpha]] = f]$ |
| $x \coloneqq \texttt{nondet()}$ | $\lambda f \colon \reflectbox{S}\mu\colon\ \mathit{ff}(\mu) \curlywedge [\bigoplus_\alpha \mu\,[x/\alpha] = f]$ |
| $\odot\, w$ | $\lambda f \colon \reflectbox{S}\mu\colon\ \mathit{ff}(\mu) \curlywedge [\mu \odot w = \nu]$ |
| $C_1 \fatsemi C_2$ | $\mathsf{shp}\ \llbracket C_2 \rrbracket \big(\mathsf{shp}\ \llbracket C_1 \rrbracket\,(\mathit{ff})\big)$ |
| $\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}$ | $\reflectbox{S}\mu\colon\ \mathit{ff}(\mu) \curlywedge \big(\mathsf{shp}\ \llbracket C_1 \rrbracket\,([\mu]) \boxtimes \mathsf{shp}\ \llbracket C_2 \rrbracket\,([\mu])\,\big)$ |
| $C^{\langle e, e' \rangle}$ | $\lambda f \colon \reflectbox{S}\mu\colon\ \mathit{ff}(\mu) \curlywedge \big[\big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X \colon \mathsf{trnsf}(\mathsf{sp}\,\llbracket C \rrbracket\,(X \odot \llbracket e \rrbracket)) \oplus X \odot \llbracket e' \rrbracket\big)(\mu) = f\big]$ |

**Table 5.2:** Rules for defining the quantitative strongest hyper post transformer.

subdistributions, $\mu_1$ and $\mu_2$, such that $\mu_1 \in \phi$ and $\mu_2 \in \psi$.

We are now ready to define our Quantitative Strongest Hyper Post. Similarly to our Quantitative Strongest Post defined in Section 3.4, $\mathsf{shp}\ \llbracket C \rrbracket\,(\mathit{ff}) : (\mathbb{AA} \to \mathbb{AA})$ is a function that takes as input a *final* quantity $\nu$, determines all initial quantities $\mu$ that can reach $\nu$ by executing $C$, and evaluates the hyperquantity $\mathit{ff}(\mu)$ in each of those initial quantities $\mu$, and finally returns the supremum over all these so-determined quantities. As a transformer, we obtain the following:

**Definition 5.4.3** (Quantitative Strongest Hyper Post)**.** *The* quantitative strongest hyper post transformer

$$\mathsf{shp}\colon \quad \mathsf{wReg} \to (\mathbb{AA} \to \mathbb{AA})$$

*is defined inductively according to the rules in Table 5.2.* ◁

Let us show for some of the rules how the quantitative strongest hyper post semantics can be developed and understood analogously to Dijkstra's classical strongest postconditions.

**Assignment.**

The strongest postcondition of an assignment is given by

$$\mathsf{sp}\,\llbracket x \coloneqq e \rrbracket\,(\psi) \qquad = \qquad \exists\, \alpha\colon \quad x = e\,[x/\alpha] \land \psi\,[x/\alpha] \ .$$

Intuitively, the quantified $\alpha$ represents an *initial* value that $x$ could have had *before* executing the assignment. This way, the strongest postcondition operates by retrocipating the operational semantics: it examines whether, given a final state $\tau$, the initial state $\sigma[x \mapsto \alpha]$ adheres to the precondition $\psi$.

For the *quantitative strongest hyper post*, a similar approach is taken, but we consider the strongest postcondition rather than the operational semantics. Therefore, given a final distribution (or set) $f$, the value of $\mathit{ff}$ in the initial distribution (or set of states) before the execution of $x \coloneqq e$ corresponds to $\mathit{ff}$, but evaluated at the initial distribution $\mu$ – this is possible by ensuring that $\mathsf{sp}\,[\![C]\!]\,(\mu) = f$. In general, we may have multiple initial distributions, so we take the supremum over such values, yielding the rule:

$$\begin{aligned}\mathsf{shp}\,[\![x \coloneqq e]\!]\,(\mathit{ff}) \;&=\; \lambda f \colon \reflectbox{$\mathsf{S}$}\mu\colon\; \mathit{ff}(\mu) \curlywedge [\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\mu) = f] \\ &=\; \lambda f \colon \reflectbox{$\mathsf{S}$}\mu\colon\; \mathit{ff}(\mu) \curlywedge \left[\bigoplus_{\alpha} \mu\,[x/\alpha] \odot [x = e\,[x/\alpha]] = f\right]\end{aligned}$$

**Nondeterministic Assignment:**

The nondeterministic assignment is analogous to the standard assignment, but now with $x$ ranging over any possible value. Therefore, the strongest hyper post of a nondeterministic assignment is given by:

$$\mathsf{shp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\mathit{ff}) \;=\; \lambda f \colon \reflectbox{$\mathsf{S}$}\mu\colon\; \mathit{ff}(\mu) \curlywedge \left[\bigoplus_{\alpha} \mu\,[x/\alpha] = f\right]$$

**Assume/Weighting.**

For the classical strongest postcondition, we have $\mathsf{sp}\,[\![\texttt{assume}\ \varphi]\!]\,(\psi) \;=\; \varphi \wedge \psi$. Indeed, if the final state $\tau$ satisfies the combined postcondition $\varphi \wedge \psi$, then the state is definitely reachable starting from $\tau$ itself, which satisfies the precondition $\varphi$. In fact, $\varphi \wedge \psi$ is exactly the set of reachable states starting from $\psi$.

The *quantitative strongest hyper post* for a final distribution (or set) $f$ computes the maximum value of $\mathit{ff}$ evaluated in the initial distribution (or set)

$\mu$ (i.e., such that $\mathsf{sp}\,\llbracket C \rrbracket\,(\mu) = f$), resulting in the following rule:

$$\begin{aligned}\mathsf{shp}\,\llbracket \texttt{assume}\ \varphi \rrbracket\,(\mathit{ff}) \;&=\; \lambda f\colon \reflectbox{$\mathsf{S}$}\mu\colon\; \mathit{ff}(\mu) \curlywedge [\mathsf{sp}\,\llbracket \texttt{assume}\ \varphi \rrbracket\,(\mu) = f] \\ &=\; \lambda f\colon \reflectbox{$\mathsf{S}$}\mu\colon\; \mathit{ff}(\mu) \curlywedge [\mu \odot [\varphi] = f]\end{aligned}$$

For the more general weighting statement, we obtain the rule:

$$\mathsf{shp}\,\llbracket \odot\, w \rrbracket\,(\mathit{ff}) \;=\; \lambda f\colon \reflectbox{$\mathsf{S}$}\mu\colon\; \mathit{ff}(\mu) \curlywedge [\mu \odot w = f]\ ,$$

where $w$ can be any quantity.

**Nondeterministic Choice.**

The *angelic* strongest post*condition* of $\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}$ is given by

$$\mathsf{sp}\,\llbracket \{\, C_1 \,\} \,\Box\, \{\, C_2 \,\} \rrbracket\,(\psi) \;=\; \mathsf{sp}\,\llbracket C_1 \rrbracket\,(\psi) \;\vee\; \mathsf{sp}\,\llbracket C_2 \rrbracket\,(\psi)\ .$$

Indeed, the set of reachable states starting from initial states satisfying $\psi$ is the union of the reachable set after executing $C_1$ and the ones after executing $C_2$.

Moving to hyperquantities, we want to retrocipate the value of a hyperquantity $\mathit{ff}$ before executing either $C_1$ or $C_2$. A first attempt would be:

$$\mathsf{shp}\,\llbracket \{\, C_1 \,\} \,\Box\, \{\, C_2 \,\} \rrbracket\,(\mathit{ff}) \;=\; \lambda f\colon \big(\mathsf{shp}\,\llbracket C_1 \rrbracket\,(\mathit{ff}) \boxtimes \mathsf{shp}\,\llbracket C_2 \rrbracket\,(\mathit{ff})\big)(f)\ .$$

However, as discussed by Zilberstein [74, p. 7], this rule is sound, but not complete: it does not account for the relationships between the two branches. The correct rule considers every possible partition of $f$ into two parts, $f_1$ and $f_2$, such that $f_1$ is reachable by executing $C_1$ and $f_2$ is reachable by executing $C_2$ from the *common* initial distribution (or set) $\mu$. This leads to the following rule:

$$\begin{aligned}&\mathsf{shp}\,[\![\{\,C_1\,\}\,\square\,\{\,C_2\,\}]\!]\,(\mathit{ff})\\&\quad=\;\lambda f\colon \mathcal{S}\mu\colon\; \mathit{ff}(\mu)\curlywedge\Big(\bigcurlyvee_{f_1,f_2\colon f_1\oplus f_2=f}\mathsf{shp}\,[\![C_1]\!]\,([\mu])\,(f_1)\curlywedge\mathsf{shp}\,[\![C_2]\!]\,([\mu])\,(f_2)\Big)\;.\end{aligned}$$

This means that $\mu$ is the initial distribution (or set) that leads to $f_1$ after executing $C_1$ and to $f_2$ after executing $C_2$, with their semiring sum being $f$. In other words, $\mu$ is a valid initial distribution (or set) for the nondeterministic choice $\{\,C_1\,\}\,\square\,\{\,C_2\,\}$. We then take the supremum to compute the maximum value of $\mathit{ff}$ over all possible initial distributions $\mu$.

By employing the quantitative outcome conjunction, we obtain the syntactically simpler rule:

$$\mathsf{shp}\,[\![\{\,C_1\,\}\,\square\,\{\,C_2\,\}]\!]\,(\mathit{ff})\;=\;\mathcal{S}\mu\colon\;\mathit{ff}(\mu)\curlywedge\big(\mathsf{shp}\,[\![C_1]\!]\,([\mu])\boxtimes\mathsf{shp}\,[\![C_2]\!]\,([\mu])\big)\;.$$

**Remark 5.4.1.** *Similarly to our strongest hyper post,* OL *and* HHL *also have a forward-style rule for* $\{\,C_1\,\}\,\square\,\{\,C_2\,\}$*, which utilizes the outcome conjunction. While these rules preserve soundness, completeness requires an existential rule, which we introduce by quantifying over* $\mu$*. Our sound and complete rule has already been studied by Zilberstein [74, p. 7], but in a classical (non-quantitative) setting.*

**Sequential Composition.**

What is the retrocipated value of $\mathit{ff}$ before executing $C_1\,⨾\,C_2$? For this, we first retrocipate the value of $\mathit{ff}$ before executing $C_1$ which gives $\mathsf{shp}\,[\![C_1]\!]\,(\mathit{ff})$. Then, we retrocipate the value $\mathsf{shp}\,[\![C_1]\!]\,(\mathit{ff})$ before executing $C_2$, yielding:

$$\mathsf{shp}\,[\![C_1\,⨾\,C_2]\!]\,(\mathit{ff})=\mathsf{shp}\,[\![C_2]\!]\,(\mathsf{shp}\,[\![C_1]\!]\,(\mathit{ff}))\;\;.$$

**Iteration**

The rule for $C^{\langle e,e'\rangle}$ is derived by retrocipating the execution of $C^{\langle e,e'\rangle}$. Specifically, for a given distribution (or set) $f$, we aim to find the maximum value of a hyperquantity $\mathit{ff}$ computed in an initial quantity $\mu$ that terminates in $f$, resulting in the following rule:

$$\begin{aligned}&\mathsf{shp}\ \left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\mathit{ff})\\ &\quad= \lambda f\colon \text{Ƨ}\mu\colon \mathit{ff}(\mu) \text{⅄} \left[\big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(\mu) = f\right]\end{aligned}$$

Similarly to $\mathsf{whp}$, it is consistent in the sense that it is a solution of the equation:

$$\begin{aligned}&\mathsf{shp}\ \left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right]\\ &\quad= \mathsf{shp}\ \left[\!\!\left[\left\{\,\odot\, e\,;\, C\,;\, C^{\langle e,e'\rangle}\right\} \Box \left\{\,\odot\, e'\,\right\}\right]\!\!\right]\\ &\quad= \lambda \mathit{hh}\,\lambda f\colon\\ &\qquad\quad \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \text{⅄} \Big(\mathsf{shp}\ \left[\!\!\left[\odot\, e\,;\, C\,;\, C^{\langle e,e'\rangle}\right]\!\!\right]([\mu]) \boxtimes \mathsf{shp}\ [\![\odot\, e']\!]\,([\mu])\Big)(f)\\ &\quad= \lambda \mathit{hh}\,\lambda f\colon\\ &\qquad\quad \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \text{⅄} \Big(\mathsf{shp}\ \left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\mathsf{shp}\ [\![C]\!]\,(\mathsf{shp}\ [\![\odot\, e]\!]\,([\mu]))) \boxtimes \mathsf{shp}\ [\![\odot\, e']\!]\,([\mu])\Big)(f)\end{aligned}$$

Indeed one can show the following.

**Proposition 5.4.2** (Consistency of iteration rule)**.** *Let*

$$\begin{aligned}&\Psi(\mathit{trnsf})\\ &= \lambda \mathit{hh}\,\lambda f\colon \text{Ƨ}\mu\colon\ \mathit{hh}(\mu) \text{⅄} \Big(\mathit{trnsf}(\mathsf{shp}\ [\![C]\!]\,(\mathsf{shp}\ [\![\odot\, e]\!]\,([\mu]))) \boxtimes \mathsf{shp}\ [\![\odot\, e']\!]\,([\mu])\Big)(f)\end{aligned}$$

*Then,* $\mathsf{shp}\ [\![C^{\langle e,e'\rangle}]\!]$ *is a fixpoint of the higher order function* $\Psi(\mathit{trnsf})$*, that is:*

$$\begin{aligned}&\Psi\Big(\lambda \mathit{ff}\,\lambda f\colon \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \text{⅄} \left[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu) = f\right]\Big)\\ &\quad= \lambda \mathit{ff}\,\lambda f\colon \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \text{⅄} \left[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu) = f\right]\end{aligned}$$

After having provided an intuition on the rules, let us show that $\mathsf{shp}$ does actually retrocipate $\mathsf{sp}$.

**Theorem 5.4.3** (Characterization of $\mathsf{shp}$)**.** *For all programs $C$, hyperquantities $\mathit{ff} \in \mathbb{AA}$ and quantities $f \in \mathbb{A}$*

$$\mathsf{shp}\,[\![C]\!]\,(\mathit{ff})\,(f) \quad = \quad \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C]\!](\mu)=f} \quad \mathit{ff}(\mu)\ .$$

For a given hyperquantity $\mathit{ff}$ and final quantity $f$, $\mathsf{shp}\,[\![C]\!]\,(\mathit{ff})\,(f)$ is the *supremum* over all the values that $\mathit{ff}$ can assume in those initial quantities $\mu$ from which executing $C$ terminates in $f$. Similarly to our quantitative strongest post, in case that the final quantity $f$ is *unreachable*, i.e. $\forall \mu\colon \mathsf{sp}\,[\![C]\!]\,(\mu) \neq f$, that supremum automatically becomes $\mathbb{0}$ — the absolute minimal value. In particular, if $\forall\, f\colon\ \mathit{ff}(f) \neq \mathbb{0}$, then $\mathsf{shp}\,[\![C]\!]\,(\mathit{ff})\,(f) = \mathbb{0}$ unambiguously indicates *unreachability* of $f$ by executing $C$ on any input $\mu$.

## 5.5 Quantitative Strongest Liberal Hyper Post

In Section 5.6, we will demonstrate that $\mathsf{whp}$ has no liberal variant because it already provides a precise anticipation of the strongest postcondition $\mathsf{sp}$. However, this one-to-one correspondence does not extend to $\mathsf{shp}$. The inverse operation—computing the initial quantities that produce a specific final quantity through $\mathsf{sp}$ execution—is inherently non-unique. This is because multiple distinct initial quantities $\mu$ can lead to the same final quantity $f$ when transformed by $\mathsf{sp}[\![C]\!]$.

Drawing inspiration from our Quantitative Strongest Liberal Post defined in Section 3.4, we propose a natural liberal variant of $\mathsf{shp}$ by taking the *infimum* over all evaluations of the hyperquantity $\mathit{ff}$. This approach yields a novel *strongest liberal hyper postcondition transformer* denoted $\mathsf{slhp}\,[\![C]\!]$ with type

$$\mathsf{slhp}\,[\![C]\!] : \mathbb{BB} \to \mathbb{BB}$$

associating to each program $C$ a mapping from hyperproperties to hyperproperties. Since $\mathsf{slhp}$ is associated with the infimum, we will consider a *demonic* setting, where the nondeterminism is resolved to our *dis*advantage. Analogously to $\mathsf{slp}$, strongest liberal post deems *unreachability* good behavior.

Specifically, the *demonic* strongest liberal hyper post transformer $\mathsf{slhp}\,[\![C]\!]$ maps a hyper *pre*condition $\not{\phi}$ over initial preconditions to a hyper *post*condition $\mathsf{slhp}\,[\![C]\!]\,(\not{\phi})$ over final postconditions, such that for a given final postcondition $\psi$ satisfying $\mathsf{slhp}\,[\![C]\!]\,(\not{\phi})$, all initial preconditions that reach $\psi$ satisfy the hyper precondition $\not{\phi}$. More symbolically, recalling that $\mathsf{sp}\,[\![C]\!]\,(\phi)$ is the set of all final states reachable after termination of $C$ on all states in $\phi$,

$$\psi \;\models\; \mathsf{slhp}\,[\![C]\!]\,(\not{\phi}) \qquad \text{iff} \qquad \forall\,\phi \text{ with } \mathsf{sp}\,[\![C]\!]\,(\phi) = \psi\colon \quad \phi \;\models\; \not{\phi}\ ,$$

where the right-hand-side of the implication is vacuously true if $\psi$ is unreachable. This is in constrast with $\mathsf{shp}$, where if restricted to a boolean setting, it is associated with the existential quantifier, and thus deems *unreachability* bad behavior:

$$\psi \;\models\; \mathsf{shp}\,[\![C]\!]\,(\not{\phi}) \qquad \text{iff} \qquad \exists\,\phi \text{ with } \mathsf{sp}\,[\![C]\!]\,(\phi) = \psi\colon \quad \phi \;\models\; \not{\phi}\ .$$

From a map perspective, $\mathsf{slhp}\,[\![C]\!]\,(\not{\phi})\,(\psi)$ identifies all initial preconditions $\phi$ that result in the postcondition $\psi$ after executing $C$ (i.e., $\mathsf{sp}\,[\![C]\!]\,(\phi) = \psi$). It then checks if these preconditions satisfy the hyperprecondition $\not{\phi}$, effectively determining whether the hyperpredicate $\psi$ *must* have been true before executing $C$, thereby <u>retro</u>cipating the truth of $\not{\phi}$. Formally, we have:

$$\mathsf{slhp}\,[\![C]\!]\,(\not{\phi})\,(\psi) \quad = \bigwedge_{\phi \text{ with } \mathsf{sp}\,[\![C]\!](\phi) = \psi} \not{\phi}(\phi)\ ,$$

where the conjunction over an empty set is defined — as is standard — as $\mathsf{true}$. For quantities, we essentially replace $\wedge$ by $\curlywedge$ and define the following quantitative strongest liberal hyper post transformer:

| $\boldsymbol{C}$ | $\textsf{\textbf{slhp}}\,[\![\boldsymbol{C}]\!]\,(\boldsymbol{\mathit{ff}})$ |
|---|---|
| $x \coloneqq e$ | $\lambda f\colon \boldsymbol{\iota}\,\mu\colon\; \mathit{ff}(\mu) \curlyvee [\bigoplus_\alpha \mu\,[x/\alpha] \odot [x = e\,[x/\alpha]] \neq f]$ |
| $x \coloneqq \texttt{nondet()}$ | $\lambda f\colon \boldsymbol{\iota}\,\mu\colon\; \mathit{ff}(\mu) \curlyvee [\bigoplus_\alpha \mu\,[x/\alpha] \neq \nu]$ |
| $\odot\, w$ | $\lambda f\colon \boldsymbol{\iota}\,\mu\colon\; \mathit{ff}(\mu) \curlyvee [\mu \odot w \neq f]$ |
| $C_1 \fatsemi C_2$ | $\textsf{slhp}\;[\![C_2]\!]\,\big(\textsf{slhp}\;[\![C_1]\!]\,(\mathit{ff})\big)$ |
| $\{\,C_1\,\} \,\square\, \{\,C_2\,\}$ | $\boldsymbol{\iota}\,\mu\colon\; \mathit{ff}(\mu) \curlyvee \big(\textsf{slhp}\;[\![C_1]\!]\,([\neg\mu]) \boxplus \textsf{slhp}\;[\![C_2]\!]\,([\neg\mu])\,\big)$ |
| $C^{\langle e,e'\rangle}$ | $\lambda f\colon \boldsymbol{\iota}\,\mu\colon \mathit{ff}(\mu) \curlyvee \big[\big(\textsf{lfp trnsf}\colon \lambda X\colon \textsf{trnsf}(\textsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(\mu) \neq f\big]$ |

**Table 5.3:** Rules for defining the quantitative strongest hyper post transformer.

**Definition 5.5.1** (Quantitative Strongest Liberal Hyper Post)**.** *The* quantitative strongest liberal hyper post transformer

$$\textsf{slhp}\colon \quad \textsf{wReg} \to (\mathbb{AA} \to \mathbb{AA})$$

*is defined inductively according to the rules in Table 5.3.* ◁

Let us thus go over the language constructs where the rules for $\textsf{slhp}$ and $\textsf{shp}$ differ and explain both strongest liberal hyper post and quantitative strongest liberal hyper post.

**Assignment.**

The strongest liberal post of an assignment is given by

$$\textsf{slp}[\![x \coloneqq e]\!]\,(\psi) \qquad = \qquad \forall\,\alpha\colon \quad \underbrace{x \neq e\,[x/\alpha]}_{(1)} \;\vee\; \underbrace{\psi\,[x/\alpha]}_{(2)}\;.$$

Intuitively, the disjunction can be interpreted as an implication stating that if $\alpha$ is a valid initial candidate value for $x$ before the assignment, then the precondition $\psi$ evaluated on the initial state before the assignment must hold. In other words, $\textsf{slp}$ retrocipates the operational semantics of the assignment, to compute all initial states, and verify if the precondition hold on all such states.

For the *quantitative strongest liberal hyper post*, we retrocipate the strongest postcondition rather than the operational semantics. Therefore, given a final distribution (or set) $f$, the value of $\mathit{ff}$ in the initial distribution (or set of states) before the execution of $x \coloneqq e$ corresponds to $\mathit{ff}$, but evaluated at the initial distribution $\mu$ – this is possible by ensuring that $\mathsf{sp}\,[\![C]\!]\,(\mu) = f$. In general, we may have multiple initial distributions, so differently to $\mathsf{whp}$, we take the infimum over such values, yielding the rule:

$$\begin{aligned}\mathsf{shp}\,[\![x \coloneqq e]\!]\,(ff) &= \lambda f\colon \text{\reflectbox{$\mathsf{L}$}}\,\mu\colon\; ff(\mu) \curlyvee [\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\mu) \neq f] \\ &= \lambda f\colon \text{\reflectbox{$\mathsf{L}$}}\,\mu\colon\; ff(\mu) \curlyvee \left[\bigoplus_{\alpha} \mu\,[x/\alpha] \odot [x = e\,[x/\alpha]] \neq f\right] ,\end{aligned}$$

where $[\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\mu) \neq f]$ ensures that initial quantities $\mu$ for which $\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\mu) = f$ are excluded from the infimum, unless $f$ is not reachable, in which case we yield $+\infty$.

**Nondeterministic Assignment:**

The quantitative strongest liberal hyper post of a nondeterministic assignment is given by:

$$\mathsf{slhp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(ff) \;=\; \lambda f\colon \text{\reflectbox{$\mathsf{L}$}}\,\mu\colon\; ff(\mu) \curlyvee \left[\bigoplus_{\alpha} \mu\,[x/\alpha] \neq f\right] ,$$

which is a straightforward extension of the assignment rule by taking into account that now $x$ could be assigned to any value of $\alpha$.

**Assume/Weighting.**

For the *quantitative strongest liberal hyper post*, for a final distribution (or set) $f$ it computes the minimum value of $ff$ evaluated in the initial distribution (or set) $\mu$ (i.e., such that $\mathsf{sp}\,[\![C]\!]\,(\mu) = f$), resulting in the following rule:

$$\begin{aligned}\mathsf{slhp}\,[\![\texttt{assume}\;\varphi]\!]\,(ff) &= \lambda f\colon \text{\reflectbox{$\mathsf{L}$}}\,\mu\colon\; ff(\mu) \curlyvee [\mathsf{sp}\,[\![\texttt{assume}\;\varphi]\!]\,(\mu) \neq f] \\ &= \lambda f\colon \text{\reflectbox{$\mathsf{L}$}}\,\mu\colon\; ff(\mu) \curlyvee [\mu \odot [\varphi] \neq f]\end{aligned}$$

For the more general weighting statement, we obtain the rule:

$$\mathsf{slhp}\,[\![\odot\, w]\!]\,(ff) \;=\; \lambda f\colon \text{\reflectbox{$\mathsf{L}$}}\,\mu\colon\; ff(\mu) \curlyvee [\mu \odot w \neq f] ,$$

where $w$ can be any quantity.

**Nondeterministic Choice.**

For the nondeterministic choice, the quantitative strongest liberal hyper post computes the infimum of all $\mathit{ff}(\mu)$ where $\mu$ is an initial distribution (or set) that leads to a final distribution (or set) $f$ after executing $\{ C_1 \} \square \{ C_2 \}$. More precisely, the rule is:

$$
\begin{aligned}
&\mathsf{slhp}\ [\![\{ C_1 \} \square \{ C_2 \}]\!]\ (\mathit{ff}) \\
&= \curlywedge \mu \colon \lambda f \colon \mathit{ff}(\mu) \curlyvee \underbrace{\Big( \bigcurlywedge_{f_1, f_2 \colon f_1 \oplus f_2 = f} \mathsf{slhp}\ [\![C_1]\!]\ ([\neg \mu])\ (f_1) \curlyvee \mathsf{slhp}\ [\![C_2]\!]\ ([\neg \mu])\ (f_2) \Big)}_{(1)}
\end{aligned}
$$

If $\mu$ is not a valid initial distribution that terminates in $f$, then for every partition of $f$ into $f_1$ and $f_2$, either $\mu$ will not reach $f_1$ or $\mu$ will not reach $f_2$ (or both). This is exactly represented by the braced expression (1), which in this case will be evaluated to $+\infty$, and hence such $\mu$ will not be considered in the infimum. While for $\mathsf{shp}$ we used the outcome conjunction to derive a syntactically simpler rule, here for $\mathsf{slhp}$ we define a novel connective, namely the quantitative outcome disjunction.

**Definition 5.5.2** (Quantitative Outcome Disjunction)**.** *The quantitative outcome disjunction $\boxplus$ is defined as follows:*

$$
(\mathit{ff} \boxplus \mathit{gg})(\nu) \triangleq \bigcurlywedge_{\mu_1, \mu_2 \in \mathbb{A} \colon \mu_1 \oplus \mu_2 = \nu} \mathit{ff}(\mu_1) \curlyvee \mathit{gg}(\mu_2)
$$

◁

Unsurprisingly, there is a duality between the quantitative outcome disjunction $\boxplus$ and the quantitative outcome conjunction $\boxtimes$.

**Theorem 5.5.1** (Duality of Quantitative Outcome Conjunction and Disjunction)**.** *The quantitative outcome disjunction $\boxplus$ is the De Morgan dual of* *the quantitative outcome conjunction $\boxtimes$. Formally, for any hyperquantities $ff, gg \in \mathbb{AA}$,*

$$-(ff \boxtimes gg) = (-ff) \boxplus (-gg) \ .$$

*Proof.* For all $\nu \in \mathbb{A}$ we have:

$$\begin{aligned}
& -(ff \boxtimes gg)(\nu) \\
= & - \bigcurlyvee_{\mu_1,\mu_2 \in \mathbb{A}\colon\, \mu_1 \oplus \mu_2 = \nu} ff(\mu_1) \curlywedge gg(\mu_2) \\
= & \bigcurlywedge_{\mu_1,\mu_2 \in \mathbb{A}\colon\, \mu_1 \oplus \mu_2 = \nu} -(ff(\mu_1) \curlywedge gg(\mu_2)) \\
= & \bigcurlywedge_{\mu_1,\mu_2 \in \mathbb{A}\colon\, \mu_1 \oplus \mu_2 = \nu} (-ff(\mu_1)) \curlyvee (-gg(\mu_2)) \\
= & (-ff \boxplus -gg)(\nu) \ .
\end{aligned}$$

□

By employing the quantitative outcome disjunction, we obtain the rule:

$$\mathsf{slhp}\, [\![\{ C_1 \} \square \{ C_2 \}]\!]\, (ff) = \curlywedge \mu\colon\ ff(\mu) \curlyvee \big(\mathsf{slhp}\, [\![C_1]\!]\, ([\neg\mu]) \boxplus \mathsf{slhp}\, [\![C_2]\!]\, ([\neg\mu])\big)$$

**Iteration**

Similarly to $\mathsf{shp}$, the rule for $C^{\langle e,e'\rangle}$ is derived by retrocipating the execution of $C^{\langle e,e'\rangle}$. Specifically, for a given distribution (or set) $f$, we aim to find the minimum value of a hyperquantity $ff$ computed in an initial quantity $\mu$ that terminates in $f$, resulting in the following rule:

$$\begin{aligned}
& \mathsf{slhp}\, \left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right] (ff) \\
& = \lambda f\colon \curlywedge \mu\colon ff(\mu) \curlyvee \underbrace{\left[\big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\, [\![C]\!]\, (X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(\mu) \neq f\right]}_{(1)} \ ,
\end{aligned}$$

where the underlined expression (1) ensures that the final distribution (or set) $f$ is reachable from the initial distribution (or set) $\mu$ after executing $C^{\langle e,e'\rangle}$, and if not, it yields $+\infty$ which is ignored by the infimum. We also show that $\mathsf{slhp}$ is consistent in the sense that it is a solution of the equation:

$$
\begin{aligned}
&\mathsf{slhp}\left\llbracket C^{\langle e,e'\rangle}\right\rrbracket\\
&\quad= \mathsf{slhp}\left\llbracket\left\{\odot\, e \fatsemi C \fatsemi C^{\langle e,e'\rangle}\right\} \square \left\{\odot\, e'\right\}\right\rrbracket\\
&\quad= \lambda \mathit{hh}\, \lambda f\colon \boldsymbol{\iota}\, \mu\colon\\
&\qquad\qquad \mathit{ff}(\mu) \curlyvee \big(\mathsf{slhp}\left\llbracket \odot\, e \fatsemi C \fatsemi C^{\langle e,e'\rangle}\right\rrbracket ([\neg\mu]) \boxplus \mathsf{shp}\left\llbracket \odot\, e'\right\rrbracket ([\neg\mu])\big)(f)\\
&\quad= \lambda \mathit{hh}\, \lambda f\colon \boldsymbol{\iota}\, \mu\colon\\
&\qquad\qquad \mathit{ff}(\mu) \curlyvee \big(\mathsf{slhp}\left\llbracket C^{\langle e,e'\rangle}\right\rrbracket (\mathsf{slhp}\llbracket C\rrbracket (\mathsf{slhp}\llbracket \odot\, e\rrbracket ([\neg\mu]))) \boxplus \mathsf{slhp}\left\llbracket \odot\, e'\right\rrbracket ([\neg\mu])\big)(f)
\end{aligned}
$$

Indeed one can show the following.

**Proposition 5.5.2** (Consistency of iteration rule)**.** *Let*

$$
\begin{aligned}
\Psi(\mathit{trnsf}) \;=\; & \lambda \mathit{hh}\, \lambda f\colon \boldsymbol{\iota}\, \mu\colon\\
& \qquad \mathit{hh}(\mu) \curlyvee \big(\mathit{trnsf}(\mathsf{slhp}\llbracket C\rrbracket (\mathsf{shp}\llbracket \odot\, e\rrbracket ([\neg\mu]))) \boxplus \mathsf{shp}\llbracket \odot\, e'\rrbracket ([\neg\mu])\big)(f)
\end{aligned}
$$

*Then,* $\mathsf{slhp}\llbracket C^{\langle e,e'\rangle}\rrbracket$ *is a fixpoint of the higher order function* $\Psi(\mathit{trnsf})$*, that is:*

$$
\begin{aligned}
&\Psi\Big(\lambda \mathit{ff}\, \lambda f\colon \boldsymbol{\iota}\, \mu\colon\; \mathit{ff}(\mu) \curlyvee \left[\mathsf{sp}\llbracket C^{\langle e,e'\rangle}\rrbracket (\mu) \neq f\right]\Big)\\
&\quad= \lambda \mathit{ff}\, \lambda f\colon \boldsymbol{\iota}\, \mu\colon\; \mathit{ff}(\mu) \curlyvee \left[\mathsf{sp}\llbracket C^{\langle e,e'\rangle}\rrbracket (\mu) \neq f\right]
\end{aligned}
$$

After having provided an intuition on the rules, let us show that $\mathsf{slhp}$ does actually retrocipate $\mathsf{sp}$.

**Theorem 5.5.3** (Characterization of $\mathsf{slhp}$)**.** *For all programs $C$, hyperquantities $\mathit{ff} \in \mathbb{AA}$ and quantities $f \in \mathbb{A}$*

$$
\mathsf{slhp}\llbracket C\rrbracket (\mathit{ff})(f) \quad = \quad \bigcurlywedge_{\mu\colon \mathsf{sp}\llbracket C\rrbracket(\mu)=f} \mathit{ff}(\mu)\ .
$$

For a given hyperquantity $\mathit{ff}$ and final quantity $f$, $\mathsf{slhp}\llbracket C\rrbracket (\mathit{ff})(f)$ is the *infimum* over all the values that $\mathit{ff}$ can assume in those initial quantities $\mu$ from which executing $C$ terminates in $f$. Similarly to our quantitative strongest liberal post, in case that the final quantity $f$ is *unreachable*, i.e. $\forall \mu\colon \mathsf{sp}\,⟦C⟧\,(\mu) \neq f$, that infimum automatically becomes $\mathbb{1}$ — the absolute maximal value. In particular, if $\forall\, f\colon ff(f) \neq \mathbb{1}$, then $\mathsf{shp}\ ⟦C⟧\,(ff)\,(f) \;=\; \mathbb{1}$ unambiguously indicates *unreachability* of $f$ by executing $C$ on any input $\mu$.

## 5.6 Properties

### 5.6.1 Hyper Galois Connections

The classical strongest postcondition is the left adjoint to the weakest liberal precondition [64, Section 12], i.e. the transformers wlp and sp form the Galois connection

$$G \;\Longrightarrow\; \mathsf{wlp}⟦C⟧\,(F) \qquad \text{iff} \qquad \mathsf{sp}\,⟦C⟧\,(G) \;\Longrightarrow\; F\;, \tag{†}$$

which intuitively is true because $G \;\Longrightarrow\; \mathsf{wlp}\,⟦C⟧\,(F)$ means that starting from $G$ the program $C$ will either diverge or terminate in a state satisfying $F$, and $\mathsf{sp}\,⟦C⟧\,(G) \;\Longrightarrow\; F$ means that starting from $G$ any state reachable by executing $C$ satisfies $F$. Quantitative Galois connections between forward and backward transformers have been studied in [17]. We show that the above Galois connection is preserved in our hyper quantitative setting; in fact, by substituting the partial order $\Longrightarrow$ on predicates with the partial order $\preceq$ on $\mathbb{AA}$ we obtain:

**Theorem 5.6.1** (Hyper Galois Connection)**.** *For all $C \in \mathsf{Reg}$ and $gg, ff \in \mathbb{AA}$:*

$$\begin{aligned} gg \;\preceq\; \mathsf{whp}\ ⟦C⟧\,(ff) \qquad &\text{iff} \qquad \mathsf{shp}\ ⟦C⟧\,(gg) \;\preceq\; ff\;, \\ \mathsf{whp}\ ⟦C⟧\,(ff) \;\preceq\; gg \qquad &\text{iff} \qquad ff \;\preceq\; \mathsf{slhp}\ ⟦C⟧\,(gg)\;. \end{aligned}$$

It is noteworthy that whp functions as both a left and right adjoint in these Galois connections. This dual role stems from the unique nature of whp, which behaves simultaneously as both a liberal and non-liberal transformer. In fact, according to Theorem 5.3.2, $\mathsf{whp}\,[\![C]\!]\,(\mathit{ff})\,(\mu)$ precisely computes $\mathsf{sp}\,[\![C]\!]\,(\mu)$ enabling it to capture both suprema and infima of the singleton $\{\mathsf{sp}\,[\![C]\!]\,(\mu)\}$. We will continue to demonstrate this in the remainder of this section.

### 5.6.2 Relationship between Liberal and Non-liberal Transformers

Various dualities between $\mathsf{wp}$ and $\mathsf{wlp}$ have been extensively explored in the literature. In Dijkstra's classical calculus, the duality relationship is expressed as $\mathsf{wp}\,[\![C]\!]\,(\psi) = \neg\mathsf{wlp}[\![C]\!]\,(\neg\psi)$. In quantitative settings, particularly in the work of Kozen and McIver & Morgan on probabilistic programs, this duality extends to $\mathsf{wp}\,[\![C]\!]\,(f) = 1 - \mathsf{wlp}[\![C]\!]\,(1-f)$ for 1-bounded functions $f$. This concept is further generalized to $\mathsf{wp}\,[\![C]\!]\,(f) = -\mathsf{wlp}[\![C]\!]\,(-f)$ in the case of non-probabilistic programs and unbounded quantities, as demonstrated in Zhang and Kaminski [17, Theorem 5.3], which also showcases dualities between strongest post transformers: $\mathsf{sp}\,[\![C]\!]\,(f) = -\mathsf{slp}[\![C]\!]\,(-f)$.

Our $\mathsf{whp}$ calculus exhibits a unique duality property that distinguishes it from classical predicate transformers. Unlike traditional calculi that are either liberal or non-liberal, $\mathsf{whp}$ exhibits characteristics of both, as demonstrated by the following duality relationship:

**Theorem 5.6.2** (Liberal–Non-liberal Duality, $\mathsf{whp}$)**.** *For any program $C$ and any $k$-bounded hyperquantity $\mathit{ff}$, we have* $\mathsf{whp}\,[\![C]\!]\,(\mathit{ff}) \;=\; k - \mathsf{whp}\,[\![C]\!]\,(k - \mathit{ff})$.

Unsurprisingly, our strongest hyper post calculi enjoy the following duality.

**Theorem 5.6.3** (Liberal–Non-liberal Duality, $\mathsf{shp}$ and $\mathsf{slhp}$)**.** *For any program $C$ and hyperquantity $\mathit{ff}$, we have*

$$\mathsf{shp}\,[\![C]\!]\,(\mathit{ff}) \;=\; -\,\mathsf{slhp}\,[\![C]\!]\,(-\mathit{ff})\,.$$

As a consequence of the liberal–non-liberal duality of Theorems 5.6.2 and 5.6.3, for hyperproperties we have:

$$\phi\!\!\!/ \;\Longrightarrow\; \mathsf{whp}\,[\![C]\!]\,(\psi\!\!\!/) \qquad \text{iff} \qquad \mathsf{whp}\,[\![C]\!]\,(\neg\psi\!\!\!/) \;\Longrightarrow\; \neg\phi\!\!\!/.$$

### 5.6.3 Healthiness Properties

Similarly to Sections 3.5 and 4.7, our quantitative hyper transformers also exhibit several *healthiness properties*, some of which are analogous to those found in the calculi of Dijkstra, Kozen, and McIver & Morgan.

We will begin by showcasing the properties that $\mathsf{whp}$ enjoys, arguing that there exists only one backward hyperpredicate transformer. This is because $\mathsf{whp}$ possesses several properties and dualities that are characteristic of both liberal and non-liberal weakest precondition calculi. Following this, we will explore our strongest hyper post transformers, providing motivation for why the characterization of $\mathsf{shp}$ and $\mathsf{slhp}$ is consistent.

**Theorem 5.6.4** (Healthiness Properties of $\mathsf{whp}$)**.** *For all programs $C$, $\mathsf{whp}\,[\![C]\!]$ satisfies the following properties:*

1. *Monotonicity:* $\quad \mathit{ff} \preceq \mathit{gg} \quad$ implies $\quad \mathsf{whp}\,[\![C]\!]\,(\mathit{ff}) \preceq \mathsf{whp}\,[\![C]\!]\,(\mathit{gg})$ .

2. *Quantitative universal conjunctiveness and disjunctiveness: For any set of hyperquantities $S \subseteq \mathbb{AA}$,*

$$\begin{aligned} \mathsf{whp}\,[\![C]\!]\,(\curlywedge S) &= \bigcurlywedge_{\mathit{f} \in S} \mathsf{whp}\,[\![C]\!]\,(\mathit{ff}) \textit{ and} \\ \mathsf{whp}\,[\![C]\!]\,(\curlyvee S) &= \bigcurlyvee_{\mathit{f} \in S} \mathsf{whp}\,[\![C]\!]\,(\mathit{ff}) \end{aligned}$$

3. *$k$-Strictness:* $\quad$ *For any $k \in \mathbb{R}^{\pm\infty}$,* $\mathsf{whp}\,[\![C]\!]\,(\lambda f\colon k) \;=\; \lambda f\colon k.$

$\lhd$

Quantitative universal conjunctiveness and strictness in the context of $\mathsf{wp}$, as well as the notions of disjunctiveness and co-strictness for $\mathsf{wlp}$, serve as quantitative analogues of Dijkstra and Scholten's original calculi. These properties have been explored in [17, Section 5.1]. We demonstrate that $\mathsf{whp}$ exhibits all these characteristics, as the $k$-strictness of $\mathsf{whp}$ implies both strictness and co-strictness. This observation aligns with our intuition that $\mathsf{whp}$ functions as both a liberal and a non-liberal calculus.

On the other hand, our Strongest Post calculi exhibit similar healthiness properties to those in Section 3.5, differentiating between liberal and non-liberal. More precisely, our non-libel transformers $\mathsf{sp}, \mathsf{shp}$ enjoy of disjunctiveness and strictness, whereas our liberal ones $\mathsf{slp}, \mathsf{slhp}$ enjoy of the conjunctiveness and strictness.

**Theorem 5.6.5** (Healthiness Properties of $\mathsf{shp}$, $\mathsf{slhp}$)**.** *Both* $\mathsf{shp}$ *and* $\mathsf{slhp}$ *satisfy the following property:*

5. *Monotonicity:*

$$ff \;\preceq\; gg \qquad \text{implies} \qquad \mathsf{ttt}\,[\![C]\!]\,(ff) \;\preceq\; \mathsf{ttt}\,[\![C]\!]\,(gg)\;, \qquad \text{for } \mathsf{ttt} \in \{\mathsf{shp},\, \mathsf{slhp}\}$$

*For all programs $C$,* $\mathsf{shp}$ *satisfies the following properties:*

1. *Quantitative universal disjunctiveness: For any set of hyperquantities $S \subseteq \mathbb{AA}$,*

$$\mathsf{shp}\;[\![C]\!]\,(\curlyvee S) \;=\; \bigcurlyvee_{\boldsymbol{f} \in S} \mathsf{shp}\;[\![C]\!]\,(ff)\;\;.$$

2. *Strictness:*

$$\mathsf{slhp}\;[\![C]\!]\,(\lambda f\colon -\infty) \;=\; \lambda f\colon -\infty\;.$$

*The liberal transformer* $\mathsf{slhp}\,[\![C]\!]$ *satisfies the following properties:*

1. *Quantitative universal conjunctiveness: For any set of hyperquantities $S \subseteq \mathbb{AA}$,*

$$\mathsf{slhp}\;[\![C]\!]\,(\curlywedge S) \;=\; \bigcurlywedge_{\boldsymbol{f} \in S} \mathsf{slhp}\;[\![C]\!]\,(ff)\;\;.$$

2. *Co-strictness:*

$$\mathsf{slhp}\;[\![C]\!]\,(\lambda f\colon +\infty) \;=\; \lambda f\colon +\infty\;.$$

$\triangleleft$

Healthiness properties are mainly beneficial for conducting compositional proofs. Some of the key properties are outlined below:

### Monotonicity

Larger (hyper)quantities as inputs yield larger (hyper)quantity as results. Monotonicity is a fundamental property that allows compositional reasoning and, for classical predicate transformers, it is closely related to the rule of consequence in Hoare logic. An in-depth treatment of this particular connection can be found in Kaminski [16, p.95]. In our context, unsurprisingly, monotonicity enables the proof of the *Cons* rule from Dardinier and Müller [73, Fig. 2].

### Continuity, Disjunctiveness and Conjunctiveness

This property allows a complex (hyper)property to be broken down into simpler ones, which can be proved separately. The results can then be soundly recombined to complete the proof of the original complex (hyper)property. Contrary to our previously defined transformers, here continuity and co-continuity do not related with well-definedness. In fact, our transformers are always well-defined just because they rely on the well-definedness of the underlying semantics, i.e., $\mathsf{sp}$.

### Strictness, Co-strictness and $k$-strictness

In the context of classical $\mathsf{wp}$, strictness (also known as the "Law of the Excluded Miracle" [63]) ensures that no initial state can terminate in a state satisfying "false". Quantitative generalisations of strictness [16, Definition 4.13], defined as $\mathsf{wp}\,[\![C]\!]\,(0) = 0$, mean that the expected value of the constantly 0 random variable after executing a program $C$ is 0.

In our setting, we show that our $\mathsf{whp}$ enjoys of a novel property, namely $k$-strictness (see Theorem 5.6.4, item 3), which generalises strictness and co-strictness. In fact, we can represent strictness by taking $k = -\infty$: for predicates, it means it is impossible to terminate in a set of states that satisfies the hyperpostcondition "false". Conversely, for $k = +\infty$, we have a generalisation of the so-called co-strictness: any initial precondition will terminate in a postcondition that satisfies the hyperpostcondition "true".

Our $\mathsf{shp}$ and $\mathsf{slhp}$, on the other hand, enjoy of strictness and co-strictness, respectively. The difference between forward and backward transformers is that the underlying semantics, $\mathsf{sp}$, is deterministic, so $\mathsf{whp}\,[\![C]\!]$ is always able to anticipate the exact postcondition $\mathsf{sp}[\![C]\!]$. In contrast, $\mathsf{shp}$ and $\mathsf{slhp}$ cannot do so, because the inverse of $\mathsf{sp}[\![C]\!]$ is non-deterministic. In the context of probability distributions, strictness of $\mathsf{shp}$ ensures that starting from no distribution we end in no distributiion again, whereas co-strictness of $\mathsf{slhp}$ ensures that for all distributions, they are either reachable by an initial distribution or not reachable at all.

## Linearity

Sub- and superlinearity, extensively studied by Kozen, McIver & Morgan, and Kaminski for probabilistic $\mathsf{w(l)p}$ transformers, also find applications in our hyper setting. Our $\mathsf{whp}$ obeys to linearity.

**Theorem 5.6.6** (Linearity). *For all programs $C$, $\mathsf{whp}\,[\![C]\!]$ is linear, i.e. for all $\mathit{ff}, \mathit{gg} \in \mathbb{AA}$ and* non-negative *constants $r \in \mathbb{R}_{\geq 0}$,*

$$\mathsf{whp}\ [\![C]\!]\,(r \cdot \mathit{ff} + \mathit{gg}) \;=\; r \cdot \mathsf{whp}\ [\![C]\!]\,(\mathit{ff}) + \mathsf{whp}\ [\![C]\!]\,(\mathit{gg}) \ .$$

$\triangleleft$

Our forward hyper transformers $\mathsf{shp}$ and $\mathsf{slhp}$ do not enjoy linearity, but similarly to what we showed in Section 3.5, they exhibit sub- and superlinearity, respectively.

**Theorem 5.6.7** (Linearity). *For all programs $C$, $\mathsf{shp}\,[\![C]\!]$ is sublinear and*

$\mathsf{slp}[\![C]\!]$ *is superlinear, i.e. for all* $\mathbb{f}, \mathbb{g} \in \mathbb{A}\mathbb{A}$ *and constants* $r \in U$*,*

$$\mathsf{shp}\ [\![C]\!]\,(r \cdot \mathbb{f} + \mathbb{g}) \;\preceq\; r \cdot \mathsf{shp}\ [\![C]\!]\,(\mathbb{f}) + \mathsf{shp}\ [\![C]\!]\,(\mathbb{g}) \ , \quad \text{and}$$
$$r \cdot \mathsf{slhp}\ [\![C]\!]\,(\mathbb{f}) + \mathsf{slhp}\ [\![C]\!]\,(\mathbb{g}) \;\preceq\; \mathsf{slhp}\ [\![C]\!]\,(r \cdot \mathbb{f} + \mathbb{g}) \ .$$

$\lhd$

## Multiplicativity

We additionally show that our hyper transformers enjoy of a novel property, namely, multiplicativity.

**Theorem 5.6.8** (Multiplicativity)**.** *For all programs* $C$*,* $\mathsf{whp}\,[\![C]\!]$ *is multiplicative, i.e. for all* $\mathbb{f}, \mathbb{g} \in \mathbb{A}\mathbb{A}$ *and* non-negative *constants* $r \in \mathbb{R}_{\geq 0}$*,*

$$\mathsf{whp}\ [\![C]\!]\,(r \cdot \mathbb{f} \cdot \mathbb{g}) \;=\; r \cdot \mathsf{whp}\ [\![C]\!]\,(\mathbb{f}) \cdot \mathsf{whp}\ [\![C]\!]\,(\mathbb{g}) \ .$$

$\lhd$

Similarly to linearity, multiplicativity also aids compositional reasoning. Let us show an example of its validity.

**Example 5.6.1** (Validity of Multiplicativity for $\mathsf{whp}$)**.** *Consider the program* $C = \{\, x \coloneqq 0 \,\}\ [\, 0.5 \,]\ \{\, x \coloneqq 1 \,\}$ *that assigns 0 to* $x$ *with probability 0.5 and 1 with probability 0.5. Let's verify the multiplicativity property of* $\mathsf{whp}$ *using two hyperquantities:* $\mathbb{f} = \mathbb{g} = \mathbb{E}[x]$ *and considering an initial distribution* $\mu = \mathbf{1}_{x=1}$*. We start by computing* $\mathsf{whp}\ [\![C]\!]\,(\mathbb{f})\,(\mu)$ *for any generic hyperquantity* $\mathbb{f}$*, and then we compute the left and right-hand side of Theorem 5.6.8.*

$$\begin{aligned}
&\mathsf{whp}\ [\![C]\!]\,(\mathbb{f})\,(\mu) \\
&\quad = \; Ƨ\,\nu_1, \nu_2\colon\ \mathbb{f}(\nu_1 + \nu_2) \\
&\qquad\qquad ⅄\ \mathsf{whp}\ [\![\odot\, 0.5 \fatsemi\ x \coloneqq 0]\!]\,([\nu_1])\,(\mu)\ ⅄\ \mathsf{whp}\ [\![\odot\, 0.5 \fatsemi\ x \coloneqq 1]\!]\,([\nu_2])\,(\mu)
\end{aligned}$$

$$= \textit{ff}(0.5 \cdot \mathbf{1}_{x=0} + 0.5 \cdot \mathbf{1}_{x=1})$$

*Now, let's compute the left-hand side of Theorem 5.6.8*

$$\begin{aligned}
&\mathsf{whp}\ [\![C]\!]\ (\mathbb{E}[x] \cdot \mathbb{E}[x])\ (\mu) \\
&= (\mathbb{E}[x] \cdot \mathbb{E}[x])(0.5 \cdot \mathbf{1}_{x=0} + 0.5 \cdot \mathbf{1}_{x=1}) \\
&= \mathbb{E}[x](0.5 \cdot \mathbf{1}_{x=0} + 0.5 \cdot \mathbf{1}_{x=1}) \cdot \mathbb{E}[x](0.5 \cdot \mathbf{1}_{x=0} + 0.5 \cdot \mathbf{1}_{x=1}) \\
&= \frac{1}{4}\ .
\end{aligned}$$

*Finally, we can compute the right-hand side of Theorem 5.6.8 to see that it coincides with the result above:*

$$\begin{aligned}
&\mathsf{whp}\ [\![C]\!]\ (\mathbb{E}[x])\ (\mu) \cdot \mathsf{whp}\ [\![C]\!]\ (\mathbb{E}[x])\ (\mu) \\
&= \mathbb{E}[x](0.5 \cdot \mathbf{1}_{x=0}) \cdot \mathbb{E}[x](0.5 \cdot \mathbf{1}_{x=0}) \\
&= \frac{1}{4}\ .
\end{aligned}$$

$\triangleleft$

In general, it is easy to see that multiplicativity does not hold for wp.

**Example 5.6.2** (Invalidity of Multiplicativity for wp)**.** *Let* $C = \{\, x \coloneqq 0 \,\}\ [\,0.5\,]\ \{\, x \coloneqq 1 \,\}$. *Then:*

$$\begin{aligned}
&\mathsf{wp}\,[\![C]\!]\,(x \cdot x) \\
&= \frac{1}{2} \cdot \mathsf{wp}\,[\![x \coloneqq 0]\!]\,(x \cdot x) + \frac{1}{2} \cdot \mathsf{wp}\,[\![x \coloneqq 1]\!]\,(x \cdot x) \\
&= \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 \\
&= \frac{1}{2}\ .
\end{aligned}$$

*However,*

$$\begin{aligned}
&\mathsf{wp}\,[\![C]\!]\,(x)\cdot\mathsf{wp}\,[\![C]\!]\,(x)\\
&=\ \left[\frac{1}{2}\cdot\mathsf{wp}\,[\![x\coloneqq 0]\!]\,(x)+\frac{1}{2}\cdot\mathsf{wp}\,[\![x\coloneqq 1]\!]\,(x)\right]^2\\
&=\ \left[\frac{1}{2}\cdot 0+\frac{1}{2}\cdot 1\right]^2\\
&=\ \frac{1}{4}\ .
\end{aligned}$$

◁

Similarly, multiplicativity does not hold for $\mathsf{sp}$ and $\mathsf{slp}$.

**Example 5.6.3** (Invalidity of Multiplicativity for $\mathsf{sp},\mathsf{slp}$)**.** *Let* $C = \{\,x\coloneqq x-1\,\}\ [0.5]\ \{\,x\coloneqq x+1\,\}$. *Then:*

$$\begin{aligned}
&\mathsf{sp}\,[\![C]\!]\,(x\cdot x)\\
&=\ \frac{1}{2}\cdot\mathsf{sp}\,[\![x\coloneqq x-1]\!]\,(x\cdot x)+\frac{1}{2}\cdot\mathsf{sp}\,[\![x\coloneqq x+1]\!]\,(x\cdot x)\\
&=\ \frac{1}{2}\cdot(x+1)\cdot(x+1)+\frac{1}{2}\cdot(x-1)\cdot(x-1)\\
&=\ x^2+1\ .
\end{aligned}$$

*However,*

$$\begin{aligned}
&\mathsf{sp}\,[\![C]\!]\,(x)\cdot\mathsf{sp}\,[\![C]\!]\,(x)\\
&=\ \left[\frac{1}{2}\cdot\mathsf{sp}\,[\![x\coloneqq x-1]\!]\,(x)+\frac{1}{2}\cdot\mathsf{sp}\,[\![x\coloneqq x+1]\!]\,(x)\right]^2\\
&=\ \left[\frac{1}{2}\cdot(x+1)+\frac{1}{2}\cdot(x-1)\right]^2\\
&=\ x^2\ ,
\end{aligned}$$

*and analogously for* $\mathsf{slp}$. ◁

On the other hand, we conclude by stating that our strongest hyper transformers obey a restricted form of multiplicativity.

**Theorem 5.6.9** (Multiplicativity)**.** *For all programs $C$,* $\mathsf{shp}\,[\![C]\!]$ *is submultiplicative and* $\mathsf{slhp}\,[\![C]\!]$ *is supermultiplicative, i.e. for all $\mathbb{f}, \mathbb{g} \in \mathbb{AA}$ and* non-negative *constants $r \in \mathbb{R}_{\geq 0}$,*

$$\begin{aligned} \mathsf{shp}\,[\![C]\!]\,(r \cdot \mathbb{f} \cdot \mathbb{g}) &\preceq r \cdot \mathsf{shp}\,[\![C]\!]\,(\mathbb{f}) \cdot \mathsf{shp}\,[\![C]\!]\,(\mathbb{g}) \ , \quad \text{and} \\ r \cdot \mathsf{slhp}\,[\![C]\!]\,(\mathbb{f}) \cdot \mathsf{slhp}\,[\![C]\!]\,(\mathbb{g}) &\preceq \mathsf{slhp}\,[\![C]\!]\,(r \cdot \mathbb{f} \cdot \mathbb{g}) \ . \end{aligned}$$

◁

By combining these properties, we can extend our reasoning to encompass other important statistical measures within whp. Covariance, which measures how two random variables vary together, is a fundamental concept in statistics and probabilistic reasoning about programs. Our framework allows us to derive them compositionally from simpler properties.

**Example 5.6.4** (Computing Covariance Compositionally)**.**

$$\begin{aligned} \mathsf{whp}\,[\![C]\!]\,(\mathit{Cov}[f,g]) &= \mathsf{whp}\,[\![C]\!]\,(\mathbb{E}[fg] - \mathbb{E}[f]\cdot\mathbb{E}[g]) \\ &= \mathsf{whp}\,[\![C]\!]\,(\mathbb{E}[fg]) - \mathsf{whp}\,[\![C]\!]\,(\mathbb{E}[f]\cdot\mathbb{E}[g]) \\ &\qquad\qquad \text{(by Theorem 5.6.6)} \\ &= \mathsf{whp}\,[\![C]\!]\,(\mathbb{E}[fg]) - \mathsf{whp}\,[\![C]\!]\,(\mathbb{E}[f])\cdot\mathsf{whp}\,[\![C]\!]\,(\mathbb{E}[g]) \\ &\qquad\qquad \text{(by Theorem 5.6.8)} \end{aligned}$$

◁

This decomposition allows us to compute the covariance between random variables $f$ and $g$ after program execution by computing simpler expected values and combining them. We will demonstrate the practical utility of this approach in Section 5.8.3, where we compute the variance of a random variable representing the outcome of a probabilistic game.

### 5.6.4 Additive Hyperquantities

In this section, we explore a specific category of hyperquantities from which we can deduce simplified rules akin to established wp calculi.

| $C$ | $\mathsf{whp}\,[\![C]\!]\,(\mathit{ff})$ |
|---|---|
| $x \coloneqq e$ | $\mathit{ff}\,[x/e]$ |
| $x \coloneqq \mathtt{nondet()}$ | $\lambda f\colon \mathit{ff}(\bigoplus_\alpha f\,[x/\alpha])$ |
| $\odot\, w$ | $\mathit{ff} \odot w$ |
| $C_1 ⨾ C_2$ | $\mathsf{whp}\,[\![C_1]\!]\,\big(\mathsf{whp}\,[\![C_2]\!]\,(\mathit{ff})\big)$ |
| $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$ | $\mathsf{whp}\,[\![C_1]\!]\,(\mathit{ff}) \ast \mathsf{whp}\,[\![C_2]\!]\,(\mathit{ff})$ |
| $C^{\langle e, e' \rangle}$ | $\mathsf{lfp}\ X\colon \Phi_{\mathit{ff}}(X)$ |
| $\mathtt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \mathtt{else}\ \{\, C_2 \,\}$ | $\mathsf{whp}\,[\![C_1]\!]\,(\mathit{ff}) \odot [\varphi] \ast \mathsf{whp}\,[\![C_2]\!]\,(\mathit{ff}) \odot [\neg\varphi]$ |
| $\{\, C_1 \,\}\ [\,p\,]\ \{\, C_2 \,\}$ | $\mathsf{whp}\,[\![C_1]\!]\,(\mathit{ff}) \odot p \ast \mathsf{whp}\,[\![C_2]\!]\,(\mathit{ff}) \odot (1-p)$ |
| $\mathtt{while}\,(\,\varphi\,)\,\{\, C \,\}$ | $\mathsf{lfp}\ X\colon \mathit{ff} \odot [\neg\varphi] \ast \mathsf{whp}\,[\![C]\!]\,(X) \odot [\varphi]$ |

**Table 5.4:** Rules for the weakest hyper pre transformer for additive posts $\mathit{ff}$. Here, $\mathsf{lfp}\ f\colon \Phi(f)$ denotes the least fixed point of $\Phi$.

**Definition 5.6.1** (Additive Hyperquantities). *A hyperquantity $\mathit{ff} \in \mathbb{AA}$ is* additive *if for any quantity $g, f \in \mathbb{A}$, there exists a binary operator $\ast\colon \mathbb{R}^{\pm\infty} \times \mathbb{R}^{\pm\infty} \to \mathbb{R}^{\pm\infty}$ such that:*

$$\mathit{ff}(g \oplus f) \quad = \quad \mathit{ff}(g) \ast \mathit{ff}(f)\ .$$

**Theorem 5.6.10** (Weakest Hyper Pre for Additive Hyperquantities). *For additive hyperquantities $\mathit{ff} \in \mathbb{AA}$, the simpler rules in* Table 5.4 *are valid.*

Similarly to other quantitative settings [14; 16; 17], the loop rule can be defined via a least fixed point of the characteristic function.

**Definition 5.6.2** ($\mathsf{whp}$–*characteristic function*). *The* $\mathsf{whp}$–characteristic function *(of $C^{\langle e, e' \rangle}$ w.r.t. $\mathit{ff}$) is:*

$$\Phi_{\mathit{ff}}(X) \;=\; \mathit{ff} \odot [\![e']\!] \ast \mathsf{whp}\,[\![C]\!]\,(X) \odot [\![e]\!]. \qquad \triangleleft$$

When examining the semiring $\langle \mathbb{R}^{\pm\infty}, \max, \min, -\infty, +\infty \rangle$, our calculus closely resembles the quantitative $\mathsf{wp}$ as described in Zhang and

Kaminski [17], albeit in a more expressive context. Further, by adopting $\langle \mathbb{R}^{\pm\infty}, \min, \max, +\infty, -\infty \rangle$, we derive rules analogous to quantitative $\mathsf{wlp}$ from Zhang and Kaminski [17]. Notably, in the latter semiring, the natural order is reversed compared to the semiring $\langle \mathbb{R}^{\pm\infty}, \max, \min, -\infty, +\infty \rangle$. In essence, for $\langle \mathbb{R}^{\pm\infty}, \min, \max, +\infty, -\infty \rangle$, the least fixed point resulting from our iteration rule aligns with the rule of $\mathsf{wlp}$ defined through the greatest fixed point in Zhang and Kaminski [17].

Among additive hyperquantities we have all those in Example 5.2.4 and of Section 5.7.3.

### 5.6.5 Loops Rules for Additive Hyperquantities

Reasoning about loops is undecidable, even for classical properties. Previously, we have shown that our $\mathsf{whp}$ calculus uses least fixed points, which is often impractical. In this section, we show how to derive simpler rules that can aid in $\mathsf{whp}$ reasoning for loops. For additive hyperquantities, we obtain an inductive invariant based rule similar to the existing ones for quantitative transformers [17, Theorem 7.1].

**Theorem 5.6.11** (Quantitative Inductive Reasoning for $\mathsf{whp}$). *For any program $C$ and any additive hyperquantity $\mathit{ff}$, we have:*

$$\Phi_{\mathit{ff}}(\mathit{ii}) \preceq \mathit{ii} \implies \mathsf{whp}\left[\!\left[ C^{\langle e, e' \rangle} \right]\!\right](\mathit{ff}) \preceq \mathit{ii},$$

*where $\Phi_{\mathit{ff}}(X) = \mathit{ff} \odot [\![e']\!] \ast \mathsf{whp}\,[\![C]\!]\,(X) \odot [\![e]\!]$ is the characteristic function of $C^{\langle e, e' \rangle}$ w.r.t. $\mathit{ff}$.* $\lhd$

As a corollary, one can derive simpler rules for guarded loops, for example, the analogue of Theorem 5.4 of Kaminski [16], but in our hyper setting.

**Corollary 5.6.11.1** (Quantitative Inductive Rule for `while`).

$$\frac{\mathit{ff} \odot [\neg\varphi] \ast \mathsf{whp}\,[\![C]\!]\,(\mathit{ii}) \odot [\varphi] \;\preceq\; \mathit{ii} \;\preceq\; \mathit{gg} \qquad \mathit{ff} \textit{ is additive}}{\mathsf{whp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(\mathit{ff}) \;\preceq\; \mathit{gg}} \;\text{while}-\mathsf{whp}$$

$\lhd$

We shall observe that Corollary 5.6.11.1 subsumes *both* while-$\mathsf{wp}$ and while-$\mathsf{wlp}$ of Zhang and Kaminski [17, Theorem 7.1]. This depends on the choice of the semiring: $\langle \mathbb{R}^{\pm\infty}, \min, \max, +\infty, -\infty \rangle$ for $\mathsf{wp}$, and $\langle \mathbb{R}^{\pm\infty}, \max, \min, -\infty, +\infty \rangle$ for $\mathsf{wlp}$.

Let us provide an intuition over while$-\mathsf{whp}$ in our quantitative hyper setting, for example taking into account the semiring $\mathsf{Prob} = \langle [0,1], +, \cdot, 0, 1 \rangle$ and the expected value hyperquantity $\mathbb{f} = \mathbb{E}[f]$. Intuitively, the rule while$-\mathsf{whp}$ requires finding an invariant $\mathbb{i}$ that satisfies three conditions:

1. $\mathbb{i} \preceq \mathbb{g}$, meaning that $\mathbb{g}$ is overapproximating the invariant $\mathbb{i}$;

2. $\mathbb{E}[f] \cdot [\neg\varphi] \preceq \mathbb{i}$, meaning that the expected value of $f$, when evaluated in the filtered probability distribution (i.e., the loop is executed at most 0 times), is bounded by $\mathbb{i}$;

3. $\mathsf{whp}\ [\![C]\!]\,(\mathbb{i}) \cdot [\varphi] \preceq \mathbb{i}$, meaning that for any initial probability distribution $\mu$, the value of $\mathbb{i}$ computed over this initial distribution will be greater than or equal to the value of $\mathbb{i}$ after performing one more iteration and computing it over the resulting distribution.

By induction, conditions (2) and (3), which represent the first premise of while-$\mathsf{whp}$, imply that $\mathbb{i}$ overapproximates the expected value $\mathbb{E}[f]$ computed in the final probability distribution after the loop execution. Indeed, starting from the base case in (2), we assume for the inductive step that $\mathbb{i}$ over-approximates the expected value after $n$ loop iterations. By condition (3), $\mathbb{i}$ is also an upper bound for $\mathsf{whp}\ [\![C]\!]\,(\mathbb{i}) \cdot [\varphi] \preceq \mathbb{i}$, meaning that it over-approximates the probability distribution obtained after $n+1$ iterations.

Condition (1) ensures that the initial expected value $\mathbb{g}$ overapproximates $\mathbb{i}$, and thus $\mathbb{g}$ computed in the initial probability distribution overapproximates the final expected value $\mathbb{E}[f]$.

We showcase an example of induction reasoning that extends [16, Example 5.5] by taking into account probability distributions instead of single states.

**Example 5.6.5** (Upper Bounds on $\mathsf{whp}$)**.** *Consider the probabilistic loop $C_{geo} = x \coloneqq x+1^{\langle 0.5, 0.5\rangle}$ modeling a geometric distribution. We want to prove that $\mathbb{E}[x+1]$ is an upper bound of the expected value $\mathbb{E}[x]$ after executing $C_{geo}$. We have:*

$$\begin{aligned} & \mathbb{E}[x] \cdot [\![0.5]\!] + \mathsf{whp}\ [\![x \coloneqq x+1]\!]\,(\mathbb{E}[x+1]) \cdot [\![0.5]\!] \\ = \;& \mathbb{E}[x \cdot 0.5] + \mathbb{E}[(x+2) \cdot 0.5] \\ = \;& \mathbb{E}[x+1], \end{aligned}$$

*and hence by Corollary 5.6.11.1 we conclude that $\mathsf{whp}\ [\![C_{geo}]\!]\,(\mathbb{E}[x]) \preceq \mathbb{E}[x+1]$, i.e., $\mathbb{E}[x+1]$ (evaluated in the initial probability distribution) is an upper bound on $\mathbb{E}[x]$ (evaluated in the final probability distribution) after executing $C_{geo}$.*

## 5.7 Expressivity

In the preceding sections, we characterized our quantitative hyper transformers. In this section, we aim to illustrate the expressive capabilities of the calculi by demonstrating that they subsume several other logics and calculi.

### 5.7.1 An Overview of Several Hoare-Like Logics

We subsume Hyper Hoare Logic for non-probabilistic programs (since $\mathsf{HHL}$ is non-probabilistic).

**Theorem 5.7.1** (Subsumption of $\mathsf{HHL}$)**.** *For hyperpredicates $\psi$, $\phi$ and non-probabilistic program $C$:*

$$\begin{aligned} & \models_{\mathrm{hh}} \{\, \psi \,\}\ C\ \{\, \phi \,\} \\ & \quad \text{iff} \quad \mathsf{supp}\,([\psi]) \subseteq \mathsf{supp}\,(\mathsf{whp}\ [\![C]\!]\,([\phi])) \\ & \quad \text{iff} \quad \mathsf{supp}\,(\mathsf{shp}\ [\![C]\!]\,([\phi])) \subseteq \mathsf{supp}\,([\psi]) \\ & \quad \text{iff} \quad \mathsf{supp}\,([\neg\psi]) \subseteq \mathsf{supp}\,(\mathsf{slhp}\ [\![C]\!]\,([\neg\phi])) \end{aligned}$$

As a byproduct, all our hyper transformers subsume demonic partial correctness, angelic total correctness, partial incorrectness, and total incorrectness (according to the terminology in [17]). To highlight this, we will utilize the following modality syntax introduced in [74]:

$$\Box P \;=\; \lambda\rho\colon [\rho \subseteq P] \qquad \text{and} \qquad \Diamond P \;=\; \lambda\rho\colon [P \cap \rho \neq \emptyset]$$

These modalities provide a concise notation for expressing universal ($\Box$) and existential ($\Diamond$) properties over execution traces, enabling elegant formulations of various program logics without the verbosity of lambda expressions. When reasoning about hyperproperties, we can further simplify the notation by omitting Iverson brackets and writing $\psi\!\!\!/ \subseteq \mathsf{whp}\,[\![C]\!]\,(\phi\!\!\!/)$ instead of $\mathsf{supp}\,([\psi\!\!\!/]) \subseteq \mathsf{supp}\,(\mathsf{whp}\,[\![C]\!]\,([\phi\!\!\!/]))$. In Table 5.5, we demonstrate how our whp transformer subsumes various existing verification logics, providing a unified framework for reasoning about them. Moreover, as shown in Table 5.6, whp elegantly represents the falsification conditions for these logics as well. While Theorem 5.7.1 guarantees that analogous rules can be derived for shp and slhp, we focus our presentation on whp for brevity.

| **Logic** | **Syntax** | **Semantics** | **Semantics via whp** |
|---|---|---|---|
| Hoare Logic (partial correctness) | $\models_{\text{pc}} \{P\}\, C\, \{Q\}$ | $P \subseteq \mathsf{wlp}[\![C]\!]\,(Q)$ | $\Box P \subseteq \mathsf{whp}\,[\![C]\!]\,(\Box Q)$ |
| Lisbon Logic | $\models_{\text{atc}} \{P\}\, C\, \{Q\}$ | $P \subseteq \mathsf{wp}\,[\![C]\!]\,(Q)$ | $\Diamond P \subseteq \mathsf{whp}\,[\![C]\!]\,(\Diamond Q)$ |
| Partial Incorrectness Logic | $\models_{\text{pi}} [P]\, C\, [Q]$ | $Q \subseteq \mathsf{slp}[\![C]\!]\,(P)$ | $\{\neg P\} \subseteq \mathsf{whp}\,[\![C]\!]\,(\Box(\neg Q))$ |
| Incorrectness Logic | $\models_{\text{ti}} [P]\, C\, [Q]$ | $Q \subseteq \mathsf{sp}\,[\![C]\!]\,(P)$ | $\{P\} \subseteq \mathsf{whp}\,[\![C]\!]\,(\lambda\rho.\ Q \subseteq \rho)$ |

**Table 5.5:** Partial and total (in)correctness using classical predicate transformers and whp.

| **Syntax** | **Semantics** | **Semantics via whp** |
|---|---|---|
| $\not\models_{\text{pc}} \{P\}\, C\, \{Q\}$ | $P \cap \mathsf{wp}\,[\![C]\!]\,(\neg Q) \neq \emptyset$ | $\{P\} \subseteq \mathsf{whp}\,[\![C]\!]\,(\Diamond(\neg Q))$ |
| $\not\models_{\text{atc}} \{P\}\, C\, \{Q\}$ | $P \cap \mathsf{wlp}[\![C]\!]\,(\neg Q) \neq \emptyset$ | $\exists \sigma \in P\colon \{\{\sigma\}\} \subseteq \mathsf{whp}\,[\![C]\!]\,(\Box\neg Q)$ |
| $\not\models_{\text{pi}} [P]\, C\, [Q]$ | $Q \cap \mathsf{sp}\,[\![C]\!]\,(\neg P) \neq \emptyset$ | $\{\neg P\} \subseteq \mathsf{whp}\,[\![C]\!]\,(\Diamond Q)$ |
| $\not\models_{\text{ti}} [P]\, C\, [Q]$ | $Q \cap \mathsf{slp}[\![C]\!]\,(\neg P) \neq \emptyset$ | $\{P\} \subseteq \mathsf{whp}\,[\![C]\!]\,(\lambda\rho.\ Q \cap \neg\rho \neq \emptyset)$ |

**Table 5.6:** Disproving partial and total (in)correctness using classical predicate transformers and whp. Non-liberal transformers can be expressed via liberal transformers and vice versa by duality [17, Section 5.3]

Arguably, Hoare-like logics are designed to be accessible to programmers to prove correctness, whereas reasoning about $\mathsf{whp}$ (and $\mathsf{HHL}$, $\mathsf{OL}$) enables better understanding of relationships between different program logics, leading to definitions of new logics, as we will show in the following.

We conclude by noting that the aforementioned logics are capable of proving nontermination, as shown in Figure 4.3. Consequently, our $\mathsf{whp}$ calculus can also be used to prove nontermination, as demonstrated by the rules for $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ in Figure 5.2.

$$\frac{P \subseteq \varphi \qquad \Diamond P \subseteq \mathsf{whp}\ [\![C]\!]\ (\Diamond P)}{\forall \sigma \in P\colon [\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!](\sigma) \text{ may diverge}}$$

$$\frac{P \subseteq \varphi \qquad \{P\} \subseteq \mathsf{whp}\ [\![C]\!]\ (\Box P)}{\forall \sigma \in P\colon [\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!](\sigma) \text{ must diverge}}$$

**Figure 5.2:** Nontermination rules for the $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ construct using $\mathsf{whp}$.

### 5.7.2 Designing (Falsifying) Hoare-Like Logics via Hyperpredicate Transformers

The observations above indicate that there is no advantage for new program logics to falsify triples from an expressivity point of view, as they can be converted into existing triples via Theorem 4.5.1. However, one may wonder whether it is possible to design triples that are more useful in practice. In this regard, we emphasize that the design of program logics should follow predicate transformer reasoning. We provide an intuition on how $\mathsf{whp}$ aids in reasoning about designing logics (rather than triples). We illustrate this with an example of partial correctness.

### Partial Correctness as Classical Predicate Transformers

Partial correctness amounts to a logic that takes $Q \subseteq \mathcal{P}(\Sigma)$ and proves every $P$ such that $P \subseteq \mathsf{wlp}[\![C]\!]\,(Q)$.

### Partial Correctness as a Hyperproperty

We observe that partial correctness, as a logic, is a hyperproperty. Indeed, $P \subseteq \mathsf{wlp}[\![C]\!]\,(Q)$ iff $P \in \{S \mid S \subseteq \mathsf{wlp}[\![C]\!]\,(Q)\}$, and this is a predicate over sets of states. Also, by Galois connection, this is equivalent to proving $\mathsf{sp}\,[\![C]\!]\,(P) \subseteq Q$ iff $\mathsf{sp}\,[\![C]\!]\,(P) \in \{S \mid S \subseteq Q\}$, explaining why our $\mathsf{whp}$ captures partial correctness (via $P \in \mathsf{whp}\,[\![C]\!]\,(\lambda\rho\colon \rho \subseteq Q)$).

### (Dis)proving Partial Correctness, Practically

One may wonder why partial correctness is much easier than our $\mathsf{whp}$ calculus. At first glance, it seems that, for a given post $Q$, one may want to find $\{S \mid S \subseteq \mathsf{wlp}[\![C]\!]\,(Q)\}$. However, the actual logic aims to find just $\mathsf{wlp}[\![C]\!]\,(Q)$ since $\mathsf{wlp}[\![C]\!]\,(Q)$ fully characterizes the original hyperproperty. Even if $\mathsf{wlp}[\![C]\!]\,(Q)$ itself is not found, any $S \subseteq \mathsf{wlp}[\![C]\!]\,(Q)$ allows soundly proving $\models_{\text{pc}} \{\, P \,\}\; C\; \{\, Q \,\}$ by checking $P \subseteq S$. The same reasoning applies to falsify partial correctness triples. Our key insight is that it is enough to find any $\mathsf{wlp}[\![C]\!]\,(Q) \subseteq S$ and then prove $\not\models_{\text{pc}} \{\, P \,\}\; C\; \{\, Q \,\}$ by checking $P \not\subseteq S$. With this in mind, we argue that the most sensible proof system to falsify partial correctness should aim for $\mathsf{wlp}[\![C]\!]\,(Q) \subseteq P$.

So we obtain the following sound and complete falsifying partial correctness logic, which is the same as partial correctness except for the following different rules:

$$\frac{G \Longleftarrow G' \quad \models \{\, G' \,\}\; C\; \{\, F' \,\} \quad F' \Longleftarrow F}{\models \{\, G \,\}\; C\; \{\, F \,\}}\ \text{Antecedence}^3$$

$$\frac{\forall n\colon\ \models \{\, p(n+1) \,\}\; C\; \{\, p(n) \,\}}{\models \{\, \forall n. p(n) \,\}\; C^\star\; \{\, p(0) \,\}}\ \text{Kleene}$$

We argue that by similar reasoning, it is easy to find falsifying logics for the other triples.

### Do we need falsifying logics?

It is known from [17, p.22] that $\mathsf{wlp}[\![C]\!]\,(Q) \subseteq S$ corresponds to the contrapositive of Lisbon Logic, i.e., amounts to $\neg S \subseteq \mathsf{wp}\,[\![C]\!]\,(\neg Q)$. This means that, to prove $\not\models_{\text{pc}} \{\, P \,\}\; C\; \{\, Q \,\}$, one should prove $\models_{\text{atc}} \{\, \neg S \,\}\; C\; \{\, \neg Q \,\}$ (possibly keeping $\neg S$ large) and then check $P \not\subseteq S$. Similar reasoning applies if we want to apply Theorem 4.5.1, and so we argue that reasoning via contrapositive is a lot harder to do for the average programmer.

## 5.7.3 Expressing Quantitative Weakest Pre

In this section we show that our calculus subsumes several existing calculi. We define $\mathbf{1}_\sigma(\tau) = \mathbb{1}$ if $\tau = \sigma$ and $\mathbf{1}_\sigma(\tau) = \mathbb{0}$ otherwise.

### Nondeterministic Programs

We start by defining hyperquantities subsuming existing angelic weakest pre and demonic weakest liberal pre [17].

**Definition 5.7.1** (Hyper Suprema and Infima). *For a given semiring $\mathcal{A} = \langle U, \oplus, \odot, \mathbb{0}, \mathbb{1} \rangle$ and a quantity $f\colon \Sigma \to U$, we define hyperquantities*

$$\curlyvee[f] \triangleq \lambda\mu.\; \curlyvee_{\sigma \in \mathsf{supp}(\mu)} f(\sigma)\ ,$$

$$\curlywedge[f] \triangleq \lambda\mu.\; \curlywedge_{\sigma \in \mathsf{supp}(\mu)} f(\sigma)\ ,$$

*that take as input quantities $\mu\colon \Sigma \to U$. Intuitively, $\curlyvee[f]$ and $\curlywedge[f]$ map a given $\mu$ to the maximum (minimum) value of $f(\sigma)$ where $\sigma$ is drawn from the support set $\mathsf{supp}\,(\mu)$.* ◁

[3]Which replaces the rule of *consequence*.

**Theorem 5.7.2** (Subsumption of Quantitative $\mathsf{wp}$, $\mathsf{wlp}$ for Nondeterministic Programs [17]). *Let $\mathcal{A} = \langle \mathbb{R}^{\pm\infty}, \max, \min, \mathbb{0}, \mathbb{1} \rangle$. For any quantities $g, f$ and any program $C$ satisfying the syntax of* [17, Section 2]*:*

$$\begin{aligned} \mathsf{whp}\ [\![C]\!] \left( \bigcurlywedge[f] \right) (\mathbf{1}_\sigma) &= \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma) \ , \\ \mathsf{whp}\ [\![C]\!] \left( \bigcurlyvee[f] \right) (\mathbf{1}_\sigma) &= \mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \ . \end{aligned}$$

The result follows from the fact that $\mathsf{whp}\ [\![C]\!]\,(\bigcurlywedge[f])\,(\mathbf{1}_\sigma)$ and $\mathsf{whp}\ [\![C]\!]\,(\bigcurlyvee[f])\,(\mathbf{1}_\sigma)$ compute respectively the maximum and the minimum value of $f$ in the support of $\mathsf{sp}\,[\![C]\!]\,(\mathbf{1}_\sigma)$, which is the set of reachable states starting from $\sigma$. Our calculus is strictly more expressive than [17] as our syntax is richer and allows to reason about weighted programs as well.

## Probabilistic Programs

By employing the expected value hyperquantity, we show how $\mathsf{whp}$ subsumes $\mathsf{wp}$ and $\mathsf{wlp}$ for deterministic and probabilistic programs [16] as well.

**Theorem 5.7.3** (Subsumption of Quantitative $\mathsf{wp}$, $\mathsf{wlp}$ for Probabilistic Programs [16]). *Let $\mathsf{Prob} = \langle [0,1], +, \cdot, 0, 1 \rangle$. For any quantities $g, f$ and any* <u>*non-non*</u>*deterministic (possibly probabilistic) program $C$:*

$$\begin{aligned} &\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[f])\,(\mathbf{1}_\sigma) = \mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \\ &\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[f] + 1 - \mathbb{E}[1])\,(\mathbf{1}_\sigma) = \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma) \ . \end{aligned}$$

The results stem from our calculus, which computes $\mathbb{E}[f]$ on the final distribution $\mathsf{sp}\,[\![C]\!]\,(\mathbf{1}_\sigma)$ using the expected values hyperquantity, which precisely yields $\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)$. Additionally, it is known [16, Theorem 4.25] that for nondeterministic programs $\mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)$ calculates the expected value of $f$ in the final distribution $\mathsf{sp}\,[\![C]\!]\,(\mathbf{1}_\sigma)$ , but adjusted for the probability of nontermination. This latter probability is in our setting the hyperquantity $1 - \mathbb{E}[1]$.

### Probabilistic Termination

Since our calculus subsumes many existing quantitative $\mathsf{wp}$ calculi such as those of McIver and Morgan [14]; Zhang and Kaminski [17], we know that is can also prove probabilistic termination (see Kaminski [16, Section 6] for a comprehensive overview). For example, almost-sure termination amounts to proving that $\mathsf{wp}\,[\![C]\!]\,(1)\,(\sigma) = 1$, which in our setting is just $\mathsf{whp}\;[\![C]\!]\,(\mathbb{E}[1])\,(\mathbf{1}_\sigma) = \mathbf{1}$. Bounds over expected values, such as those in Hark et al. [15], are easily handled as well; for example, $\mathsf{whp}\;[\![C]\!]\,(\mathbb{E}[f])\,(\mu) < k$ checks whether the expected value of $f$ after execution of the program is less than $k$. While at first one may argue that this expressiveness comes at the cost of more complex rules, we will show in Section 5.6.3 that when using *additive hyperquantities* (Section 5.6.4), reasoning via $\mathsf{whp}$ is indeed very similar to reasoning via $\mathsf{wp}$.

### Nondeterminism, Regular Languages, and Schedulers

While the results above highlight that many existing $\mathsf{wp}$ are mere specializations of $\mathsf{whp}$ for single initial pre-states, we claim that there are some limitations as well, particularly in how nondeterminism is resolved. The main reason is that all of our transformers, being related to the strongest post $\mathsf{sp}$, cannot detect whether a program $C$ starting from $\sigma$ diverges for at least one possible execution. Therefore we cannot express demonic $\mathsf{wp}$ and angelic $\mathsf{wlp}$. The closest attempt is to define the following hyperquantities.

**Definition 5.7.2** (Demonic Weakest Pre and Angelic Weakest Liberal Pre)**.** *Let the ambient semiring be $\mathcal{A} = \langle \mathbb{R}^{\pm\infty}, \max, \min, -\infty, +\infty\rangle$. Given a quantity $f\colon \Sigma \to \mathbb{R}^{\pm\infty}$, we define hyperquantities*

$$
\begin{aligned}
\bigcurlywedge[f]_{\Downarrow} &\triangleq \lambda\mu\colon \bigcurlywedge_{\sigma\in\mathsf{supp}(\mu)} f(\sigma) \;\curlywedge \bigcurlyvee_{\sigma\in\mathsf{supp}(\mu)} +\infty \\
\bigcurlyvee[f]_{\Uparrow} &\triangleq \lambda\mu\colon \bigcurlyvee_{\sigma\in\mathsf{supp}(\mu)} f(\sigma) \;\curlyvee \bigcurlywedge_{\sigma\in\mathsf{supp}(\mu)} -\infty\,.
\end{aligned}
$$

*One can define two novel transformers:*

$$\mathsf{wp}_{\inf}[\![C]\!]\,(f)\,(\sigma) \triangleq \mathsf{whp}\ [\![C]\!]\left(\curlywedge[f]_{\Downarrow}\right)(\mathbf{1}_{\sigma}) \quad \text{and}$$
$$\mathsf{wlp}_{\sup}[\![C]\!]\,(f)\,(\sigma) \triangleq \mathsf{whp}\ [\![C]\!]\left(\curlyvee[f]_{\Uparrow}\right)(\mathbf{1}_{\sigma}) \qquad \lhd$$

Intuitively, $\mathsf{wp}_{\inf}[\![C]\!]\,(f)\,(\sigma)$ operates akin to a demonic weakest pre calculus by determining the minimum value of $f$ after the execution of program $C$ starting from $\sigma$. However, unlike the demonic weakest pre calculus in [16], we do not necessarily assign the value bottom $\mathbb{0}$ if the program has a single diverging trace; instead, we do so only when all traces are diverging. Similarly, for $\mathsf{wlp}_{\sup}$, our calculus outputs $\mathbb{1}$ if all traces are diverging. In other words, both our $\mathsf{wp}_{\inf}$ and angelic $\mathsf{wlp}_{\sup}$ attempt to avoid termination whenever possible, mirroring the behavior of the angelic $\mathsf{wp}$ and demonic $\mathsf{wlp}$ as discussed in [17, Section 6.2]. To the best of our knowledge these transformers are novel and have not been considered in existing works such as [16; 93; 140].

Let us demonstrate how our demonic weakest pre ($\mathsf{wp}_{\inf}$) and angelic weakest liberal pre ($\mathsf{wlp}_{\sup}$) transformers differ from those in [16] through an example.

**Example 5.7.1** (Comparing Nondeterminism)**.** *Let* $\mathsf{dwp}$ *and* $\mathsf{awlp}$ *be the demonic weakest pre and angelic weakest liberal pre in [16], and let* $C = \{\,\texttt{diverge}\,\} \,\square\, \{\,\texttt{skip}\,\}$. *Then:*

- $\mathsf{dwp}[\![C]\!]\,([\mathsf{true}]) = [\mathsf{false}] \quad \neq \quad [\mathsf{true}] = \mathsf{wp}_{\inf}[\![C]\!]\,([\mathsf{true}])$
- $\mathsf{awlp}[\![C]\!]\,([\mathsf{false}]) = [\mathsf{true}] \quad \neq \quad [\mathsf{false}] = \mathsf{wlp}_{\sup}[\![C]\!]\,([\mathsf{false}])$ $\lhd$

Conventional treatment of nondeterministic programs in established weakest pre calculi inherently involve schedulers [16, Definition 3.7] designed to resolve nondeterminism, seeking the maximum or minimum expected value across all possible schedulers. In contrast, our approach aligns with the Incorrectness Logic literature, using Kleene Algebra and strongest-post-style calculi as program semantics [6; 19; 73; 17]: for nondeterministic programs, we treat all choices as if they were executed. To further highlight the differences between our approach and scheduler-based semantics, we observe that extending $\mathsf{dwp}$ in the sense of Kaminski [16] would invalidate common syntactic sugar for control structures. Most notably, the standard equivalence between $\mathtt{if}\ (\varphi)\ \{\, C_1 \,\}\ \mathtt{else}\ \{\, C_2 \,\}$ and $\{\, \mathtt{assume}\ \varphi \fatsemi C_1 \,\} \mathbin{\square} \{\, \mathtt{assume}\ \neg\varphi \fatsemi C_2 \,\}$ breaks down under demonic weakest pre and angelic weakest liberal pre semantics.

**Example 5.7.2.** *Let* $\mathsf{dwp}$ *and* $\mathsf{awlp}$ *be the demonic weakest pre and angelic weakest liberal pre of Kaminski [16]. We extend both for the assume statement, obtaining:*

$$\mathsf{dwp}[\![\mathtt{assume}\ \varphi]\!]\,(f) = \varphi \curlywedge f \qquad \text{and} \qquad \mathsf{awlp}[\![\mathtt{assume}\ \varphi]\!]\,(f) = [\neg\varphi] \curlyvee f$$

*We have* $\mathsf{dwp}[\![\mathtt{if}\ (\mathsf{true})\ \{\, \mathtt{skip} \,\}\ \mathtt{else}\ \{\, \mathtt{skip} \,\}]\!]\,([\mathsf{true}]) = [\mathsf{true}]$*, whereas for the seemingly equivalent* $\{\, \mathtt{assume}\ \mathsf{true} \fatsemi \mathtt{skip} \,\} \mathbin{\square} \{\, \mathtt{assume}\ \mathsf{false} \fatsemi \mathtt{skip} \,\}$ *we have:*

$$\begin{aligned}
&\mathsf{dwp}[\![\{\, \mathtt{assume}\ \mathsf{true} \fatsemi \mathtt{skip} \,\} \mathbin{\square} \{\, \mathtt{assume}\ \mathsf{false} \fatsemi \mathtt{skip} \,\}]\!]\,([\mathsf{true}]) \\
&\quad= \mathsf{dwp}[\![\mathtt{assume}\ \mathsf{true} \fatsemi \mathtt{skip}]\!]\,([\mathsf{true}]) \curlywedge \mathsf{dwp}[\![\mathtt{assume}\ \mathsf{false} \fatsemi \mathtt{skip}]\!]\,([\mathsf{true}]) \\
&\quad= \mathsf{dwp}[\![\mathtt{assume}\ \mathsf{true}]\!]\,(\mathsf{dwp}[\![\mathtt{skip}]\!]\,([\mathsf{true}])) \curlywedge \mathsf{dwp}[\![\mathtt{assume}\ \mathsf{false}]\!]\,(\mathsf{dwp}[\![\mathtt{skip}]\!]\,([\mathsf{true}])) \\
&\quad= \mathsf{dwp}[\![\mathtt{assume}\ \mathsf{true}]\!]\,([\mathsf{true}]) \curlywedge \mathsf{dwp}[\![\mathtt{assume}\ \mathsf{false}]\!]\,([\mathsf{true}]) \\
&\quad= [\mathsf{true}] \curlywedge [\mathsf{false}] \;=\; [\mathsf{false}]
\end{aligned}$$

*Similarly,* $\mathsf{awlp}[\![\mathtt{if}\ (\mathsf{true})\ \{\, \mathtt{skip} \,\}\ \mathtt{else}\ \{\, \mathtt{skip} \,\}]\!]\,([\mathsf{false}]) = [\mathsf{false}]$ *but:*

$$\begin{aligned}
&\mathsf{awlp}[\![\{\, \mathtt{assume}\ \mathsf{true} \fatsemi \mathtt{skip} \,\} \mathbin{\square} \{\, \mathtt{assume}\ \mathsf{false} \fatsemi \mathtt{skip} \,\}]\!]\,([\mathsf{false}]) \\
&\quad= \mathsf{awlp}[\![\mathtt{assume}\ \mathsf{true} \fatsemi \mathtt{skip}]\!]\,([\mathsf{false}]) \curlyvee \mathsf{awlp}[\![\mathtt{assume}\ \mathsf{false} \fatsemi \mathtt{skip}]\!]\,([\mathsf{false}]) \\
&\quad= \mathsf{awlp}[\![\mathtt{assume}\ \mathsf{true}]\!]\,(\mathsf{awlp}[\![\mathtt{skip}]\!]\,([\mathsf{false}])) \\
&\qquad \curlyvee \mathsf{awlp}[\![\mathtt{assume}\ \mathsf{false}]\!]\,(\mathsf{awlp}[\![\mathtt{skip}]\!]\,([\mathsf{false}])) \\
&\quad= \mathsf{awlp}[\![\mathtt{assume}\ \mathsf{true}]\!]\,([\mathsf{false}]) \curlyvee \mathsf{awlp}[\![\mathtt{assume}\ \mathsf{false}]\!]\,([\mathsf{false}]) \\
&\quad= [\mathsf{false}] \curlyvee [\mathsf{true}] = [\mathsf{true}]
\end{aligned}$$

◁

Whilst the fact that demonic total correctness is inexpressible in KAT [173] because it lacks a way of reasoning about nontermination [174], here we argue that also angelic partial correctness in the sense of [16] is inexpressible. This highlights the fact that regular languages, such as KAT variants, are not equivalent to guarded imperative languages in general.

## 5.8 Case Studies

$\mathit{gg}'$
$\mathit{gg}$
$C$
$\mathit{ff}$
$\mathit{ff}$
$C$
$\mathit{gg}$
$\mathit{gg}'$

**Figure 5.3:** Annotation styles used to express that $\mathit{gg} = \mathsf{whp}\,[\![C]\!]\,(\mathit{ff})$ and $\mathit{gg}' = \mathit{gg}$ (left), and $\mathit{gg} = \mathsf{shp}\,[\![C]\!]\,(\mathit{ff})$ and $\mathit{gg} = \mathit{gg}'$ (right).

In this section, we demonstrate the efficacy of our quantitative hyper transformers. We use the annotation style on the left to express that $\mathit{gg} = \mathsf{shp}\,[\![C]\!]\,(\mathit{ff})$ and that $\mathit{gg} = \mathit{gg}'$. We use the one on the right to express that $\mathit{gg} = \mathsf{whp}\,[\![C]\!]\,(\mathit{ff})$ and $\mathit{gg}' = \mathit{gg}$.

### 5.8.1 Proving Hyperproperties

In this section we show how to prove noninterference [54] and generalized noninterference [175; 176] within $\mathsf{whp}$ and $\mathsf{shp}$.

#### Proving Noninterference

Noninterference, also known as observational nondeterminism [177, Equation 6], amounts to proving that any two executions of the program with the same low-sensitivity inputs must have the same low outputs. This can be formalised by defining $\mathrm{low}(l) \triangleq \lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l)$ and proving $\mathrm{low}(l) \subseteq \mathsf{whp}\,[\![C]\!]\,(\mathrm{low}(l))$. For example consider the program and the $\mathsf{whp}$ annotations in Figure 5.4. The program satisfies NI since $\mathrm{low}(l) \subseteq \lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(h) > 0 \wedge \sigma_2(h) > 0 \implies \sigma_1(l) = \sigma_2(l)$.

$^{=}$/// $\boldsymbol{\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(h) > 0 \land \sigma_2(h) > 0}$

$\quad \boldsymbol{\implies \sigma_1(l) = \sigma_2(l)}$

/// $\boldsymbol{\lambda S\colon \forall \sigma_1, \sigma_2 \in (h > 0)(S)\colon \sigma_1(l) = \sigma_2(l)}$

`assume` $h > 0$

$^{=}$/// $\boldsymbol{\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l)}$

/// $\boldsymbol{\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l+1) = \sigma_2(l+1)}$

$l \coloneqq l + 1$

/// $\boldsymbol{\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l)}$

**Figure 5.4:** Proving noninterference via whp

By Theorem 5.6.1, we can prove NI via shp as well. In fact, starting from the hyperprecondition low($l$), we obtain that shp $[\![C]\!]$ (low($l$)) $= \lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l+1) = \sigma_2(l+1)) \land (h > 0)(S') = S$ from the annotations in Figure 5.5, and hence shp $[\![C]\!]$ (low($l$)) $\subseteq$ low($l$).

/// $\boldsymbol{\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l)}$

`assume` $h > 0$

/// $\boldsymbol{\lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \land (h > 0)(S') = S}$

$l \coloneqq l + 1$

/// $\boldsymbol{\lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l-1) = \sigma_2(l-1)) \land (h > 0)(S') = S}$

**Figure 5.5:** Proving noninterference via shp

### Proving Generalized Noninterference

Generalized noninterference is a weaker property of NI: it permits two executions of the program with identical low-sensitivity inputs to yield different low outputs, provided that the discrepancy does not arise from their secret input. This concept can be formally expressed by defining glow($l$) $\triangleq \lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \exists \sigma \in$

$S\colon \sigma(h) = \sigma_1(h) \land \sigma(l) = \sigma_2(l)$, where $\sigma$ denotes a potential third execution sharing the same secret input as $\sigma_1$ but producing the same low output as $\sigma_2$. GNI can be proved by checking $\text{low}(l) \subseteq \mathsf{whp}\ [\![C]\!]\ (\text{glow}(l))$. For example consider the program and the whp annotations in Figure 5.6. The program satisfies GNI since $\text{low}(l) \subseteq \lambda S\colon \forall \sigma_1, \sigma_2 \in \{\sigma\,[y/\alpha] \mid \sigma \in S\}\colon \exists \sigma \in x\{\sigma\,[y/\alpha] \mid \sigma \in S\}\colon \sigma(h) = \sigma_1(h) \land \sigma(y+h) = \sigma_2(y+h)$.

```
///  $\lambda S\colon \forall \sigma_1, \sigma_2 \in \{\sigma\,[y/\alpha] \mid \sigma \in S\}\colon$
       $\exists \sigma \in \{\sigma\,[y/\alpha] \mid \sigma \in S\}\colon \sigma(h) = \sigma_1(h) \land \sigma(y+h) = \sigma_2(y+h)$
///  $\lambda S\colon \forall \sigma_1, \sigma_2 \in \exists_\alpha\ S\,[y/\alpha]\ \colon$
       $\exists \sigma \in \exists_\alpha\ S\,[y/\alpha]\ \colon \sigma(h) = \sigma_1(h) \land \sigma(y+h) = \sigma_2(y+h)$
$y$ := nondet()
///  $\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \exists \sigma \in S\colon \sigma(h) = \sigma_1(h) \land \sigma(y+h) = \sigma_2(y+h)$
$l := y + h$
///  $\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \exists \sigma \in S\colon \sigma(h) = \sigma_1(h) \land \sigma(l) = \sigma_2(l)$
```

**Figure 5.6:** Proving generalized noninterference (GNI) via whp

By Theorem 5.6.1, we can prove GNI via shp as well. In fact, starting from the hyperprecondition $\text{low}(l)$, we obtain that $\mathsf{shp}\ [\![C]\!]\ (\text{low}(l)) = \lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \land \exists_{\alpha,\beta}(S'\,[y/\alpha]\,[l/\beta]) \land l = y + h = S$ from the annotations in Figure 5.7, and hence $\mathsf{shp}\ [\![C]\!]\ (\text{low}(l)) \subseteq \text{glow}(l)$.

```
///  $\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l)$
$y$ := nondet()
///  $\lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \land \exists_\alpha(S'\,[y/\alpha]) = S$
$l := y + h$
///  $\lambda S\colon \exists S''\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \land \exists_\alpha(S'\,[y/\alpha]) = S''$
       $\land\ \exists_\alpha(S''\,[l/\alpha] \land l = y + h) = S$
```

$^{=}/\!\!/\!\!/\ \ \lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l))$

$\quad \land\ (\exists_{\alpha,\beta}(S' [y/\alpha]\, [l/\beta]) \land l = y + h) = S$

**Figure 5.7:** Proving generalized noninterference (GNI) via shp

### 5.8.2 Disproving Hyperproperties

As pointed in Section 5.2, evaluating whether a program satisfies a specific hyperproperty necessitates proving two HHL triples. For instance, when tackling noninterference, one must attempt to establish *both* $\models_{\text{hh}} \{\, \text{low}(l) \,\}\ C_{\text{ni}}\ \{\, \text{low}(l) \,\}$ and $\models_{\text{hh}}\ \{\, Q \,\}\ C_{\text{ni}}\ \{\, \neg\text{low}(l) \,\}$ (for some $Q \Rightarrow \text{low}(l)$). In this section, we illustrate the advantage of our calculus by disproving NI and GNI.

#### Disproving Noninterference

Disproving NI amounts to proving $\text{low}(l) \not\subseteq \mathsf{whp}\ [\![C]\!]\ (\text{low}(l))$, which is true for the program in Figure 5.8. For example, take $S = \{\sigma_1, \sigma_2\}$ such that $\sigma_1(l) = \sigma_2(l) = 0$ and $\sigma_1(h) = 1 \neq \sigma_2(h) = 2$. Clearly $S \in \text{low}(l)$ but $S \notin \mathsf{whp}\ [\![C]\!]\ (\text{low}(l))$.

$^{=}/\!\!/\!\!/\ \ \lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(h) > 0 \land \sigma_2(h) > 0 \implies \sigma_1(l + h) = \sigma_2(l + h)$

$/\!\!/\!\!/\ \ \lambda S\colon \forall \sigma_1, \sigma_2 \in (h > 0)(S)\colon \sigma_1(l + h) = \sigma_2(l + h)$

`assume` $h > 0$

$/\!\!/\!\!/\ \ \lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l + h) = \sigma_2(l + h)$

$l \coloneqq l + h$

$/\!\!/\!\!/\ \ \lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l)$

**Figure 5.8:** Disproving noninterference (NI) via whp

Via shp, one can reach similar conclusions. In fact, consider $\mathsf{shp}\ [\![C]\!]\ (\text{low}(l))$ in Figure 5.9. The program does not satisfy NI since $\mathsf{shp}\ [\![C]\!]\ (\text{low}(l)) \not\subseteq \text{low}(l)$. For example, take $S = \{\sigma_1, \sigma_2\}$ such that $\sigma_1(l) = 1, \sigma_2(l) = 2$ and $\sigma_1(h) = 1, \sigma_2(h) = 2$. Clearly $S \in \mathsf{shp}\ [\![C]\!]\ (\text{low}(l))$ but $S \notin \text{low}(l)$.

```
/// $\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l)$
assume $h > 0$
/// $\lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \land (h > 0)(S') = S$
$l \coloneqq l + h$
/// $\lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l - h) = \sigma_2(l - h)) \land (h > 0)(S') = S$
```

**Figure 5.9:** Disproving noninterference (NI) via shp

### Disproving Generalized Noninterference

Disproving GNI amounts to prove $\text{low}(l) \not\subseteq \mathsf{whp}\ [\![C]\!]\ (\text{glow}(l))$, which is true for the program in Figure 5.10. For example, take $S = \{\sigma_1, \sigma_2\}$ such that $\sigma_1(l) = \sigma_2(l) = 0$ and $\sigma_1(h) = 0 \neq \sigma_2(h) = 100$. Clearly $S \in \text{low}(l)$ but $S \notin \mathsf{whp}\ [\![C]\!]\ (\text{glow}(l))$.

```
/// $\lambda S\colon \forall \sigma_1, \sigma_2 \in A = \{\sigma[y/\alpha] \mid \sigma \in S, \alpha \in [0, 10]\}\colon$
        $\exists \sigma \in A\colon \sigma(h) = \sigma_1(h) \land \sigma(y + h) = \sigma_2(y + h)$
$y \coloneqq$ nondet()
/// $\lambda S\colon \forall \sigma_1, \sigma_2 \in A = \{\sigma \mid \sigma \in S \land \sigma(y) \in [0, 10]\}\colon$
        $\exists \sigma \in A\colon \sigma(h) = \sigma_1(h) \land \sigma(y + h) = \sigma_2(y + h)$
assume $0 \leq y \leq 10$
/// $\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \exists \sigma \in S\colon \sigma(h) = \sigma_1(h) \land \sigma(y + h) = \sigma_2(y + h)$
$l \coloneqq y + h$
/// $\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \exists \sigma \in S\colon \sigma(h) = \sigma_1(h) \land \sigma(l) = \sigma_2(l)$
```

**Figure 5.10:** Disproving generalized noninterference (GNI) via whp

Via shp, one can reach similar conclusions. In fact, consider $\mathsf{shp}\ [\![C]\!]\ (\text{low}(l))$ in Figure 5.11. The program does not satisfy GNI since $\mathsf{shp}\ [\![C]\!]\ (\text{low}(l)) \not\subseteq \text{glow}(l)$. For example, take $S = \{\sigma_i, \sigma'_i \mid 0 \leq i \leq 10\}$ such that $\sigma_i(l) =$

$i, \sigma_i'(l) = i + 100$ and $\sigma_i(h) = 0, \sigma_i'(h) = 100$. Clearly $S \in \mathsf{shp}\ [\![C]\!]\ (\text{glow}(l))$ but $S \notin \text{low}(l)$.

$/\!\!/\!\!/\ \ \boldsymbol{\lambda S\colon \forall \sigma_1, \sigma_2 \in S\colon \sigma_1(l) = \sigma_2(l)}$

$y \coloneqq$ `nondet()`

$/\!\!/\!\!/\ \ \boldsymbol{\lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \wedge \exists_\alpha (S'\,[y/\alpha]) = S}$

`assume` $0 \leq y \leq 10$

$/\!\!/\!\!/\ \ \boldsymbol{\lambda S\colon \exists S''\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \wedge \exists_\alpha (S'\,[y/\alpha]) = S''}$

$\quad \boldsymbol{\wedge\ (S'' \wedge 0 \leq y \leq 10) = S}$

$=\!\!/\!\!/\!\!/\ \ \boldsymbol{\lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \wedge \exists_{0 \leq \alpha \leq 10} (S'\,[y/\alpha]) = S}$

$l \coloneqq y + h$

$/\!\!/\!\!/\ \ \boldsymbol{\lambda S\colon \exists S''\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l)) \wedge \exists_{0 \leq \alpha \leq 10} (S'\,[y/\alpha]) = S''}$

$\quad \boldsymbol{\wedge\ \exists_\alpha (S''\,[l/\alpha] \wedge l = y + h) = S}$

$=\!\!/\!\!/\!\!/\ \ \boldsymbol{\lambda S\colon \exists S'\colon (\forall \sigma_1, \sigma_2 \in S'\colon \sigma_1(l) = \sigma_2(l))}$

$\quad \boldsymbol{\wedge\ (\exists_{0 \leq \alpha \leq 10, \beta} (S'\,[y/\alpha]\,[l/\beta]) \wedge l = y + h) = S}$

**Figure 5.11:** Disproving generalized noninterference (GNI) via $\mathsf{shp}$

## 5.8.3 Quantitative reasoning

In this section, we demonstrate how $\mathsf{whp}$ enables quantitative reasoning.

### Quantitative Information Flow

Consider the program $C_{\text{qif}}$ in Figure 5.12. Similarly to [17, Section 8.1], we want to infer what is the maximum initial value that the secret variable $h$ can have, by observing a final value $l'$ for the low-sensitive variable $l$. By using $\mathsf{whp}$, it is sufficient to consider the hyperpostquantity $\mathit{ff}_{l'} = \lambda f\colon \curlyvee_\tau\ ([l = l'] \odot f)(\tau)$. Indeed, $\mathsf{whp}\ [\![C_{\text{qif}}]\!]\ (\mathit{ff}_{l'})\ (h)$ tells, what is the maximum value of $\mathsf{sp}\ [\![C_{\text{qif}}]\!]\ (h)\ (\tau)$ among those final states $\tau$ where the value $l'$ has been observed. Since we know from [17] that $\mathsf{sp}\ [\![C_{\text{qif}}]\!]\ (f)\ (\tau)$ produces the maximum initial value of $h$, we have that $\mathsf{whp}\ [\![C_{\text{qif}}]\!]\ (\mathit{ff}_{l'})\ (h)$ correctly yields the maximum initial value of $h$.

For example, $\mathsf{whp}\ \llbracket C_{\text{qif}} \rrbracket\,(\mathit{ff}_{80})\,(h) = 7$, meaning that if we observe 80 as the value of $l$, we know that initially $h$ would have been at most 7.

```
/// $\boldsymbol{\lambda f\colon\ \curlyvee_{\sigma}\ ([99 = l'] \odot [h > 7] \curlyvee [80 = l'] \odot [h \leq 7])(\sigma) \odot f(\sigma)}$
if ( $h > 7$ ) {   /// $\boldsymbol{\lambda f\colon\ \curlyvee_{\tau}\ ([99 = l'] \odot f)(\tau)}$
    $l \coloneqq 99$
} else {   /// $\boldsymbol{\lambda f\colon\ \curlyvee_{\tau}\ ([80 = l'] \odot f)(\tau)}$
    $l \coloneqq 80$
}
/// $\boldsymbol{\lambda f\colon\ \bigcurlyvee_{\tau}\ ([l = l'] \odot f)(\tau)}$
```

**Figure 5.12:** Computing quantitative information flow

We conclude by remarking that the main advantage of whp over sp in this setting is that we can reuse $\mathsf{whp}\ \llbracket C_{\text{qif}} \rrbracket\,(\mathit{ff}_{l'})$ to infer any computation of $\mathsf{sp}\ \llbracket C_{\text{qif}} \rrbracket\,(f)$ for any quantity $f$ we are interested in. With sp, we would need to repeat the computation for each quantity we are interested in reasoning about.

## Variance

We show how to compute the variance of a random variable using whp. Let's consider the following gaming scenario: a player flips a fair coin continuously until a head appears. To assess the variance in the number of flips required to conclude the game, we model this scenario with the program in Figure 5.13.

```
/// $\boldsymbol{\bigoplus_{n\in\mathbb{N}} \mathbb{E}[(1+n)^2] \odot 0.5^{n+1} - (\bigoplus_{n\in\mathbb{N}} \mathbb{E}[1+n] \odot 0.5^{n+1})^2}$
$x \coloneqq 1$
/// $\boldsymbol{\bigoplus_{n\in\mathbb{N}} \mathbb{E}[(x+n)^2] \odot 0.5^{n+1} - (\bigoplus_{n\in\mathbb{N}} \mathbb{E}[x+n] \odot 0.5^{n+1})^2}$
$(x \coloneqq x+1)^{\langle \frac{1}{2}, \frac{1}{2} \rangle}$
/// $\boldsymbol{\mathbb{E}[x^2] - \mathbb{E}[x]^2}$
```

$/\!/\!/\ \mathsf{Cov}[\boldsymbol{x}, \boldsymbol{x}]$

**Figure 5.13:** Computing the variance of a random variable

We leverage Example 5.6.4 to compute $\mathsf{whp}\left[\!\left[x \coloneqq x + 1^{\langle \frac{1}{2}, \frac{1}{2} \rangle}\right]\!\right](\mathsf{Cov}[x,x])$ compositionally. We first compute $\mathsf{whp}\left[\!\left[x \coloneqq x + 1^{\langle \frac{1}{2}, \frac{1}{2} \rangle}\right]\!\right](\mathbb{E}[x^2])$ via subsequent Kleene's iterates, obtaining:

$$\begin{aligned}
W^0_{0.5}(\mathbb{E}[x^2] \odot 0.5) &= \mathbb{E}[x^2] \odot 0.5 \\
W^1_{0.5}(\mathbb{E}[x^2] \odot 0.5) &= \mathsf{whp}\ [\![x \coloneqq x + 1]\!]\ \big(\mathbb{E}[x^2] \odot 0.5\big) \odot 0.5 \\
&= \mathbb{E}[(x+1)^2] \odot 0.5^2 \\
W^2_{0.5}(\mathbb{E}[x^2] \odot 0.5) &= \mathbb{E}[(x+2)^2] \odot 0.5^3 \\
&\vdots \\
W^n_{0.5}(\mathbb{E}[x^2] \odot 0.5) &= \mathbb{E}[(x+n)^2] \odot 0.5^{n+1}
\end{aligned}$$

This leads to:

$$\begin{aligned}
&\mathsf{whp}\left[\!\left[x \coloneqq x + 1^{\langle \frac{1}{2}, \frac{1}{2} \rangle}\right]\!\right]\big(\mathbb{E}[x^2]\big) \\
&= \bigoplus_{n \in \mathbb{N}} W^n_{0.5}(\mathbb{E}[x^2] \odot 0.5) \\
&= \bigoplus_{n \in \mathbb{N}} \mathbb{E}[(x+n)^2] \odot 0.5^{n+1}
\end{aligned}$$

Now we compute $\mathsf{whp}\left[\!\left[x \coloneqq x + 1^{\langle \frac{1}{2}, \frac{1}{2} \rangle}\right]\!\right](\mathbb{E}[x])$, again via subsequent Kleene's iterates obtaining:

$$\begin{aligned}
W^0_{0.5}(\mathbb{E}[x] \odot 0.5) &= \mathbb{E}[x] \odot 0.5 \\
W^1_{0.5}(\mathbb{E}[x] \odot 0.5) &= \mathsf{whp}\ [\![x \coloneqq x + 1]\!]\ (\mathbb{E}[x] \odot 0.5) \odot 0.5 \ = \ \mathbb{E}[x+1] \odot 0.5^2 \\
W^2_{0.5}(\mathbb{E}[x] \odot 0.5) &= \mathbb{E}[x+2] \odot 0.5^3 \\
&\vdots
\end{aligned}$$

$$W^n_{0.5}(\mathbb{E}[x] \odot 0.5) \;=\; \mathbb{E}[x+n] \odot 0.5^{n+1}$$

This leads to:

$$\begin{aligned}
&\mathsf{whp}\ \left[\!\left[x \coloneqq x + 1^{\langle\frac{1}{2},\frac{1}{2}\rangle}\right]\!\right] \left(\mathbb{E}[x]^2\right)\\
&= \big(\bigoplus_{n\in\mathbb{N}} W^n_{0.5}(\mathbb{E}[x] \odot 0.5)\big)^2\\
&= \big(\bigoplus_{n\in\mathbb{N}} \mathbb{E}[x+n] \odot 0.5^{n+1}\big)^2
\end{aligned}$$

$$\begin{aligned}
&\mathsf{whp}\ \left[\!\left[x \coloneqq x + 1^{\langle\frac{1}{2},\frac{1}{2}\rangle}\right]\!\right] \left(\mathbb{E}[x^2] - \mathbb{E}[x]^2\right)\\
&= \bigoplus_{n\in\mathbb{N}} W^n_{0.5}(\mathbb{E}[x^2] \odot 0.5) - \big(\bigoplus_{n\in\mathbb{N}} W^n_{0.5}(\mathbb{E}[x] \odot 0.5)\big)^2\\
&= \bigoplus_{n\in\mathbb{N}} \mathbb{E}[(x+n)^2] \odot 0.5^{n+1} - \big(\bigoplus_{n\in\mathbb{N}} \mathbb{E}[x+n] \odot 0.5^{n+1}\big)^2
\end{aligned}$$

Finally, we take as input any probability distribution $\mu$ and compute the variance via:

$$\begin{aligned}
&\mathsf{whp}\ [\![C]\!]\,(\mathsf{Cov}[x,x])\,(\mu)\\
&= \big(\bigoplus_{n\in\mathbb{N}} \mathbb{E}[(1+n)^2] \odot 0.5^{n+1} - \big(\bigoplus_{n\in\mathbb{N}} \mathbb{E}[1+n] \odot 0.5^{n+1}\big)^2\big)(\mu)\\
&= \sum(1+n)^2 \cdot 0.5^{n+1} - (\sum(1+n)\cdot 0.5^{n+1})^2 \;=\; 6-4=2\ .
\end{aligned}$$

We contend that employing whp offers the advantage of mechanization and compositional computation without necessitating specialized knowledge of probability theory.

### 5.8.4 Conditional expected values

You decide to play a coin-toss game where winning yields 1, and losing results in a loss of 5. You plan ahead by adding specially crafted fake coins to your pocket that guarantee a win when tossed. In addition, you ensure you have some genuine fair coins to display to your opponent. How many coins must be in your pocket (at least) to have a non-negative expected return?

```
/// $[c = 0] \cdot \mathbb{E}[1] + [c \neq 0] \cdot (\frac{1}{2}\mathbb{E}[-5] + \frac{1}{2}\mathbb{E}[1])$
if ( $c = 0$ ) {
    /// $\mathbb{E}[1]$
    $x := 1$
    /// $\mathbb{E}[x]$
} else {
    /// $\frac{1}{2}\mathbb{E}[-5] + \frac{1}{2}\mathbb{E}[1]$
    { $x := -5$ } $\left[\frac{1}{2}\right]$ { $x := 1$ }
    /// $\mathbb{E}[x]$
}
/// $\mathbb{E}[x]$
```

With an input boolean variable $c$ we represent whether we have a fair or a fake coin. We represent the game with the simple program $C$ above and compute $\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[x])$ which yields the expected return for a given input distribution. We observe that the shape of the input distribution must be $\mu = \frac{n-1}{n} \cdot \mathbf{1}_{c=0} + \frac{1}{n} \cdot \mathbf{1}_{c=1}$ and solve: $\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[x])\,(\mu) \geq 0$, leading to:

$$\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[x]) \left(\frac{n-1}{n} \cdot \mathbf{1}_{c=0} + \frac{1}{n} \cdot \mathbf{1}_{c=1}\right) \geq 0$$

$$\left([c = 0] \cdot \mathbb{E}[1] + [c \neq 0] \cdot \left(\frac{1}{2}\mathbb{E}[-5] + \frac{1}{2}\mathbb{E}[1]\right)\right) \left(\frac{n-1}{n} \cdot \mathbf{1}_{c=0} + \frac{1}{n} \cdot \mathbf{1}_{c=1}\right) \geq 0$$

$$([c = 0] \cdot \mathbb{E}[1]) \left(\frac{n-1}{n} \cdot \mathbf{1}_{c=0} + \frac{1}{n} \cdot \mathbf{1}_{c=1}\right)$$
$$+ \left([c \neq 0] \cdot \left(\frac{1}{2}\mathbb{E}[-5] + \frac{1}{2}\mathbb{E}[1]\right)\right) \left(\frac{n-1}{n} \cdot \mathbf{1}_{c=0} + \frac{1}{n} \cdot \mathbf{1}_{c=1}\right) \geq 0$$

$$\frac{n-1}{n} - \frac{2}{n} \geq 0$$

$$\frac{n-3}{n} \geq 0$$
$$n \geq 3$$

The result obtained implies that you need at least 3 coins in your pocket (at least two fake coins and one fair coin) to guarantee a non-negative expected return in this coin-toss game.

### 5.8.5 Automation

Unsurprisingly, our whp calculus (in its full generality) cannot be fully automated, since we generalize existing undecidable calculi, expressing both termination and reachability properties for a Turing-complete computational model—both of which are known to be undecidable [106; 57].

For this reason, we have proposed a fully theoretical framework, providing a holistic view of different program logics and serving as a foundation for future tools to automate quantitative proofs. This approach is common in foundational program logic research such as Hoare Logic, Probabilistic PDL, Incorrectness Logic, Hyper Hoare Logic, and Outcome Logic.

Nevertheless, we believe that our calculi are at least syntactically mechanizable. Accordingly, we plan to investigate an expressive "assertion" language for hyperquantities, such as the one proposed by Batz et al. [102] for quantitative reasoning about probabilistic programs. This would allow us to prove *relative completeness* in the sense of Cook [107], i.e., decidability modulo checking whether $\mathbb{g} \preceq \mathbb{f}$ holds, where $\mathbb{g}, \mathbb{f}$ may contain suprema and infima. A similar result (decidability modulo checking a logical implication) is well known for classical predicate transformer and Hoare Logic [107]. Once such a relatively complete language is found, we expect it will be possible to fully automate whp reasoning for syntactic fragments of the programming and the assertion language.

## 5.9 Related Work

### Relational program logics

Relational Hoare Logics were initially introduced by Benton [65]. Subsequently, several extensions emerged, including to reason about probabilistic programs via couplings [178]. Later, Maillard et al. [179], proposed a general framework for developing relational program logics with effects based on Dijkstra Monads [180]. While effective, this framework is limited to 2-properties and thus does not apply to, e.g., monotonicity and transitivity, which are properties of *more than* two executions.

Sousa and Dillig [181]; D'Osualdo et al. [182] introduced logics for $k$-safety properties, but they cannot prove liveness. Dickerson et al. [183] introduced the first logic tailored for $\forall^*\exists^*$-hyperproperties, enabling, among others, proof and disproof of $k$-safety properties. Nonetheless, it has limited under-approximation capabilities: e.g., it does not suport incorrectness à la O'Hearn [6], and cannot disprove triples within the same logic. For instance, it cannot disprove GNI, a task which can only be completed by—to the best of our knowledge—HHL, OL, and our framework.

### Unified Program Logics

Similar to Outcome Logic (OL) [19; 20] and Weighted Programming [24], our calculus utilizes semirings to capture branch weights. This approach enables the development of a weakest-pre style calculus for Outcome Logic. While OL is relatively complete [74], the derivations are not always straightforward. Weakest hyper pre can be used to *mechanically* derive OL triples with the weakest precondition for a given postcondition. Weakest hyper pre also subsumes Hyper Hoare Logic [73], which is similar to OL, but specialized to nondeterministic programs.

Our approach surpasses Weighted Programming by facilitating reasoning about multiple outcomes. Our calculus also supports quantitative reasoning, demonstrating its versatility by encompassing various existing quantitative wp instances through the adaptation of hyperquantities.

### Predicate Transformers

These were first introduced by Dijkstra [1]; Dijkstra and Scholten [64], who created propositional weakest pre- and strongest postcondition calculi. Kozen [13]; McIver and Morgan [14] lifted these to a quantitative setting, introducing Probabilistic Propositional Dynamic Logic and weakest preexpectations for computing expected values over probabilistic programs. Many variants of weakest preexpectation now exist [16; 96]. We build on this line of work by extending these predicate transformers to hyperproperties. This gives us the flexibility to express a broader range of quantitative properties, as shown in Section 5.8.

### Hyper Predicate Transformers

The notion of Hyper Collecting Semantics [184; 185] is similar to our Strongest Hyper Postcondition in that it can prove valid Hyper Hoare Triples. However, it is not complete, as it does not provide the most precise hyper postcondition and is limited to non-quantitative properties.

## 5.10 Conclusion

Recent years have seen a focus on logics for proving properties other than classical partial correctness. E.g., program *security* is a *hyperproperty*, and *incorrectness* must *witness a faulty execution.*

Recent work on Outcome Logic [19; 74; 20] and Hyper Hoare Logic [73] has shown that all of these properties can be captured via a single proof system. In this chapter, we build upon those logics, but approach the problem using quantitative predicate transformers. This has allowed us to create a single calculus that can be used to *prove*, but also *disprove*, a variety of correctness properties. In addition, it can be used to derive advanced quantitative properties for programs too, such as variance in probabilistic programs.

The predicate transformer approach has two key benefits. First, it provides a calculus to mechanically derive specifications. Second, it finds the *most* *precise* pre, so as to remove guesswork around obtaining a precondition in the aforementioned logics. As we have demonstrated, this brings about new ways of proving—and disproving—hyperproperties for a variety of program types.

Chapter 6

# Conclusions & Future Work

This thesis has provided a holistic view of strongest-postcondition-style calculus and weakest-precondition-style calculus for quantitative program analysis, progressively elevating the reasoning to a more general setting by making the calculi parametrized to a class of semirings and to hyperproperties.

The primary goal of our research was to develop a quantitative strongest-post calculus for probabilistic programs, which we achieved in Chapter 4 (Definition 4.4.1). This achievement is validated by the numerous dualities we established between $\mathsf{wp}$ and $\mathsf{sp}$. Building on this foundation, we identified an opportunity to extend our framework even further. Specifically, we aimed to create a calculus that could compute the *initial* expected value of a quantity before program execution, given final state information—complementing the existing weakest precondition calculus (Definition 4.3.1) that computes *final* expected values from initial states.

This extension required more sophisticated transformers capable of handling probability distributions rather than individual states. In fact, by observing a single final state without further information, it is neither possible nor meaningful to compute the initial expected value of a quantity. This led to our development of quantitative hyper transformers in Chapter 5 (Definitions 5.3.3, 5.4.3, and 5.5.1), which successfully address this challenge by computing expected values bidirectionally across program execution.

The complexity of the rules for these hyper transformers, particularly for shp and slhp, increased significantly compared to our weighted calculi. This increased complexity stems from the inherent challenge of reversing program semantics through forward transformers. Despite this complexity, we believe that the calculi developed in this thesis are a significant step forward in the comprehension of predicate transformers.

For most chapters in this thesis, individual conclusions and directions for future work have already been provided. Rather than repeating those conclusions here, we will focus on drawing a broader picture and outlining several promising directions for future research that emerge from the themes explored throughout this thesis.

## The Intensional Approach

In this thesis, we have adopted an extensional approach to develop our calculi, focusing on semantic assertions to concentrate on the key mathematical properties of the objects studied in our framework. This approach is commonly followed by quantitative predicate transformer calculi [42; 13; 14; 16; 96; 24], as well as by program logics such as [73; 19; 74]. While the extensional approach often yields elegant formalisms, it is unsuitable for developing practical verification tools, which ultimately rely on some syntax.

Intensional approaches, on the other hand, rely on the actual assertion language syntax used to reason about program verification. The main complexity when reasoning about syntactic assertions is to preserve expressiveness, ensuring that the assertion language is sufficiently expressive to represent the properties of interest. This, in conjunction with the soundness and completeness of the assertion language, is a challenging problem that requires careful analysis [186]. For example, most probabilistic program verification techniques either take the extensional approach or do not aim for completeness.

Relative completeness of Hoare logic was shown by Cook [107]. Winskel [187] and Loeckx et al. [188] proved the expressiveness of first-order arithmetic for Dijkstra's weakest precondition calculus. For *separation logic* [118], expressiveness was demonstrated by Tatsuta et al. [189, 190], almost a decade after the logic was originally developed. For *loop-free* probabilistic programs and *restricted postconditions*, den Hartog and de Vink [191] proved relative completeness, leaving the expressiveness for loops as an open problem that has been solved only recently by Batz et al. [102].

We believe that the intensional approach is a promising direction for future research, as it can bridge the gap between the theoretical elegance of extensional approaches and the practicality of verification tools. In particular, we believe that developing an expressive and complete assertion language for our quantitative calculi is an open problem. Our weighted calculi (Chapter 4) support non-deterministic choices (even unbounded non-deterministic assignment!), which are not supported by Batz et al. [102]. Our hyperproperty calculi (Chapter 5) are even more expressive, and while some attempts to derive simpler syntactic rules have been put forward by Dardinier and Müller [73, Section 4], the problem of relative completeness when reasoning about syntactic assertions remains open.

### Compositional Program Analysis

In our work, we have developed a compositional approach to program analysis, where the analysis of a composite program is derived from the analysis of its components. This approach is particularly useful for modular verification, where the verification of a program is divided into smaller, more manageable parts. However, our compositional approach is currently limited to the analysis of programs that are composed sequentially or through non-deterministic choices, and is restricted to an intraprocedural setting.

A promising future direction is to extend our compositional approach to an interprocedural setting, as seen in works like [60; 192; 193]. In such a setting, the results of the analysis of smaller procedures can be evaluated, potentially in a distributed manner, and stored for on-demand and incremental querying [194].

Additionally, we believe that combining backward and forward reasoning may help to scale software verification. Similar to how tree pruning optimizes the search process in various algorithms, particularly in decision trees and search trees, we believe that weakest-precondition-style reasoning can be used to prune the search space of strongest-postcondition-style reasoning (and vice versa). Each calculus has unique advantages depending on the scenario: backward reasoning is fundamental when the postcondition is known, for example, when it represents some safety property or an incorrectness specification. On the other hand, forward reasoning is useful to determine what is reachable, for example, to determine if a bug is exploitable from the user. By combining these two approaches, we can leverage the strengths of each to develop a more efficient verification framework. Practically, this means using backward reasoning to eliminate infeasible paths early and forward reasoning to explore feasible paths more effectively. To the best of our knowledge, this has not been explored in the context of program verification; the closest work is [71], which is the first program logic that combines correctness and incorrectness (over and underapproximation), but does not consider forward and backward reasoning together. Another dimension that can be combined to achieve better precision is dynamic and static analysis, as seen in [195].

**Simpler Fragments of Quantitative Program Analysis**

In Section 5.6.4, we have studied a class of hyperquantities—namely additive hyperquantities—that enable simpler rules for our calculi. Unfortunately, the same class does not work for our forward hyper transformers, $\mathsf{shp}$ and $\mathsf{slhp}$. An open problem is to identify fragments that facilitate forward reasoning as well. By doing so, we could automatically develop specialized techniques for reasoning about these fragments. For example, in the realm of computing initial expected values, this could aid in solving problems such as Bayesian reasoning.

In general, while our hyper transformers are expressive, they are also complex, particularly our loop rules. It is still unclear whether simpler rules that can be defined as convergence of Kleene's iterates exist. If not, identifying simpler fragments that are still expressive enough to capture interesting properties remains a promising direction for future research.

### Higher-order Predicate Transformers

While hyperproperties offer a powerful framework for reasoning about complex program properties, there is potential to extend beyond their current scope. Future research could explore frameworks that encompass hyperhyperpredicates. In fact, the primary impediment that prevents common regular languages from accommodating demonic total correctness and angelic partial correctness, as defined in [16], stems from their use of the set of reachable states $\mathsf{sp}$ as semantics. This choice results in a loss of the ability to discern whether an initial state terminates or not, as the reachable states are merely aggregated. An open problem would be to extend our framework to consider unbounded hyper$^n$properties of type $\mathcal{P}(\ldots(\mathcal{P}(\Sigma)))$. Exploring the theoretical limits of hyperproperties is not only a theoretical challenge but also a practical one. By pushing the boundaries of what hyperproperties can express, we may gain deeper insights into existing program logics and uncover new patterns and properties that can be leveraged to develop more efficient verification tools.

Appendix A

# Appendix

## A.1 Proofs of Section 3.3

### A.1.1 Proof of Soundness for wp, Theorem 3.3.1

**Theorem 3.3.1** (Characterization of $\mathsf{wp}$). *For all programs $C$ and initial states $\sigma$,*

$$\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \quad = \quad \bigcurlyvee_{\tau\in[\![C]\!](\sigma)} f(\tau)~.$$

*Proof.* We prove Theorem 3.3.1 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The effectless program `skip`:**

We have

$$\begin{aligned}\mathsf{wp}\,[\![\mathtt{skip}]\!]\,(f)\,(\sigma) &= f(\sigma)\\ &= \sup_{\tau\in\{\sigma\}} f(\tau)\\ &= \sup_{\tau\in[\![\mathtt{skip}]\!](\sigma)} f(\tau)~.\end{aligned}$$

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}\mathsf{wp}\, [\![x \coloneqq e]\!]\, (f)\, (\sigma) &= f\, [x/e]\, (\sigma) \\ &= f(\sigma\, [x/\sigma(e)]) \\ &= \sup_{\tau \in \{\sigma[x/\sigma(e)]\}} f(\tau) \\ &= \sup_{\tau \in [\![x \coloneqq e]\!](\sigma)} f(\tau)\ .\end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$\begin{aligned}\mathsf{wp}\, [\![C_1 \fatsemi C_2]\!]\, (f)\, (\sigma) &= \mathsf{wp}\, [\![C_1]\!]\, (\mathsf{wp}\, [\![C_2]\!]\, (f))\, (\sigma) \\ &= \sup_{\tau' \in [\![C_1]\!](\sigma)} \mathsf{wp}\, [\![C_2]\!]\, (f)\, (\tau') && \text{(by I.H. on } C_1\text{)} \\ &= \sup_{\tau' \in [\![C_1]\!](\sigma) \wedge \tau \in [\![C_2]\!](\tau')} f(\tau) && \text{(by I.H. on } C_2\text{)} \\ &= \sup_{\tau \in [\![C_2]\!]([\![C_1]\!](\sigma))} f(\tau) \\ &= \sup_{\tau \in [\![C_1 \fatsemi C_2]\!](\sigma)} f(\tau)\ .\end{aligned}$$

**The conditional branching $\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}$:**

We have

$$\mathsf{wp}\, [\![\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}]\!]\, (f)\, (\sigma)$$

$$
\begin{aligned}
&= \big([\varphi] \curlywedge \mathsf{wp}\,[\![C_1]\!]\,(f) \;\;\curlyvee\;\; [\neg\varphi] \curlywedge \mathsf{wp}\,[\![C_2]\!]\,(f)\,\big)(\sigma) \\
&= \begin{cases} \mathsf{wp}\,[\![C_1]\!]\,(f)\,(\sigma) & \text{if } \sigma \models \varphi \\ \mathsf{wp}\,[\![C_2]\!]\,(f)\,(\sigma) & \text{otherwise} \end{cases} \\
&= \begin{cases} \sup_{\tau \in [\![C_1]\!](\sigma)} f(\tau) & \text{if } \sigma \models \varphi \\ \sup_{\tau \in [\![C_2]\!](\sigma)} f(\tau) & \text{otherwise} \end{cases} && \text{(by I.H. on } C_1, C_2) \\
&= \sup_{\tau \in ([\![C_1]\!] \circ [\![\varphi]\!])(\sigma) \cup ([\![C_2]\!] \circ [\![\neg\varphi]\!])(\sigma)} f(\tau) \\
&= \sup_{\tau \in [\![\mathtt{if}\,(\,\varphi\,)\,\{\,C_1\,\}\,\mathtt{else}\,\{\,C_2\,\}]\!](\sigma)} f(\tau)\ .
\end{aligned}
$$

**The nondeterministic choice $\{\,C_1\,\} \mathbin{\square} \{\,C_2\,\}$:**

We have

$$
\begin{aligned}
\mathsf{wp}\,[\![\{\,C_1\,\} \mathbin{\square} \{\,C_2\,\}]\!]\,(f)\,(\sigma) &= \big(\mathsf{wp}\,[\![C_1]\!]\,(f) \;\;\curlyvee\;\; \mathsf{wp}\,[\![C_2]\!]\,(f)\,\big)(\sigma) \\
&= \sup_{\tau \in [\![C_1]\!](\sigma)} f(\tau) \;\curlyvee\; \sup_{\tau \in [\![C_2]\!](\sigma)} f(\tau) \\
& \qquad \text{(by I.H. on } C_1, C_2) \\
&= \sup_{\tau \in [\![C_1]\!](\sigma) \cup [\![C_2]\!](\sigma)} f(\tau) \\
&= \sup_{\tau \in [\![\{\,C_1\,\} \square \{\,C_2\,\}]\!](\sigma)} f(\tau)\ .
\end{aligned}
$$

**The loop $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$:**

Let

$$
\Phi_f(X) \;=\; [\neg\varphi] \curlywedge f \;\;\curlyvee\;\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(X)\ ,
$$

be the $\mathsf{wp}$-characteristic functions of the loop $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$ with respect to postanticipation $f$ and

$$
F_S(X) \;=\; S \cup ([\![C]\!] \circ [\![\varphi]\!])X\ ,
$$

be the collecting semantics characteristic functions of the loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ with respect to any input $S \in \mathcal{P}(\texttt{Conf})$. We now prove by induction on $n$ that, for all $\sigma \in \Sigma$

$$\Phi_f^n(-\infty)(\sigma) \;=\; \sup_{\tau \in \llbracket \neg\varphi \rrbracket F^n_{\{\sigma\}}(\emptyset)} f(\tau)\ . \tag{A.1}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}
\Phi_f^0(-\infty)(\sigma) \;&=\; -\infty \\
&=\; \sup \emptyset \\
&=\; \sup_{\tau \in \emptyset} f(\tau) \\
&=\; \sup_{\tau \in \llbracket \neg\varphi \rrbracket F^0_{\{\sigma\}}(\emptyset)} f(\tau)\ .
\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$ and all $\sigma \in \Sigma$,

$$\Phi_f^n(-\infty)(\sigma) \;=\; \sup_{\tau \in \llbracket \neg\varphi \rrbracket F^n_{\{\sigma\}}(\emptyset)} f(\tau)\ .$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}
&\Phi_f^{n+1}(-\infty)(\sigma) \\
&=\; ([\neg\varphi] \curlywedge f)\,(\sigma) \;\curlyvee\; \big([\varphi] \curlywedge \mathsf{wp}\,\llbracket C \rrbracket\,\big(\Phi_f^n(-\infty)\big)\big)\,(\sigma) \\
&=\; ([\neg\varphi] \curlywedge f)(\sigma) \;\curlyvee \sup_{\tau \in \llbracket C \rrbracket(\sigma) \wedge \sigma \;\models\; \varphi} \Phi_f^n(-\infty)(\tau) && \text{(by I.H. on } C\text{)} \\
&=\; \begin{cases} \sup_{\tau \in \llbracket C \rrbracket(\sigma)} \Phi_f^n(-\infty)(\tau) & \text{if } \sigma \;\models\; \varphi \\ f(\sigma) & \text{otherwise} \end{cases} \\
&=\; \begin{cases} \sup_{\tau \in \llbracket C \rrbracket(\sigma)} \;\sup_{\tau' \in \llbracket \neg\varphi \rrbracket\; F^n_{\{\tau\}}(\emptyset)} f(\tau') & \text{if } \sigma \;\models\; \varphi \\ f(\sigma) & \text{otherwise} \end{cases} && \text{(by I.H. on } n\text{)} \\
&=\; \begin{cases} \sup_{\tau' \in \llbracket \neg\varphi \rrbracket\; F^n_{\llbracket C \rrbracket(\sigma)}(\emptyset)} f(\tau') & \text{if } \sigma \;\models\; \varphi \\ f(\sigma) & \text{otherwise} \end{cases}
\end{aligned}$$

$$
\begin{aligned}
&= \begin{cases} \sup_{\tau' \in [\![\neg\varphi]\!] \; F^n_{([\![C]\!]\circ[\varphi])(\sigma)}(\emptyset)} f(\tau') & \text{if } \sigma \models \varphi \\ f(\sigma) & \text{otherwise} \end{cases} \\
&= \sup_{\tau' \in [\![\neg\varphi]\!](\{\sigma\} \cup F^n_{([\![C]\!]\circ[\![\varphi]\!])(\sigma)}(\emptyset))} f(\tau') \\
&= \sup_{\tau \in [\![\neg\varphi]\!] F^{n+1}_{\{\sigma\}}(\emptyset)} f(\tau) \ .
\end{aligned}
$$

This concludes the induction on $n$. Now we have:

$$
\begin{aligned}
\mathsf{wp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)\,(\sigma) &= \big(\mathsf{lfp}\; X\colon\; [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(X)\big)(\sigma) \\
&= \sup_{n\in\mathbb{N}} \Phi^n_f(-\infty)(\sigma) \\
&\qquad\qquad \text{(By Kleene's fixpoint theorem)} \\
&= \sup_{n\in\mathbb{N}} \; \sup_{\tau\in[\![\neg\varphi]\!] F^n_{\{\sigma\}}(\emptyset)} f(\tau) \qquad \text{(by Equation A.1)} \\
&= \sup_{\tau\in\cup_{n\in\mathbb{N}}([\![\neg\varphi]\!] F^n_{\{\sigma\}}(\emptyset))} f(\tau) \\
&= \sup_{\tau\in[\![\neg\varphi]\!](\cup_{n\in\mathbb{N}} F^n_{\{\sigma\}}(\emptyset))} f(\tau) \\
&\qquad\qquad \text{(by continuity of } [\![\neg\varphi]\!]) \\
&= \sup_{\tau\in[\![\neg\varphi]\!](\mathsf{lfp}\;\; X\colon\, \{\sigma\}\cup([\![C]\!]\circ[\![\varphi]\!])X)} f(\tau) \\
&\qquad\qquad \text{(by Kleene's fixpoint theorem)} \\
&= \sup_{\tau\in[\![\texttt{while}(\,\varphi\,)\{\,C\,\}]\!](\sigma)} f(\tau) \ ,
\end{aligned}
$$

and this concludes the proof. □

### A.1.2 Proof of Soundness for wlp, Theorem 3.3.2

**Theorem 3.3.2** (Characterization of wlp)**.** *For all programs $C$ and states $\sigma \in \Sigma$,*

$$
\mathsf{wlp}[\![C]\!]\,(f)\,(\sigma) \quad = \quad \bigcurlywedge_{\tau\in[\![C]\!](\sigma)} f(\tau) \ .
$$

*Proof.* We prove Theorem 3.3.2 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The effectless program skip:**

We have

$$\begin{aligned}\mathsf{wlp}[\![\texttt{skip}]\!]\,(f)\,(\sigma) &= f(\sigma)\\ &= \inf_{\tau\in\{\sigma\}} f(\tau)\\ &= \inf_{\tau\in[\![\texttt{skip}]\!](\sigma)} f(\tau)\ .\end{aligned}$$

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}\mathsf{wlp}[\![x \coloneqq e]\!]\,(f)\,(\sigma) &= f\,[x/e]\,(\sigma)\\ &= f(\sigma\,[x/\sigma(e)])\\ &= \inf_{\tau\in\{\sigma[x/\sigma(e)]\}} f(\tau)\\ &= \inf_{\tau\in[\![x\coloneqq e]\!](\sigma)} f(\tau)\ .\end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1\,\fatsemi\,C_2$:**

We have

$$\begin{aligned}\mathsf{wlp}[\![C_1\,\fatsemi\,C_2]\!]\,(f)\,(\sigma) &= \mathsf{wlp}[\![C_1]\!]\,(\mathsf{wlp}[\![C_2]\!]\,(f))\,(\sigma)\\ &= \inf_{\tau'\in[\![C_1]\!](\sigma)} \mathsf{wlp}[\![C_2]\!]\,(f)\,(\tau') && \text{(by I.H. on } C_1)\\ &= \inf_{\tau'\in[\![C_1]\!](\sigma)\wedge\tau\in[\![C_2]\!](\tau')} f(\tau) && \text{(by I.H. on } C_2)\end{aligned}$$

$$
\begin{aligned}
&= \inf_{\tau \in [\![C_2]\!]([\![C_1]\!](\sigma))} f(\tau) \\
&= \inf_{\tau \in [\![C_1 \mathbin{⨟} C_2]\!](\sigma)} f(\tau) ~.
\end{aligned}
$$

**The conditional branching $\texttt{if}~(\varphi)~\{C_1\}~\texttt{else}~\{C_2\}$:**

We have

$$
\begin{aligned}
&\mathsf{wlp}[\![\texttt{if}~(\varphi)~\{C_1\}~\texttt{else}~\{C_2\}]\!]\,(f)\,(\sigma) \\
&= \big([\varphi] \curlywedge \mathsf{wlp}[\![C_1]\!]\,(f) ~~\curlyvee~~ [\neg\varphi] \curlywedge \mathsf{wlp}[\![C_2]\!]\,(f)\big)(\sigma) \\
&= \begin{cases} \mathsf{wlp}[\![C_1]\!]\,(f)\,(\sigma) & \text{if } \sigma \models \varphi \\ \mathsf{wlp}[\![C_2]\!]\,(f)\,(\sigma) & \text{otherwise} \end{cases} \\
&= \begin{cases} \inf_{\tau \in [\![C_1]\!](\sigma)} f(\tau) & \text{if } \sigma \models \varphi \\ \inf_{\tau \in [\![C_2]\!](\sigma)} f(\tau) & \text{otherwise} \end{cases} \qquad \text{(by I.H. on } C_1, C_2) \\
&= \inf_{\tau \in ([\![C_1]\!] \circ [\![\varphi]\!])(\sigma) \cup ([\![C_2]\!] \circ [\![\neg\varphi]\!])(\sigma)} f(\tau) \\
&= \inf_{\tau \in [\![\texttt{if}~(\varphi)~\{C_1\}~\texttt{else}~\{C_2\}]\!](\sigma)} f(\tau) ~.
\end{aligned}
$$

**The nondeterministic choice $\{C_1\} \mathrel{\square} \{C_2\}$:**

We have

$$
\begin{aligned}
\mathsf{wlp}[\![\{C_1\} \mathrel{\square} \{C_2\}]\!]\,(f)\,(\sigma) &= \big(\mathsf{wlp}[\![C_1]\!]\,(f) ~\curlywedge~ \mathsf{wlp}[\![C_2]\!]\,(f)\big)(\sigma) \\
&= \inf_{\tau \in [\![C_1]\!](\sigma)} f(\tau) ~\curlywedge~ \inf_{\tau \in [\![C_2]\!](\sigma)} f(\tau) \\
&\qquad \text{(by I.H. on } C_1, C_2) \\
&= \inf_{\tau \in [\![C_1]\!](\sigma) \cup [\![C_2]\!](\sigma)} f(\tau) \\
&= \inf_{\tau \in [\![\{C_1\} \square \{C_2\}]\!](\sigma)} f(\tau) ~.
\end{aligned}
$$

**The loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$:**

Let

$$\Phi_f(X) \;=\; [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(X)\;\;,$$

be the $\mathsf{wlp}$-characteristic functions of the loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ with respect to postanticipation $f$ and

$$F_S(X) \;=\; S \cup ([\![C]\!] \circ [\![\varphi]\!])X\;\;,$$

be the collecting semantics characteristic functions of the loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ with respect to any input $S \in \mathcal{P}(\texttt{Conf})$. We now prove by induction on $n$ that, for all $\sigma \in \Sigma$

$$\Phi^n_f(+\infty)(\sigma) \;=\; \inf_{\tau \in [\![\neg\varphi]\!] F^n_{\{\sigma\}}(\emptyset)} f(\tau)\;\;. \tag{A.2}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}
\Phi^0_f(+\infty)(\sigma) \;&=\; +\infty \\
&=\; \inf \emptyset \\
&=\; \inf_{\tau \in \emptyset} f(\tau) \\
&=\; \inf_{\tau \in [\![\neg\varphi]\!] F^0_{\{\sigma\}}(\emptyset)} f(\tau)\;\;.
\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$ and all $\sigma \in \Sigma$,

$$\Phi^n_f(+\infty)(\sigma) \;=\; \inf_{\tau \in [\![\neg\varphi]\!] F^n_{\{\sigma\}}(\emptyset)} f(\tau)\;\;.$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}
&\Phi^{n+1}_f(+\infty)(\sigma) \\
&\quad=\; ([\neg\varphi] \curlywedge f)\,(\sigma) \;\curlyvee\; \big([\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,\big(\Phi^n_f(+\infty)\big)\big)\,(\sigma) \\
&\quad=\; ([\neg\varphi] \curlywedge f)(\sigma) \;\curlyvee\; [\varphi]\,(\sigma) \curlywedge \inf_{\tau \in [\![C]\!](\sigma)} \Phi^n_f(+\infty)(\tau) \qquad\qquad \text{(by I.H. on } C\text{)}
\end{aligned}$$

$$
\begin{aligned}
&= \begin{cases} \inf_{\tau \in [\![C]\!](\sigma)} \Phi_f^n(+\infty)(\tau) & \text{if } \sigma \models \varphi \\ f(\sigma) & \text{otherwise} \end{cases} \\
&= \begin{cases} \inf_{\tau \in [\![C]\!](\sigma)} \ \inf_{\tau' \in [\![\neg\varphi]\!] \ F^n_{\{\tau\}}(\emptyset)} f(\tau') & \text{if } \sigma \models \varphi \\ f(\sigma) & \text{otherwise} \end{cases} && \text{(by I.H. on } n\text{)} \\
&= \begin{cases} \inf_{\tau' \in [\![\neg\varphi]\!] \ F^n_{[\![C]\!](\sigma)}(\emptyset)} f(\tau') & \text{if } \sigma \models \varphi \\ f(\sigma) & \text{otherwise} \end{cases} \\
&= \begin{cases} \inf_{\tau' \in [\![\neg\varphi]\!] \ F^n_{([\![C]\!] \circ [\varphi])(\sigma)}(\emptyset)} f(\tau') & \text{if } \sigma \models \varphi \\ f(\sigma) & \text{otherwise} \end{cases} \\
&= \inf_{\tau' \in [\![\neg\varphi]\!](\{\sigma\} \cup F^n_{([\![C]\!] \circ [\![\varphi]\!])(\sigma)}(\emptyset))} f(\tau') \\
&= \inf_{\tau \in [\![\neg\varphi]\!] F^{n+1}_{\{\sigma\}}(\emptyset)} f(\tau) \ .
\end{aligned}
$$

This concludes the induction on $n$. Now we have:

$$
\begin{aligned}
\mathsf{wlp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)\,(\sigma) &= \big(\mathsf{gfp}\ X\colon\ [\neg\varphi] \curlywedge f \ \curlyvee \ [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(X)\big)(\sigma) \\
&= \inf_{n \in \mathbb{N}} \Phi_f^n(+\infty)(\sigma) \\
& \qquad \text{(by Kleene's fixpoint theorem)} \\
&= \inf_{n \in \mathbb{N}} \ \inf_{\tau \in [\![\neg\varphi]\!] F^n_{\{\sigma\}}(\emptyset)} f(\tau) \qquad \text{(by Equation A.2)} \\
&= \inf_{\tau \in \cup_{n \in \mathbb{N}}([\![\neg\varphi]\!] F^n_{\{\sigma\}}(\emptyset))} f(\tau) \\
&= \inf_{\tau \in [\![\neg\varphi]\!](\cup_{n \in \mathbb{N}} F^n_{\{\sigma\}}(\emptyset))} f(\tau) \\
& \qquad \text{(by continuity of } [\![\neg\varphi]\!]\text{)} \\
&= \inf_{\tau \in [\![\neg\varphi]\!](\mathsf{lfp}\ X\colon \{\sigma\} \cup ([\![C]\!] \circ [\![\varphi]\!])X)} f(\tau) \\
& \qquad \text{(by Kleene's fixpoint theorem)} \\
&= \inf_{\tau \in [\![\mathtt{while}(\,\varphi\,)\{\,C\,\}]\!](\sigma)} f(\tau) \ ,
\end{aligned}
$$

and this concludes the proof. $\square$

## A.2 Proofs of Section 3.4

### A.2.1 Proof of Soundness for sp, Theorem 3.4.1

**Theorem 3.4.1** (Characterization of sp). *For all programs $C$ and final states $\tau$,*

$$\mathsf{sp}\,[\![C]\!]\,(f)\,(\tau) \quad = \bigcurlyvee_{\sigma \text{ with } \tau\in[\![C]\!](\sigma)} f(\sigma)\ .$$

*Proof.* We prove Theorem 3.4.1 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The effectless program skip:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![\mathtt{skip}]\!]\,(f)\,(\tau) &= f(\tau) \\
&= \sup_{\sigma\in\Sigma,\tau\in\{\sigma\}} f(\sigma) \\
&= \sup_{\sigma\in\Sigma,\tau\in[\![\mathtt{skip}]\!](\sigma)} f(\sigma)\ .
\end{aligned}$$

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq e]\!]\,(f)\,(\tau) &= (\reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge f\,[x/\alpha])(\tau) \\
&= (\sup_{\alpha}\ \ [x = e\,[x/\alpha]] \curlywedge f\,[x/\alpha])(\tau) \\
&= \sup_{\alpha\colon\,\tau(x)=\tau(e[x/\alpha])} (f\,[x/\alpha])(\tau) \\
&= \sup_{\alpha\colon\,\tau(x)=\tau(e[x/\alpha])} f(\tau\,[x/\alpha]) \\
&= \sup_{\alpha\colon\,\tau[x/\alpha][x/\tau(e[x/\alpha])]=\tau} f(\tau\,[x/\alpha]) \\
&= \sup_{\alpha\colon\,\tau[x/\alpha][x/\tau[x/\alpha](e)]=\tau} f(\tau\,[x/\alpha])
\end{aligned}$$

$$
\begin{aligned}
&= \sup_{\sigma\in\Sigma,\sigma[x/\sigma(e)]=\tau} f(\sigma) && \text{(By taking } \sigma = \tau\,[x/\alpha])\\
&= \sup_{\sigma\in\Sigma,\tau\in\{\sigma[x/\sigma(e)]\}} f(\sigma)\\
&= \sup_{\sigma\in\Sigma,\tau\in[\![x:=e]\!](\sigma)} f(\sigma)\ .
\end{aligned}
$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 ⨟ C_2$:**

We have

$$
\begin{aligned}
\mathsf{sp}\,[\![C_2 ⨟ C_1]\!]\,(f)\,(\tau) &= \mathsf{sp}\,[\![C_2]\!]\,(\mathsf{sp}\,[\![C_1]\!]\,(f))\,(\tau)\\
&= \sup_{\sigma'\in\Sigma,\tau\in[\![C_2]\!](\sigma')} \mathsf{sp}\,[\![C_1]\!]\,(f)\,(\sigma') && \text{(by I.H. on } C_2)\\
&= \sup_{\sigma\in\Sigma,\tau\in[\![C_2]\!](\sigma')\wedge\sigma'\in[\![C_1]\!](\sigma)} f(\sigma) && \text{(by I.H. on } C_2)\\
&= \sup_{\sigma\in\Sigma,\tau\in[\![C_2]\!]([\![C_1]\!](\sigma))} f(\sigma)\\
&= \sup_{\sigma\in\Sigma,\tau\in[\![C_1 ⨟ C_2]\!](\sigma)} f(\sigma)\ .
\end{aligned}
$$

**The conditional branching `if` $(\,\varphi\,)\ \{\,C_1\,\}$ `else` $\{\,C_2\,\}$:**

We have

$$
\begin{aligned}
&\mathsf{sp}\,[\![\texttt{if}\ (\,\varphi\,)\ \{\,C_1\,\}\ \texttt{else}\ \{\,C_2\,\}]\!]\,(f)\,(\tau)\\
&= \big(\mathsf{sp}\,[\![C_1]\!]\,([\varphi] \curlywedge f)\ \curlyvee\ \mathsf{sp}\,[\![C_2]\!]\,([\neg\varphi] \curlywedge f)\big)(\tau)\\
&= \sup_{\sigma\in\Sigma,\tau\in[\![C_1]\!](\sigma)} ([\varphi] \curlywedge f)(\sigma)\ \curlyvee \sup_{\sigma\in\Sigma,\tau\in[\![C_2]\!](\sigma)} ([\neg\varphi] \curlywedge f)(\sigma)\\
&\qquad\qquad \text{(by I.H. on } C_1, C_2)
\end{aligned}
$$

$$\begin{aligned}
&= \sup_{\sigma\in\Sigma,\tau\in(\llbracket C_1\rrbracket\circ\llbracket\varphi\rrbracket)(\sigma)} f(\sigma) \curlyvee \sup_{\sigma\in\Sigma,\tau\in(\llbracket C_2\rrbracket\circ\llbracket\neg\varphi\rrbracket)(\sigma)} f(\sigma)\\
&= \sup_{\sigma\in\Sigma,\tau\in(\llbracket C_1\rrbracket\circ\llbracket\varphi\rrbracket)(\sigma)\cup(\llbracket C_2\rrbracket\circ\llbracket\neg\varphi\rrbracket)(\sigma)} f(\sigma)\\
&= \sup_{\sigma\in\Sigma,\tau\in\llbracket \mathtt{if}\,(\,\varphi\,)\,\{\,C_1\,\}\,\mathtt{else}\,\{\,C_2\,\}\rrbracket(\sigma)} f(\sigma)\ .
\end{aligned}$$

**The nondeterministic choice $\{\,C_1\,\}\,\Box\,\{\,C_2\,\}$:**

We have

$$\begin{aligned}
\mathsf{sp}\,\llbracket\{\,C_1\,\}\,\Box\,\{\,C_2\,\}\rrbracket\,(f)\,(\tau) &= \big(\mathsf{sp}\,\llbracket C_1\rrbracket\,(f) \;\curlyvee\; \mathsf{sp}\,\llbracket C_2\rrbracket\,(f)\,\big)(\tau)\\
&= \sup_{\sigma\in\Sigma,\tau\in\llbracket C_1\rrbracket(\sigma)} f(\sigma) \curlyvee \sup_{\sigma\in\Sigma,\tau\in\llbracket C_2\rrbracket(\sigma)} f(\sigma)\\
&\qquad\qquad\qquad\qquad\text{(by I.H. on } C_1, C_2)\\
&= \sup_{\sigma\in\Sigma,\tau\in\llbracket C_1\rrbracket(\sigma)\cup\llbracket C_2\rrbracket(\sigma)} f(\sigma)\\
&= \sup_{\sigma\in\Sigma,\tau\in\llbracket\{\,C_1\,\}\Box\{\,C_2\,\}\rrbracket(\sigma)} f(\sigma)\ .
\end{aligned}$$

**The loop $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$:**

Let

$$\Psi_f(X) \;=\; f \;\curlyvee\; \mathsf{sp}\,\llbracket C\rrbracket\,([\varphi] \curlywedge X)\ ,$$

be the $\mathsf{sp}$-characteristic functions of the loop $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$ with respect to preanticipation $f$ and

$$F_S(X) \;=\; S \cup (\llbracket C\rrbracket\circ\llbracket\varphi\rrbracket)X\ ,$$

be the collecting semantics characteristic functions of the loop $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$ with respect to any input $S\in\mathcal{P}(\mathtt{Conf})$. We now prove by induction on $n$ that, for all $\tau\in\Sigma$

$$\Psi_f^n(-\infty)(\tau) \;=\; \sup_{\sigma\in\Sigma,\tau\in F^n_{\{\sigma\}}(\emptyset)} f(\sigma)\ . \tag{A.3}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}\Psi_f^0(-\infty)(\tau) &= -\infty \\ &= \sup \emptyset \\ &= \sup_{\sigma \in \Sigma, \tau \in \emptyset} f(\sigma) \\ &= \sup_{\sigma \in \Sigma, \tau \in F^0_{\{\sigma\}}(\emptyset)} f(\sigma)\ .\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$ and all $\tau \in \Sigma$

$$\Psi_f^n(-\infty)(\tau) = \sup_{\sigma \in \Sigma, \tau \in F^n_{\{\sigma\}}(\emptyset)} f(\sigma)\ .$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}&\Psi_f^{n+1}(-\infty)(\tau) \\ &= \big(f \ \curlyvee\ \mathsf{sp}\,[\![C]\!]\,\big([\varphi] \curlywedge \Psi_f^n(-\infty)\big)\big)\,(\tau) \\ &= f(\tau) \ \curlyvee \sup_{\sigma \in \Sigma, \tau \in [\![C]\!](\sigma)} \big([\varphi] \curlywedge \Psi_f^n(-\infty)\big)(\sigma) && \text{(by I.H. on } C) \\ &= f(\tau) \ \curlyvee \sup_{\sigma \in \Sigma, \tau \in [\![C]\!](\sigma)} \ \sup_{\sigma' \in \Sigma, \sigma \in [\![\varphi]\!] F^n_{\{\sigma'\}}(\emptyset)} f(\sigma') && \text{(by I.H. on } n) \\ &= f(\tau) \ \curlyvee \sup_{\sigma' \in \Sigma, \tau \in ([\![C]\!] \circ [\![\varphi]\!]) F^n_{\{\sigma'\}}(\emptyset)} f(\sigma') \\ &= \sup_{\sigma' \in \Sigma, \tau \in ([\![C]\!] \circ [\![\varphi]\!]) F^n_{\{\sigma'\}}(\emptyset) \cup \{\sigma'\}} f(\sigma') \\ &= \sup_{\sigma \in \Sigma, \tau \in F^{n+1}_{\{\sigma\}}(\emptyset)} f(\sigma)\ .\end{aligned}$$

This concludes the induction on $n$. Now we have:

$$\begin{aligned}\mathsf{sp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)\,(\tau) &= \big([\neg\varphi] \curlywedge \big(\mathsf{lfp}\ X\colon f \ \curlyvee\ \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge X)\big)\big)\,(\tau) \\ &= \big([\neg\varphi] \curlywedge \sup_{n \in \mathbb{N}}\ \Psi_f^n(-\infty)\big)(\tau) \\ &\qquad\qquad \text{(by Kleene's fixpoint theorem)}\end{aligned}$$

$$
\begin{aligned}
&= \sup_{n\in\mathbb{N}} \big([\neg\varphi] \curlywedge \Psi_f^n(-\infty)\big)(\tau) \\
&\qquad\qquad\qquad\text{(by continuity of } \lambda X\colon\ [\neg\varphi] \curlywedge X\text{)} \\
&= \sup_{n\in\mathbb{N}} \sup_{\sigma\in\Sigma, \tau\in[\![\neg\varphi]\!]F^n_{\{\sigma\}}(\emptyset)} f(\sigma) \quad \text{(by Equation A.3)} \\
&= \sup_{\sigma\in\Sigma, \tau\in\cup_{n\in\mathbb{N}}([\![\neg\varphi]\!]F^n_{\{\sigma\}}(\emptyset))} f(\sigma) \\
&= \sup_{\sigma\in\Sigma, \tau\in[\![\neg\varphi]\!](\cup_{n\in\mathbb{N}}F^n_{\{\sigma\}}(\emptyset))} f(\sigma) \\
&\qquad\qquad\qquad\text{(by continuity of } [\![\neg\varphi]\!]\text{)} \\
&= \sup_{\sigma\in\Sigma, \tau\in[\![\neg\varphi]\!](\mathsf{lfp}\ X\colon \{\sigma\}\cup([\![C]\!]\circ[\![\varphi]\!])X)} f(\sigma) \\
&\qquad\qquad\qquad\text{(by Kleene's fixpoint theorem)} \\
&= \sup_{\sigma\in\Sigma, \tau\in[\![\mathtt{while}(\,\varphi\,)\{\,C\,\}]\!](\sigma)} f(\sigma)\ ,
\end{aligned}
$$

and this concludes the proof. □

### A.2.2 Proof of Soundness for slp, Theorem 3.4.2

**Theorem 3.4.2** (Characterization of slp)**.** *For all programs $C$ and states $\tau \in \Sigma$,*

$$
\mathsf{slp}[\![C]\!]\,(f)\,(\tau) \quad = \quad \bigcurlywedge_{\sigma \text{ with } \tau\in[\![C]\!]\sigma} f(\sigma)
$$

*Proof.* We prove Theorem 3.4.2 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The effectless program `skip`:**

We have

$$
\begin{aligned}
\mathsf{slp}[\![\mathtt{skip}]\!]\,(f)\,(\tau) &= f(\tau) \\
&= \inf_{\sigma\in\Sigma, \tau\in\{\sigma\}} f(\sigma) \\
&= \inf_{\sigma\in\Sigma, \tau\in[\![\mathtt{skip}]\!](\sigma)} f(\sigma)\ .
\end{aligned}
$$

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{slp}[\![x \coloneqq e]\!]\,(f)\,(\tau) &= (\boldsymbol{\ell}\,\alpha\colon\;\; [x \neq e\,[x/\alpha]] \curlyvee f\,[x/\alpha])(\tau) \\
&= (\inf_{\alpha}\;\; [x \neq e\,[x/\alpha]] \curlyvee f\,[x/\alpha])(\tau) \\
&= \inf_{\alpha\colon\, \tau(x)=\tau(e[x/\alpha])} \;(f\,[x/\alpha])(\tau) \\
&= \inf_{\alpha\colon\, \tau(x)=\tau(e[x/\alpha])} \;f(\tau\,[x/\alpha]) \\
&= \inf_{\alpha\colon\, \tau[x/\alpha][x/\tau(e[x/\alpha])]=\tau} \;f(\tau\,[x/\alpha]) \\
&= \inf_{\alpha\colon\, \tau[x/\alpha][x/\tau[x/\alpha](e)]=\tau} \;f(\tau\,[x/\alpha]) \\
&= \inf_{\sigma\in\Sigma, \sigma[x/\sigma(e)]=\tau} \;f(\sigma) \qquad\qquad (\text{By taking } \sigma = \tau\,[x/\alpha]) \\
&= \inf_{\sigma\in\Sigma, \tau\in\{\sigma[x/\sigma(e)]\}} f(\sigma) \\
&= \inf_{\sigma\in\Sigma, \tau\in[\![x\coloneqq e]\!](\sigma)} f(\sigma)\;.
\end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi\; C_2$:**

We have

$$\begin{aligned}
\mathsf{slp}[\![C_2 \fatsemi\; C_1]\!]\,(f)\,(\tau) &= \mathsf{slp}[\![C_2]\!]\,(\mathsf{slp}[\![C_1]\!]\,(f))\,(\tau) \\
&= \inf_{\sigma'\in\Sigma, \tau\in[\![C_2]\!](\sigma')} \mathsf{slp}[\![C_1]\!]\,(f)\,(\sigma') \qquad\qquad (\text{by I.H. on } C_2) \\
&= \inf_{\sigma\in\Sigma, \tau\in[\![C_2]\!](\sigma')\wedge\sigma'\in[\![C_1]\!](\sigma)} f(\sigma) \qquad\qquad (\text{by I.H. on } C_2) \\
&= \inf_{\sigma\in\Sigma, \tau\in[\![C_2]\!]([\![C_1]\!](\sigma))} f(\sigma)
\end{aligned}$$

$$= \inf_{\sigma\in\Sigma,\tau\in[\![C_1\,\mathbin{;}\,C_2]\!](\sigma)} f(\sigma)\ .$$

**The conditional branching $\texttt{if}\,(\,\varphi\,)\,\{\,C_1\,\}\,\texttt{else}\,\{\,C_2\,\}$:**

We have

$$\begin{aligned}
&\mathsf{slp}[\![\texttt{if}\,(\,\varphi\,)\,\{\,C_1\,\}\,\texttt{else}\,\{\,C_2\,\}]\!]\,(f)\,(\tau)\\
&= \big(\mathsf{slp}[\![C_1]\!]\,([\neg\varphi] \curlyvee f) \ \curlywedge\ \mathsf{slp}[\![C_2]\!]\,([\varphi] \curlyvee f)\big)(\tau)\\
&= \inf_{\sigma\in\Sigma,\tau\in[\![C_1]\!](\sigma)}([\neg\varphi] \curlyvee f)(\sigma) \ \curlywedge\ \inf_{\sigma\in\Sigma,\tau\in[\![C_2]\!](\sigma)}([\varphi] \curlyvee f)(\sigma)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{(by I.H. on } C_1, C_2)\\
&= \inf_{\sigma\in\Sigma,\tau\in([\![C_1]\!]\circ[\![\varphi]\!])(\sigma)} f(\sigma) \ \curlywedge\ \inf_{\sigma\in\Sigma,\tau\in([\![C_2]\!]\circ[\![\neg\varphi]\!])(\sigma)} f(\sigma)\\
&= \inf_{\sigma\in\Sigma,\tau\in([\![C_1]\!]\circ[\![\varphi]\!])(\sigma)\cup([\![C_2]\!]\circ[\![\neg\varphi]\!])(\sigma)} f(\sigma)\\
&= \inf_{\sigma\in\Sigma,\tau\in[\![\texttt{if}\,(\,\varphi\,)\,\{\,C_1\,\}\,\texttt{else}\,\{\,C_2\,\}]\!](\sigma)} f(\sigma)\ .
\end{aligned}$$

**The nondeterministic choice $\{\,C_1\,\}\,\square\,\{\,C_2\,\}$:**

We have

$$\begin{aligned}
\mathsf{slp}[\![\{\,C_1\,\}\,\square\,\{\,C_2\,\}]\!]\,(f)\,(\tau) &= \big(\mathsf{slp}[\![C_1]\!]\,(f) \ \curlywedge\ \mathsf{slp}[\![C_2]\!]\,(f)\,\big)(\tau)\\
&= \inf_{\sigma\in\Sigma,\tau\in[\![C_1]\!](\sigma)} f(\sigma) \ \curlywedge\ \inf_{\sigma\in\Sigma,\tau\in[\![C_2]\!](\sigma)} f(\sigma)\\
&\qquad\qquad\qquad\qquad\qquad\text{(by I.H. on } C_1, C_2)\\
&= \inf_{\sigma\in\Sigma,\tau\in[\![C_1]\!](\sigma)\cup[\![C_2]\!](\sigma)} f(\sigma)\\
&= \inf_{\sigma\in\Sigma,\tau\in[\![\{\,C_1\,\}\square\{\,C_2\,\}]\!](\sigma)} f(\sigma)\ .
\end{aligned}$$

**The loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$:**

Let

$$\Psi_f(X) \;=\; f \ \curlywedge\ \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee X)\ ,$$

be the slp-characteristic functions of the loop `while` ( $\varphi$ ) { $C$ } with respect to preanticipation $f$ and

$$F_S(X) \;=\; S \cup (\llbracket C \rrbracket \circ \llbracket \varphi \rrbracket) X \ ,$$

be the collecting semantics characteristic functions of the loop `while` ( $\varphi$ ) { $C$ } with respect to any input $S \in \mathcal{P}(\texttt{Conf})$. We now prove by induction on $n$ that, for all $\tau \in \Sigma$

$$\Psi_f^n(+\infty)(\tau) \;=\; \inf_{\sigma \in \Sigma, \tau \in F^n_{\{\sigma\}}(\emptyset)} f(\sigma) \ . \tag{A.4}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned} \Psi_f^0(+\infty)(\tau) &= +\infty \\ &= \inf \emptyset \\ &= \inf_{\sigma \in \Sigma, \tau \in \emptyset} f(\sigma) \\ &= \inf_{\sigma \in \Sigma, \tau \in F^0_{\{\sigma\}}(\emptyset)} f(\sigma) \ . \end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$ and all $\tau \in \Sigma$

$$\Psi_f^n(+\infty)(\tau) \;=\; \inf_{\sigma \in \Sigma, \tau \in F^n_{\{\sigma\}}(\emptyset)} f(\sigma) \ .$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned} &\Psi_f^{n+1}(+\infty)(\tau) \\ &= \big(f \ \curlywedge \ \mathsf{slp}\llbracket C \rrbracket \left([\neg\varphi] \curlyvee \Psi_f^n(+\infty)\right)\big)(\tau) \\ &= f(\tau) \ \curlywedge \ \inf_{\sigma \in \Sigma, \tau \in \llbracket C \rrbracket(\sigma)} \big([\neg\varphi] \curlyvee \Psi_f^n(+\infty)\big)(\sigma) && \text{(by I.H. on } C\text{)} \\ &= f(\tau) \ \curlywedge \ \inf_{\sigma \in \Sigma, \tau \in \llbracket C \rrbracket(\sigma)} \ \inf_{\sigma' \in \Sigma, \sigma \in \llbracket \varphi \rrbracket F^n_{\{\sigma'\}}(\emptyset)} f(\sigma') && \text{(by I.H. on } n\text{)} \\ &= f(\tau) \ \curlywedge \ \inf_{\sigma' \in \Sigma, \tau \in (\llbracket C \rrbracket \circ \llbracket \varphi \rrbracket) F^n_{\{\sigma'\}}(\emptyset)} f(\sigma') \\ &= \inf_{\sigma' \in \Sigma, \tau \in (\llbracket C \rrbracket \circ \llbracket \varphi \rrbracket) F^n_{\{\sigma'\}}(\emptyset) \cup \{\sigma'\}} f(\sigma') \end{aligned}$$

$$= \inf_{\sigma \in \Sigma, \tau \in F^{n+1}_{\{\sigma\}}(\emptyset)} f(\sigma) \ .$$

This concludes the induction on $n$. Now we have:

$$\begin{aligned}
\mathsf{slp}[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)\,(\tau) &= \big([\varphi] \curlyvee \big(\mathsf{gfp}\ X\colon f \ \curlywedge\ \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee X)\big)\big)\,(\tau) \\
&= \big([\varphi] \curlyvee \inf_{n\in\mathbb{N}}\ \Psi^n_f(+\infty)\big)(\tau) \\
&\qquad\qquad \text{(by Kleene's fixpoint theorem)} \\
&= \inf_{n\in\mathbb{N}}\ \big([\varphi] \curlyvee \Psi^n_f(+\infty)\big)(\tau) \\
&\qquad\qquad \text{(by co-continuity of } \lambda X\colon\ [\varphi] \curlyvee X) \\
&= \inf_{n\in\mathbb{N}}\ \inf_{\sigma\in\Sigma, \tau\in[\![\neg\varphi]\!]F^n_{\{\sigma\}}(\emptyset)} f(\sigma) \ \ \text{(by Equation A.4)} \\
&= \inf_{\sigma\in\Sigma, \tau\in\cup_{n\in\mathbb{N}}([\![\neg\varphi]\!]F^n_{\{\sigma\}}(\emptyset))} f(\sigma) \\
&= \inf_{\sigma\in\Sigma, \tau\in[\![\neg\varphi]\!](\cup_{n\in\mathbb{N}}F^n_{\{\sigma\}}(\emptyset))} f(\sigma) \\
&\qquad\qquad \text{(by continuity of } [\![\neg\varphi]\!]) \\
&= \inf_{\sigma\in\Sigma, \tau\in[\![\neg\varphi]\!](\mathsf{lfp}\ X\colon \{\sigma\}\cup([\![C]\!]\circ[\![\varphi]\!])X)} f(\sigma) \\
&\qquad\qquad \text{(by Kleene's fixpoint theorem)} \\
&= \inf_{\sigma\in\Sigma, \tau\in[\![\texttt{while}(\,\varphi\,)\{\,C\,\}]\!](\sigma)} f(\sigma) \ ,
\end{aligned}$$

and this concludes the proof. □

## A.3 Proofs of Section 3.5

### A.3.1 Proof of Healthiness Properties of Quantitative Transformers, Theorem 3.5.1

**Theorem 3.5.1** (Healthiness Properties of Quantitative Transformers)**.** *For all programs $C$, all quantitive transformers are monotonic, i.e.*

$$f \ \preceq\ g \quad \text{implies} \quad \mathsf{ttt}\,[\![C]\!]\,(f) \ \preceq\ \mathsf{ttt}\,[\![C]\!]\,(g) \ , \quad \text{for } \mathsf{ttt} \in \{\mathsf{wp},\, \mathsf{wlp},\, \mathsf{sp},\, \mathsf{slp}\} \ .$$

*The non-liberal transformers* $\mathsf{wp}[\![C]\!]$ *and* $\mathsf{sp}[\![C]\!]$ *satisfy the following properties:*

1. *Quantitative universal disjunctiveness: For any set of quantities* $S \subseteq \mathbb{A}$*,*

$$\mathsf{wp}\,[\![C]\!]\,(\curlyvee S) \;=\; \curlyvee\, \mathsf{wp}\,[\![C]\!]\,(S) \quad \text{and} \quad \mathsf{sp}\,[\![C]\!]\,(\curlyvee S) \;=\; \curlyvee\, \mathsf{sp}\,[\![C]\!]\,(S)~.$$

2. *Strictness:* $\mathsf{wp}\,[\![C]\!]\,(-\infty) \;=\; -\infty \quad \text{and} \quad \mathsf{sp}\,[\![C]\!]\,(-\infty) \;=\; -\infty~.$

*The liberal transformers* $\mathsf{wlp}[\![C]\!]$ *and* $\mathsf{slp}[\![C]\!]$ *satisfy the following properties:*

3. *Quantitative universal conjunctiveness: For any set of quantities* $S \subseteq \mathbb{A}$*,*

$$\mathsf{wlp}[\![C]\!]\,(\curlywedge S) \;=\; \curlywedge\, \mathsf{wlp}[\![C]\!]\,(S) \quad \text{and} \quad \mathsf{slp}[\![C]\!]\,(\curlywedge S) \;=\; \curlywedge\, \mathsf{slp}[\![C]\!]\,(S)~.$$

4. *Costrictness:* $\mathsf{wlp}[\![C]\!]\,(+\infty) \;=\; +\infty \quad \text{and} \quad \mathsf{slp}[\![C]\!]\,(+\infty) \;=\; +\infty~.$

*Proof.* Each of the properties is proven individually below.

- Quantitative universal disjunctiveness: Theorems A.3.1 and A.3.2;
- Quantitative universal conjunctiveness: Theorems A.3.3 and A.3.4;
- Strictness: Corollaries A.3.4.1 and A.3.4.2;
- Costrictness: Corollaries A.3.4.3 and A.3.4.4;
- Monotonicity: Corollary A.3.4.5

□

**Theorem A.3.1** (Quantitative universal disjunctiveness of $\mathsf{wp}$)**.** *For any set of quantities* $\subseteq \mathbb{A}$*,*

$$\mathsf{wp}\,[\![C]\!]\,(\sup S) \quad = \quad \sup\; \mathsf{wp}\,[\![C]\!]\,(S)~.$$

*Proof.* We prove Theorem A.3.1 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The effectless program skip:**

We have

$$\begin{aligned}\mathsf{wp}\,[\![\texttt{skip}]\!]\,(\sup S) &= \sup S\\ &= \sup_{g\in S} g\\ &= \sup_{g\in S}\ \mathsf{wp}\,[\![\texttt{skip}]\!]\,(g)\\ &= \sup\ \mathsf{wp}\,[\![\texttt{skip}]\!]\,(S)\ .\end{aligned}$$

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}\mathsf{wp}\,[\![x \coloneqq e]\!]\,(\sup S) &= (\sup S)\,[x/e]\\ &= \left(\lambda\sigma\colon\ \sup_{g\in S} g(\sigma)\right)[x/e]\\ &= \left(\lambda\sigma\colon\ \sup_{g\in S} g\,[x/e]\,(\sigma)\right)\\ &= \sup_{g\in S} g\,[x/e]\\ &= \sup_{g\in S}\ \mathsf{wp}\,[\![x \coloneqq e]\!]\,(g)\\ &= \sup\ \mathsf{wp}\,[\![x \coloneqq e]\!]\,(S)\ .\end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, Theorem A.3.1 holds.

We proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$\begin{aligned}
\mathsf{wp}\,[\![C_1 \fatsemi C_2]\!]\,(\sup S) &= \mathsf{wp}\,[\![C_1]\!]\,(\mathsf{wp}\,[\![C_2]\!]\,(\sup S)) \\
&= \mathsf{wp}\,[\![C_1]\!]\,(\sup\ \mathsf{wp}\,[\![C_2]\!]\,(S)) && \text{(by I.H. on } C_2\text{)} \\
&= \sup\ \mathsf{wp}\,[\![C_1]\!]\,(\mathsf{wp}\,[\![C_2]\!]\,(S)) && \text{(by I.H. on } C_1\text{)} \\
&= \sup\ \mathsf{wp}\,[\![C_1 \fatsemi C_2]\!]\,(S)\ .
\end{aligned}$$

**The nondeterministic choice $\{\,C_1\,\} \mathrel{\Box} \{\,C_2\,\}$:**

We have

$$\begin{aligned}
\mathsf{wp}\,[\![\{\,C_1\,\} \mathrel{\Box} \{\,C_2\,\}]\!]\,(\sup S) &= \mathsf{wp}\,[\![C_1]\!]\,(f) \ \curlyvee\ \mathsf{wp}\,[\![C_2]\!]\,(f) \\
&= \sup_{g \in S}\ \mathsf{wp}\,[\![C_1]\!]\,(g) \ \curlyvee\ \sup_{g \in S}\ \mathsf{wp}\,[\![C_2]\!]\,(g) \\
&\qquad\qquad\qquad \text{(by I.H. on } C_1, C_2\text{)} \\
&= \sup_{g \in S}\ \mathsf{wp}\,[\![C_1]\!]\,(g) \ \curlyvee\ \mathsf{wp}\,[\![C_2]\!]\,(g) \\
&= \sup_{g \in S}\ \mathsf{wp}\,[\![\{\,C_1\,\} \mathrel{\Box} \{\,C_2\,\}]\!]\,(g)\ .
\end{aligned}$$

**The conditional branching $\texttt{if}\ (\,\varphi\,)\ \{\,C_1\,\}\ \texttt{else}\ \{\,C_2\,\}$:**

Here we reason in the reverse direction from the cases before. We have

$$\begin{aligned}
&\mathsf{wp}\,[\![\texttt{if}\ (\,\varphi\,)\ \{\,C_1\,\}\ \texttt{else}\ \{\,C_2\,\}]\!]\,(\sup S) \\
&= [\varphi] \curlywedge \mathsf{wp}\,[\![C_1]\!]\,(\sup S) \ \curlyvee\ [\neg\varphi] \curlywedge \mathsf{wp}\,[\![C_2]\!]\,(\sup S) \\
&= [\varphi] \curlywedge \sup\ \mathsf{wp}\,[\![C_1]\!]\,(S) \ \curlyvee\ [\neg\varphi] \curlywedge \sup\ \mathsf{wp}\,[\![C_2]\!]\,(S) \\
&\qquad\qquad\qquad \text{(by I.H. on } C_1 \text{ and } C_2\text{)} \\
&= \sup\ \big([\varphi] \curlywedge \mathsf{wp}\,[\![C_1]\!]\,(S)\big) \ \curlyvee\ \sup\ \big([\neg\varphi] \curlywedge \mathsf{wp}\,[\![C_2]\!]\,(S)\big) \\
&= \sup\ \big([\varphi] \curlywedge \mathsf{wp}\,[\![C_1]\!]\,(S) \ \curlyvee\ [\neg\varphi] \curlywedge \mathsf{wp}\,[\![C_2]\!]\,(S)\big) \\
&= \sup\ \mathsf{wp}\,[\![\texttt{if}\ (\,\varphi\,)\ \{\,C_1\,\}\ \texttt{else}\ \{\,C_2\,\}]\!]\,(S)\ .
\end{aligned}$$

**The loop** $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$**:**

Let

$$\Phi_f(X) \;=\; [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(X) \;,$$

be the $\mathsf{wp}$-characteristic function of the loop $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$ with respect to any postanticipation $f \in \mathbb{A}$. Observe that $\Phi_f(X)$ is continuous by inductive hypothesis on $C$ and by composition of continuous functions. We now prove by induction on $n$ that

$$\Phi^n_{\sup S}(-\infty) \;=\; \sup_{g \in S} \Phi^n_g(-\infty) \;. \tag{A.5}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}
\Phi^0_{\sup S}(-\infty) \;&=\; -\infty \\
&=\; \sup_{g \in S} -\infty \\
&=\; \sup_{g \in S} \Phi^0_g(-\infty) \;.
\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$

$$\Phi^n_{\sup S}(-\infty) \;=\; \sup_{g \in S} \Phi^n_g(-\infty) \;.$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}
&\Phi^{n+1}_{\sup S}(-\infty) \\
&=\; [\neg\varphi] \curlywedge \sup S \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\left(\Phi^n_{\sup S}(-\infty)\right) \\
&=\; [\neg\varphi] \curlywedge \sup S \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\left(\sup_{g \in S} \Phi^n_g(-\infty)\right) && \text{(by I.H. on } n\text{)} \\
&=\; [\neg\varphi] \curlywedge \sup S \;\curlyvee\; [\varphi] \curlywedge \sup_{g \in S}\; \mathsf{wp}\,[\![C]\!]\left(\Phi^n_g(-\infty)\right) && \text{(by I.H. on } C\text{)} \\
&=\; \sup_{g \in S}\; \left([\neg\varphi] \curlywedge g\right) \;\curlyvee\; \sup_{g \in S}\; \left([\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\left(\Phi^n_g(-\infty)\right)\right)
\end{aligned}$$

$$
\begin{aligned}
&= \sup_{g\in S} \big([\neg\varphi] \curlywedge g \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(\Phi_g^n(-\infty))\big) \\
&= \sup_{g\in S} \Phi_g^{n+1}(-\infty)\ .
\end{aligned}
$$

This concludes the induction on $n$. Now we have:

$$
\begin{aligned}
\mathsf{wp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(\sup S) &= \mathsf{lfp}\ X\colon\ [\neg\varphi] \curlywedge \sup S \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(X) \\
&= \sup_{n\in\mathbb{N}}\ \Phi^n_{\sup S}(-\infty) \\
&\qquad\qquad\text{(by Kleene's fixpoint theorem)} \\
&= \sup_{n\in\mathbb{N}}\ \sup_{g\in S}\ \Phi_g^n(-\infty) \qquad \text{(by Equation A.5)} \\
&= \sup_{g\in S}\ \sup_{n\in\mathbb{N}}\ \Phi_g^n(-\infty) \\
&= \sup_{g\in S} \mathsf{wp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g) \\
&\qquad\qquad\text{(by Kleene's fixpoint theorem)} \\
&= \sup \mathsf{wp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(S)\,,
\end{aligned}
$$

and this concludes the proof. □

**Theorem A.3.2** (Quantitative universal disjunctiveness of sp)**.** *For any set of quantities* $\subseteq \mathbb{A}$*,*

$$
\mathsf{sp}\,[\![C]\!]\,(\sup S) \quad = \quad \sup\ \mathsf{sp}\,[\![C]\!]\,(S)\ .
$$

*Proof.* We prove Theorem A.3.2 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The effectless program `skip`:**

We have

$$
\mathsf{sp}\,[\![\mathtt{skip}]\!]\,(\sup S) \;=\; \sup S
$$

$$
\begin{aligned}
&= \sup_{g \in S} g \\
&= \sup_{g \in S} \ \mathsf{sp}\,[\![\texttt{skip}]\!]\,(g) \\
&= \sup\ \mathsf{sp}\,[\![\texttt{skip}]\!]\,(S)\ .
\end{aligned}
$$

**The assignment $x \coloneqq e$:**

We have

$$
\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\sup S) \ &= \ \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge (\sup S)\,[x/\alpha] \\
&= \ \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge \left(\lambda\sigma\colon\ \sup_{g \in S} g(\sigma)\right)[x/\alpha] \\
&= \ \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge \left(\lambda\sigma\colon\ \sup_{g \in S} g\,[x/\alpha]\,(\sigma)\right) \\
&= \ \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge \sup_{g \in S} g\,[x/\alpha] \\
&= \ \reflectbox{$\mathsf{S}$}\alpha\colon\ \ \sup_{g \in S}\,[x = e\,[x/\alpha]] \curlywedge g\,[x/\alpha] \\
&= \ \sup_{g \in S}\ \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge g\,[x/\alpha] \\
&= \ \sup_{g \in S}\ \mathsf{sp}\,[\![x \coloneqq e]\!]\,(g) \\
&= \ \sup\ \mathsf{sp}\,[\![x \coloneqq e]\!]\,(S)\ .
\end{aligned}
$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, Theorem A.3.2 holds.

We proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$
\mathsf{sp}\,[\![C_1 \fatsemi C_2]\!]\,(\sup S) \ = \ \mathsf{sp}\,[\![C_2]\!]\,(\mathsf{sp}\,[\![C_1]\!]\,(\sup S))
$$

$$
\begin{aligned}
&= \mathsf{sp}\,[\![C_2]\!]\,(\sup\ \mathsf{sp}\,[\![C_1]\!]\,(S)) && \text{(by I.H. on } C_1\text{)}\\
&= \sup\ \mathsf{sp}\,[\![C_2]\!]\,(\mathsf{sp}\,[\![C_1]\!]\,(S)) && \text{(by I.H. on } C_2\text{)}\\
&= \sup\ \mathsf{sp}\,[\![C_1\,⨾\ C_2]\!]\,(S)\ .
\end{aligned}
$$

**The nondeterministic choice $\{\, C_1 \,\} \mathrel{\Box} \{\, C_2 \,\}$:**

We have

$$
\begin{aligned}
\mathsf{sp}\,[\![\{\, C_1 \,\} \mathrel{\Box} \{\, C_2 \,\}]\!]\,(\sup S) &= \mathsf{sp}\,[\![C_1]\!]\,(f) \ \curlyvee\ \mathsf{sp}\,[\![C_2]\!]\,(f)\\
&= \sup_{g\in S}\ \mathsf{sp}\,[\![C_1]\!]\,(g) \ \curlyvee\ \sup_{g\in S}\ \mathsf{sp}\,[\![C_2]\!]\,(g)\\
&\qquad\qquad\qquad\qquad \text{(by I.H. on } C_1, C_2\text{)}\\
&= \sup_{g\in S}\ \mathsf{sp}\,[\![C_1]\!]\,(g) \ \curlyvee\ \mathsf{sp}\,[\![C_2]\!]\,(g)\\
&= \sup_{g\in S}\ \mathsf{sp}\,[\![\{\, C_1 \,\} \mathrel{\Box} \{\, C_2 \,\}]\!]\,(g)\ .
\end{aligned}
$$

**The conditional branching $\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}$:**

We have

$$
\begin{aligned}
&\mathsf{sp}\,[\![\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}]\!]\,(\sup S)\\
&\quad= \mathsf{sp}\,[\![C_1]\!]\,([\varphi] \curlywedge \sup S) \ \curlyvee\ \mathsf{sp}\,[\![C_2]\!]\,([\neg\varphi] \curlywedge \sup S)\\
&\quad= \mathsf{sp}\,[\![C_1]\!]\,(\sup\,[\varphi] \curlywedge S) \ \curlyvee\ \mathsf{sp}\,[\![C_2]\!]\,(\sup\,[\neg\varphi] \curlywedge S)\\
&\quad= \sup\ \mathsf{sp}\,[\![C_1]\!]\,([\varphi] \curlywedge S) \ \curlyvee\ \sup\ \mathsf{sp}\,[\![C_2]\!]\,([\neg\varphi] \curlywedge S)\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(by I.H. on } C_1 \text{ and } C_2\text{)}\\
&\quad= \sup\ \big(\mathsf{sp}\,[\![C_1]\!]\,([\varphi] \curlywedge S) \ \curlyvee\ \mathsf{sp}\,[\![C_2]\!]\,([\neg\varphi] \curlywedge S)\big)\\
&\quad= \sup\ \mathsf{sp}\,[\![\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}]\!]\,(S)\ .
\end{aligned}
$$

**The loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$:**

Let

$$\Psi_f(X) \;=\; f \;\curlyvee\; \mathsf{sp}\,\llbracket C \rrbracket\,([\varphi] \curlywedge X)\;,$$

be the $\mathsf{sp}$-characteristic function of the loop $\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}$ with respect to any preanticipation $f \in \mathbb{A}$. Observe that $\Psi_f(X)$ is continuous by inductive hypothesis on $C$ and by composition of continuous functions. We now prove by induction on $n$ that

$$\Psi^n_{\sup S}(-\infty) \;=\; \sup_{g \in S} \Psi^n_g(-\infty)\;. \tag{A.6}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}
\Psi^0_{\sup S}(-\infty) \;&=\; -\infty \\
&=\; \sup_{g \in S} -\infty \\
&=\; \sup_{g \in S} \Psi^0_g(-\infty)\;.
\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$

$$\Psi^n_{\sup S}(-\infty) \;=\; \sup_{g \in S} \Psi^n_g(-\infty)\;.$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}
&\Psi^{n+1}_{\sup S}(-\infty) \\
&=\; \sup S \;\curlyvee\; \mathsf{sp}\,\llbracket C \rrbracket\,\big([\varphi] \curlywedge \Psi^n_{\sup S}(-\infty)\big) \\
&=\; \sup S \;\curlyvee\; \mathsf{sp}\,\llbracket C \rrbracket\left([\varphi] \curlywedge \sup_{g \in S} \Psi^n_g(-\infty)\right) && \text{(by I.H. on } n\text{)} \\
&=\; \sup S \;\curlyvee\; \mathsf{sp}\,\llbracket C \rrbracket\left(\sup_{g \in S}\, [\varphi] \curlywedge \Psi^n_g(-\infty)\right) \\
&=\; \sup S \;\curlyvee\; \sup_{g \in S} \mathsf{sp}\,\llbracket C \rrbracket\,\big([\varphi] \curlywedge \Psi^n_g(-\infty)\big) && \text{(by I.H. on } C\text{)}
\end{aligned}$$

$$
\begin{aligned}
&= \sup_{g\in S} g \;\curlyvee\; \sup_{g\in S} \mathsf{sp}\,[\![C]\!]\,\big([\varphi] \curlywedge \Psi_g^n(-\infty)\big)\\
&= \sup_{g\in S} \big(g \;\curlyvee\; \mathsf{sp}\,[\![C]\!]\,\big([\varphi] \curlywedge \Psi_g^n(-\infty)\big)\big)\\
&= \sup_{g\in S} \Psi_g^{n+1}(-\infty)\ .
\end{aligned}
$$

This concludes the induction on $n$. Now we have:

$$
\begin{aligned}
\mathsf{sp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(\sup S) \quad &= \quad [\neg\varphi] \curlywedge \big(\mathsf{lfp}\ X\colon\ \sup S \;\curlyvee\; \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge X)\big)\\
&= \quad [\neg\varphi] \curlywedge \sup_{n\in\mathbb{N}}\ \Psi^n_{\sup S}(-\infty)\\
&\qquad\qquad\qquad \text{(by Kleene's fixpoint theorem)}\\
&= \quad [\neg\varphi] \curlywedge \sup_{n\in\mathbb{N}}\ \sup_{g\in S} \Psi_g^n(-\infty)\\
&\qquad\qquad\qquad \text{(by Equation A.6)}\\
&= \quad [\neg\varphi] \curlywedge \sup_{g\in S}\ \sup_{n\in\mathbb{N}} \Psi_g^n(-\infty)\\
&= \quad [\neg\varphi] \curlywedge \sup_{g\in S}\ \sup_{n\in\mathbb{N}} \Psi_g^n(-\infty)\\
&= \quad \sup_{g\in S}([\neg\varphi] \curlywedge \sup_{n\in\mathbb{N}} \Psi_g^n(-\infty))\\
&= \quad \sup_{g\in S} \mathsf{sp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g)\\
&\qquad\qquad\qquad \text{(by Kleene's fixpoint theorem)}\\
&= \quad \sup \mathsf{sp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(S)\,,
\end{aligned}
$$

and this concludes the proof. □

**Theorem A.3.3** (Quantitative universal conjunctiveness of wlp). *For any set of quantities* $\subseteq \mathbb{A}$,

$$
\mathsf{wlp}[\![C]\!]\,(\inf S) \quad = \quad \inf\ \mathsf{wlp}[\![C]\!]\,(S)\ .
$$

*Proof.* We prove Theorem A.3.3 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The effectless program skip:**

We have

$$\begin{aligned}
\mathsf{wlp}[\![\texttt{skip}]\!]\,(\inf S) &= \inf S \\
&= \inf_{g\in S} g \\
&= \inf_{g\in S}\ \mathsf{wlp}[\![\texttt{skip}]\!]\,(g) \\
&= \inf\ \mathsf{wlp}[\![\texttt{skip}]\!]\,(S)\ .
\end{aligned}$$

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{wlp}[\![x \coloneqq e]\!]\,(\inf S) &= (\inf S)\,[x/e] \\
&= \left(\lambda\sigma\colon\ \inf_{g\in S} g(\sigma)\right)[x/e] \\
&= \left(\lambda\sigma\colon\ \inf_{g\in S} g\,[x/e]\,(\sigma)\right) \\
&= \inf_{g\in S} g\,[x/e] \\
&= \inf_{g\in S}\ \mathsf{wlp}[\![x \coloneqq e]\!]\,(g) \\
&= \inf\ \mathsf{wlp}[\![x \coloneqq e]\!]\,(S)\ .
\end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, Theorem A.3.3 holds.

We proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$\mathsf{wlp}[\![C_1 \fatsemi C_2]\!]\,(\inf S) = \mathsf{wlp}[\![C_1]\!]\,(\mathsf{wlp}[\![C_2]\!]\,(\inf S))$$

$$
\begin{aligned}
&= \mathsf{wlp}[\![C_1]\!]\,(\inf\ \mathsf{wlp}[\![C_2]\!]\,(S)) && \text{(by I.H. on } C_2\text{)}\\
&= \inf\ \mathsf{wlp}[\![C_1]\!]\,(\mathsf{wlp}[\![C_2]\!]\,(S)) && \text{(by I.H. on } C_1\text{)}\\
&= \inf\ \mathsf{wlp}[\![C_1 \fatsemi C_2]\!]\,(S)\ .
\end{aligned}
$$

**The nondeterministic choice $\{\, C_1 \,\} \mathrel{\square} \{\, C_2 \,\}$:**

We have

$$
\begin{aligned}
\mathsf{wlp}[\![\{\, C_1 \,\} \mathrel{\square} \{\, C_2 \,\}]\!]\,(\inf S) &= \mathsf{wlp}[\![C_1]\!]\,(f) \curlywedge \mathsf{wlp}[\![C_2]\!]\,(f)\\
&= \inf_{g \in S}\ \mathsf{wlp}[\![C_1]\!]\,(g) \curlywedge \inf_{g \in S}\ \mathsf{wlp}[\![C_2]\!]\,(g)\\
&\qquad\qquad \text{(by I.H. on } C_1, C_2\text{)}\\
&= \inf_{g \in S}\ \mathsf{wlp}[\![C_1]\!]\,(g) \curlywedge \mathsf{wlp}[\![C_2]\!]\,(g)\\
&= \inf_{g \in S}\ \mathsf{wlp}[\![\{\, C_1 \,\} \mathrel{\square} \{\, C_2 \,\}]\!]\,(g)\ .
\end{aligned}
$$

**The conditional branching $\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}$:**

We have

$$
\begin{aligned}
&\mathsf{wlp}[\![\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}]\!]\,(\inf S)\\
&= [\varphi] \curlywedge \mathsf{wlp}[\![C_1]\!]\,(\inf S) \ \curlyvee\ [\neg\varphi] \curlywedge \mathsf{wlp}[\![C_2]\!]\,(\inf S)\\
&= [\varphi] \curlywedge \inf\ \mathsf{wlp}[\![C_1]\!]\,(S) \ \curlyvee\ [\neg\varphi] \curlywedge \inf\ \mathsf{wlp}[\![C_2]\!]\,(S)\\
&\qquad\qquad \text{(by I.H. on } C_1 \text{ and } C_2\text{)}\\
&= \inf\ \big([\varphi] \curlywedge \mathsf{wlp}[\![C_1]\!]\,(S)\big) \ \curlyvee\ \inf\ \big([\neg\varphi] \curlywedge \mathsf{wlp}[\![C_2]\!]\,(S)\big)\\
&= \lambda\sigma\colon \begin{cases} \inf\ \big(\mathsf{wlp}[\![C_1]\!]\,(S)\big) & \text{if } \sigma \models \varphi\\ \inf\ \big(\mathsf{wlp}[\![C_2]\!]\,(S)\big) & \text{otherwise} \end{cases}\\
&= \inf\ \big([\varphi] \curlywedge \mathsf{wlp}[\![C_1]\!]\,(S) \ \curlyvee\ [\neg\varphi] \curlywedge \mathsf{wlp}[\![C_2]\!]\,(S)\big)\\
&= \inf\ \mathsf{wlp}[\![\texttt{if}\ (\,\varphi\,)\ \{\, C_1 \,\}\ \texttt{else}\ \{\, C_2 \,\}]\!]\,(S)\ .
\end{aligned}
$$

**The loop `while` $(\varphi)\{C\}$:**

Let

$$\Phi_f(X) \;=\; [\neg\varphi] \curlywedge f \;\;\curlyvee\;\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(X) \;,$$

be the $\mathsf{wlp}$-characteristic function of the loop `while` $(\varphi)\{C\}$ with respect to any postanticipation $f \in \mathbb{A}$. Observe that $\Phi_f(X)$ is continuous by inductive hypothesis on $C$ and by composition of continuous functions. We now prove by induction on $n$ that

$$\Phi^n_{\inf S}(+\infty) \;=\; \inf_{g \in S} \Phi^n_g(+\infty) \;. \tag{A.7}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}\Phi^0_{\inf S}(+\infty) &= +\infty \\ &= \inf_{g\in S} +\infty \\ &= \inf_{g\in S} \Phi^0_g(+\infty) \;.\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$

$$\Phi^n_{\inf S}(+\infty) \;=\; \inf_{g \in S} \Phi^n_g(+\infty) \;.$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}&\Phi^{n+1}_{\inf S}(+\infty) \\ &= [\neg\varphi] \curlywedge \inf S \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(\Phi^n_{\inf S}(+\infty)) \\ &= [\neg\varphi] \curlywedge \inf S \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\left(\inf_{g\in S} \Phi^n_g(+\infty)\right) && \text{(by I.H. on } n) \\ &= [\neg\varphi] \curlywedge \inf S \;\curlyvee\; [\varphi] \curlywedge \inf_{g\in S}\; \mathsf{wlp}[\![C]\!]\left(\Phi^n_g(+\infty)\right) && \text{(by I.H. on } C) \\ &= \inf_{g\in S}\; ([\neg\varphi] \curlywedge g) \;\curlyvee\; \inf_{g\in S}\; \left([\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\left(\Phi^n_g(+\infty)\right)\right)\end{aligned}$$

$$
\begin{aligned}
&= \lambda\sigma\colon \begin{cases} \inf_{g\in S} \; \big(\mathsf{wlp}[\![C]\!]\,(\Phi_g^n(+\infty))\big) & \text{if } \sigma \models \varphi \\ \inf_{g\in S}\; (g) & \text{otherwise} \end{cases} \\
&= \inf_{g\in S} \big([\neg\varphi] \curlywedge g \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(\Phi_g^n(+\infty))\big) \\
&= \inf_{g\in S} \Phi_g^{n+1}(+\infty)\ .
\end{aligned}
$$

This concludes the induction on $n$. Now we have:

$$
\begin{aligned}
\mathsf{wlp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(\inf S) &= \mathsf{gfp}\; X\colon\; [\neg\varphi] \curlywedge \inf S \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(X) \\
&= \inf_{n\in\mathbb{N}}\; \Phi^n_{\inf S}(+\infty) \\
&\qquad\qquad \text{(by Kleene's fixpoint theorem)} \\
&= \inf_{n\in\mathbb{N}}\; \inf_{g\in S}\; \Phi_g^n(+\infty) \qquad \text{(by Equation A.7)} \\
&= \inf_{g\in S}\; \inf_{n\in\mathbb{N}}\; \Phi_g^n(+\infty) \\
&= \inf_{g\in S} \mathsf{wlp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g) \\
&\qquad\qquad \text{(by Kleene's fixpoint theorem)} \\
&= \inf \mathsf{wlp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(S)\,,
\end{aligned}
$$

and this concludes the proof. □

**Theorem A.3.4** (Quantitative universal conjunctiveness of $\mathsf{slp}$)**.** *For any set of quantities* $\subseteq \mathbb{A}$*,*

$$
\mathsf{slp}[\![C]\!]\,(\inf S) \quad = \quad \inf\; \mathsf{slp}[\![C]\!]\,(S)\ .
$$

*Proof.* We prove Theorem A.3.4 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The effectless program skip:**

We have

$$\begin{aligned}
\mathsf{slp}[\![\texttt{skip}]\!]\,(\inf S) &= \inf S \\
&= \inf_{g \in S} g \\
&= \inf_{g \in S}\ \mathsf{slp}[\![\texttt{skip}]\!]\,(g) \\
&= \inf\ \mathsf{slp}[\![\texttt{skip}]\!]\,(S)\ .
\end{aligned}$$

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{slp}[\![x \coloneqq e]\!]\,(\inf S) &= \raisebox{.6\depth}{\rotatebox{-30}{\textsf{\reflectbox{J}}}}\,\alpha\colon\ \ [x \neq e\,[x/\alpha]] \curlyvee (\inf S)\,[x/\alpha] \\
&= \raisebox{.6\depth}{\rotatebox{-30}{\textsf{\reflectbox{J}}}}\,\alpha\colon\ \ [x \neq e\,[x/\alpha]] \curlyvee \left(\lambda\sigma\colon\ \inf_{g \in S} g(\sigma)\right)[x/\alpha] \\
&= \raisebox{.6\depth}{\rotatebox{-30}{\textsf{\reflectbox{J}}}}\,\alpha\colon\ \ [x \neq e\,[x/\alpha]] \curlyvee \left(\lambda\sigma\colon\ \inf_{g \in S} g\,[x/\alpha]\,(\sigma)\right) \\
&= \raisebox{.6\depth}{\rotatebox{-30}{\textsf{\reflectbox{J}}}}\,\alpha\colon\ \ [x \neq e\,[x/\alpha]] \curlyvee \inf_{g \in S} g\,[x/\alpha] \\
&= \raisebox{.6\depth}{\rotatebox{-30}{\textsf{\reflectbox{J}}}}\,\alpha\colon\ \ \inf_{g \in S}\,[x \neq e\,[x/\alpha]] \curlyvee g\,[x/\alpha] \\
&= \inf_{g \in S}\ \raisebox{.6\depth}{\rotatebox{-30}{\textsf{\reflectbox{J}}}}\,\alpha\colon\ \ [x \neq e\,[x/\alpha]] \curlyvee g\,[x/\alpha] \\
&= \inf_{g \in S}\ \mathsf{slp}[\![x \coloneqq e]\!]\,(g) \\
&= \inf\ \mathsf{slp}[\![x \coloneqq e]\!]\,(S)\ .
\end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, Theorem A.3.4 holds.

We proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$
\begin{aligned}
\mathsf{slp}[\![C_1 \fatsemi C_2]\!]\,(\inf S) &= \mathsf{slp}[\![C_2]\!]\,(\mathsf{slp}[\![C_1]\!]\,(\inf S)) \\
&= \mathsf{slp}[\![C_2]\!]\,(\inf\ \mathsf{slp}[\![C_1]\!]\,(S)) && \text{(by I.H. on } C_1) \\
&= \inf\ \mathsf{slp}[\![C_2]\!]\,(\mathsf{slp}[\![C_1]\!]\,(S)) && \text{(by I.H. on } C_2) \\
&= \inf\ \mathsf{slp}[\![C_1 \fatsemi C_2]\!]\,(S)\ .
\end{aligned}
$$

**The conditional branching $\texttt{if}\ (\varphi)\ \{C_1\}\ \texttt{else}\ \{C_2\}$:**

We have

$$
\begin{aligned}
&\mathsf{slp}[\![\texttt{if}\ (\varphi)\ \{C_1\}\ \texttt{else}\ \{C_2\}]\!]\,(\inf S) \\
&= \mathsf{slp}[\![C_1]\!]\,([\neg\varphi] \curlyvee \inf S) \ \curlywedge\ \mathsf{slp}[\![C_2]\!]\,([\varphi] \curlyvee \inf S) \\
&= \mathsf{slp}[\![C_1]\!]\,(\inf\,[\neg\varphi] \curlyvee S) \ \curlywedge\ \mathsf{slp}[\![C_2]\!]\,(\inf\,[\varphi] \curlyvee S) \\
&= \inf\ \mathsf{slp}[\![C_1]\!]\,([\neg\varphi] \curlyvee S) \ \curlywedge\ \inf\ \mathsf{slp}[\![C_2]\!]\,([\varphi] \curlyvee S) \quad \text{(by I.H. on } C_1 \text{ and } C_2) \\
&= \inf\ \big(\mathsf{slp}[\![C_1]\!]\,([\neg\varphi] \curlyvee S) \ \curlywedge\ \mathsf{slp}[\![C_2]\!]\,([\varphi]\,[\neg\varphi]\,S)\big) \\
&= \inf\ \mathsf{slp}[\![\texttt{if}\ (\varphi)\ \{C_1\}\ \texttt{else}\ \{C_2\}]\!]\,(S)\ .
\end{aligned}
$$

**The nondeterministic choice $\{C_1\}\ \square\ \{C_2\}$:**

We have

$$
\begin{aligned}
\mathsf{slp}[\![\{C_1\}\ \square\ \{C_2\}]\!]\,(\inf S) &= \mathsf{slp}[\![C_1]\!]\,(f) \curlywedge \mathsf{slp}[\![C_2]\!]\,(f) \\
&= \inf_{g \in S}\ \mathsf{slp}[\![C_1]\!]\,(g) \curlywedge \inf_{g \in S}\ \mathsf{slp}[\![C_2]\!]\,(g) \\
&\qquad\qquad \text{(by I.H. on } C_1, C_2) \\
&= \inf_{g \in S}\ \mathsf{slp}[\![C_1]\!]\,(g) \curlywedge \mathsf{slp}[\![C_2]\!]\,(g) \\
&= \inf_{g \in S}\ \mathsf{slp}[\![\{C_1\}\ \square\ \{C_2\}]\!]\,(g)\ .
\end{aligned}
$$

**The loop `while` $(\varphi)\{C\}$:**

Let

$$\Psi_f(X) \;=\; f \;\curlywedge\; \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee X)\;,$$

be the $\mathsf{slp}$-characteristic function of the loop `while` $(\varphi)\{C\}$ with respect to any preanticipation $f \in \mathbb{A}$. Observe that $\Psi_f(X)$ is continuous by inductive hypothesis on $C$ and by composition of continuous functions. We now prove by induction on $n$ that

$$\Psi^n_{\inf S}(+\infty) \;=\; \inf_{g\in S} \Psi^n_g(+\infty)\;. \tag{A.8}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}\Psi^0_{\inf S}(+\infty) &= +\infty\\ &= \inf_{g\in S} +\infty\\ &= \inf_{g\in S} \Psi^0_g(+\infty)\;.\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$

$$\Psi^n_{\inf S}(+\infty) \;=\; \inf_{g\in S} \Psi^n_g(+\infty)\;.$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}&\Psi^{n+1}_{\inf S}(+\infty)\\ &= \inf S \;\curlywedge\; \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee \Psi^n_{\inf S}(+\infty))\\ &= \inf S \;\curlywedge\; \mathsf{slp}[\![C]\!]\left([\neg\varphi] \curlyvee \inf_{g\in S} \Psi^n_g(+\infty)\right) && \text{(by I.H. on } n)\\ &= \inf S \;\curlywedge\; \mathsf{slp}[\![C]\!]\left(\inf_{g\in S} [\neg\varphi] \curlyvee \Psi^n_g(+\infty)\right)\\ &= \inf S \;\curlywedge\; \inf_{g\in S} \mathsf{slp}[\![C]\!]\,\big([\neg\varphi] \curlyvee \Psi^n_g(+\infty)\big) && \text{(by I.H. on } C)\end{aligned}$$

$$
\begin{aligned}
&= \inf_{g\in S} g \;\curlywedge\; \inf_{g\in S} \mathsf{slp}[\![C]\!]\left([\neg\varphi] \curlyvee \Psi_g^n(+\infty)\right)\\
&= \inf_{g\in S} \left(g \;\curlywedge\; \mathsf{slp}[\![C]\!]\left([\neg\varphi] \curlyvee \Psi_g^n(+\infty)\right)\right)\\
&= \inf_{g\in S} \Psi_g^{n+1}(+\infty) \;.
\end{aligned}
$$

This concludes the induction on $n$. Now we have:

$$
\begin{aligned}
\mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(\inf S) &= [\varphi] \curlyvee \left(\mathsf{gfp}\; X\colon\; \inf S \;\curlywedge\; \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee X)\right)\\
&= [\varphi] \curlyvee \inf_{n\in\mathbb{N}}\; \Psi_{\inf S}^n(+\infty)\\
&\qquad\qquad\text{(by Kleene's fixpoint theorem)}\\
&= [\varphi] \curlyvee \inf_{n\in\mathbb{N}}\; \inf_{g\in S} \Psi_g^n(+\infty) \qquad \text{(by Equation A.8)}\\
&= [\varphi] \curlyvee \inf_{g\in S}\; \inf_{n\in\mathbb{N}} \Psi_g^n(+\infty)\\
&= [\varphi] \curlyvee \inf_{g\in S}\; \inf_{n\in\mathbb{N}} \Psi_g^n(+\infty)\\
&= \inf_{g\in S}([\varphi] \curlyvee \inf_{n\in\mathbb{N}} \Psi_g^n(+\infty))\\
&= \inf_{g\in S} \mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g)\\
&\qquad\qquad\text{(by Kleene's fixpoint theorem)}\\
&= \inf \mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(S)\,,
\end{aligned}
$$

and this concludes the proof. □

**Corollary A.3.4.1** (Strictness of wp)**.** *For all programs $C$,* $\mathsf{wp}[\![C]\!]$ *is strict, i.e.*

$$
\mathsf{wp}\,[\![C]\!]\,(-\infty) \;=\; -\infty \;.
$$

*Proof.*

$$
\mathsf{wp}\,[\![C]\!]\,(-\infty) \;=\; \lambda\sigma\colon \sup_{\tau\in[\![C]\!](\sigma)} -\infty(\tau) \qquad \text{(by Theorem 3.3.1)}
$$

$$= -\infty \;.$$

$\square$

**Corollary A.3.4.2** (Strictness of $\mathsf{sp}$)**.** *For all programs $C$, $\mathsf{sp}[\![C]\!]$ is strict, i.e.*

$$\mathsf{sp}\,[\![C]\!]\,(-\infty) \;=\; -\infty \;.$$

*Proof.*

$$\begin{aligned} \mathsf{sp}\,[\![C]\!]\,(-\infty) &= \lambda\tau\colon \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} -\infty(\sigma) && \text{(by Theorem 3.4.1)} \\ &= -\infty \;. \end{aligned}$$

$\square$

**Corollary A.3.4.3** (Co-strictness of $\mathsf{wlp}$)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ is co-strict, i.e.*

$$\mathsf{wlp}[\![C]\!]\,(+\infty) \;=\; +\infty \;.$$

*Proof.*

$$\begin{aligned} \mathsf{wlp}[\![C]\!]\,(+\infty) &= \lambda\sigma\colon \inf_{\tau\in[\![C]\!](\sigma)} +\infty(\tau) && \text{(by Theorem 3.3.2)} \\ &= +\infty \;. \end{aligned}$$

$\square$

**Corollary A.3.4.4** (Co-strictness of $\mathsf{slp}$)**.** *For all programs $C$, $\mathsf{slp}[\![C]\!]$ is co-strict, i.e.*

$$\mathsf{slp}[\![C]\!]\,(+\infty) \;=\; +\infty \;.$$

*Proof.*

$$\begin{aligned}\mathsf{slp}[\![C]\!]\,(+\infty) &= \lambda\tau\colon \inf_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} +\infty(\sigma) && \text{(by Theorem 3.4.2)}\\ &= +\infty\ .\end{aligned}$$

□

**Corollary A.3.4.5** (Monotonicity of Quantitative Transformers)**.** *For all programs $C$, $f, g \in \mathbb{A}$, we have*

$$f \ \preceq\ g \quad \text{implies} \quad \mathsf{ttt}\,[\![C]\!]\,(f) \ \preceq\ \mathsf{ttt}\,[\![C]\!]\,(g)\ , \quad \text{for } \mathsf{ttt} \in \{\mathsf{wp}, \mathsf{wlp}, \mathsf{sp}, \mathsf{slp}\}$$

*Proof.* Direct consequence of universal conjunctiveness and universal disjunctiveness. □

## A.3.2 Proof of Linearity, Theorem 3.5.2

**Theorem 3.5.2** (Linearity)**.** *For all programs $C$,* $\mathsf{wp}[\![C]\!]$ *and* $\mathsf{sp}[\![C]\!]$ *are sublinear, and* $\mathsf{wlp}[\![C]\!]$ *and* $\mathsf{slp}[\![C]\!]$ *are superlinear, i.e. for all $f, g \in \mathbb{A}$ and* non-negative *constants $r \in \mathbb{R}_{\geq 0}$,*

$$\begin{aligned}\mathsf{wp}\,[\![C]\!]\,(r\cdot f+g) &\ \preceq\ r\cdot \mathsf{wp}\,[\![C]\!]\,(f) + \mathsf{wp}\,[\![C]\!]\,(g)\ ,\\ \mathsf{sp}\,[\![C]\!]\,(r\cdot f+g) &\ \preceq\ r\cdot \mathsf{sp}\,[\![C]\!]\,(f) + \mathsf{sp}\,[\![C]\!]\,(g)\ ,\\ r\cdot\mathsf{wlp}[\![C]\!]\,(f) + \mathsf{wlp}[\![C]\!]\,(g) &\ \preceq\ \mathsf{wlp}[\![C]\!]\,(r\cdot f+g)\ , \quad \text{and}\\ r\cdot\mathsf{slp}[\![C]\!]\,(f) + \mathsf{slp}[\![C]\!]\,(g) &\ \preceq\ \mathsf{slp}[\![C]\!]\,(r\cdot f+g)\ .\end{aligned}$$

*Proof.* For $\mathsf{wp}$ we have:

$$\begin{aligned}&\mathsf{wp}\,[\![C]\!]\,(r\cdot f+g)\\ &= \lambda\sigma\colon \sup_{\tau\in[\![C]\!]\sigma} (r\cdot f+g)(\tau) && \text{(by Theorem 3.3.1)}\\ &= \lambda\sigma\colon \sup_{\tau\in[\![C]\!]\sigma} \big((r\cdot f)(\tau)+g(\tau)\big)\end{aligned}$$

$$
\begin{aligned}
&\preceq \lambda\sigma\colon \sup_{\tau\in[\![C]\!]\sigma} (r\cdot f)(\tau) + \sup_{\tau\in[\![C]\!]\sigma} g(\tau) \\
&= \lambda\sigma\colon r\cdot \sup_{\tau\in[\![C]\!]\sigma} f(\tau) + \sup_{\tau\in[\![C]\!]\sigma} g(\tau) \\
&\qquad\qquad (\sup(r\cdot A) = r\cdot\sup A \text{ for } A\subseteq\mathbb{R}, r\in\mathbb{R}_{\geq 0}) \\
&= r\cdot\lambda\sigma\colon \sup_{\tau\in[\![C]\!]\sigma} f(\tau) + \lambda\sigma\colon \sup_{\tau\in[\![C]\!]\sigma} g(\tau) \\
&= r\cdot \mathsf{wp}\,[\![C]\!]\,(f) + \mathsf{wp}\,[\![C]\!]\,(g)\ . \qquad \text{(by Theorem 3.3.1)}
\end{aligned}
$$

For $\mathsf{wp}$ we have:

$$
\begin{aligned}
&\mathsf{sp}\,[\![C]\!]\,(r\cdot f+g) \\
&= \lambda\tau\colon \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} (r\cdot f+g)(\sigma) \qquad \text{(by Theorem 3.4.1)} \\
&= \lambda\tau\colon \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} \big((r\cdot f)(\sigma) + g(\sigma)\big) \\
&\preceq \lambda\tau\colon \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} (r\cdot f)(\sigma) + \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} g(\sigma) \\
&= \lambda\tau\colon r\cdot \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} f(\sigma) + \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} g(\sigma) \\
&\qquad\qquad (\sup(r\cdot A) = r\cdot\sup A \text{ for } A\subseteq\mathbb{R}, r\in\mathbb{R}_{\geq 0}) \\
&= r\cdot\lambda\tau\colon \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} f(\sigma) + \lambda\tau\colon \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} g(\sigma) \\
&= r\cdot \mathsf{sp}\,[\![C]\!]\,(f) + \mathsf{sp}\,[\![C]\!]\,(g)\ . \qquad \text{(by Theorem 3.4.1)}
\end{aligned}
$$

For $\mathsf{wlp}$ we have:

$$
\begin{aligned}
&r\cdot \mathsf{wlp}[\![C]\!]\,(f) + \mathsf{wlp}[\![C]\!]\,(g) \\
&= r\cdot\lambda\sigma\colon \inf_{\tau\in[\![C]\!]\sigma} f(\tau) + \lambda\sigma\colon \inf_{\tau\in[\![C]\!]\sigma} g(\tau) \qquad \text{(by Theorem 3.3.2)} \\
&= \lambda\sigma\colon r\cdot \inf_{\tau\in[\![C]\!]\sigma} f(\tau) + \inf_{\tau\in[\![C]\!]\sigma} g(\tau) \\
&= \lambda\sigma\colon \inf_{\tau\in[\![C]\!]\sigma} (r\cdot f)(\tau) + \inf_{\tau\in[\![C]\!]\sigma} g(\tau) \\
&\qquad\qquad (\inf(r\cdot A) = r\cdot\inf A \text{ for } A\subseteq\mathbb{R}, r\in\mathbb{R}_{\geq 0}) \\
&\preceq \lambda\sigma\colon \inf_{\tau\in[\![C]\!]\sigma} \big((r\cdot f)(\tau) + g(\tau)\big)
\end{aligned}
$$

$$
\begin{aligned}
&= \lambda\sigma\colon \inf_{\tau \in [\![C]\!]\sigma} (r \cdot f + g)(\tau) \\
&= \mathsf{wlp}[\![C]\!]\,(r \cdot f + g) \;; && \text{(by Theorem 3.3.2)}
\end{aligned}
$$

For $\mathsf{slp}$ we have:

$$
\begin{aligned}
& r \cdot \mathsf{slp}[\![C]\!]\,(f) + \mathsf{slp}[\![C]\!]\,(g) \\
&= r \cdot \lambda\tau\colon \inf_{\sigma\in\Sigma, \tau \in [\![C]\!]\sigma} f(\sigma) + \lambda\tau\colon \inf_{\sigma\in\Sigma, \tau \in [\![C]\!]\sigma} g(\sigma) && \text{(by Theorem 3.4.2)} \\
&= \lambda\tau\colon r \cdot \inf_{\sigma\in\Sigma, \tau \in [\![C]\!]\sigma} f(\sigma) + \inf_{\sigma\in\Sigma, \tau \in [\![C]\!]\sigma} g(\sigma) \\
&= \lambda\tau\colon \inf_{\sigma\in\Sigma, \tau \in [\![C]\!]\sigma} (r \cdot f)(\sigma) + \inf_{\sigma\in\Sigma, \tau \in [\![C]\!]\sigma} g(\sigma) \\
& && (\inf(r \cdot A) = r \cdot \inf A \text{ for } A \subseteq \mathbb{R}, r \in \mathbb{R}_{\geq 0}) \\
&\preceq \lambda\tau\colon \inf_{\sigma\in\Sigma, \tau \in [\![C]\!]\sigma} \big((r \cdot f)(\sigma) + g(\sigma)\big) \\
&= \lambda\tau\colon \inf_{\sigma\in\Sigma, \tau \in [\![C]\!]\sigma} (r \cdot f + g)(\sigma) \\
&= \mathsf{slp}[\![C]\!]\,(r \cdot f + g) \;. && \text{(by Theorem 3.4.2)}
\end{aligned}
$$

□

### A.3.3 Proof of Embedding Classical into Quantitative Transformers, Theorem 3.5.3

**Theorem 3.5.3** (Embedding Classical into Quantitative Transformers). *For all* <u>*deterministic*</u> *programs $C$ and predicates $\psi$, we have*

$$
\mathsf{wp}\,[\![C]\!]\,([\psi]) \;=\; [\mathsf{wp}\,[\![C]\!]\,(\psi)] \qquad \text{and} \qquad \mathsf{wlp}[\![C]\!]\,([\psi]) \;=\; [\mathsf{wlp}\,[\![C]\!]\,(\psi)] \;,
$$

*and for* <u>*all*</u> *programs $C$ and predicates $\psi$, we have*

$$
\mathsf{sp}\,[\![C]\!]\,([\psi]) \;=\; [\mathsf{sp}\,[\![C]\!]\,(\psi)] \qquad \text{and} \qquad \mathsf{slp}[\![C]\!]\,([\psi]) \;=\; [\mathsf{slp}[\![C]\!]\,(\psi)] \;.
$$

*Proof.* For $\mathsf{wp}$ we have:

$$
\begin{aligned}
\mathsf{wp}\,[\![C]\!]\,([F]) &= \lambda\sigma\colon \begin{cases} [F]\,(\tau) & \text{if } [\![C]\!](\sigma) = \{\tau\} \\ -\infty & \text{otherwise} \end{cases} && \text{(by Corollary 3.5.4.1)} \\
&= \lambda\sigma\colon \begin{cases} +\infty & \text{if } [\![C]\!](\sigma) = \{\tau\} \wedge \tau \models F \\ -\infty & \text{otherwise} \end{cases} \\
&= [\mathsf{wp}\,[\![C]\!]\,(F)] \ .
\end{aligned}
$$

For $\mathsf{wlp}$ we have:

$$
\begin{aligned}
\mathsf{wlp}[\![C]\!]\,([F]) &= \lambda\sigma\colon \begin{cases} [F]\,(\tau) & \text{if } [\![C]\!](\sigma) = \{\tau\} \\ +\infty & \text{otherwise} \end{cases} && \text{(by Corollary 3.5.4.1)} \\
&= \lambda\sigma\colon \begin{cases} -\infty & \text{if } [\![C]\!](\sigma) = \{\tau\} \wedge \tau \not\models F \\ +\infty & \text{otherwise} \end{cases} \\
&= [\mathsf{wlp}\,[\![C]\!]\,(F)] \ .
\end{aligned}
$$

For $\mathsf{sp}$ we have:

$$
\begin{aligned}
\mathsf{sp}\,[\![C]\!]\,([G]) &= \lambda\tau\colon \sup_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} [G]\,(\sigma) && \text{(by Theorem 3.4.1)} \\
&= \lambda\tau\colon \begin{cases} +\infty & \text{if } \exists\sigma \in \Sigma, \tau \in [\![C]\!](\sigma) \wedge \sigma \models G \\ -\infty & \text{otherwise} \end{cases} \\
&= [\mathsf{sp}\,[\![C]\!]\,(G)] \ ,
\end{aligned}
$$

For $\mathsf{slp}$ we have:

$$
\begin{aligned}
\mathsf{slp}[\![C]\!]\,([G]) &= \lambda\tau\colon \inf_{\sigma\in\Sigma,\tau\in[\![C]\!]\sigma} [G]\,(\sigma) && \text{(by Theorem 3.4.2)} \\
&= \lambda\tau\colon \begin{cases} -\infty & \text{if } \exists\sigma \in \Sigma, \tau \in [\![C]\!](\sigma) \wedge \sigma \not\models G \\ +\infty & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
&= \lambda\tau\colon \begin{cases} +\infty & \text{if } \forall \sigma \in \Sigma, \tau \notin [\![C]\!](\sigma) \vee \sigma \models G \\ -\infty & \text{otherwise} \end{cases} \\
&= \lambda\tau\colon \begin{cases} +\infty & \text{if } \forall \sigma \in \Sigma, \tau \in [\![C]\!](\sigma) \implies \sigma \models G \\ -\infty & \text{otherwise} \end{cases} \\
&= [\mathsf{slp}[\![C]\!]\,(\psi)] \enspace .
\end{aligned}
$$

□

### A.3.4 Proof of Liberal-Non-liberal Duality, Theorem 3.5.4

**Theorem 3.5.4** (Liberal–Non-liberal Duality). *For any program $C$ and quantity $f$, we have*

$$
\mathsf{wp}\,[\![C]\!]\,(f) \;=\; -\,\mathsf{wlp}[\![C]\!]\,(-f) \qquad \text{and} \qquad \mathsf{sp}\,[\![C]\!]\,(f) \;=\; -\,\mathsf{slp}[\![C]\!]\,(-f) \enspace .
$$

*Proof.* For $\mathsf{wp}$ and $\mathsf{wlp}$ we have:

$$
\begin{aligned}
\mathsf{wp}\,[\![C]\!]\,(f) &= \lambda\sigma\colon \sup_{\tau \in [\![C]\!]\sigma} f(\tau) && \text{(by Theorem 3.3.1)} \\
&= \lambda\sigma\colon -\inf_{\tau \in [\![C]\!]\sigma} -f(\tau) && (\sup A = -\inf(-A)) \\
&= -\,\mathsf{wlp}[\![C]\!]\,(-f) \enspace .
\end{aligned}
$$

For $\mathsf{sp}$ and $\mathsf{slp}$ we have:

$$
\begin{aligned}
\mathsf{sp}\,[\![C]\!]\,(g) &= \lambda\tau\colon \sup_{\sigma \in \Sigma, \tau \in [\![C]\!]\sigma} g(\sigma) && \text{(by Theorem 3.4.1)} \\
&= \lambda\tau\colon -\inf_{\sigma \in \Sigma, \tau \in [\![C]\!]\sigma} -g(\sigma) && (\sup A = -\inf(-A)) \\
&= -\,\mathsf{slp}[\![C]\!]\,(-g) \enspace .
\end{aligned}
$$

□

## A.4 Proofs of Section 3.6

### A.4.1 Proof of Galois Connection between $\mathsf{wlp}$ and $\mathsf{sp}$, Theorem 3.6.1

**Theorem 3.6.1** (Galois Connection between $\mathsf{wlp}$ and $\mathsf{sp}$)**.** *For all $C \in \mathsf{nGCL}$ and $g, f \in \mathbb{A}$:*

$$g \;\preceq\; \mathsf{wlp}[\![C]\!]\,(f) \qquad \text{iff} \qquad \mathsf{sp}\,[\![C]\!]\,(g) \;\preceq\; f\ .$$

*Proof.*

$$\begin{aligned}
g \;\preceq\; \mathsf{wlp}[\![C]\!]\,(f) &\iff \forall \sigma \in \Sigma\colon g(\sigma) \leq \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)\\
&\iff \forall \sigma \in \Sigma\colon g(\sigma) \leq \inf_{\tau \in [\![C]\!](\sigma)} f(\tau) \qquad \text{(by Theorem 3.3.2)}\\
&\iff \forall \sigma, \tau \in \Sigma\colon \tau \in [\![C]\!](\sigma)\colon g(\sigma) \leq f(\tau)\\
&\iff \forall \tau \in \Sigma\colon \sup_{\sigma \in \Sigma, \tau \in [\![C]\!](\sigma)} g(\sigma) \leq f(\tau)\\
&\iff \forall \tau \in \Sigma\colon \mathsf{sp}\,[\![C]\!]\,(g)\,(\tau) \leq f(\tau) \qquad \text{(by Theorem 3.4.1)}\\
&\iff \mathsf{sp}\,[\![C]\!]\,(g) \;\preceq\; f\ .
\end{aligned}$$

□

### A.4.2 Proof of Galois Connection between $\mathsf{wp}$ and $\mathsf{slp}$, Theorem 3.6.2

**Theorem 3.6.2** (Galois Connection between $\mathsf{wp}$ and $\mathsf{slp}$)**.** *For all $C \in \mathsf{nGCL}$ and $g, f \in \mathbb{A}$:*

$$\mathsf{wp}\,[\![C]\!]\,(f) \;\preceq\; g \qquad \text{iff} \qquad f \;\preceq\; \mathsf{slp}[\![C]\!]\,(g)$$

*Proof.*

$$\begin{aligned}
\mathsf{wp}\,[\![C]\!]\,(f) \;\preceq\; g &\iff \forall \sigma \in \Sigma\colon \mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \leq g(\sigma)\\
&\iff \forall \sigma \in \Sigma\colon \sup_{\tau\in[\![C]\!](\sigma)} f(\tau) \leq g(\sigma) \qquad \text{(by Theorem 3.3.1)}\\
&\iff \forall \sigma, \tau \in \Sigma\colon \tau \in [\![C]\!](\sigma)\colon f(\tau) \leq g(\sigma)\\
&\iff \forall \tau \in \Sigma\colon f(\tau) \leq \inf_{\sigma\in\Sigma, \tau\in[\![C]\!](\sigma)} g(\sigma)\\
&\iff \forall \tau \in \Sigma\colon f(\tau) \leq \mathsf{slp}[\![C]\!]\,(g)\,(\tau) \qquad \text{(by Theorem 3.4.1)}\\
&\iff f \;\preceq\; \mathsf{slp}[\![C]\!]\,(g)\ .
\end{aligned}$$

□

## A.5 Proofs of Section 3.7

### A.5.1 Proof of Induction Rules for Loops, Theorem 3.7.1

**Theorem 3.7.1** (Induction Rules for Loops)**.** *For any quantities $i, f, g \in \mathbb{A}$, boolean expression $\varphi$ and program $C$, the following proof rules for loops are valid:*

$$\frac{g \;\preceq\; i \;\preceq\; [\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(i)}{g \;\preceq\; \mathsf{wlp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f)}\ \text{while}{-}\mathsf{wlp}$$

$$\frac{g \;\curlyvee\; \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge i) \;\preceq\; i \quad\text{and}\quad [\neg\varphi] \curlywedge i \;\preceq\; f}{\mathsf{sp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g) \;\preceq\; f}\ \text{while}{-}\mathsf{sp}$$

$$\frac{[\neg\varphi] \curlywedge f \;\curlyvee\; [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(i) \;\preceq\; i \;\preceq\; g}{\mathsf{wp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \;\preceq\; g}\ \text{while}{-}\mathsf{wp}$$

$$\frac{i \;\preceq\; g \;\curlywedge\; \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee i) \quad\text{and}\quad f \;\preceq\; [\varphi] \curlyvee i}{f \;\preceq\; \mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g)}\ \text{while}{-}\mathsf{slp}$$

*Proof.* We prove each rule individually.

For while$-$wlp we have:

$$
\begin{aligned}
& i \preceq [\neg\varphi] \curlywedge f \curlyvee [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(i) && \text{(Premise of the rule)}\\
&\quad\Longrightarrow\ i \preceq \mathsf{gfp}\ X\colon\ [\neg\varphi] \curlywedge f \curlyvee [\varphi] \curlywedge \mathsf{wlp}[\![C]\!]\,(X) && \text{(by Park's Induction [94])}\\
&\quad\Longrightarrow\ i \preceq \mathsf{wlp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) && \text{(by Definition 3.3.4)}\\
&\quad\Longrightarrow\ g \preceq \mathsf{wlp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) && (g \preceq i \text{ and transitivity of } \preceq\,)
\end{aligned}
$$

For while$-$sp we have:

$$
\begin{aligned}
& g \curlyvee \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge i) \preceq i && \text{(Premise of the rule)}\\
&\quad\Longrightarrow\ \mathsf{lfp}\ X\colon g \curlyvee \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge X) \preceq i && \text{(by Park's Induction [94])}\\
&\quad\Longrightarrow\ [\neg\varphi] \curlywedge \mathsf{lfp}\ X\colon g \curlyvee \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge X) \preceq [\neg\varphi] \curlywedge i &&\\
& && \text{(by monotonicity of } \lambda X.\,[\neg\varphi] \curlywedge X)\\
&\quad\Longrightarrow\ \mathsf{sp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g) \preceq [\neg\varphi] \curlywedge i && \text{(by Definition 3.4.1)}\\
&\quad\Longrightarrow\ \mathsf{sp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g) \preceq f &&\\
& && ([\neg\varphi] \curlywedge i \preceq f \text{ and transitivity of } \preceq\,)
\end{aligned}
$$

For while$-$wp we have:

$$
\begin{aligned}
& [\neg\varphi] \curlywedge f \curlyvee [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(i) \preceq i && \text{(Premise of the rule)}\\
&\quad\Longrightarrow\ \mathsf{lfp}\ X\colon\ [\neg\varphi] \curlywedge f \curlyvee [\varphi] \curlywedge \mathsf{wp}\,[\![C]\!]\,(X) \preceq i && \text{(by Park's Induction [94])}\\
&\quad\Longrightarrow\ \mathsf{wp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \preceq i && \text{(by Definition 3.3.2)}\\
&\quad\Longrightarrow\ \mathsf{wp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \preceq g && (i \preceq g \text{ and transitivity of } \preceq\,)
\end{aligned}
$$

For while$-$slp we have:

$$
\begin{aligned}
& i \preceq g \curlywedge \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee i) && \text{(Premise of the rule)}\\
&\quad\Longrightarrow\ i \preceq \mathsf{gfp}\ X\colon g \curlywedge \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee X) && \text{(by Park's Induction [94])}\\
&\quad\Longrightarrow\ [\varphi] \curlyvee i \preceq [\varphi] \curlyvee \mathsf{gfp}\ X\colon g \curlywedge \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee X) &&\\
& && \text{(by monotonicity of } \lambda X.\,[\varphi] \curlyvee X)
\end{aligned}
$$

$$\Longrightarrow\ [\varphi] \curlyvee i \ \preceq\ \mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g) \qquad \text{(by Definition 3.4.2)}$$

$$\Longrightarrow\ f \ \preceq\ \mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(g) \quad (f \ \preceq\ [\varphi] \curlyvee i \text{ and transitivity of } \preceq\ )$$

□

## A.5.2 Proof of Proposition 3.7.2

**Proposition 3.7.2.** *The following proof rules for loops are valid:*

$$\frac{\mathsf{sp}\,[\![C]\!]\,(f) \quad \preceq \quad f}{\mathsf{sp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \quad = \quad [\neg\varphi] \curlywedge f}$$

$$\frac{f \quad \preceq \quad \mathsf{slp}[\![C]\!]\,(f)}{\mathsf{slp}[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \quad = \quad [\varphi] \curlyvee f}$$

*Proof.* We prove each statement individually. Let ${}^{\mathsf{sp}}\Psi_f$ and ${}^{\mathsf{slp}}\Psi_f$ be, respectively, the $\mathsf{sp}$–characteristic and $\mathsf{slp}$–characteristic functions of $\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}$. For $\mathsf{sp}$ we have:

$$\begin{aligned}
&[\varphi] \curlywedge f \ \preceq\ f \\
&\mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge f) \ \preceq\ \mathsf{sp}\,[\![C]\!]\,(f) && \text{(by Monotonicity of } \mathsf{sp}\text{)} \\
&\mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge f) \ \preceq\ f && \text{(by hypothesis and transitivity of } \preceq\ \text{)} \\
&f \curlyvee \mathsf{sp}\,[\![C]\!]\,([\varphi] \curlywedge f) \ \preceq\ f && \text{(by Monotonicity of } \lambda X\colon f \curlyvee X\text{)} \\
&{}^{\mathsf{sp}}\Psi_f^2(-\infty) \ \preceq\ {}^{\mathsf{sp}}\Psi_f(-\infty) && \text{(by Definition 3.4.1)}
\end{aligned}$$

Hence, the Kleene's iterates have converged immediately and the least fixpoint is exactly:

$$\mathsf{lfp}\ X\colon {}^{\mathsf{sp}}\Psi_f(X) \ = \ {}^{\mathsf{sp}}\Psi_f(-\infty) \ = \ f\ ,$$

and thus we conclude:

$$\begin{aligned}
\mathsf{sp}\,[\![\mathtt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) \ &= \ [\neg\varphi] \curlywedge \mathsf{lfp}\ X\colon {}^{\mathsf{sp}}\Psi_f(X) && \text{(by Definition 3.4.1)} \\
&= \ [\neg\varphi] \curlywedge f\ . && (\mathsf{lfp}\ X\colon {}^{\mathsf{sp}}\Psi_f(X) \ = \ f)
\end{aligned}$$

For $\mathsf{slp}$ we have:

$$\begin{aligned}
& f \preceq [\neg\varphi] \curlyvee f && (\dagger) \\
& \mathsf{slp}[\![C]\!]\,(f) \preceq \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee f) && (\text{by Monotonicity of } \mathsf{slp}) \\
& f \preceq \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee f) && (\text{by hypothesis and transitivity of } \preceq) \\
& f \preceq f \curlywedge \mathsf{slp}[\![C]\!]\,([\neg\varphi] \curlyvee f) && (\text{by Monotonicity of } \lambda X\colon f \curlywedge X) \\
& {}^{\mathsf{slp}}\Psi_f(+\infty) \preceq {}^{\mathsf{slp}}\Psi_f^2(+\infty) && (\text{by Definition 3.4.2})
\end{aligned}$$

Hence, the Kleene's iterates have converged immediately and the greatest fixpoint is exactly:

$$\mathsf{gfp}\ X\colon {}^{\mathsf{slp}}\Psi_f(X) =^{\mathsf{slp}} \Psi_f(+\infty) = f\ ,$$

and thus we conclude:

$$\begin{aligned}
\mathsf{sp}\,[\![\texttt{while}\,(\,\varphi\,)\,\{\,C\,\}]\!]\,(f) &= [\varphi] \curlyvee \mathsf{gfp}\ X\colon {}^{\mathsf{slp}}\Psi_f(X) && (\text{by Definition 3.4.2}) \\
&= [\varphi] \curlyvee f\ . && (\mathsf{gfp}\ X\colon {}^{\mathsf{slp}}\Psi_f(X) = f)
\end{aligned}$$

□

## A.6 Full calculations of Section 3.8

### A.6.1 Full calculations of Example 3.8.1

**Example A.6.1.** *The strongest post of*

$$C = \texttt{if}\ (\,hi > 7\,)\ \{\,lo \coloneqq 99\,\}\ \texttt{else}\ \{\,lo \coloneqq 80\,\}$$

*for the preanticipation $hi = \lambda\sigma\colon \sigma(hi)$ are:*

$$\begin{aligned}
& \mathsf{sp}\,[\![\texttt{if}\ (\,hi > 7\,)\ \{\,lo \coloneqq 99\,\}\ \texttt{else}\ \{\,lo \coloneqq 80\,\}]\!]\,(hi) \\
&\quad = \mathsf{sp}\,[\![lo \coloneqq 99]\!]\,([hi > 7] \curlywedge hi) \curlyvee \mathsf{sp}\,[\![lo \coloneqq 80]\!]\,([hi \leq 7] \curlywedge hi)
\end{aligned}$$

$$
\begin{aligned}
&= \text{Ƨ}\alpha\colon\ \ [lo = 99] \curlywedge ([hi > 7] \curlywedge hi)\,[lo/\alpha] \quad \curlyvee \\
&\quad\ \text{Ƨ}\alpha\colon\ \ [lo = 80] \curlywedge ([hi \leq 7] \curlywedge hi)\,[lo/\alpha] \\
&= [lo = 99] \curlywedge [hi > 7] \curlywedge hi \ \ \curlyvee \ \ [lo = 80] \curlywedge [hi \leq 7] \curlywedge hi
\end{aligned}
$$

*and*

$$
\begin{aligned}
&\mathsf{slp}[\![\texttt{if}\ (\,hi > 7\,)\ \{\,lo \coloneqq 99\,\}\ \texttt{else}\ \{\,lo \coloneqq 80\,\}]\!]\,(hi) \\
&\quad = \mathsf{slp}[\![lo \coloneqq 99]\!]\,([hi \leq 7] \curlyvee hi) \ \ \curlywedge\ \ \mathsf{slp}[\![lo \coloneqq 80]\!]\,([hi > 7] \curlyvee hi) \\
&\quad = \big(\text{ℓ}\,\alpha\colon\ \ [lo \neq 99] \curlyvee ([hi \leq 7] \curlyvee hi)\,[lo/\alpha]\big) \\
&\qquad\quad \curlywedge\ \big(\text{ℓ}\,\alpha\colon\ \ [lo \neq 80] \curlyvee ([hi > 7] \curlyvee hi)\,[lo/\alpha]\big) \\
&\quad = \big([lo \neq 99] \curlyvee [hi \leq 7] \curlyvee hi\big) \ \ \curlywedge\ \ \big([lo \neq 80] \curlyvee [hi > 7] \curlyvee hi\big)\ .
\end{aligned}
$$

### A.6.2 Full calculations of Example 3.8.2

**Example A.6.2.** *The strongest post of*

$$C = hi \coloneqq hi + 5\,;\ \texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo + 1\,\}$$

*for the preanticipation* $hi = \lambda\sigma\colon \sigma(hi)$ *are:*

$$
\begin{aligned}
\mathsf{sp}\,[\![C]\!]\,(hi) &= [lo \geq hi] \curlywedge (hi - 5) \\
\mathsf{slp}[\![C]\!]\,(hi) &= [lo < hi] \curlyvee (hi - 5)
\end{aligned}
$$

*In fact, we have:*

$$
\begin{aligned}
&\mathsf{sp}\,[\![hi \coloneqq hi + 5\,;\ \texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo + 1\,\}]\!]\,(hi) \\
&\quad = \mathsf{sp}\,[\![\texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo + 1\,\}]\!]\,(\mathsf{sp}\,[\![hi \coloneqq hi + 5]\!]\,(hi)) \\
&\quad = \mathsf{sp}\,[\![\texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo + 1\,\}]\!]\,(\text{Ƨ}\alpha\colon\ \ [hi = \alpha + 5] \curlywedge \alpha) \\
&\quad = \mathsf{sp}\,[\![\texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo + 1\,\}]\!]\,(hi - 5) \qquad (\alpha = hi - 5 \text{ is selected}) \\
&\quad = [lo \geq hi] \curlywedge \mathsf{lfp}\ X\colon \Psi_{hi-5}(X)
\end{aligned}
$$

$$= \; [lo \geq hi] \curlywedge \Psi^{\omega}_{hi-5}(-\infty) \qquad \text{(by Kleene's fixpoint theorem)}$$

*Let us compute some Kleene's iterates:*

$$\begin{aligned}
\Psi_{hi-5}(-\infty) &= (hi-5) \;\curlyvee\; \mathsf{sp}\,[\![lo \coloneqq lo+1]\!]\,([lo < hi] \curlywedge -\infty) \\
&= (hi-5) \;\curlyvee\; \mathsf{sp}\,[\![lo \coloneqq lo+1]\!]\,(-\infty) \\
&= (hi-5) \;\curlyvee\; (-\infty) \qquad \text{(by Theorem 3.5.1 (2))} \\
&= (hi-5) \\
\Psi^{2}_{hi-5}(-\infty) &= (hi-5) \;\curlyvee\; \mathsf{sp}\,[\![lo \coloneqq lo+1]\!]\,([lo < hi] \curlywedge (hi-5)) \\
&= (hi-5) \;\curlyvee\; (\text{Ƨ}\alpha\colon\; [lo = \alpha+1] \curlywedge [\alpha < hi] \curlywedge (hi-5)) \\
&= (hi-5) \;\curlyvee\; ([lo < hi+1] \curlywedge (hi-5)) \\
&\qquad\qquad (\alpha = lo - 1 \text{ is selected}) \\
&= (hi-5)
\end{aligned}$$

*The iteration sequence has converged (in just 2 iterations), so we obtain:*

$$\begin{aligned}
&\mathsf{sp}\,[\![hi \coloneqq hi+5\,;\; \texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo+1\,\}]\!]\,(hi) \\
&= \; [lo \geq hi] \curlywedge \Psi^{\omega}_{hi-5}(-\infty) \\
&= \; [lo \geq hi] \curlywedge (hi-5)
\end{aligned}$$

*Similarly, for* slp *we have:*

$$\begin{aligned}
&\mathsf{slp}[\![hi \coloneqq hi+5\,;\; \texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo+1\,\}]\!]\,(hi) \\
&= \mathsf{slp}[\![\texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo+1\,\}]\!]\,(\mathsf{slp}[\![hi \coloneqq hi+5]\!]\,(hi)) \\
&= \mathsf{slp}[\![\texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo+1\,\}]\!]\,(\text{ɩ}\,\alpha\colon\; [hi \neq \alpha+5] \curlyvee \alpha) \\
&= \mathsf{slp}[\![\texttt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo+1\,\}]\!]\,(hi-5) \qquad (\alpha = hi-5 \text{ is selected}) \\
&= [lo < hi] \curlyvee \mathsf{gfp}\; X\colon\, \Psi_{hi-5}(X) \\
&= [lo < hi] \curlyvee \Psi^{\omega}_{hi-5}(+\infty) \qquad \text{(by Kleene's fixpoint theorem)}
\end{aligned}$$

*Let us compute some Kleene's iterates:*

$$
\begin{aligned}
\Psi_{hi-5}(+\infty) &= (hi-5) \curlywedge \mathsf{slp}[\![lo \coloneqq lo+1]\!]\,([lo \geq hi] \curlyvee +\infty) \\
&= (hi-5) \curlywedge \mathsf{slp}[\![lo \coloneqq lo+1]\!]\,(+\infty) \\
&= (hi-5) \curlywedge +\infty \qquad \text{(by Theorem 3.5.1 (4))} \\
&= (hi-5) \\
\Psi^2_{hi-5}(+\infty) &= (hi-5) \curlywedge \mathsf{slp}[\![lo \coloneqq lo+1]\!]\,([lo \geq hi] \curlyvee (hi-5)) \\
&= (hi-5) \curlywedge (\text{\Large ℓ}\, \alpha\colon\ [lo \neq \alpha+1] \curlyvee [\alpha \geq hi] \curlyvee (hi-5)) \\
&= (hi-5) \curlywedge ([lo \geq hi+1] \curlyvee (hi-5)) \\
&\qquad\qquad (\alpha = lo-1 \text{ is selected}) \\
&= (hi-5)
\end{aligned}
$$

*Again, the iteration sequence has converged in 2 iterations, so we conclude:*

$$
\begin{aligned}
&\mathsf{slp}[\![hi \coloneqq hi+5\,;\, \mathtt{while}\,(\,lo < hi\,)\,\{\,lo \coloneqq lo+1\,\}]\!]\,(hi) \\
&= [lo < hi] \curlyvee \Psi^{\omega}_{hi-5}(+\infty) \\
&= [lo < hi] \curlyvee (hi-5)
\end{aligned}
$$

## A.7 Well-definedness of the Semantics for Weighted Programs

In this section we prove that the denotational semantics of Figure 4.1 is a total function. We assume that the operations $\oplus$, $\odot$ belong to a complete, Scott continuous, naturally ordered, partial semiring with a top element.

### A.7.1 Fixed point existence

**Proposition A.7.1.** *Let*

$$
\Phi_{C,e,e'}(X)(\sigma,\tau) = [\![e]\!](\sigma) \odot \Big( \bigoplus_{\iota\colon\, [\![C]\!](\sigma,\iota) \neq \mathbb{0}} [\![C]\!](\sigma,\iota) \odot X(\iota,\tau)\Big) \oplus [\![e']\!](\sigma) \odot [\sigma = \tau]
$$

*If $\Phi_{C,e,e'}$ is a total function, the semantics of loops:*

$$[\![C^{\langle e,e'\rangle}]\!](\sigma,\tau) = (\mathsf{lfp}\ X\colon \Phi_{C,e,e'}(X))(\sigma,\tau)$$

*is well-defined, i.e., the least fixed point of $\Phi_{C,e,e'}$ exists.*

*Proof.* It is sufficient to show that $\Phi_{C,e,e'}$ is Scott-continuous and rely on Kleene's fixpoint theorem to conclude that the fixpoint exists. For all directed sets $D \subseteq (\Sigma \times \Sigma \to U)$ we have:

$$\begin{aligned}
&\sup_{f\in D}\ \Phi_{C,e,e'}(f)(\sigma,\tau)\\
=\ &\sup_{f\in D}\ [\![e]\!](\sigma) \odot \big(\bigoplus_{\iota\in\Sigma} [\![C]\!](\sigma,\iota)\odot f(\iota,\tau)\big)\ \oplus\ [\![e']\!](\sigma)\odot[\sigma=\tau]\\
=\ &[\![e]\!](\sigma) \odot \big(\sup_{f\in D}\ \bigoplus_{\iota\in\Sigma} [\![C]\!](\sigma,\iota)\odot f(\iota,\tau)\big)\ \oplus\ [\![e']\!](\sigma)\odot[\sigma=\tau]\\
&\hfill\text{(by continuity of } \oplus \text{ and } \odot)\\
=\ &[\![e]\!](\sigma) \odot \big(\bigoplus_{\iota\in\Sigma} [\![C]\!](\sigma,\iota)\odot \sup\ D(\iota,\tau)\big)\ \oplus\ [\![e']\!](\sigma)\odot[\sigma=\tau]\\
&\hfill\text{(by [74, Lemma A.4] with } f_\iota(X) = [\![C]\!](\sigma,\iota)\odot X(\iota,\tau) \text{ for } \iota\in\Sigma)\\
=\ &\Phi_{C,e,e'}(\sup\ D)(\sigma,\tau)
\end{aligned}$$

And hence we conclude by Kleene's fixpoint theorem. □

### A.7.2 Syntactic restrictions for partial semirings

Proposition A.7.1 ensures the well-definedness of the iteration rule, provided that $\Phi_{C,e,e'}$ is total. In this section, we investigate syntactic constraints to ensure the totality of $\Phi_{C,e,e'}$ (and all other statements). Notably, challenges arise in partial semirings only, where $\oplus$ might be undefined. The constraints and results above are adapted from [74, Appendix A.3] to our framework.

**Definition A.7.1** (Compatibility [74]). *The expressions $e_1$ and $e_2$ are compatible in semiring $A = \langle U, \oplus, \odot, \mathbb{0}, \mathbb{1}\rangle$ if $[\![e_1]\!](\sigma) \oplus [\![e_2]\!](\sigma)$ is defined for any $\sigma \in \Sigma$.*

**Proposition A.7.2.** *If $e_1, e_2$ are compatible and $[\![C_1]\!], [\![C_2]\!]$ are total functions, then*

$$[\![\{\, \odot\, e_1 \fatsemi\ C_1 \,\} \,\square\, \{\, \odot\, e_2 \fatsemi\ C_2 \,\}]\!]$$

*is a total function.*

*Proof.*

$$\begin{aligned}
&[\![\{\, \odot\, e_1 \fatsemi\ C_1 \,\} \,\square\, \{\, \odot\, e_2 \fatsemi\ C_2 \,\}]\!](\sigma)\\
&\quad= [\![\odot\, e_1 \fatsemi\ C_1]\!](\sigma,\tau) \;\oplus\; [\![\odot\, e_2 \fatsemi\ C_2]\!](\sigma,\tau)\\
&\quad= \bigoplus_{\iota\colon [\![\odot\, e_1]\!](\sigma,\iota)\neq \mathbb{0}} [\![\odot\, e_1]\!](\sigma,\iota) \odot [\![C_1]\!](\iota,\tau)\\
&\qquad \oplus \bigoplus_{\iota\colon [\![\odot\, e_2]\!](\sigma,\iota)\neq \mathbb{0}} [\![\odot\, e_2]\!](\sigma,\iota) \odot [\![C_2]\!](\iota,\tau)\\
&\quad= \bigoplus_{\iota\colon [\![e_1]\!](\sigma)\odot[\sigma=\iota]\neq \mathbb{0}} [\![e_1]\!](\sigma) \odot [\sigma=\iota] \odot [\![C_1]\!](\iota,\tau)\\
&\qquad \oplus \bigoplus_{\iota\colon [\![e_2]\!](\sigma)\odot[\sigma=\iota]\neq \mathbb{0}} [\![e_2]\!](\sigma) \odot [\sigma=\iota] \odot [\![C_2]\!](\iota,\tau)\\
&\quad= [\![e_1]\!](\sigma) \odot [\![C_1]\!](\sigma,\tau) \oplus [\![e_2]\!](\sigma) \odot [\![C_2]\!](\sigma,\tau)
\end{aligned}$$

which is well-defined by [74, Lemma A.5] (since $[\![e_1]\!](\sigma) \oplus [\![e_2]\!](\sigma)$ is well-defined). □

**Proposition A.7.3** (Well-definedness of $C^{\langle e,e'\rangle}$)**.** *If $e, e'$ are compatible and $[\![C]\!]$ is a total function, then $[\![C^{\langle e,e'\rangle}]\!]$ is a total function.*

*Proof.* Let $\Phi_{C,e,e'}(X)(\sigma,\tau) = [\![e]\!](\sigma) \odot \big(\bigoplus_{\iota\in\Sigma} [\![C]\!](\sigma,\iota) \odot X(\iota,\tau)\big) \;\oplus\; [\![e']\!](\sigma) \odot [\sigma=\tau]$. By [74, Lemma A.5], $\Phi_{C,e,e'}(X)(\sigma,\tau)$ is well-defined, ensuring the well-definedness of $[\![C^{\langle e,e'\rangle}]\!]$ as well (as per Proposition A.7.1). □

## A.8 Proofs of Section 4.3

### A.8.1 Proof of Soundness for wp, Theorem 4.3.1

**Theorem 4.3.1** (Characterization of wp)**.** *For all programs $C \in \mathsf{wReg}$ and final states $\tau \in \Sigma$, the following equality holds:*

$$\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) \quad = \quad \bigoplus_{\tau \in \Sigma} \quad [\![C]\!](\sigma, \tau) \odot f(\tau) \ .$$

*Proof.* We prove Theorem 4.3.1 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{wp}\,[\![x \coloneqq e]\!]\,(f)\,(\sigma) \;&=\; f\,[x/e]\,(\sigma) \\
&=\; f(\sigma\,[x/\sigma(e)]) \\
&=\; \bigoplus_{\tau \in \Sigma} [\sigma\,[x/\sigma(e)] = \tau] \odot f(\tau) \\
&=\; \bigoplus_{\tau \in \Sigma} [\![x \coloneqq e]\!](\sigma, \tau) \odot f(\tau) \ .
\end{aligned}$$

**The nondeterministic assignment $x \coloneqq$ `nondet()`:**

We have

$$\begin{aligned}
\mathsf{wp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(f)\,(\sigma) \;&=\; \Big(\bigoplus_{\alpha} \; f\,[x/\alpha]\,\Big)(\sigma) \\
&=\; \bigoplus_{\alpha} \; f(\sigma\,[x/\alpha]) \\
&=\; \bigoplus_{\tau \in \Sigma, \exists \alpha\colon\, \sigma[x/\alpha] = \tau} \quad f(\tau) \quad \text{(by taking } \tau = \sigma\,[x/\alpha]) \\
&=\; \bigoplus_{\tau \in \Sigma} \; \bigoplus_{\alpha \in \mathbb{N}} [\sigma\,[x/\alpha] = \tau] \odot f(\tau)
\end{aligned}$$

$$= \bigoplus_{\tau \in \Sigma} [\![x \coloneqq \texttt{nondet()}]\!](\sigma,\tau) \odot f(\tau)\ .$$

**The weighting $\odot\, w$:**

We have

$$\begin{aligned}
\mathsf{wp}\,[\![\odot\, w]\!]\,(f)\,(\sigma) &= (w \odot f)(\sigma) \\
&= w(\sigma) \odot f(\sigma) \\
&= \bigoplus_{\tau \in \Sigma} w(\sigma) \odot [\sigma = \tau] \odot f(\sigma) \\
&= \bigoplus_{\tau \in \Sigma} w(\sigma) \odot [\sigma = \tau] \odot f(\tau) \\
&= \bigoplus_{\tau \in \Sigma} [\![\odot\, w]\!](\sigma,\tau) \odot f(\tau)\ .
\end{aligned}$$

This concludes the proof for the atomic statement.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$\begin{aligned}
\mathsf{wp}\,[\![C_1 \fatsemi C_2]\!]\,(f)\,(\sigma) &= \mathsf{wp}\,[\![C_1]\!]\,(\mathsf{wp}\,[\![C_2]\!]\,(f))\,(\sigma) \\
&= \bigoplus_{\sigma' \in \Sigma} [\![C_1]\!](\sigma,\sigma') \odot \mathsf{wp}\,[\![C_2]\!]\,(f)\,(\sigma') \quad \text{(by I.H. on } C_1\text{)} \\
&= \bigoplus_{\sigma' \in \Sigma} [\![C_1]\!](\sigma,\sigma') \odot \bigoplus_{\tau \in \Sigma} [\![C_2]\!](\sigma',\tau) \odot f(\tau) \\
&\qquad \text{(by I.H. on } C_2\text{)} \\
&= \bigoplus_{\sigma' \in \Sigma} \bigoplus_{\tau \in \Sigma} [\![C_1]\!](\sigma,\sigma') \odot\ [\![C_2]\!](\sigma',\tau) \odot f(\tau) \\
&\qquad \text{(by distributivity of } \odot\text{)}
\end{aligned}$$

$$= \bigoplus_{\tau\in\Sigma} \big(\bigoplus_{\sigma'\in\Sigma} [\![C_1]\!](\sigma,\sigma') \odot [\![C_2]\!](\sigma',\tau)\big) \odot f(\tau)$$

(by commutativity of $\oplus$)

$$= \bigoplus_{\tau\in\Sigma} [\![C_1 \,;\, C_2]\!](\sigma,\tau) \odot f(\tau) \ .$$

**The nondeterministic choice $\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}$:**

We have

$$\begin{aligned}
\mathsf{wp}\,[\![\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}]\!]\,(f)\,(\sigma) &= \mathsf{wp}\,[\![C_1]\!]\,(f) \oplus \mathsf{wp}\,[\![C_2]\!]\,(f) \\
&= \bigoplus_{\tau\in\Sigma}[\![C_1]\!](\sigma,\tau) \odot f(\tau) \oplus \bigoplus_{\tau\in\Sigma}[\![C_2]\!](\sigma,\tau) \odot f(\tau) \\
&\qquad\qquad \text{(by I.H. on } C_1, C_2\text{)} \\
&= \bigoplus_{\tau\in\Sigma} ([\![C_1]\!](\sigma,\tau) \oplus [\![C_2]\!](\sigma,\tau)) \odot f(\tau) \\
&\qquad\qquad \text{(by distributivity of } \odot\text{)} \\
&= \bigoplus_{\tau\in\Sigma} [\![\{\, C_1 \,\} \,\Box\, \{\, C_2 \,\}]\!](\sigma,\tau) \odot f(\sigma) \ .
\end{aligned}$$

**The Iteration $C^{\langle e,e'\rangle}$:**

Let

$$\Phi_f(X) \;=\; [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X) \ \ ,$$

be the $\mathsf{wp}$-characteristic function of the iteration $C^{\langle e,e'\rangle}$ with respect to any preanticipation $f$ and

$$F(X)(\sigma,\tau) \;=\; \sigma(e) \odot \left(\bigoplus_{\sigma'\in\Sigma}[\![C]\!](\sigma,\sigma') \odot X(\sigma',\tau)\right) \oplus \sigma(e') \odot [\sigma = \tau] \ \ ,$$

be the denotational semantics characteristic function of the loop $C^{\langle e, e' \rangle}$ for any input $\sigma, \tau \in \Sigma$. We first prove by induction on $n$ that, for all $\sigma \in \Sigma, f \in \mathbb{A}$

$$\Phi_f^n(\mathbb{0})(\sigma) \;=\; \bigoplus_{\tau \in \Sigma} F^n(\mathbb{0})(\sigma, \tau) \odot f(\tau)\ . \tag{A.9}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned} \Phi_f^n(\mathbb{0})(\sigma) \;&=\; \mathbb{0} \\ &=\; \bigoplus_{\tau \in \Sigma} F^0(\mathbb{0})(\sigma, \tau) \odot f(\tau)\ . \end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$ and all $\tau \in \Sigma, f \in \mathbb{A}$

$$\Phi_f^n(\mathbb{0})(\sigma) \;=\; \bigoplus_{\tau \in \Sigma} F^n(\mathbb{0})(\sigma, \tau) \odot f(\tau)\ .$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned} &\Phi_f^{n+1}(\mathbb{0})(\sigma) \\ &=\; \big(\llbracket e' \rrbracket \odot f \oplus \llbracket e \rrbracket \odot \mathsf{wp}\,\llbracket C \rrbracket \left(\Phi_f^n(\mathbb{0})\right)\big)\,(\sigma) \\ &=\; \llbracket e' \rrbracket(\sigma) \odot f(\sigma) \oplus \llbracket e \rrbracket(\sigma) \odot \mathsf{wp}\,\llbracket C \rrbracket \left(\Phi_f^n(\mathbb{0})\right)(\sigma) \\ &=\; \sigma(e') \odot f(\sigma) \oplus \sigma(e) \odot \bigoplus_{\sigma' \in \Sigma} \llbracket C \rrbracket(\sigma, \sigma') \odot \Phi_f^n(\mathbb{0})(\sigma') \qquad \text{(by I.H. on } C\text{)} \\ &=\; \sigma(e') \odot f(\sigma) \oplus \sigma(e) \odot \bigoplus_{\sigma' \in \Sigma} \llbracket C \rrbracket(\sigma, \sigma') \odot \bigoplus_{\tau \in \Sigma} F^n(\mathbb{0})(\sigma', \tau) \odot f(\tau) \\ &\hphantom{=\;} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by I.H. on } n\text{)} \\ &=\; \sigma(e') \odot f(\sigma) \oplus \bigoplus_{\tau \in \Sigma} \big(\sigma(e) \odot \bigoplus_{\sigma' \in \Sigma} \llbracket C \rrbracket(\sigma, \sigma') \odot F^n(\mathbb{0})(\sigma', \tau)\big) \odot f(\tau) \\ &\hphantom{=\;} \qquad \text{(by distributivity of } \odot\text{, commutativity and associativity of } \oplus\text{)} \\ &=\; \big(\bigoplus_{\tau \in \Sigma} \sigma(e') \odot [\sigma = \tau] \odot f(\tau)\big) \\ &\hphantom{=\;} \oplus \bigoplus_{\tau \in \Sigma} \big(\sigma(e) \odot \bigoplus_{\sigma' \in \Sigma} \llbracket C \rrbracket(\sigma, \sigma') \odot F^n(\mathbb{0})(\sigma', \tau)\big) \odot f(\tau) \\ &=\; \bigoplus_{\tau \in \Sigma} \big(\sigma(e) \odot \bigoplus_{\sigma' \in \Sigma} \llbracket C \rrbracket(\sigma, \sigma') \odot F^n(\mathbb{0})(\sigma', \tau)\big) \odot f(\tau) \end{aligned}$$

$$
\begin{aligned}
&\oplus \Big( \bigoplus_{\tau \in \Sigma} \sigma(e') \odot [\sigma = \tau] \odot f(\tau) \Big) && \text{(by commutativity of } \oplus\text{)} \\
= {} & \bigoplus_{\tau \in \Sigma} \Big( \sigma(e) \odot \bigoplus_{\sigma' \in \Sigma} [\![C]\!](\sigma, \sigma') \odot F^n(\mathbb{0})(\sigma', \tau) \oplus \sigma(e') \odot [\sigma = \tau] \Big) \odot f(\tau) \\
& && \text{(by associativity of } \oplus \text{ and distributivity of } \odot\text{)} \\
= {} & \bigoplus_{\tau \in \Sigma} F^{n+1}(\mathbb{0})(\sigma, \tau) \odot f(\tau) \ .
\end{aligned}
$$

This concludes the induction on $n$. Now we have:

$$
\begin{aligned}
\mathsf{wp}\, [\![C^{\langle e, e' \rangle}]\!]\, (f)\, (\sigma) &= \Big( \mathsf{lfp}\ X\colon [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\, [\![C]\!]\, (X) \Big)(\sigma) \\
&= \sup_{n \in \mathbb{N}}\ \Phi_f^n(0)(\sigma) && \text{(by Kleene's fixpoint theorem)} \\
&= \sup_{n \in \mathbb{N}}\ \bigoplus_{\tau \in \Sigma}\ F^n(\mathbb{0})(\sigma, \tau) \odot f(\tau) && \text{(by Equation A.9)} \\
&= \bigoplus_{\tau \in \Sigma} \sup_{n \in \mathbb{N}}\ F^n(\mathbb{0})(\sigma, \tau) \odot f(\tau) \\
& && \text{(by continuity of } \lambda X\colon \textstyle\bigoplus_\tau X(\sigma, \tau) \odot f(\tau)\text{)} \\
&= \bigoplus_{\tau \in \Sigma}\ [\![C^{\langle e, e' \rangle}]\!](\sigma, \tau) \odot f(\tau) \ . \\
& && \text{(by Kleene's fixpoint theorem)}
\end{aligned}
$$

and this concludes the proof. □

## A.9 Proofs of Section 4.4

### A.9.1 Proof of Soundness for sp, Theorem 4.4.2

**Theorem 4.4.2** (Characterization of sp). *For all programs $C \in \mathsf{wReg}$ and final states $\tau \in \Sigma$,*

$$
\mathsf{sp}\, [\![C]\!]\, (\mu)\, (\tau) \quad = \quad \bigoplus_{\sigma \in \Sigma}\ \mu(\sigma) \odot [\![C]\!](\sigma, \tau) \ .
$$

*Proof.* We prove Theorem 4.4.2 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq e]\!]\,(f)\,(\tau) &= \Big(\bigoplus_{\alpha} f\,[x/\alpha] \odot [x = e\,[x/\alpha]]\Big)(\tau) \\
&= \bigoplus_{\alpha\colon \tau(x)=\tau(e[x/\alpha])} f\,[x/\alpha]\,(\tau) \\
&= \bigoplus_{\alpha\colon \tau(x)=\tau(e[x/\alpha])} f(\tau\,[x/\alpha]) \\
&= \bigoplus_{\alpha\colon \tau[x/\alpha][x/\tau(e[x/\alpha])]=\tau} f(\tau\,[x/\alpha]) \\
&= \bigoplus_{\alpha\colon \tau[x/\alpha][x/\tau[x/\alpha](e)]=\tau} f(\tau\,[x/\alpha]) \\
&= \bigoplus_{\sigma\in\Sigma, \sigma[x/\sigma(e)]=\tau} f(\sigma) \qquad \text{(by taking } \sigma = \tau\,[x/\alpha]\text{)} \\
&= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot [\sigma\,[x/\sigma(e)] = \tau] \\
&= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot [\![x \coloneqq e]\!](\sigma,\tau)\ .
\end{aligned}$$

**The nondeterministic assignment $x \coloneqq$ `nondet()`:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(f)\,(\tau) &= \Big(\bigoplus_{\alpha} f\,[x/\alpha]\Big)(\tau) \\
&= \bigoplus_{\alpha} f(\tau\,[x/\alpha]) \\
&= \bigoplus_{\sigma\in\Sigma, \exists\alpha\colon \tau[x/\alpha]=\sigma} f(\sigma) \quad \text{(by taking } \sigma = \tau\,[x/\alpha]\text{)} \\
&= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot \bigoplus_{\alpha\in\mathbb{N}} [\sigma\,[x/\alpha] = \tau] \\
&= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot [\![x \coloneqq \texttt{nondet()}]\!](\sigma,\tau)\ .
\end{aligned}$$

**The weighting $\odot\, w$:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![\odot\, w]\!]\,(f)\,(\tau) &= (f \odot w)(\tau) \\
&= f(\tau) \odot w(\tau) \\
&= \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot w(\tau) \odot [\sigma = \tau] \\
&= \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot w(\sigma) \odot [\sigma = \tau] \\
&= \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot [\![\odot\, w]\!](\sigma, \tau)\ .
\end{aligned}$$

This concludes the proof for the atomic statement.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi\ C_2$:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![C_1 \fatsemi\ C_2]\!]\,(f)\,(\tau) &= \mathsf{sp}\,[\![C_2]\!]\,(\mathsf{sp}\,[\![C_1]\!]\,(f))\,(\tau) \\
&= \bigoplus_{\sigma' \in \Sigma} \mathsf{sp}\,[\![C_1]\!]\,(f)\,(\sigma') \odot [\![C_2]\!](\sigma', \tau) \qquad \text{(by I.H. on } C_2\text{)} \\
&= \bigoplus_{\sigma' \in \Sigma} \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot [\![C_1]\!](\sigma, \sigma') \odot [\![C_2]\!](\sigma', \tau) \\
&\qquad \text{(by I.H. on } C_1\text{)} \\
&= \bigoplus_{\sigma \in \Sigma} \bigoplus_{\sigma' \in \Sigma} f(\sigma) \odot [\![C_1]\!](\sigma, \sigma') \odot [\![C_2]\!](\sigma', \tau) \\
&\qquad \text{(by commutativity of } \oplus\text{)} \\
&= \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot \bigoplus_{\sigma' \in \Sigma} [\![C_1]\!](\sigma, \sigma') \odot [\![C_2]\!](\sigma', \tau) \\
&\qquad \text{(by distributivity of } \odot\text{)}
\end{aligned}$$

$$= \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot [\![C_1 \fatsemi C_2]\!](\sigma, \tau) \ .$$

**The nondeterministic choice $\{ C_1 \} \mathbin{\Box} \{ C_2 \}$:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![\{ C_1 \} \mathbin{\Box} \{ C_2 \}]\!]\,(f)\,(\tau) \;&=\; \mathsf{sp}\,[\![C_1]\!]\,(f) \oplus \mathsf{sp}\,[\![C_2]\!]\,(f) \\
&=\; \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot [\![C_1]\!](\sigma, \tau) \oplus \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot [\![C_2]\!](\sigma, \tau) \\
&\hspace{14em} \text{(by I.H. on } C_1, C_2\text{)} \\
&=\; \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot ([\![C_1]\!](\sigma, \tau) \oplus [\![C_2]\!](\sigma, \tau)) \\
&\hspace{14em} \text{(by distributivity of } \odot\text{)} \\
&=\; \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot [\![\{ C_1 \} \mathbin{\Box} \{ C_2 \}]\!](\sigma, \tau) \ .
\end{aligned}$$

**The Iteration $C^{\langle e, e' \rangle}$:**

Let

$$\Psi(\mathsf{trnsf}) \;=\; \lambda f \colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!] \ .$$

be the $\mathsf{sp}$-characteristic function of the iteration $C^{\langle e, e' \rangle}$ and

$$F(X)(\sigma, \tau) \;=\; \sigma(e) \odot \left( \bigoplus_{\sigma' \in \Sigma} [\![C]\!](\sigma, \sigma') \odot X(\sigma', \tau) \right) \oplus \sigma(e') \odot [\sigma = \tau] \ ,$$

be the denotational semantics characteristic function of the loop $C^{\langle e, e' \rangle}$ for any input $\sigma, \tau \in \Sigma$. We prove by induction on $n$ that, for all $\tau \in \Sigma, f \in \mathbb{A}$

$$\Psi^n(\lambda g \colon \mathbb{0})(f)(\tau) \;=\; \bigoplus_{\sigma \in \Sigma} f(\sigma) \odot F^n(\mathbb{0})(\sigma, \tau) \ . \tag{A.10}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}\Psi^0(\lambda g\colon \mathbb{0})(f)(\tau) &= \mathbb{0}\\ &= \bigoplus_{\sigma\in\Sigma} f(\sigma)\odot F^0(\mathbb{0})(\sigma,\tau)\ .\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$ and all $\tau\in\Sigma, f\in\mathbb{A}$

$$\Psi^n(\lambda g\colon \mathbb{0})(f)(\tau) = \bigoplus_{\sigma\in\Sigma} f(\sigma)\odot F^n(\mathbb{0})(\sigma,\tau)\ .$$

For the induction step $n\longrightarrow n+1$, consider the following:

$$\begin{aligned}
&\Psi^{n+1}(\lambda g\colon \mathbb{0})(f)(\tau)\\
&= (\Psi^n(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(f\odot[\![e]\!]))\oplus f\odot[\![e']\!])(\tau)\\
&= (f\odot[\![e']\!])(\tau)\oplus\Psi^n(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(f\odot[\![e]\!]))(\tau)\\
&= f(\tau)\odot\tau(e')\oplus\Psi^n(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(f\odot[\![e]\!]))(\tau)\\
&= f(\tau)\odot\tau(e')\oplus\bigoplus_{\sigma'\in\Sigma}\ \mathsf{sp}\,[\![C]\!]\,(f\odot[\![e]\!])\,(\sigma')\odot F^n(\mathbb{0})(\sigma',\tau) && \text{(by I.H. on } n\text{)}\\
&= f(\tau)\odot\tau(e')\oplus\bigoplus_{\sigma'\in\Sigma}\ \bigoplus_{\sigma\in\Sigma}\ f(\sigma)\odot\sigma(e)\odot[\![C]\!](\sigma,\sigma')\odot F^n(\mathbb{0})(\sigma',\tau)\\
& && \text{(by I.H. on } C\text{)}\\
&= f(\tau)\odot\tau(e')\oplus\bigoplus_{\sigma\in\Sigma}\ \bigoplus_{\sigma'\in\Sigma}\ f(\sigma)\odot\sigma(e)\odot[\![C]\!](\sigma,\sigma')\odot F^n(\mathbb{0})(\sigma',\tau)\\
&= f(\tau)\odot\tau(e')\oplus\bigoplus_{\sigma\in\Sigma}\ f(\sigma)\odot\sigma(e)\odot\bigoplus_{\sigma'\in\Sigma}[\![C]\!](\sigma,\sigma')\odot F^n(\mathbb{0})(\sigma',\tau)\\
& && \text{(by distributivity of } \odot\text{)}\\
&= \Big(\bigoplus_{\sigma\in\Sigma}\ f(\sigma)\odot\sigma(e')\odot[\sigma=\tau]\Big)\\
&\quad\oplus\Big(\bigoplus_{\sigma\in\Sigma}\ f(\sigma)\odot\sigma(e)\odot\bigoplus_{\sigma'\in\Sigma}[\![C]\!](\sigma,\sigma')\odot F^n(\mathbb{0})(\sigma',\tau)\Big)\\
&= \Big(\bigoplus_{\sigma\in\Sigma}\ f(\sigma)\odot\sigma(e)\odot\bigoplus_{\sigma'\in\Sigma}[\![C]\!](\sigma,\sigma')\odot F^n(\mathbb{0})(\sigma',\tau)\Big)\\
&\quad\oplus\Big(\bigoplus_{\sigma\in\Sigma}\ f(\sigma)\odot\sigma(e')\odot[\sigma=\tau]\Big) && \text{(by commutativity of } \oplus\text{)}
\end{aligned}$$

$$= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot \left( \big(\sigma(e) \odot \bigoplus_{\sigma'\in\Sigma} [\![C]\!](\sigma,\sigma') \odot F^n(\mathbb{0})(\sigma',\tau)\big) \oplus \sigma(e') \odot [\sigma=\tau] \right)$$

(by associativity of $\oplus$ and distributivity of $\odot$)

$$= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot F^{n+1}(\mathbb{0})(\sigma,\tau) \ .$$

This concludes the induction on $n$. Now we have:

$$\begin{aligned}
\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f)\,(\tau) &= \big(\mathsf{lfp}\ X\colon f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])\,\big)(\tau) \odot [\![e']\!](\tau) \\
&= \big(\sup_{n\in\mathbb{N}}\ \Psi_f^n(0)(\tau)\big) \odot \tau(e') \quad \text{(by Kleene's fixpoint theorem)} \\
&= \sup_{n\in\mathbb{N}}\ \Psi_f^n(0)(\tau) \odot \tau(e') \\
&= \sup_{n\in\mathbb{N}}\ \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot F^n(\mathbb{0})(\sigma,\tau) \qquad \text{(by Equation A.10)} \\
&= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot \sup_{n\in\mathbb{N}}\ F^n(\mathbb{0})(\sigma,\tau) \\
&\qquad \text{(by continuity of } \lambda X\colon \textstyle\bigoplus_\sigma f(\sigma)\odot X(\sigma,\tau)) \\
&= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot [\![C^{\langle e,e'\rangle}]\!](\sigma,\tau) \ . \\
&\qquad \text{(by Kleene's fixpoint theorem)}
\end{aligned}$$

$$\begin{aligned}
\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f)\,(\tau) &= \big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(f)(\tau) \\
&= (\sup_{n\in\mathbb{N}}\ \Psi^n(\lambda g\colon \mathbb{0}))(f)(\tau) \quad \text{(by Kleene's fixpoint theorem)} \\
&= \sup_{n\in\mathbb{N}}\ (\Psi^n(\lambda g\colon \mathbb{0}))(f)(\tau) \\
&= \sup_{n\in\mathbb{N}}\ \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot F^n(\mathbb{0})(\sigma,\tau) \qquad \text{(by Equation A.10)} \\
&= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot \sup_{n\in\mathbb{N}}\ F^n(\mathbb{0})(\sigma,\tau) \\
&\qquad \text{(by continuity of } \lambda X\colon \textstyle\bigoplus_\sigma f(\sigma)\odot X(\sigma,\tau))
\end{aligned}$$

$$= \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot [\![C^{\langle e,e'\rangle}]\!](\sigma,\tau) \;.$$

(by Kleene's fixpoint theorem)

and this concludes the proof. □

### A.9.2 Proof of Loop rule for total semirings, Theorem 4.4.1

**Theorem 4.4.1** (Loop rule for total semirings)**.** *For all programs $C \in \mathsf{wReg}$, if the ambient semiring is a total semiring, the simplified loop rule:*

$$\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \;=\; \big(\mathsf{lfp}\ X\colon f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])\,\big) \odot [\![e']\!]$$

*holds for all $f \in \mathbb{A}$.*

*Proof.* We define the general characteristic function $\Psi$ as:

$$\Psi(\mathsf{trnsf}) \;=\; \lambda f\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!] \;.$$

We define the characteristic function for total semirings $S_f$ as:

$$S_f(X) \;=\; f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!]) \;.$$

We prove by induction on $n$ that, for all $f \in \mathbb{A}$

$$\Psi^n(\lambda g\colon \mathbb{0})(f) \;=\; S_f^n(\mathbb{0}) \odot [\![e']\!] \;. \tag{A.11}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}\Psi^0(\lambda g\colon \mathbb{0})(f) &= \mathbb{0}\\ &= \mathbb{0} \odot [\![e']\!]\\ &= S_f^0(\mathbb{0}) \odot [\![e']\!] \;.\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$ and all $f \in \mathbb{A}$

$$\Psi^n(\lambda g\colon \mathbb{0})(f) \;=\; S^n_f(\mathbb{0}) \odot [\![e']\!]\ .$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}
\Psi^{n+1}(\lambda g\colon \mathbb{0})(f) &= \Psi(\Psi^n(\lambda g\colon \mathbb{0}))(f)\\
&= \Psi^n(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!]\\
&= S^n_{\mathsf{sp}\,[\![C]\!](f\odot[\![e]\!])}(\mathbb{0}) \odot [\![e']\!] \oplus f \odot [\![e']\!] && \text{(by I.H. on } n)\\
&= \big(S^n_{\mathsf{sp}\,[\![C]\!](f\odot[\![e]\!])}(\mathbb{0}) \oplus f\big) \odot [\![e']\!]\\
& && \text{(by distributivity; total semiring)}\\
&= S^{n+1}_f(\mathbb{0}) \odot [\![e']\!]\ .
\end{aligned}$$

This concludes the induction on $n$. Now we have:

$$\begin{aligned}
\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) &= \big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(f)\\
&= (\sup_{n\in\mathbb{N}}\ \Psi^n(\lambda g\colon \mathbb{0}))(f) && \text{(by Kleene's fixpoint theorem)}\\
&= \sup_{n\in\mathbb{N}}\ (\Psi^n(\lambda g\colon \mathbb{0}))(f)\\
&= \sup_{n\in\mathbb{N}}\ S^n_f(\mathbb{0}) \odot [\![e']\!] && \text{(by Equation A.11)}\\
&= \big(\mathsf{lfp}\ X\colon f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])\big) \odot [\![e']\!],\\
& && \text{(by Kleene's fixpoint theorem)}
\end{aligned}$$

and this concludes the proof. □

## A.10 Proofs of Section 4.5

### A.10.1 Proof of Falsifying correctness triples via correctness triples, Theorem 4.5.1

**Theorem 4.5.1** (Falsifying correctness triples via correctness triples)**.**

$$\begin{aligned}
\models_{\mathrm{pc}} \{\,P\,\}\; C\; \{\,Q\,\} \quad &\text{iff} \quad \forall \sigma \in P\colon\ \not\models_{\mathrm{atc}} \{\,\{\sigma\}\,\}\; C\; \{\,\neg Q\,\} \\
\models_{\mathrm{atc}} \{\,P\,\}\; C\; \{\,Q\,\} \quad &\text{iff} \quad \forall \sigma \in P\colon\ \not\models_{\mathrm{pc}} \{\,\{\sigma\}\,\}\; C\; \{\,\neg Q\,\} \\
\models_{\mathrm{pi}} [\,P\,]\; C\; [\,Q\,] \quad &\text{iff} \quad \forall \sigma \in Q\colon\ \not\models_{\mathrm{ti}} [\,\neg P\,]\; C\; [\,\{\sigma\}\,] \\
\models_{\mathrm{ti}} [\,P\,]\; C\; [\,Q\,] \quad &\text{iff} \quad \forall \sigma \in Q\colon\ \not\models_{\mathrm{pi}} [\,\neg P\,]\; C\; [\,\{\sigma\}\,]
\end{aligned}$$

*Proof.* First, let us observe that

$$A \subseteq B \quad \text{iff} \quad \forall x \in A\colon \{x\} \cap B \neq \emptyset$$

Now, we have:

1.

$$\begin{aligned}
\models_{\mathrm{pc}} \{\,P\,\}\; C\; \{\,Q\,\} \quad &\text{iff} \quad P \subseteq \mathsf{wlp}[\![C]\!]\,(Q) \\
&\text{iff} \quad \forall \sigma \in P\colon \{\sigma\} \cap \mathsf{wlp}[\![C]\!]\,(Q) \neq \emptyset \\
&\text{iff} \quad \forall \sigma \in P\colon\ \not\models_{\mathrm{atc}} \{\,\{\sigma\}\,\}\; C\; \{\,\neg Q\,\}
\end{aligned}$$

2.

$$\begin{aligned}
\models_{\mathrm{atc}} \{\,P\,\}\; C\; \{\,Q\,\} \quad &\text{iff} \quad P \subseteq \mathsf{wp}\,[\![C]\!]\,(Q) \\
&\text{iff} \quad \forall \sigma \in P\colon \{\sigma\} \cap \mathsf{wp}\,[\![C]\!]\,(Q) \neq \emptyset \\
&\text{iff} \quad \forall \sigma \in P\colon\ \not\models_{\mathrm{pc}} \{\,\{\sigma\}\,\}\; C\; \{\,\neg Q\,\}
\end{aligned}$$

3.

$$\begin{aligned}\models_{\mathrm{pi}} [\, P\,]\; C\; [\, Q\,] \quad &\text{iff} \quad Q \subseteq \mathsf{slp}[\![C]\!]\,(P)\\ &\text{iff} \quad \forall \sigma \in Q \colon \{\sigma\} \cap \mathsf{slp}[\![C]\!]\,(P) \neq \emptyset\\ &\text{iff} \quad \forall \sigma \in Q \not\models_{\mathrm{ti}} [\, \neg P\,]\; C\; [\, \{\sigma\}\,]\end{aligned}$$

4.

$$\begin{aligned}\models_{\mathrm{ti}} [\, P\,]\; C\; [\, Q\,] \quad &\text{iff} \quad Q \subseteq \mathsf{sp}\,[\![C]\!]\,(P)\\ &\text{iff} \quad \forall \sigma \in Q \colon \{\sigma\} \cap \mathsf{sp}\,[\![C]\!]\,(P) \neq \emptyset\\ &\text{iff} \quad \forall \sigma \in Q \not\models_{\mathrm{pi}} [\, \neg P\,]\; C\; [\, \{\sigma\}\,]\end{aligned}$$

□

## A.11 Proofs of Section 4.7

### A.11.1 Proof of Healthiness Properties of Weighted Transformers, Theorem 4.7.1

**Theorem 4.7.1** (Healthiness Properties of Weighted Transformers)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ and $\mathsf{sp}$ satisfy the following properties:*

1. *Monotonicity:*

$$f \;\preceq\; g \qquad \text{implies} \qquad \mathsf{ttt}\,[\![C]\!]\,(f) \;\preceq\; \mathsf{ttt}\,[\![C]\!]\,(g)\;\;, \quad \text{for } \mathsf{ttt} \in \{\mathsf{wp}, \mathsf{sp}\}\;.$$

2. *Quantitative universal disjunctiveness: For any set of quantities $S \subseteq \mathbb{A}$,*

$$\mathsf{wp}\,[\![C]\!]\,(\Upsilon S) \;=\; \Upsilon\; \mathsf{wp}\,[\![C]\!]\,(S) \quad \text{and} \quad \mathsf{sp}\,[\![C]\!]\,(\Upsilon S) \;=\; \Upsilon\; \mathsf{sp}\,[\![C]\!]\,(S)\;.$$

3. *Strictness:*

$$\mathsf{wp}\,[\![C]\!]\,(\mathbb{0}) \;=\; \mathbb{0} \quad \text{and} \quad \mathsf{sp}\,[\![C]\!]\,(\mathbb{0}) \;=\; \mathbb{0}\;.$$

*Proof.* Each of the properties is proven individually below.

- Quantitative universal disjunctiveness: Theorems A.11.1 and A.11.2;
- Strictness: Corollaries A.11.2.1 and A.11.2.2;
- Monotonicity: Corollary A.11.2.3.

□

**Theorem A.11.1** (Quantitative Universal Disjunctiveness)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ preserve all suprema, i.e. for all $S \subseteq \mathbb{A}$,*

$$\mathsf{wp}\,[\![C]\!]\,(\sup S) \quad=\quad \sup_{g\in S}\ \mathsf{wp}\,[\![C]\!]\,(g)\ \ .$$

*Proof.* We prove Theorem A.11.1 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{wp}\,[\![x \coloneqq e]\!]\,(\sup S) \quad&=\quad (\sup S)\,[x/e]\\
&=\quad \left(\lambda\sigma\colon\ \sup_{g\in S} g(\sigma)\right)[x/e]\\
&=\quad \left(\lambda\sigma\colon\ \sup_{g\in S} g\,[x/e]\,(\sigma)\right)\\
&=\quad \sup_{g\in S} g\,[x/e]\\
&=\quad \sup_{g\in S}\ \mathsf{wp}\,[\![x \coloneqq e]\!]\,(g)\ \ .
\end{aligned}$$

**The nondeterministic assignment $x \coloneqq \texttt{nondet()}$:**

We have

$$\mathsf{wp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\sup S) \quad=\quad \bigoplus_{\alpha}\ (\sup S)\,[x/\alpha]$$

$$
\begin{aligned}
&= \bigoplus_{\alpha} \left(\lambda\sigma\colon \sup_{g\in S} g(\sigma)\right)[x/\alpha] \\
&= \bigoplus_{\alpha} \left(\lambda\sigma\colon \sup_{g\in S} g\,[x/\alpha]\,(\sigma)\right) \\
&= \sup_{g\in S} \bigoplus_{\alpha} g\,[x/\alpha] && \text{(by continuity of } \oplus\text{)} \\
&= \sup_{g\in S} \ \mathsf{wp}\,[\![x \coloneqq \mathtt{nondet()}]\!]\,(g)\ .
\end{aligned}
$$

**The weighting $\odot\, w$:**

We have

$$
\begin{aligned}
\mathsf{wp}\,[\![\odot\, w]\!]\,(\sup S) &= w \odot \sup S \\
&= w \odot \sup_{g\in S}\ g \\
&= \sup_{g\in S}\ (w \odot g) && \text{(by continuity of } \odot\text{)} \\
&= \sup_{g\in S}\ \mathsf{wp}\,[\![\odot\, w]\!]\,(g)\ .
\end{aligned}
$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$
\begin{aligned}
\mathsf{wp}\,[\![C_1 \fatsemi C_2]\!]\,(\sup S) &= \mathsf{wp}\,[\![C_1]\!]\,(\mathsf{wp}\,[\![C_2]\!]\,(\sup S)) \\
&= \mathsf{wp}\,[\![C_1]\!]\left(\sup_{g\in S}\ \mathsf{wp}\,[\![C_2]\!]\,(g)\right) && \text{(by I.H. on } C_2\text{)} \\
&= \sup_{g\in S}\ \mathsf{wp}\,[\![C_1]\!]\,(\mathsf{wp}\,[\![C_2]\!]\,(g)) && \text{(by I.H. on } C_1\text{)} \\
&= \sup_{g\in S}\ \mathsf{wp}\,[\![C_1 \fatsemi C_2]\!]\,(g)\ .
\end{aligned}
$$

**The nondeterministic choice $\{ C_1 \} \Box \{ C_2 \}$:**

We have

$$\begin{aligned}\mathsf{wp}\,[\![\{ C_1 \} \Box \{ C_2 \}]\!]\,(\sup S) &= \mathsf{wp}\,[\![C_1]\!]\,(f) \oplus \mathsf{wp}\,[\![C_2]\!]\,(f) \\ &= \sup_{g\in S}\ \mathsf{wp}\,[\![C_1]\!]\,(g) \oplus \sup_{g\in S}\ \mathsf{wp}\,[\![C_2]\!]\,(g) \\ &\qquad\qquad\qquad\text{(by I.H. on } C_1, C_2\text{)} \\ &= \sup_{g\in S}\ \mathsf{wp}\,[\![C_1]\!]\,(g) \oplus \mathsf{wp}\,[\![C_2]\!]\,(g) \\ &\qquad\qquad\qquad\text{(by continuity of } \oplus\text{)} \\ &= \sup_{g\in S}\ \mathsf{wp}\,[\![\{ C_1 \} \Box \{ C_2 \}]\!]\,(g)\ .\end{aligned}$$

**The Iteration $C^{\langle e,e'\rangle}$:**

Let

$$\Phi_f(X) \;=\; [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X)\ ,$$

be the $\mathsf{wp}$-characteristic function of the iteration $C^{\langle e,e'\rangle}$ with respect to any postquantity $f$. Observe that $\Psi_f(X)$ is continuous by inductive hypothesis on $C$ and by composition of continuous functions. We now prove by induction on $n$ that

$$\Phi^n_{\sup S}(\mathbb{0}) \;=\; \sup_{g\in S} \Phi^n_g(\mathbb{0})\ . \tag{A.12}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}\Phi^0_{\sup S}(\mathbb{0}) &= \mathbb{0} \\ &= \sup_{g\in S} \mathbb{0} \\ &= \sup_{g\in S} \Phi^0_g(\mathbb{0})\ .\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$

$$\Phi^n_{\sup S}(\mathbb{0}) \;=\; \sup_{g \in S} \Phi^n_g(\mathbb{0}) \ .$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}
&\Phi^{n+1}_{\sup S}(\mathbb{0}) \\
&= \llbracket e' \rrbracket \odot \sup S \oplus \llbracket e \rrbracket \odot \mathsf{wp}\,\llbracket C \rrbracket\left(\Phi^n_{\sup S}(\mathbb{0})\right) \\
&= \llbracket e' \rrbracket \odot \sup S \oplus \llbracket e \rrbracket \odot \mathsf{wp}\,\llbracket C \rrbracket\left(\sup_{g \in S} \Phi^n_g(\mathbb{0})\right) && \text{(by I.H. on } n\text{)} \\
&= \llbracket e' \rrbracket \odot \sup S \oplus \llbracket e \rrbracket \odot \sup_{g \in S}\ \mathsf{wp}\,\llbracket C \rrbracket\left(\Phi^n_g(\mathbb{0})\right) && \text{(by I.H. on } C\text{)} \\
&= \sup_{g \in S}\ \llbracket e' \rrbracket \odot g \oplus \sup_{g \in S}\ \llbracket e \rrbracket \odot \mathsf{wp}\,\llbracket C \rrbracket\left(\Phi^n_g(\mathbb{0})\right) && \text{(by continuity of } \odot\text{)} \\
&= \sup_{g \in S}\ \llbracket e' \rrbracket \odot g \oplus \llbracket e \rrbracket \odot \mathsf{wp}\,\llbracket C \rrbracket\left(\Phi^n_g(\mathbb{0})\right) && \text{(by continuity of } \oplus\text{)} \\
&= \sup_{g \in S} \Phi^{n+1}_g(\mathbb{0}) \ .
\end{aligned}$$

This concludes the induction on $n$. Now we have:

$$\begin{aligned}
\mathsf{wp}\,\llbracket C^{\langle e, e' \rangle} \rrbracket\,(\sup S) &= \mathsf{lfp}\ X\colon\ \llbracket e' \rrbracket \odot \sup S \oplus \llbracket e \rrbracket \odot \mathsf{wp}\,\llbracket C \rrbracket\,(X) \\
&= \sup_{n \in \mathbb{N}}\ \Phi^n_{\sup S}(\mathbb{0}) && \text{(by Kleene's fixpoint theorem)} \\
&= \sup_{n \in \mathbb{N}}\ \sup_{g \in S}\ \Phi^n_g(\mathbb{0}) && \text{(by Equation A.12)} \\
&= \sup_{g \in S}\ \sup_{n \in \mathbb{N}}\ \Phi^n_g(\mathbb{0}) \\
&= \sup_{g \in S}\ \mathsf{wp}\,\llbracket C^{\langle e, e' \rangle} \rrbracket\,(g)\,, && \text{(by Kleene's fixpoint theorem)}
\end{aligned}$$

and this concludes the proof. □

**Theorem A.11.2** (Quantitative Universal Disjunctiveness)**.** *For all programs $C$, $\mathsf{sp}\llbracket C \rrbracket$ preserve all suprema, i.e. for all $S \subseteq \mathbb{A}$,*

$$\mathsf{sp}\,\llbracket C \rrbracket\,(\sup S) \;=\; \sup_{g \in S}\ \mathsf{sp}\,\llbracket C \rrbracket\,(g) \ .$$

*Proof.* We prove Theorem A.11.2 by induction on the structure of $C$. For the induction base, we have the atomic statements:

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\sup S) &= \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge (\sup S)\,[x/\alpha] \\
&= \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge \left(\lambda\sigma\colon\ \sup_{g \in S} g(\sigma)\right)[x/\alpha] \\
&= \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge \left(\lambda\sigma\colon\ \sup_{g \in S} g\,[x/\alpha]\,(\sigma)\right) \\
&= \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge \sup_{g \in S} g\,[x/\alpha] \\
&= \reflectbox{$\mathsf{S}$}\alpha\colon\ \ \sup_{g \in S}\,[x = e\,[x/\alpha]] \curlywedge g\,[x/\alpha] \\
&= \sup_{g \in S}\ \reflectbox{$\mathsf{S}$}\alpha\colon\ \ [x = e\,[x/\alpha]] \curlywedge g\,[x/\alpha] \\
&= \sup_{g \in S}\ \mathsf{sp}\,[\![x \coloneqq e]\!]\,(g) \\
&= \sup\ \mathsf{sp}\,[\![x \coloneqq e]\!]\,(S)\ \ .
\end{aligned}$$

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\sup S) &= \bigoplus_{\alpha}\ (\sup S)\,[x/\alpha] \odot [x = e\,[x/\alpha]] \\
&= \bigoplus_{\alpha}\ \left(\lambda\sigma\colon\ \sup_{g \in S}\ g(\sigma)\right)[x/\alpha] \odot [x = e\,[x/\alpha]] \\
&= \bigoplus_{\alpha}\ \left(\lambda\sigma\colon\ \sup_{g \in S}\ g\,[x/\alpha]\,(\sigma)\right) \odot [x = e\,[x/\alpha]] \\
&= \bigoplus_{\alpha}\ \sup_{g \in S}\ (g\,[x/\alpha]) \odot [x = e\,[x/\alpha]] \\
&= \bigoplus_{\alpha}\ \sup_{g \in S}\ g\,[x/\alpha] \odot [x = e\,[x/\alpha]]\ \ \text{(by continuity of $\odot$)} \\
&= \sup_{g \in S}\ \bigoplus_{\alpha}\ g\,[x/\alpha] \odot [x = e\,[x/\alpha]]\ \ \text{(by continuity of $\oplus$)} \\
&= \sup_{g \in S}\ \mathsf{sp}\,[\![x \coloneqq e]\!]\,(g)\ \ .
\end{aligned}$$

**The nondeterministic assignment $x \coloneqq \texttt{nondet()}$:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\sup S) &= \bigoplus_{\alpha} (\sup S)\,[x/\alpha] \\
&= \bigoplus_{\alpha} \Big(\lambda\sigma\colon \sup_{g\in S} g(\sigma)\Big)\,[x/\alpha] \\
&= \bigoplus_{\alpha} \Big(\lambda\sigma\colon \sup_{g\in S} g\,[x/\alpha]\,(\sigma)\Big) \\
&= \sup_{g\in S} \bigoplus_{\alpha} g\,[x/\alpha] \qquad \text{(by continuity of } \oplus\text{)} \\
&= \sup_{g\in S} \mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(g)~.
\end{aligned}$$

$$\begin{aligned}
\mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\sup S) &= \bigoplus_{\alpha} (\sup S)\,[x/\alpha] \\
&= \bigoplus_{\alpha} \Big(\lambda\sigma\colon \sup_{g\in S}~ g(\sigma)\Big)\,[x/\alpha] \\
&= \bigoplus_{\alpha} \Big(\lambda\sigma\colon \sup_{g\in S}~ g\,[x/\alpha]\,(\sigma)\Big) \\
&= \bigoplus_{\alpha} \sup_{g\in S}~ g\,[x/\alpha] \\
&= \sup_{g\in S} \bigoplus_{\alpha} g\,[x/\alpha] \qquad \text{(by continuity of } \oplus\text{)} \\
&= \sup_{g\in S} \mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(g)~.
\end{aligned}$$

**The weighting $\odot\, w$:**

We have

$$\begin{aligned}
\mathsf{sp}\,[\![\odot\, w]\!]\,(\sup S) &= \sup S \odot w \\
&= (\sup_{g\in S}~ g) \odot w \\
&= \sup_{g\in S}~ (g \odot w) \qquad \text{(by continuity of } \odot\text{)}
\end{aligned}$$

$$= \sup_{g \in S} \mathsf{sp}\,\llbracket \odot\, w \rrbracket\,(g) \ .$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi\ C_2$:**

We have

$$\begin{aligned}
\mathsf{sp}\,\llbracket C_1 \fatsemi\ C_2 \rrbracket\,(\sup S) &= \mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\sup S)) \\
&= \mathsf{sp}\,\llbracket C_2 \rrbracket \left( \sup_{g \in S} \ \mathsf{sp}\,\llbracket C_1 \rrbracket\,(g) \right) && \text{(by I.H. on } C_1) \\
&= \sup_{g \in S} \ \mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mathsf{sp}\,\llbracket C_2 \rrbracket\,(g)) && \text{(by I.H. on } C_2) \\
&= \sup_{g \in S} \ \mathsf{sp}\,\llbracket C_1 \fatsemi\ C_2 \rrbracket\,(g) \ .
\end{aligned}$$

**The nondeterministic choice $\{\, C_1 \,\} \mathrel{\Box} \{\, C_2 \,\}$:**

We have

$$\begin{aligned}
\mathsf{sp}\,\llbracket \{\, C_1 \,\} \mathrel{\Box} \{\, C_2 \,\} \rrbracket\,(\sup S) &= \mathsf{sp}\,\llbracket C_1 \rrbracket\,(\sup S) \oplus \mathsf{sp}\,\llbracket C_2 \rrbracket\,(\sup S) \\
&= \sup_{g \in S} \ \mathsf{sp}\,\llbracket C_1 \rrbracket\,(g) \oplus \sup_{g \in S} \ \mathsf{sp}\,\llbracket C_2 \rrbracket\,(g) \\
& \qquad\qquad \text{(by I.H. on } C_1, C_2) \\
&= \sup_{g \in S} \ \mathsf{sp}\,\llbracket C_1 \rrbracket\,(g) \oplus \mathsf{sp}\,\llbracket C_2 \rrbracket\,(g) \\
& \qquad\qquad \text{(by continuity of } \oplus) \\
&= \sup_{g \in S} \ \mathsf{sp}\,\llbracket \{\, C_1 \,\} \mathrel{\Box} \{\, C_2 \,\} \rrbracket\,(g) \ .
\end{aligned}$$

**The Iteration $C^{\langle e,e'\rangle}$:**

Let

$$\Psi(\mathsf{trnsf}) \;=\; \lambda f\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!] \ ,$$

be the $\mathsf{sp}$-characteristic function of the iteration $C^{\langle e,e'\rangle}$. We first show that $\Psi$ is continuous, i.e., for all directed sets $D \subseteq (\mathbb{A} \to \mathbb{A})$ and functions we have:

$$\begin{aligned}
&\sup_{\mathsf{trnsf}\in D} \Psi(\mathsf{trnsf})(f)\\
=\;& \sup_{\mathsf{trnsf}\in D} (\lambda f\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!])(f)\\
=\;& \sup_{\mathsf{trnsf}\in D} (\mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!])\\
=\;& \big(\sup_{\mathsf{trnsf}\in D} \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!]))\big) \oplus f \odot [\![e']\!] && \text{(by continuity of } \oplus)\\
=\;& (\sup_{\mathsf{trnsf}\in D} \mathsf{trnsf})(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!]\\
=\;& \Psi(\sup_{\mathsf{trnsf}\in D} \mathsf{trnsf})(f)
\end{aligned}$$

We now prove by induction on $n$ that

$$\Psi^n(\lambda g\colon \mathbb{0})(\sup S) \;=\; \sup_{f\in S} \Psi^n(\lambda g\colon \mathbb{0})(f) \ . \tag{A.13}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}
\Psi^0(\lambda g\colon \mathbb{0})(\sup S) \;&=\; \lambda g\colon \mathbb{0}\\
&=\; \sup_{f\in S} \, (\lambda g\colon \mathbb{0})(f)\\
&=\; \sup_{f\in S} \Psi^0(\lambda g\colon \mathbb{0})(f) \ .
\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$

$$\Psi^n(\lambda g\colon \mathbb{0})(\sup S) \;=\; \sup_{f\in S} \, \Psi^n(\lambda g\colon \mathbb{0})(f) \ .$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}
&\Psi^{n+1}(\lambda g\colon \mathbb{0})(\sup S) \\
&= \Psi^{n}(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(\sup S \odot [\![e]\!])) \oplus \sup S \odot [\![e']\!] \\
&= \Psi^{n}(\lambda g\colon \mathbb{0})(\sup_{f\in S}\ \mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus \sup S \odot [\![e']\!] && \text{(by I.H. on } C\text{)} \\
&= \sup_{f\in S}\ \Psi^{n}(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus \sup S \odot [\![e']\!] && \text{(by I.H. on } n\text{)} \\
&= \sup_{f\in S}\ \Psi^{n}(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus \sup_{f\in S}\ f \odot [\![e']\!] \\
&= \sup_{f\in S}\ \left(\Psi^{n}(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!]\right) && \text{(by continuity of } \oplus\text{)} \\
&= \sup_{f\in S} \Psi^{n+1}(\lambda g\colon \mathbb{0})(f)\ .
\end{aligned}$$

This concludes the induction on $n$. Now we have:

$$\begin{aligned}
&\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\sup S) \\
&= \big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X \colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(\sup S) \\
&= (\sup_{n\in\mathbb{N}}\ \Psi^{n}(\lambda g\colon \mathbb{0}))(\sup S) && \text{(by Kleene's fixpoint theorem)} \\
&= \sup_{n\in\mathbb{N}}\ (\Psi^{n}(\lambda g\colon \mathbb{0}))(\sup S) \\
&= \sup_{n\in\mathbb{N}}\ \sup_{f\in S}\ \Psi^{n}(\lambda g\colon \mathbb{0})(f) && \text{(by Equation A.13)} \\
&= \sup_{f\in S}\ \sup_{n\in\mathbb{N}}\ \Psi^{n}(\lambda g\colon \mathbb{0})(f) \\
&= \sup_{f\in S}\ \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f)\ , && \text{(by Kleene's fixpoint theorem)}
\end{aligned}$$

and this concludes the proof. □

**Corollary A.11.2.1** (Strictness of $\mathsf{wp}$)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ is strict, i.e.*

$$\mathsf{wp}\,[\![C]\!]\,(\mathbb{0}) \;=\; \mathbb{0}\ .$$

*Proof.*

$$\begin{aligned}\mathsf{wp}\,[\![C]\!]\,(\mathbb{0}) &= \lambda\sigma\colon \bigoplus_{\tau\in\Sigma} [\![C]\!](\sigma,\tau)\odot\mathbb{0}(\tau) && \text{(by Theorem 4.3.1)}\\ &= \mathbb{0}\;.\end{aligned}$$

□

**Corollary A.11.2.2** (Strictness of $\mathsf{sp}$)**.** *For all programs $C$, $\mathsf{sp}[\![C]\!]$ is strict, i.e.*

$$\mathsf{sp}\,[\![C]\!]\,(\mathbb{0}) \;=\; \mathbb{0}\;.$$

*Proof.*

$$\begin{aligned}\mathsf{sp}\,[\![C]\!]\,(\mathbb{0}) &= \lambda\sigma\colon \bigoplus_{\tau\in\Sigma} \mathbb{0}(\tau)\odot[\![C]\!](\tau,\sigma) && \text{(by Theorem 4.4.2)}\\ &= \mathbb{0}\;.\end{aligned}$$

□

**Corollary A.11.2.3** (Monotonicity of Weighted Transformers)**.** *For all programs $C$, $f,g\in\mathbb{A}$, we have*

$$f \;\preceq\; g \qquad \text{implies} \qquad \mathsf{ttt}\,[\![C]\!]\,(f) \;\preceq\; \mathsf{ttt}\,[\![C]\!]\,(g)\;, \quad \text{for } \mathsf{ttt}\in\{\mathsf{wp},\mathsf{sp}\}$$

*Proof.* Direct consequence of universal disjunctiveness. □

## A.11.2 Proof of Extended Healthiness Properties of Weighted Transformers, Theorem 4.7.2

**Theorem 4.7.2** (Extended Healthiness Properties on Weighted Transformers)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ and $\mathsf{sp}$ satisfy the following properties:*

1. *Additivity: For all $f, g \in \mathbb{A}$, we have*

$$\begin{aligned} \mathsf{wp}\,[\![C]\!]\,(f \oplus g) &\quad=\quad \mathsf{wp}\,[\![C]\!]\,(f) \oplus \mathsf{wp}\,[\![C]\!]\,(g) \quad \text{and} \\ \mathsf{sp}\,[\![C]\!]\,(f \oplus g) &\quad=\quad \mathsf{sp}\,[\![C]\!]\,(f) \oplus \mathsf{sp}\,[\![C]\!]\,(g)\ . \end{aligned}$$

2. *Right-homogeneity: For all $a \in U, f \in \mathbb{A}$, we have*

$$\mathsf{wp}\,[\![C]\!]\,(f \odot a) \quad=\quad \mathsf{wp}\,[\![C]\!]\,(f) \odot a\ .$$

3. *Left-homogeneity: For all $a \in U, f \in \mathbb{A}$, we have*

$$\mathsf{sp}\,[\![C]\!]\,(a \odot f) \quad=\quad a \odot \mathsf{sp}\,[\![C]\!]\,(f)\ .$$

*Proof.* Each of the properties is proven individually below.

- Additivity: Theorems A.11.3 and A.11.4;
- Right-homogeneity: Theorem A.11.5;
- Left-homogeneity: Theorem A.11.6;

□

**Theorem A.11.3** (Additivity of $\mathsf{wp}$)**.** *For all programs $C$, $f, g \in \mathbb{A}$, we have*

$$\mathsf{wp}\,[\![C]\!]\,(f \oplus g) \quad=\quad \mathsf{wp}\,[\![C]\!]\,(f) \oplus \mathsf{wp}\,[\![C]\!]\,(g)\ .$$

*Proof.*

$$\begin{aligned} \mathsf{wp}\,[\![C]\!]\,(f \oplus g) &= \lambda\sigma\colon \bigoplus_{\tau\in\Sigma}[\![C]\!](\sigma,\tau) \odot (f \oplus g)(\tau) && \text{(by Theorem 4.3.1)} \\ &= \lambda\sigma\colon \bigoplus_{\tau\in\Sigma}[\![C]\!](\sigma,\tau) \odot (f(\tau) \oplus g(\tau)) && \text{(by definition of } \oplus\text{)} \\ &= \lambda\sigma\colon \bigoplus_{\tau\in\Sigma}([\![C]\!](\sigma,\tau) \odot f(\tau) \oplus [\![C]\!](\sigma,\tau) \odot g(\tau)) \\ &&& \text{(by distributivity of } \odot \text{ over } \oplus\text{)} \end{aligned}$$

$$= \lambda\sigma\colon \left(\bigoplus_{\tau\in\Sigma}[\![C]\!](\sigma,\tau)\odot f(\tau)\oplus\bigoplus_{\tau\in\Sigma}[\![C]\!](\sigma,\tau)\odot g(\tau)\right)$$

(by associativity and commutativity of $\oplus$)

$$= \lambda\sigma\colon (\mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)\oplus\mathsf{wp}\,[\![C]\!]\,(g)\,(\sigma)) \quad \text{(by Theorem 4.3.1)}$$

$$= \mathsf{wp}\,[\![C]\!]\,(f)\oplus\mathsf{wp}\,[\![C]\!]\,(g) \quad \text{(by definition of } \oplus)$$

□

**Theorem A.11.4** (Additivity of sp). *For all programs $C$, $f, g\in\mathbb{A}$, we have*

$$\mathsf{sp}\,[\![C]\!]\,(f\oplus g) \quad = \quad \mathsf{sp}\,[\![C]\!]\,(f)\oplus\mathsf{sp}\,[\![C]\!]\,(g)\ .$$

*Proof.*

$$\begin{aligned}
\mathsf{sp}\,[\![C]\!]\,(f\oplus g) &= \lambda\tau\colon \bigoplus_{\sigma\in\Sigma}(f\oplus g)(\sigma)\odot[\![C]\!](\sigma,\tau) && \text{(by Theorem 4.4.2)}\\
&= \lambda\tau\colon \bigoplus_{\sigma\in\Sigma}(f(\sigma)\oplus g(\sigma))\odot[\![C]\!](\sigma,\tau) && \text{(by definition of } \oplus)\\
&= \lambda\tau\colon \bigoplus_{\sigma\in\Sigma}(f(\sigma)\odot[\![C]\!](\sigma,\tau)\oplus g(\sigma)\odot[\![C]\!](\sigma,\tau))\\
& && \text{(by distributivity of } \odot \text{ over } \oplus)\\
&= \lambda\tau\colon \left(\bigoplus_{\sigma\in\Sigma}f(\sigma)\odot[\![C]\!](\sigma,\tau)\oplus\bigoplus_{\sigma\in\Sigma}g(\sigma)\odot[\![C]\!](\sigma,\tau)\right)\\
& && \text{(by associativity and commutativity of } \oplus)\\
&= \lambda\tau\colon (\mathsf{sp}\,[\![C]\!]\,(f)\,(\tau)\oplus\mathsf{sp}\,[\![C]\!]\,(g)\,(\tau)) && \text{(by Theorem 4.4.2)}\\
&= \mathsf{sp}\,[\![C]\!]\,(f)\oplus\mathsf{sp}\,[\![C]\!]\,(g) && \text{(by definition of } \oplus)
\end{aligned}$$

□

**Theorem A.11.5** (Right-homogeneity of wp). *For all $a\in U, f\in\mathbb{A}$, we have*

$$\mathsf{wp}\,[\![C]\!]\,(f\odot a) \quad = \quad \mathsf{wp}\,[\![C]\!]\,(f)\odot a\ .$$

*Proof.*

$$
\begin{aligned}
\mathsf{wp}\,[\![C]\!]\,(f \odot a) &= \lambda\sigma\colon \bigoplus_{\tau\in\Sigma} [\![C]\!](\sigma,\tau) \odot f(\tau) \odot a && \text{(by Theorem 4.3.1)}\\
&= \lambda\sigma\colon \left(\bigoplus_{\tau\in\Sigma} [\![C]\!](\sigma,\tau) \odot f(\tau)\right) \odot a \\
& && \text{(by distributivity of } \odot \text{ over } \oplus)\\
&= \mathsf{wp}\,[\![C]\!]\,(f) \odot a && \text{(by Theorem 4.3.1)}
\end{aligned}
$$

□

**Theorem A.11.6** (Left-homogeneity of $\mathsf{sp}$)**.** *For all $a \in U, f \in \mathbb{A}$, we have*

$$\mathsf{sp}\,[\![C]\!]\,(a \odot f) \quad = \quad a \odot \mathsf{sp}\,[\![C]\!]\,(f) \ .$$

*Proof.*

$$
\begin{aligned}
\mathsf{sp}\,[\![C]\!]\,(a \odot f) &= \lambda\tau\colon \bigoplus_{\sigma\in\Sigma} a \odot f(\sigma) \odot [\![C]\!](\sigma,\tau) && \text{(by Theorem 4.4.2)}\\
&= \lambda\tau\colon a \odot \bigoplus_{\sigma\in\Sigma} f(\sigma) \odot [\![C]\!](\sigma,\tau) \\
& && \text{(by distributivity of } \odot \text{ over } \oplus)\\
&= a \odot \mathsf{sp}\,[\![C]\!]\,(f) && \text{(by Theorem 4.4.2)}
\end{aligned}
$$

□

## A.11.3 Proof of Right-linearity of wp and Left-linearity of sp, Theorem 4.7.3

**Theorem 4.7.3** (Linearity)**.** *For all programs $C$, $\mathsf{wp}[\![C]\!]$ is right-linear $\mathsf{sp}[\![C]\!]$ is left-linear. That is, for all $f, g \in \mathbb{A}$ and $a \in U$, we have:*

$$
\begin{aligned}
\mathsf{wp}\,[\![C]\!]\,(f \odot a \oplus g) &= \mathsf{wp}\,[\![C]\!]\,(f) \odot a \oplus \mathsf{wp}\,[\![C]\!]\,(g) \ ,\\
\mathsf{sp}\,[\![C]\!]\,(a \odot f \oplus g) &= a \odot \mathsf{sp}\,[\![C]\!]\,(f) \oplus \mathsf{sp}\,[\![C]\!]\,(g) \ .
\end{aligned}
$$

*Proof.*

$$\begin{aligned}\mathsf{wp}\,[\![C]\!]\,(f \odot a \oplus g) &= \mathsf{wp}\,[\![C]\!]\,(f \odot a) \oplus \mathsf{wp}\,[\![C]\!]\,(g) && \text{(by Additivity)}\\ &= \mathsf{wp}\,[\![C]\!]\,(f) \odot a \oplus \mathsf{wp}\,[\![C]\!]\,(g) && \text{(by Right-homogeneity)}\end{aligned}$$

$$\begin{aligned}\mathsf{sp}\,[\![C]\!]\,(a \odot f \oplus g) &= \mathsf{sp}\,[\![C]\!]\,(a \odot f) \oplus \mathsf{sp}\,[\![C]\!]\,(g) && \text{(by Additivity)}\\ &= a \odot \mathsf{sp}\,[\![C]\!]\,(f) \oplus \mathsf{sp}\,[\![C]\!]\,(g) && \text{(by Left-homogeneity)}\end{aligned}$$

□

### A.11.4 Proof of Linearity, Corollary 4.7.3.1

**Corollary 4.7.3.1** (Linearity). *For all programs $C$, if $\odot$ is commutative, both* $\mathsf{wp}[\![C]\!]$ *and* $\mathsf{sp}[\![C]\!]$ *are linear. That is, for all $f, g \in \mathbb{A}$ and $a \in U$, we have:*

$$\begin{aligned}\mathsf{wp}\,[\![C]\!]\,(a \odot f \oplus g) &= a \odot \mathsf{wp}\,[\![C]\!]\,(f) \oplus \mathsf{wp}\,[\![C]\!]\,(g) \enspace ,\\ \mathsf{sp}\,[\![C]\!]\,(a \odot f \oplus g) &= a \odot \mathsf{sp}\,[\![C]\!]\,(f) \oplus \mathsf{sp}\,[\![C]\!]\,(g) \enspace .\end{aligned}$$

*Proof.* Direct consequence of Theorem 4.7.3 and commutativity of $\odot$. □

### A.11.5 Proof of Weighted sp-wp Duality, Theorem 4.7.4

**Theorem 4.7.4** (Weighted sp-wp Duality). *For all programs $C$ and all functions $\mu, g \in \mathbb{A}$, we have*

$$\bigoplus_{\tau \in \Sigma} \mathsf{sp}\,[\![C]\!]\,(\mu)\,(\tau) \odot g(\tau) \quad = \quad \bigoplus_{\sigma \in \Sigma} \mu(\sigma) \odot \mathsf{wp}\,[\![C]\!]\,(g)\,(\sigma) \enspace .$$

*Proof.*

$$\begin{aligned}\bigoplus_{\tau \in \Sigma} \mathsf{sp}\,[\![C]\!]\,(\mu)\,(\tau) \odot g(\tau) &= \bigoplus_{\tau \in \Sigma} \bigoplus_{\sigma \in \Sigma} \mu(\sigma) \odot [\![C]\!](\sigma, \tau) \odot g(\tau)\\ & \qquad\qquad \text{(by Theorem 4.4.2)}\\ &= \bigoplus_{\sigma \in \Sigma} \bigoplus_{\tau \in \Sigma} \mu(\sigma) \odot [\![C]\!](\sigma, \tau) \odot g(\tau)\end{aligned}$$

$$
\begin{aligned}
&= \bigoplus_{\sigma\in\Sigma} \mu(\sigma) \odot \bigoplus_{\tau\in\Sigma} [\![C]\!](\sigma,\tau) \odot g(\tau) \\
&= \bigoplus_{\sigma\in\Sigma} \mu(\sigma) \odot \mathsf{wp}\,[\![C]\!]\,(g)\,(\sigma)\ .
\end{aligned}
$$

(by Theorem 4.3.1)

□

## A.11.6 Proof of Quantitative Inductive Reasoning for wp, Theorem 4.7.6

**Theorem 4.7.6** (Quantitative Inductive Reasoning for wp, Batz et al. [24])**.** *For any program $C$ and any quantities $i, f \in \mathbb{A}$, we have:*

$$\Phi_f(i) \preceq i \implies \mathsf{wp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \preceq i,$$

*where* $\Phi_f(X) = [\![e']\!] \odot f \oplus [\![e]\!] \odot \mathsf{wp}\,[\![C]\!]\,(X)$ *is the characteristic function of* $C^{\langle e,e'\rangle}$ *w.r.t.* $f$. ◁

*Proof.*

$$
\begin{aligned}
&\Phi_f(i) \preceq i && \text{(Premise of the rule)}\\
&\implies \mathsf{lfp}\ X\colon \Phi_f(X) \ \preceq\ i && \text{(by Park's Induction [94])}\\
&\implies \mathsf{wp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \ \preceq\ i && \text{(by Definition 4.3.1)}
\end{aligned}
$$

□

## A.11.7 Proof of Quantitative Inductive Reasoning for sp, Theorem 4.7.7

**Theorem 4.7.7** (Quantitative Inductive Reasoning for sp)**.** *For any program $C$ and any quantities $i, f \in \mathbb{A}$, we have:*

$$\Psi(i) \preceq i \implies \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \preceq i(f),$$

*where* $\Phi(\textit{trnsf}) \;=\; \lambda f\colon \textit{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!]))$ *is the characteristic function of* $C^{\langle e,e'\rangle}$. $\lhd$

*Proof.*

$$\begin{aligned}
&\Psi(i) \preceq i && \text{(Premise of the rule)}\\
&\implies \mathsf{lfp}\ \mathsf{trnsf}\colon \Psi(\mathsf{trnsf}) \;\preceq\; i && \text{(by Park's Induction [94])}\\
&\implies \mathsf{sp}[\![C^{\langle e,e'\rangle}]\!] \;\preceq\; i && \text{(by Definition 4.4.1)}\\
&\implies \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \;\preceq\; i(f)
\end{aligned}$$

$\Box$

## A.11.8 Proof of Quantitative Inductive Reasoning for sp (total semirings), Theorem 4.7.8

**Theorem 4.7.8** (Quantitative Inductive Reasoning for sp (total semirings))**.** *For any program $C$ and any quantities $i, f \in \mathbb{A}$, if the ambient semiring is a total semiring, we have:*

$$\Psi_f(i) \preceq i \implies \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \preceq i \odot [\![e']\!],$$

*where* $\Psi_f(X) \;=\; f \oplus \mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])$. $\lhd$

*Proof.*

$$\begin{aligned}
&\Psi_f(i) \preceq i && \text{(Premise of the rule)}\\
&\implies \mathsf{lfp}\ X\colon \Psi_f(X) \;\preceq\; i && \text{(by Park's Induction [94])}\\
&\implies (\mathsf{lfp}\ X\colon \Psi_f(X)) \odot [\![e']\!] \;\preceq\; i \odot [\![e']\!] && \text{(by monotonicity)}\\
&\implies \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f) \;\preceq\; i \odot [\![e']\!] && \text{(by Definition 4.4.1)}
\end{aligned}$$

$\Box$

## A.12 Proofs of Section 5.3

### A.12.1 Proof of Soundness for whp, Theorem 5.3.2

**Theorem 5.3.2** (Characterization of $\mathsf{whp}$). *For all programs $C$, hyperquantities $\mathit{ff} \in \mathbb{AA}$, and quantities $f \in \mathbb{A}$*

$$\mathsf{whp}\,[\![C]\!]\,(\mathit{ff})\,(f) \quad = \quad \mathit{ff}(\mathsf{sp}\,[\![C]\!]\,(f))\ .$$

*Proof.* We prove Theorem 5.3.2 by induction on the structure of $C$. For the induction base, we have the atomic statement:

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}\mathsf{whp}\,[\![x \coloneqq e]\!]\,(\mathit{ff})\,(\mu) &= \mathit{ff}(\bigoplus_{\alpha}\,[x = e\,[x/\alpha]] \odot \mu\,[x/\alpha]) \\ &= \mathit{ff}(\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\mu))\ .\end{aligned}$$

**The nondeterministic assignment $x \coloneqq \texttt{nondet()}$:**

We have

$$\begin{aligned}\mathsf{whp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\mathit{ff})\,(\mu) &= \mathit{ff}(\bigoplus_{\alpha}\,\mu\,[x/\alpha]) \\ &= \mathit{ff}(\mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\mu))\ .\end{aligned}$$

**The weighting $\odot\, w$:**

We have

$$\begin{aligned}\mathsf{whp}\,[\![\odot\, w]\!]\,(\mathit{ff})\,(\mu) &= (\mathit{ff} \odot w)(\mu) \\ &= \mathit{ff}(\mu \odot w) \\ &= \mathit{ff}(\mathsf{sp}\,[\![\odot\, w]\!]\,(\mu))\ .\end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \mathbin{⨟} C_2$:**

We have

$$\begin{aligned}
\mathsf{whp}\ \llbracket C_1 \mathbin{⨟}\ C_2 \rrbracket\,(ff)\,(\mu) &= \mathsf{whp}\ \llbracket C_1 \rrbracket\,(\mathsf{whp}\ \llbracket C_2 \rrbracket\,(ff))\,(\mu) \\
&= \mathsf{whp}\ \llbracket C_2 \rrbracket\,(ff)\,(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu)) && \text{(by I.H. on } C_1\text{)} \\
&= ff(\mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu))) && \text{(by I.H. on } C_2\text{)} \\
&= ff(\mathsf{sp}\,\llbracket C_1 \mathbin{⨟}\ C_2 \rrbracket\,(\mu))
\end{aligned}$$

**The nondeterministic choice $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$:**

We have

$$\begin{aligned}
&\mathsf{whp}\ \llbracket \{\, C_1 \,\} \,\square\, \{\, C_2 \,\} \rrbracket\,(ff)\,(\mu) \\
=\ &\mathcal{S}\nu_1, \nu_2\colon\ ff(\nu_1 \oplus \nu_2) \cdot \mathsf{whp}\ \llbracket C_1 \rrbracket\,([\nu_1])\,(\mu) \cdot \mathsf{whp}\ \llbracket C_2 \rrbracket\,([\nu_2])\,(\mu) \\
=\ &\mathcal{S}\nu_1, \nu_2\colon\ ff(\nu_1 \oplus \nu_2) \cdot [\nu_1]\,(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu)) \cdot [\nu_2]\,(\mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mu)) \\
& \hfill \text{(by I.H. on } C_1, C_2\text{)} \\
=\ &ff(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu) \oplus \mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mu)) \\
=\ &ff(\mathsf{sp}\,\llbracket \{\, C_1 \,\} \,\square\, \{\, C_2 \,\} \rrbracket\,(\mu))\ .
\end{aligned}$$

**The Iteration $C^{\langle e, e' \rangle}$:**

$$\begin{aligned}
&\mathsf{whp}\ \left\llbracket C^{\langle e, e' \rangle} \right\rrbracket\,(ff)\,(\mu) \\
=\ &ff\big((\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\,\llbracket C \rrbracket\,(X \odot \llbracket e \rrbracket)) \oplus X \odot \llbracket e' \rrbracket)(\mu)\big)
\end{aligned}$$

$$= \mathit{ff}(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu))\ .$$

and this concludes the proof. □

## A.12.2 Proof of Consistency of iteration rule, Proposition 5.3.1

**Proposition 5.3.1** (Consistency of iteration rule)**.** *Let*

$$\Phi(\mathit{trnsf}) = \lambda \mathit{hh}\, \lambda f\colon \mathcal{E}\nu\colon\ \mathit{hh}(\nu \oplus f \odot [\![e']\!]) \cdot \mathsf{whp}\ [\![C]\!]\,(\mathit{trnsf}([\nu]))\,(f \odot [\![e]\!])$$

*Then,* $\mathsf{whp}\,\left[\!\left[C^{\langle e,e'\rangle}\right]\!\right]$ *is a fixpoint of the higher order function* $\Phi(\mathit{trnsf})$*, that is:*

$$\Phi(\lambda \mathit{ff}\, \lambda\mu\colon \mathit{ff}(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu))) = \lambda \mathit{ff}\, \lambda\mu\colon \mathit{ff}(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu))$$

*Proof.*

$$\begin{aligned}
& \Phi(\lambda \mathit{ff}\, \lambda\mu\colon \mathit{ff}(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu))) \\
={}& \lambda \mathit{hh}\, \lambda f\colon \mathcal{E}\nu\colon\ \mathit{hh}(\nu \oplus f \odot [\![e']\!])\cdot \\
& \qquad \mathsf{whp}\ [\![C]\!]\left(\lambda\mu\colon\ [\nu]\,(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu))\right)(f \odot [\![e]\!]) \\
={}& \lambda \mathit{hh}\, \lambda f\colon \mathcal{E}\nu\colon\ \mathit{hh}(\nu \oplus f \odot [\![e']\!]) \cdot [\nu]\,(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!]))) \\
& \hfill \text{(by Theorem 5.3.2)} \\
={}& \lambda \mathit{hh}\, \lambda f\colon\ \mathit{hh}(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!]) \\
={}& \lambda \mathit{hh}\, \lambda f\colon\ \mathit{hh}(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(f)) \\
& (\mathsf{sp}[\![C^{\langle e,e'\rangle}]\!] \text{ is a fixpoint of } \Psi(X) = \lambda f\colon X(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!]) \oplus f \odot [\![e']\!])) \\
={}& \lambda \mathit{ff}\, \lambda\mu\colon \mathit{ff}(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu))\ .
\end{aligned}$$

□

## A.13 Proofs of Section 5.4

### A.13.1 Proof of Soundness for shp, Theorem 5.4.3

**Theorem 5.4.3** (Characterization of shp)**.** *For all programs $C$, hyperquantities $\mathit{ff} \in \mathbb{AA}$ and quantities $f \in \mathbb{A}$*

$$\mathsf{shp}\, [\![C]\!]\,(\mathit{ff})\,(f) \quad = \quad \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C]\!](\mu)=f} \mathit{ff}(\mu)\ .$$

*Proof.* We prove Theorem 5.4.3 by induction on the structure of $C$. For the induction base, we have the atomic statement:

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}\mathsf{shp}\,[\![x \coloneqq e]\!]\,(\mathit{ff})\,(\nu) &= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \curlywedge \left[\bigoplus_{\alpha} \mu\,[x/\alpha] \odot [x = e\,[x/\alpha]] = \nu\right] \\ &= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \curlywedge [\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\mu) = \nu] \\ &= \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![x\coloneqq e]\!](\mu)=\nu} \mathit{ff}(\mu)\ .\end{aligned}$$

**The nondeterministic assignment $x \coloneqq \texttt{nondet()}$:**

We have

$$\begin{aligned}\mathsf{shp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\mathit{ff})\,(\nu) &= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \curlywedge \left[\bigoplus_{\alpha} \mu\,[x/\alpha] = \nu\right] \\ &= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \curlywedge [\mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\mu) = \nu] \\ &= \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![x\coloneqq\texttt{nondet()}]\!](\mu)=\nu} \mathit{ff}(\mu)\ .\end{aligned}$$

**The weighting $\odot\, w$:**

We have

$$\begin{aligned}
\mathsf{shp}\,[\![\odot\, w]\!]\,(\mathit{ff})\,(\nu) &= \reflectbox{$\mathsf{S}$}\mu\colon\ \mathit{ff}(\mu) \curlywedge [\mu \odot w = \nu] \\
&= \reflectbox{$\mathsf{S}$}\mu\colon\ \mathit{ff}(\mu) \curlywedge [\mathsf{sp}\,[\![\odot\, w]\!]\,(\mu) = \nu] \\
&= \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![\odot\, w]\!](\mu)=\nu} \mathit{ff}(\mu)\ .
\end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$\begin{aligned}
\mathsf{shp}\,[\![C_1 \fatsemi C_2]\!]\,(\mathit{ff})\,(\nu) &= \mathsf{shp}\,[\![C_2]\!]\,(\mathsf{shp}\,[\![C_1]\!]\,(\mathit{ff}))\,(\nu) \\
&= \bigcurlyvee_{\mu'\colon \mathsf{sp}\,[\![C_2]\!](\mu')=\nu} \mathsf{shp}\,[\![C_1]\!]\,(\mathit{ff})\,(\mu') \quad \text{(by I.H. on } C_2\text{)} \\
&= \bigcurlyvee_{\mu'\colon \mathsf{sp}\,[\![C_2]\!](\mu')=\nu} \ \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C_1]\!](\mu)=\mu'} \mathit{ff}(\mu) \\
&\qquad\qquad \text{(by I.H. on } C_1\text{)} \\
&= \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C_2]\!](\mathsf{sp}\,[\![C_1]\!](\mu))=\nu} \mathit{ff}(\mu) \\
&= \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C_1 \fatsemi C_2]\!](\mu)=\nu} \mathit{ff}(\mu)\ .
\end{aligned}$$

**The nondeterministic choice $\{\, C_1 \,\} \mathrel{\Box} \{\, C_2 \,\}$:**

We have

$$\mathsf{shp}\,[\![\{\, C_1 \,\} \mathrel{\Box} \{\, C_2 \,\}]\!]\,(\mathit{ff})\,(\nu)$$

$$
\begin{aligned}
&= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \curlywedge \big(\mathsf{shp}\ [\![C_1]\!]\,([\mu]) \boxtimes \mathsf{shp}\ [\![C_2]\!]\,([\mu])\,\big)(\nu)\\
&= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \curlywedge \bigcurlyvee_{\nu_1,\nu_2\colon \nu_1\oplus\nu_2=\nu} \mathsf{shp}\ [\![C_1]\!]\,([\mu])\,(\nu_1) \curlywedge \mathsf{shp}\ [\![C_2]\!]\,([\mu])\,(\nu_2)\\
&= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu)\\
&\qquad \curlywedge \bigcurlyvee_{\nu_1,\nu_2\colon \nu_1\oplus\nu_2=\nu}\ \bigcurlyvee_{\mu'\colon \mathsf{sp}\,[\![C_1]\!](\mu')=\nu_1} [\mu]\,(\mu') \curlywedge \bigcurlyvee_{\mu'\colon \mathsf{sp}\,[\![C_2]\!](\mu')=\nu_2} [\mu]\,(\mu')\\
&\hspace{20em}\text{(by I.H. on } C_1, C_2)\\
&= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \curlywedge \bigcurlyvee_{\nu_1,\nu_2\colon \nu_1\oplus\nu_2=\nu} [\mathsf{sp}\,[\![C_1]\!]\,(\mu) = \nu_1] \curlywedge [\mathsf{sp}\,[\![C_2]\!]\,(\mu) = \nu_2]\\
&= \text{Ƨ}\mu\colon\ \mathit{ff}(\mu) \curlywedge [\mathsf{sp}\,[\![C_1]\!]\,(\mu) \oplus \mathsf{sp}\,[\![C_2]\!]\,(\mu) = \nu]\\
&= \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C_1]\!](\mu)\oplus\mathsf{sp}\,[\![C_2]\!](\mu)=\nu} \mathit{ff}(\mu)\\
&= \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![\{\,C_1\,\}\square\{\,C_2\,\}]\!](\mu)=\nu} \mathit{ff}(\mu)\ .
\end{aligned}
$$

**The Iteration $C^{\langle e,e'\rangle}$:**

$$
\begin{aligned}
&\mathsf{shp}\ \left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\mathit{ff})\,(\nu)\\
&= \text{Ƨ}\mu\colon \mathit{ff}(\mu) \curlywedge \big[\big(\mathsf{lfp\ trnsf}\colon \lambda X : \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(\mu) = \nu\big]\\
&= \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!](\mu)=\nu} \mathit{ff}(\mu)\ .
\end{aligned}
$$

□

## A.13.2 Proof of Consistency of iteration rule, Proposition 5.4.2

**Proposition 5.4.2** (Consistency of iteration rule)**.** *Let*

$$
\begin{aligned}
&\Psi(\mathit{trnsf})\\
&= \lambda \mathit{hh}\, \lambda f : \text{Ƨ}\mu\colon\ \mathit{hh}(\mu) \curlywedge \big(\mathit{trnsf}(\mathsf{shp}\ [\![C]\!]\,(\mathsf{shp}\ [\![\odot\, e]\!]\,([\mu]))) \boxtimes \mathsf{shp}\ [\![\odot\, e']\!]\,([\mu])\,\big)(f)
\end{aligned}
$$

*Then,* $\mathsf{shp}\left[\!\left[C^{\langle e,e'\rangle}\right]\!\right]$ *is a fixpoint of the higher order function* $\Psi(\textit{trnsf})$*, that is:*

$$\begin{aligned}&\Psi\Big(\lambda\mathit{ff}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{ff}(\mu)\curlywedge\Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu)=f\Big]\Big)\\&\quad=\ \lambda\mathit{ff}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{ff}(\mu)\curlywedge\Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu)=f\Big]\end{aligned}$$

*Proof.*

$$\begin{aligned}
&\Psi\Big(\lambda\mathit{ff}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{ff}(\mu)\cdot\Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu)=f\Big]\Big)\\
&=\ \lambda\mathit{hh}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{hh}(\mu)\cdot\Big(\lambda f'\colon \text{Ƨ}\mu'\colon\ \mathsf{shp}\,[\![C]\!]\,(\mathsf{shp}\,[\![\odot\, e]\!]\,([\mu]))\,(\mu')\\
&\qquad\qquad\cdot\Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu')=f'\Big]\boxtimes\mathsf{shp}\,[\![\odot\, e']\!]\,([\mu])\Big)(f)\\
&=\ \lambda\mathit{hh}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{hh}(\mu)\\
&\quad\cdot\mathop{\curlyvee}_{\mu_1,\mu_2\in\mathbb{A}\colon\,\mu_1\oplus\mu_2=f}\ \text{Ƨ}\mu'\colon\ \mathsf{shp}\,[\![\odot\, e\,⨾\ C]\!]\,([\mu])\,(\mu')\cdot\Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu')=\mu_1\Big]\\
&\qquad\qquad\cdot\mathsf{shp}\,[\![\odot\, e']\!]\,([\mu])\,(\mu_2)\\
&=\ \lambda\mathit{hh}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{hh}(\mu)\\
&\quad\cdot\mathop{\curlyvee}_{\mu_1,\mu_2\in\mathbb{A}\colon\,\mu_1\oplus\mu_2=f}\ \text{Ƨ}\mu'\colon\ \mathop{\curlyvee}_{\mathsf{sp}\,[\![\odot\, e\,⨾\ C]\!](\mu)=\mu'\wedge\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!](\mu')=\mu_1}+\infty\\
&\qquad\qquad\cdot\mathop{\curlyvee}_{\mathsf{sp}\,[\![\odot\, e']\!](\mu)=\mu_2}+\infty\qquad\qquad\text{(by Theorem 5.4.3)}\\
&=\ \lambda\mathit{hh}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{hh}(\mu)\\
&\quad\cdot\mathop{\curlyvee}_{\mu_1,\mu_2\in\mathbb{A}\colon\,\mu_1\oplus\mu_2=f}\ \mathop{\curlyvee}_{\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!](\mathsf{sp}\,[\![\odot\, e\,⨾\ C]\!](\mu))=\mu_1\wedge\mathsf{sp}\,[\![\odot\, e']\!](\mu)=\mu_2}+\infty\\
&=\ \lambda\mathit{hh}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{hh}(\mu)\cdot\Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mathsf{sp}\,[\![\odot\, e\,⨾\ C]\!]\,(\mu))\oplus\mathsf{sp}\,[\![\odot\, e']\!]\,(\mu)=f\Big]\\
&=\ \lambda\mathit{hh}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{hh}(\mu)\cdot\Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mathsf{sp}\,[\![C]\!]\,(\mu\odot e'))\oplus\mu\odot e'=f\Big]\\
&=\ \lambda\mathit{hh}\,\lambda f: \text{Ƨ}\mu\colon\ \mathit{hh}(\mu)\cdot\Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu)=f\Big]\\
&\qquad(\mathsf{sp}[\![C^{\langle e,e'\rangle}]\!]\text{ is a fixpoint of }\Psi(X)=\lambda f\colon X(\mathsf{sp}\,[\![C]\!]\,(f\odot[\![e]\!])\oplus f\odot[\![e']\!]))
\end{aligned}$$

□

## A.13.3 Proof of Soundness for slhp, Theorem 5.5.3

**Theorem 5.5.3** (Characterization of slhp). *For all programs $C$, hyperquantities $\mathit{ff} \in \mathbb{AA}$ and quantities $f \in \mathbb{A}$*

$$\mathsf{slhp}\,\llbracket C\rrbracket\,(\mathit{ff})\,(f) \quad = \quad \bigcurlywedge_{\mu\colon\ \mathsf{sp}\,\llbracket C\rrbracket(\mu)=f} \quad \mathit{ff}(\mu)\ .$$

*Proof.* We prove Theorem 5.5.3 by induction on the structure of $C$. For the induction base, we have the atomic statement:

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned}
\mathsf{slhp}\,\llbracket x \coloneqq e\rrbracket\,(\mathit{ff})\,(\nu) &= \ell\,\mu\colon\ \mathit{ff}(\mu) \curlyvee \left[\bigoplus_{\alpha} \mu\,[x/\alpha] \odot [x = e\,[x/\alpha]] \neq \nu\right] \\
&= \ell\,\mu\colon\ \mathit{ff}(\mu) \curlyvee [\mathsf{sp}\,\llbracket x \coloneqq e\rrbracket\,(\mu) \neq \nu] \\
&= \bigcurlywedge_{\mu\colon\ \mathsf{sp}\,\llbracket x\coloneqq e\rrbracket(\mu)=\nu} \quad \mathit{ff}(\mu)\ .
\end{aligned}$$

**The nondeterministic assignment $x \coloneqq$ `nondet()`:**

We have

$$\begin{aligned}
\mathsf{slhp}\,\llbracket x \coloneqq \texttt{nondet()}\rrbracket\,(\mathit{ff})\,(\nu) &= \ell\,\mu\colon\ \mathit{ff}(\mu) \curlyvee \left[\bigoplus_{\alpha} \mu\,[x/\alpha] \neq \nu\right] \\
&= \ell\,\mu\colon\ \mathit{ff}(\mu) \curlyvee [\mathsf{sp}\,\llbracket x \coloneqq \texttt{nondet()}\rrbracket\,(\mu) \neq \nu] \\
&= \bigcurlywedge_{\mu\colon\ \mathsf{sp}\,\llbracket x\coloneqq \texttt{nondet()}\rrbracket(\mu)=\nu} \quad \mathit{ff}(\mu)\ .
\end{aligned}$$

**The weighting $\odot\, w$:**

We have

$$\mathsf{slhp}\,\llbracket \odot\, w\rrbracket\,(\mathit{ff})\,(\nu) \;=\; \ell\,\mu\colon\ \mathit{ff}(\mu) \curlyvee [\mu \odot w \neq \nu]$$

$$
\begin{aligned}
&= \mathop{\text{\large$\curlywedge$}}\mu\colon\; ff(\mu) \curlyvee [\mathsf{sp}\, [\![\odot\, w]\!]\,(\mu) \neq \nu] \\
&= \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![\odot\, w]\!](\mu)=\nu} ff(\mu)\;.
\end{aligned}
$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$
\begin{aligned}
\mathsf{slhp}\,[\![C_1 \fatsemi C_2]\!]\,(ff)\,(\nu) &= \mathsf{slhp}\,[\![C_2]\!]\,(\mathsf{slhp}\,[\![C_1]\!]\,(ff))\,(\nu) \\
&= \bigcurlywedge_{\mu'\colon \mathsf{sp}\,[\![C_2]\!](\mu')=\nu} \mathsf{slhp}\,[\![C_1]\!]\,(ff)\,(\mu') \quad \text{(by I.H. on } C_2) \\
&= \bigcurlywedge_{\mu'\colon \mathsf{sp}\,[\![C_2]\!](\mu')=\nu}\; \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![C_1]\!](\mu)=\mu'} ff(\mu) \\
&\qquad \text{(by I.H. on } C_1) \\
&= \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![C_2]\!](\mathsf{sp}\,[\![C_1]\!](\mu))=\nu} ff(\mu) \\
&= \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![C_1 \fatsemi C_2]\!](\mu)=\nu} ff(\mu)\;.
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{slhp}\,[\![\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}]\!]\,(ff)\,(\nu) \\
=\;& \mathop{\text{\large$\curlywedge$}}\mu\colon\; ff(\mu) \curlyvee \big(\mathsf{slhp}\,[\![C_1]\!]\,([\neg\mu]) \boxplus \mathsf{slhp}\,[\![C_2]\!]\,([\neg\mu])\,\big)(\nu) \\
=\;& \mathop{\text{\large$\curlywedge$}}\mu\colon\; ff(\mu) \curlyvee \bigcurlywedge_{\nu_1,\nu_2\colon \nu_1\oplus\nu_2=\nu} \mathsf{slhp}\,[\![C_1]\!]\,([\neg\mu])\,(\nu_1) \curlyvee \mathsf{slhp}\,[\![C_2]\!]\,([\neg\mu])\,(\nu_2) \\
=\;& \mathop{\text{\large$\curlywedge$}}\mu\colon\; ff(\mu) \\
&\quad \curlyvee \bigcurlywedge_{\nu_1,\nu_2\colon \nu_1\oplus\nu_2=\nu}\; \bigcurlywedge_{\mu'\colon \mathsf{sp}\,[\![C_1]\!](\mu')=\nu_1} [\neg\mu]\,(\mu') \curlyvee \bigcurlywedge_{\mu'\colon \mathsf{sp}\,[\![C_2]\!](\mu')=\nu_2} [\neg\mu]\,(\mu') \\
&\qquad \text{(by I.H. on } C_1, C_2)
\end{aligned}
$$

$$
\begin{aligned}
&= \text{℩}\,\mu\colon\ \mathit{ff}(\mu) \curlyvee \bigcurlywedge_{\nu_1,\nu_2\colon \nu_1\oplus\nu_2=\nu} [\mathsf{sp}\,[\![C_1]\!]\,(\mu) \neq \nu_1] \curlyvee [\mathsf{sp}\,[\![C_2]\!]\,(\mu) \neq \nu_2] \\
&= \text{℩}\,\mu\colon\ \mathit{ff}(\mu) \curlyvee [\mathsf{sp}\,[\![C_1]\!]\,(\mu) \oplus \mathsf{sp}\,[\![C_2]\!]\,(\mu) \neq \nu] \\
&= \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![C_1]\!](\mu)\oplus\mathsf{sp}\,[\![C_2]\!](\mu)=\nu} \mathit{ff}(\mu) \\
&= \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![\{\,C_1\,\}\Box\{\,C_2\,\}]\!](\mu)=\nu} \mathit{ff}(\mu)\ .
\end{aligned}
$$

**The Iteration $C^{\langle e,e'\rangle}$:**

$$
\begin{aligned}
&\mathsf{slhp}\ \left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\mathit{ff})\,(\nu) \\
&= \text{℩}\,\mu\colon \mathit{ff}(\mu) \curlyvee \left[\big(\mathsf{lfp}\ \mathsf{trnsf}\colon \lambda X\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(X \odot [\![e]\!])) \oplus X \odot [\![e']\!]\big)(\mu) \neq \nu\right] \\
&= \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!](\mu)=\nu} \mathit{ff}(\mu)\ .
\end{aligned}
$$

□

## A.13.4 Proof of Consistency of iteration rule, Proposition 5.5.2

**Proposition 5.5.2** (Consistency of iteration rule)**.** *Let*

$$
\begin{aligned}
\Psi(\mathit{trnsf}) \;=\;& \lambda \mathit{hh}\,\lambda f\colon \text{℩}\,\mu\colon \\
& \mathit{hh}(\mu) \curlyvee \big(\mathit{trnsf}(\mathsf{slhp}\ [\![C]\!]\,(\mathsf{shp}\ [\![\odot\, e]\!]\,([\neg\mu]))) \boxplus \mathsf{shp}\ [\![\odot\, e']\!]\,([\neg\mu])\,\big)(f)
\end{aligned}
$$

*Then,* $\mathsf{slhp}\ [\![C^{\langle e,e'\rangle}]\!]$ *is a fixpoint of the higher order function* $\Psi(\mathit{trnsf})$*, that is:*

$$
\begin{aligned}
&\Psi\Big(\lambda\mathit{ff}\,\lambda f\colon \text{℩}\,\mu\colon\ \mathit{ff}(\mu) \curlyvee \left[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu) \neq f\right]\Big) \\
&\quad = \lambda\mathit{ff}\,\lambda f\colon \text{℩}\,\mu\colon\ \mathit{ff}(\mu) \curlyvee \left[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu) \neq f\right]
\end{aligned}
$$

*Proof.*

$$
\Psi\Big(\lambda\mathit{ff}\,\lambda f\colon \text{℩}\,\mu\colon\ \mathit{ff}(\mu) \curlyvee \left[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu) \neq f\right]\Big)
$$

$$
\begin{aligned}
&= \lambda hh\, \lambda f\colon \boldsymbol{\ell}\,\mu\colon\ hh(\mu) \curlyvee \Big(\lambda f'\colon \boldsymbol{\ell}\,\mu'\colon\ \mathsf{slhp}\ [\![C]\!]\,(\mathsf{slhp}\ [\![\odot\, e]\!]\,([\neg\mu]))\,(\mu') \\
&\qquad \curlyvee \Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu') \neq f'\Big] \boxplus \mathsf{slhp}\ [\![\odot\, e']\!]\,([\neg\mu])\Big)(f) \\
&= \lambda hh\, \lambda f\colon \boldsymbol{\ell}\,\mu\colon\ hh(\mu) \\
&\qquad \curlyvee \bigcurlywedge_{\mu_1,\mu_2\in\mathbb{A}\colon\, \mu_1\oplus\mu_2=f} \boldsymbol{\ell}\,\mu'\colon\ \mathsf{slhp}\ [\![\odot\, e\,⨟\ C]\!]\,([\neg\mu])\,(\mu') \curlyvee \Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu') \neq \mu_1\Big] \\
&\qquad\qquad\qquad \curlyvee \mathsf{slhp}\ [\![\odot\, e']\!]\,([\neg\mu])\,(\mu_2) \\
&= \lambda hh\, \lambda f\colon \boldsymbol{\ell}\,\mu\colon\ hh(\mu) \\
&\qquad \curlyvee \bigcurlywedge_{\mu_1,\mu_2\in\mathbb{A}\colon\, \mu_1\oplus\mu_2=f} \boldsymbol{\ell}\,\mu'\colon \bigcurlywedge_{\mathsf{sp}\,[\![\odot\, e\,⨟\ C]\!](\mu'')=\mu'} [\neg\mu]\,(\mu'') \curlyvee \Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu') \neq \mu_1\Big] \\
&\qquad\qquad\qquad \curlyvee \bigcurlywedge_{\mathsf{sp}\,[\![\odot\, e']\!](\mu')=\mu_2} [\neg\mu]\,(\mu') \qquad\qquad \text{(by Theorem 5.4.3)} \\
&= \lambda hh\, \lambda f\colon \boldsymbol{\ell}\,\mu\colon\ hh(\mu) \\
&\qquad \curlyvee \bigcurlywedge_{\mu_1,\mu_2\in\mathbb{A}\colon\, \mu_1\oplus\mu_2=f} \Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mathsf{sp}\,[\![\odot\, e\,⨟\ C]\!]\,(\mu)) \neq \mu_1\Big] \curlyvee \big[\mathsf{sp}\,[\![\odot\, e']\!]\,(\mu) \neq \mu_2\big] \\
&= \lambda hh\, \lambda f\colon \boldsymbol{\ell}\,\mu\colon\ hh(\mu) \curlyvee \Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mathsf{sp}\,[\![\odot\, e\,⨟\ C]\!]\,(\mu)) \oplus \mathsf{sp}\,[\![\odot\, e']\!]\,(\mu) \neq f\Big] \\
&= \lambda hh\, \lambda f\colon \boldsymbol{\ell}\,\mu\colon\ hh(\mu) \curlyvee \Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,\big(\mathsf{sp}\,[\![C]\!]\,(\mu\odot e')\big) \oplus \mu\odot e' \neq f\Big] \\
&= \lambda hh\, \lambda f\colon \boldsymbol{\ell}\,\mu\colon\ hh(\mu) \curlyvee \Big[\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu) \neq f\Big] \\
&\qquad\qquad (\mathsf{sp}[\![C^{\langle e,e'\rangle}]\!] \text{ is a fixpoint of } \Psi(X) = \lambda f\colon X(\mathsf{sp}\,[\![C]\!]\,(f\odot[\![e]\!]) \oplus f\odot[\![e']\!]))
\end{aligned}
$$

□

## A.14 Proofs of Section 5.6

### A.14.1 Proof of Healthiness Properties of **whp**, Theorem 5.6.4

Each of the properties is proven individually below.

- Quantitative universal conjunctiveness: Theorem A.14.1;
- Quantitative universal disjunctiveness: Theorem A.14.2;
- $k$-strictness: Corollary A.14.2.1;

- Monotonicity: Corollary A.14.2.2.

**Theorem A.14.1** (Quantitative universal conjunctiveness of $\mathsf{whp}$)**.** *For any set of quantities $S \subseteq \mathbb{AA}$,*

$$\mathsf{whp}\ [\![C]\!] \left(\curlywedge S\right) \quad = \quad \curlywedge_{f \in S} \mathsf{whp}\ [\![C]\!]\,(f\!f) \ .$$

*Proof.*

$$\begin{aligned}
\mathsf{whp}\ [\![C]\!] \left(\curlywedge S\right) \quad &= \quad \lambda\mu\colon (\curlywedge S)(\mathsf{sp}\,[\![C]\!]\,(\mu)) && \text{(by Theorem 5.3.2)}\\
&= \quad \lambda\mu\colon \curlywedge_{f \in S} f\!f(\mathsf{sp}\,[\![C]\!]\,(\mu)) \\
&= \quad \curlywedge_{f \in S} \mathsf{whp}\ [\![C]\!]\,(f\!f) \ . && \text{(by Theorem 5.3.2)}
\end{aligned}$$

□

**Theorem A.14.2** (Quantitative universal disjunctiveness of $\mathsf{whp}$)**.** *For any set of quantities $S \subseteq \mathbb{AA}$,*

$$\mathsf{whp}\ [\![C]\!] \left(\curlyvee S\right) \quad = \quad \curlyvee_{f \in S} \mathsf{whp}\ [\![C]\!]\,(f\!f) \ .$$

*Proof.*

$$\begin{aligned}
\mathsf{whp}\ [\![C]\!] \left(\curlyvee S\right) \quad &= \quad \lambda\mu\colon (\curlyvee S)(\mathsf{sp}\,[\![C]\!]\,(\mu)) && \text{(by Theorem 5.3.2)}\\
&= \quad \lambda\mu\colon \curlyvee_{f \in S} f\!f(\mathsf{sp}\,[\![C]\!]\,(\mu)) \\
&= \quad \curlyvee_{f \in S} \mathsf{whp}\ [\![C]\!]\,(f\!f) \ . && \text{(by Theorem 5.3.2)}
\end{aligned}$$

□

**Corollary A.14.2.1** ($k-$strictness of $\mathsf{whp}$)**.** *For all programs $C$, $\mathsf{whp}\,[\![C]\!]$ is*

*$k$-strict, i.e.*

$$\mathsf{whp}\,[\![C]\!]\,(\lambda f\colon k) \;=\; \lambda f\colon k\ .$$

*Proof.*

$$\begin{aligned}\mathsf{whp}\,[\![C]\!]\,(\lambda f\colon k) &= \lambda\mu\colon (\lambda f\colon k)(\mathsf{sp}\,[\![C]\!]\,(\mu)) && \text{(by Theorem 5.3.2)}\\ &= \lambda f\colon k\ .\end{aligned}$$

□

**Corollary A.14.2.2** (Monotonicity of Quantitative Transformers)**.** *For all programs $C$, $ff, gg \in \mathbb{AA}$, we have*

$$ff \;\preceq\; gg \qquad \text{implies} \qquad \mathsf{whp}\,[\![C]\!]\,(ff) \;\preceq\; \mathsf{whp}\,[\![C]\!]\,(gg)$$

*Proof.*

$$\begin{aligned}\mathsf{whp}\,[\![C]\!]\,(ff) &= \lambda\mu\colon ff(\mathsf{sp}\,[\![C]\!]\,(\mu)) && \text{(by Theorem 5.3.2)}\\ &\preceq \lambda\mu\colon gg(\mathsf{sp}\,[\![C]\!]\,(\mu)) && (ff \;\preceq\; gg)\\ &= \mathsf{whp}\,[\![C]\!]\,(gg) && \text{(by Theorem 5.3.2)}\end{aligned}$$

□

### A.14.2 Proof of Healthiness Properties of **shp**, **slhp**, Theorem 5.6.5

Each of the properties is proven individually below.

- Quantitative universal disjunctiveness: Theorem A.14.3;
- Quantitative universal conjunctiveness: Theorem A.14.4;
- Strictness: Corollary A.14.4.1;
- Costrictness: Corollary A.14.4.2;

- Monotonicity: Corollary A.14.4.3.

**Theorem A.14.3** (Quantitative universal disjunctiveness of $\mathsf{shp}$)**.** *For any set of quantities $S \subseteq \mathbb{AA}$,*

$$\mathsf{shp}\ \llbracket C \rrbracket \left(\bigcurlyvee S\right) \;=\; \bigcurlyvee_{f \in S} \mathsf{shp}\ \llbracket C \rrbracket (ff)\ .$$

*Proof.*

$$\begin{aligned}
\mathsf{shp}\ \llbracket C \rrbracket \left(\bigcurlyvee S\right) &= \lambda f\colon \bigcurlyvee_{\mu\colon\, \mathsf{sp}\, \llbracket C \rrbracket(\mu)=f} (\bigcurlyvee S)(\mu) && \text{(by Theorem 5.4.3)}\\
&= \lambda f\colon \bigcurlyvee_{\mu\colon\, \mathsf{sp}\, \llbracket C \rrbracket(\mu)=f} (\bigcurlyvee_{f \in S} ff)(\mu)\\
&= \lambda f\colon \bigcurlyvee_{f \in S} \bigcurlyvee_{\mu\colon\, \mathsf{sp}\, \llbracket C \rrbracket(\mu)=f} ff(\mu)\\
&= \bigcurlyvee_{f \in S} \mathsf{shp}\ \llbracket C \rrbracket (ff)\ . && \text{(by Theorem 5.4.3)}
\end{aligned}$$

□

**Theorem A.14.4** (Quantitative universal conjunctiveness of $\mathsf{slhp}$)**.** *For any set of quantities $S \subseteq \mathbb{AA}$,*

$$\mathsf{slhp}\ \llbracket C \rrbracket \left(\bigcurlywedge S\right) \;=\; \bigcurlywedge_{f \in S} \mathsf{slhp}\ \llbracket C \rrbracket (ff)\ .$$

*Proof.*

$$\begin{aligned}
\mathsf{slhp}\ \llbracket C \rrbracket \left(\bigcurlywedge S\right) &= \lambda f\colon \bigcurlywedge_{\mu\colon\, \mathsf{sp}\, \llbracket C \rrbracket(\mu)=f} (\bigcurlywedge S)(\mu) && \text{(by Theorem 5.5.3)}\\
&= \lambda f\colon \bigcurlywedge_{\mu\colon\, \mathsf{sp}\, \llbracket C \rrbracket(\mu)=f} (\bigcurlywedge_{f \in S} ff)(\mu)\\
&= \lambda f\colon \bigcurlywedge_{f \in S} \bigcurlywedge_{\mu\colon\, \mathsf{sp}\, \llbracket C \rrbracket(\mu)=f} ff(\mu)\\
&= \bigcurlywedge_{f \in S} \mathsf{slhp}\ \llbracket C \rrbracket (ff)\ . && \text{(by Theorem 5.5.3)}
\end{aligned}$$

□

**Corollary A.14.4.1** (Strictness of $\mathsf{shp}$)**.** *For all programs $C$, $\mathsf{shp}\,[\![C]\!]$ is strict, i.e.*

$$\mathsf{shp}\;[\![C]\!]\,(\lambda f\colon -\infty) \;=\; \lambda f\colon -\infty\;.$$

*Proof.*

$$\begin{aligned}\mathsf{shp}\;[\![C]\!]\,(\lambda f\colon -\infty) \;&=\; \lambda \nu\colon \curlyvee_{\mu\colon\, \mathsf{sp}\,[\![C]\!](\mu)=f} (\lambda f\colon -\infty)(\mu) \quad \text{(by Theorem 5.4.3)}\\ &=\; \lambda f\colon -\infty\;.\end{aligned}$$

□

**Corollary A.14.4.2** (Co-strictness of $\mathsf{slhp}$)**.** *For all programs $C$, $\mathsf{slhp}\,[\![C]\!]$ is co-strict, i.e.*

$$\mathsf{slhp}\;[\![C]\!]\,(\lambda f\colon +\infty) \;=\; \lambda f\colon +\infty\;.$$

*Proof.*

$$\begin{aligned}\mathsf{slhp}\;[\![C]\!]\,(\lambda f\colon +\infty) \;&=\; \lambda \nu\colon \curlywedge_{\mu\colon\, \mathsf{sp}\,[\![C]\!](\mu)=f} (\lambda f\colon +\infty)(\mu) \quad \text{(by Theorem 5.5.3)}\\ &=\; \lambda f\colon +\infty\;.\end{aligned}$$

□

**Corollary A.14.4.3** (Monotonicity of Quantitative Transformers)**.** *For all programs $C$, $\mathbb{f}, \mathbb{g} \in \mathbb{A}$, we have*

$$\mathbb{f} \;\preceq\; \mathbb{g} \qquad \text{implies} \qquad \mathsf{shp}\;[\![C]\!]\,(\mathbb{f}) \;\preceq\; \mathsf{shp}\;[\![C]\!]\,(\mathbb{g})$$

*and*

$$ff \preceq gg \qquad \text{implies} \qquad \mathsf{slhp}\,[\![C]\!]\,(ff) \preceq \mathsf{slhp}\,[\![C]\!]\,(gg)$$

*Proof.*

$$\begin{aligned}
\mathsf{shp}\,[\![C]\!]\,(ff) &= \lambda\nu\colon \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C]\!](\mu)=f} ff(\mu) && \text{(by Theorem 5.5.3)}\\
&\preceq \lambda\nu\colon \bigcurlyvee_{\mu\colon \mathsf{sp}\,[\![C]\!](\mu)=f} gg(\mu) && (ff \preceq gg)\\
&= \mathsf{shp}\,[\![C]\!]\,(gg) && \text{(by Theorem 5.4.3)}
\end{aligned}$$

$$\begin{aligned}
\mathsf{slhp}\,[\![C]\!]\,(ff) &= \lambda\nu\colon \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![C]\!](\mu)=f} ff(\mu) && \text{(by Theorem 5.5.3)}\\
&\preceq \lambda\nu\colon \bigcurlywedge_{\mu\colon \mathsf{sp}\,[\![C]\!](\mu)=f} gg(\mu) && (ff \preceq gg)\\
&= \mathsf{slhp}\,[\![C]\!]\,(gg) && \text{(by Theorem 5.5.3)}
\end{aligned}$$

□

### A.14.3 Proof of Linearity, Theorem 5.6.6

**Theorem 5.6.6** (Linearity)**.** *For all programs $C$,* $\mathsf{whp}\,[\![C]\!]$ *is linear, i.e. for all* $ff, gg \in \mathbb{AA}$ *and* non-negative *constants* $r \in \mathbb{R}_{\geq 0}$,

$$\mathsf{whp}\,[\![C]\!]\,(r \cdot ff + gg) = r \cdot \mathsf{whp}\,[\![C]\!]\,(ff) + \mathsf{whp}\,[\![C]\!]\,(gg) \ .$$

◁

*Proof.*

$$\begin{aligned}
&\mathsf{whp}\,[\![C]\!]\,(r \cdot ff + gg)\\
&\quad= \lambda\mu\colon (r \cdot ff + gg)(\mathsf{sp}\,[\![C]\!]\,(\mu)) && \text{(by Theorem 5.3.2)}\\
&\quad= \lambda\mu\colon (r \cdot ff)(\mathsf{sp}\,[\![C]\!]\,(\mu)) + gg(\mathsf{sp}\,[\![C]\!]\,(\mu))
\end{aligned}$$

$$
\begin{aligned}
&= \lambda\mu\colon r \cdot ff(\mathsf{sp}\,[\![C]\!]\,(\mu)) + gg(\mathsf{sp}\,[\![C]\!]\,(\mu)) \\
&= r \cdot \mathsf{whp}\,[\![C]\!]\,(ff) + \mathsf{whp}\,[\![C]\!]\,(gg)\ . && \text{(by Theorem 5.3.2)}
\end{aligned}
$$

□

### A.14.4 Proof of Multiplicativity, Theorem 5.6.8

**Theorem 5.6.8** (Multiplicativity). *For all programs $C$,* $\mathsf{whp}\,[\![C]\!]$ *is multiplicative, i.e. for all $ff, gg \in \mathbb{AA}$ and* non-negative *constants $r \in \mathbb{R}_{\geq 0}$,*

$$
\mathsf{whp}\,[\![C]\!]\,(r \cdot ff \cdot gg) \ = \ r \cdot \mathsf{whp}\,[\![C]\!]\,(ff) \cdot \mathsf{whp}\,[\![C]\!]\,(gg)\ .
$$

◁

*Proof.*

$$
\begin{aligned}
&\mathsf{whp}\,[\![C]\!]\,(r \cdot ff \cdot gg) \\
&\quad= \lambda\mu\colon (r \cdot ff \cdot gg)(\mathsf{sp}\,[\![C]\!]\,(\mu)) && \text{(by Theorem 5.3.2)} \\
&\quad= \lambda\mu\colon r \cdot ff(\mathsf{sp}\,[\![C]\!]\,(\mu)) \cdot gg(\mathsf{sp}\,[\![C]\!]\,(\mu)) \\
&\quad= r \cdot \mathsf{whp}\,[\![C]\!]\,(ff) \cdot \mathsf{whp}\,[\![C]\!]\,(gg)\ . && \text{(by Theorem 5.3.2)}
\end{aligned}
$$

□

### A.14.5 Proof of Liberal-Non-liberal Duality, Theorem 5.6.2

**Theorem 5.6.2** (Liberal–Non-liberal Duality, $\mathsf{whp}$). *For any program $C$ and any $k$-bounded hyperquantity $ff$, we have* $\mathsf{whp}\,[\![C]\!]\,(ff) \ = \ k - \mathsf{whp}\,[\![C]\!]\,(k - ff)$.

*Proof.*

$$
\begin{aligned}
\mathsf{whp}\,[\![C]\!]\,(ff) &= \lambda\mu\colon ff(\mathsf{sp}\,[\![C]\!]\,(\mu)) f(\tau) && \text{(by Theorem 5.3.2)} \\
&= k - \lambda\mu\colon k - ff(\mathsf{sp}\,[\![C]\!]\,(\mu)) \\
&= k - \mathsf{whp}\,[\![C]\!]\,(k - ff)\ .
\end{aligned}
$$

□

## A.14.6 Proof of Linearity, Theorem 5.6.7

**Theorem 5.6.7** (Linearity). *For all programs $C$, $\mathsf{shp}\,[\![C]\!]$ is sublinear and $\mathsf{slp}[\![C]\!]$ is superlinear, i.e. for all $ff, gg \in \mathbb{AA}$ and constants $r \in U$,*

$$\begin{aligned}\mathsf{shp}\;[\![C]\!]\,(r\cdot ff + gg) &\preceq r\cdot \mathsf{shp}\;[\![C]\!]\,(ff) + \mathsf{shp}\;[\![C]\!]\,(gg)\;, \quad \text{and}\\ r\cdot \mathsf{slhp}\;[\![C]\!]\,(ff) + \mathsf{slhp}\;[\![C]\!]\,(gg) &\preceq \mathsf{slhp}\;[\![C]\!]\,(r\cdot ff + gg)\;.\end{aligned}$$

◁

*Proof.* For $\mathsf{shp}$ we have:

$$\begin{aligned}&\mathsf{shp}\;[\![C]\!]\,(r\cdot ff + gg)\\ =\;& \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot ff + gg)(\mu) && \text{(by Theorem 5.4.3)}\\ =\;& \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot ff(\mu) + gg(\mu))\\ \preceq\;& \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot ff)(\mu) + \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu)\\ =\;& \lambda\nu\colon r\cdot \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} ff(\mu) + \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu)\\ & && (\sup(r\cdot A) = r\cdot \sup A \text{ for } A\subseteq\mathbb{R}, r\in\mathbb{R}_{\geq 0})\\ =\;& r\cdot\lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} ff(\mu) + \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu)\\ =\;& r\cdot \mathsf{shp}\;[\![C]\!]\,(ff) + \mathsf{shp}\;[\![C]\!]\,(gg)\;. && \text{(by Theorem 5.4.3)}\end{aligned}$$

For $\mathsf{slhp}$ we have:

$$\begin{aligned}&r\cdot \mathsf{slhp}\;[\![C]\!]\,(ff) + \mathsf{slhp}\;[\![C]\!]\,(gg)\\ =\;& r\cdot\lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} ff(\mu) + \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu) && \text{(by Theorem 5.5.3)}\\ =\;& \lambda\nu\colon r\cdot \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} ff(\mu) + \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu)\end{aligned}$$

$$
\begin{aligned}
&= \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot ff)(\mu) + \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu) \\
&\qquad\qquad\qquad\qquad (\inf(r\cdot A) = r\cdot \inf A \text{ for } A\subseteq\mathbb{R}, r\in\mathbb{R}_{\geq 0}) \\
&\preceq \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} \big((r\cdot ff)(\mu) + gg(\mu)\big) \\
&= \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot ff + gg)(\mu) \\
&= \mathsf{slhp}\ [\![C]\!]\,(r\cdot ff + gg)\ . \qquad\qquad \text{(by Theorem 5.5.3)}
\end{aligned}
$$

□

## A.14.7 Proof of Multiplicativity, Theorem 5.6.9

**Theorem 5.6.9** (Multiplicativity)**.** *For all programs $C$,* $\mathsf{shp}\,[\![C]\!]$ *is submultiplicative and* $\mathsf{slhp}\,[\![C]\!]$ *is supermultiplicative, i.e. for all $ff, gg \in \mathbb{AA}$ and* non-negative *constants $r\in\mathbb{R}_{\geq 0}$,*

$$
\begin{aligned}
\mathsf{shp}\ [\![C]\!]\,(r\cdot ff\cdot gg) &\preceq r\cdot \mathsf{shp}\ [\![C]\!]\,(ff)\cdot \mathsf{shp}\ [\![C]\!]\,(gg)\ , \quad \text{and} \\
r\cdot \mathsf{slhp}\ [\![C]\!]\,(ff)\cdot \mathsf{slhp}\ [\![C]\!]\,(gg) &\preceq \mathsf{slhp}\ [\![C]\!]\,(r\cdot ff\cdot gg)\ .
\end{aligned}
$$

◁

*Proof.* For shp we have:

$$
\begin{aligned}
&\mathsf{shp}\ [\![C]\!]\,(r\cdot ff\cdot gg) \\
&= \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot ff\cdot gg)(\mu) \qquad \text{(by Theorem 5.4.3)} \\
&= \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} \big((r\cdot ff)(\mu)\cdot gg(\mu)\big) \\
&\preceq \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot ff)(\mu)\cdot \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu) \\
&= \lambda\nu\colon r\cdot \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} ff(\mu)\cdot \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu) \\
&\qquad\qquad\qquad\qquad (\sup(r\cdot A) = r\cdot \sup A \text{ for } A\subseteq\mathbb{R}, r\in\mathbb{R}_{\geq 0}) \\
&= r\cdot \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} ff(\mu)\cdot \lambda\nu\colon \sup_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} gg(\mu) \\
&= r\cdot \mathsf{shp}\ [\![C]\!]\,(ff)\cdot \mathsf{shp}\ [\![C]\!]\,(gg)\ . \qquad\qquad \text{(by Theorem 5.4.3)}
\end{aligned}
$$

For $\mathsf{slhp}$ we have:

$$
\begin{aligned}
& r \cdot \mathsf{slhp}\,[\![C]\!]\,(f) \cdot \mathsf{slhp}\,[\![C]\!]\,(g) \\
={} & r \cdot \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} f\!\!f(\mu) \cdot \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} g\!\!g(\mu) && \text{(by Theorem 5.5.3)} \\
={} & \lambda\nu\colon r \cdot \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} f\!\!f(\mu) \cdot \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} g\!\!g(\mu) \\
={} & \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot f\!\!f)(\mu) \cdot \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} g\!\!g(\mu) \\
& && (\inf(r\cdot A) = r\cdot \inf A \text{ for } A\subseteq\mathbb{R}, r\in\mathbb{R}_{\geq 0}) \\
\preceq{} & \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} \big((r\cdot f\!\!f)(\mu)\cdot g\!\!g(\mu)\big) \\
={} & \lambda\nu\colon \inf_{\mu\in\mathbb{A},\nu=\mathsf{sp}\,[\![C]\!](\mu)} (r\cdot f\!\!f\cdot g\!\!g)(\mu) \\
={} & \mathsf{slhp}\,[\![C]\!]\,(r\cdot f\!\!f\cdot g\!\!g)\ . && \text{(by Theorem 5.5.3)}
\end{aligned}
$$

□

### A.14.8 Proof of Liberal-Non-liberal Duality, Theorem 5.6.3

**Theorem 5.6.3** (Liberal–Non-liberal Duality, $\mathsf{shp}$ and $\mathsf{slhp}$)**.** *For any program $C$ and hyperquantity $f\!\!f$, we have*

$$\mathsf{shp}\,[\![C]\!]\,(f\!\!f) \;=\; -\,\mathsf{slhp}\,[\![C]\!]\,(-f\!\!f)\,.$$

*Proof.*

$$
\begin{aligned}
\mathsf{shp}\,[\![C]\!]\,(f\!\!f) &= \lambda\nu\colon \sup_{\mu\in\Sigma,\nu=\mathsf{sp}\,[\![C]\!](\mu)} f\!\!f(\mu) && \text{(by Theorem 5.4.3)} \\
&= \lambda\nu\colon -\inf_{\mu\in\Sigma,\nu=\mathsf{sp}\,[\![C]\!](\mu)} -f\!\!f(\mu) && (\sup A = -\inf(-A)) \\
&= -\,\mathsf{slp}[\![C]\!]\,(-f\!\!f)\ . && \text{(by Theorem 5.5.3)}
\end{aligned}
$$

□

### A.14.9 Proof of Galois Connection between **whp**, **shp** and **slhp**, Theorem 5.6.1

**Theorem 5.6.1** (Hyper Galois Connection)**.** *For all $C \in \mathsf{Reg}$ and $\mathscr{g}, \mathscr{f\!f} \in \mathbb{AA}$:*

$$\begin{aligned} \mathscr{g} \;\preceq\; \mathsf{whp}\,[\![C]\!]\,(\mathscr{f\!f}) \quad &\text{iff} \quad \mathsf{shp}\,[\![C]\!]\,(\mathscr{g}) \;\preceq\; \mathscr{f\!f}\ , \\ \mathsf{whp}\,[\![C]\!]\,(\mathscr{f\!f}) \;\preceq\; \mathscr{g} \quad &\text{iff} \quad \mathscr{f\!f} \;\preceq\; \mathsf{slhp}\,[\![C]\!]\,(\mathscr{g})\ . \end{aligned}$$

*Proof.*

$$\begin{aligned} \mathscr{g} \;\preceq\; \mathsf{whp}\,[\![C]\!]\,(\mathscr{f\!f}) &\iff \forall \mu \in \mathbb{A}\colon \mathscr{g}(\mu) \leq \mathsf{whp}\,[\![C]\!]\,(\mathscr{f\!f})\,(\mu) \\ &\iff \forall \mu \in \mathbb{A}\colon \mathscr{g}(\mu) \leq \mathscr{f\!f}(\mathsf{sp}\,[\![C]\!]\,(\mu)) \quad \text{(by Theorem 5.3.2)} \\ &\iff \forall \mu, \nu \in \mathbb{A}\colon \nu = \mathsf{sp}\,[\![C]\!]\,(\mu) : \mathscr{g}(\mu) \leq \mathscr{f\!f}(\nu) \\ &\iff \forall \nu \in \mathbb{A}\colon \sup_{\mu \in \mathbb{A}, \nu = \mathsf{sp}\,[\![C]\!](\mu)} \mathscr{g}(\mu) \leq \mathscr{f\!f}(\nu) \\ &\iff \forall \nu \in \mathbb{A}\colon \mathsf{shp}\,[\![C]\!]\,(\mathscr{g})\,(\nu) \leq \mathscr{f\!f}(\nu) \\ &\qquad\qquad \text{(by Theorem 5.4.3)} \\ &\iff \mathsf{shp}\,[\![C]\!]\,(\mathscr{g}) \;\preceq\; \mathscr{f\!f}\ . \end{aligned}$$

$$\begin{aligned} \mathsf{whp}\,[\![C]\!]\,(\mathscr{f\!f}) \;\preceq\; \mathscr{g} &\iff \forall \mu \in \mathbb{A}\colon \mathsf{whp}\,[\![C]\!]\,(\mathscr{f\!f})\,(\mu) \leq \mathscr{g}(\mu) \\ &\iff \forall \mu \in \mathbb{A}\colon \mathscr{f\!f}(\mathsf{sp}\,[\![C]\!]\,(\mu)) \leq \mathscr{g}(\mu) \quad \text{(by Theorem 5.3.2)} \\ &\iff \forall \mu, \nu \in \mathbb{A}\colon \nu = \mathsf{sp}\,[\![C]\!]\,(\mu) : \mathscr{f\!f}(\nu) \leq \mathscr{g}(\mu) \\ &\iff \forall \nu \in \mathbb{A}\colon \mathscr{f\!f}(\nu) \leq \inf_{\mu \in \mathbb{A}, \nu = \mathsf{sp}\,[\![C]\!](\mu)} \mathscr{g}(\mu) \\ &\iff \forall \nu \in \mathbb{A}\colon \mathscr{f\!f}(\nu) \leq \mathsf{slhp}\,[\![C]\!]\,(\mathscr{g})\,(\nu) \\ &\qquad\qquad \text{(by Theorem 5.5.3)} \\ &\iff \mathscr{f\!f} \;\preceq\; \mathsf{slhp}\,[\![C]\!]\,(\mathscr{g})\ . \end{aligned}$$

□

### A.14.10 Proof of whp rules for additive hyperquantities, Theorem 5.6.10

**Theorem 5.6.10** (Weakest Hyper Pre for Additive Hyperquantities)**.** *For additive hyperquantities* $\mathit{ff} \in \mathbb{AA}$*, the simpler rules in* Table 5.4 *are valid.*

*Proof.* We prove Theorem 5.3.2 by induction on the structure of $C$. For the induction base, we have the atomic statement:

**The assignment $x \coloneqq e$:**

We have

$$\begin{aligned} \mathsf{whp}\,[\![x \coloneqq e]\!]\,(\mathit{ff})\,(\mu) &= \bigoplus_{\alpha} \mathit{ff}([x = e\,[x/\alpha]] \odot \mu\,[x/\alpha]) \\ &= \mathit{ff}(\bigoplus_{\alpha} [x = e\,[x/\alpha]] \odot \mu\,[x/\alpha]) \\ &= \mathit{ff}(\mathsf{sp}\,[\![x \coloneqq e]\!]\,(\mu))\ . \end{aligned}$$

**The nondeterministic assignment $x \coloneqq$ `nondet()`:**

We have

$$\begin{aligned} \mathsf{whp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\mathit{ff})\,(\mu) &= \mathit{ff}(\bigoplus_{\alpha} \mu\,[x/\alpha]) \\ &= \mathit{ff}(\mathsf{sp}\,[\![x \coloneqq \texttt{nondet()}]\!]\,(\mu))\ . \end{aligned}$$

**The weighting $\odot\, a$:**

We have

$$\begin{aligned} \mathsf{whp}\,[\![\odot\, w]\!]\,(\mathit{ff})\,(\mu) &= (\mathit{ff} \odot w)(\mu) \\ &= \mathit{ff}(\mu \odot w) \\ &= \mathit{ff}(\mathsf{sp}\,[\![\odot\, w]\!]\,(\mu))\ . \end{aligned}$$

This concludes the proof for the atomic statements.

**Induction Hypothesis:**

For arbitrary but fixed programs $C$, $C_1$, $C_2$, we proceed with the inductive step on the composite statements.

**The sequential composition $C_1 \fatsemi C_2$:**

We have

$$\begin{aligned}
\mathsf{whp}\;\llbracket C_1 \fatsemi C_2 \rrbracket\,(\mathit{ff})\,(\mu) &= \mathsf{whp}\;\llbracket C_1 \rrbracket\,(\mathsf{whp}\;\llbracket C_2 \rrbracket\,(\mathit{ff}))\,(\mu) \\
&= \mathsf{whp}\;\llbracket C_2 \rrbracket\,(\mathit{ff})\,(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu)) && \text{(by I.H. on } C_1\text{)} \\
&= \mathit{ff}(\mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu))) && \text{(by I.H. on } C_2\text{)} \\
&= \mathit{ff}(\mathsf{sp}\,\llbracket C_1 \fatsemi C_2 \rrbracket\,(\mu))
\end{aligned}$$

**The nondeterministic choice $\{\, C_1 \,\} \,\square\, \{\, C_2 \,\}$:**

We have

$$\begin{aligned}
& \mathsf{whp}\;\llbracket \{\, C_1 \,\} \,\square\, \{\, C_2 \,\} \rrbracket\,(\mathit{ff})\,(\mu) \\
={}& \mathsf{whp}\;\llbracket C_1 \rrbracket\,(\mathit{ff})\,(\mu) \oplus \mathsf{whp}\;\llbracket C_2 \rrbracket\,(\mathit{ff})\,(\mu) \\
={}& \mathit{ff}(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu)) \oplus \mathit{ff}(\mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mu)) && \text{(by I.H. on } C_1, C_2\text{)} \\
={}& \mathit{ff}(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu) \oplus \mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mu)) && \text{(by Definition 5.6.1)} \\
={}& \bigoplus_{\nu_1,\nu_2} \mathit{ff}(\nu_1 \oplus \nu_2) \odot [\nu_1]\,(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu)) \odot [\nu_2]\,(\mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mu)) \\
={}& \mathit{ff}(\mathsf{sp}\,\llbracket C_1 \rrbracket\,(\mu) \oplus \mathsf{sp}\,\llbracket C_2 \rrbracket\,(\mu)) \\
={}& \mathit{ff}(\mathsf{sp}\,\llbracket \{\, C_1 \,\} \,\square\, \{\, C_2 \,\} \rrbracket\,(\mu))\ .
\end{aligned}$$

**The Iteration $C^{\langle e,e' \rangle}$:**

Let

$$\Phi_{\mathit{ff}}(\mathsf{trnsf}) \;=\; \mathit{ff} \odot \llbracket e' \rrbracket \mathbin{+\!\!\!\!*} \mathsf{whp}\;\llbracket C \rrbracket\,(\mathsf{trnsf}) \odot \llbracket e \rrbracket$$

be the $\mathsf{whp}$-characteristic function of the iteration $C^{\langle e,e'\rangle}$ with respect to any hyperquantity $\mathit{ff}$. Let

$$\Psi(\mathsf{trnsf}) \;=\; \lambda f\colon \mathsf{trnsf}(\mathsf{sp}\,[\![C]\!]\,(f \odot [\![e]\!])) \oplus f \odot [\![e']\!]$$

be the $\mathsf{sp}$-characteristic function of the iteration $C^{\langle e,e'\rangle}$.

We first prove by induction on $n$ that:

$$\Phi^n_{\mathit{ff}}(\lambda g\colon \mathit{ff}(\mathbb{0}))(\mu) \;=\; \mathit{ff}(\Psi^n(\lambda g\colon \mathbb{0})(\mu)) \tag{A.14}$$

For the induction base $n = 0$, consider the following:

$$\begin{aligned}\Phi^0_{\mathit{ff}}(\lambda g\colon \mathit{ff}(\mathbb{0}))(\mu) &= \mathit{ff}(\mathbb{0})\\ &= \mathit{ff}(\Psi^0(\lambda g\colon \mathbb{0})(\mu))\ .\end{aligned}$$

As induction hypothesis, we have for arbitrary but fixed $n$ and all $\mu$

$$\Phi^n_{\mathit{ff}}(\lambda g\colon \mathit{ff}(\mathbb{0}))(\mu) \;=\; \mathit{ff}(\Psi^n(\lambda g\colon \mathbb{0})(\mu))$$

For the induction step $n \longrightarrow n+1$, consider the following:

$$\begin{aligned}\Phi^{n+1}_{\mathit{ff}}(\lambda g\colon \mathit{ff}(\mathbb{0}))(\mu) &= \Phi_{\mathit{ff}}(\Phi^n_{\mathit{ff}}(\lambda g\colon \mathit{ff}(\mathbb{0})))(\mu)\\ &= (\mathit{ff} \odot [\![e']\!] \ast \mathsf{whp}\ [\![C]\!]\left(\Phi^n_{\mathit{ff}}(\lambda g\colon \mathit{ff}(\mathbb{0}))\right) \odot [\![e]\!])(\mu)\\ &= \mathit{ff}(\mu \odot [\![e']\!]) \ast \mathsf{whp}\ [\![C]\!]\left(\Phi^n_{\mathit{ff}}(\lambda g\colon \mathit{ff}(\mathbb{0}))\right)(\mu \odot [\![e]\!])\\ &= \mathit{ff}(\mu \odot [\![e']\!]) \ast \Phi^n_{\mathit{ff}}(\lambda g\colon \mathit{ff}(\mathbb{0}))(\mathsf{sp}\,[\![C]\!]\,(\mu \odot [\![e]\!]))\\ &\qquad\text{(by I.H. on } C\text{)}\\ &= \mathit{ff}(\mu \odot [\![e']\!]) \ast \mathit{ff}(\Psi^n(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(\mu \odot [\![e]\!])))\\ &\qquad\text{(by I.H. on } n\text{)}\\ &= \mathit{ff}(\mu \odot [\![e']\!] \oplus \Psi^n(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(\mu \odot [\![e]\!])))\\ &\qquad\text{(by additivity)}\end{aligned}$$

$$= \mathit{ff}(\Psi^n(\lambda g\colon \mathbb{0})(\mathsf{sp}\,[\![C]\!]\,(\mu \odot [\![e]\!])) \oplus \mu \odot [\![e']\!]) \qquad \text{(by commutativity)}$$

$$= \mathit{ff}(\Psi^{n+1}(\lambda g\colon \mathbb{0})(\mu))\ .$$

This concludes the induction on $n$. Now we have:

$$\begin{aligned}
\mathsf{whp}\ \left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\mathit{ff})\,(\mu) &= \big(\mathsf{lfp\ trnsf}\colon \Phi_{\mathit{ff}}(\mathsf{trnsf})\big)(\mu) \\
&= \sup_{n\in\mathbb{N}}\ \Phi^n_{\mathit{ff}}(\mathit{ff}(\mathbb{0}))(\mu) \quad \text{(by Kleene's fixpoint theorem)} \\
&= \sup_{n\in\mathbb{N}}\ \mathit{ff}(\Psi^n(\lambda g\colon \mathbb{0})(\mu)) \qquad \text{(by Equation (A.14))} \\
&= \mathit{ff}(\sup_{n\in\mathbb{N}}\ \Psi^n(\lambda g\colon \mathbb{0})(\mu)) \\
&= \mathit{ff}(\mathsf{sp}\,[\![C^{\langle e,e'\rangle}]\!]\,(\mu))\ . \quad \text{(by Kleene's fixpoint theorem)}
\end{aligned}$$

□

## A.15 Proofs of Section 5.7

### A.15.1 Proof of Subsumption of HHL, Theorem 5.7.1

**Theorem 5.7.1** (Subsumption of HHL). *For hyperpredicates $\psi$, $\phi$ and non-probabilistic program $C$:*

$$\begin{aligned}
&\models_{\mathrm{hh}} \{\, \psi \,\}\ C\ \{\, \phi \,\} \\
&\quad \text{iff}\quad \mathsf{supp}\,([\psi]) \subseteq \mathsf{supp}\,(\mathsf{whp}\ [\![C]\!]\,([\phi])) \\
&\quad \text{iff}\quad \mathsf{supp}\,(\mathsf{shp}\ [\![C]\!]\,([\phi])) \subseteq \mathsf{supp}\,([\psi]) \\
&\quad \text{iff}\quad \mathsf{supp}\,([\neg\psi]) \subseteq \mathsf{supp}\,(\mathsf{slhp}\ [\![C]\!]\,([\neg\phi]))
\end{aligned}$$

*Proof.*

$$\models_{\mathrm{hh}} \{\, \psi \,\}\ C\ \{\, \phi \,\} \quad \text{iff} \quad \forall S \in \mathcal{P}(\Sigma)\colon S \in \psi \implies [\![C]\!](S) \in \phi$$

$$
\begin{aligned}
&\text{iff} \quad \forall S \in \mathcal{P}(\Sigma)\colon\ S \in \not\psi \implies \mathsf{supp}\left(\mathsf{sp}\,[\![C]\!]\,([S])\right) \in \not\phi \\
&\text{iff} \quad \forall S \in \mathcal{P}(\Sigma)\colon\ [\not\psi]\,([S]) \leq [\not\phi]\,(\mathsf{sp}\,[\![C]\!]\,([S])) \\
&\text{iff} \quad \forall S \in \mathcal{P}(\Sigma)\colon\ [\not\psi]\,([S]) \leq \mathsf{whp}\ [\![C]\!]\,([\not\phi])\,([S]) \\
&\text{iff} \quad \forall \mu \in \mathbb{A}\colon\ [\not\psi]\,(\mu) \leq \mathsf{whp}\ [\![C]\!]\,([\not\phi])\,(\mu) \\
&\text{iff} \quad \mathsf{supp}\,([\not\psi]) \implies \mathsf{supp}\,(\mathsf{whp}\ [\![C]\!]\,([\not\phi]))
\end{aligned}
$$

□

### A.15.2 Proof of Subsumption of Quantitative wp, wlp for Nondeterministic Programs, Theorem 5.7.2

**Theorem 5.7.2** (Subsumption of Quantitative wp, wlp for Nondeterministic Programs [17])**.** *Let* $\mathcal{A} = \langle \mathbb{R}^{\pm\infty}, \max, \min, \mathbb{0}, \mathbb{1}\rangle$*. For any quantities* $g, f$ *and any program* $C$ *satisfying the syntax of* [17, Section 2]*:*

$$
\begin{aligned}
\mathsf{whp}\ [\![C]\!]\left(\curlywedge [f]\right)(\mathbf{1}_\sigma) \ &=\ \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)\ , \\
\mathsf{whp}\ [\![C]\!]\left(\curlyvee [f]\right)(\mathbf{1}_\sigma) \ &=\ \mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)\ .
\end{aligned}
$$

*Proof.*

$$
\begin{aligned}
\mathsf{whp}\ [\![C]\!]\left(\curlyvee [f]\right)(\mathbf{1}_\sigma) \ &=\ \curlyvee [f](\mathsf{sp}\,[\![C]\!]\,(\mathbf{1}_\sigma)) \\
&=\ \curlyvee_{\tau\colon \mathsf{sp}\,[\![C]\!](\mathbf{1}_\sigma)(\tau)>0} f(\tau) \\
&=\ \mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{whp}\ [\![C]\!]\left(\curlywedge [f]\right)(\mathbf{1}_\sigma) \ &=\ \curlywedge [f](\mathsf{sp}\,[\![C]\!]\,(\mathbf{1}_\sigma)) \\
&=\ \curlywedge_{\tau\colon \mathsf{sp}\,[\![C]\!](\mathbf{1}_\sigma)(\tau)>0} f(\tau) \\
&=\ \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)
\end{aligned}
$$

□

### A.15.3 Proof of Subsumption of Quantitative wp, wlp for Probabilistic Programs, Theorem 5.7.3

**Theorem 5.7.3** (Subsumption of Quantitative $\mathsf{wp}$, $\mathsf{wlp}$ for Probabilistic Programs [16])**.** *Let $\mathsf{Prob} = \langle [0,1], +, \cdot, 0, 1\rangle$. For any quantities $g, f$ and any <u>non-non</u>deterministic (possibly probabilistic) program $C$:*

$$\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[f])\,(\mathbf{1}_\sigma) = \mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)$$
$$\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[f] + 1 - \mathbb{E}[1])\,(\mathbf{1}_\sigma) = \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)\ .$$

*Proof.*

$$\begin{aligned}\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[f])\,(\mathbf{1}_\sigma) &= \mathbb{E}[f](\mathsf{sp}\,[\![C]\!]\,(\mathbf{1}_\sigma)) \\ &= \mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma)\end{aligned}$$

$$\begin{aligned}\mathsf{whp}\ [\![C]\!]\,(\mathbb{E}[f] + 1 - \mathbb{E}[1])\,(\mathbf{1}_\sigma) &= (\mathbb{E}[f] + 1 - \mathbb{E}[1])(\mathsf{sp}\,[\![C]\!]\,(\mathbf{1}_\sigma)) \\ &= \mathsf{wp}\,[\![C]\!]\,(f)\,(\sigma) + 1 - \mathsf{wp}\,[\![C]\!]\,(1)\,(\sigma) \\ &= \mathsf{wlp}[\![C]\!]\,(f)\,(\sigma)\end{aligned}$$

□

# Bibliography

[1] Edsger W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976. ISBN 013215871X.

[2] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 1969. ISSN 0001-0782.

[3] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, page 238–252, New York, NY, USA, 1977. Association for Computing Machinery. ISBN 9781450373500. doi: 10.1145/512950.512973. URL `https://doi.org/10.1145/512950.512973`.

[4] Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. *J. ACM*, 58(6), dec 2011. ISSN 0004-5411. doi: 10.1145/2049697.2049700. URL `https://doi.org/10.1145/2049697.2049700`.

[5] 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, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44802-0.

[6] Peter W. O'Hearn. Incorrectness logic. *Proc. ACM Program. Lang.*, 4

(POPL), December 2020. doi: 10.1145/3371078. URL `https://doi.org/10.1145/3371078`.

[7] Quang Loc Le, Azalea Raad, Jules Villard, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. Finding real bugs in big programs with incorrectness logic. *Proc. ACM Program. Lang.*, 6(OOPSLA1), apr 2022. doi: 10.1145/3527325. URL `https://doi.org/10.1145/3527325`.

[8] Peter W. O'Hearn. Continuous reasoning: Scaling the impact of formal methods. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 13–25, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355834. doi: 10.1145/3209108.3209109. URL `https://doi.org/10.1145/3209108.3209109`.

[9] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. Scaling static analyses at facebook. *Commun. ACM*, 62(8): 62–70, jul 2019. ISSN 0001-0782. doi: 10.1145/3338112. URL `https://doi.org/10.1145/3338112`.

[10] Ke Mao, Cons Åhs, Sopot Cela, Dino Distefano, Nick Gardner, Radu Grigore, Per Gustafsson, Ákos Hajdu, Timotej Kapus, Matteo Marescotti, Gabriela Cunha Sampaio, and Thibault Suzanne. Privacycat: Privacy-aware code analysis at scale. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '24, page 106–117, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705014. doi: 10.1145/3639477.3639742. URL `https://doi.org/10.1145/3639477.3639742`.

[11] Radu Grigore, Dino Distefano, and Nikos Tzevelekos. Automatic compositional checking of multi-object typestate properties of software. In Nils Jansen, Sebastian Junges, Benjamin Lucien Kaminski, Christoph Matheja, Thomas Noll, Tim Quatmann, Mariëlle Stoelinga, and Matthias

Volk, editors, *Principles of Verification: Cycling the Probabilistic Landscape: Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday, Part I*, pages 3–40. Springer Nature Switzerland, Cham, 2025. ISBN 978-3-031-75783-9. doi: 10.1007/978-3-031-75783-9_1. URL `https://doi.org/10.1007/978-3-031-75783-9_1`.

[12] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. Lessons from building static analysis tools at google. *Commun. ACM*, 61(4):58–66, mar 2018. ISSN 0001-0782. doi: 10.1145/3188720. URL `https://doi.org/10.1145/3188720`.

[13] Dexter Kozen. A probabilistic PDL. *Journal of Computer and System Sciences*, 30(2):162–178, 1985.

[14] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Monographs in Computer Science. Springer, 2005.

[15] Marcel Hark, Benjamin Lucien Kaminski, Jürgen Giesl, and Joost-Pieter Katoen. Aiming low is harder: induction for lower bounds in probabilistic program verification. *Proc. ACM Program. Lang.*, 4(POPL), dec 2019. doi: 10.1145/3371105. URL `https://doi.org/10.1145/3371105`.

[16] Benjamin Lucien Kaminski. *Advanced Weakest Precondition Calculi for Probabilistic Programs.* PhD thesis, RWTH Aachen University, Germany, 2019.

[17] Linpeng Zhang and Benjamin Lucien Kaminski. Quantitative Strongest Post: A Calculus for Reasoning about the Flow of Quantitative Information. *Proc. ACM Program. Lang.*, 6(OOPSLA1):1–29, 2022.

[18] Linpeng Zhang, Noam Zilberstein, Benjamin Lucien Kaminski, and Alexandra Silva. Quantitative weakest hyper pre: Unifying correctness and incorrectness hyperproperties via predicate transformers. *Proc. ACM*

*Program. Lang.*, 8(OOPSLA2), October 2024. doi: 10.1145/3689740. URL `https://doi.org/10.1145/3689740`.

[19] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. Outcome logic: A unifying foundation of correctness and incorrectness reasoning. *Proc. ACM Program. Lang.*, 7(OOPSLA1), Apr 2023. doi: 10.1145/3586045. URL `https://doi.org/10.1145/3586045`.

[20] Noam Zilberstein, Angelina Saliling, and Alexandra Silva. Outcome separation logic: Local reasoning for correctness and incorrectness with computational effects. *Proc. ACM Program. Lang.*, 8(OOPSLA1), Apr 2024. doi: 10.1145/3649821.

[21] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010.

[22] Paolo Baldan, Francesco Ranzato, and Linpeng Zhang. A rice's theorem for abstract semantics. *CoRR*, abs/2105.14579, 2021. URL `https://arxiv.org/abs/2105.14579`.

[23] Paolo Baldan, Francesco Ranzato, and Linpeng Zhang. Intensional kleene and rice theorems for abstract program semantics. *Information and Computation*, 289:104953, 2022. ISSN 0890-5401. doi: https://doi.org/10.1016/j.ic.2022.104953. URL `https://www.sciencedirect.com/science/article/pii/S0890540122001080`.

[24] Kevin Batz, Adrian Gallus, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Tobias Winkler. Weighted Programming: A Programming Paradigm for Specifying Mathematical Models. *Proc. ACM Program. Lang.*, 6(OOPSLA1):1–30, 2022.

[25] D. Brière, C. Favre, and P. Traverse. A family of fault-tolerant systems: electrical flight controls, from airbus a320/330/340 to future military transport aircraft. *Microprocessors and Microsystems*, 19(2):

75–82, 1995. ISSN 0141-9331. doi: https://doi.org/10.1016/0141-9331(95)98982-P. URL `https://www.sciencedirect.com/science/article/pii/014193319598982P`.

[26] Daniel Kästner, Jörg Barrho, Ulrich Wünsche, Marc Schlickling, Bernhard Schommer, Michael Schmidt, Christian Ferdinand, Xavier Leroy, and Sandrine Blazy. CompCert: Practical Experience on Integrating and Qualifying a Formally Verified Optimizing Compiler. In *ERTS2 2018 - 9th European Congress Embedded Real-Time Software and Systems*, pages 1–9, Toulouse, France, January 2018. 3AF, SEE, SIE. URL `https://inria.hal.science/hal-01643290`.

[27] Joel Finch. Toyota sudden acceleration: A case study of the national highway traffic safety administration—recalls for change. *Loyola Consumer Law Review*, 22:472, 2009.

[28] Patrick Cousot. *Principles of Abstract Interpretation.* The MIT Press, 2022. URL `https://mitpress.mit.edu/9780262044905/principles-of-abstractinterpretation`.

[29] W. Eric Wong, Xue-Lin Li, and Phillip A. Laplante. Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures. *Journal of Systems and Software*, 133:68–94, 2017. doi: 10.1016/J.JSS.2017.06.069.

[30] Herb Krasner. The cost of poor software quality in the us: A 2020 report. In *Proceedings of the Consortium for Information Software QualityTM (CISQTM)*, pages 1–46, 2021.

[31] Robert Skeel. Roundoff error and the patriot missile. *SIAM News*, 25(4): 11, 1992.

[32] Mark Dowson. The ariane 5 software failure. *ACM SIGSOFT Software Engineering Notes*, 22(2):84, 1997. doi: 10.1145/251880.251992.

[33] K. Zuse, F.L. Bauer, and H. Zemanek. *Der Computer: Mein Lebenswerk*. Springer Berlin Heidelberg, 2013. ISBN 9783662065167. URL `https://books.google.co.uk/books?id=twipBgAAQBAJ`.

[34] Edsger W. Dijkstra. Letters to the editor: go to statement considered harmful. *Commun. ACM*, 11(3):147–148, mar 1968. ISSN 0001-0782. doi: 10.1145/362929.362947. URL `https://doi.org/10.1145/362929.362947`.

[35] C. A. R. Hoare. Quicksort. *Comput. J.*, 5(1):10–15, 1962. doi: 10.1093/COMJNL/5.1.10. URL `https://doi.org/10.1093/comjnl/5.1.10`.

[36] Michael O. Rabin. Probabilistic automata. *Inf. Control.*, 6(3):230–245, 1963. doi: 10.1016/S0019-9958(63)90290-0. URL `https://doi.org/10.1016/S0019-9958(63)90290-0`.

[37] Joost-Pieter Katoen. The probabilistic model checking landscape. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 31–45. ACM, 2016. doi: 10.1145/2933575.2934574. URL `https://doi.org/10.1145/2933575.2934574`.

[38] Dexter Kozen. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 101–114. IEEE Computer Society, 1979. doi: 10.1109/SFCS.1979.38. URL `https://doi.org/10.1109/SFCS.1979.38`.

[39] Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981. doi: 10.1016/0022-0000(81)90036-2. URL `https://doi.org/10.1016/0022-0000(81)90036-2`.

[40] Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent programs. In Richard A. DeMillo, editor, *Conference Record*

*of the Ninth Annual ACM Symposium on Principles of Programming Languages, Albuquerque, New Mexico, USA, January 1982*, pages 1–6. ACM Press, 1982. doi: 10.1145/582153.582154. URL `https://doi.org/10.1145/582153.582154`.

[41] Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent program. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, 1983. doi: 10.1145/2166.357214. URL `https://doi.org/10.1145/2166.357214`.

[42] Dexter Kozen. A probabilistic PDL. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 291–297. ACM, 1983. doi: 10.1145/800061.808758. URL `https://doi.org/10.1145/800061.808758`.

[43] Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the hardness of almost–sure termination. In Giuseppe F Italiano, Giovanni Pighizzini, and Donald T. Sannella, editors, *Mathematical Foundations of Computer Science 2015*, pages 307–318, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-48057-1.

[44] Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. On the hardness of analyzing probabilistic programs. *Acta Informatica*, 56:255 – 285, 2018. URL `https://api.semanticscholar.org/CorpusID:253768579`.

[45] Michael E. Akintunde, Andreea Kevorchian, Alessio Lomuscio, and Edoardo Pirovano. Verification of rnn-based neural agent-environment systems. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial*

*Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33016006. URL `https://doi.org/10.1609/aaai.v33i01.33016006`.

[46] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017. URL `http://arxiv.org/abs/1702.01135`.

[47] Vincent Tjeng and Russ Tedrake. Verifying neural networks with mixed integer programming. *CoRR*, abs/1711.07356, 2017. URL `http://arxiv.org/abs/1711.07356`.

[48] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Reachable set computation and safety verification for neural networks with relu activations. *CoRR*, abs/1712.08163, 2017. URL `http://arxiv.org/abs/1712.08163`.

[49] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018. doi: 10.1109/SP.2018.00058.

[50] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *CoRR*, abs/1809.08098, 2018. URL `http://arxiv.org/abs/1809.08098`.

[51] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10825–10836, Red Hook, NY, USA, 2018. Curran Associates Inc.

[52] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, 2019. doi: 10.1145/3290354. URL `https://doi.org/10.1145/3290354`.

[53] Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. Perfectly parallel fairness certification of neural networks. *CoRR*, abs/1912.02499, 2019. URL `http://arxiv.org/abs/1912.02499`.

[54] J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*, pages 11–11, 1982. doi: 10.1109/SP.1982.10014.

[55] Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *J. Comput. Secur.*, 4:167–188, 1996. URL `https://api.semanticscholar.org/CorpusID:1256259`.

[56] Dennis M. Volpano and Geoffrey Smith. A type-based approach to program security. In *Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, TAPSOFT '97, page 607–621, Berlin, Heidelberg, 1997. Springer-Verlag. ISBN 3540627812.

[57] H.G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74:358–366, 1953. doi: 10.2307/1990888.

[58] Shuo Ding and Qirun Zhang. Witnessability of undecidable problems. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. doi: 10.1145/3571227. URL `https://doi.org/10.1145/3571227`.

[59] Alan Mathison Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69. Univ. Math. Lab., Cambridge, 1949.

[60] Patrice Godefroid, Aditya V. Nori, Sriram K. Rajamani, and Sai Deep Tetali. Compositional may-must program analysis: unleashing the power of alternation. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '10, page 43–56, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605584799. doi: 10.1145/1706299.1706307. URL `https://doi.org/10.1145/1706299.1706307`.

[61] Edsko de Vries and Vasileios Koutavas. Reverse hoare logic. In *SEFM*, volume 7041 of *Lecture Notes in Computer Science*, pages 155–171. Springer, 2011.

[62] Robert W Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19(19-32):1, 1967.

[63] Edsger Wybe Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[64] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, Berlin, Heidelberg, 1990. ISBN 0387969578.

[65] Nick Benton. Simple relational correctness proofs for static analyses and program transformations. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '04)*, page 43. ACM, January 2004. URL `https://www.microsoft.com/en-us/research/publication/simple-relational-correctness-proofs-for-static-analyses-and-program-transformations/`.

[66] Torben Amtoft, Sruthi Bandhakavi, and Anindya Banerjee. A logic for information flow in object-oriented programs. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '06, page 91–102, New York, NY, USA, 2006.

Association for Computing Machinery. ISBN 1595930272. doi: 10.1145/1111037.1111046. URL `https://doi.org/10.1145/1111037.1111046`.

[67] David Costanzo and Zhong Shao. A separation logic for enforcing declarative information flow control policies. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust*, pages 179–198, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-642-54792-8.

[68] Marco Eilers, Thibault Dardinier, and Peter Müller. Commcsl: Proving information flow security for concurrent programs using abstract commutativity. *Proc. ACM Program. Lang.*, 7(PLDI), jun 2023. doi: 10.1145/3591289. URL `https://doi.org/10.1145/3591289`.

[69] Gidon Ernst and Toby Murray. Seccsl: Security concurrent separation logic. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 208–230, Cham, 2019. Springer International Publishing. ISBN 978-3-030-25543-5.

[70] Toby C. Murray. An under-approximate relational logic: Heralding logics of insecurity, incorrect implementation & more. *ArXiv*, abs/2003.04791, 2020. URL `https://api.semanticscholar.org/CorpusID:212644477`.

[71] Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. A logic for locally complete abstract interpretations. In *LICS*, pages 1–13. IEEE, 2021.

[72] Petar Maksimović, Caroline Cronjäger, Andreas Lööw, Julian Sutherland, and Philippa Gardner. Exact Separation Logic: Towards Bridging the Gap Between Verification and Bug-Finding. In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming (ECOOP 2023)*, volume 263 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:27, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-281-5. doi:

10.4230/LIPIcs.ECOOP.2023.19. URL `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2023.19`.

[73] Thibault Dardinier and Peter Müller. Hyper hoare logic: (dis-)proving program hyperproperties. *Proc. ACM Program. Lang.*, 8(PLDI), jun 2024. doi: 10.1145/3656437. URL `https://doi.org/10.1145/3656437`.

[74] Noam Zilberstein. A relatively complete program logic for effectful branching, 2024.

[75] Zohar Manna and Amir Pnueli. Axiomatic approach to total correctness of programs. *Acta Informatica*, 3:243–263, 1974. doi: 10.1007/BF00288637. URL `https://doi.org/10.1007/BF00288637`.

[76] Azalea Raad, Josh Berdine, Hoang-Hai Dang, Derek Dreyer, Peter O'Hearn, and Jules Villard. Local reasoning about the presence of bugs: Incorrectness separation logic. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 225–252, Cham, 2020. Springer International Publishing. ISBN 978-3-030-53291-8.

[77] Bernhard Möller, Peter O'Hearn, and Tony Hoare. On algebra of program correctness and incorrectness. In Uli Fahrenberg, Mai Gehrke, Luigi Santocanale, and Michael Winter, editors, *Relational and Algebraic Methods in Computer Science*, pages 325–343, Cham, 2021. Springer International Publishing. ISBN 978-3-030-88701-8.

[78] Flavio Ascari, Roberto Bruni, Roberta Gori, and Francesco Logozzo. Sufficient incorrectness logic: Sil and separation sil, 2023.

[79] Azalea Raad, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. Concurrent incorrectness separation logic. *Proc. ACM Program. Lang.*, 6(POPL), jan 2022. doi: 10.1145/3498695. URL `https://doi.org/10.1145/3498695`.

[80] Azalea Raad, Julien Vanegue, Josh Berdine, and Peter O'Hearn. A General Approach to Under-Approximate Reasoning About Concurrent Programs. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory (CONCUR 2023)*, volume 279 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:17, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-299-0. doi: 10.4230/LIPIcs.CONCUR.2023.25. URL `https://drops.dagstuhl.de/opus/volltexte/2023/19019`.

[81] Azalea Raad, Julien Vanegue, and Peter O'Hearn. Non-termination proving at scale. *Proc. ACM Program. Lang.*, 8(OOPSLA2), October 2024. doi: 10.1145/3689720. URL `https://doi.org/10.1145/3689720`.

[82] Dana S. Scott. A type-theoretical alternative to iswim, cuch, OWHY. *Theor. Comput. Sci.*, 121(1&2):411–440, 1993. doi: 10.1016/0304-3975(93)90095-B. URL `https://doi.org/10.1016/0304-3975(93)90095-B`.

[83] G. Grätzer. *Lattice Theory: Foundation.* SpringerLink : Bücher. Springer Basel, 2011. ISBN 9783034800181. URL `https://books.google.co.uk/books?id=6XJX5-zCoIQC`.

[84] G. Birkhoff. *Lattice Theory.* Number v. 25, pt. 2 in American Mathematical Society colloquium publications. American Mathematical Society, 1940. ISBN 9780821810255. URL `https://books.google.co.uk/books?id=ePqVAwAAQBAJ`.

[85] Dana Scott. Outline of a mathematical theory of computation. Technical Report PRG02, OUCL, November 1970.

[86] Daniel Rosiak. *Sheaf Theory through Examples.* 10 2022. ISBN 9780262370424. doi: 10.7551/mitpress/12581.001.0001.

[87] Samson Abramsky and Achim Jung. *Domain Theory*, volume 3. Oxford University Press, 1994. Corrected and expanded version available at `http://www.cs.bham.ac.uk/~axj/pub/papers/handy1.pdf`.

[88] Bronisław Knaster. Un théorème sur les functions d'ensembles. *Annales de la Societe Polonaise de Mathematique*, 6:133–134, 1928.

[89] Jean-Louis Lassez, V. L. Nguyen, and Liz Sonenberg. Fixed point theorems and semantics: A folk tale. *Inf. Process. Lett.*, 14:112–116, 1982. URL `https://api.semanticscholar.org/CorpusID:19971115`.

[90] Alfred Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–310, 1955.

[91] Dana S. Scott. The lattice of flow diagrams. In *Symposium on Semantics of Algorithmic Languages*, volume 188 of *Lecture Notes in Mathematics*, pages 311–366. Springer, 1971.

[92] Patrick Cousot and Radhia Cousot. Constructive versions of tarski's fixed point theorems. *Pacific Journal of Mathematics*, 81(1):43–57, 1979.

[93] Patrick Cousot. Calculational design of [in]correctness transformational program logics by abstract interpretation. *Proc. ACM Program. Lang.*, 8(POPL), jan 2024. doi: 10.1145/3632849. URL `https://doi.org/10.1145/3632849`.

[94] David Michael Ritchie Park. Fixpoint induction and proofs of program properties. volume 5. Machine intelligence, 1969.

[95] Claire Jones. *Probabilistic Non-Determinism.* PhD thesis, University of Edinburgh, UK, 1990.

[96] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Thomas Noll. Quantitative separation logic. *CoRR*, abs/1802.10467, 2018. URL `http://arxiv.org/abs/1802.10467`.

[97] Lena Verscht, Ānrán Wáng, and Benjamin Lucien Kaminski. Partial incorrectness logic, 2025. URL `https://arxiv.org/abs/2502.14626`.

[98] Yukihiro Oda. Proof systems for partial incorrectness logic (partial reverse hoare logic), 2025. URL `https://arxiv.org/abs/2502.21053`.

[99] Matthew S. Hecht. *Flow Analysis of Computer Programs.* Elsevier, 1977.

[100] Xavier Rival and Kwangkeun Yi. *Introduction to Static Analysis – An Abstract Interpretation Perspective.* MIT Press, 2020.

[101] Benjamin Lucien Kaminski and Joost-Pieter Katoen. A weakest pre-expectation semantics for mixed-sign expectations. In *LICS*, pages 1–12. IEEE Computer Society, 2017.

[102] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Relatively complete verification of probabilistic programs: an expressive language for expectation-based reasoning. *Proc. ACM Program. Lang.*, 5(POPL):1–30, 2021.

[103] Donald E. Knuth. Two notes on notation. *Am. Math. Monthly*, 99(5): 403–422, May 1992.

[104] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Monographs in Computer Science. Springer, 2005.

[105] Patrick Cousot, Radhia Cousot, Manuel Fähndrich, and Francesco Logozzo. Automatic inference of necessary preconditions. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 128–148, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[106] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1): 230–265, 1936. doi: 10.2307/2268810.

[107] Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7:70–90, 1978.

[108] Alejandro Aguirre and Shin-ya Katsumata. Weakest preconditions in fibrations. *Electronic Notes in Theoretical Computer Science*, 352:5–27, 2020. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2020.09.002. URL `https://www.sciencedirect.com/science/article/pii/S1571066120300487`. The 36th Mathematical Foundations of Programming Semantics Conference, 2020.

[109] R. J. R. Back. A calculus of refinements for program derivations. *Acta Inf.*, 25(6):593–624, August 1988. ISSN 0001-5903. doi: 10.1007/BF00291051. URL `https://doi.org/10.1007/BF00291051`.

[110] Dean Jacobs and David Gries. General correctness: A unification of partial and total correctness. *Acta Inf.*, 22(1):67–83, April 1985. ISSN 0001-5903. doi: 10.1007/BF00290146. URL `https://doi.org/10.1007/BF00290146`.

[111] Gia S. Wulandari and Detlef Plump. Verifying graph programs with first-order logic. *Electronic Proceedings in Theoretical Computer Science*, 330:181–200, Dec 2020. ISSN 2075-2180. doi: 10.4204/eptcs.330.11. URL `http://dx.doi.org/10.4204/EPTCS.330.11`.

[112] P. Ørbæk and J. Palsberg. Trust in the $\lambda$-calculus. *J. Funct. Program.*, 7(6):557–591, November 1997. ISSN 0956-7968. doi: 10.1017/S0956796897002906. URL `https://doi.org/10.1017/S0956796897002906`.

[113] Torben Amtoft and Anindya Banerjee. Information flow analysis in logical form. In Roberto Giacobazzi, editor, *Static Analysis*, pages 100–115, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-27864-1.

[114] Caterina Urban and Peter Müller. An Abstract Interpretation Framework for Input Data Usage. In *ESOP*, pages 683–710, 2018.

[115] Patrick Cousot. Abstract semantic dependency. In *SAS*, volume 11822 of *Lecture Notes in Computer Science*, pages 389–410. Springer, 2019.

[116] Denis Mazzucato, Marco Campion, and Caterina Urban. Quantitative input usage static analysis. In *NASA Formal Methods: 16th International Symposium, NFM 2024, Moffett Field, CA, USA, June 4–6, 2024, Proceedings*, page 79–98, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-60697-7. doi: 10.1007/978-3-031-60698-4_5. URL `https://doi.org/10.1007/978-3-031-60698-4_5`.

[117] Peter W. O'Hearn. Resources, concurrency and local reasoning. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, pages 49–67, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-28644-8.

[118] J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74, 2002. doi: 10.1109/LICS.2002.1029817.

[119] Geoffrey Smith. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computational Structures*, pages 288–302. Springer, 2009.

[120] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[121] Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Trans. Program. Lang. Syst.*, 18(3):325–353, 1996.

[122] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Future of Software Engineering Proceedings*, FOSE 2014, page 167–181, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328654. doi: 10.1145/2593882.2593900. URL `https://doi.org/10.1145/2593882.2593900`.

[123] Daniele Varacca and Glynn Winskel. Distributing probabililty over nondeterminism. *Mathematical Structures in Computer Science*, 16:87–113, 02 2006. doi: 10.1017/S0960129505005074.

[124] Daniele Varacca. *Probability, Nondeterminism and Concurrency. Two Denotational Models for Probabilistic Computation.* PhD thesis, BRICS – Aarhus University, 2003. URL `http://www.brics.dk/DS/03/14/`.

[125] Regina Tix, Klaus Keimel, and Gordon Plotkin. Semantic domains for combining probability and non-determinism. *Electronic Notes in Theoretical Computer Science*, 222:3–99, 2009. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2009.01.002. URL `https://www.sciencedirect.com/science/article/pii/S1571066109000036`.

[126] Michael W. Mislove. Nondeterminism and probabilistic choice: Obeying the laws. In *Proceedings of the 11th International Conference on Concurrency Theory*, CONCUR '00, page 350–364, Berlin, Heidelberg, 2000. Springer-Verlag. ISBN 3540678972.

[127] M.W. Mislove. On combining probability and nondeterminism. *Electronic Notes in Theoretical Computer Science*, 162: 261–265, 2006. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2005.12.113. URL `https://www.sciencedirect.com/science/article/pii/S1571066106004440`. Proceedings of the Workshop "Essays on Algebraic Process Calculi" (APC 25).

[128] Klaus Keimel and Gordon Plotkin. Mixed powerdomains for probability

and nondeterminism. *Logical Methods in Computer Science*, 13, 12 2017. doi: 10.23638/LMCS-13(1:2)2017.

[129] Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. A core quantitative coeffect calculus. In Zhong Shao, editor, *Programming Languages and Systems*, pages 351–370, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-642-54833-8.

[130] Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, and Tetsuya Sato. Graded hoare logic and its categorical semantics. In Nobuko Yoshida, editor, *Programming Languages and Systems*, pages 234–263, Cham, 2021. Springer International Publishing. ISBN 978-3-030-72019-3.

[131] J. Golan. Semirings and affine equations over them: Theory and applications. 2003. doi: 10.1007/978-94-017-0383-3.

[132] Georg Karner. Continuous monoids and semirings. *Theoretical Computer Science*, 318(3):355–372, 2004. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2004.01.020. URL `https://www.sciencedirect.com/science/article/pii/S0304397504000714`.

[133] Alessio Ferrarini. *Abstract Hoare logic.* PhD thesis, University of Padova, 2024. URL `https://thesis.unipd.it/retrieve/32553177-ce4e-4ee0-bc91-285964a4f8de/thesis.pdf`.

[134] Patrick Cousot and Radhia Cousot. An abstract interpretation framework for termination. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, page 245–258, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450310833. doi: 10.1145/2103656.2103687. URL `https://doi.org/10.1145/2103656.2103687`.

[135] Dennis Komm. *An Introduction to Online Computation - Determinism, Randomization, Advice.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016. doi: 10.1007/978-3-319-42749-2.

[136] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[137] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms, The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, 1998. Springer. doi: 10.1007/BFb0029561. the book grow out of a Dagstuhl Seminar, June 1996.

[138] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. In *SODA*, pages 835–844. SIAM, 2009. doi: 10.1137/1.9781611973068.91.

[139] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms*, 6(2):28:1–28:36, 2010. doi: 10.1145/1721837.1721844.

[140] Lena Verscht and Benjamin Lucien Kaminski. A taxonomy of hoare-like logics: Towards a holistic view using predicate transformers and kleene algebras with top and tests. *Proc. ACM Program. Lang.*, 9(POPL), January 2025. doi: 10.1145/3704896. URL `https://doi.org/10.1145/3704896`.

[141] Robert Endre Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic Discrete Methods*, 6(2):306–318, 1985. doi: 10.1137/0606031. URL `https://doi.org/10.1137/0606031`.

[142] Quentin Carbonneaux. *Modular and certified resource-bound analyses*. PhD thesis, Yale University, 2018.

[143] Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. Compositional certified resource bounds. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '15, page 467–478, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334686. doi: 10.1145/2737924.2737955. URL `https://doi.org/10.1145/2737924.2737955`.

[144] James Li, Noam Zilberstein, and Alexandra Silva. Total outcome logic: Proving termination and nontermination in programs with branching, 2024. URL `https://arxiv.org/abs/2411.00197`.

[145] Noam Zilberstein, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. A demonic outcome logic for randomized nondeterminism, 2024. URL `https://arxiv.org/abs/2410.22540`.

[146] Lena Verscht and Benjamin Kaminski. Hoare-like triples and kleene algebras with top and tests: Towards a holistic perspective on hoare logic, incorrectness logic, and beyond, 2023. URL `https://arxiv.org/abs/2312.09662`.

[147] Russel O'Conner. A very general method of computing shortest paths. Personal blog entry, 2012. URL `http://r6.ca/blog/20110808T035622Z.html`.

[148] Stephen Dolan. Fun with semirings: A functional pearl on the abuse of linear algebra. In *ICFP*, pages 101–110. ACM, 2013. doi: 10.1145/2500365.2500613.

[149] Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *LICS*, pages 301–310. IEEE Computer Society, 2013. doi: 10.1109/lics.2013.36.

[150] Dexter Kozen. On hoare logic and kleene algebra with tests. In *LICS*, pages 167–172. IEEE Computer Society, 1999. doi: 10.1109/lics.1999.782610.

[151] Dexter Kozen. On hoare logic and kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000. doi: 10.1145/343369.343378.

[152] Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proc. ACM Program.*

*Lang.*, 4(POPL), December 2019. doi: 10.1145/3371129. URL `https://doi.org/10.1145/3371129`.

[153] Wojciech Różowski, Tobias Kappé, Dexter Kozen, Todd Schmid, and Alexandra Silva. Probabilistic Guarded KAT Modulo Bisimilarity: Completeness and Complexity. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 136:1–136:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-278-5. doi: 10.4230/LIPIcs.ICALP.2023.136. URL `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2023.136`.

[154] Leandro Gomes, Alexandre Madeira, and Luís Soares Barbosa. Generalising KAT to verify weighted computations. *Sci. Ann. Comput. Sci.*, 29 (2):141–184, 2019. doi: 10.7561/sacs.2019.2.141.

[155] Kevin Batz, Mingshuai Chen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Philipp Schröer. Latticed k-induction with an application to probabilistic programs. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification*, pages 524–549, Cham, 2021. Springer International Publishing. ISBN 978-3-030-81688-9.

[156] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In Warren A. Hunt and Steven D. Johnson, editors, *Formal Methods in Computer-Aided Design*, pages 127–144, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-40922-9.

[157] Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis.

*2013 Formal Methods in Computer-Aided Design*, pages 1–8, 2013. URL `https://api.semanticscholar.org/CorpusID:6705760`.

[158] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1):90–121, January 1980. ISSN 0164-0925. doi: 10.1145/357084.357090. URL `https://doi.org/10.1145/357084.357090`.

[159] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A new proof rule for almost-sure termination. *Proc. ACM Program. Lang.*, 2(POPL), December 2017. doi: 10.1145/3158121. URL `https://doi.org/10.1145/3158121`.

[160] Rupak Majumdar and V. R. Sathiyanarayana. Sound and complete proof rules for probabilistic termination, 2024. URL `https://arxiv.org/abs/2404.19724`.

[161] Kevin Batz, Tom Jannik Biskup, Joost-Pieter Katoen, and Tobias Winkler. Programmatic strategy synthesis: Resolving nondeterminism in probabilistic programs. *Proc. ACM Program. Lang.*, 8(POPL), January 2024. doi: 10.1145/3632935. URL `https://doi.org/10.1145/3632935`.

[162] Philipp Schröer, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. A deductive verification infrastructure for probabilistic programs - artifact evaluation (version 1). `https://doi.org/10.5281/zenodo.8146987`, July 2023. URL `https://doi.org/10.5281/zenodo.8146987`. Accessed on YYYY-MM-DD.

[163] Vaughan R. Pratt. Semantical Considerations on Floyd-Hoare Logic. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 109–121, 1976. doi: 10.1109/SFCS.1976.27.

[164] Edmund Melson Clarke. Programming language constructs for which it is impossible to obtain good hoare axiom systems. *J. ACM*, 26(1):

129–147, jan 1979. ISSN 0004-5411. doi: 10.1145/322108.322121. URL `https://doi.org/10.1145/322108.322121`.

[165] Kevin R. O'Neill, Michael R. Clarkson, and Stephen Chong. Information-flow security for interactive programs. In *Proceedings of the 19th IEEE Workshop on Computer Security Foundations*, CSFW '06, page 190–201, USA, 2006. IEEE Computer Society. ISBN 0769526152. doi: 10.1109/CSFW.2006.16. URL `https://doi.org/10.1109/CSFW.2006.16`.

[166] Jerry den Hartog. *Probabilistic Extensions of Semantical Models*. PhD thesis, Vrije Universiteit Amsterdam, 2002. URL `https://core.ac.uk/reader/15452110`.

[167] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *The Bulletin of Symbolic Logic*, 5(2):215–244, 1999. ISSN 10798986. URL `http://www.jstor.org/stable/421090`.

[168] David J. Pym, Peter W. O'Hearn, and Hongseok Yang. Possible worlds and resources: the semantics of bi. *Theoretical Computer Science*, 315(1):257–305, 2004. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2003.11.020. URL `https://www.sciencedirect.com/science/article/pii/S0304397503006248`. Mathematical Foundations of Programming Semantics.

[169] Ira Fesefeldt, Joost-Pieter Katoen, and Thomas Noll. Towards Concurrent Quantitative Separation Logic. In Bartek Klin, Sławomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory (CONCUR 2022)*, volume 243 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:24, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-246-4. doi: 10.4230/LIPIcs.CONCUR.2022.25. URL `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CONCUR.2022.25`.

[170] Stephen Brookes. A semantics for concurrent separation logic. *Theoretical Computer Science*, 375(1):227–270, 2007. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2006.12.034. URL `https://www.sciencedirect.com/science/article/pii/S0304397506009248`. Festschrift for John C. Reynolds's 70th birthday.

[171] Gilles Barthe, Justin Hsu, and Kevin Liao. A probabilistic separation logic. *Proc. ACM Program. Lang.*, 4(POPL), dec 2019. doi: 10.1145/3371123. URL `https://doi.org/10.1145/3371123`.

[172] Noam Zilberstein, Alexandra Silva, and Joseph Tassarotti. Probabilistic concurrent reasoning in outcome logic: Independence, conditioning, and invariants, 2024. URL `https://arxiv.org/abs/2411.11662`.

[173] Dexter Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3):427–443, may 1997. ISSN 0164-0925. doi: 10.1145/256167.256195. URL `https://doi.org/10.1145/256167.256195`.

[174] Joakim von Wright. From kleene algebra to refinement algebra. In *International Conference on Mathematics of Program Construction*, 2002. URL `https://api.semanticscholar.org/CorpusID:2003560`.

[175] Daryl McCullough. Specifications for multi-level security and a hook-up. In *1987 IEEE Symposium on Security and Privacy*, pages 161–161, 1987. doi: 10.1109/SP.1987.10009.

[176] J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996. doi: 10.1109/32.481534.

[177] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust*, pages 265–284, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-642-54792-8.

[178] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. *SIGPLAN Not.*, 44(1): 90–101, jan 2009. ISSN 0362-1340. doi: 10.1145/1594834.1480894. URL `https://doi.org/10.1145/1594834.1480894`.

[179] Kenji Maillard, Cătălin Hriţcu, Exequiel Rivas, and Antoine Van Muylder. The next 700 relational program logics. *Proc. ACM Program. Lang.*, 4(POPL), dec 2019. doi: 10.1145/3371072. URL `https://doi.org/10.1145/3371072`.

[180] Kenji Maillard, Danel Ahman, Robert Atkey, Guido Martínez, Cătălin Hriţcu, Exequiel Rivas, and Éric Tanter. Dijkstra monads for all. *Proc. ACM Program. Lang.*, 3(ICFP), jul 2019. doi: 10.1145/3341708. URL `https://doi.org/10.1145/3341708`.

[181] Marcelo Sousa and Isil Dillig. Cartesian hoare logic for verifying k-safety properties. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, page 57–69, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342612. doi: 10.1145/2908080.2908092. URL `https://doi.org/10.1145/2908080.2908092`.

[182] Emanuele D'Osualdo, Azadeh Farzan, and Derek Dreyer. Proving hypersafety compositionally. *Proc. ACM Program. Lang.*, 6(OOPSLA2), oct 2022. doi: 10.1145/3563298. URL `https://doi.org/10.1145/3563298`.

[183] Robert Dickerson, Qianchuan Ye, Michael K. Zhang, and Benjamin Delaware. Rhle: Modular deductive verification of relational $\forall\exists$ properties. In *Programming Languages and Systems: 20th Asian Symposium, APLAS 2022, Auckland, New Zealand, December 5, 2022, Proceedings*, page 67–87, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-21036-5. doi: 10.1007/978-3-031-21037-2_4. URL `https://doi.org/10.1007/978-3-031-21037-2_4`.

[184] Mounir Assaf, David A. Naumann, Julien Signoles, Éric Totel, and Frédéric Tronel. Hypercollecting semantics and its application to static analysis of information flow. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, page 874–887, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346603. doi: 10.1145/3009837.3009889. URL `https://doi.org/10.1145/3009837.3009889`.

[185] Isabella Mastroeni and Michele Pasqua. Verifying bounded subset-closed hyperproperties. In Andreas Podelski, editor, *Static Analysis*, pages 263–283, Cham, 2018. Springer International Publishing. ISBN 978-3-319-99725-4.

[186] Krzysztof R. Apt and Ernst-Rüdiger Olderog. Fifty years of hoare's logic. *Form. Asp. Comput.*, 31(6):751–807, December 2019. ISSN 0934-5043. doi: 10.1007/s00165-019-00501-3. URL `https://doi.org/10.1007/s00165-019-00501-3`.

[187] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction.* MIT Press, Cambridge, MA, USA, 1993. ISBN 0262231697.

[188] J. Loeckx, K. Sieber, and R.D. Stansifer. *The Foundations of Program Verification.* Wiley Teubner Series on Applicable Theory in Computer Science Series. John Wiley, 1984. ISBN 9783519021018. URL `https://books.google.de/books?id=wagmAAAAMAAJ`.

[189] Makoto Tatsuta, Wei-Ngan Chin, and Mahmudul Faisal Al Ameen. Completeness of pointer program verification by separation logic. In *Software Engineering and Formal Methods*, pages 179–188. IEEE Computer Society, 2009.

[190] Makoto Tatsuta, Wei-Ngan Chin, and Mahmudul Faisal Al Ameen. Completeness and expressiveness of pointer program verification by separation logic. *Inf. Comput.*, 267:1–27, 2019.

[191] Jerry den Hartog and Erik P. de Vink. Verifying probabilistic programs using a hoare like logic. *Int. J. Found. Comput. Sci.*, 13(3):315–340, 2002.

[192] Simon Holm Jensen, Anders Møller, and Peter Thiemann. Interprocedural analysis with lazy propagation. In Radhia Cousot and Matthieu Martel, editors, *Static Analysis*, pages 320–339, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15769-1.

[193] Helmut Seidl and Christian Fecht. Interprocedural analyses: a comparison. *The Journal of Logic Programming*, 43(2):123–156, 2000. ISSN 0743-1066. doi: https://doi.org/10.1016/S0743-1066(99)00058-8. URL `https://www.sciencedirect.com/science/article/pii/S0743106699000588`.

[194] Benno Stein, Bor-Yuh Evan Chang, and Manu Sridharan. Interactive abstract interpretation with demanded summarization. *ACM Trans. Program. Lang. Syst.*, 46(1), March 2024. ISSN 0164-0925. doi: 10.1145/3648441. URL `https://doi.org/10.1145/3648441`.

[195] Dino Distefano, Matteo Marescotti, Cons T Åhs, Sopot Cela, Gabriela Cunha Sampaio, Radu Grigore, Ákos Hajdu, Timotej Kapus, Ke Mao, and Thibault Suzanne. Enhancing compositional static analysis with dynamic analysis. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 2121—2129. ACM, 2024. doi: 10.1145/3691620.3695599.