A Relatively Complete Program Logic for Effectful Branching

Noam Zilberstein
noamz@cs.cornell.edu
Cornell University
USA

ABSTRACT

Starting with Hoare Logic over 50 years ago, numerous sound and relatively complete program logics have been devised to reason about the diverse programs encountered in the real world. This includes reasoning about computational effects, particularly those effects that cause the program execution to branch into multiple paths due to, *e.g.,* nondeterministic or probabilistic choice.

The recently introduced Outcome Logic reimagines Hoare Logic with effects at its core, using an algebraic representation of choice to capture a variety of effects. In this paper, we give the first relatively complete proof system for Outcome Logic, handling general purpose looping for the first time. We also show that this proof system applies to programs with various effects and that it facilitates the reuse of proof fragments across different kinds of specifications.

1 INTRODUCTION

The seminal work of Floyd [26] and Hoare [29] on program logics in the 1960s has paved the way towards modern program analysis. The resulting *Hoare Logic*—still ubiquitous today—defines triples $\{P\}\ C\ \{Q\}$ to specify the behavior of a program C in terms of a precondition P and a postcondition Q. In the ensuing years, many variants of Hoare Logic have emerged, in part to handle the numerous computational effects found in real-world programs.

Such effects include nontermination, arising from while loops; nondeterminism, useful for modeling opaque aspects of program evaluation such as user input or concurrent scheduling; and randomization, required for security and machine learning applications.

These effects have historically warranted specialized program logics with distinct inference rules. For example, partial correctness [26, 29] vs total correctness [43] can be used to specify that the postcondition holds *if* the program terminates vs that it holds *and* the programs terminates, respectively. While Hoare Logic has classically taken a demonic view of nondeterminism (the postcondition must apply to *all* possible outcomes), recent work on formal methods for incorrectness [45, 47] has motivated the need for new program logics based on angelic nondeterminism (the postcondition applies to *some* reachable outcome). Further, probabilistic Hoare Logics are quantitative, allowing one to specify the likelihood of each outcome, not just that they may occur [3, 19, 20, 53].

Despite these apparent differences, all of the aforementioned program logics share common reasoning principles. For instance, sequences of commands $C_1 \fatsemi C_2$ are analyzed compositionally and the precondition (resp., postcondition) can be strengthened (resp., weakened) using logical consequences, as shown below.

$$\frac{\{P\}\ C_1\ \{Q\} \quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1 \fatsemi C_2\ \{R\}} \qquad \frac{P' \Rightarrow P \quad \{P\}\ C\ \{Q\} \quad Q \Rightarrow Q'}{\{P'\}\ C\ \{Q'\}}$$

As we show in this paper, those common reasoning principles are no mere coincidence. We give a uniform metatheoretic treatment to program logics with a variety of computational effects—including nondeterminism and randomization—culminating in a single relatively complete proof system for all of them. We also show how specialized reasoning principles (*e.g.,* loop invariants for partial correctness) are derived from our more general rules and how proof fragments can be shared between programs with different effects.

This work is not only of value to theoreticians. Recent interest in static analysis for incorrectness [39, 45, 47, 50, 51] has prompted the development of new program logics, distinct from Hoare Logic. Subsequently—and largely with the goal of consolidating static analysis tools—more logics were proposed to capture *both* correctness (*i.e.,* Hoare Logic) *and* incorrectness [9, 10, 17, 41, 57, 58].

One such effort, which we build upon in this paper, is Outcome Logic (OL). Outcome Logic was first proposed as a unified basis for correctness and incorrectness reasoning in nondeterministic and probabilistic programs, with semantics parametric on a *monad* and a *monoid* [57]. The semantics was later refined such that each trace is weighted using an element of a semiring [58]. For example, Boolean weights specify which states are in the set of outcomes for a nondeterministic program whereas real-valued weights quantify the probabilities of outcomes in a probabilistic program. Exposing these weights in pre- and postconditions means that a single program logic can express multiple termination criteria, angelic *and* demonic nondeterminism, probabilistic properties, and more.

The previous work on Outcome Logic has investigated its semantics and connection to separation logic, leaving the proof theory largely unexplored. Most notably, prior work only supports reasoning about loops via bounded unrolling, which is not suitable for loops that iterate an indeterminate number of times. In this paper, we give a full account of the Outcome Logic proof theory and explore more instances than have been investigated previously. More precisely, our contributions are as follows:

- We define the Outcome Logic semantics and give five models (Sections 2 and 3), including a multiset model (Example 2.6) not supported by previous formalizations due to more restrictive algebraic constraints. Our new looping construct naturally supports deterministic (while loops), nondeterministic, and probabilistic iteration—whereas previous OL versions supported fewer kinds of iteration [58] or used a non-unified, ad-hoc semantics [57].
- We provide a proof system and prove that it is sound and relatively complete (Section 4). It is the first OL proof system that handles loops that iterate an indeterminate number of times. Our Iter rule is sufficient for analyzing any iterative command, and from it we derive the typical rules for loop invariants (for partial correctness), loop variants (with termination guarantees), as well as more complex probabilistic while loops (Section 5).
- We prove that OL subsumes Hoare Logic (Section 3.3) and derive the entire Hoare Logic proof system (*e.g.,* loop invariants) in Outcome Logic (Section 5). Inspired by Dynamic Logic [49],

our encoding of Hoare Logic uses modalities to extend partial correctness beyond just nondeterministic programs.

- ▹ We demonstrate the reusability of proofs across different effects (*e.g.,* nondeterminism or randomization) and properties (*e.g.,* angelic or demonic nondeterminism) (Section 7). Whereas choices about how to handle loops typically require selecting a specific program logic (*e.g.,* partial vs total correctness), loop analysis strategies can be mixed within a single OL derivation, meaning that static analysis algorithms can avoid recomputing specifications when analyzing codebases with many procedures.
- ▹ We perform combinatorial analysis of graph algorithms based on alternative computation models (Section 8).
- ▹ We contextualize the paper in terms of related work and discuss the outlooks (Section 9).

2 WEIGHTED PROGRAM SEMANTICS

We begin the technical development by defining a basic programming language and describing its semantics based on various interpretations of choice. The syntax for the language is shown below.

$$\begin{aligned} C &::= \mathsf{skip} \mid C_1 \fatsemi C_2 \mid C_1 + C_2 \mid \mathsf{assume}\ e \mid C^{\langle e,e'\rangle} \mid a \in \mathsf{Act} \\ e &::= b \mid u \in U \\ b &::= \mathsf{true} \mid \mathsf{false} \mid b_1 \vee b_2 \mid b_1 \wedge b_2 \mid \neg b \mid t \in \mathsf{Test} \end{aligned}$$

At first glance, this language appears similar to imperative languages such as Dijkstra's Guarded Command Language (GCL) [21], with familiar constructs such as skip, sequential composition ($C_1 \fatsemi C_2$), choice ($C_1 + C_2$), and primitive actions $a \in \mathsf{Act}$. The differences arise from the generalized assume operation, which weights the current computation branch using an expression e (either a test b or a *weight* $u \in U$, which will be described in Section 2.1).

Weighting is also used in the iteration command $C^{\langle e,e'\rangle}$, which iterates C with weight e and exits with weight e'. It is a generalization of the Kleene star $C^\star$, and is also more general than the iteration constructs found in previous Outcome Logic work [57, 58]. In Section 2.3, we will show how to encode while loops, Kleene star, and probabilistic loops using $C^{\langle e,e'\rangle}$. Although the latter constructs can be encoded using while loops and auxiliary variables, capturing this behavior *without* state opens up the possibility for complete equational theories over uninterpreted atomic commands [37, 54].

Tests b contain the typical operations of Boolean algebras as well as primitive tests $t \in \mathsf{Test}$, assertions about a program state. Primitive tests are represented semantically, so $\mathsf{Test} \subseteq 2^\Sigma$ where Σ is the set of program states (each primitive test $t \subseteq \Sigma$ is the set of states that it describes). Tests evaluate to $\mathbb{0}$ or $\mathbb{1}$, which are abstract Booleans representing false and true, respectively.

The values $\mathbb{0}$ and $\mathbb{1}$ are two examples of weights from the set $\{\mathbb{0}, \mathbb{1}\} \subseteq U$. These weights have particular algebraic properties that will be described fully in Section 2.1. The command assume b can be thought of as choosing whether or not to continue evaluating the current branch of computation, whereas assume u more generally picks a weight for the branch, which may be a Boolean ($\mathbb{0}$ or $\mathbb{1}$), but may also be some other type of weight such as a probability. In the remainder of this section, we will define the semantics formally.

2.1 Algebraic Preliminaries

We begin by reviewing some algebraic structures. First, we define the properties of the weights for each computation branch.

Definition 2.1 (Monoid). A monoid $\langle U, +, \mathbb{0}\rangle$ consists of a carrier set U, an associative binary operation $+\colon U \times U \to U$, and an identity element $\mathbb{0} \in U$ ($u + \mathbb{0} = \mathbb{0} + u = u$). If $+\colon U \times U \rightharpoonup U$ is partial, then the monoid is partial. If $+$ is commutative ($u + v = v + u$), then the monoid is commutative.

As an example, $\langle \{0, 1\}, \vee, 0\rangle$ is a monoid on Booleans.

Definition 2.2 (Semiring). A semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ is an algebraic structure such that $\langle U, +, \mathbb{0}\rangle$ is a commutative monoid, $\langle U, \cdot, \mathbb{1}\rangle$ is a monoid, and the following additional properties hold:

(1) Distributivity: $u\cdot(v+w) = u\cdot v+u\cdot w$ and $(u+v)\cdot w = u\cdot w+v\cdot w$
(2) Annihilation: $\mathbb{0} \cdot u = u \cdot \mathbb{0} = \mathbb{0}$

The semiring is partial if $\langle U, +, \mathbb{0}\rangle$ is a partial monoid (but $\cdot$ is total).

Semirings elements will act as the *weights* for traces in our semantics. That is, the interpretation of a program at a state $\sigma \in \Sigma$ will map each end state to a semiring element $[\![C]\!]\,(\sigma)\colon \Sigma \to U$. Varying the semiring will give us different kinds of effects. For example, a Boolean semiring where $U = \{0, 1\}$ corresponds to nondeterministic computation; $[\![C]\!]\,(\sigma)\colon \Sigma \to \{0, 1\} \cong 2^\Sigma$ tells us which states are in the set of nondeterministic outcomes. A probabilistic semiring where $U = [0, 1]$ (the unit interval of real numbers) gives us a map from states to probabilities—a *distribution* of outcomes. More formally, the result is a *weighting function*, defined below.

Definition 2.3 (Weighting Function). Given a set X and a partial semiring $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$, the set of weighting functions is:

$$\mathcal{W}_\mathcal{A}(X) \triangleq \left\{ m\colon X \to U \;\middle|\; |m| \text{ is defined and } \mathrm{supp}(m) \text{ is countable} \right\}$$

Where $\mathrm{supp}(m) \triangleq \{\sigma \mid m(\sigma) \neq \mathbb{0}\}$, $|m| \triangleq \sum_{\sigma\in\mathrm{supp}(m)} m(\sigma)$, and $\sum$ is an operation based on $+$ described in Appendix A.

Weighting functions can encode the following types of computation.

Example 2.4 (Nondeterminism). Nondeterministic computation is based on the Boolean semiring $\mathsf{Bool} = \langle \mathbb{B}, \vee, \wedge, 0, 1\rangle$, where weights are drawn from $\mathbb{B} = \{0, 1\}$ and *conjunction* $\wedge$ and *disjunction* $\vee$ are the usual logical operations. This gives us $\mathcal{W}_{\mathsf{Bool}}(X) \cong 2^X$—weighting functions on Bool are isomorphic to sets.

Example 2.5 (Determinism). Deterministic computation also uses Boolean weights, but with a different interpretation of the semiring $+$; that is, $0 + x = x + 0 = x$, but $1 + 1$ is undefined. The semiring is therefore $\mathsf{Bool}' = \langle \mathbb{B}, +, \wedge, 0, 1\rangle$. With this definition of $+$, the requirement of Definition 2.3 that $|m|$ is defined means that $|\mathrm{supp}(m)| \leq 1$, so we get that $\mathcal{W}_{\mathsf{Bool}'}(X) \cong X + 1$—it is either a single value $x \in X$, or $\star \in 1$, indicating that the program diverged.

Example 2.6 (Multiset Nondeterminism). Rather than indicating which outcomes are possible using Booleans, we use natural numbers (extended with ∞) $n \in \mathbb{N}^\infty$ to count the traces leading to each outcome. This gives us the semiring $\mathsf{Nat} = \langle \mathbb{N}^\infty, +, \cdot, 0, 1\rangle$ where $+$ and $\cdot$ are the standard arithmetic operations, and we get that $\mathcal{W}_{\mathsf{Nat}}(X) \cong \mathcal{M}(X)$ where $\mathcal{M}(X)$ is a *multiset.*

Example 2.7 (Randomization). Probabilities $p \in [0,1] \subset \mathbb{R}$ form a partial semiring Prob $= \langle [0,1], +, \cdot, 0, 1\rangle$ where $+$ and $\cdot$ are real-valued arithmetic operations, but $+$ is undefined if $x+y > 1$ (just like in Example 2.5). This gives us $\mathcal{W}_{\mathsf{Prob}}(X) \cong \mathcal{D}(X)$, where $\mathcal{D}(X)$ is a probability sub-distribution (meaning that the mass can be less than 1 if some traces diverge).

Example 2.8 (Tropical Computation). The tropical semiring Tropical $= \langle [0,\infty], \min, +, \infty, 0\rangle$ uses real-valued weights, but $+$ is minimum and $\cdot$ is addition. Computations in $\mathcal{W}_{\mathsf{Tropical}}(X)$ therefore correspond to programs that choose the *cheapest* path for each outcome.

We will occasionally write $\mathcal{W}(X)$ instead of $\mathcal{W}_{\mathcal{A}}(X)$ when $\mathcal{A}$ is obvious. The semiring operations for addition, scalar multiplication, and zero are lifted pointwise to weighting functions as follows

$$(m_1 + m_2)(x) \triangleq m_1(x) + m_2(x) \qquad (u \cdot m)(x) \triangleq u \cdot m(x) \qquad \mathbb{0}(x) \triangleq \mathbb{0}$$

These lifted semiring operations give us a way to interpret branching, but we also need an interpretation for sequential composition. As is standard in program semantics with effects, we use a monad, which we define as a Klesli triple [42, 44].

Definition 2.9 (Kleisli Triple). A Kleisli triple $\langle T, \eta, (-)^\dagger\rangle$ in **Set** consists of a functor $T\colon \mathbf{Set} \to \mathbf{Set}$, and two morphisms $\eta : \mathsf{Id} \Rightarrow T$ and $(-)^\dagger \colon (X \to T(Y)) \to T(X) \to T(Y)$ such that:

$$\eta^\dagger = \mathsf{id} \qquad f^\dagger \circ \eta = f \qquad f^\dagger \circ g^\dagger = (f^\dagger \circ g)^\dagger$$

For any semiring $\mathcal{A}$, $\langle \mathcal{W}_{\mathcal{A}}, \eta, (-)^\dagger\rangle$ is a Kleisli triple where the operations η and $(-)^\dagger$ are defined below.

$$\eta(x)(y) \triangleq \begin{cases} \mathbb{1} & \text{if } x = y \\ \mathbb{0} & \text{if } x \neq y \end{cases} \qquad f^\dagger(m)(y) \triangleq \sum_{x \in \mathsf{supp}(m)} m(x) \cdot f(x)(y)$$

2.2 Denotational Semantics

We interpret the semantics of our language using the five-tuple $\langle \mathcal{A}, \Sigma, \mathsf{Act}, \mathsf{Test}, [\![\cdot]\!]_{\mathsf{Act}}\rangle$, where the components are:

1. $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ is a naturally ordered, complete, Scott continuous[1], partial semiring with a top element $\top \in U$ such that $\top \geq u$ for all $u \in U$.
2. Σ is the set of concrete program states.
3. Act is the set of atomic actions.
4. $\mathsf{Test} \subseteq 2^\Sigma$ is the set of primitive tests.
5. $[\![-]\!]_{\mathsf{Act}} : \mathsf{Act} \to \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma)$ is the semantic interpretation of atomic actions.

This definition is a generalized version of the one used in Outcome Separation Logic [58]. For example, we have dropped the requirement that $\top = \mathbb{1}$, meaning that we can capture more types of computation, such as the multiset model (Example 2.6).

Commands are interpreted as maps from states $\sigma \in \Sigma$ to weighting functions on states $[\![C]\!] : \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma)$, as shown in Figure 1. The first three commands are defined in terms of the monad and semiring operations (Definitions 2.2 and 2.3): skip uses η, sequential composition $C_1 \fatsemi C_2$ uses $(-)^\dagger$, and $C_1 + C_2$ uses the (lifted) semiring $+$. Since $+$ is partial, the semantics of $C_1 + C_2$ may be undefined. In Section 2.3, we discuss simple syntactic checks to ensure that the semantics is total. Atomic actions are interpreted using $[\![\cdot]\!]_{\mathsf{Act}}$.

[1]These concepts are defined formally in Appendix A.

$$\begin{aligned}
[\![\mathsf{skip}]\!](\sigma) &\triangleq \eta(\sigma) \\
[\![C_1 \fatsemi C_2]\!](\sigma) &\triangleq [\![C_2]\!]^\dagger([\![C_1]\!](\sigma)) \\
[\![C_1 + C_2]\!](\sigma) &\triangleq [\![C_1]\!](\sigma) + [\![C_2]\!](\sigma) \\
[\![a]\!](\sigma) &\triangleq [\![a]\!]_{\mathsf{Act}}(\sigma) \\
[\![\mathsf{assume}\ e]\!](\sigma) &\triangleq [\![e]\!](\sigma) \cdot \eta(\sigma) \\
\left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\sigma) &\triangleq \left(\mu f.\Phi_{\langle C,e,e'\rangle}(f)\right)(\sigma) \\
&\quad \text{where} \\
\Phi_{\langle C,e,e'\rangle}(f)(\sigma) &= [\![e]\!](\sigma) \cdot f^\dagger([\![C]\!](\sigma)) + [\![e']\!](\sigma) \cdot \eta(\sigma)
\end{aligned}$$

Figure 1: Denotational semantics for commands $[\![C]\!] : \Sigma \rightharpoonup \mathcal{W}_{\mathcal{A}}(\Sigma)$, given a partial semiring $\mathcal{A} = \langle X, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$, a set of program states Σ, atomic actions Act, primitive tests Test, and an interpretation of atomic actions $[\![a]\!]_{\mathsf{Act}} : \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma)$.

The interpretation of assume relies on the ability to interpret expressions and tests. We first describe the interpretation of tests, which maps tests b to the weights $\mathbb{0}$ or $\mathbb{1}$, that is $[\![b]\!]_{\mathsf{Test}} : \Sigma \to \{\mathbb{0}, \mathbb{1}\}$, with $\mathbb{0}$ representing false and $\mathbb{1}$ representing true, so $[\![\mathsf{false}]\!](\sigma) = \mathbb{0}$ and $[\![\mathsf{true}]\!](\sigma) = \mathbb{1}$. The operators $\wedge$, $\vee$, and $\neg$ are interpreted in the obvious ways, and for primitive tests $[\![t]\!](\sigma) = \mathbb{1}$ if $\sigma \in t$ otherwise $[\![t]\!](\sigma) = \mathbb{0}$. The full semantics of tests is given in Appendix A.1.

Since an expression is either a test or a weight, it remains only to describe the interpretation of weights, which is $[\![u]\!](\sigma) = u$ for any $u \in U$. So, assume e uses $[\![e]\!] : \Sigma \to U$ to obtain a program weight, and then scales the current state by it. If a test evaluates to false, then the weight of the branch is $\mathbb{0}$, so it is eliminated. If it evaluates to true, then the weight is scaled by $\mathbb{1}$—the identity of multiplication—so the weight is unchanged.

The iteration command continues with weight e and terminates with weight e'. We can attempt to define it recursively as follows.

$$\begin{aligned}
\left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\sigma) &= \left[\!\!\left[\mathsf{assume}\ e \fatsemi C \fatsemi C^{\langle e,e'\rangle} + \mathsf{assume}\ e'\right]\!\!\right](\sigma) \\
&= [\![e]\!](\sigma) \cdot \left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right]^\dagger([\![C]\!](\sigma)) + [\![e']\!](\sigma) \cdot \eta(\sigma)
\end{aligned}$$

In reality, we define the loop semantics as a least fixed point (denoted by the μ operator). Some additional requirements on the partial semiring (completeness and Scott continuity) ensure that this fixed point exists. For the full details, see Appendix A.

2.3 Syntactic Sugar for Total Programs

As mentioned in the previous section, the semantics of $C_1 + C_2$ and $C^{\langle e,e'\rangle}$ are not always defined given the partiality of the semiring $+$. The ways that $+$ can be used in programs depends on the particular semiring. However, regardless of the semiring, guarded choice (*i.e.,* if statements) are always valid, which we define as syntactic sugar.

$$\mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \triangleq (\mathsf{assume}\ b \fatsemi C_1) + (\mathsf{assume}\ \neg b \fatsemi C_2)$$

Since Bool, Nat, and Tropical are total semirings, unguarded choice is always valid in those execution models. In the probabilistic case, choice can be used as long as the sum of the weights of both branches is at most 1. One way to achieve this is to weight one branch by a probability $p \in [0,1]$ and the other branch by $1-p$, a biased coin-flip. We provide syntactic sugar for that operation:

$$C_1 +_p C_2 \triangleq (\mathsf{assume}\ p \fatsemi C_1) + (\mathsf{assume}\ 1-p \fatsemi C_2)$$

We also provide syntactic sugar for iterating constructs below.

$$\mathsf{while}\ b\ \mathsf{do}\ C \triangleq C^{\langle b, \neg b\rangle} \qquad C^\star \triangleq C^{\langle \mathbb{1}, \mathbb{1}\rangle} \qquad C^{\langle p\rangle} \triangleq C^{\langle p, 1-p\rangle}$$

While loops use a test to determine whether iteration should continue, making them deterministic. The Kleene star $C^\star$ is defined for interpretations based on total semirings only; it iterates C nondeterministically many times.[2]

Finally, the probabilistic iterator $C^{\langle p\rangle}$ continues to execute with probability p and exits with probability $1-p$. This behavior can be replicated using a while loop and auxiliary variables, but adding state complicates reasoning about the programs and precludes, *e.g.*, devising equational theories over uninterpreted atomic commands [54]. This construct—which was not included in previous Outcome Logic work—is therefore advantageous.

In Appendix A, we prove that programs constructed using appropriate syntax have total semantics. For the remainder of the paper, we assume that programs are constructed in this way, and are thus always well-defined.

3 OUTCOME LOGIC

In this section, we define Outcome Logic, and show how it relates to Hoare Logic and Dynamic Logic [49].

3.1 Outcome Assertions

Outcome assertions are the basis for expressing pre- and postconditions in Outcome Logic. Unlike pre- and postconditions of Hoare Logic—which can only describe *individual program states*—outcome assertions expose the program weights from Section 2.1 to enable reasoning about branching and the weights of reachable outcomes.

We represent these assertions semantically; outcome assertions $\varphi, \psi \in 2^{\mathcal{W}_{\mathcal{A}}(\Sigma)}$ are the sets of weighted collections of program states representing their true assignments. For any $m \in \mathcal{W}_{\mathcal{A}}(\Sigma)$, we write $m \vDash \varphi$ (m satisfies φ) to mean that $m \in \varphi$.

The use of semantic assertions allows us to focus on the rules of inference pertaining to the structure of programs, showing that the proof system is sufficient for all practical purposes. No program logic is truly complete, as analyzing loops inevitably reduces to the (undecidable) halting problem [1, 15]. It is well known that in order to express intermediate assertions and loop invariants, the assertion language must at least contain Peano arithmetic [40], making the details of this language somewhat uninteresting. As a result, many modern developments such as Separation Logic [11, 55], Incorrectness Logic [47], Iris [31, 32], probabilistic Hoare-style logics [3, 33], and others [2, 16, 17, 52] use semantic assertions.

We will now define useful notation for common assertions, which are also repeated in Figure 2. For example $\top$ (always true) is the set of all weighted collections, $\bot$ (always false) is the empty set, and logical negation is the set complement.

$$\top \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma) \qquad \bot \triangleq \emptyset \qquad \neg\varphi \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma) \setminus \varphi$$

[2] In nondeterministic languages, while b do $C \equiv (\mathsf{assume}\ b \fatsemi C)^\star \fatsemi \mathsf{assume}\ \neg b$, however this encoding does not work in general since $(\mathsf{assume}\ b \fatsemi C)^\star$ is not a well-defined program when using a partial semiring (*e.g.*, Examples 2.5 and 2.7).

$$\begin{array}{ll} \top \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma) & \neg\varphi \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma) \setminus \varphi \\ \bot \triangleq \emptyset & \varphi \Rightarrow \psi \triangleq (\mathcal{W}_{\mathcal{A}}(\Sigma) \setminus \varphi) \cup \psi \\ \varphi \vee \psi \triangleq \varphi \cup \psi & \mathbf{1}_m \triangleq \{m\} \\ \varphi \wedge \psi \triangleq \varphi \cap \psi & \exists x : T. \phi(x) \triangleq \bigcup_{t \in T} \phi(t) \end{array}$$

$$\varphi \oplus \psi \triangleq \{m_1 + m_2 \mid m_1 \in \varphi, m_2 \in \psi\}$$
$$\varphi^{(u)} \triangleq \{u \cdot m \mid m \in \varphi\} \cup \{\mathbb{0} \mid u = \mathbb{0}\}$$
$$\ulcorner P \urcorner \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma) \mid |m| = \mathbb{1}, \mathsf{supp}(m) \subseteq P\}$$

Figure 2: Outcome assertion semantics, given a partial semiring $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ where $u \in U$, $\phi\colon T \to 2^{\mathcal{W}_{\mathcal{A}}(\Sigma)}$, and $P \in 2^\Sigma$.

Disjunction, conjunction, and implication are defined as usual:

$$\varphi \vee \psi \triangleq \varphi \cup \psi \qquad \varphi \wedge \psi \triangleq \varphi \cap \psi \qquad \varphi \Rightarrow \psi \triangleq (\mathcal{W}_{\mathcal{A}}(\Sigma) \setminus \varphi) \cup \psi$$

Given a predicate $\phi\colon T \to 2^{\mathcal{W}_{\mathcal{A}}(\Sigma)}$ on some (possibly infinite) set T, existential quantification over T is the union of $\phi(t)$ for all $t \in T$, meaning it is true iff there is some $t \in T$ that makes $\phi(t)$ true.

$$\exists x : T.\ \phi(x) \triangleq \bigcup_{t \in T} \phi(t)$$

Next, we define assertions based on the operations of the semiring $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$. The *outcome conjunction* $\varphi \oplus \psi$ asserts that the collection of outcomes m can be split into two parts $m = m_1 + m_2$ such that φ holds in m_1 and ψ holds in m_2[3].

$$\varphi \oplus \psi \triangleq \{m_1 + m_2 \mid m_1 \in \varphi, m_2 \in \psi\}$$

For example, in the nondeterministic interpretation, we can view m_1 and m_2 as sets (not necessarily disjoint), such that $m = m_1 \cup m_2$, so φ and ψ each describe subsets of the reachable states.

The weighting operation $\varphi^{(u)}$ means that φ occurs with weight u, where $u \in U$ is a literal weight. We also ensure that $\mathbb{0} \in \varphi^{(\mathbb{0})}$ so that $\varphi \oplus \psi^{(\mathbb{0})} \Leftrightarrow \varphi$ even if ψ is unsatisfiable.

$$\varphi^{(u)} \triangleq \{u \cdot m \mid m \in \varphi\} \cup \{\mathbb{0} \mid u = \mathbb{0}\}$$

Finally, we provide a way to lift atomic assertions $P \subseteq \Sigma$ describing some subset of the program states. When lifted to be an outcome assertion, $\ulcorner P \urcorner$ must cover all the reachable states ($\mathsf{supp}(m) \subseteq P$). We also require that $|m| = \mathbb{1}$. In the nondeterministic case (Example 2.4), this simply means that $m \neq \emptyset$, and so P is non-vacuously satisfied. In the probabilistic case (Example 2.7), this means that the probability of P occurring is exactly 1. It also means that in $\ulcorner P \urcorner^{(p)}$, the probability of P occurring is exactly p and that $\eta(\sigma) \vDash \ulcorner P \urcorner$ for any $\sigma \in P$.

$$\ulcorner P \urcorner \triangleq \{m \mid |m| = \mathbb{1}, \mathsf{supp}(m) \subseteq P\}$$

We will often omit the lifting brackets $\ulcorner - \urcorner$ and simply write $\langle P \rangle\ C\ \langle Q \rangle$. We also permit the use of tests b in triples. For instance, the precondition of $\langle P \wedge b \rangle\ C\ \langle Q \rangle$ is the set:

$$\{m \in \mathcal{W}(\Sigma) \mid |m| = \mathbb{1}, \forall \sigma \in \mathsf{supp}(m).\ \sigma \in P \wedge [\![b]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1}\}$$

[3] We remark that $\oplus$ is semantically equivalent to the separating conjunction ($*$) from the logic of Bunched Implications (BI) [48], but a deeper exploration is out of scope.

There is a close connection between the $\oplus$ of outcome assertions and the choice operator $C_1 + C_2$ for programs. If P is an assertion describing the outcome of C_1 and Q describes the outcome of C_2, then $P \oplus Q$ describes the outcome of $C_1 + C_2$ by stating that both P and Q are reachable outcomes via a non-vacuous program trace. This is more expressive than using the disjunction $P \vee Q$, since the disjunction does not guarantee that *both* P and Q are reachable. Suppose P describes a desirable program outcome whereas Q describes an erroneous one; then $P \oplus Q$ tells us that the program has a bug (it can reach an error state) whereas $P \vee Q$ is not strong enough to make this determination [57].

Similar to the syntactic sugar for probabilistic programs in Section 2.3, we let $\varphi \oplus_p \psi \triangleq \varphi^{(p)} \oplus \psi^{(1-p)}$. If P and Q are the results of running C_1 and C_2, then $P \oplus_p Q$—meaning that P occurs with probability p and Q occurs with probability $1-p$—is the result of running $C_1 +_p C_2$.

3.2 Outcome Triples

Inspired by Hoare Logic, Outcome Triples $\langle\varphi\rangle\ C\ \langle\psi\rangle$ specify program behavior in terms of pre- and postconditions [57]. The difference is that Outcome Logic describes weighted collections of states as opposed to Hoare Logic, which can only describe individual states. We write $\vDash \langle\varphi\rangle\ C\ \langle\psi\rangle$ to mean that a triple is semantically valid, as defined below.

Definition 3.1 (Outcome Triples). Given $\langle \mathcal{A}, \Sigma, \mathsf{Act}, \mathsf{Test}, [\![\cdot]\!]_{\mathsf{Act}}\rangle$, the semantics of outcome triples is defined as follows:

$$\vDash \langle\varphi\rangle\ C\ \langle\psi\rangle \quad \text{iff} \quad \forall m \in \mathcal{W}_{\mathcal{A}}(\Sigma).\ m \vDash \varphi \implies [\![C]\!]^\dagger(m) \vDash \psi$$

Informally, $\langle\varphi\rangle\ C\ \langle\psi\rangle$ is valid if the result of running the program C on a weighted collection of states satisfying φ satisfies ψ. The power to describe the collection of states in the postcondition means that Outcome Logic can express many types of properties including reachability ($P \oplus Q$), probabilities ($P \oplus_p Q$), and nontermination (the lack of outcomes, $\top^{(\mathbb{0})}$). Next, we will see how Outcome Logic can be used to encode familiar program logics.

3.3 Dynamic Logic and Hoare Logic

Outcome Logic, in its full generality, allows one to quantify the precise weights of each outcome. Nevertheless, many common program logics do not provide this much power, which can be advantageous as they offer simplified reasoning principles—Hoare Logic's loop Invariant rule (Section 5.3) is considerably simpler than the While rule needed for general Outcome Logic (Section 5.2).

In this section, we devise an assertion syntax in order to show the connections between Outcome Logic and Hoare Logic. We take inspiration from modal logic and Dynamic Logic [49], using the modalities $\Box$ and $\Diamond$ to express that assertions always or sometimes occur, respectively. We encode these modalities using the operations from Section 3.1, where U is the set of semiring weights.

$$\begin{array}{lcll} \Box P & \triangleq & \exists u : U.P^{(u)} & = \{m \mid \mathrm{supp}(m) \subseteq P\} \\ \Diamond P & \triangleq & \exists u : (U \setminus \{\mathbb{0}\}).P^{(u)} \oplus \top & = \{m \mid \mathrm{supp}(m) \cap P \neq \emptyset\} \end{array}$$

We define $\Box P$ to mean that P occurs with some weight, so $m \vDash \Box P$ exactly when $\mathrm{supp}(m) \subseteq P$. Dually, $\Diamond P$ requires that P has nonzero weight and the $- \oplus \top$ allows there to be other elements in the support too. So, $m \vDash \Diamond P$ when $\sigma \vDash P$ for some $\sigma \in \mathrm{supp}(m)$. It is relatively easy to see that these two modalities are De Morgan duals, that is $\Box P \Leftrightarrow \neg\Diamond\neg P$ and $\Diamond P \Leftrightarrow \neg\Box\neg P$.

For Boolean-valued semirings (Examples 2.4 and 2.5), we get that $\Box P = P^{(0)} \vee P^{(1)}$. Only $\mathbb{0}$ satisfies $P^{(0)}$, indicating that the program diverged (let us call this assertion div), and $P^{(1)}$ is equivalent to P. So, $\Box P = P \vee \mathrm{div}$, meaning that either P covers all the reachable outcomes, or the program diverged ($\Box$ will be useful for expressing partial correctness). Similarly, $\Diamond P = P \oplus \top$, meaning that P is one of the (possibly many) reachable outcomes.

Now, we are going to use these modalities to show that Outcome Logic subsumes other program logics. We start with nondeterministic, partial correctness Hoare Logic, where the meaning of the triple $\{P\}\ C\ \{Q\}$ is that any state resulting from running the program C on a state satisfying P must satisfy Q. There are many equivalent ways to formally define the semantics of Hoare Logic; we will use a characterization based on Dynamic Logic [49], which is inspired by modal logic in that it defines modalities similar to $\Box$ and $\Diamond$.

$$[C]Q = \{\sigma \mid [\![C]\!](\sigma) \subseteq Q\} \qquad \langle C\rangle Q = \{\sigma \mid [\![C]\!](\sigma) \cap Q \neq \emptyset\}$$

That is, $[C]Q$ is an assertion stating that Q must hold after running the program C (if it terminates). In the predicate transformer literature, $[C]Q$ is called the weakest liberal precondition [21, 22]. The dual modality $\langle C\rangle Q$ states that Q might hold after running C (sometimes referred to as the weakest possible precondition [30, 45]).

A Hoare Triple $\{P\}\ C\ \{Q\}$ is valid iff $P \subseteq [C]Q$, so to show that Outcome Logic subsumes Hoare Logic, it suffices to prove that we can express $P \subseteq [C]Q$. We do so using the $\Box$ modality defined previously. More precisely, we capture Hoare Triples as follows.

Theorem 3.2 (Subsumption of Hoare Logic).

$$\vDash \langle P\rangle\ C\ \langle\Box Q\rangle \quad \text{iff} \quad P \subseteq [C]Q \quad \text{iff} \quad \vDash \{P\}\ C\ \{Q\}$$

While it has already been shown that Outcome Logic subsumes Hoare Logic [57], our characterization is not tied to nondeterminism; the triple $\langle P\rangle\ C\ \langle\Box Q\rangle$ does not necessarily have to be interpreted in a nondeterministic way, but can rather be taken to mean that running C in a state satisfying P results in Q covering all the terminating traces with some weight. When we later develop rules for reasoning about loops using invariants (Section 5), those techniques will be applicable to *any* instance of Outcome Logic.

Given that the formula $P \subseteq [C]Q$ gives rise to a meaningful program logic, it is natural to ask whether the same is true for $P \subseteq \langle C\rangle Q$. In fact, this formula is colloquially known as Lisbon Logic (it was proposed during a meeting in Lisbon as a possible foundation for incorrectness reasoning [45, 47, 57]). The semantics of Lisbon triples, denoted $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$, is that for any start state satisfying P, there exists a state resulting from running C that satisfies Q. Given that Q only covers a subset of the outcomes, it is not typically suitable for correctness, however it is useful for incorrectness as some bugs only occur some of the time.

Theorem 3.3 (Subsumption of Lisbon Logic).

$$\vDash \langle P\rangle\ C\ \langle\Diamond Q\rangle \quad \text{iff} \quad P \subseteq \langle C\rangle Q \quad \text{iff} \quad \vDash \{\!|P|\!\}\ C\ \{\!|Q|\!\}$$

In the following section, we will see a complete proof system for Outcome Logic and, given that we have just shown that Outcome Logic subsumes Hoare and Lisbon Logic, it will allow us to derive any specification in those two logics as well. However, given the

Commands

$$\frac{}{\langle \varphi \rangle \ \mathsf{skip} \ \langle \varphi \rangle}\textsc{Skip} \qquad \frac{\langle \varphi \rangle \ C_1 \ \langle \vartheta \rangle \qquad \langle \vartheta \rangle \ C_2 \ \langle \psi \rangle}{\langle \varphi \rangle \ C_1 \fatsemi C_2 \ \langle \psi \rangle}\textsc{Seq} \qquad \frac{\langle \varphi \rangle \ C_1 \ \langle \psi_1 \rangle \qquad \langle \varphi \rangle \ C_2 \ \langle \psi_2 \rangle}{\langle \varphi \rangle \ C_1 + C_2 \ \langle \psi_1 \oplus \psi_2 \rangle}\textsc{Plus}$$

$$\frac{\varphi \vDash e = u}{\langle \varphi \rangle \ \mathsf{assume}\ e \ \langle \varphi^{(u)} \rangle}\textsc{Assume} \qquad \frac{(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty \qquad \forall n \in \mathbb{N}. \quad \langle \varphi_n \rangle \ \mathsf{assume}\ e \fatsemi C \ \langle \varphi_{n+1} \rangle \qquad \langle \varphi_n \rangle \ \mathsf{assume}\ e' \ \langle \psi_n \rangle}{\langle \varphi_0 \rangle \ C^{\langle e, e' \rangle} \ \langle \psi_\infty \rangle}\textsc{Iter}$$

Structural Rules

$$\frac{}{\langle \bot \rangle \ C \ \langle \varphi \rangle}\textsc{False} \qquad \frac{}{\langle \varphi \rangle \ C \ \langle \top \rangle}\textsc{True} \qquad \frac{\langle \varphi \rangle \ C \ \langle \psi \rangle}{\langle \varphi^{(u)} \rangle \ C \ \langle \psi^{(u)} \rangle}\textsc{Scale}$$

$$\frac{\langle \varphi_1 \rangle \ C \ \langle \psi_1 \rangle \qquad \langle \varphi_2 \rangle \ C \ \langle \psi_2 \rangle}{\langle \varphi_1 \vee \varphi_2 \rangle \ C \ \langle \psi_1 \vee \psi_2 \rangle}\textsc{Disj} \qquad \frac{\langle \varphi_1 \rangle \ C \ \langle \psi_1 \rangle \qquad \langle \varphi_2 \rangle \ C \ \langle \psi_2 \rangle}{\langle \varphi_1 \wedge \varphi_2 \rangle \ C \ \langle \psi_1 \wedge \psi_2 \rangle}\textsc{Conj} \qquad \frac{\langle \varphi_1 \rangle \ C \ \langle \psi_1 \rangle \qquad \langle \varphi_2 \rangle \ C \ \langle \psi_2 \rangle}{\langle \varphi_1 \oplus \varphi_2 \rangle \ C \ \langle \psi_1 \oplus \psi_2 \rangle}\textsc{Choice}$$

$$\frac{\forall t \in T. \quad \langle \phi(t) \rangle \ C \ \langle \phi'(t) \rangle}{\langle \exists x : T.\ \phi(x) \rangle \ C \ \langle \exists x : T.\ \phi'(x) \rangle}\textsc{Exists} \qquad \frac{\varphi' \Rightarrow \varphi \qquad \langle \varphi \rangle \ C \ \langle \psi \rangle \qquad \psi \Rightarrow \psi'}{\langle \varphi' \rangle \ C \ \langle \psi' \rangle}\textsc{Consequence}$$

Figure 3: Rules of inference for Outcome Logic

generality of Outcome Logic, some of the proof rules are not ergonomic for use in less expressive variants. Later, in Section 5, we show how we can derive simpler rules, for example, to analyze loops using invariants (Hoare Logic) or variants (Lisbon Logic).

4 PROOF THEORY

We now describe the Outcome Logic rules of inference, which are shown in Figure 3. The rules are split into three categories.

Standard Commands. The rules for standard (non-looping) commands mostly resemble those of Hoare Logic. The Skip rule stipulates that the precondition is preserved after running a no-op. Seq derives a specification for a sequential composition from two subderivations for each command. Similarly, Plus joins the derivations of two program branches using an outcome conjunction.

Assume has a side condition that $\varphi \vDash e = u$, where $u \in U$ is a semiring element. Informally, this means that the precondition entails that the expression e is some concrete weight u. More formally, it is defined as follows:

$$\varphi \vDash e = u \qquad \text{iff} \qquad \forall m \in \varphi. \quad \forall \sigma \in \mathsf{supp}(m). \quad [\![e]\!](\sigma) = u$$

If e is a weight literal, then $\varphi \vDash u = u$ vacuously holds, so the rule can be simplified to $\vdash \langle \varphi \rangle \ \mathsf{assume}\ u \ \langle \varphi^{(u)} \rangle$, but if it is a test b, then φ must contain enough information to conclude that b is true or false. Additional rules to decide $\varphi \vDash e = u$ are given in Appendix C.

Iteration. The Iter rule uses two families of predicates: φ_n represents the result of n iterations of $\mathsf{assume}\ e \fatsemi C$ and ψ_n is the result of iterating n times and then weighting the result by e', so $\bigoplus_{n\in\mathbb{N}} \psi_n$ represents all the terminating traces. To avoid the infinitary outcome conjunction, we instead use the assertion ψ_∞, which must have the following property.

Definition 4.1 (Converging Assertions). A family $(\psi_n)_{n\in\mathbb{N}}$ *converges* (written $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$) iff for any collection $(m_n)_{n\in\mathbb{N}}$, if $m_n \vDash \psi_n$ for each $n \in \mathbb{N}$, then $\sum_{n\in\mathbb{N}} m_n \vDash \psi_\infty$.

Structural Rules. We also give additional rules that are not dependent on the program command. This includes rules for trivial pre- and postconditions (True and False), scaling by a weight (Scale), combining subderivations using logical connectives (Disj, Conj, and Choice), introducing existential quantification (Exists), and weakening (Consequence). Note that the implications in the rule of Consequence are semantic ones: $\varphi' \Rightarrow \varphi$ iff $\varphi' \subseteq \varphi$. We do not explore the proof theory for outcome assertions, although it has been done for similar logics [25, 48].

4.1 Soundness and Relative Completeness

Soundness of the Outcome Logic proof system means that any derivable triple (using the inference rules in Figure 3 and axioms about atomic actions) is semantically valid. We write $\Gamma \vdash \langle \varphi \rangle \ C \ \langle \psi \rangle$ to mean that $\langle \varphi \rangle \ C \ \langle \psi \rangle$ is derivable given a collection of axioms $\Gamma = \langle \varphi_1 \rangle \ a_1 \ \langle \psi_1 \rangle, \ldots, \langle \varphi_n \rangle \ a_n \ \langle \psi_n \rangle$. Let Ω consist of all triples $\langle \varphi \rangle \ a \ \langle \psi \rangle$ such that $a \in \mathsf{Act}$, and $\vDash \langle \varphi \rangle \ a \ \langle \psi \rangle$ (all the true statements about atomic actions). We also presume that the program C is well-formed as described in Section 2.3. The soundness theorem is stated formally below.

Theorem 4.2 (Soundness). $\Omega \vdash \langle \varphi \rangle \ C \ \langle \psi \rangle \implies \vDash \langle \varphi \rangle \ C \ \langle \psi \rangle$

The full proof is shown in Appendix C and proceeds by induction on the structure of the derivation $\Omega \vdash \langle \varphi \rangle \ C \ \langle \psi \rangle$, with cases in which each rule is the last inference. Most of the cases are straightforward, but the following lemma is needed to justify the soundness of the Iter case, where $C^0 = \mathsf{skip}$ and $C^{n+1} = C^n \fatsemi C$.

Lemma 4.3. *The following equation holds:*

$$\left[\!\!\left[C^{\langle e, e' \rangle} \right]\!\!\right](\sigma) = \sum_{n\in\mathbb{N}} [\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!](\sigma)$$

Completeness—the converse of soundness—tells us that our inference rules are sufficient to deduce any true statement about a program. As is typical, Outcome Logic is *relatively* complete, meaning that proving any valid triple can be reduced to implications

$\varphi \Rightarrow \psi$ in the assertion language. For OL instances involving state (and Hoare Logic), those implications are undecidable since they must, at the very least, encode Peano arithmetic [1, 15, 40].

The first step is to show that given any program C and precondition φ, we can derive the triple $\langle \varphi \rangle\ C\ \langle \psi \rangle$, where ψ is the *strongest postcondition* [23], *i.e.,* the strongest assertion making that triple true. As defined below, ψ is exactly the set resulting from evaluating C on each $m \in \varphi$. The proceeding lemma shows that the triple with the strongest postcondition is derivable.

Definition 4.4 (Strongest Postcondition).

$$\mathrm{post}(C, \varphi) \triangleq \{\llbracket C \rrbracket^\dagger(m) \mid m \in \varphi\}$$

Lemma 4.5. $\Omega \vdash \langle \varphi \rangle\ C\ \langle \mathrm{post}(C, \varphi) \rangle$

The proof is by induction on the structure of the program, and is shown in its entirety in Appendix C. The cases for skip and $C_1 ⨟ C_2$ are straightforward, but the other cases are more challenging and involve existential quantification. To give an intuition as to why existentials are needed, let us examine an example involving branching. We use a concrete instance of Outcome Logic with variable assignment (formalized in Section 6).

Consider the program skip $+ (x \coloneqq x + 1)$ and the precondition $x \geq 0$. It is tempting to say that post is obtained compositionally by joining the post of the two branches using $\oplus$:

$$\begin{aligned}
&\mathrm{post}(\mathrm{skip} + (x \coloneqq x + 1),\ x \geq 0) \\
&\quad = \mathrm{post}(\mathrm{skip}, x \geq 0) \oplus \mathrm{post}(x \coloneqq x + 1,\ x \geq 0) \\
&\quad = (x \geq 0) \oplus (x \geq 1)
\end{aligned}$$

However, that is incorrect. While it is a valid postcondition, it is not the strongest one because it does not account for the relationship between the values of x in the two branches; if $x = n$ in the first branch, then it must be $n+1$ in the second branch. A second attempt could use existential quantification to dictate that relationship.

$$\exists n : \mathbb{N}.\ (x = n) \oplus (x = n + 1)$$

Unfortunately, that is also incorrect; it does not properly account for the fact that that precondition $x \geq 0$ may be satisfied by a *set* of states in which x has many different values—the existential quantifier requires that x takes on a single value in all the initial outcomes. The solution is to quantify over the collections $m \in \varphi$ satisfying the precondition, and then to take the post of $\mathbf{1}_m = \{m\}$.

$$\mathrm{post}(C_1 + C_2, \varphi) = \exists m : \varphi.\ \mathrm{post}(C_1, \mathbf{1}_m) \oplus \mathrm{post}(C_2, \mathbf{1}_m)$$

While it may seem unwieldy that the strongest post is hard to characterize even in this seemingly innocuous example, the same problem arises in logics for probabilistic [3, 19] and hyper-property [17] reasoning, both of which are instances of OL. Although the *strongest* postcondition is quite complicated, something weaker suffices in most cases. We will later see how rules for those simpler cases are derived (Section 5) and used (Sections 7 and 8).

The main result is now a straightforward corollary of Lemma 4.5 using the rule of Consequence, since any valid postcondition is implied by the strongest one.

Theorem 4.6 (Relative Completeness).

$$\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle \implies \Omega \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$$

Proof. We first establish that $\mathrm{post}(C, \varphi) \Rightarrow \psi$. Suppose that $m \in \mathrm{post}(C, \varphi)$. That means that there must be some $m' \in \varphi$ such that $m = \llbracket C \rrbracket^\dagger(m')$. Using $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$, we get that $m \vDash \psi$. Now, we complete the derivation as follows:

$$\dfrac{\dfrac{\Omega}{\langle \varphi \rangle\ C\ \langle \mathrm{post}(C, \varphi) \rangle}\ \textsc{Lemma 4.5} \qquad \mathrm{post}(C, \varphi) \Rightarrow \psi}{\langle \varphi \rangle\ C\ \langle \psi \rangle}\ \textsc{Consequence}$$

□

5 DERIVED RULES

We now present derived rules for simplified reasoning involving if statements, while loops, and the encodings of Hoare and Lisbon Logic from Section 3.3. Recall that Hoare triples $\{P\}\ C\ \{Q\}$ are semantically equivalent to $\langle P \rangle\ C\ \langle \square Q \rangle$ in Outcome Logic (Theorem 3.2) and Lisbon triples $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$ are equivalent to $\langle P \rangle\ C\ \langle \diamond Q \rangle$ (Theorem 3.3). For the full derivations, refer to Appendix D.

5.1 Sequencing in Hoare and Lisbon Logic

The Seq rule requires that the postcondition of the first command exactly matches the precondition of the next. This is at odds with our encodings of Hoare and Lisbon Logic, which have asymmetry between the modalities used in the pre- and postconditions. Still, sequencing is possible using derived rules.

$$\dfrac{\langle P \rangle\ C_1\ \langle \square Q \rangle \qquad \langle Q \rangle\ C_2\ \langle \square R \rangle}{\langle P \rangle\ C_1 ⨟ C_2\ \langle \square R \rangle}\ \textsc{Seq (Hoare)}$$

$$\dfrac{\langle P \rangle\ C_1\ \langle \diamond Q \rangle \qquad \langle Q \rangle\ C_2\ \langle \diamond R \rangle}{\langle P \rangle\ C_1 ⨟ C_2\ \langle \diamond R \rangle}\ \textsc{Seq (Lisbon)}$$

Since both $\square$ and $\diamond$ are encoded using existential quantifiers, the derivations (Appendix D.1) use Scale and Exists to conclude:

$$\begin{aligned}
\langle Q \rangle\ C_2\ \langle \square R \rangle \quad &\vdash \quad \langle Q^{(v)} \rangle\ C_2\ \langle \exists u : U. R^{(u \cdot v)} \rangle \\
&\vdash \quad \langle \exists v : U. Q^{(v)} \rangle\ C_2\ \langle \exists v, u : U. R^{(u \cdot v)} \rangle \\
&\vdash \quad \langle \square Q \rangle\ C_2\ \langle \square R \rangle
\end{aligned}$$

The case for $\diamond$ is similar, also making use of the fact that $- \oplus \top$ is idempotent ($(R \oplus \top) \oplus \top \Leftrightarrow R \oplus \top$). Lisbon Logic adds an additional requirement on the semiring; $\mathbb{0}$ must be the *unique* annihilator of multiplication ($u \cdot v = \mathbb{0}$ iff $u = \mathbb{0}$ or $v = \mathbb{0}$), which ensures that a finite sequence of commands does not eventually cause a branch to have zero weight. Examples 2.4 to 2.8 all obey this property.

5.2 If Statements and While Loops

Recall from Section 2.3 that we encode if statements and while loops using the choice and iteration constructs. We now derive convenient inference rules for those cases. If statements are defined as (assume $b ⨟ C_1$) + (assume $\neg b ⨟ C_2$). Reasoning about them generally requires the precondition to be separated into two parts, φ_1 and φ_2, representing the collections of states in which b is true and false, respectively. This may require—*e.g.,* in the probabilistic case—that φ_1 and φ_2 quantify the weight (likelihood) of the guard.

If it is possible to separate the precondition in that way, then φ_1 and φ_2 act as the preconditions for C_1 and C_2, respectively, and the overall postcondition is an outcome conjunction of the results of both branches.

$$\frac{\varphi_1 \vDash b \qquad \langle \varphi_1 \rangle\ C_1\ \langle \psi_1 \rangle \qquad \varphi_2 \vDash \neg b \qquad \langle \varphi_2 \rangle\ C_2\ \langle \psi_2 \rangle}{\langle \varphi_1 \oplus \varphi_2 \rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \psi_2 \oplus \psi_2 \rangle}\textsc{If}$$

From If, we derive the familiar rules for Hoare and Lisbon Logic.

$$\frac{\langle P \land b \rangle\ C_1\ \langle \Box Q \rangle \quad \langle P \land \neg b \rangle\ C_2\ \langle \Box Q \rangle}{\langle P \rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Box Q \rangle}\textsc{If (Hoare)}$$

$$\frac{\langle P \land b \rangle\ C_1\ \langle \Diamond Q \rangle \quad \langle P \land \neg b \rangle\ C_2\ \langle \Diamond Q \rangle}{\langle P \rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q \rangle}\textsc{If (Lisbon)}$$

The derivations rely on the fact that if P holds, then there exist u and v such that $u + v = \mathbb{1}$ and $(P \land b)^{(u)} \oplus (P \land \neg b)^{(v)}$ holds. We complete the proof using Scale and Split, and existentially quantify the new weights using $\Box$ and $\Diamond$.

The rule for while loops is slightly simplified compared to Iter, as it only generates a proof obligation for a single triple instead of two. There are still two families of assertions, but φ_n now represents the portion of the program configuration where the guard b is true, and ψ_n represents the portion where it is false. So, on each iteration, φ_n continues to evaluate and ψ_n exits; the final postcondition ψ_∞ is an aggregation of all the terminating traces.

$$\frac{(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty \quad \langle \varphi_n \rangle\ C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle \quad \varphi_n \vDash b \quad \psi_n \vDash \neg b}{\langle \varphi_0 \oplus \psi_0 \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \psi_\infty \rangle}\textsc{While}$$

This While rule is similar to those found in probabilistic Hoare Logics [3, 20].

5.3 Loop Invariants

Loop invariants are a popular analysis technique in partial correctness logics. The idea is to find an invariant P that is preserved by the loop body and therefore must remain true when—*and if*—the loop terminates. Because loop invariants are unable to guarantee termination, the Outcome Logic rule must indicate that the program may diverge. We achieve this using the $\Box$ modality from Section 3.3. The rule for Outcome Logic loop invariants is as follows:

$$\frac{\langle P \land b \rangle\ C\ \langle \Box P \rangle}{\langle P \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Box (P \land \neg b) \rangle}\textsc{Invariant}$$

This rule states that if the program starts in a state described by P, which is also preserved by each execution of the loop, then $P \land \neg b$ is true of every reachable end state. If the program diverges and there are no reachable end states, then $\Box(P \land \neg b)$ is vacuously satisfied, just like in Hoare Logic.

Invariant is derived using the While rule with $\varphi_n = \Box(P \land b)$ and $\psi_n = \Box(P \land \neg b)$. To show $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$, first note that $m_n \vDash \Box(P \land \neg b)$ simply means that $\mathrm{supp}(m_n) \subseteq (P \land \neg b)$. Since this is true for all $n \in \mathbb{N}$, then all the reachable states satisfy $P \land \neg b$.

In Section 3.3 we used $\langle P \rangle\ C\ \langle \Box Q \rangle$ to encode nondeterministic Hoare Logic, but the Invariant rule applies to all instances of Outcome Logic. For example, this rule can be used for probabilistic programs to state that $P \land \neg b$ covers all the terminating outcomes, and occurs with some probability.

It is well known that Skip, Seq (Hoare), If (Hoare), Invariant, and Consequence constitute a complete proof system for Hoare Logic [15, 38]. It follows that these rules are also complete for deriving any Outcome Logic triples of the form $\langle P \rangle\ C\ \langle \Box Q \rangle$, avoiding the more complex machinery of Lemma 4.5[4].

5.4 Loop Variants

Loop variants are an alternative way to reason about loops when termination guarantees are needed. They were first studied in the context of total Hoare Logic [43], but are also used in other logics that require termination guarantees such as Reverse Hoare Logic [18], Incorrectness Logic [47], and Lisbon Logic [2, 45, 52].[5]

Rather than using an invariant that is preserved by the loop body, we now use a family of changing *variants* $(\varphi_n)_{n\in\mathbb{N}}$ such that φ_n implies that the loop guard b is true for all $n > 0$, and φ_0 implies that it is false, guaranteeing that the loop exits. The inference rule is shown below, and states that starting at some φ_n, the execution will eventually count down to φ_0, at which point it terminates.

$$\frac{\forall n \in \mathbb{N}. \qquad \varphi_0 \vDash \neg b \qquad \varphi_{n+1} \vDash b \qquad \langle \varphi_{n+1} \rangle\ C\ \langle \varphi_n \rangle}{\langle \exists n : \mathbb{N}.\varphi_n \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \varphi_0 \rangle}\textsc{Variant}$$

Since the premise guarantees termination after precisely n steps, it is easy to establish convergence—the postcondition only consists of a single trace.

Although loop variants are valid in any Outcome Logic instance, they require loops to be *deterministic*—the loop executes for the same number of iterations regardless of any computational effects that occur in the body. Examples of such scenarios include for loops, where the number of iterations is fixed upfront.

We also present a more flexible loop variant rule geared towards Lisbon triples. In this case, we use the $\Diamond$ modality to only require that *some* trace is moving towards termination.

$$\frac{\forall n \in \mathbb{N}. \quad P_0 \vDash \neg b \quad P_{n+1} \vDash b \quad \langle P_{n+1} \rangle\ C\ \langle \Diamond P_n \rangle}{\langle \exists n : \mathbb{N}.P_n \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Diamond P_0 \rangle}\textsc{Lisbon Variant}$$

In other words, Lisbon Variant witnesses a single terminating trace. As such, it does not require the lockstep termination of all outcomes like Variant does.

6 ADDING VARIABLES AND STATE

We now develop a concrete Outcome Logic instance with variable assignment as atomic actions. Let Var be a countable set of variable names and $\mathsf{Val} = \mathbb{Z}$ be integer program values. Program stores $s \in \mathcal{S} \triangleq \mathsf{Var} \to \mathsf{Val}$ are maps from variables to values and we write $s[x \mapsto v]$ to denote the store obtained by extending $s \in \mathcal{S}$ such that x has value v. Actions $a \in \mathsf{Act}$ are variable assignments $x \coloneqq E$, where $x \in \mathsf{Var}$ and E can be a variable $x \in \mathsf{Var}$, constant $v \in \mathsf{Val}$, test b, or an arithmetic operation $(+, -, \times)$.

$$a \in \mathsf{Act} ::= x \coloneqq E$$
$$E ::= x \in \mathsf{Var} \mid v \in \mathsf{Val} \mid b \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times E_2$$

[4] *N.B.*, this only includes the deterministic program constructs—if statements and while loops instead of $C_1 + C_2$ and $C^{\langle e,e' \rangle}$. The inclusion of a few more derived rules completes the proof system for nondeterministic programs

[5] Outcome Logic guarantees the *existence* of terminating traces, but it is not a total correctness logic in that it cannot ensure that *all* traces terminate. This stems from the program semantics, which collects the finite traces, but does not preclude additional nonterminating ones. For example, $[\![\mathsf{skip}]\!](\sigma) = [\![\mathsf{skip} + \mathsf{while\ true\ do\ skip}]\!](\sigma)$. The exception is the probabilistic interpretation, where *almost sure termination* can be established by proving that the weight of the postcondition is 1.

In addition, we let the set of primitive tests $\mathsf{Test} = 2^{\mathcal{S}}$ be all subsets of the program states $\mathcal{S}$. We will often write these tests symbolically, for example $x \geq 5$ represents the set $\{s \in \mathcal{S} \mid s(x) \geq 5\}$. The interpretation of atomic actions is shown below, where the interpretation of expressions $[\![E]\!]_{\mathsf{Exp}} : \mathcal{S} \to \mathsf{Val}$ is in Appendix E.

$$[\![x \coloneqq E]\!]_{\mathsf{Act}}(s) \triangleq \eta(s[x \mapsto [\![E]\!]_{\mathsf{Exp}}(s)])$$

We define substitutions in the standard way [2, 3, 17, 33], as follows:

$$\varphi[E/x] \triangleq \{m \in \mathcal{W}(\mathcal{S}) \mid (\lambda s.\eta(s[x \mapsto [\![E]\!]_{\mathsf{Exp}}(s))])^{\dagger}(m) \in \varphi\}$$

That is, $m \in \varphi[E/x]$ exactly when assigning x to E in m satisfies φ. This behaves as expected in conjunction with symbolic tests, for example $(x \geq 5)[y + 1/x] = (y + 1 \geq 5) = (y \geq 4)$. It also distributes over most of the operations in Figure 2, *e.g.,* $(\varphi \oplus \psi)[E/x] = \varphi[E/x] \oplus \psi[E/x]$. Using substitution, we add an inference rule for assignment, mirroring the typical weakest-precondition style rule of Hoare Logic [29].

$$\frac{}{\langle \varphi[E/x] \rangle\ x \coloneqq E\ \langle \varphi \rangle}\ \textsc{Assign}$$

When used in combination with the rule of Consequence, Assign can be used to derive any semantically valid triple about variable assignment. Though it is not needed for completeness, we also include the rule of Constancy, which allows us to add information about unmodified variables to a completed derivation. Here, $\mathsf{free}(P)$ is the set of *free variables* that are used by the assertion P (*e.g.,* $\mathsf{free}(x \geq 5) = \{x\}$) and $\mathsf{mod}(C)$ are the variables modified by C, both defined in Appendix E. The $\Box$ modality guarantees that the rule applies regardless of whether or not C terminates.

$$\frac{\langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \mathsf{free}(P) \cap \mathsf{mod}(C) = \emptyset}{\langle \varphi \land \Box P \rangle\ C\ \langle \psi \land \Box P \rangle}\ \textsc{Constancy}$$

In this particular Outcome Logic instance, all triples can be derived without the axioms Ω from Theorem 4.6.

Theorem 6.1 (Soundness and Completeness).

$$\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \iff \qquad \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$$

7 CASE STUDY: REUSING PROOF FRAGMENTS

The following case study serves as a proof of concept for how Outcome Logic's unified reasoning principles can benefit large-scale program analysis. The efficiency of such systems relies on pre-computing procedure specifications, which can simply be inserted whenever those procedures are invoked rather than being recomputed at every call-site. Present analysis systems operate over homogenous effects. Moreover—when dealing with nondeterministic programs—they must also fix either a demonic interpretation (for correctness) or an angelic interpretation (for bug-finding).

But many procedures do not have effects—they do not branch into multiple outcomes and use only limited forms of looping where termination is easily established (*e.g.,* iterating over a data structure)—suggesting that specifications for such procedures can be reused across multiple types of programs (*e.g.,* nondeterministic or probabilistic) and specifications (*e.g.,* partial or total correctness). Indeed, this is the case for the program in Section 7.1. We then show how a single proof about that program can be reused in both a partial correctness specification (Section 7.2) and a probabilistic program (Section 7.3). The full derivations are given in Appendix F.

7.1 Integer Division

In order to avoid undefined behavior related to division by zero, our expression syntax from Section 6 does not include division. However, we can write a simple procedure to divide two natural numbers a and b using repeated subtraction.

$$\mathsf{DIV} \triangleq \left\{ \begin{array}{l} q \coloneqq 0\,⨟\, r \coloneqq a\,⨟ \\ \mathsf{while}\ r \geq b\ \mathsf{do} \\ \quad r \coloneqq r - b\,⨟ \\ \quad q \coloneqq q + 1 \end{array} \right.$$

At the end of the execution, q holds the quotient and r is the remainder. Although the DIV program uses a while loop, it is quite easy to establish that it terminates. To do so, we use the Variant rule with the family of variants φ_n shown below.

$$\varphi_n \triangleq \begin{cases} q + n = \lfloor a \div b \rfloor \land r = (a \bmod b) + n \times b & \text{if } n \leq \lfloor a \div b \rfloor \\ \mathsf{false} & \text{if } n > \lfloor a \div b \rfloor \end{cases}$$

Executing the loop body in a state satisfying φ_n results in a state satisfying φ_{n-1}. At the end, φ_0 stipulates that $q = \lfloor a \div b \rfloor$ and $r = a \bmod b$, which immediately implies that $r < b$, so the loop must exit. This allows us to give the following specification for the program.

$$\langle a \geq 0 \land b > 0 \rangle\ \mathsf{DIV}\ \langle q = \lfloor a \div b \rfloor \land r = a \bmod b \rangle$$

Note that the DIV program is fully deterministic; we did not make any assumptions about which interpretation of choice is used. This will allow us to reuse the proof of DIV in programs with different kinds of effects in the remainder of the section.

7.2 The Collatz Conjecture

Consider the function f defined below.

$$f(n) \triangleq \left\{ \begin{array}{lll} n \div 2 & \text{if} & n \bmod 2 = 0 \\ 3n + 1 & \text{if} & n \bmod 2 = 1 \end{array} \right.$$

The Collatz Conjecture—an elusive open problem in the field of mathematics—postulates that for any positive n, repeated applications of f will eventually yield the value 1. Let the *stopping time* S_n be the minimum number of applications of f to n that it takes to reach 1. For example, $S_1 = 0$, $S_2 = 1$, and $S_3 = 7$. When run in an initial state where $a = n$, the following program computes S_n, storing the result in i. Note that this program makes use of DIV, defined previously.

$$\mathsf{COLLATZ} \triangleq \left\{ \begin{array}{l} i \coloneqq 0\,⨟ \\ \mathsf{while}\ a \neq 1\ \mathsf{do} \\ \quad b \coloneqq 2\,⨟\, \mathsf{DIV}\,⨟ \\ \quad \mathsf{if}\ r = 0\ \mathsf{then}\ a \coloneqq q\ \mathsf{else}\ a \coloneqq 3 \times a + 1\,⨟ \\ \quad i \coloneqq i + 1 \end{array} \right.$$

Since some numbers may not have a finite stopping time—in which case the program will not terminate—this is a perfect candidate for a partial correctness proof. Assuming that a initially holds the value n, we can use a loop invariant stating that $a = f^i(n)$ on each iteration. *If* the program terminates, then $a = f^i(n) = 1$, and so

$S_n = i$. We capture this using the following triple, where the $\Box$ modality indicates that the program may diverge.

$$\langle a = n \land n > 0\rangle \ \mathsf{COLLATZ}\ \langle \Box(i = S_n)\rangle$$

7.3 Embedding Division in a Probabilistic Program

The following program loops for a random number of iterations, deciding whether to continue by flipping a fair coin. It is interpreted using the Prob semiring from Example 2.7.

$$a \coloneqq 0 \fatsemi r \coloneqq 0 \fatsemi (a \coloneqq a + 1 \fatsemi b \coloneqq 2 \fatsemi \mathsf{DIV})^{\langle \frac{1}{2}\rangle}$$

Suppose we want to know the probability that it terminates after an even or odd number of iterations. The program makes use of DIV to divide the current iteration number a by 2, therefore the remainder r will indicate whether the program looped an even or odd number of times. We can analyze the program with the Iter rule, using the following two families of assertions.

$$\varphi_n \triangleq (a = n \land r = a \bmod 2)^{(\frac{1}{2^n})}$$

$$\psi_n \triangleq (a = n \land r = a \bmod 2)^{(\frac{1}{2^{n+1}})}$$

According to Iter, the final postcondition can be obtained by taking an outcome conjunction of all the ψ_n for $n \in \mathbb{N}$. However, we do not care about the precise value of a, only whether r is 0 or 1. The probability that $r = 0$ is $\frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots$, a geometric series whose sum converges to $\frac{2}{3}$. A similar calculation for the $r = 1$ case gives us the following specification, indicating that the program terminates after an even or odd number of iterations with probability $\frac{2}{3}$ and $\frac{1}{3}$, respectively.

$$\begin{array}{l}\langle \mathsf{true}\rangle \\ \quad a \coloneqq 0 \fatsemi r \coloneqq 0 \fatsemi (a \coloneqq a + 1 \fatsemi b \coloneqq 2 \fatsemi \mathsf{DIV})^{\langle \frac{1}{2}\rangle} \\ \langle (r = 0) \oplus_{\frac{2}{3}} (r = 1)\rangle\end{array}$$

8 CASE STUDY: GRAPH PROBLEMS AND QUANTITATIVE ANALYSIS

We now examine case studies using Outcome Logic to derive quantitative properties in alternative models of computation.

8.1 Counting Random Walks

Suppose we wish to count the number of paths between the origin and the point (N, M) on a two dimensional grid. To achieve this, we first write a program that performs a random walk on the grid; while the destination is not yet reached, it nondeterministically chooses to take a step on either the x or y-axis (or steps in a fixed direction if the destination on one axis is already reached).

$$\mathsf{WALK} \triangleq \left\{\begin{array}{l}\mathsf{while}\ x < N \lor y < M\ \mathsf{do} \\ \quad \mathsf{if}\ x < N \land y < M\ \mathsf{then} \\ \quad\quad (x \coloneqq x + 1) + (y \coloneqq y + 1) \\ \quad \mathsf{else\ if}\ x \geq N\ \mathsf{then} \\ \quad\quad y \coloneqq y + 1 \\ \quad \mathsf{else} \\ \quad\quad x \coloneqq x + 1\end{array}\right.$$

Using a standard program logic, it is relatively easy to prove that the program will always terminate in a state where $x = N$ and $y = M$. However, we are going to interpret this program using the Nat semiring (Example 2.6) in order to count how many traces (*i.e.,* random walks) reach that outcome.

First of all, we know it will take exactly $N + M$ steps to reach the destination, so we can analyze the program using the Variant rule, where the loop variant φ_n records the state of the program n steps away from reaching (N, M).

If we are n steps away, then there are several outcomes ranging from $x = N - n \land y = M$ to $x = N \land y = M - n$. More precisely, let k be the distance to N on the x-axis, meaning that the distance to M on the y-axis must be $n - k$, so $x = N - k$ and $y = M - (n - k)$. At all times, it must be true that $0 \leq x \leq N$ and $0 \leq y \leq M$, so it must also be true that $0 \leq N - k \leq N$ and $0 \leq M - (n - k) \leq M$. solving for k, we get that $0 \leq k \leq N$ and $n - M \leq k \leq n$. So, k can range between $\max(0, n - M)$ and $\min(N, n)$.

In addition, the number of paths to (x, y) is $\binom{x+y}{x}$, *i.e.,* the number of ways to pick x steps on the x-axis out of $x + y$ total steps. Putting all of that together, we define our loop variant as follows:

$$\varphi_n \triangleq \bigoplus_{k=\max(0,n-M)}^{\min(N,n)} (x = N - k \land y = M - (n - k))^{(\binom{N+M-n}{N-k})}$$

The loop body moves the program state from φ_{n+1} to φ_n. The outcomes of φ_{n+1} get divided among the three if branches. In the outcome where $x = N$ already, y must step, so this goes to the second branch. Similarly, if $y = M$ already, then x must step, corresponding to the third branch. All other outcomes go to the first branch, which further splits into two outcomes due to the nondeterministic choice.

Since we start $N + M$ steps from the destination, we get the following precondition:

$$\varphi_{N+M} = \bigoplus_{k=N}^{N} (x = N - k \land y = N - k)^{(\binom{0}{N-N})} = (x = 0 \land y = 0)$$

In addition, the postcondition is:

$$\varphi_0 = \bigoplus_{k=0}^{0} (x = N - k \land y = M + k)^{(\binom{N+M}{N})} = (x = N \land y = M)^{(\binom{N+M}{N})}$$

This gives us the final specification below, which tells us that there are $\binom{N+M}{N}$ paths to reach (N, M) from the origin. The full derivation is given in Appendix G.1.

$$\langle x = 0 \land y = 0\rangle\ \mathsf{WALK}\ \langle (x = N \land y = M)^{(\binom{N+M}{N})}\rangle$$

8.2 Shortest Paths

We will now use an alternative interpretation of computation to analyze a program that nondeterministically finds the shortest path from s to t in a directed graph. Let G be the $N \times N$ Boolean adjacency matrix of a directed graph, so that $G[i][j] = \mathsf{true}$ if there is an edge from i to j (or false if no such edge exists). We also add the following expression syntax to read edge weights in a program, noting that $G[E_1][E_2] \in \mathsf{Test}$ since it is Boolean-valued.

$$E ::= \cdots \mid G[E_1][E_2]$$

$$[\![G[E_1][E_2]]\!]_{\mathsf{Exp}}(s) \triangleq G\left[[\![E_1]\!]_{\mathsf{Exp}}(s)\right]\left[[\![E_2]\!]_{\mathsf{Exp}}(s)\right]$$

The following program loops until the current position pos reaches the destination t. At each step, it nondeterministically chooses which edge ($next$) to traverse using an iterator; for all $next \le N$, each trace is selected if there is an edge from pos to $next$, and a weight of 1 is then added to the path, signifying that we took a step.

$$\mathsf{SP} \triangleq \begin{cases} \text{while } pos \neq t \text{ do} \\ \quad next := 1\,\fatsemi \\ \quad (next := next + 1)^{\langle next<N,\ G[pos][next]\rangle}\,\fatsemi \\ \quad pos := next\,\fatsemi \\ \quad \text{assume } 1 \end{cases}$$

We will interpret this program using the Tropical semiring from Example 2.8, in which addition corresponds to min and multiplication corresponds to addition. So, path lengths get accumulated via addition and nondeterministic choices correspond to taking the path with minimal weight. That means that at the end of the program execution, we should end up in a scenario where $pos = t$, with weight equal to the shortest path length from s to t.

To prove this, we first formalize the notion of shortest paths below: $\mathsf{sp}_n^t(G, s, s')$ indicates whether there is a path of length n from s to s' in G in without passing through t and $\mathsf{sp}(G, s, t)$ is the shortest path length from s to t. Let $I = \{1, \dots, N\} \setminus \{t\}$.

$$\mathsf{sp}_0^t(G, s, s') \triangleq (s = s')$$
$$\mathsf{sp}_{n+1}^t(G, s, s') \triangleq \bigvee_{i \in I} \mathsf{sp}_n^t(G, s, i) \wedge G[i][s']$$
$$\mathsf{sp}(G, s, t) \triangleq \min\{n \in \mathbb{N} \mid \mathsf{sp}_n^t(G, s, t)\}$$

We analyze the while loop using the While rule, which requires φ_n and ψ_n to record the outcomes where the loop guard is true or false, respectively, after n iterations. We define these as follows:

$$\varphi_n = \bigoplus_{i \in I} (pos = i)^{(\mathsf{sp}_n^t(G,s,i)+n)}$$

$$\psi_n = (pos = t)^{(\mathsf{sp}_n^t(G,s,t)+n)} \qquad \psi_\infty = (pos = t)^{(\mathsf{sp}(G,s,t))}$$

Recall that in the tropical semiring false $= \infty$ and true $= 0$. So, after n iterations, the weight of the outcome $pos = i$ is equal to n if there is an n-step path from s to i, and ∞ otherwise. The final postcondition ψ_∞ is the shortest path length to t, which is also the minimum of $\mathsf{sp}_n^t(G, s, t) + n$ for all n. Using the Iter rule we get the following derivation for the inner loop:

$$\begin{array}{l}
\left\langle \bigoplus_{i \in I} (pos = i \wedge next = 1)^{(\mathsf{sp}_n^t(G,s,i)+n)} \right\rangle \\
\quad (next := next + 1)^{\langle next<N,\ G[pos][next]\rangle}\,\fatsemi \\
\left\langle \bigoplus_{j=1}^N \bigoplus_{i \in I} (pos = i \wedge next = j)^{((\mathsf{sp}_n^t(G,s,i) \wedge G[i][j])+n)} \right\rangle \\
\quad pos := next\,\fatsemi\, \text{assume } 1 \\
\left\langle \bigoplus_{j=1}^N \bigoplus_{i \in I} (pos = j)^{((\mathsf{sp}_n^t(G,s,i) \wedge G[i][j])+n+1)} \right\rangle \implies \\
\left\langle \bigoplus_{j=1}^N (pos = j)^{(\mathsf{sp}_{n+1}^t(G,s,j)+n+1)} \right\rangle
\end{array}$$

The outcome conjunction over $i \neq t$ (corresponding to the minimum weight path) gives us $pos = j$ with weight $\mathsf{sp}_{n+1}^t(G, s, j) + n + 1$—it is $n + 1$ if there is path of length n to some i and $G[i][j]$.

The precondition is $\varphi_0 \oplus \psi_0 = (pos = s)$, since $\mathsf{sp}_0^t(G, s, i) = \text{false}$ when $i \neq s$ and true when $i = s$. Putting this all together, we get the following triple, stating that the final position is t and the weight is equal to the shortest path.

$$\langle pos = s \rangle\ \mathsf{SP}\ \langle (pos = t)^{(\mathsf{sp}(G,s,t))} \rangle$$

Note that the program does not terminate if there is no path from s to t. In that case, since there are no reachable outcomes, the interpretation of the program should be $\mathbb{0}$. Indeed, $\mathbb{0} = \infty$ in the tropical semiring, which is also the shortest path between two disconnected nodes. The postcondition is therefore $(pos = t)^{(\infty)}$, meaning that the program diverged.

9 DISCUSSION AND RELATED WORK

Computational effects have traditionally beckoned disjoint program logics across two dimensions: different *kinds* of effects (*e.g.,* nondeterminism vs randomization) and different *assertions* about those effects (*e.g.,* angelic vs demonic nondeterminism). Outcome Logic [57] captures all of those properties in a unified way, but until now the proof theory has not been thoroughly explored.

This paper provides a relatively complete proof system for Outcome Logic, showing that programs with effects are not only *semantically* similar, but also share common reasoning principles. In addition, specialized techniques (*i.e.,* analyzing loops with variants or invariants) are particular modes of use of our more general framework, and are compatible with each other rather than requiring semantically distinct program logics. This new perspective invites increased sharing across formal methods for diverse types of programs, and properties about those programs.

Correctness, Incorrectness, and Unified Program Logics. While formal verification has long been the aspiration for automated static analysis, bug-finding tools are often more practical in real engineering settings. This partly comes down to efficiency—bugs can be found without considering all the program traces—and partly due to the fact that most real world software just is not correct [24].

However, standard logical foundations of program analysis such as Hoare Logic are prone to *false positives* when used for bug-finding—they cannot witness the existence of erroneous traces. In response, O'Hearn developed Incorrectness Logic, which under-approximates the reachable states (as opposed to Hoare Logic's over-approximation) so as to only report bugs that truly occur [47].

Although Incorrectness Logic successfully serves as a logical foundation for bug-finding tools [39, 50], it is semantically incompatible with correctness analysis, making sharing of toolchains difficult. Attention has therefore turned to ways to unify the theories of correctness and incorrectness. This includes Exact Separation Logic, which combines Hoare Logic and Incorrectness Logic to generate specifications that are valid for both, but that also preludes under- or over-approximation via the rule of consequence [41]. Local Completeness Logic combines Incorrectness Logic with an over-approximate abstract domain, to similar effect [9, 10].

Outcome Logic. Outcome Logic unifies correctness and incorrectness reasoning without compromising the use of logical consequences. This builds on an idea colloquially known as *Lisbon Logic*, first proposed by Derek Dreyer and Ralf Jung in 2019, that has similarities to the diamond modality of Dynamic Logic [49] and Hoare's calculus of *possible correctness* [30]. The idea was briefly mentioned in the Incorrectness Logic literature [39, 45, 47], but using Lisbon Logic as a foundation of incorrectness analysis was not fully explored until the introduction of Outcome Logic [57], which generalizes both Lisbon Logic and Hoare Logic. The metatheory of Lisbon Logic has subsequently been explored more deeply in a variety of unpublished manuscripts [2, 52]. Hyper Hoare Logic also generalizes Hoare and Lisbon Logics [17], and is semantically equivalent to the Boolean instance of OL (Example 2.4), but does not support effects other than nondeterminism.

The initial Outcome Logic paper used a model based on both a monad and a monoid, with looping defined via the Kleene star $C^\star$ [57]. The semantics of $C^\star$ had to be justified for each instance. However, $C^\star$ is not compatible with probabilistic computation (see Footnote 2), so an ad-hoc semantics was used in the probabilistic case. Moreover, only the Induction rule was provided for reasoning about $C^\star$, which amounts to unrolling the loop one time. Some loops can be analyzed by applying Induction repeatedly, but it is inadequate if the number of iterations depends at all on the program state. Our $C^{\langle e,e' \rangle}$ construct fixes this, defining iteration in a way that supports both Kleene star ($C^{\langle \mathbb{1},\mathbb{1} \rangle}$) and also probabilistic computation. Our Iter rule can be used to reason about any loop, even ones that iterate an unbounded number of times.

The next Outcome Logic paper focused on a particular instance based on separation logic [58]. The model was refined to use semirings, and the language included while loops instead of $C^\star$ so that a single well-definedness proof could extend to all instances. However, the evaluation model included additional constraints ($\mathbb{1} = \top$ and normalization) that preclude, *e.g.*, the multiset model (Example 2.6) that we use in this paper. Rather than giving inference rules, the paper provided a symbolic execution algorithm, which also only supported loops via bounded unrolling.

This paper goes beyond prior work on Outcome Logic by giving a more general model with more instances and better support for iteration, providing a relatively complete proof system that is able to handle any loops, and exploring case studies related to previously unsupported types of computation and looping.

Computational Effects. Effects have been present since the early years of program analysis. Even basic programming languages with while loops introduce the possibility of *nontermination.* Partial correctness was initially used to sidestep the termination question [26, 29], but total correctness (requiring termination) was later introduced too [43]. More recently, automated tools were developed to prove (non)termination in real-world software [6, 7, 12–14, 52].

Nondeterminism also showed up in early variants of Hoare Logic, stemming from Dijkstra's Guarded Command Language (GCL) [21] and Dynamic Logic [49]; it is useful for modeling backtracking algorithms [27] and opaque aspects of program evaluation such as user input and concurrent scheduling. While Hoare Logic has traditionally used demonic nondeterminism [8], other program logics have recently arisen to deal with nondeterminism in different ways, particularly for incorrectness [2, 18, 45, 47, 52, 57].

Beginning with the seminal work of Kozen [35, 36], the study of probabilistic programs has a rich history. This eventually led to the development of probabilistic Hoare Logic variants [3, 19, 20, 53] that enable reasoning about programs in terms of likelihoods and expected values. Doing so requires pre- and postconditions to be predicates on probability distributions rather than individual states.

Outcome Logic generalizes reasoning about all of those effects using using a common set of inference rules. This opens up the possibility for static analysis tools that soundly share proof fragments between different types of programs, as shown in Section 7.

Relative Completeness and Expressivity. Any sufficiently expressive program logic must necessarily be incomplete since, for example, the Hoare triple $\{\mathsf{true}\}\ C\ \{\mathsf{false}\}$ states that the program C never halts, which is not provable in an axiomatic deduction system. In response, Cook devised the idea of *relative* completeness to convey that a proof system is adequate for analyzing a program, but not necessarily assertions about the program states [15].

Expressivity requires that the assertion language used in pre- and postconditions can describe the intermediate program states needed to, *e.g.*, apply the Seq rule. In other words, the assertion syntax must be able to express $\mathrm{post}(C, P)$ from Definition 4.4. Implications for an *expressive* language quickly become undecidable, as they must encode Peano arithmetic [1, 40]. With this in mind, the best we can hope for is a program logic that is complete *relative* to an oracle that decides implications in the rule of Consequence.

The question of what an expressive (syntactic) assertion language for Outcome Logic looks like remains open. In fact, the question of expressive assertion languages for probabilistic Hoare Logics (which are subsumed by Outcome Logic) is also open [3, 20]. A complete probabilistic logic with syntactic assertions does exist, but the programming language does not include loops and is therefore considerably simplified [19]; it is unclear if this approach would extend to looping programs. To avoid the question of expressivity, modern program logics (including our own) typically use semantic assertions [2, 3, 11, 16, 17, 31–33, 47, 52, 55]. This includes logics that are mechanized within proof assistants [3, 17, 31, 32].

Our completeness proof (Theorem 4.6) has parallels to Propositional Hoare Logic, as we assume that axioms are available to prove properties about atomic commands [38]. However, in Theorem 6.1, we also show that a particular OL instance with variable assignment is relatively complete without additional axioms.

Quantitative Reasoning and Weighted Programming. Whereas Hoare Logic provides a foundation for *propositional* program analysis, quantitative program analysis has been explored too. Probabilistic Propositional Dynamic Logic [36] and weakest pre-expectation calculi [5, 33, 46] are used to reason about randomized programs in terms of expected values. This idea has been extended to non-probabilistic quantitative properties too [4, 56].

Weighted programming [4] generalizes pre-expectation reasoning using semirings to model branch weights, much like the model of Outcome Logic presented in this paper. Outcome Logic is a propositional analogue to weighted programming's quantitative model, but it is also more expressive in its ability to reason about quantities over *multiple outcomes*. For example, in Section 7.3, we derive a single OL triple that gives the probabilities of two outcomes, whereas weighted programming (or weakest pre-expectations) would need to compute each probability individually.

In the examples in Section 8 with only one outcome, OL is still more informative—those triples not only indicate the weight of the outcome, but also that it is the *only* possible outcome. That is, we know that WALK cannot terminate in a position other than (N, M) and that SP cannot terminate in a position other than t. With weighted programming, there is not a straightforward way to determine the full set of outcomes.

Finally, weighted programming cannot encode the $\Box$ and $\Diamond$ modalities from Section 3.3 and therefore cannot embed the Hoare and Lisbon Logic reasoning principles that we showed in Section 5.

REFERENCES

[1] Krzysztof R. Apt. 1981. Ten Years of Hoare's Logic: A Survey–Part I. *ACM Trans. Program. Lang. Syst.* 3, 4 (oct 1981), 431–483. https://doi.org/10.1145/357146.357150

[2] Flavio Ascari, Roberto Bruni, Roberta Gori, and Francesco Logozzo. 2023. Sufficient Incorrectness Logic: SIL and Separation SIL. arXiv:2310.18156 [cs.LO]

[3] Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. An Assertion-Based Program Logic for Probabilistic Programs. In *Programming Languages and Systems*. Springer International Publishing, Cham, 117–144. https://doi.org/10.1007/978-3-319-89884-1_5

[4] Kevin Batz, Adrian Gallus, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Tobias Winkler. 2022. Weighted Programming: A Programming Paradigm for Specifying Mathematical Models. *Proc. ACM Program. Lang.* 6, OOPSLA1, Article 66 (apr 2022), 30 pages. https://doi.org/10.1145/3527310

[5] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Thomas Noll. 2019. Quantitative Separation Logic: A Logic for Reasoning about Probabilistic Pointer Programs. *Proc. ACM Program. Lang.* 3, POPL, Article 34 (Jan 2019), 29 pages. https://doi.org/10.1145/3290347

[6] Josh Berdine, Byron Cook, Dino Distefano, and Peter W. O'Hearn. 2006. Automatic Termination Proofs for Programs with Shape-Shifting Heaps. In *Computer Aided Verification*. Springer Berlin Heidelberg, Berlin, Heidelberg, 386–400. https://doi.org/10.1007/11817963_35

[7] Marc Brockschmidt, Byron Cook, and Carsten Fuhs. 2013. Better Termination Proving through Cooperation. In *Computer Aided Verification*. Springer Berlin Heidelberg, Berlin, Heidelberg, 413–429. https://doi.org/10.1007/978-3-642-39799-8_28

[8] Manfred Broy and Martin Wirsing. 1981. On the Algebraic Specification of Nondeterministic Programming Languages. In *Proceedings of the 6th Colloquium on Trees in Algebra and Programming (CAAP '81)*. Springer-Verlag, Berlin, Heidelberg, 162–179. https://doi.org/10.5555/648216.750907

[9] Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2021. A Logic for Locally Complete Abstract Interpretations. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–13. https://doi.org/10.1109/LICS52264.2021.9470608

[10] Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2023. A Correctness and Incorrectness Program Logic. *J. ACM* 70, 2, Article 15 (mar 2023), 45 pages. https://doi.org/10.1145/3582267

[11] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. 2007. Local Action and Abstract Separation Logic. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*. 366–378. https://doi.org/10.1109/LICS.2007.30

[12] Byron Cook, Carsten Fuhs, Kaustubh Nimkar, and Peter O'Hearn. 2014. Disproving Termination with Overapproximation. In *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design* (Lausanne, Switzerland) *(FMCAD '14)*. FMCAD Inc, Austin, Texas, 67–74. https://doi.org/10.1109/FMCAD.2014.6987597

[13] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. 2006. Termination Proofs for Systems Code. *SIGPLAN Not.* 41, 6 (jun 2006), 415–426. https://doi.org/10.1145/1133255.1134029

[14] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. 2006. Termination Proofs for Systems Code. In *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Ottawa, Ontario, Canada) *(PLDI '06)*. Association for Computing Machinery, New York, NY, USA, 415–426. https://doi.org/10.1145/1133981.1134029

[15] Stephen A. Cook. 1978. Soundness and Completeness of an Axiom System for Program Verification. *SIAM J. Comput.* 7, 1 (feb 1978), 70–90. https://doi.org/10.1137/0207005

[16] Patrick M. Cousot, Radhia Cousot, Francesco Logozzo, and Michael Barnett. 2012. An Abstract Interpretation Framework for Refactoring with Application to Extract Methods with Contracts. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications* (Tucson, Arizona, USA) *(OOPSLA '12)*. Association for Computing Machinery, New York, NY, USA, 213–232. https://doi.org/10.1145/2384616.2384633

[17] Thibault Dardinier and Peter Müller. 2023. Hyper Hoare Logic: (Dis-)Proving Program Hyperproperties (extended version). https://doi.org/10.48550/ARXIV.2301.10037

[18] Edsko de Vries and Vasileios Koutavas. 2011. Reverse Hoare Logic. In *Software Engineering and Formal Methods*. Springer Berlin Heidelberg, Berlin, Heidelberg, 155–171. https://doi.org/10.1007/978-3-642-24690-6_12

[19] Jerry den Hartog. 2002. *Probabilistic Extensions of Semantical Models*. Ph.D. Dissertation. Vrije Universiteit Amsterdam. https://core.ac.uk/reader/15452110

[20] J. I. den Hartog. 1999. Verifying Probabilistic Programs Using a Hoare like Logic. In *Advances in Computing Science — ASIAN'99*, P. S. Thiagarajan and Roland Yap (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 113–125.

[21] Edsger W. Dijkstra. 1975. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Commun. ACM* 18, 8 (Aug 1975), 453–457. https://doi.org/10.1145/360933.360975

[22] Edsger W. Dijkstra. 1976. *A Discipline of Programming*. Prentice-Hall. I–XVII, 1–217 pages.

[23] Edsger W. Dijkstra and Carel S. Schölten. 1990. *The strongest postcondition*. Springer New York, New York, NY, 209–215. https://doi.org/10.1007/978-1-4612-3228-5_12

[24] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling Static Analyses at Facebook. *Commun. ACM* 62, 8 (jul 2019), 62–70. https://doi.org/10.1145/3338112

[25] Simon Docherty. 2019. *Bunched logics: a uniform approach*. Ph.D. Dissertation. University College London. https://discovery.ucl.ac.uk/id/eprint/10073115/

[26] Robert W. Floyd. 1967. Assigning Meanings to Programs. In *Mathematical Aspects of Computer Science (Proceedings of Symposia in Applied Mathematics, Vol. 19)*. American Mathematical Society, Providence, Rhode Island, 19–32.

[27] Robert W. Floyd. 1967. Nondeterministic Algorithms. *J. ACM* 14, 4 (oct 1967), 636–644. https://doi.org/10.1145/321420.321422

[28] Jonathan S. Golan. 2003. *Semirings and Affine Equations over Them*. Springer Dordrecht. https://doi.org/10.1007/978-94-017-0383-3

[29] C. A. R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580. https://doi.org/10.1145/363235.363259

[30] C. A. R. Hoare. 1978. Some Properties of Predicate Transformers. *J. ACM* 25, 3 (Jul 1978), 461–480. https://doi.org/10.1145/322077.322088

[31] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming* 28 (2018). https://doi.org/10.1017/S0956796818000151

[32] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Mumbai, India) *(POPL '15)*. Association for Computing Machinery, New York, NY, USA, 637–650. https://doi.org/10.1145/2676726.2676980

[33] Benjamin Lucien Kaminski. 2019. *Advanced weakest precondition calculi for probabilistic programs*. Dissertation. RWTH Aachen University, Aachen. https://doi.org/10.18154/RWTH-2019-01829

[34] Georg Karner. 2004. Continuous monoids and semirings. *Theoretical Computer Science* 318, 3 (2004), 355–372. https://doi.org/10.1016/j.tcs.2004.01.020

[35] Dexter Kozen. 1979. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science (SFCS '79)*. 101–114. https://doi.org/10.1109/SFCS.1979.38

[36] Dexter Kozen. 1983. A Probabilistic PDL. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC '83)*. Association for Computing Machinery, New York, NY, USA, 291–297. https://doi.org/10.1145/800061.808758

[37] Dexter Kozen. 1997. Kleene Algebra with Tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (May 1997), 427–443. https://doi.org/10.1145/256167.256195

[38] Dexter Kozen and Jerzy Tiuryn. 2001. On the completeness of propositional Hoare logic. *Information Sciences* 139, 3 (2001), 187–195. https://doi.org/10.1016/S0020-0255(01)00164-5 Relational Methods in Computer Science.

[39] Quang Loc Le, Azalea Raad, Jules Villard, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. 2022. Finding Real Bugs in Big Programs with Incorrectness Logic. *Proc. ACM Program. Lang.* 6, OOPSLA1, Article 81 (Apr 2022), 27 pages. https://doi.org/10.1145/3527325

[40] Richard J. Lipton. 1977. A necessary and sufficient condition for the existence of hoare logics. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 1–6. https://doi.org/10.1109/SFCS.1977.1

[41] Petar Maksimović, Caroline Cronjäger, Andreas Lööw, Julian Sutherland, and Philippa Gardner. 2023. Exact Separation Logic: Towards Bridging the Gap Between Verification and Bug-Finding. In *37th European Conference on Object-Oriented Programming (ECOOP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 263)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 19:1–19:27. https://doi.org/10.4230/LIPIcs.ECOOP.2023.19

[42] Ernest G. Manes. 1976. *Algebraic Theories*. Springer New York. https://doi.org/10.1007/978-1-4612-9860-1

[43] Zohar Manna and Amir Pnueli. 1974. Axiomatic Approach to Total Correctness of Programs. *Acta Inf.* 3, 3 (sep 1974), 243–263. https://doi.org/10.1007/BF00288637

[44] Eugenio Moggi. 1991. Notions of computation and monads. *Information and Computation* 93, 1 (1991), 55–92. https://doi.org/10.1016/0890-5401(91)90052-4

[45] Bernhard Möller, Peter O'Hearn, and Tony Hoare. 2021. On Algebra of Program Correctness and & Incorrectness. In *Relational and Algebraic Methods in Computer Science: 19th International Conference, RAMiCS 2021, Marseille, France, November 2–5, 2021, Proceedings* (Marseille, France). Springer-Verlag, Berlin, Heidelberg, 325–343. https://doi.org/10.1007/978-3-030-88701-8_20

[46] Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic Predicate Transformers. *ACM Trans. Program. Lang. Syst.* 18, 3 (may 1996), 325–353. https://doi.org/10.1145/229542.229547

[47] Peter W. O'Hearn. 2019. Incorrectness Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 10 (Dec. 2019), 32 pages. https://doi.org/10.1145/3371078

[48] Peter W. O'Hearn and David J. Pym. 1999. The Logic of Bunched Implications. *The Bulletin of Symbolic Logic* 5, 2 (1999), 215–244.

[49] Vaughan R. Pratt. 1976. Semantical Considerations on Floyd-Hoare Logic. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. 109–121. https://doi.org/10.1109/SFCS.1976.27

[50] Azalea Raad, Josh Berdine, Hoang-Hai Dang, Derek Dreyer, Peter O'Hearn, and Jules Villard. 2020. Local Reasoning About the Presence of Bugs: Incorrectness Separation Logic. In *Computer Aided Verification*. Springer International Publishing, Cham, 225–252. https://doi.org/10.1007/978-3-030-53291-8_14

[51] Azalea Raad, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. 2022. Concurrent Incorrectness Separation Logic. *Proc. ACM Program. Lang.* 6, POPL, Article 34 (Jan 2022), 29 pages. https://doi.org/10.1145/3498695

[52] Azalea Raad, Julien Vanegue, and Peter O'Hearn. 2023. Compositional Non-Termination Proving. https://www.soundandcomplete.org/papers/Unter.pdf

[53] Robert Rand and Steve Zdancewic. 2015. VPHL: A Verified Partial-Correctness Logic for Probabilistic Programs. In *Electronic Notes in Theoretical Computer Science*, Vol. 319. 351–367. https://doi.org/10.1016/j.entcs.2015.12.021 The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI).

[54] Wojciech Różowski, Tobias Kappé, Dexter Kozen, Todd Schmid, and Alexandra Silva. 2023. Probabilistic Guarded KAT Modulo Bisimilarity: Completeness and Complexity. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 261)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 136:1–136:20. https://doi.org/10.4230/LIPIcs.ICALP.2023.136

[55] Hongseok Yang. 2001. *Local Reasoning for Stateful Programs*. Ph.D. Dissertation. USA. Advisor(s) Reddy, Uday S. https://doi.org/10.5555/933728

[56] Linpeng Zhang and Benjamin Lucien Kaminski. 2022. Quantitative Strongest Post: A Calculus for Reasoning about the Flow of Quantitative Information. *Proc. ACM Program. Lang.* 6, OOPSLA1, Article 87 (apr 2022), 29 pages. https://doi.org/10.1145/3527331

[57] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 93 (Apr 2023), 29 pages. https://doi.org/10.1145/3586045

[58] Noam Zilberstein, Angelina Saliling, and Alexandra Silva. 2023. Outcome Separation Logic: Local Reasoning for Correctness and Incorrectness with Computational Effects. (2023). https://doi.org/10.48550/arXiv.2305.04842 arXiv:2305.04842 [cs.LO]

A TOTALITY OF LANGUAGE SEMANTICS

Before proving that the language semantics is total, we must first introduce a few new concepts.

Definition A.1 (Natural Ordering). Given a (partial) semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$, the natural order is defined to be:

$$u \leq v \qquad \text{iff} \qquad \exists w.\, u + w = v$$

The semiring is naturally ordered if the natural order $\leq$ is a partial order. Note that $\leq$ is trivially reflexive and transitive, but it remains to show that it is anti-symmetric.

Natural orders extend to weighting functions too, where $m_1 \sqsubseteq m_2$ iff there exists m such that $m_1 + m = m_2$. This corresponds exactly to the pointwise order as well, so $m_1 \sqsubseteq m_2$ iff $m_1(\sigma) \leq m_2(\sigma)$ for all $\sigma \in \mathsf{supp}(m_1)$.

Definition A.2 (Complete Semiring [28]). A (partial) semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ is complete if there is a sum operator $\sum_{i\in I}$ with the following properties:

(1) If $I = \{i_1, \dots, i_n\}$ is finite, then $\sum_{i\in I} u_i = u_{i_1} + \cdots + u_{i_n}$
(2) If $\sum_{i\in I} x_i$ is defined, then $v \cdot \sum_{i\in I} u_i = \sum_{i\in I} v \cdot u_i$ and $(\sum_{i\in I} u_i) \cdot v = \sum_{i\in I} u_i \cdot v$
(3) Let $(J_k)_{k\in K}$ be a family of nonempty disjoint subsets of I ($I = \bigcup_{k\in K} J_k$ and $J_k \cap J_\ell = \emptyset$ if $k \neq \ell$), then $\sum_{k\in K}\sum_{j\in J_k} u_j = \sum_{i\in I} u_i$

Definition A.3 (Scott Continuity [34]). 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:

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

Lemma A.4. *Let $\langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ be a complete, continuous, naturally ordered, partial semiring. For any family of Scott continuous functions $(f_i : X \to X)_{i\in I}$ and directed set $D \subseteq X$:*

$$\sup_{x\in D}\sum_{i\in I} f_i(x) = \sum_{i\in I} f_i(\sup D)$$

Proof. Since each f_i is Scott continuous, then we know that $\{f_i(x) \mid x \in D\}$ is a directed set. The proof proceeds by transfinite induction on I.

- Base case: $I = \{i_1\}$, then we simply need to show that $\sup_{x\in D} f_{i_1}(x) = f_{i_1}(\sup D)$, which follows from the fact that f_{i_1} is Scott continuous.
- Limit case: suppose that the claim holds for all sets smaller than I. It must be possible to divide I into two disjoint parts I_1 and I_2 such that $I = I_1 \cup I_2$ and $I_1 \cap I_2 = \emptyset$. Now, given the definition of the sum operator:

$$\sup_{x\in D}\sum_{i\in I} f_i(x) = \sup_{x\in D}\left(\sum_{i\in I_1} f_i(x) + \sum_{i\in I_2} f_i(x)\right)$$

By the induction hypothesis, we know that $\lambda x.\, \sum_{i\in J} f_i(x)$ is Scott continuous for any $J \subseteq I$, so given that the semiring is Scott continuous, we can move the supremum inside the outer $+$.

$$= (\sup_{x\in D}\sum_{i\in I_1} f_i(x)) + (\sup_{x\in D}\sum_{i\in I_2} f_i(x))$$

By the induction hypothesis again:

$$= \left(\sum_{i\in I_1} f_i(\sup D)\right) + \left(\sum_{i\in I_2} f_i(\sup D)\right)$$
$$= \sum_{i\in I} f_i(\sup D)$$

□

A.1 Semantics of Tests and Expressions

Given some semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$, the definition of the semantics of tests $[\![b]\!]_{\mathsf{Test}} : \Sigma \to \{\mathbb{0}, \mathbb{1}\}$ is below.

$$[\![\mathsf{true}]\!]_{\mathsf{Test}}(\sigma) \triangleq \mathbb{1}$$
$$[\![\mathsf{false}]\!]_{\mathsf{Test}}(\sigma) \triangleq \mathbb{0}$$
$$[\![b_1 \vee b_2]\!]_{\mathsf{Test}}(\sigma) \triangleq \begin{cases} \mathbb{1} & \text{if } [\![b_1]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \text{ or } [\![b_2]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \\ \mathbb{0} & \text{otherwise} \end{cases}$$
$$[\![b_1 \wedge b_2]\!]_{\mathsf{Test}}(\sigma) \triangleq \begin{cases} \mathbb{1} & \text{if } [\![b_1]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \text{ and } [\![b_2]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \\ \mathbb{0} & \text{otherwise} \end{cases}$$
$$[\![\neg b]\!]_{\mathsf{Test}}(\sigma) \triangleq \begin{cases} \mathbb{1} & \text{if } [\![b]\!]_{\mathsf{Test}}(\sigma) = \mathbb{0} \\ \mathbb{0} & \text{if } [\![b]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \end{cases}$$
$$[\![t]\!]_{\mathsf{Test}}(\sigma) \triangleq \begin{cases} \mathbb{1} & \text{if } \sigma \in t \\ \mathbb{0} & \text{if } \sigma \notin t \end{cases}$$

Based on that, we define the semantics of expressions $[\![e]\!] : \Sigma \to U$.

$$[\![b]\!](\sigma) \triangleq [\![b]\!]_{\mathsf{Test}}(\sigma)$$
$$[\![u]\!](\sigma) \triangleq u$$

A.2 Fixed Point Existence

For all the proofs in this section, we assume that the operations $+$, $\cdot$, and $\sum$ belong to a complete, Scott continuous, naturally ordered, partial semiring with a top element (as described in Section 2.2).

Lemma A.5. *If $\sum_{i\in I} u_i$ is defined, then for any $(v_i)_{i\in I}$, $\sum_{i\in I} u_i \cdot v_i$ is defined.*

Proof. Let v be the top element of U, so $v \geq v_i$ for all $i \in I$. That means that for each $i \in I$, there is a v_i' such that $v_i + v_i' = v$. Now, since multiplication is total, then we know that $(\sum_{i\in I} u_i) \cdot v$ is defined. This gives us:

$$(\sum_{i\in I} u_i) \cdot v = \sum_{i\in I} u_i \cdot (v_i + v_i') = \sum_{i\in I} u_i \cdot v_i + \sum_{i\in I} u_i \cdot v_i'$$

And since $\sum_{i\in I} u_i \cdot v_i$ is a subexpression of the above well-defined term, then it must be well-defined. □

Lemma A.6. *For any $m \in \mathcal{W}(X)$ and $f : X \to \mathcal{W}(Y)$, we know that $f^\dagger : \mathcal{W}(X) \to \mathcal{W}(Y)$ is a total function.*

Proof. First, recall the definition of $(-)^\dagger$:

$$f^\dagger(m)(y) = \sum_{x\in\mathsf{supp}(m)} m(x) \cdot f(x)(y)$$

To show that this is well, defined we need to show both that the sum exists, and that the resulting weighting function has a well-defined mass. First, we remark that since $m \in \mathcal{W}(A)$, then $|m| = \sum_{x\in\mathsf{supp}(m)} m(x)$ must be defined. By Lemma A.5, the sum in the definition of $(-)^\dagger$ is therefore defined. Now, we need to show that $|f^\dagger(m)|$ is defined:

$$|f^\dagger(m)| = \sum_{y\in\mathrm{supp}(f^\dagger(m))} f^\dagger(m)(y)$$
$$= \sum_{y\in\bigcup_{a\in\mathrm{supp}(m)}\mathrm{supp}(f(a))}\ \sum_{x\in\mathrm{supp}(m)} m(x)\cdot f(x)(y)$$

By commutativity and associativity:

$$= \sum_{x\in\mathrm{supp}(m)} m(x) \cdot \sum_{y\in\mathrm{supp}(f(x))} f(x)(y) = \sum_{x\in\mathrm{supp}(m)} m(x)\cdot|f(x)|$$

Now, since $f(x) \in \mathcal{W}(Y)$ for all $x \in X$, we know that $|f(x)|$ must be defined. The outer sum also must be defined by Lemma A.5. ◻

In the following, when comparing functions $f, g\colon X \to \mathcal{W}(Y)$, we will use the pointwise order. That is, $f \sqsubseteq g$ iff $f(x) \sqsubseteq g(x)$ for all $x \in X$.

Lemma A.7. *$(-)^\dagger\colon (X \to \mathcal{W}(Y)) \to (\mathcal{W}(X) \to \mathcal{W}(Y))$ is Scott continuous.*

Proof. Let $D \subseteq (X \to \mathcal{W}(Y))$ be a directed set. First, we show that for any $x \in X$, the function $g(f) = m(x)\cdot f(x)(y)$ is Scott continuous:

$$\sup_{f\in D} g(f) = \sup_{f\in D} m(x)\cdot f(x)(y)$$

By Scott continuity of the $\cdot$ operator:

$$= m(x)\cdot \sup_{f\in D} f(x)(y)$$

Since we are using the pointwise ordering:

$$= m(x)\cdot(\sup D)(x)(y) = g(\sup D)$$

Now, we show that $(-)^\dagger$ is Scott continuous:

$$\sup_{f\in D} f^\dagger = \sup_{f\in D} \lambda m. \sum_{x\in\mathrm{supp}(m)} m(x)\cdot f(x)$$

By Lemma A.4, using the property we just proved.

$$= \lambda m. \sum_{x\in\mathrm{supp}(m)} m(x)\cdot(\sup D)(x) = (\sup D)^\dagger$$

◻

Lemma A.8. *Let $\Phi_{\langle C,e,e'\rangle}(f)(\sigma) = \llbracket e\rrbracket(\sigma)\cdot f^\dagger(\llbracket C\rrbracket(\sigma)) + \llbracket e'\rrbracket(\sigma)\cdot\eta(\sigma)$ and suppose that it is a total function, then $\Phi_{\langle C,e,e'\rangle}$ is Scott continuous with respect to the pointwise order: $f_1 \sqsubseteq f_2$ iff $f_1(\sigma) \sqsubseteq f_2(\sigma)$ for all $\sigma \in \Sigma$.*

Proof. For all directed sets $D \subseteq (\Sigma \to \mathcal{W}(\Sigma))$ and $\sigma \in \Sigma$, we have:

$$\sup_{f\in D} \Phi_{\langle C,e,e'\rangle}(f)(\sigma)$$
$$= \sup_{f\in D}\left(\llbracket e\rrbracket(\sigma)\cdot f^\dagger(\llbracket C\rrbracket(\sigma)) + \llbracket e'\rrbracket(\sigma)\cdot\eta(\sigma)\right)$$

By the continuity of $+$ and $\cdot$, we can move the supremum up to the bind, which is the only term that depends on f.

$$= \llbracket e\rrbracket(\sigma)\cdot(\sup_{f\in D} f^\dagger(\llbracket C\rrbracket(\sigma))) + \llbracket e'\rrbracket(\sigma)\cdot\eta(\sigma)$$

By Lemma A.7.

$$= \llbracket e\rrbracket(\sigma)\cdot(\sup D)^\dagger(\llbracket C\rrbracket(\sigma)) + \llbracket e'\rrbracket(\sigma)\cdot\eta(\sigma)$$
$$= \Phi_{\langle C,e,e'\rangle}(\sup D)(\sigma)$$

Since this is true for all $\sigma \in \Sigma$, $\Phi_{\langle C,e,e'\rangle}$ is Scott continuous. ◻

Now, given Lemma A.8 and the Kleene fixed point theorem, we know that the least fixed point is defined and is equal to:

$$\mu f.\Phi_{\langle C,e,e'\rangle}(f) = \sup_{n\in\mathbb{N}} \Phi^n_{\langle C,e,e'\rangle}(\lambda\tau.\mathbb{0})$$

Therefore the semantics of iteration loops is well-defined, assuming that $\Phi_{\langle C,e,e'\rangle}$ is total. In the next section, we will see simple syntactic conditions to ensure this.

A.3 Syntactic Sugar

Depending on whether a partial or total semiring is used to interpret the language semantics, unrestricted use of the $C_1 + C_2$ and $C^{\langle e,e'\rangle}$ constructs may be undefined. In this section, we give some sufficient conditions to ensure that program semantics is well-defined. This is based on the notion of compatible expressions, introduced below.

Definition A.9 (Compatibility). The expressions e_1 and e_2 are compatible in semiring $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ if $\llbracket e_1\rrbracket(\sigma) + \llbracket e_2\rrbracket(\sigma)$ is defined for any $\sigma \in \Sigma$.

The nondeterministic (Examples 2.4 and 2.6) and tropical (Example 2.8) instances use total semirings, so any program has well-defined semantics. In other interpretations, we must ensure that programs are well-defined by ensuring that all uses of choice and iteration use compatible expressions. We begin by showing that any two collections can be combined if they are scaled by compatible expressions.

Lemma A.10. *If e_1 and e_2 are compatible, then $\llbracket e_1\rrbracket(\sigma)\cdot m_1 + \llbracket e_2\rrbracket(\sigma)\cdot m_2$ is defined for any m_1 and m_2.*

Proof. Since e_1 and e_2 are compatible, then $\llbracket e_1\rrbracket(\sigma) + \llbracket e_2\rrbracket(\sigma)$ is defined. By Lemma A.5, that also means that $\llbracket e_1\rrbracket(\sigma)\cdot|m_1| + \llbracket e_2\rrbracket(\sigma)\cdot|m_2|$ is defined too. Now, we have:

$$\llbracket e_1\rrbracket(\sigma)\cdot|m_1| + \llbracket e_2\rrbracket(\sigma)\cdot|m_2|$$
$$= \llbracket e_1\rrbracket(\sigma)\cdot\sum_{\tau\in\mathrm{supp}(m_1)} m_1(\tau) + \llbracket e_2\rrbracket(\sigma)\cdot\sum_{\tau\in\mathrm{supp}(m_2)} m_2(\tau)$$
$$= \sum_{\tau\in\mathrm{supp}(m_1)} \llbracket e_1\rrbracket(\sigma)\cdot m_1(\tau) + \sum_{\tau\in\mathrm{supp}(m_2)} \llbracket e_2\rrbracket(\sigma)\cdot m_2(\tau)$$
$$= \sum_{\tau\in\mathrm{supp}(m_1)\cup\mathrm{supp}(m_2)} \llbracket e_1\rrbracket(\sigma)\cdot m_1(\tau) + \llbracket e_2\rrbracket(\sigma)\cdot m_2(\tau)$$
$$= |\llbracket e_1\rrbracket(\sigma)\cdot m_1 + \llbracket e_2\rrbracket(\sigma)\cdot m_2|$$

Therefore $\llbracket e_1\rrbracket(\sigma)\cdot m_1 + \llbracket e_2\rrbracket(\sigma)\cdot m_2$ must be well-defined. ◻

Now, we show how this result relates to program semantics. We begin with choice, by showing that guarding the two branches using compatible expressions yields a program that is well-defined.

Lemma A.11. *If e_1 and e_2 are compatible and $\llbracket C_1\rrbracket$ and $\llbracket C_2\rrbracket$ are total functions, then $\llbracket(\mathsf{assume}\ e_1 ⨟ C_1) + (\mathsf{assume}\ e_2 ⨟ C_2)\rrbracket$ is a total function.*

Proof. Take any $\sigma \in \Sigma$, then we have:

$$\begin{aligned}
&\llbracket (\mathsf{assume}\ e_1 \fatsemi C_1) + (\mathsf{assume}\ e_2 \fatsemi C_2) \rrbracket (\sigma) \\
&= \llbracket C_1 \rrbracket^\dagger (\llbracket \mathsf{assume}\ e_1 \rrbracket (\sigma)) + \llbracket C_2 \rrbracket^\dagger (\llbracket \mathsf{assume}\ e_2 \rrbracket (\sigma)) \\
&= \llbracket C_1 \rrbracket^\dagger (\llbracket e_1 \rrbracket (\sigma) \cdot \eta(\sigma)) + \llbracket C_2 \rrbracket^\dagger (\llbracket e_2 \rrbracket (\sigma) \cdot \eta(\sigma)) \\
&= \llbracket e_1 \rrbracket (\sigma) \cdot \llbracket C_1 \rrbracket^\dagger (\eta(\sigma)) + \llbracket e_2 \rrbracket (\sigma) \cdot \llbracket C_2 \rrbracket^\dagger (\eta(\sigma)) \\
&= \llbracket e_1 \rrbracket (\sigma) \cdot \llbracket C_1 \rrbracket (\sigma) + \llbracket e_2 \rrbracket (\sigma) \cdot \llbracket C_2 \rrbracket (\sigma)
\end{aligned}$$

By Lemma A.10, we know that this sum is defined, therefore the semantics is valid. □

For iteration, we can similarly use compatibility to ensure well-definedness.

Lemma A.12. *If e and e' are compatible and $\llbracket C \rrbracket$ is a total function, then $\left\llbracket C^{\langle e,e' \rangle} \right\rrbracket$ is a total function.*

Proof. Let $\Phi_{\langle C,e,e' \rangle}(f)(\sigma) = \llbracket e \rrbracket (\sigma) \cdot f^\dagger(\llbracket C \rrbracket (\sigma)) + \llbracket e' \rrbracket (\sigma) \cdot \eta(\sigma)$. Since e and e' are compatible, it follows from Lemma A.10 that $\Phi_{\langle C,e,e' \rangle}$ is a total function. By Lemma A.8, we therefore also know that $\left\llbracket C^{\langle e,e' \rangle} \right\rrbracket$ is total. □

To conclude, we will provide a few examples of compatible expressions. For any test b, it is easy to see that b and $\neg b$ are compatible. This is because at any state σ, one of $\llbracket b \rrbracket (\sigma)$ or $\llbracket \neg b \rrbracket (\sigma)$ must be $\mathbb{0}$, and given the semiring laws, $\mathbb{0} + u$ is defined for any $u \in U$. Given this, our encodings of if statements and while loops from Section 2.3 are well-defined in all interpretations.

In the probabilistic interpretation (Example 2.7), the weights p and $1 - p$ are compatible for any $p \in [0, 1]$. That means that our encoding of probabilistic choice $C_1 +_p C_2$ and probabilistic iteration $C^{\langle p \rangle}$ are both well-defined too.

B SUBSUMPTION OF PROGRAM LOGICS

In this section, we provide proofs for the theorems in Section 3.3. Note that the following two theorems assume a nondeterministic interpretation of Outcome Logic, using the semiring defined in Example 2.4.

Theorem 3.2 (Subsumption of Hoare Logic).

$$\vDash \langle P \rangle\ C\ \langle \Box Q \rangle \quad \textit{iff} \quad P \subseteq [C]Q \quad \textit{iff} \quad \vDash \{P\}\ C\ \{Q\}$$

Proof. We only prove that $\vDash \langle P \rangle\ C\ \langle \Box Q \rangle$ iff $P \subseteq [C]Q$, since $P \subseteq [C]Q$ iff $\vDash \{P\}\ C\ \{Q\}$ is a well-known result [49].

($\Rightarrow$) Suppose $\sigma \in P$, then $\eta(\sigma) \vDash P$ and since $\vDash \{P\}\ C\ \{\Box Q\}$ we get that $\llbracket C \rrbracket^\dagger (\eta(\sigma)) \vDash \Box Q$, which is equivalent to $\llbracket C \rrbracket (\sigma) \vDash \exists u : U.Q^{(u)}$. This means that $\mathrm{supp}(\llbracket C \rrbracket (\sigma)) \subseteq Q$, therefore by definition $\sigma \in [C]Q$. Therefore, we have shown that $P \subseteq [C]Q$.

($\Leftarrow$) Suppose that $P \subseteq [C]Q$ and $m \vDash P$, so $|m| \neq 0$ and $\mathrm{supp}(m) \subseteq P \subseteq [C]Q$. This means that $\mathrm{supp}(\llbracket C \rrbracket (\sigma)) \subseteq Q$ for all $\sigma \in \mathrm{supp}(m)$, so we also get that:

$$\mathrm{supp}(\llbracket C \rrbracket^\dagger (m)) = \bigcup_{\sigma \in \mathrm{supp}(m)} \llbracket C \rrbracket (\sigma) \subseteq Q$$

This means that $\llbracket C \rrbracket^\dagger (m) \vDash \Box Q$, therefore $\vDash \langle P \rangle\ C\ \langle \Box Q \rangle$. □

Theorem 3.3 (Subsumption of Lisbon Logic).

$$\vDash \langle P \rangle\ C\ \langle \Diamond Q \rangle \quad \textit{iff} \quad P \subseteq \langle C \rangle Q \quad \textit{iff} \quad \vDash \{\!|P|\!\}\ C\ \{\!|Q|\!\}$$

Proof. We only prove that $\vDash \langle P \rangle\ C\ \langle \Diamond Q \rangle$ iff $P \subseteq \langle C \rangle Q$, since $P \subseteq \langle C \rangle Q$ iff $\vDash \{\!|P|\!\}\ C\ \{\!|Q|\!\}$ follows by definition [45, 57].

($\Rightarrow$) Suppose $\sigma \in P$, then $\eta(\sigma) \vDash P$ and since $\vDash \{P\}\ C\ \{\Diamond Q\}$ we get that $\llbracket C \rrbracket^\dagger (\eta(\sigma)) \vDash \Diamond Q$, which is equivalent to $\llbracket C \rrbracket (\sigma) \vDash Q \oplus \top$. This means that there exists a $\tau \in \llbracket C \rrbracket (\sigma)$ such that $\tau \in Q$, therefore by definition $\sigma \in \langle C \rangle Q$. So, we have shown that $P \subseteq \langle C \rangle Q$.

($\Leftarrow$) Suppose that $P \subseteq \langle C \rangle Q$ and $m \vDash P$, so $|m| \neq 0$ and $\mathrm{supp}(m) \subseteq P \subseteq \langle C \rangle Q$. This means that $\mathrm{supp}(\llbracket C \rrbracket (\sigma)) \cap Q \neq \emptyset$ for all $\sigma \in \mathrm{supp}(m)$. In other words, for each $\sigma \in \mathrm{supp}(m)$, there exists a $\tau \in \mathrm{supp}(\llbracket C \rrbracket (\sigma))$ such that $\tau \in Q$. Since $\mathrm{supp}(\llbracket C \rrbracket^\dagger (m)) = \bigcup_{\sigma \in \mathrm{supp}(m)} \llbracket C \rrbracket (\sigma)$, then there is also a $\tau \in \mathrm{supp}(\llbracket C \rrbracket^\dagger (m))$ such that $\tau \in Q$, so $\llbracket C \rrbracket^\dagger (m) \vDash \Diamond Q$, therefore $\vDash \langle P \rangle\ C\ \langle \Diamond Q \rangle$.

□

C SOUNDNESS AND COMPLETENESS OF OUTCOME LOGIC

We provide a formal definition of assertion entailment $\varphi \vDash e = u$, which informally means that φ has enough information to determine that the expression e evaluates to the value u.

Definition C.1 (Assertion Entailment). Given an outcome assertion φ, an expression e, and a weight $u \in U$, we define the following:

$$\varphi \vDash e = u \quad \text{iff} \quad \forall m \in \varphi, \sigma \in \mathrm{supp}(m). \quad \llbracket e \rrbracket (\sigma) = u$$

Occasionally we will also write $\varphi \vDash b$ for some test b, which is shorthand for $\varphi \vDash b = \mathbb{1}$. It is relatively easy to see that the following statements hold given this definition:

$$\begin{array}{rcll}
\top \vDash e = u & \text{iff} & \forall \sigma \in \Sigma. & \llbracket e \rrbracket (\sigma) = u \\
\bot \vDash e = u & \text{always} & & \\
\varphi \vee \psi \vDash e = u & \text{iff} & \varphi \vDash e = u \quad \text{and} & \psi \vDash e = u \\
\varphi \wedge \psi \vDash e = u & \text{iff} & \varphi \vDash e = u \quad \text{or} & \psi \vDash e = u \\
\varphi \oplus \psi \vDash e = u & \text{iff} & \varphi \vDash e = u \quad \text{and} & \psi \vDash e = u \\
\varphi^{(v)} \vDash e = u & \text{iff} & v = \mathbb{0} \quad \text{or} & \varphi \vDash e = u \\
\mathbf{1}_m \vDash e = u & \text{iff} & \forall \sigma \in \mathrm{supp}(m). & \llbracket e \rrbracket (\sigma) = u \\
P \vDash e = u & \text{iff} & \forall \sigma \in P. & \llbracket e \rrbracket (\sigma) = u \\
\Box P \vDash e = u & \text{iff} & P \vDash e = u &
\end{array}$$

We now present the soundness proof, following the sketch from Section 4.1. The first results pertain to the semantics of iteration. We start by recalling the characteristic function:

$$\Phi_{\langle C,e,e' \rangle}(f)(\sigma) = \llbracket e \rrbracket (\sigma) \cdot f^\dagger(\llbracket C \rrbracket (\sigma)) + \llbracket e' \rrbracket (\sigma) \cdot \eta(\sigma)$$

Note that as defined in Figure 1, $\left\llbracket C^{\langle e,e' \rangle} \right\rrbracket (\sigma) = (\mu f.\Phi_{\langle C,e,e' \rangle}(f))(\sigma)$. The first lemma relates $\Phi_{\langle C,e,e \rangle}$ to a sequence of unrolled commands.

Lemma C.2. *For all $n \in \mathbb{N}$ and $\sigma \in \Sigma$:*

$$\Phi^{n+1}_{\langle C,e,e' \rangle}(\lambda x.\mathbb{0})(\sigma) = \sum_{k=0}^{n} \left\llbracket (\mathsf{assume}\ e \fatsemi C)^k \fatsemi \mathsf{assume}\ e' \right\rrbracket (\sigma)$$

Proof. By mathematical induction on n.

- $n = 0$. Unfolding the definition of $\Phi_{\langle C,e_1,e_2\rangle}$, we get:

$$\begin{aligned}
&\Phi_{\langle C,e,e'\rangle}(\lambda x.\mathbb{0})(\sigma)\\
&= [\![e]\!]\,(\sigma)\cdot(\lambda x.\mathbb{0})^\dagger([\![C]\!]\,(\sigma)) + [\![e']\!]\,(\sigma)\cdot\eta(\sigma)\\
&= \mathbb{0} + [\![e']\!]\,(\sigma)\cdot\eta(\sigma)\\
&= [\![\mathsf{assume}\ e']\!]\,(\sigma)\\
&= \left[\!\!\left[(\mathsf{assume}\ e\fatsemi C)^0\fatsemi \mathsf{assume}\ e'\right]\!\!\right](\sigma)
\end{aligned}$$

- Inductive step, suppose the claim holds for n:

$$\begin{aligned}
&\Phi^{n+2}_{\langle C,e,e'\rangle}(\lambda x.\mathbb{0})(\sigma)\\
&= [\![e]\!]\,(\sigma)\cdot\Phi^{n+1}_{(\langle b,C\rangle)}(\lambda x.\mathbb{0})^\dagger([\![C]\!]\,(\sigma)) + [\![e']\!]\,(\sigma)\cdot\eta(\sigma)
\end{aligned}$$

By the induction hypothesis

$$\begin{aligned}
&= [\![e]\!]\,(\sigma)\cdot\left(\lambda\tau.\sum_{k=0}^{n}\left[\!\!\left[(\mathsf{assume}\ e\fatsemi C)^k\fatsemi\mathsf{assume}\ e'\right]\!\!\right](\tau)\right)^\dagger([\![C]\!]\,(\sigma))\\
&\qquad + [\![e']\!]\,(\sigma)\cdot\eta(\sigma)\\
&= \sum_{k=1}^{n+1}\left[\!\!\left[(\mathsf{assume}\ e\fatsemi C)^k\fatsemi\mathsf{assume}\ e'\right]\!\!\right](\sigma) + [\![\mathsf{assume}\ e']\!]\,(\sigma)\\
&= \sum_{k=0}^{n+1}\left[\!\!\left[(\mathsf{assume}\ e\fatsemi C)^k\fatsemi\mathsf{assume}\ e'\right]\!\!\right](\sigma)
\end{aligned}$$

□

Lemma 4.3. *The following equation holds:*

$$\left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\sigma) = \sum_{n\in\mathbb{N}}[\![(\mathsf{assume}\ e\fatsemi C)^n\fatsemi\mathsf{assume}\ e']\!]\,(\sigma)$$

Proof. First, by the Kleene fixed point theorem and the semantics of programs (Figure 1), we get:

$$\left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right](\sigma) = \sup_{n\in\mathbb{N}}\Phi^n_{\langle C,e,e'\rangle}(\lambda x.\mathbb{0})(\sigma)$$

Now, since $\Phi^0_{\langle C,e,e'\rangle}(\lambda x.\mathbb{0})(\sigma) = \mathbb{0}$ and $\mathbb{0}$ is the bottom of the order $\sqsubseteq$, we can rewrite the supremum as follows.

$$= \sup_{n\in\mathbb{N}}\Phi^{n+1}_{\langle C,e,e'\rangle}(\lambda x.\mathbb{0})(\sigma)$$

By Lemma C.2:

$$= \sup_{n\in\mathbb{N}}\sum_{k=0}^{n}\left[\!\!\left[(\mathsf{assume}\ e\fatsemi C)^k\fatsemi\mathsf{assume}\ e'\right]\!\!\right](\sigma)$$

Since we use the natural order, then $\sup(u, u+v) = u+v$, and therefore the supremum above is the following infinite sum:

$$= \sum_{n\in\mathbb{N}}[\![(\mathsf{assume}\ e\fatsemi C)^n\fatsemi\mathsf{assume}\ e']\!]\,(\sigma)$$

□

Theorem 4.2 (Soundness). $\Omega\vdash\langle\varphi\rangle\ C\ \langle\psi\rangle \implies\ \vDash\langle\varphi\rangle\ C\ \langle\psi\rangle$

Proof. The triple $\langle\varphi\rangle\ C\ \langle\psi\rangle$ is proven using inference rules from Figure 3, or by applying an axiom in Ω. If the last step is using an axiom, then the proof is trivial since we assumed that all the axioms in Ω are semantically valid. If not, then the proof is by induction on the derivation $\Omega\vdash\langle\varphi\rangle\ C\ \langle\psi\rangle$.

- Skip. We need to show that $\vDash\langle\varphi\rangle\ \mathsf{skip}\ \langle\varphi\rangle$. Suppose that $m\vDash\varphi$. Since $[\![\mathsf{skip}]\!]^\dagger(m) = m$, then clearly $[\![\mathsf{skip}]\!]^\dagger(m)\vDash\varphi$.
- Seq. Given that $\Omega\vdash\langle\varphi\rangle\ C_1\ \langle\vartheta\rangle$ and $\Omega\vdash\langle\vartheta\rangle\ C_2\ \langle\psi\rangle$, we need to show that $\vDash\langle\varphi\rangle\ C_1\fatsemi C_2\ \langle\psi\rangle$. Note that since $\Omega\vdash\langle\varphi\rangle\ C_1\ \langle\vartheta\rangle$ and $\Omega\vdash\langle\vartheta\rangle\ C_2\ \langle\psi\rangle$, those triples must be derived either using inference rules (in which case the induction hypothesis applies), or by applying an axiom in Ω. In either case, we can conclude that $\vDash\langle\varphi\rangle\ C_1\ \langle\vartheta\rangle$ and $\vDash\langle\vartheta\rangle\ C_2\ \langle\psi\rangle$. Suppose that $m\vDash\varphi$. Since $\vDash\langle\varphi\rangle\ C_1\ \langle\vartheta\rangle$, we get that $[\![C_1]\!]^\dagger(m)\vDash\vartheta$ and using $\vDash\langle\vartheta\rangle\ C_2\ \langle\psi\rangle$, we get that $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m))\vDash\psi$. Since $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) = [\![C_1\fatsemi C_2]\!]^\dagger(m)$, we are done.
- Plus. Given that $\Omega\vdash\langle\varphi\rangle\ C_1\ \langle\psi_1\rangle$ and $\Omega\vdash\langle\varphi\rangle\ C_2\ \langle\psi_2\rangle$, we need to show that $\vDash\langle\varphi\rangle\ C_1+C_2\ \langle\psi_1\oplus\psi_2\rangle$. Suppose that $m\vDash\varphi$, so by the induction hypotheses, $[\![C_1]\!]^\dagger(m)\vDash\psi_1$ and $[\![C_2]\!]^\dagger(m)\vDash\psi_2$. Recall from the remark at the end of Section 2.3 that we are assuming that programs are well-formed, and therefore $[\![C_1+C_2]\!]^\dagger(m)$ is defined and it is equal to $[\![C_1]\!]^\dagger(m)+[\![C_2]\!]^\dagger(m)$. Therefore by the semantics of $\oplus$, $[\![C_1+C_2]\!]^\dagger(m)\vDash\psi_1\oplus\psi_2$.
- Assume. Given $\varphi\vDash e = u$, we must show $\vDash\langle\varphi\rangle\ \mathsf{assume}\ e\ \langle\varphi^{(u)}\rangle$. Suppose $m\vDash\varphi$. Since $\varphi\vDash e = u$, then $[\![e]\!]\,(\sigma) = u$ for all $\sigma\in\mathsf{supp}(m)$. This means that:

$$\begin{aligned}
[\![\mathsf{assume}\ e]\!]^\dagger(m) &= (\lambda\sigma.\,[\![e]\!]\,(\sigma)\cdot\eta(\sigma))^\dagger(m)\\
&= (\lambda\sigma.u\cdot\eta(\sigma))^\dagger(m)\\
&= u\cdot m
\end{aligned}$$

And by definition, $u\cdot m\vDash\varphi^{(u)}$, so we are done.
- Iter. We know that $\vDash\langle\varphi_n\rangle\ \mathsf{assume}\ e\fatsemi C\ \langle\varphi_{n+1}\rangle$ and that $\vDash\langle\varphi_n\rangle\ \mathsf{assume}\ e'\ \langle\psi_n\rangle$ for all $n\in\mathbb{N}$ by the induction hypotheses. Now, we need to show that $\vDash\langle\varphi_0\rangle\ C^{\langle e,e'\rangle}\ \langle\psi_\infty\rangle$. Suppose $m\vDash\varphi_0$. It is easy to see that for all $n\in\mathbb{N}$:

$$[\![(\mathsf{assume}\ e\fatsemi C)^n]\!]^\dagger(m)\vDash\varphi_n$$

and

$$[\![(\mathsf{assume}\ e\fatsemi C)^n\fatsemi\mathsf{assume}\ e']\!]^\dagger(m)\vDash\psi_n$$

by mathematical induction on n, and the two induction hypotheses. Now, since $(\psi_n)_{n\in\mathbb{N}}\rightsquigarrow\psi_\infty$, we also know that:

$$\sum_{n\in\mathbb{N}}[\![(\mathsf{assume}\ e\fatsemi C)^n\fatsemi\mathsf{assume}\ e']\!]^\dagger(m)\vDash\psi_\infty$$

Finally, by Lemma 4.3 we get that $\left[\!\!\left[C^{\langle e,e'\rangle}\right]\!\!\right]^\dagger(m)\vDash\psi_\infty$.
- False. We must show that $\vDash\langle\bot\rangle\ C\ \langle\varphi\rangle$. Suppose that $m\vDash\bot$. This is impossible, so the claim follows vacuously.
- True. We must show that $\vDash\langle\varphi\rangle\ C\ \langle\top\rangle$. Suppose $m\vDash\varphi$. It is trivial that $[\![C]\!]^\dagger(m)\vDash\top$, so the triple is valid.
- Scale. By the induction hypothesis, we get that $\vDash\langle\varphi\rangle\ C\ \langle\psi\rangle$ and we must show that $\vDash\langle\varphi^{(u)}\rangle\ C\ \langle\psi^{(u)}\rangle$. Suppose $m\vDash\varphi^{(u)}$. If $u = \mathbb{0}$, then $m = \mathbb{0}$ and $[\![C]\!]^\dagger(\mathbb{0}) = \mathbb{0}$, and clearly $\mathbb{0}\vDash\psi^{(\mathbb{0})}$, so we are done. If $x\neq\mathbb{0}$, then there is some m' such that $m'\vDash\varphi$ and $m = u\cdot m'$. We therefore get that $[\![C]\!]^\dagger(m')\vDash\psi$. Now, observe that $[\![C]\!]^\dagger(m) = [\![C]\!]^\dagger(u\cdot m') = u\cdot[\![C]\!]^\dagger(m)$. Finally, by the definition of $(-)^{(u)}$, we get that $u\cdot[\![C]\!]^\dagger(m)\vDash\psi^{(u)}$.
- Disj. By the induction hypothesis, we know that $\vDash\langle\varphi_1\rangle\ C\ \langle\psi_1\rangle$ and $\vDash\langle\varphi_2\rangle\ C\ \langle\psi_2\rangle$ and we need to show $\vDash\langle\varphi_1\vee\varphi_2\rangle\ C\ \langle\psi_1\vee\psi_2\rangle$. Suppose $m\vDash\varphi_1\vee\varphi_2$. Without loss of generality, suppose $m\vDash$

φ_1. By the induction hypothesis, we get $[\![C]\!]^\dagger(m) \vDash \psi_1$. We can weaken this to conclude that $[\![C]\!]^\dagger(m) \vDash \psi_1 \vee \psi_2$. The case where instead $m \vDash \varphi_2$ is symmteric.

- Conj. By the induction hypothesis, we get that $\vDash \langle\varphi_1\rangle\ C\ \langle\psi_1\rangle$ and $\vDash \langle\varphi_2\rangle\ C\ \langle\psi_2\rangle$ and we need to show $\vDash \langle\varphi_1 \wedge \varphi_2\rangle\ C\ \langle\psi_1 \wedge \psi_2\rangle$. Suppose $m \vDash \varphi_1 \wedge \varphi_2$, so $m \vDash \varphi_1$ and $m \vDash \varphi_2$. By the induction hypotheses, $[\![C]\!]^\dagger(m) \vDash \psi_1$ and $[\![C]\!]^\dagger(m) \vDash \psi_2$, so $[\![C]\!]^\dagger(m) \vDash \psi_1 \wedge \psi_2$.
- Choice. By the induction hypothesis, $\vDash \langle\varphi_1\rangle\ C\ \langle\psi_1\rangle$ and $\vDash \langle\varphi_2\rangle\ C\ \langle\psi_2\rangle$ and we need to show $\vDash \langle\varphi_1 \oplus \varphi_2\rangle\ C\ \langle\psi_1 \oplus \psi_2\rangle$. Suppose $m \vDash \varphi_1 \oplus \varphi_2$, so $m_1 \vDash \varphi_1$ and $m_2 \vDash \varphi_2$ such that $m = m_1 + m_2$. By the induction hypotheses, $[\![C]\!]^\dagger(m_1) \vDash \psi_1$ and $[\![C]\!]^\dagger(m_2) \vDash \psi_2$. Now, since $[\![C]\!]^\dagger(m) = [\![C]\!]^\dagger(m_1+m_2) = [\![C]\!]^\dagger(m_1)+[\![C]\!]^\dagger(m_2)$, we get that $[\![C]\!]^\dagger(m) \vDash \psi_1 \oplus \psi_2$.
- Exists. By the induction hypothesis, $\vDash \langle\phi(t)\rangle\ C\ \langle\phi'(t)\rangle$ for all $t \in T$ and we need to show $\vDash \langle\exists x : T.\phi(x)\rangle\ C\ \langle\exists x : T.\phi'(x)\rangle$. Now suppose $m \in \exists x : T.\phi(x) = \bigcup_{t\in T}\phi(t)$. This means that there is some $t \in T$ such that $m \in \phi(t)$. By the induction hypothesis, this means that $[\![C]\!]^\dagger(m) \vDash \phi'(t)$, so we get that $[\![C]\!]^\dagger(m) \vDash \exists x : T.\phi'(x)$.
- Consequence. We know that $\varphi' \Rightarrow \varphi$ and $\psi \Rightarrow \psi'$ and by the induction hypothesis $\vDash \langle\varphi\rangle\ C\ \langle\psi\rangle$, and we need to show that $\vDash \langle\varphi'\rangle\ C\ \langle\psi'\rangle$. Suppose that $m \vDash \varphi'$, then it also must be the case that $m \vDash \varphi$. By the induction hypothesis, $[\![C]\!]^\dagger(m) \vDash \psi$. Now, using the second consequence $[\![C]\!]^\dagger(m) \vDash \psi'$.

□

Now, moving to completeness, we prove the following lemma.

Lemma 4.5. $\Omega \vdash \langle\varphi\rangle\ C\ \langle\mathrm{post}(C,\varphi)\rangle$

Proof. By induction on the structure of the program C.

- $C = \mathsf{skip}$. Since $[\![\mathsf{skip}]\!]^\dagger(m) = m$ for all m, then clearly $\mathrm{post}(\mathsf{skip},\varphi) = \varphi$. We complete the proof by applying the Skip rule.

$$\dfrac{}{\langle\varphi\rangle\ \mathsf{skip}\ \langle\varphi\rangle}\ \textsc{Skip}$$

- $C = C_1 \fatsemi C_2$. First, observe that:

$$\begin{aligned}\mathrm{post}(C_1 \fatsemi C_2,\varphi) &= \{[\![C_1 \fatsemi C_2]\!]^\dagger(m) \mid m \in \varphi\}\\ &= \{[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) \mid m \in \varphi\}\\ &= \{[\![C_2]\!]^\dagger(m') \mid m' \in \{[\![C_1]\!]^\dagger(m) \mid m \in \varphi\}\}\\ &= \mathrm{post}(C_2,\mathrm{post}(C_1,\varphi))\end{aligned}$$

Now, by the induction hypothesis, we know that:

$$\Omega \vdash \langle\varphi\rangle\ C_1\ \langle\mathrm{post}(C_1,\varphi)\rangle$$
$$\Omega \vdash \langle\mathrm{post}(C_1,\varphi)\rangle\ C_2\ \langle\mathrm{post}(C_1 \fatsemi C_2,\varphi)\rangle$$

Now, we complete the derivation as follows:

$$\dfrac{\dfrac{\Omega}{\langle\varphi\rangle\ C_1\ \langle\mathrm{post}(C_1,\varphi)\rangle} \qquad \dfrac{\Omega}{\langle\mathrm{post}(C_1,\varphi)\rangle\ C_2\ \langle\mathrm{post}(C_1 \fatsemi C_2,\varphi)\rangle}}{\langle\varphi\rangle\ C_1 \fatsemi C_2\ \langle\mathrm{post}(C_1 \fatsemi C_2,\varphi)\rangle}\ \textsc{Seq}$$

- $C = C_1 + C_2$. So, we have that:

$$\begin{aligned}\mathrm{post}(C_1 + C_2,\varphi) &= \{[\![C_1 + C_2]\!]^\dagger(m) \mid m \in \varphi\}\\ &= \{[\![C_1]\!]^\dagger(m) + [\![C_2]\!]^\dagger(m) \mid m \in \varphi\}\\ &= \bigcup_{m\in\varphi}\{[\![C_1]\!]^\dagger(m) + [\![C_2]\!]^\dagger(m) \mid m \in \mathbf{1}_m\}\\ &= \exists m : \varphi.\ \mathrm{post}(C_1,\mathbf{1}_m) \oplus \mathrm{post}(C_2,\mathbf{1}_m)\end{aligned}$$

We now complete the derivation as follows:

$$\dfrac{\dfrac{\dfrac{\Omega}{\langle\mathbf{1}_m\rangle\ C_1\ \langle\mathrm{post}(C_1,\mathbf{1}_m)\rangle} \qquad \dfrac{\Omega}{\langle\mathbf{1}_m\rangle\ C_2\ \langle\mathrm{post}(C_2,\mathbf{1}_m)\rangle}}{\langle\mathbf{1}_m\rangle\ C_1 + C_2\ \langle\mathrm{post}(C_1,\mathbf{1}_m) \oplus \mathrm{post}(C_2,\mathbf{1}_m)\rangle}\ \textsc{Plus}}{\langle\varphi\rangle\ C_1 + C_2\ \langle\mathrm{post}(C_1 + C_2,\varphi)\rangle}\ \textsc{Exists}$$

- $C = \mathsf{assume}\ e$, and e must either be a test b or a weight $u \in U$. Suppose that e is a test b. Now, for any m define the operator $b?m$ as follows:

$$b?m(\sigma) = \begin{cases}\mathbb{0} & \text{if} \quad [\![b]\!](\sigma) = \mathbb{0}\\ m(\sigma) & \text{if} \quad [\![b]\!](\sigma) = \mathbb{1}\end{cases}$$

Therefore $m = b?m + \neg b?m$ and $\mathbf{1}_m = \mathbf{1}_{b?m} \oplus \mathbf{1}_{\neg b?m}$. We also have:

$$\begin{aligned}\mathrm{post}(\mathsf{assume}\ b,\varphi) &= \{[\![\mathsf{assume}\ b]\!]^\dagger(m) \mid m \in \varphi\}\\ &= \{[\![\mathsf{assume}\ b]\!]^\dagger(b?m + \neg b?m) \mid m \in \varphi\}\\ &= \{b?m \mid m \in \varphi\}\\ &= \exists m : \varphi.\ \mathbf{1}_{b?m}\end{aligned}$$

Clearly also $\mathbf{1}_{b?m} \vDash b$ and $\mathbf{1}_{\neg b?m} \vDash \neg b$. We now complete the derivation:

$$\dfrac{\dfrac{\dfrac{\mathbf{1}_{b?m} \vDash b = \mathbb{1}}{\langle\mathbf{1}_{b?m}\rangle\ \mathsf{assume}\ b\ \langle\mathbf{1}_{b?m}^{(\mathbb{1})}\rangle}\ \textsc{Assume} \qquad \dfrac{\mathbf{1}_{\neg b?m} \vDash b = \mathbb{0}}{\langle\mathbf{1}_{\neg b?m}\rangle\ \mathsf{assume}\ b\ \langle\mathbf{1}_{\neg b?m}^{(\mathbb{0})}\rangle}\ \textsc{Assume}}{\langle\mathbf{1}_{b?m} \oplus \mathbf{1}_{\neg b?m}\rangle\ \mathsf{assume}\ b\ \langle\mathbf{1}_{b?m}\rangle}\ \textsc{Split}}{\langle\varphi\rangle\ \mathsf{assume}\ b\ \langle\mathrm{post}(\mathsf{assume}\ b,\varphi)\rangle}\ \textsc{Exists}$$

Now, suppose $e = u$, so $\varphi \vDash u = u$ and $[\![\mathsf{assume}\ u]\!]^\dagger(m) = u \cdot m$ for all $m \in \varphi$ and therefore $\mathrm{post}(\mathsf{assume}\ u,\varphi) = \varphi^{(u)}$. We can complete the proof as follows:

$$\dfrac{\varphi \vDash u = u}{\langle\varphi\rangle\ \mathsf{assume}\ u\ \langle\varphi^{(u)}\rangle}\ \textsc{Assume}$$

- $C = C^{\langle e,e'\rangle}$. For all $n \in \mathbb{N}$, let $\varphi_n(m)$ and $\psi_n(m)$ be defined as follows:

$$\begin{aligned}\varphi_n(m) &\triangleq \mathrm{post}((\mathsf{assume}\ e \fatsemi C)^n,\mathbf{1}_m)\\ &= \mathbf{1}_{[\![(\mathsf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)}\\ \psi_n(m) &\triangleq \mathrm{post}(\mathsf{assume}\ e',\varphi_n(m))\\ &= \mathbf{1}_{[\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m)}\\ \psi_\infty(m) &\triangleq \mathrm{post}(C^{\langle e,e'\rangle},\mathbf{1}_m)\\ &= \mathbf{1}_{[\![C^{\langle e,e'\rangle}]\!]^\dagger(m)}\end{aligned}$$

Note that by definition, $\varphi_0(m) = \mathbf{1}_m$, $\varphi = \exists m : \varphi.\varphi_0(m)$, and $\mathrm{post}(C^{\langle e,e'\rangle},\varphi) = \exists m : \varphi.\psi_\infty(m)$.

We now show that $(\psi_n)_{n\in\mathbb{N}^\infty}$ converges ($(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$). Take any $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \psi_n$ for each n. That means that $m_n = [\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m)$. Therefore by Lemma 4.3 we get that $\sum_{n\in\mathbb{N}} m_n = \left[\!\left[C^{\langle e,e'\rangle} \right]\!\right]^\dagger(m)$, and therefore $\sum_{n\in\mathbb{N}} m_n \vDash \psi_\infty(m)$. We now complete the derivation as follows:

$$\dfrac{\dfrac{\dfrac{\Omega}{\langle \varphi_n(m)\rangle\ \mathsf{assume}\ e \fatsemi C\ \langle \varphi_{n+1}(m)\rangle} \quad \vdots \quad \dfrac{\Omega}{\langle\varphi_n(m)\rangle\ \mathsf{assume}\ e'\ \langle \psi_n(m)\rangle}}{\langle\varphi_0(m)\rangle\ C^{\langle e,e'\rangle}\ \langle\psi_\infty(m)\rangle}\ \textsc{Iter}}{\langle\varphi\rangle\ C^{\langle e,e'\rangle}\ \langle \mathrm{post}(C^{\langle e,e'\rangle},\varphi)\rangle}\ \textsc{Exists}$$

- $C = a$. We assumed that Ω contains all the valid triples pertaining to atomic actions $a \in \mathsf{Act}$, so $\Omega \vdash \langle\varphi\rangle\ a\ \langle \mathrm{post}(a,\varphi)\rangle$ since $\vDash \langle\varphi\rangle\ a\ \langle\mathrm{post}(a,\varphi)\rangle$.

□

D DERIVED RULES

D.1 Sequencing in Hoare and Lisbon Logic

We first prove the results about sequencing Hoare Logic encodings.

Lemma D.1. *The following inference is derivable.*

$$\dfrac{\langle P\rangle\ C\ \langle \square Q\rangle}{\langle \square P\rangle\ C\ \langle\square Q\rangle}$$

Proof. Before we show the derivation, we argue that:

$$\exists u : U.\left(\exists v : U.Q^{(v)}\right)^{(u)} \Rightarrow \exists w : U.Q^{(w)}$$

Suppose that $m \vDash \exists u : U.\left(\exists v : U.Q^{(v)}\right)^{(u)}$, this means that there is some m' such that $m = u \cdot m'$ for some $u \in U$ and $m' \vDash \exists v : U.Q^{(v)}$. Now, there must also be $m'' = v \cdot m'$ for some $v \in U$ and $m'' \vDash Q$. This means that $m = (u\cdot v)\cdot m''$, so $m \vDash Q^{(u\cdot v)}$. Now let $w = u \cdot v$, so clearly $m \vDash \exists w : U.Q^{(w)}$. Using this as a consequence, we now complete the derivation.

$$\dfrac{\dfrac{\dfrac{\forall u \in U.\quad \dfrac{\dfrac{\langle P\rangle\ C\ \langle\square Q\rangle}{\langle P\rangle\ C\ \langle \exists v : U.Q^{(v)}\rangle}}{\langle P^{(u)}\rangle\ C\ \langle (\exists v : U.Q^{(v)})^{(u)}\rangle}\ \textsc{Scale}}{\langle \exists u : U.P^{(u)}\rangle\ C\ \langle \exists u : U.(\exists v : U.Q^{(v)})^{(u)}\rangle}\ \textsc{Exists}}{\langle \exists u : U.P^{(u)}\rangle\ C\ \langle \exists w : U.Q^{(w)}\rangle}\ \textsc{Consequence}}{\langle\square P\rangle\ C\ \langle\square Q\rangle}$$

□

Lemma D.2. *The following inference is derivable.*

$$\dfrac{\langle P\rangle\ C_1\ \langle\square Q\rangle \qquad \langle Q\rangle\ C_2\ \langle\square R\rangle}{\langle P\rangle\ C_1 \fatsemi C_2\ \langle\square R\rangle}$$

Proof.

$$\dfrac{\langle P\rangle\ C_1\ \langle\square Q\rangle \qquad \dfrac{\langle Q\rangle\ C_2\ \langle\square R\rangle}{\langle\square Q\rangle\ C_2\ \langle\square R\rangle}\ \textsc{Lemma D.1}}{\langle P\rangle\ C_1\fatsemi C_2\ \langle\square R\rangle}\ \textsc{Seq}$$

□

Now, we turn to Lisbon Logic

Lemma D.3. *The following inference is derivable.*

$$\dfrac{\langle P\rangle\ C\ \langle\Diamond Q\rangle}{\langle\Diamond P\rangle\ C\ \langle\Diamond Q\rangle}$$

Proof. First, let $U' = U \setminus \{\mathbb{0}\}$. The derivation is done as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle P\rangle\ C\ \langle\Diamond Q\rangle}{\langle P\rangle\ C\ \langle \exists v : U'.Q^{(v)} \oplus \top\rangle}}{\langle P^{(u)}\rangle\ C\ \langle (\exists v : U'.Q^{(v)}\oplus\top)^{(u)}\rangle}\ \textsc{Scale} \qquad \dfrac{}{\langle\top\rangle\ C\ \langle\top\rangle}\ \textsc{True}}{\langle P^{(u)}\oplus\top\rangle\ C\ \langle (\exists v : U'.Q^{(v)}\oplus\top)^{(u)}\oplus\top\rangle}\ \textsc{Split}}{\langle \exists u : U'.P^{(u)}\oplus\top\rangle\ C\ \langle \exists u : U'.(\exists v : U'.Q^{(v)}\oplus\top)^{(u)}\oplus\top\rangle}\ \textsc{Exists}}{\langle\Diamond P\rangle\ C\ \langle\Diamond Q\rangle}\ \textsc{Cons}$$

Now, we show that the application of the rule of consequence (abbreviated at Cons) is valid as follows. First, we note that weighting by u distributes over the existential quanitifier and $\oplus$, and that $\top^{(u)} \Rightarrow \top$.

$$\begin{aligned} &\exists u : U'.(\exists v : U'.Q^{(v)} \oplus \top)^{(u)} \oplus \top \\ &\quad\implies \exists u : U'.\exists v : U'.Q^{(v\cdot u)} \oplus \top \oplus \top \end{aligned}$$

Now, since $u \neq \mathbb{0}$ and $v \neq \mathbb{0}$, then $v \cdot u \neq \mathbb{0}$ (see remark at the end of Section 5.1), therefore we can combine the two existentials into a single variable $w \neq \mathbb{0}$. Similarly, $\top\oplus\top \Rightarrow \top$.

$$\begin{aligned} &\implies \exists w : U'.Q^{(w)} \oplus \top \\ &\implies \Diamond Q \end{aligned}$$

□

Lemma D.4. *The following inference is derivable.*

$$\dfrac{\langle P\rangle\ C_1\ \langle\Diamond Q\rangle \qquad \langle Q\rangle\ C_2\ \langle\Diamond R\rangle}{\langle P\rangle\ C_1\fatsemi C_2\ \langle\Diamond R\rangle}$$

Proof.

$$\dfrac{\langle P\rangle\ C_1\ \langle\Diamond Q\rangle \qquad \dfrac{\langle Q\rangle\ C_2\ \langle\Diamond R\rangle}{\langle\Diamond Q\rangle\ C_2\ \langle\Diamond R\rangle}\ \textsc{Lemma D.3}}{\langle P\rangle\ C_1\fatsemi C_2\ \langle\Diamond R\rangle}\ \textsc{Seq}$$

□

D.2 If Statements and While Loops

Lemma D.5. *The following inference is derivable.*

$$\dfrac{\varphi_1 \vDash b \qquad \langle\varphi_1\rangle\ C_1\ \langle\psi_1\rangle \qquad \varphi_2 \vDash \neg b \qquad \langle\varphi_2\rangle\ C_2\ \langle\psi_2\rangle}{\langle\varphi_1\oplus\varphi_2\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle\psi_2\oplus\psi_2\rangle}\ \textit{If}$$

Proof. First note that $\varphi \vDash b$ is syntactic sugar for $\varphi \vDash b = \mathbb{1}$, and so from the assumptions that $\varphi_1 \vDash b$ and $\varphi_2 \vDash \neg b$, we get $\varphi_1 \vDash b = \mathbb{1}$, $\varphi_2 \vDash b = \mathbb{0}$, $\varphi_1 \vDash \neg b = \mathbb{0}$, and $\varphi_2 \vDash \neg b = \mathbb{1}$. We split the derivation into two parts. Part (1) is shown below:

$$\dfrac{\dfrac{\dfrac{\varphi_1 \vDash b = \mathbb{1}}{\langle \varphi_1 \rangle \text{ assume } b \ \langle \varphi_1^{(\mathbb{1})} \rangle}\textsc{Assume} \quad \dfrac{\varphi_2 \vDash b = \mathbb{0}}{\langle \varphi_2 \rangle \text{ assume } b \ \langle \varphi_2^{(\mathbb{0})} \rangle}\textsc{Assume}}{\langle \varphi_1 \oplus \varphi_2 \rangle \text{ assume } b \ \langle \varphi_1 \rangle}\textsc{Split} \quad \vdots \quad \langle \varphi_1 \rangle\ C_1\ \langle \psi_1 \rangle}{\langle \varphi_1 \oplus \varphi_2 \rangle \text{ assume } b \fatsemi C_1 \ \langle \psi_1 \rangle}\textsc{Seq}$$

We omit the proof with part (2), since it is nearly identical. Now, we combine (1) and (2):

$$\dfrac{\dfrac{(1) \qquad (2)}{\langle \varphi_1 \oplus \varphi_2 \rangle \ (\text{assume } b \fatsemi C_1) + (\text{assume } \neg b \fatsemi C_2) \ \langle \psi_1 \oplus \psi_2 \rangle}\textsc{Plus}}{\langle \varphi_1 \oplus \varphi_2 \rangle \text{ if } b \text{ then } C_1 \text{ else } C_2 \ \langle \psi_1 \oplus \psi_2 \rangle}$$

□

Lemma D.6. *For any assertion P and test b:*

$$P \implies \exists u : U. \exists v : \{v \in V \mid u + v = \mathbb{1}\}.(P \wedge b)^{(u)} \oplus (P \wedge \neg b)^{(v)}$$

Proof. Suppose $m \vDash P$, so $|m| = \mathbb{1}$ and $\text{supp}(m) \subseteq P$. Now, let:

$$m_1(\sigma) \triangleq \begin{cases} m(\sigma) & \text{if} \quad [\![b]\!](\sigma) = \mathbb{1} \\ \mathbb{0} & \text{if} \quad [\![b]\!](\sigma) = \mathbb{0} \end{cases}$$

$$m_2(\sigma) \triangleq \begin{cases} \mathbb{0} & \text{if} \quad [\![b]\!](\sigma) = \mathbb{1} \\ m(\sigma) & \text{if} \quad [\![b]\!](\sigma) = \mathbb{0} \end{cases}$$

Clearly, since b is a test, $m = m_1 + m_2$. Now, let $u = |m_1|$ and $v = |m_2|$. Since $|m| = \mathbb{1}$, then $u + v = \mathbb{1}$. By construction, $m_1 \vDash (P \wedge b)^{(u)}$ and $m_2 \vDash (P \wedge \neg b)^{(v)}$, so $m \vDash (P \wedge b)^{(u)} \oplus (P \wedge \neg b)^{(v)}$. By existentially quantifying u and v, we get:

$$m \vDash \exists u : U. \exists v : \{v \in U \mid u + v = \mathbb{1}\}.(P \wedge b)^{(u)} \oplus (P \wedge \neg b)^{(v)}$$

□

Lemma D.7. $Q^{(u)} \oplus Q^{(v)} \implies Q^{(u+v)}$.

Proof. Suppose $m \vDash Q^{(u)} \oplus Q^{(v)}$, so there are m_1 and m_2 such that $m = m_1 + m_2$ and $|m_1| = u$ and $|m_2| = v$, and $\text{supp}(m_1) \subseteq Q$ and $\text{supp}(m_2) \subseteq Q$. We also have that $|m| = |m_1| + |m_2| = u + v$ and $\text{supp}(m) = \text{supp}(m_1) \cup \text{supp}(m_2) \subseteq Q$, so $m \vDash Q^{(u+v)}$. □

Lemma D.8 (Hoare Logic If Rule). *The following inference is derivable.*

$$\dfrac{\langle P \wedge b \rangle\ C_1\ \langle \Box Q \rangle \qquad \langle P \wedge \neg b \rangle\ C_2\ \langle \Box Q \rangle}{\langle P \rangle \text{ if } b \text{ then } C_1 \text{ else } C_2 \ \langle \Box Q \rangle}\textit{If (Hoare)}$$

Proof. The derivation is shown in Figure 4a. The application of the rule of Consequence uses Lemmas D.6 and D.7. The sets in the existential quantifiers are omitted for brevity, but should read $u \in U$ and $v \in \{v \in U \mid u + v = \mathbb{1}\}$, so we know that $u + v = \mathbb{1}$. The implication in the postcondition is justified as follows. First, we simply unfold the definition of $\Box$.

$$\exists u, v.(\Box Q)^{(u)} \oplus (\Box Q)^{(v)} \implies \exists u, v.(\exists w.Q^{(w)})^{(u)} \oplus (\exists z.Q^{(z)})^{(v)}$$

Next, we can lift the existential quantifiers to the top level since w and z are fresh variables and do not affect u or v. We can also collapse the two weighting operations

$$\implies \exists u, v, w, z.Q^{(w \cdot u)} \oplus Q^{(z \cdot v)}$$

By Lemma D.7.

$$\implies \exists u, v, w, z.Q^{(w \cdot u + z \cdot v)}$$
$$\implies \Box Q$$

The application of If also introduces proof obligations for $(P \wedge b)^{(u)} \vDash b$ and $(P \wedge \neg b)^{(v)} \vDash \neg b$, which both hold trivially.

□

Lemma D.9. $(\varphi \oplus \psi)^{(u)} \implies \varphi^{(u)} \oplus \psi^{(u)}$

Proof. Suppose $m \vDash (\varphi \oplus \psi)^{(u)}$, therefore there is m' such that $m = u \cdot m'$ and $m' \vDash \varphi \oplus \psi$. Therefore, there is also m_1 and m_2 such that $m' = m_1 + m_2$ and $m_1 \vDash \varphi$ and $m_2 \vDash \psi$. So, this means that $u \cdot m_1 \vDash \varphi^{(u)}$ and $u \cdot m_2 \vDash \psi^{(u)}$ and since $m = u \cdot m' = u \cdot (m_1 + m_2) = u \cdot m_1 + u \cdot m_2$, we get that $m \vDash \varphi^{(u)} \oplus \psi^{(u)}$. □

Lemma D.10 (Lisbon Logic If Rule). *The following inference is derivable.*

$$\dfrac{\langle P \wedge b \rangle\ C_1\ \langle \Diamond Q \rangle \qquad \langle P \wedge \neg b \rangle\ C_2\ \langle \Diamond Q \rangle}{\langle P \rangle \text{ if } b \text{ then } C_1 \text{ else } C_2 \ \langle \Diamond Q \rangle}\textit{If (Lisbon)}$$

Proof. The derivation is shown in Figure 4b. The implication in the postcondition is justified as follows. First, we simply unfold the definition of $\Diamond$. Also, note that although the sets are omitted, we know that $u, v, w, z \in U$ such that $u + v = \mathbb{1}$, $w \neq \mathbb{0}$ and $z \neq \mathbb{0}$.

$$\exists u, v.(\Diamond Q)^{(u)} \oplus (\Diamond Q)^{(v)}$$
$$\implies \exists u, v.(\exists w.Q^{(w)} \oplus \top)^{(u)} \oplus (\exists z.Q^{(z)} \oplus \top)^{(v)}$$

Next, we can lift the existential quantifiers to the top level since w and z are fresh variables and do not affect u, v or $\top$. We can also collapse the two weighting operations, and rearrange the terms of $\oplus$, including using Lemma D.9 and $\top^{(u)} \oplus \top^{(v)} \Rightarrow \top$.

$$\implies \exists u, v, w, z.Q^{(w \cdot u)} \oplus Q^{(z \cdot v)} \oplus \top$$

By Lemma D.7.

$$\implies \exists u, v, w, z.Q^{(w \cdot u + z \cdot v)} \oplus \top$$

Since $u + v = \mathbb{1}$, then one of u or v is nonzero. We also know that w and z are nonzero, so if $u \neq \mathbb{0}$ then $w \cdot u \neq \mathbb{0}$ and therefore $w \cdot u + z \cdot v \neq \mathbb{0}$. The same is true if instead $v \neq \mathbb{0}$.

$$\implies \exists x : (U \setminus \{\mathbb{0}\}).Q^{(x)} \oplus \top$$
$$\implies \Diamond Q$$

The application of If also introduces proof obligations for $(P \wedge b)^{(u)} \vDash b$ and $(P \wedge \neg b)^{(v)} \vDash \neg b$, which both hold trivially. □

Lemma D.11 (While Rule). *The following inference is derivable.*

$$\dfrac{(\psi_n)_{n \in \mathbb{N}} \rightsquigarrow \psi_\infty \quad \langle \varphi_n \rangle\ C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle \quad \varphi_n \vDash b \quad \psi_n \vDash \neg b}{\langle \varphi_0 \oplus \psi_0 \rangle \text{ while } b \text{ do } C \ \langle \psi_\infty \rangle}\textit{While}$$

$$\dfrac{(P\land b)^{(u)}\vDash b \qquad \dfrac{\langle P\land b\rangle\; C_1\; \langle \square Q\rangle}{\langle (P\land b)^{(u)}\rangle\; C_1\; \langle (\square Q)^{(u)}\rangle}\,\textsc{Scale} \qquad (P\land\neg b)^{(v)}\vDash \neg b \qquad \dfrac{\langle P\land \neg b\rangle\; C_2\; \langle \square Q\rangle}{\langle (P\land \neg b)^{(v)}\rangle\; C_1\; \langle (\square Q)^{(v)}\rangle}\,\textsc{Scale}}{\dfrac{\dfrac{\langle (P\land b)^{(u)}\oplus (P\land\neg b)^{(v)}\rangle\; \mathsf{if}\; b\; \mathsf{then}\; C_1\; \mathsf{else}\; C_2\; \langle (\square Q)^{(u)}\oplus(\square Q)^{(v)}\rangle}{\langle \exists u,v.(P\land b)^{(u)}\oplus (P\land\neg b)^{(v)}\rangle\; \mathsf{if}\; b\; \mathsf{then}\; C_1\; \mathsf{else}\; C_2\; \langle \exists u,v.(\square Q)^{(u)}\oplus(\square Q)^{(v)}\rangle}\,\textsc{Exists}}{\langle P\rangle\; \mathsf{if}\; b\; \mathsf{then}\; C_1\; \mathsf{else}\; C_2\; \langle \square Q\rangle}\,\textsc{Consequence}}\,\textsc{If}$$

(a) Derivation of If (Hoare) from Lemma D.8

$$\dfrac{(P\land b)^{(u)}\vDash b \qquad \dfrac{\langle P\land b\rangle\; C_1\; \langle \lozenge Q\rangle}{\langle (P\land b)^{(u)}\rangle\; C_1\; \langle (\lozenge Q)^{(u)}\rangle}\,\textsc{Scale} \qquad (P\land\neg b)^{(v)}\vDash \neg b \qquad \dfrac{\langle P\land \neg b\rangle\; C_2\; \langle \lozenge Q\rangle}{\langle (P\land \neg b)^{(v)}\rangle\; C_1\; \langle (\lozenge Q)^{(v)}\rangle}\,\textsc{Scale}}{\dfrac{\dfrac{\langle (P\land b)^{(u)}\oplus (P\land\neg b)^{(v)}\rangle\; \mathsf{if}\; b\; \mathsf{then}\; C_1\; \mathsf{else}\; C_2\; \langle (\lozenge Q)^{(u)}\oplus(\lozenge Q)^{(v)}\rangle}{\langle \exists u,v.(P\land b)^{(u)}\oplus (P\land\neg b)^{(v)}\rangle\; \mathsf{if}\; b\; \mathsf{then}\; C_1\; \mathsf{else}\; C_2\; \langle \exists u,v.(\lozenge Q)^{(u)}\oplus(\lozenge Q)^{(v)}\rangle}\,\textsc{Exists}}{\langle P\rangle\; \mathsf{if}\; b\; \mathsf{then}\; C_1\; \mathsf{else}\; C_2\; \langle \lozenge Q\rangle}\,\textsc{Consequence}}\,\textsc{If}$$

(b) Derivation of If (Lisbon) from Lemma D.10

Figure 4: Derivations of inference rules for if statements

Proof. First, let $\varphi'_n = \varphi_n \oplus \psi_n$. The derivation has two parts. First, part (1):

$$\dfrac{\dfrac{\varphi_n \vDash b}{\langle\varphi_n\rangle\; \mathsf{assume}\; b\; \langle\varphi_n\rangle}\,\textsc{Assume} \qquad \dfrac{\psi_n\vDash\neg b}{\langle\psi_n\rangle\; \mathsf{assume}\; b\; \langle(\psi_n)_{\mathbb{0}}\rangle}\,\textsc{Assume}}{\dfrac{\langle\varphi_n\oplus\psi_n\rangle\; \mathsf{assume}\; b\; \langle\varphi_n\rangle \qquad \vdots \qquad \langle\varphi_n\rangle\; C\; \langle\varphi_{n+1}\oplus\psi_{n+1}\rangle}{\langle\varphi_n\oplus\psi_n\rangle\; \mathsf{assume}\; b \fatsemi C\; \langle\varphi_{n+1}\oplus\psi_{n+1}\rangle}\,\textsc{Seq}}\,\textsc{Split}$$

Now part (2):

$$\dfrac{\dfrac{\varphi_n \vDash b}{\langle\varphi_n\rangle\; \mathsf{assume}\; \neg b\; \langle(\varphi_n)_{\mathbb{0}}\rangle}\,\textsc{Assume} \qquad \dfrac{\psi_n\vDash\neg b}{\langle\psi_n\rangle\; \mathsf{assume}\; \neg b\; \langle\psi_n\rangle}\,\textsc{Assume}}{\langle\varphi_n\oplus\psi_n\rangle\; \mathsf{assume}\; \neg b\; \langle\psi_n\rangle}\,\textsc{Split}$$

Finally, we complete the derivation as follows:

$$\dfrac{\dfrac{(1)}{\langle\varphi_n\oplus\psi_n\rangle\; \mathsf{assume}\; b\fatsemi C\; \langle\varphi_{n+1}\oplus\psi_{n+1}\rangle} \qquad \vdots \qquad \dfrac{(2)}{\langle\varphi_n\oplus\psi_n\rangle\; \mathsf{assume}\; \neg b\; \langle\psi_n\rangle}}{\langle\varphi_0\oplus\psi_0\rangle\; \mathsf{while}\; b\; \mathsf{do}\; C\; \langle\psi_\infty\rangle}\,\textsc{Iter}$$

□

D.3 Loop Invariants

Lemma D.12 (Loop Invariant Rule). *The following inference is derivable.*

$$\dfrac{\langle P\land b\rangle\; C\; \langle\square P\rangle}{\langle P\rangle\; \mathsf{while}\; b\; \mathsf{do}\; C\; \langle\square(P\land\neg b)\rangle}\,\textit{Invariant}$$

Proof. We will derive this rule using the While rule. For all n, let $\varphi_n = \square(P\land b)$ and $\psi_n = \square(P\land\neg b)$. We will now show that $(\psi_n)_{n\in\mathbb{N}^\infty}$ converges. Suppose that $m_n \vDash \square(P\land\neg b)$ for each $n\in\mathbb{N}$. That means that $\mathsf{supp}(m_n)\subseteq P\land\neg b$. We also have that $\mathsf{supp}(\sum_{n\in\mathbb{N}} m_n) = \bigcup_{n\in\mathbb{N}}\mathsf{supp}(m_n)$, and since each $m_n\subseteq P\land\neg b$, then their union must also be contained in $P\land\neg b$, and thus $\sum_{n\in\mathbb{N}} m_n\vDash\square(P\land\neg b)$. We remark that $\square(P\land b)\vDash b$ and $\square(P\land\neg b)\vDash\neg b$ trivially. We complete the derivation as follows:

$$\dfrac{\dfrac{\dfrac{\langle P\land b\rangle\; C\; \langle\square P\rangle}{\langle\square(P\land b)\rangle\; C\; \langle\square P\rangle}\,\textsc{Lemma D.1}}{\langle\square(P\land b)\rangle\; C\; \langle\square(P\land b)\oplus\square(P\land\neg b)\rangle}\,\textsc{Consequence}}{\dfrac{\langle\square(P\land b)\oplus\square(P\land\neg b)\rangle\; \mathsf{while}\; b\; \mathsf{do}\; C\; \langle\square(P\land\neg b)\rangle}{\langle P\rangle\; \mathsf{while}\; b\; \mathsf{do}\; C\; \langle\square(P\land\neg b)\rangle}\,\textsc{Consequence}}\,\textsc{While}$$

Both usages of the rule of Consequence follow from Lemma D.6. □

D.4 Loop Variants

Lemma D.13. *The following inference is derivable.*

$$\dfrac{\forall n<N.\qquad \varphi_0\vDash\neg b \qquad \varphi_{n+1}\vDash b \qquad \langle\varphi_{n+1}\rangle\; C\; \langle\varphi_n\rangle}{\langle\exists n:\mathbb{N}.\varphi_n\rangle\; \mathsf{while}\; b\; \mathsf{do}\; C\; \langle\varphi_0\rangle}\,\textit{Variant}$$

Proof. For the purpose of applying the While rule, we define the following for all n and N:

$$\varphi'_n = \begin{cases}\varphi_{N-n} & \text{if}\quad n<N\\ \top^{(\mathbb{0})} & \text{if}\quad n\geq N\end{cases} \qquad \psi_n = \begin{cases}\varphi_0 & \text{if}\quad n\in\{N,\infty\}\\ \top^{(\mathbb{0})} & \text{otherwise}\end{cases}$$

It is easy to see that $(\psi_n)_{n\in\mathbb{N}}\rightsquigarrow\psi_\infty$. Each ψ_n except for ψ_N and ψ_∞ is only satisfied by $\mathbb{0}$, so taking $(m_n)_{n\in\mathbb{N}}$ such that $m_n\vDash\psi_n$ for each $n\in\mathbb{N}$, it must be the case that $\sum_{n\in\mathbb{N}} m_n = m_N$. By assumption, we know that $m_N\vDash\psi_\infty$ since $\psi_\infty=\psi_N=\varphi_0$. We also know that $\varphi'_n\vDash b$ and $\psi_n\vDash\neg b$ by our assumptions and the fact that $\top^{(\mathbb{0})}\vDash e=u$ for any e and u. There are two cases for the premise of the While rule (1) where $n<N$ (left) and $n\geq N$ (right).

$$\dfrac{\dfrac{\forall m<N.\;\langle\varphi_{m+1}\rangle\; C\; \langle\varphi_m\rangle}{\forall n<N.\;\langle\varphi_{N-n}\rangle\; C\; \langle\varphi_{N-(n+1)}\rangle}}{\forall n<N.\;\langle\varphi'_n\rangle\; C\; \langle\varphi'_{n+1}\oplus\psi_{n+1}\rangle} \qquad \dfrac{\dfrac{\overline{\langle\top\rangle\; C\; \langle\top\rangle}\,\textsc{True}}{\langle\top^{(\mathbb{0})}\rangle\; C\; \langle\top^{(\mathbb{0})}\rangle}\,\textsc{Scale}}{\forall n\geq N.\;\langle\varphi'_n\rangle\; C\; \langle\varphi'_{n+1}\oplus\psi_{n+1}\rangle}$$

Finally, we complete the derivation.

$$\dfrac{\forall N\in\mathbb{N}.\quad \dfrac{\dfrac{(1)}{\langle\varphi'_n\rangle\ C\ \langle\varphi'_{n+1}\oplus\psi_{n+1}\rangle}}{\langle\varphi_N\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle\varphi_0\rangle}\ \textsc{While}}{\langle\exists n:\mathbb{N}.\varphi_n\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle\varphi_0\rangle}\ \textsc{Exists}$$

□

Lemma D.14 (Lisbon Logic Loop Variants). *The following inference is derivable.*

$$\dfrac{\forall n\in\mathbb{N}.\quad P_0\vDash\neg b\quad P_{n+1}\vDash b\quad \langle P_{n+1}\rangle\ C\ \langle\Diamond P_n\rangle}{\langle\exists n:\mathbb{N}.P_n\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle\Diamond P_0\rangle}\ \textit{Lisbon Variant}$$

Proof. First, for all $N\in\mathbb{N}$, let φ_n and ψ_n be defined as follows:

$$\varphi_n=\begin{cases}\Diamond P_{N-n} & \text{if}\quad n\le N\\ \top & \text{if}\quad n>N\end{cases}\qquad \psi_n=\begin{cases}\Diamond P_0 & \text{if}\quad n\in\{N,\infty\}\\ \top & \text{otherwise}\end{cases}$$

Now, we prove that $(\psi_n)_{n\in\mathbb{N}}\rightsquigarrow\psi_\infty$. Take any $(m_n)_{n\in\mathbb{N}}$ such that $m_n\vDash\psi_n$ for each $n\in\mathbb{N}$. Since $m_N\vDash\Diamond P_0$, then there is some $\sigma\in\mathsf{supp}(m_N)$ such that $\sigma\in P_0$. By definition $(\sum_{n\in\mathbb{N}}m_n)(\sigma)\ge m_N(\sigma)>\mathbb{0}$, so $\sum_{n\in\mathbb{N}}m_n\vDash\Diamond P_0$ as well. We opt to derive this rule with the Iter rule rather than While since it is inconvenient to split the assertion into components where b is true and false. We complete the derivation in two parts, and each part is broken into two cases. We start with (1), and the case where $n<N$. In this case, we know that $\varphi_n=\Diamond P_{N-n}$ and $\varphi_{n+1}=\Diamond P_{N-n-1}$ (even if $n=N-1$, then we get $\varphi_N=\Diamond P_0=\Diamond P_{N-(N-1)-1}$).

$$\dfrac{\dfrac{\dfrac{\dfrac{P_{N-n}\vDash b}{\langle P_{N-n}\rangle\ \mathsf{assume}\ b\ \langle P_{N-n}\rangle}\ \textsc{Assume}\quad \vdots\quad \langle P_{N-n}\rangle\ C\ \langle\Diamond P_{N-n-1}\rangle}{\langle P_{N-n}\rangle\ \mathsf{assume}\ b\fatsemi C\ \langle\Diamond P_{N-n-1}\rangle}\ \textsc{Seq}}{\langle\Diamond P_{N-n}\rangle\ \mathsf{assume}\ b\fatsemi C\ \langle\Diamond P_{N-n-1}\rangle}\ \text{Lemma D.3}}{\langle\varphi_n\rangle\ \mathsf{assume}\ b\fatsemi C\ \langle\varphi_{n+1}\rangle}$$

Now, we prove (1) where $n\ge N$, $\varphi_{n+1}=\top$.

$$\dfrac{\dfrac{}{\langle\varphi_n\rangle\ \mathsf{assume}\ b\fatsemi C\ \langle\top\rangle}\ \textsc{True}}{\langle\varphi_n\rangle\ \mathsf{assume}\ b\fatsemi C\ \langle\varphi_{n+1}\rangle}$$

We now move on to (2) below. On the left, $n=N$, and so $\varphi_n=\Diamond P_0$ and $\psi_n=\Diamond P_0$. On the right, $n\ne N$, so $\psi_n=\top$.

$$\dfrac{\dfrac{\dfrac{\dfrac{P_0\vDash\neg b}{\langle P_0\rangle\ \mathsf{assume}\ \neg b\ \langle P_0\rangle}\ \textsc{Assume}}{\langle P_0\rangle\ \mathsf{assume}\ \neg b\ \langle\Diamond P_0\rangle}\ \textsc{Consequence}}{\langle\Diamond P_0\rangle\ \mathsf{assume}\ \neg b\ \langle\Diamond P_0\rangle}\ \text{Lemma D.3}}{\langle\varphi_n\rangle\ \mathsf{assume}\ \neg b\ \langle\psi_n\rangle}\qquad \dfrac{\dfrac{}{\langle\varphi_n\rangle\ \mathsf{assume}\ \neg b\ \langle\top\rangle}\ \textsc{True}}{\langle\varphi_n\rangle\ \mathsf{assume}\ \neg b\ \langle\psi_n\rangle}$$

Finally, we complete the derivation using the Iter rule.

$$\dfrac{\forall N\in\mathbb{N}.\quad\dfrac{\dfrac{\dfrac{\dfrac{(1)}{\langle\varphi_n\rangle\ \mathsf{assume}\ b\fatsemi C\ \langle\varphi_{n+1}\rangle}\quad\dfrac{(2)}{\langle\varphi_n\rangle\ \mathsf{assume}\ \neg b\ \langle\psi_n\rangle}}{\langle\varphi_0\rangle\ C^{\langle b,\neg b\rangle}\ \langle\psi_\infty\rangle}\ \textsc{Iter}}{\langle\Diamond P_N\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle\Diamond P_0\rangle}}{\langle P_N\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle\Diamond P_0\rangle}\ \textsc{Consequence}}{\langle\exists n:\mathbb{N}.P_n\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle\Diamond P_0\rangle}\ \textsc{Exists}$$

□

E VARIABLES AND STATE

We now give additional definitions and proofs from Section 6. First, we give the interpretation of expressions $[\![E]\!]_{\mathsf{Exp}}:\mathcal{S}\to\mathsf{Val}$ where $x\in\mathsf{Var}$, $v\in\mathsf{Val}$, and b is a test.

$$\begin{aligned}
[\![x]\!]_{\mathsf{Exp}}(s) &\triangleq s(x)\\
[\![v]\!]_{\mathsf{Exp}}(s) &\triangleq v\\
[\![b]\!]_{\mathsf{Exp}}(s) &\triangleq [\![b]\!]_{\mathsf{Test}}(s)\\
[\![E_1+E_2]\!]_{\mathsf{Exp}}(s) &\triangleq [\![E_1]\!]_{\mathsf{Exp}}(s)+[\![E_2]\!]_{\mathsf{Exp}}(s)\\
[\![E_1-E_2]\!]_{\mathsf{Exp}}(s) &\triangleq [\![E_1]\!]_{\mathsf{Exp}}(s)-[\![E_2]\!]_{\mathsf{Exp}}(s)\\
[\![E_1\times E_2]\!]_{\mathsf{Exp}}(s) &\triangleq [\![E_1]\!]_{\mathsf{Exp}}(s)\cdot[\![E_2]\!]_{\mathsf{Exp}}(s)
\end{aligned}$$

Informally, the free variables of an assertion P are the variables that are used in P. Given that assertions are semantic, we define $\mathsf{free}(P)$ to be those variables that P constrains in some way. Formally, x is free in P iff reassigning x to some value v would not satisfy P.

$$\mathsf{free}(P)\triangleq\{x\in\mathsf{Var}\mid\exists s\in P,v\in\mathsf{Val}.\ s[x\mapsto v]\notin P\}$$

The modified variables of a program C are the variables that are assigned to in the program, determined inductively on the structure of the program.

$$\begin{aligned}
\mathsf{mod}(\mathsf{skip}) &\triangleq\emptyset\\
\mathsf{mod}(C_1\fatsemi C_2) &\triangleq\mathsf{mod}(C_1)\cup\mathsf{mod}(C_2)\\
\mathsf{mod}(C_1+C_2) &\triangleq\mathsf{mod}(C_1)\cup\mathsf{mod}(C_2)\\
\mathsf{mod}(\mathsf{assume}\ e) &\triangleq\emptyset\\
\mathsf{mod}\left(C^{\langle e,e'\rangle}\right) &\triangleq\mathsf{mod}(C)\\
\mathsf{mod}(x\coloneqq E) &\triangleq\{x\}
\end{aligned}$$

Now, before the main soundness and completeness result, we prove a lemma stating that $\langle\Box P\rangle\ C\ \langle\Box P\rangle$ is valid as long as P does not contain information about variables modified by C.

Lemma E.1. *If* $\mathsf{free}(P)\cap\mathsf{mod}(C)=\emptyset$*, then:*

$$\vDash\langle\Box P\rangle\ C\ \langle\Box P\rangle$$

Proof. By induction on the program C:

- ▹ $C=\mathsf{skip}$. Clearly the claim holds using Skip.
- ▹ $C=C_1\fatsemi C_2$. By the induction hypotheses, $\vDash\langle\Box P\rangle\ C_i\ \langle\Box P\rangle$ for $i\in\{1,2\}$. We complete the proof using Seq.
- ▹ $C=C_1+C_2$. By the induction hypotheses, $\vDash\langle\Box P\rangle\ C_i\ \langle\Box P\rangle$ for $i\in\{1,2\}$. We complete the proof using Plus and the fact that $\Box P\oplus\Box P\Leftrightarrow\Box P$.
- ▹ $C=\mathsf{assume}\ e$. Since $\mathsf{assume}\ e$ can only remove states, it is clear that $\Box P$ must still hold after running the program.

- $C = C^{\langle e,e' \rangle}$. The argument is similar to that of the soundness of Invariant. Let $\varphi_n = \psi_n = \psi_\infty = \Box P$. It is obvious that $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$. We also know that $\vDash \langle \Box P \rangle\ C\ \langle \Box P \rangle$ by the induction hypothesis. The rest is a straightforward application of the Iter rule, also using the argument about assume from the previous case.
- $C = x \coloneqq E$. We know that $x \notin \mathsf{free}(P)$, so for all $s \in P$ and $v \in \mathsf{Val}$, we know that $s[x \mapsto v] \in P$. We will now show that $P[E/x] = P$. Suppose $s \in P[E/x]$, this means that $s[x \mapsto [\![E]\!]_{\mathsf{Exp}}(s)] \in P$, which also means that:

$$(s[x \mapsto [\![E]\!]_{\mathsf{Exp}}(s)])[x \mapsto s(x)] = s \in P$$

  Now suppose that $s \in P$, then clearly $s[x \mapsto [\![E]\!]_{\mathsf{Exp}}(s)] \in P$, so $s \in P[E/x]$. Since $P[E/x] = P$, then $(\Box P)[E/x] = \Box P$, so the proof follows from the Assign rule.

□

We now prove the main result. Recall that this result pertains specifically to the OL instance where variable assignment is the only atomic action.

Theorem 6.1 (Soundness and Completeness).

$$\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \iff \qquad \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$$

Proof.

($\Rightarrow$) Suppose $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$. By Theorem 4.6, we already know that this triple is derivable for all commands other than assignment so it suffices to show the case where $C = x \coloneqq E$.
Now suppose $\vDash \langle \varphi \rangle\ x \coloneqq E\ \langle \psi \rangle$. For any $m \in \varphi$, we know that $(\lambda s.\eta(s[x \mapsto [\![E]\!](s)]))^\dagger(m) \in \psi$. By definition, this means that $m \in \psi[E/x]$, so we have shown that $\varphi \Rightarrow \psi[E/x]$. Finally, we complete the derivation as follows:

$$\dfrac{\varphi \Rightarrow \psi[E/x] \qquad \dfrac{}{\langle \psi[E/x] \rangle\ x \coloneqq E\ \langle \psi \rangle}\ \textsc{Assign}}{\langle \varphi \rangle\ x \coloneqq E\ \langle \psi \rangle}\ \textsc{Consequence}$$

($\Leftarrow$) The proof is by induction on the derivation $\vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$. All the cases except for the two below follow from Theorem 4.2.
- Assign. Suppose that $m \vDash \varphi[E/x]$. By the definition of substitution, we immediately know that

$$(\lambda s.\eta(s[x \mapsto [\![E]\!](s)]))^\dagger(m) \in \varphi$$

  Since $[\![x \coloneqq E]\!](s) = \eta(s[x \mapsto [\![E]\!](s)])$, we are done.
- Constancy. Follows immediately from Lemma E.1 and the soundness of the Conj rule.

□

F REUSING PROOF FRAGMENTS

F.1 Integer Division

Recall the definition of the program below that divides two integers.

$$\mathsf{DIV} \triangleq \begin{cases} q \coloneqq 0 \,⨟\, r \coloneqq a \,⨟ \\ \mathsf{while}\ r \geq b\ \mathsf{do} \\ \quad r \coloneqq r - b \,⨟ \\ \quad q \coloneqq q + 1 \end{cases}$$

$$\begin{array}{l}
\langle a \geq 0 \land b > 0 \rangle \\
\quad q \coloneqq 0 \,⨟ \\
\langle a \geq 0 \land b > 0 \land q = 0 \rangle \\
\quad r \coloneqq a \,⨟ \\
\langle a \geq 0 \land b > 0 \land q = 0 \land r = a \rangle \implies \\
\langle q + \lfloor a \div b \rfloor = \lfloor a \div b \rfloor \land r = (a \bmod b) + \lfloor a \div b \rfloor \times b \rangle \implies \\
\langle \varphi_{\lfloor a \div b \rfloor} \rangle \implies \\
\langle \exists n : \mathbb{N}.\varphi_n \rangle \\
\quad \mathsf{while}\ r \geq b\ \mathsf{do} \\
\qquad \langle \varphi_n \rangle \implies \\
\qquad \langle q + n = \lfloor a \div b \rfloor \land r = (a \bmod b) + n \times b \rangle \\
\qquad\quad r \coloneqq r - b \,⨟ \\
\qquad \langle q + n = \lfloor a \div b \rfloor \land r = (a \bmod b) + (n-1) \times b \rangle \\
\qquad\quad q \coloneqq q + 1 \\
\qquad \langle q + (n-1) = \lfloor a \div b \rfloor \land r = (a \bmod b) + (n-1) \times b \rangle \implies \\
\qquad \langle \varphi_{n-1} \rangle \\
\langle \varphi_0 \rangle \implies \\
\langle q + 0 = \lfloor a \div b \rfloor \land r = (a \bmod b) + 0 \times b \rangle \implies \\
\langle q = \lfloor a \div b \rfloor \land r = (a \bmod b) \rangle
\end{array}$$

Figure 5: Derivation for the DIV program.

To analyze this program with the Variant rule, we need a family of variants $(\varphi_n)_{n\in\mathbb{N}}$, defined as follows.

$$\varphi_n \triangleq \begin{cases} q + n = \lfloor a \div b \rfloor \land r = (a \bmod b) + n \times b & \text{if } n \leq \lfloor a \div b \rfloor \\ \mathsf{false} & \text{if } n > \lfloor a \div b \rfloor \end{cases}$$

Additionally, it must be the case that $\varphi_n \vDash r \geq b$ for all $n \geq 1$ and $\varphi_0 \vDash r < b$. For $n > \lfloor a \div b \rfloor$, $\varphi_n = \mathsf{false}$ and $\mathsf{false} \vDash r \geq b$ vacuously. If $1 \leq n \leq \lfloor a \div b \rfloor$, then we know that $r = (a \bmod b) + n \times b$, and since $n \geq 1$, then $r \geq b$. When $n = 0$, we know that $r = a \bmod b$, and so by the definition of mod, it must be that $r < b$.

The derivation is given in Figure 5. Most of the steps are obtained by straightforward applications of the inference rules, with consequences denoted by $\implies$. In the application of the Variant rule, we only show the case where $n \leq \lfloor a \div b \rfloor$. The case where $n > \lfloor a \div b \rfloor$ is vacuous by applying the False rule from Figure 3.

F.2 The Collatz Conjecture

Recall the definition of the program below that finds the stopping time of some positive number n.

$$\mathsf{COLLATZ} \triangleq \begin{cases} i \coloneqq 0 \,⨟ \\ \mathsf{while}\ a \neq 1\ \mathsf{do} \\ \quad b \coloneqq 2 \,⨟\, \mathsf{DIV} \,⨟ \\ \quad \mathsf{if}\ r = 0\ \mathsf{then}\ a \coloneqq q\ \mathsf{else}\ a \coloneqq 3 \times a + 1 \,⨟ \\ \quad i \coloneqq i + 1 \end{cases}$$

The derivation is shown in Figure 6. Since we do not know if the program will terminate, we use the Invariant rule to obtain a partial correctness specification. We choose the loop invariant:

$$a = f^i(n) \land \forall k < i.f^k(n) \neq 1$$

$$
\begin{array}{l}
\langle a = n \land n > 0 \rangle \\
\quad i \coloneqq 0 \,\mathrm{;} \\
\langle a = n \land n > 0 \land i = 0 \rangle \implies \\
\langle a = f^i(n) \land \forall k < i. f^k(n) \neq 1 \rangle \\
\quad \text{while } a \neq 1 \text{ do} \\
\quad\quad \langle a = f^i(n) \land \forall k < i. f^k(n) \neq 1 \land a \neq 1 \rangle \implies \\
\quad\quad \langle a = f^i(n) \land \forall k < i+1. f^k(n) \neq 1 \rangle \\
\quad\quad\quad b \coloneqq 2 \,\mathrm{;} \\
\quad\quad \langle a = f^i(n) \land \forall k < i+1. f^k(n) \neq 1 \land b = 2 \rangle \\
\quad\quad\quad \mathsf{DIV} \,\mathrm{;} \\
\quad\quad \langle a = f^i(n) \land \forall k < i+1. f^k(n) \neq 1 \land b = 2 \land q = \lfloor a \div b \rfloor \land r = (a \bmod b) \rangle \implies \\
\quad\quad \langle a = f^i(n) \land \forall k < i+1. f^k(n) \neq 1 \land q = \lfloor f^i(n) \div 2 \rfloor \land r = (f^i(n) \bmod 2) \rangle \\
\quad\quad\quad \text{if } r = 0 \text{ then} \\
\quad\quad\quad\quad \langle a = f^i(n) \land \forall k < i+1. f^k(n) \neq 1 \land q = \lfloor f^i(n) \div 2 \rfloor \land r = (f^i(n) \bmod 2) \land r = 0 \rangle \implies \\
\quad\quad\quad\quad \langle \forall k < i+1. f^k(n) \neq 1 \land q = \lfloor f^i(n) \div 2 \rfloor \land (f^i(n) \bmod 2) = 0 \rangle \\
\quad\quad\quad\quad\quad a \coloneqq q \\
\quad\quad\quad\quad \langle \forall k < i+1. f^k(n) \neq 1 \land a = \lfloor f^i(n) \div 2 \rfloor \land (f^i(n) \bmod 2) = 0 \rangle \implies \\
\quad\quad\quad\quad \langle \Box(a = f^{i+1}(n) \land \forall k < i+1. f^k(n) \neq 1) \rangle \\
\quad\quad\quad \text{else} \\
\quad\quad\quad\quad \langle a = f^i(n) \land \forall k < i+1. f^k(n) \neq 1 \land q = \lfloor f^i(n) \div 2 \rfloor \land r = (f^i(n) \bmod 2) \land r \neq 0 \rangle \implies \\
\quad\quad\quad\quad \langle a = f^i(n) \land \forall k < i+1. f^k(n) \neq 1 \land (f^i(n) \bmod 2) = 1 \rangle \\
\quad\quad\quad\quad\quad a \coloneqq 3 \times a + 1 \,\mathrm{;} \\
\quad\quad\quad\quad \langle a = 3 \times f^i(n) + 1 \land \forall k < i+1. f^k(n) \neq 1 \land (f^i(n) \bmod 2) = 1 \rangle \implies \\
\quad\quad\quad\quad \langle \Box(a = f^{i+1}(n) \land \forall k < i+1. f^k(n) \neq 1) \rangle \\
\quad\quad \langle \Box(a = f^{i+1}(n) \land \forall k < i+1. f^k(n) \neq 1) \rangle \\
\quad\quad\quad i \coloneqq i + 1 \\
\quad\quad \langle \Box(a = f^i(n) \land \forall k < i. f^k(n) \neq 1) \rangle \\
\langle \Box(a = f^i(n) \land \forall k < i. f^k(n) \neq 1 \land a = 1) \rangle \implies \\
\langle \Box(i = S_n) \rangle
\end{array}
$$

Figure 6: Derivation for the COLLATZ **program.**

So, on each iteration of the loop, a holds the value of applying f repeatedly i times to n, and 1 has not yet appeared in this sequence.

Immediately upon entering the while loop, we see that $a = f^i(n) \neq 1$, and so from that and the fact that $\forall k < i. f^k(n) \neq 1$, we can conclude that $\forall k < i+1. f^k(n) \neq 1$.

The DIV program is analyzed by inserting the proof from Figure 5, along with an application of the rule of Constancy to add information about the other variables. We can omit the $\Box$ modality from rule of Constancy, since $P \land \Box Q \Leftrightarrow P \land Q$.

When it comes time to analyze the if statement, we use the If (Hoare) rule (Lemma D.8) to get a partial correctness specification. The structure of the if statement mirrors the definition of $f(n)$, so the effect is the same as applying f to a one more time, therefore we get that $a = f^{i+1}(n)$.

After exiting the while loop, we know that $f^i(n) = 1$ and $f^k(n) \neq 1$ for all $k < i$, therefore i is (by definition) the stopping time, S_n.

F.3 Embedding Division in a Probabilistic Program

Recall the program that loops an even number of iterations with probability $\frac{2}{3}$ and an odd number of iterations with probability $\frac{1}{3}$.

$$a \coloneqq 0 \,\mathrm{;}\, r \coloneqq 0 \,\mathrm{;}\, (a \coloneqq a + 1 \,\mathrm{;}\, b \coloneqq 2 \,\mathrm{;}\, \mathsf{DIV})^{\langle \frac{1}{2} \rangle}$$

To analyze this program with the Iter rule, we define the two families of assertions below for $n \in \mathbb{N}$.

$$\varphi_n \triangleq (a = n \land r = a \bmod 2)^{\left(\frac{1}{2^n}\right)} \qquad \psi_n \triangleq (a = n \land r = a \bmod 2)^{\left(\frac{1}{2^{n+1}}\right)}$$

Additionally, let $\psi_\infty = (r = 0)_{\frac{2}{3}} \oplus (r = 1)_{\frac{1}{3}}$. We now show that $(\psi_n)_{n \in \mathbb{N}^\infty}$ converges. Suppose that $m_n \vDash \psi_n$ for each $n \in \mathbb{N}$. So $m_n \vDash (r = 0)_{\frac{1}{2^{n+1}}}$ for all even n and $m_n \vDash (r = 1)_{\frac{1}{2^{n+1}}}$ for all odd n. In other words, the cummulative probability mass for each m_n where n is even is:

$$\sum_{k \in \mathbb{N}} \frac{1}{2^{2k+1}} = \frac{1}{2} \cdot \sum_{k \in \mathbb{N}} \left(\frac{1}{4}\right)^k = \frac{1}{2} \cdot \frac{1}{1 - \frac{1}{4}} = \frac{2}{3}$$

$\langle \mathsf{true} \rangle$
$\quad a \coloneqq 0 ⨟$
$\langle a = 0 \rangle$
$\quad r \coloneqq 0 ⨟$
$\langle a = 0 \land r = 0 \rangle \implies$
$\langle (a = 0 \land r = 0 \bmod 2)^{(1)} \rangle \implies$
$\langle \varphi_0 \rangle$

$$\begin{pmatrix} a \coloneqq a + 1 ⨟ \\ b \coloneqq 2 ⨟ \\ \mathsf{DIV} \end{pmatrix}^{\langle \frac{1}{2} \rangle}$$

$\langle \psi_\infty \rangle \implies$
$\langle (r = 0) \oplus_{\frac{2}{3}} (r = 1) \rangle$

(a) Derivation of the main program

$\langle \varphi_n \rangle \implies$
$\langle (a = n)^{(\frac{1}{2^n})} \rangle$
$\quad \mathsf{assume}\ \frac{1}{2} ⨟$
$\langle (a = n)^{(\frac{1}{2^{n+1}})} \rangle$
$\quad a \coloneqq a + 1 ⨟$
$\langle (a = n + 1)^{(\frac{1}{2^{n+1}})} \rangle$
$\quad b \coloneqq 2 ⨟$
$\langle (a = n + 1 \land b = 2)^{(\frac{1}{2^{n+1}})} \rangle$
$\quad \mathsf{DIV}$
$\langle (a = n + 1 \land r = a \bmod 2)^{(\frac{1}{2^{n+1}})} \rangle \implies$
$\langle \varphi_{n+1} \rangle$

(b) Derivation of the probabilistic loop

Figure 7: Derivation for the probabilistic looping program.

Where the second-to-last step is obtained using the standard formula for geometric series. Similarly, the total probability mass for n being odd is:

$$\sum_{k \in \mathbb{N}} \frac{1}{2^{2k+2}} = \frac{1}{4} \cdot \sum_{k \in \mathbb{N}} \left(\frac{1}{4}\right)^k = \frac{1}{4} \cdot \frac{1}{1 - \frac{1}{4}} = \frac{1}{3}$$

We therefore get that $\sum_{n \in \mathbb{N}} m_n \vDash (r = 0)_{\frac{2}{3}} \oplus (r = 1)_{\frac{1}{3}}$. Having shown that, we complete the derivation, shown in Figure 7a. Two proof obligations are generated by applying the Iter rule, the first is proven in Figure 7b. Note that in order to apply our previous proof for the DIV program, it is necessary to use the Scale rule. The second proof obligation of the Iter rule is to show $\langle \varphi_n \rangle\ \mathsf{assume}\ \frac{1}{2}\ \langle \psi_n \rangle$, which is easily dispatched using the Assume rule.

G GRAPH PROBLEMS AND QUANTITATIVE ANALYSIS

G.1 Counting Random Walks

Recall the following program that performs a random walk on a two dimensional grid in order to discover how many paths exist between the origin $(0, 0)$ and the point (N, M).

$$\mathsf{WALK} \triangleq \begin{cases} \mathsf{while}\ x < N \lor y < M\ \mathsf{do} \\ \quad \mathsf{if}\ x < N \land y < M\ \mathsf{then} \\ \quad\quad (x \coloneqq x + 1) + (y \coloneqq y + 1) \\ \quad \mathsf{else\ if}\ x \ge N\ \mathsf{then} \\ \quad\quad y \coloneqq y + 1 \\ \quad \mathsf{else} \\ \quad\quad x \coloneqq x + 1 \end{cases}$$

The derivation is provided in Figure 8. Since this program is guaranteed to terminate after exactly $N + M$ steps, we use the following loop Variant, where the bounds for k are described in Section 8.1.

$$\varphi_n \triangleq \bigoplus_{k=\max(0,n-M)}^{\min(N,n)} (x = N - k \land y = M - (n - k))^{(\binom{N+M-n}{N-k})}$$

Recall that n indicates how many steps (x, y) is from (N, M), so φ_{N+M} is the precondition and φ_0 is the postcondition. Upon entering the while loop, we encounter nested if statements, which we analyze with the If rule. This requires us to split φ_{n+1} into three components, satisfying $n < N \land y < M$, $n \ge N$, and $y \ge M$, respectively. The assertion $x \ge N$ is only possible if we have already taken at least N steps, or in other words, if $n+1 \le (N+M)-N = M$. Letting k range from $\max(0, n + 1 - M)$ to 0 therefore gives us a single term $k = 0$ when $n + 1 \le M$ and an empty conjunction otherwise. A similar argument holds when $y \ge M$. All the other outcomes go into the first branch, where we preclude the $k = 0$ and $k = n + 1$ cases since it must be true that $x \ne N$ and $y \ne M$.

Let $P(n, k) = (x = N - k \land y = M - (n - k))$. Using this shorthand, the postcondition at the end of the if statement is obtained by taking an outcome conjunction of the results from the three branches.

$$\bigoplus_{k=\max(1,n+1-M)}^{\min(N,n)} P(n, k-1)^{(\binom{N+M-(n+1)}{N-k})} \oplus \bigoplus_{k=\max(1,n+1-M)}^{\min(N,n)} P(n, k)^{(\binom{N+M-(n+1)}{N-k})}$$
$$\oplus \bigoplus_{k=\max(0,n+1-M)}^{0} P(n, k)^{(\binom{N+M-(n+1)}{N-k})} \oplus \bigoplus_{k=n+1}^{\min(N,n+1)} P(n, k-1)^{(\binom{N+M-(n+1)}{N-k})}$$

Now, we can combine the conjunctions with like terms.

$$\bigoplus_{k=\max(1,n+1-M)}^{\min(N,n+1)} P(n, k-1)^{(\binom{N+M-(n+1)}{N-k})} \oplus \bigoplus_{k=\max(0,n+1-M)}^{\min(N,n)} P(n, k)^{(\binom{N+M-(n+1)}{N-k})}$$

And adjust the bounds on the first conjunction by subtracting 1 from the lower and upper bounds of k:

$$\bigoplus_{k=\max(0,n-M)}^{\min(N-1,n)} P(n, k)^{(\binom{N+M-(n+1)}{N-(k+1)})} \oplus \bigoplus_{k=\max(0,n+1-M)}^{\min(N,n)} P(n, k)^{(\binom{N+M-(n+1)}{N-k})}$$

Now, we examine when the bounds of these two conjunctions differ. If $n \ge M$, then the first conjunction has an extra $k = n - M$ term. Similarly, the second conjunction has an extra $k = N$ term when $n \ge N$. Based on that observation, we split them as follows:

$$\bigoplus_{k \in \{n-M \mid n \ge M\}} P(n, k)^{(\binom{N+M-(n+1)}{N-(k+1)})} \oplus \bigoplus_{k \in \{N \mid n \ge N\}} P(n, k)^{(\binom{N+M-(n+1)}{N-k})}$$
$$\oplus \bigoplus_{k=\max(0,n+1-M)}^{\min(N-1,n)} P(n, k)^{(\binom{N+M-(n+1)}{N-(k+1)} + \binom{N+M-(n+1)}{N-k})}$$

$\langle x = 0 \land y = 0 \rangle \implies$
$\langle \varphi_{N+M} \rangle \implies$
$\langle \exists n : \mathbb{N}.\ \varphi_n \rangle$
while $x < N \lor y < M$ do
  $\langle \varphi_{n+1} \rangle$
  if $x < N \land y < M$ then
    $\langle \bigoplus_{k=\max(1,n+1-M)}^{\min(N,n)} (x = N - k \land y = M - (n+1-k)^{(\binom{N+M-(n+1)}{N-k})} \rangle$
    $(x \coloneqq x+1) + (y \coloneqq y+1)$
    $\langle \bigoplus_{k=\max(1,n+1-M)}^{\min(N,n)} (x = N - k + 1 \land y = M - (n+1-k)^{(\binom{N+M-(n+1)}{N-k})} \oplus (x = N - k \land y = M - (n-k)^{(\binom{N+M-(n+1)}{N-k})} \rangle$
  else if $x \geq N$ then
    $\langle \bigoplus_{k=\max(0,n+1-M)}^{0} (x = N - k \land y = M - (n+1-k))^{(\binom{N+M-(n+1)}{N-k})} \rangle$
    $y \coloneqq y + 1$
    $\langle \bigoplus_{k=\max(0,n+1-M)}^{0} (x = N - k \land y = M - (n-k))^{(\binom{N+M-(n+1)}{N-k})} \rangle$
  else
    $\langle \bigoplus_{k=n+1}^{\min(N,n+1)} (x = N - k \land y = M - (n+1-k))^{(\binom{N+M-(n+1)}{N-k})} \rangle$
    $x \coloneqq x + 1$
    $\langle \bigoplus_{k=n+1}^{\min(N,n+1)} (x = N - k + 1 \land y = M - (n+1-k))^{(\binom{N+M-(n+1)}{N-k})} \rangle$
  $\langle \varphi_n \rangle$
$\langle (x = N \land y = M)^{(\binom{N+M}{N})} \rangle$

Figure 8: Random walk proof

Knowing that $k = n - M$ in the first conjunction, we get that:

$$\binom{N+M-(n+1)}{N-(k+1)} = \binom{N+M-(n+1)}{N-(n-M+1)} = 1 = \binom{N+M-n}{N-k}$$

Similarly, for the second conjunction we get the same weight. Also, observe that for any a and b:

$$\begin{aligned}\binom{a}{b} + \binom{a}{b+1} &= \frac{a!}{b!(a-b)!} + \frac{a!}{(b+1)!(a-b-1)!} \\ &= \frac{a!}{b!(a-b)(a-b-1)!} + \frac{a!}{(b+1)b!(a-b-1)!} \\ &= \frac{a!(b+1) + a!(a-b)}{(b+1)b!(a-b)(a-b-1)!} \\ &= \frac{a!(b+1+a-b)}{(b+1)!(a-b)!} \\ &= \frac{(a+1)!}{(b+1)!((a+1)-(b+1))!} \\ &= \binom{a+1}{b+1}\end{aligned}$$

So, letting $a = N + M - (n+1)$ and $b = N - (k+1)$, it follows that:

$$\binom{N+M-(n+1)}{N-(k+1)} + \binom{N+M-(n+1)}{N-k} = \binom{N+M-n}{N-k}$$

We can therefore rewrite the assertion as follows:

$$\bigoplus_{k\in\{n-M \mid n\geq M\}} P(n,k)^{(\binom{N+M-n}{N-k})} \oplus \bigoplus_{k\in\{N \mid n\geq N\}} P(n,k)^{(\binom{N+M-n}{N-k})}$$
$$\oplus \bigoplus_{k=\max(0,n+1-M)}^{\min(N-1,n)} P(n,k)^{(\binom{N+M-n}{N-k})}$$

And by recombining the terms, we get:

$$\bigoplus_{k=\max(0,n-M)}^{\min(N,n)} P(n,k)^{(\binom{N+M-n}{N-k})}$$

Which is precisely φ_n. According to the VARIANT rule, the final postcondition is just φ_0.

G.2 Shortest Paths

Recall the following program that nondeterministically finds the shortest path from s to t using a model of computation based on the tropical semiring (Example 2.8).

$$\mathsf{SP} \triangleq \begin{cases} \text{while } pos \neq t \text{ do} \\ \quad next \coloneqq 1 \fatsemi \\ \quad (next \coloneqq next + 1)^{\langle next < N,\ G[pos][next] \rangle} \fatsemi \\ \quad pos \coloneqq next \fatsemi \\ \quad \text{assume } 1 \end{cases}$$

The derivation is shown in Figure 9. We use the While rule to analyze the outer loop. This requires the following families of assertions, where φ_n represents the outcomes where the guard remains true after exactly n iterations and ψ_n represents the outcomes where the loop guard is false after n iterations. Let $I = \{1, \ldots, N\} \setminus \{t\}$.

$$\varphi_n \triangleq \bigoplus_{i \in I} (pos = i)^{(\mathsf{sp}^t_n(G,s,i)+n)}$$

$$\psi_n \triangleq (pos = t)^{(\mathsf{sp}^t_n(G,s,t)+n)} \qquad \psi_\infty \triangleq (pos = t)^{(\mathsf{sp}(G,s,t))}$$

We now argue that $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$. Take any $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \psi_n$ for each n, which means that $|m_n| = \mathsf{sp}^t_n(G,s,t) + n$ and $\mathrm{supp}(m_n) \subseteq (pos = t)$. In the tropical semiring, $|m_n|$ corresponds to the minimum weight of any element in $\mathrm{supp}(m_n)$, so we know there is some $\sigma \in \mathrm{supp}(m_n)$ such that $m_n(\sigma) = \mathsf{sp}^t_n(G,s,t) + n$, and since $\mathsf{sp}^t_n(G,s,t)$ is Boolean valued and true $= 0$ and false $= \infty$, then $m_n(\sigma)$ is either n or ∞.

By definition, the minimum n for which $\mathsf{sp}^t_n(G,s,t) = \text{true}$ is $\mathsf{sp}(G,s,t)$, so for all $n < \mathsf{sp}^t_n(G,s,t)$, it must be the case that $|m_n| = \infty$ and for all $n \geq \mathsf{sp}^t_n(G,s,t)$, it must be the case that $|m_n| = n$. Now, $|\sum_{n\in\mathbb{N}} m_n| = \min_{n\in\mathbb{N}} |m_n| = \mathsf{sp}^t_n(G,s,t)$, and since all elements of each m_n satisfies $pos = t$, then we get that $\sum_{n\in\mathbb{N}} m_n \vDash \psi_\infty$.

Now, we will analyze the inner iteration using the Iter rule and the following two families of assertions, which we will assume are 1-indexed for simplicity of the proof.

$$\vartheta_j \triangleq \begin{cases} \bigoplus_{i\in I} (pos = i \land next = j)^{(\mathsf{sp}^t_n(G,s,i)+n)} & \text{if } j < N \\ \top^{(\mathbb{0})} & \text{if } j \geq N \end{cases}$$

$$\xi_j \triangleq \begin{cases} \bigoplus_{i\in I} (pos = i \land next = j)^{((\mathsf{sp}^t_n(G,s,i) \land G[i][j])+n)} & \text{if } j < N \\ \top^{(\mathbb{0})} & \text{if } j \geq N \end{cases}$$

$$\xi_\infty \triangleq \bigoplus_{j=1}^{N} \bigoplus_{i\in I} (pos = i \land next = j)^{((\mathsf{sp}^t_n(G,s,i) \land G[i][j])+n)}$$

It is easy to see that $(\xi_n)_{n\in\mathbb{N}} \rightsquigarrow \xi_\infty$ since ξ_∞ is by definition an outcome conjunction of all the non-empty terms ξ_j. When $j < N$, then we get $\vartheta_j \vDash next < N$, so we dispatch the first proof obligation of the Iter rule as follows:

$$\begin{array}{l}
\langle \vartheta_j \rangle \implies \\
\langle \bigoplus_{i\in I} (pos = i \land next = j)^{(\mathsf{sp}^t_n(G,s,i)+n)} \rangle \\
\text{assume } next < N \fatsemi \\
\langle \bigoplus_{i\in I} (pos = i \land next = j)^{(\mathsf{sp}^t_n(G,s,i)+n)} \rangle \\
next \coloneqq next + 1 \\
\langle \bigoplus_{i\in I} (pos = i \land next = j+1)^{(\mathsf{sp}^t_n(G,s,i)+n)} \rangle \implies \\
\langle \vartheta_{j+1} \rangle
\end{array}$$

$$\begin{array}{l}
\langle pos = s \rangle \implies \\
\langle \bigoplus_{i=1}^{N} (pos = i)^{\mathsf{sp}^t_0(G,s,i)} \rangle \implies \\
\langle \varphi_0 \oplus \psi_0 \rangle \\
\text{while } pos \neq \text{t do} \\
\quad \langle \varphi_n \rangle \implies \\
\quad \langle \bigoplus_{i\in I} (pos = i)^{(\mathsf{sp}^t_n(G,s,i)+n)} \rangle \\
\quad next \coloneqq 1 \fatsemi \\
\quad \langle \bigoplus_{i\in I} (pos = i \land next = 1)^{(\mathsf{sp}^t_n(G,s,i)+n)} \rangle \\
\quad (next \coloneqq next + 1)^{\langle next < N,\ G[pos][next] \rangle} \fatsemi \\
\quad \langle \bigoplus_{j=1}^{N} \bigoplus_{i\in I} (pos = i \land next = j)^{((\mathsf{sp}^t_n(G,s,i) \land G[i][j])+n)} \rangle \implies \\
\quad \langle \bigoplus_{j=1}^{N} \bigoplus_{i\in I} (next = j)^{((\mathsf{sp}^t_n(G,s,i) \land G[i][j])+n)} \rangle \\
\quad pos \coloneqq next \fatsemi \\
\quad \langle \bigoplus_{j=1}^{N} \bigoplus_{i\in I} (pos = j)^{((\mathsf{sp}^t_n(G,s,i) \land G[i][j])+n)} \rangle \\
\quad \text{assume } 1 \\
\quad \langle \bigoplus_{j=1}^{N} \bigoplus_{i\in I} (pos = j)^{((\mathsf{sp}^t_n(G,s,i) \land G[i][j])+n+1)} \rangle \implies \\
\quad \langle \bigoplus_{j=1}^{N} (pos = j)^{(\mathsf{sp}^t_{n+1}(G,s,j)+n+1)} \rangle \implies \\
\quad \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle \\
\langle \psi_\infty \rangle \implies \\
\langle (pos = t)^{(\mathsf{sp}(G,s,t))} \rangle
\end{array}$$

Figure 9: Shortest path proof

When instead $j \geq N$, then we know that $\vartheta_j \vDash \neg(next < N)$ and so it is easy to see that:

$$\langle \vartheta_j \rangle \text{ assume } next < N \fatsemi next \coloneqq next + 1 \ \langle \top^{(\mathbb{0})} \rangle$$

For the second proof obligation, we must show that:

$$\langle \vartheta_j \rangle \text{ assume } G[pos][next] \ \langle \xi_j \rangle$$

For each outcome, we know that $pos = i$ and $next = j$. If $G[i][j] = \text{true} = 0$, then $(\mathsf{sp}^t_n(G,s,i) \land G[i][j]) + n = \mathsf{sp}^t_n(G,s,i) + n$, so the postcondition is unchanged. If $G[i][j] = \text{false} = \infty$, then $(\mathsf{sp}^t_n(G,s,i) \land G[i][j]) + n = \infty$ and the outcome is eliminated as expected. We now justify the consequence after assume 1. Consider the term:

$$\bigoplus_{i\in I} (pos = j)^{((\mathsf{sp}^t_n(G,s,i) \land G[i][j])+n+1)}$$

This corresponds to just taking the outcome of minimum weight, which will be $n{+}1$ if $\mathsf{sp}_n(G,s,i) \land G[i][j]$ is true for some $i \in I$ and ∞ otherwise. By definition, this corresponds exactly to $\mathsf{sp}^t_{n+1}(G,s,j) + n + 1$.