# Outcome Logic: A Unified Approach to the Metatheory of Program Logics with Branching Effects

NOAM ZILBERSTEIN, Cornell University, USA

Starting with Hoare Logic over 50 years ago, numerous 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 branching at its core, using an algebraic representation of choice to capture programs that branch into many outcomes. In this article, we expand on prior Outcome Logic papers in order to give a more authoritative and comprehensive account of the metatheory. This includes a relatively complete proof system for Outcome Logic with the ability to reason about general purpose looping. We also show that this proof system applies to programs with various types of branching and that it facilitates the reuse of proof fragments across different kinds of specifications.

CCS Concepts: • **Theory of computation** → **Logic and verification**; **Programming logic**; **Hoare logic**.

## 1 Introduction

The seminal work of Floyd [1967a] and Hoare [1969] 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 adversarial behavior 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 [Floyd 1967a; Hoare 1969] vs total correctness [Manna and Pnueli 1974] 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 [Möller et al. 2021; O'Hearn 2020] 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 [Barthe et al. 2018; den Hartog 2002, 1999; Rand and Zdancewic 2015].

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.

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

As we show in this article, 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 proof system for all of them. We also show how specialized reasoning principles (*e.g.*, loop invariants for partial correctness) are

Author's Contact Information: Noam Zilberstein, noamz@cs.cornell.edu, Cornell University, USA.

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 [Le et al. 2022; Möller et al. 2021; O'Hearn 2020; Raad et al. 2020, 2022] 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 [Bruni et al. 2021, 2023; Dardinier and Müller 2024; Maksimović et al. 2023; Zilberstein et al. 2023, 2024]. As a result of that work, the semantic foundations of bug-finding tools have shifted to be more similar to standard Hoare Logic [Ascari et al. 2023; Raad et al. 2024; Zilberstein et al. 2024], and there is further interest in developing consolidated tools [Lööw et al. 2024].

We expand upon one one such effort in the paper: Outcome Logic (OL), which 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* [Zilberstein et al. 2023]. The semantics was later refined such that each trace is weighted using an element of a semiring [Zilberstein et al. 2024]. 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 aforementioned conference papers demonstrated the value of Outcome Logic, but left large gaps. Most notably, no proof strategies were given for unbounded iteration—loops could only be analyzed via bounded unrolling—and the connections to other logics were not deeply explored. In this journal article, we extend the prior conference papers in order to fill those gaps and provide a more authoritative and comprehensive reference on the Outcome Logic metatheory. Rather than focusing on the applications of Outcome Logic to correctness and incorrectness reasoning, we provide a cleaner account of the semiring weights, which supports more models; we provide inference rules for unbounded looping and a relative completeness proof; and we more deeply explore the connections between OL and other logics by showing how the rules of those logics can be derived from the OL proof system. The structure and contributions are follows:

- We give a cleaner Outcome Logic semantics and give six models (Examples 2.7 to 2.12), including a multiset model (Example 2.9) not supported by previous formalizations due to more restrictive algebraic constraints (Sections 2 and 3). Our new looping construct naturally supports deterministic (while loops), nondeterministic (Kleene star), and probabilistic iteration—whereas previous OL versions supported fewer kinds of iteration [Zilberstein et al. 2024] or used a non-unified, ad-hoc semantics [Zilberstein et al. 2023].
- We provide an extensional proof system based on semantic pre- and postconditions and prove that it is sound and relatively complete (Section 3.3). Relative completeness means that the rules are sufficient for deriving any true specification, provided an oracle for deciding consequences between assertions used in pre- and postcondition [Cook 1978]; it is the best case scenario for program logics because those consequences are necessarily undecidable [Apt 1981]. This 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 loop invariant rule (for partial correctness), loop variants (with termination guarantees), and probabilistic loops (Sections 3.5 and 5.1).
- In Section 5, we prove that OL subsumes Hoare Logic and derive the entire Hoare Logic proof system (*e.g.,* loop invariants) in Outcome Logic. Inspired by Dynamic Logic [Harel et al. 2001;

Pratt 1976], our encoding of Hoare Logic uses modalities to generalize partial correctness to types of branching beyond just nondeterminism. We also show that OL subsumes Lisbon Logic (a new logic for incorrectness), and its connections to Hyper Hoare Logic for proving hyperproperties of nondeterministic programs [Dardinier and Müller 2024].

- Through case studies, we demonstrate the reusability of proofs across different effects (*e.g.,* nondeterminism or randomization) and properties (*e.g.,* angelic or demonic nondeterminism) (Section 6). Whereas choices about how to handle loops typically require selecting a specific logic (*e.g.,* partial vs total correctness), loop analysis strategies can be mixed within a single OL derivation; we discuss the implications to program analysis in Section 6.4. We also perform combinatorial analysis of graph algorithms based on alternative computation models (Section 7).
- We contextualize the paper in terms of related work (Section 8) and discuss limitations and opportunities for future development (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.

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

This language is similar to imperative languages such as Dijkstra's [1975] Guarded Command Language (GCL), with familiar constructs such as **skip**, sequential composition ($C_1 \fatsemi C_2$), branching ($C_1 + C_2$), and atomic 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* $u \in U$, to be described fully 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 [Zilberstein et al. 2023, 2024]. In Section 2.4, 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 can be advantageous, as it opens up the possibility for equational reasoning over programs with uninterpreted atomic commands [Kozen 1997; Różowski et al. 2023].

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 $\mathsf{Test} \subseteq \mathbb{2}^\Sigma$ where $\Sigma$ 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 represent the Boolean values false and true, respectively.

The values $\mathbb{0}$ and $\mathbb{1}$ are examples of weights from the set $\{\mathbb{0}, \mathbb{1}\} \subseteq U$. These weights have algebraic properties, which will be described fully in Section 2.1. Using a test $b$, the command **assume** $b$ chooses whether or not to continue evaluating the current branch, 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 a structure such that $\langle U, +, \mathbb{0}\rangle$ is a commutative monoid, $\langle U, \cdot, \mathbb{1}\rangle$ is a monoid, and the following holds:

(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).

Using the addition operator of the semiring, we can define an ordering on semiring elements.

*Definition 2.3 (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.

Based on this order, we also define the notion of Scott continuity, which will be important for defining infinite sums and the semantics of while loops.

*Definition 2.4 (Scott Continuity [Karner 2004]).* 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:

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

Given a Scott continuous semiring, we can also define a notion of infinite sums.

*Definition 2.5 (Infinite Sums).* Let $\langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ be a Scott continuous semiring. Now, for any (possibly infinite) indexed collection $(u_i)_{i \in I}$, we define the following operator:

$$\sum_{i \in I} u_i \triangleq \sup\{u_{i_1} + \cdots + u_{i_n} \mid n \in \mathbb{N}, \{i_1, \ldots, i_n\} \subseteq_{\text{fin}} I\}$$

That is, an infinite sum is the supremum of the sums of all finite subsequences. As shown by Kuich [2011, Corollary 1], this operator makes $U$ a complete semiring, meaning it upholds the following properties:

(1) If $I = \{i_1, \ldots, i_n\}$ is finite, then $\sum_{i \in I} u_i = u_{i_1} + \cdots + u_{i_n}$
(2) If $\sum_{i \in I} u_i$ 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 $\sum_{k \in K} \sum_{j \in J_k} u_j = \sum_{i \in I} u_i$

## 2.2 Weighting Functions

Semirings elements will act as the *weights* for traces in our semantics. 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 yield different kinds of effects. For example, a Boolean semiring where $U = \{0, 1\}$ corresponds to nondeterminism; $[\![C]\!](\sigma)\colon \Sigma \to \{0, 1\} \cong 2^\Sigma$ tells us which states are in the set of 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 2.6 (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 $\mathrm{supp}(m) \triangleq \{\sigma \mid m(\sigma) \neq \mathbb{0}\}$ and $|m| \triangleq \sum_{\sigma \in \mathrm{supp}(m)} m(\sigma)$.

Weighting functions can encode the following types of computation.

*Example 2.7 (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 2.8 (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 2.6 that $|m|$ is defined means that $|\mathrm{supp}(m)| \leq 1$, so we get that $\mathcal{W}_{\mathsf{Bool}'}(X) \cong X \sqcup \{\lightning\}$—it is either a single value $x \in X$, or $\lightning$, indicating that the program diverged.

*Example 2.9 (Multiset Nondeterminism).* Rather than indicating which outcomes are possible using Booleans, we use natural numbers (extended with $\infty$) $n \in \mathbb{N}^\infty$ to count the traces leading to each outcome. This yields the semiring $\mathsf{Nat} = \langle \mathbb{N}^\infty, +, \cdot, 0, 1\rangle$ with the standard arithmetic operations, and we get that $\mathcal{W}_{\mathsf{Nat}}(X) \cong \mathcal{M}(X)$ where $\mathcal{M}(X)$ is the set of *multisets* over $X$.

*Example 2.10 (Randomization).* Probabilities $p \in [0, 1] \subset \mathbb{R}$ form a partial semiring $\mathsf{Prob} = \langle [0, 1], +, \cdot, 0, 1\rangle$ where $+$ and $\cdot$ are real-valued arithmetic operations, but $+$ is undefined if $x + y > 1$ (just like in Example 2.8). This gives us $\mathcal{W}_{\mathsf{Prob}}(X) \cong \mathcal{D}(X)$, where $\mathcal{D}(X)$ is the set of discrete probability sub-distributions over $X$ (the mass can be less than 1 if some traces diverge).

*Example 2.11 (Tropical Computation).* $\mathsf{Tropical} = \langle [0, \infty], \min, +, \infty, 0\rangle$ uses real-valued weights, but semiring addition is minimum and semiring multiplication is arithmetic addition. Computations in $\mathcal{W}_{\mathsf{Tropical}}(X)$ correspond to programs that choose the *cheapest* path for each outcome.

*Example 2.12 (Formal Languages).* For some alphabet $\Gamma$, let $\Gamma^*$ be the set of finite strings over that alphabet, $\Gamma^\omega$ is the set of infinite strings, and $\Gamma^\infty = \Gamma^* \cup \Gamma^\omega$ is the set of all strings. Then $\mathsf{Lang} = \langle 2^{\Gamma^\infty}, \cup, \cdot, \emptyset, \{\varepsilon\}\rangle$ is the semiring of formal languages (sets of strings) where addition is given by the standard union, multiplication is given by concatenation:

$$\ell_1 \cdot \ell_2 \triangleq \{st \mid s \in \ell_1, t \in \ell_2\}$$

The unit of addition is the empty language and the unit of multiplication is the language containing only the empty string $\varepsilon$. Note that multiplication (*i.e.,* concatenation) is not commutative. Computations in $\mathcal{W}_{\mathsf{Lang}}(\Sigma)$ correspond to programs that log sequences of letters [Batz et al. 2022].

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

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

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

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 [Manes 1976; Moggi 1991].

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

Fig. 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 $\Sigma$, atomic actions Act, primitive tests Test, and an interpretation of atomic actions $[\![a]\!]_{\mathsf{Act}} : \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma)$.

*Definition 2.13 (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 for any sets $X$ and $Y$, $(-)^\dagger\colon (X \to T(Y)) \to T(X) \to T(Y)$ 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 $\eta$ and $(-)^\dagger$ are defined below.

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

### 2.3 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, Scott continuous, partial semiring with a top element $\top \in U$ such that $\top \geq u$ for all $u \in U$.
(2) $\Sigma$ 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 interpretation of atomic actions.

This definition is a generalized version of the one used in Outcome Separation Logic [Zilberstein et al. 2024]. 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 2.9).

Commands are interpreted denotationally 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 (Definition 2.13) and semiring operations (Definitions 2.2 and 2.6): **skip** uses $\eta$, 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 2.4, we discuss simple syntactic checks to ensure that the semantics is total. Atomic actions are interpreted using $[\![\cdot]\!]_{\mathsf{Act}}$.

The interpretation of **assume** relies on the ability to interpret expressions and tests. We first describe the interpretation of tests, which maps $b$ 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 is $\mathbb{0}$, so the branch is eliminated. If it evaluates to true, then it 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 start by defining it recursively as follows.

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

To ensure that this equation has a well-defined solution, we formulate the semantics as a least fixed point. Requiring that the semiring is Scott continuous (Definition 2.4), ensures that this fixed point exists. For the full details, see Appendix A.

### 2.4 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 instance. However, regardless of which semiring is used, guarded choice (*i.e.,* if statements) are always valid, which we define as syntactic sugar.

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

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:

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

We also provide syntactic sugar for iterating constructs. While loops use a test to determine whether iteration should continue, making them deterministic.

$$\textbf{while}\ b\ \textbf{do}\ C \triangleq C^{\langle b,\, \neg b\rangle}$$

The Kleene star $C^\star$ is defined for interpretations based on total semirings only; it iterates $C$ nondeterministically many times.[1]

$$C^\star \triangleq C^{\langle \mathbb{1},\, \mathbb{1}\rangle}$$

Finally, the probabilistic iterator $C^{\langle p\rangle}$ continues to execute with probability $p$ and exits with probability $1 - p$.

$$C^{\langle p\rangle} \triangleq C^{\langle p,\, 1-p\rangle}$$

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 [Różowski et al. 2023]. 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.

[1] In nondeterministic languages, **while** $b$ **do** $C \equiv (\textbf{assume}\ b \fatsemi C)^\star \fatsemi \textbf{assume}\ \neg b$, however this encoding does not work in general since $(\textbf{assume}\ b \fatsemi C)^\star$ is not a well-defined program when using a partial semiring (*e.g.,* Examples 2.8 and 2.10).

## 3 Outcome Logic

In this section, we define Outcome Logic, provide extensional definitions for assertions used in the pre- and postconditions of Outcome Triples, and present a relatively complete proof system.

### 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 describe *individual states*—outcome assertions expose the 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 elements corresponding to their true assignments. For any $m \in \mathcal{W}_{\mathcal{A}}(\Sigma)$, we write $m \vDash \varphi$ ($m$ satisfies $\varphi$) to mean that $m \in \varphi$.

The use of semantic assertions makes our approach extensional. We will therefore show that the Outcome Logic proof system is sufficient for analyzing programs structurally, but it cannot be used to decide entailments between the pre- and postconditions themselves. No program logic is truly complete, as analyzing loops inevitably reduces to the (undecidable) halting problem [Apt 1981; Cook 1978]—it is well known that the ability to express intermediate assertions and loop invariants means that the assertion language must at least contain Peano arithmetic [Lipton 1977]. As a result, many modern developments such as Separation Logic [Calcagno et al. 2007; Yang 2001], Incorrectness Logic [O'Hearn 2020], Iris [Jung et al. 2018, 2015], probabilistic Hoare-style logics [Barthe et al. 2018; Kaminski 2019], and others [Ascari et al. 2023; Cousot et al. 2012; Dardinier and Müller 2024; Raad et al. 2024] use semantic assertions.

We now define useful notation for assertions, which is 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 complement.

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

Conjunction, disjunction, 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 notation for 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 $\varphi$ holds in $m_1$ and $\psi$ holds in $m_2$. 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 $\varphi$ and $\psi$ each describe subsets of the reachable states. We define outcome conjunctions formally for a predicate $\phi \colon T \to 2^{\mathcal{W}_{\mathcal{A}}(\Sigma)}$ over some (possibly infinite) set $T$.

$$\bigoplus_{x \in T} \phi(x) \triangleq \left\{ \sum_{t \in T} m_t \;\middle|\; \forall t \in T.\ m_t \in \phi(t) \right\}$$

That is, $m \in \bigoplus_{x \in T} \phi(x)$ if $m$ is the sum of a sequence of $m_t$ elements for each $t \in T$ such that each $m_t \in \phi(t)$. The binary $\oplus$ operator can be defined in terms of the one defined above.

$$\varphi \oplus \psi \triangleq \bigoplus_{i \in \{1,2\}} \phi(i) \qquad \text{where} \qquad \phi(1) = \varphi \qquad \text{and} \qquad \phi(2) = \psi$$

$$\top \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma) \qquad \bot \triangleq \emptyset \qquad \neg\varphi \triangleq \top \setminus \varphi$$
$$\varphi \wedge \psi \triangleq \varphi \cap \psi \qquad \varphi \vee \psi \triangleq \varphi \cup \psi \qquad \varphi \Rightarrow \psi \triangleq \neg\varphi \cup \psi$$
$$\exists x : T.\phi(x) \triangleq \bigcup_{t \in T} \phi(t) \qquad \varphi \odot u \triangleq \{m \cdot u \mid m \in \varphi\} \qquad u \odot \varphi \triangleq \{u \cdot m \mid m \in \varphi\}$$

$$\bigoplus_{x \in T} \phi(x) \triangleq \left\{ \sum_{t \in T} m_t \;\middle|\; \forall t \in T.\ m_t \in \phi(t) \right\}$$
$$\lceil P \rceil^{(u)} \triangleq \left\{ m \in \mathcal{W}_{\mathcal{A}}(\Sigma) \;\middle|\; |m| = u, \mathrm{supp}(m) \subseteq P \right\}$$

Fig. 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 $P \in 2^{\Sigma}$.

The weighting operations $\varphi \odot u$ and $u \odot \varphi$, inspired by Batz et al. [2022], scale the outcome $\varphi$ by a literal weight $u \in U$. Note that there are two variants of this assertion for scaling on the left and right, since multiplication is not necessarily commutative (see Example 2.12).

$$\varphi \odot u \triangleq \{m \cdot u \mid m \in \varphi\} \qquad u \odot \varphi \triangleq \{u \cdot m \mid m \in \varphi\}$$

Finally, given a semantic assertion on states $P \subseteq \Sigma$, we can lift $P$ to be an outcome assertion, $\lceil P \rceil^{(u)}$, meaning that $P$ covers all the reachable states ($\mathrm{supp}(m) \subseteq P$) and the cumulative weight is $u$.

$$\lceil P \rceil^{(u)} \triangleq \left\{ m \in \mathcal{W}_{\mathcal{A}}(\Sigma) \;\middle|\; |m| = u, \mathrm{supp}(m) \subseteq P \right\}$$

When the weight is $\mathbb{1}$, we will write $\lceil P \rceil$ instead of $\lceil P \rceil^{(\mathbb{1})}$. These assertions can interact with weighting assertions, for instance $u \odot \lceil P \rceil^{(v)} \Rightarrow \lceil P \rceil^{(u \cdot v)}$ and $\lceil P \rceil^{(v)} \odot u \Rightarrow \lceil P \rceil^{(v \cdot u)}$. We also permit the use of tests $b$ as assertions, for instance:

$$\lceil P \wedge b \rceil = \left\{ m \in \mathcal{W}(\Sigma) \;\middle|\; |m| = \mathbb{1}, \forall \sigma \in \mathrm{supp}(m).\ \sigma \in P \wedge [\![b]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \right\}$$

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 $\lceil P \rceil \oplus \lceil Q \rceil$ 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 $\lceil P \rceil \vee \lceil Q \rceil$ or $\lceil P \vee Q \rceil$, since the disjunction cannot not guarantee that *both* $P$ and $Q$ are reachable. Suppose $P$ describes a desirable program outcome whereas $Q$ describes an erroneous one; then $\lceil P \rceil \oplus \lceil Q \rceil$ tells us that the program has a bug (it can reach an error state) whereas neither $\lceil P \rceil \vee \lceil Q \rceil$ nor $\lceil P \vee Q \rceil$ is not strong enough to make this determination [Zilberstein et al. 2023].

Similar to the syntactic sugar for probabilistic programs in Section 2.4, we define:

$$\varphi \oplus_p \psi \triangleq (p \odot \varphi) \oplus ((1 - p) \odot \psi)$$

If $\varphi$ and $\psi$ are the results of running $C_1$ and $C_2$, then $\varphi \oplus_p \psi$—meaning that $\varphi$ occurs with probability $p$ and $\psi$ 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 [Zilberstein et al. 2023]. 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.

$$\frac{}{\langle \varphi \rangle\ \mathbf{skip}\ \langle \varphi \rangle}\textsc{Skip} \qquad \frac{\langle \varphi \rangle\ C_1\ \langle \vartheta \rangle \qquad \langle \vartheta \rangle\ C_2\ \langle \psi \rangle}{\langle \varphi \rangle\ C_1 \fatsemi C_2\ \langle \psi \rangle}\textsc{Seq}$$

$$\frac{\langle \varphi \rangle\ C_1\ \langle \psi_1 \rangle \qquad \langle \varphi \rangle\ C_2\ \langle \psi_2 \rangle}{\langle \varphi \rangle\ C_1 + C_2\ \langle \psi_1 \oplus \psi_2 \rangle}\textsc{Plus} \qquad \frac{\varphi \vDash e = u}{\langle \varphi \rangle\ \mathbf{assume}\ e\ \langle \varphi \odot u \rangle}\textsc{Assume}$$

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

Fig. 3. Inference rules for program commands

*Definition 3.1 (Outcome Triples).* Given $\langle \mathcal{A}, \Sigma, \mathrm{Act}, \mathrm{Test}, [\![\cdot]\!]_{\mathrm{Act}} \rangle$, 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 running the program $C$ on a weighted collection of states satisfying $\varphi$ results in a collection satisfying $\psi$. Using outcome assertions to describe these collections of states in the pre- and postconditions means that Outcome Logic can express many types of properties including reachability ($\lceil P \rceil \oplus \lceil Q \rceil$), probabilities ($\varphi \oplus_p \psi$), and nontermination (the lack of outcomes, $\lceil \mathrm{true} \rceil^{(\mathbb{0})}$). Next, we will see how Outcome Logic can be used to encode familiar program logics.

### 3.3 Inference Rules

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

*Sequential Commands.* The rules for sequential (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 sub-derivations 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:

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

If $e$ is a weight literal $u$, then $\varphi \vDash u = u$ vacuously holds for any $\varphi$, so the rule can be simplified to $\vdash \langle \varphi \rangle\ \mathbf{assume}\ u\ \langle \varphi \odot u \rangle$. But if it is a test $b$, then $\varphi$ must contain enough information to conclude that $b$ is true or false. Additional rules to decide $\varphi \vDash e = u$ are given in Appendix B.

*Iteration.* The Iter rule uses two families of predicates: $\varphi_n$ represents the result of $n$ iterations of $\mathbf{assume}\ e \fatsemi C$ and $\psi_n$ 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 aggregated terminating traces. This is captured by the assertion $\psi_\infty$, which must have the following property.

*Definition 3.2 (Converging Assertions).* A family $(\psi_n)_{n\in\mathbb{N}}$ *converges* to $\psi_\infty$ (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 rules that are not dependent on the program command in Figure 4. This includes rules for trivial preconditions (False) and postconditions (True). The Scale rule states that we may multiply the pre- and postconditions by a weight to obtain a new valid triple.

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

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

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

$$\frac{\varphi' \Rrightarrow \varphi \qquad \langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \psi \Rrightarrow \psi'}{\langle \varphi' \rangle\ C\ \langle \psi' \rangle}\textsc{Consequence}$$

Fig. 4. Structural rules.

Subderivations can also be combined using logical connectives as is done in the Disj, Conj, and Choice) rules. Existential quantifiers are introduced using Exists. Finally, the rule of Consequence can be used to strengthen preconditions and weaken postconditions in the style of Hoare Logic. These implications are semantic ones: $\varphi' \Rrightarrow \varphi$ iff $\varphi' \subseteq \varphi$. We do not explore the proof theory for outcome assertions, although prior work in this area exists as outcome conjunctions are similar to the separating conjunction from the logic of bunched implictions [Docherty 2019; O'Hearn and Pym 1999].

### 3.4 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 according to Definition 3.1. We write $\Gamma \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$ to mean that the triple $\langle \varphi \rangle\ C\ \langle \psi \rangle$ is derivable given a collection of axioms $\langle \varphi \rangle\ a\ \langle \psi \rangle \in \Gamma$. Let $\Omega$ consist of all triples $\langle \varphi \rangle\ a\ \langle \psi \rangle$ such that $a \in \mathrm{Act}$, 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 2.4. The soundness theorem is stated formally below.

Theorem 3.3 (Soundness).

$$\Omega \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \Longrightarrow \qquad \vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$$

The full proof is shown in Appendix B 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 $C^0 = \mathbf{skip}$ and $C^{n+1} = C^n \fatsemi C$.

Lemma 3.4. *The following equation holds:*

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

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 $\varphi \Rrightarrow \psi$ 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 [Apt 1981; Cook 1978; Lipton 1977].

The first step is to show that given any program $C$ and precondition $\varphi$, we can derive the triple $\langle \varphi \rangle\ C\ \langle \psi \rangle$, where $\psi$ is the *strongest postcondition* [Dijkstra and Schölten 1990], *i.e.,* the strongest assertion making that triple true. As defined below, $\psi$ 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 3.5 (Strongest Postcondition).*

$$\mathrm{post}(C, \varphi) \triangleq \{[\![C]\!]^{\dagger}(m) \mid m \in \varphi\}$$

Lemma 3.6.

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

The proof is by induction on the structure of the program, and is shown in its entirety in Appendix B. The cases for **skip** and $C_1 \fatsemi C_2$ 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 4).

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

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

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.

$$\exists n : \mathbb{N}.\ \lceil x = n \rceil \oplus \lceil x = n + 1 \rceil$$

Unfortunately, that is also incorrect; it does not account for the fact that that precondition $\lceil x \geq 0 \rceil$ 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 [Barthe et al. 2018; den Hartog 2002] and hyper-property [Dardinier and Müller 2024] reasoning, both of which are encodable in 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 3.5) and used (Sections 6 and 7).

The main relative completeness result is now a straightforward corollary of Lemma 3.6 using the rule of Consequence, since any valid postcondition is implied by the strongest one.

Theorem 3.7 (Relative Completeness).

$$\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \Longrightarrow \qquad \Omega \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$$

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 $m = [\![C]\!]^{\dagger}(m')$. Using $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$, we get that $m \vDash \psi$. Now,

we complete the derivation as follows:

$$\dfrac{\dfrac{\Omega}{\langle\varphi\rangle\ C\ \langle \mathsf{post}(C,\varphi)\rangle}\ \textsc{Lemma 3.6} \qquad \mathsf{post}(C,\varphi) \Rrightarrow \psi}{\langle\varphi\rangle\ C\ \langle\psi\rangle}\ \textsc{Consequence}$$

□

### 3.5 Derived Rules for Syntactic Sugar

Recall from Section 2.4 that if statements and while loops are encoded using the choice and iteration constructs. We now derive convenient inference rules for if and while. The full derivations of these rules are shown in Appendix E.

If statements are defined as $(\mathbf{assume}\ b \fatsemi C_1) + (\mathbf{assume}\ \neg b \fatsemi C_2)$. Reasoning about them generally requires the precondition to be separated into two parts, $\varphi_1$ and $\varphi_2$, representing the collections of states in which $b$ is true and false, respectively. This may require—*e.g.*, in the probabilistic case—that $\varphi_1$ and $\varphi_2$ quantify the weight (likelihood) of the guard.

If it is possible to separate the precondition in that way, then $\varphi_1$ and $\varphi_2$ act as the preconditions for $C_1$ and $C_2$, respectively, and the overall postcondition is an outcome conjunction of the results of the two branches.

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

From If, we can also derive one-sided rules, which apply when one of the branches is certainly taken.

$$\dfrac{\varphi \vDash b \qquad \langle\varphi\rangle\ C_1\ \langle\psi\rangle}{\langle\varphi\rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle\psi\rangle}\ \textsc{If1} \qquad\qquad \dfrac{\varphi \vDash \neg b \qquad \langle\varphi\rangle\ C_2\ \langle\psi\rangle}{\langle\varphi\rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle\psi\rangle}\ \textsc{If2}$$

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 $\varphi_n$ now represents the portion of the program configuration where the guard $b$ is true, and $\psi_n$ represents the portion where it is false. So, on each iteration, $\varphi_n$ continues to evaluate and $\psi_n$ exits; the final postcondition $\psi_\infty$ is an aggregation of all the terminating traces.

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

This While rule is similar to those found in probabilistic Hoare Logics [Barthe et al. 2018; den Hartog 1999] and Hyper Hoare Logic [Dardinier and Müller 2024].

Loop variants are an alternative way to reason about loops that terminate in a finite number of steps. They were first studied in the context of total Hoare Logic [Manna and Pnueli 1974], but are also used in other logics that require termination guarantees such as Reverse Hoare Logic [de Vries and Koutavas 2011], Incorrectness Logic [O'Hearn 2020], and Lisbon Logic [Ascari et al. 2023; Möller et al. 2021; Raad et al. 2024].[2]

The rule uses a family of *variants* $(\varphi_n)_{n\in\mathbb{N}}$ such that $\varphi_n$ implies that the loop guard $b$ is true for all $n > 0$, and $\varphi_0$ implies that it is false, guaranteeing that the loop exits. The inference rule is

[2]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, $[\![\mathbf{skip}]\!]\,(\sigma) = [\![\mathbf{skip} + \mathbf{while}\ \mathsf{true}\ \mathbf{do}\ \mathbf{skip}]\!]\,(\sigma)$. The exception is the probabilistic interpretation, where *almost sure termination* can be established by proving that probability of terminating is 1. See Section 9 for a more in depth discussion.

shown below, and states that starting at some $\varphi_n$, the execution will eventually count down to $\varphi_0$, at which point it terminates.

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

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

## 4 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 $x := E$, where $x \in \mathsf{Var}$ and $E \in \mathsf{Exp}$ can be a variable $x \in \mathsf{Var}$, constant $v \in \mathsf{Val}$, test $b$, or an arithmetic operation ($+, -, \times$).

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

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 C.

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

We define substitutions in the standard way [Ascari et al. 2023; Barthe et al. 2018; Dardinier and Müller 2024; Kaminski 2019], 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 $\varphi$. This behaves as expected in conjunction with symbolic tests, for example $\lceil x \geq 5 \rceil [y + 1/x] = \lceil y + 1 \geq 5 \rceil = \lceil y \geq 4 \rceil$. 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 [1969] Logic.

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

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 C. In addition, the $\Box P$ modality means that $P$ holds over the entire support of the weighting function, but does not specify the total weight. It is defined $\Box P \triangleq \exists u : U.\ \lceil P \rceil^{(u)}$, and is discussed further in Section 5.1. We use $\Box P$ to guarantee that the rule of Constancy applies regardless of whether or not $C$ terminates, branches, or alters the weights of traces.

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

In the Outcome Logic instance with variable assignment as the only atomic action, all triples can be derived without the axioms Ω from Theorem 3.7.

Theorem 4.1 (Soundness and Completeness).

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

## 5 Connections to Other Logics

Outcome Logic, in its full generality, allows one to quantify the precise weights of each outcome, providing significant expressive power. Nevertheless, many common program logics do not provide this much power, which can be advantageous as they offer simplified reasoning principles—for example, Hoare Logic's loop Invariant rule is considerably simpler than the While rule needed for general Outcome Logic (Section 3.5). In this section, we show the connections between Outcome Logic and several other logics by first showing that OL can capture the semantics of specifications in those logics, and then also deriving the proof rules of those logics using the OL proof system.

### 5.1 Dynamic Logic, Hoare Logic, and Lisbon Logic

We will now devise an assertion syntax to show the connections between Outcome Logic and Hoare Logic. We take inspiration from modal logic and Dynamic Logic [Harel et al. 2001; Pratt 1976], 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.

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

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 $- \oplus \top$ permits additional elements to appear in the support. So, $m \vDash \Diamond P$ when $\sigma \in P$ 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 $\Diamond P \Leftrightarrow \neg\Box\neg P$. Defining these constructs as syntactic sugar will allow us to reason about them with standard principles, rather than specialized inference rules. For Boolean-valued semirings (Examples 2.7 and 2.8), we get the following:

$$\Box P \quad = \quad \exists u : \{0,1\}.\ \lceil P \rceil^{(u)} \quad = \quad \lceil P \rceil^{(0)} \vee \lceil P \rceil^{(1)}$$

Only $\mathbb{0}$, the empty collection, satisfies $\lceil P \rceil^{(0)}$, indicating that there are no outcomes and therefore the program diverged (let us call this assertion div), and $\lceil P \rceil^{(1)}$ is equivalent to $\lceil P \rceil$. 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, in Boolean semirings, we have:

$$\Diamond P \quad = \quad \exists u : (\{0,1\} \setminus \{0\}).\ \lceil P \rceil^{(u)} \oplus \top \quad = \quad \lceil P \rceil^{(1)} \oplus \top \quad = \quad \lceil P \rceil \oplus \top$$

So, $\Diamond P = \lceil P \rceil \oplus \top$, which means that $P$ is one of the possibly many outcomes.

Now, we are going to use these modalities to encode other program logics in Outcome Logic. 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 [Harel et al. 2001; Pratt 1976], 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, $[C]Q$ asserts 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 [Dijkstra 1975, 1976]. The

dual modality $\langle C \rangle Q$ states that $Q$ might hold after running $C$ (also called the weakest possible precondition [Hoare 1978; Möller et al. 2021]). 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.

Theorem 5.1 (Subsumption of Hoare Logic).

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

While Zilberstein et al. [2023] previously showed that Outcome Logic subsumes Hoare Logic, our characterization is not tied to nondeterminism; the triple $\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle$ 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. We will shortly develop rules for reasoning about loops using invariants, which 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, which was proposed during a meeting in Lisbon as a possible foundation for incorrectness reasoning [Möller et al. 2021; O'Hearn 2020; Zilberstein et al. 2023]. 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 in some of the traces.

Theorem 5.2 (Subsumption of Lisbon Logic).

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

We now present derived rules for simplified reasoning in our embeddings of Hoare and Lisbon Logic within Outcome Logic. For the full derivations, refer to Appendix E.

*Sequencing.* 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.

$$\frac{\langle \lceil P \rceil \rangle\ C_1\ \langle \Box Q \rangle \quad \langle \lceil Q \rceil \rangle\ C_2\ \langle \Box R \rangle}{\langle \lceil P \rceil \rangle\ C_1 \fatsemi C_2\ \langle \Box R \rangle}\ \textsc{Seq (Hoare)} \qquad \frac{\langle \lceil P \rceil \rangle\ C_1\ \langle \Diamond Q \rangle \quad \langle \lceil Q \rceil \rangle\ C_2\ \langle \Diamond R \rangle}{\langle \lceil P \rceil \rangle\ C_1 \fatsemi C_2\ \langle \Diamond R \rangle}\ \textsc{Seq (Lisbon)}$$

These rules rely on the fact that $\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle \vdash \langle \Box P \rangle\ C\ \langle \Box Q \rangle$ and $\langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle \vdash \langle \Diamond P \rangle\ C\ \langle \Diamond Q \rangle$, which is true since for any $m \in \Box P$, it must be that each $\sigma \in \mathrm{supp}(m)$ satisfies $P$, meaning that $[\![C]\!]\,(\sigma) \in \Box Q$ and so $[\![C]\!]^\dagger(m)$ must also satisfy $\Box Q$. The $\Diamond$ case is similar, also making use of the fact that $(\lceil R \rceil \oplus \top) \oplus \top \Leftrightarrow \lceil R \rceil \oplus \top$. 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 2.7 to 2.12 all obey this property.

Note that since $\lceil P \rceil \Rightarrow \Box P$ and $\lceil P \rceil \Rightarrow \Diamond P$, the rule of Consequence also gives us $\langle \Box P \rangle\ C\ \langle \Box Q \rangle \vdash \langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle$ and $\langle \Diamond P \rangle\ C\ \langle \Diamond Q \rangle \vdash \langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle$, so we could have equivalently defined Hoare and Lisbon Logic as $\vDash \langle \Box P \rangle\ C\ \langle \Box Q \rangle$ and $\vDash \langle \Diamond P \rangle\ C\ \langle \Diamond Q \rangle$, respectively. We prefer the asymmetric use of modalities, as it allows for specifications of the form $\langle \lceil P \rceil \rangle\ C\ \langle \lceil Q \rceil \rangle$, which can be easily weakened for use in both Hoare and Lisbon Logic, as we will see in Section 6.4.

*If Statements and While Loops.* The familiar rule for if statements in Hoare Logic is also derivable, and does not require any semantic entailments, instead using the fact that $\lceil P \rceil \Rrightarrow \Box(P \wedge b) \oplus \Box(P \wedge \neg b)$.

$$\frac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \Box Q \rangle \qquad \langle \lceil P \wedge \neg b \rceil \rangle\ C_2\ \langle \Box Q \rangle}{\langle \lceil P \rceil \rangle\ \textbf{if}\ b\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \langle \Box Q \rangle}\ \textsc{If (Hoare)}$$

A similar rule is derivable for Lisbon Logic, although the derivation is a bit more complex due to the reachability guarantees provided by Lisbon Logic, and the fact that $\lceil P \rceil \Rightarrow \Diamond(P \wedge b) \oplus \Diamond(P \wedge \neg b)$. Instead, the derivation involves case analysis on whether $\Diamond(P \wedge b)$ or $\Diamond(P \wedge \neg b)$ is true.

$$\frac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \Diamond Q \rangle \qquad \langle \lceil P \wedge \neg b \rceil \rangle\ C_2\ \langle \Diamond Q \rangle}{\langle \lceil P \rceil \rangle\ \textbf{if}\ b\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{If (Lisbon)}$$

*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 5.1. The rule for Outcome Logic loop invariants is as follows:

$$\frac{\langle \lceil P \wedge b \rceil \rangle\ C\ \langle \Box P \rangle}{\langle \lceil P \rceil \rangle\ \textbf{while}\ b\ \textbf{do}\ C\ \langle \Box(P \wedge \neg b) \rangle}\ \textsc{Invariant}$$

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 $P \wedge \neg b$ is true in every terminating state. If the program diverges and there are no reachable end states, then $\Box(P \wedge \neg b)$ is vacuously satisfied, just like in Hoare Logic.

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

It is well known that Skip, Seq (Hoare), If (Hoare), Invariant, Assign, and Consequence constitute a relatively complete proof system for Hoare Logic [Cook 1978; Kozen and Tiuryn 2001]. It follows that these rules are complete for deriving any Outcome Logic triples of the form $\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle$, avoiding the more complex machinery of Lemma 3.6[3].

*Loop Variants.* 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.

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

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.

[3] *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.

### 5.2 Hyper Hoare Logic

Hyper Hoare Logic (HHL) is a generalized version of Hoare Logic that uses predicates over *sets of states* as pre- and postconditions to enable reasoning about *hyperproperties*—properties of multiple executions of a single program [Dardinier and Müller 2024]. Hyperproperties are useful for expressing information flow security properties of programs, among others. In this section, we show how hyperproperty reasoning inspired by HHL can be done in Outcome Logic.

Although HHL triples have equivalent semantics to the powerset instance of Outcome Logic (Example 2.7), the spirit of those triples are quite different.[4] In Outcome Logic the precondition often describes a single state, whereas the postcondition specifies the nondeterministic outcomes that stem from that state. By contrast, in HHL the precondition may describe the relationship between several executions of a program. To achieve this, HHL uses assertions that quantify over states. We provide equivalent notation for this below:

$$\forall\langle\sigma\rangle.\varphi \triangleq \{m \mid \forall s \in \mathsf{supp}(m).\ m \in \varphi[s/\sigma]\} \qquad \exists\langle\sigma\rangle.\varphi \triangleq \{m \mid \exists s \in \mathsf{supp}(m).\ m \in \varphi[s/\sigma]\}$$

These bound metavariables $\sigma$, referring to states, can then be referenced in hypertests $B$, using the syntax below, where $\asymp \in \{=, \leq, \ldots\}$ ranges over the usual comparators.

$$B ::= \mathsf{true} \mid \mathsf{false} \mid \sigma(x) \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid \neg B \mid B_1 \asymp B_2 \mid \cdots$$

Unlike normal tests $b$, which can reference program variables, hypertests can only reference variables from particular executions $\sigma(x)$. So, for example, the assertion $\forall\langle\sigma\rangle.\ \sigma(x) = 5$ means that the variable $x$ has the value 5 in every execution, similar to the assertion $\Box(x = 5)$ that we saw in Section 5.1. However, the quantifiers introduced by HHL provide significant expressive power over the $\Box$ and $\Diamond$ modalities, since they allow us to express the relationship between program variables in multiple executions. For example, we can define the following low predicate [Dardinier and Müller 2024], which states that the value of some variable $\ell$ is the same in all executions.

$$\mathsf{low}(\ell) \triangleq \forall\langle\sigma\rangle.\ \forall\langle\tau\rangle.\ \sigma(\ell) = \tau(\ell)$$

We call this *low*, since we will use it to indicate that a variable has low sensitivity from an information security point of view. This allows us to both prove and disprove *noninterference*, a hyperproperty stating high sensitivity information cannot flow into the low sensitivity program variables. For example, the program below on the left is secure; if two executions have the same initial values of $\ell$, then they will also have the same final values for $\ell$. On the other hand, the program on the right is insecure; information flows from $h$ (a high-sensitivity input) to $\ell$, so the final values of $\ell$ will differ in any pair of executions where the initial values of $h$ differ.

$$\langle \mathsf{low}(\ell) \rangle\ \ell := \ell + 1\ \langle \mathsf{low}(\ell) \rangle \qquad \langle \mathsf{low}(\ell) \wedge \neg\mathsf{low}(h) \rangle\ \ell := h + 1\ \langle \neg\mathsf{low}(\ell) \rangle$$

We will not explore the full expressiveness of Hyper Hoare Logic here, as it is discussed extensively by Dardinier and Müller [2024]. Instead, we will show how some of the Hyper Hoare Logic rules can be derived using the Outcome Logic proof system. In particular, although the use of quantifiers over states may appear to complicate reasoning significantly, Dardinier and Müller [2024, §4] showed that this kind of reasoning can be achieved with simple syntactic rules. We recreate this result here by deriving the syntactic rules of HHL in Outcome Logic.

The first step is to fix a syntax for assertions. We use the one below, where the semantics of the assertions are the same as were defined notationally in Figure 2, with a few caveats. First, existential qualification can now only range over program values (as defined in Section 4), so that

[4] Note that we also used the powerset instance of Outcome Logic to encode Hoare Logic and Lisbon Logic in Section 5.1, however those examples used restricted pre- and postconditions to limit their expressivity. As was shown by Dardinier and Müller [2024, Propositions 2 and 9], HHL also subsumes Hoare and Lisbon Logic.

$T \subseteq \text{Val}$. Rather than use a predicate $\phi : T \to 2^{\mathcal{W}_{\mathcal{A}}(\Sigma)}$, we also now presume that the syntactic assertion $\varphi$ may reference the newly bound variable $v$. We have added universal quantification, which is defined analogously to existential quantification, but using an intersection instead of a union. The quantifiers over states $\forall\langle\sigma\rangle.\varphi$ and $\exists\langle\sigma\rangle.\varphi$ are as described above. Finally, hypertests $B$ only have a meaning if they are *closed*, that is, they do not contain any unbound state metavariables (notationally, we use $\sigma$ and $\tau$). Closed hypertests can be evaluated to Booleans in the usual way, and open hypertests (containing unbound variables) are considered to be false.

$$\varphi ::= \varphi \wedge \psi \mid \varphi \vee \psi \mid \forall v : T.\ \varphi \mid \exists v : T.\ \varphi \mid \forall\langle\sigma\rangle.\ \varphi \mid \exists\langle\sigma\rangle.\ \varphi \mid B$$

We define $\neg$ inductively, for example $\neg(\varphi \wedge \psi) \triangleq \neg\varphi \vee \neg\psi$ and $\neg(\forall\langle\sigma\rangle.\varphi) \triangleq \exists\langle\sigma\rangle.\neg\varphi$, the full details are in Appendix F. In addition, we let $\varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi$.

We will now give syntactic rules due to Dardinier and Müller [2024] for commands that interact with state. These rules are significant, since they define precisely how substitution and satisfaction of tests interact with the new quantifiers introduced in HHL.

*Variable Assignment.* We do not need a new rule for variable assignment, but rather we will show how to syntactically transform the postcondition $\varphi$ to match the semantic substitutions that we use in the Assign rule from Section 4. We first define an operation $E[\sigma]$ which transforms an expression $E$ into a hyper-expression by replacing all occurrences of variables $x$ with $\sigma(x)$. So for example, $(x + 2 \times y)[\sigma] = \sigma(x) + 2 \times \sigma(y)$. We write $\varphi[E[\sigma]/\sigma(x)]$ to be a standard capture-avoiding substitution, syntactically replacing any occurrence of $\sigma(x)$ with $E[\sigma]$. Now, we define a transformation on assertions $\mathcal{A}^E_x[-]$, which properly substitutes the expression $E$ for the variable $x$.

$$\begin{aligned}
&\mathcal{A}^E_x[\ \ \varphi \wedge \psi\ \ ] \triangleq \mathcal{A}^E_x[\varphi] \wedge \mathcal{A}^E_x[\psi] && \mathcal{A}^E_x[\ \ \varphi \vee \psi\ \ ] \triangleq \mathcal{A}^E_x[\varphi] \vee \mathcal{A}^E_x[\psi] \\
&\mathcal{A}^E_x[\ \forall v : T.\ \varphi\ ] \triangleq \forall v : T.\ \mathcal{A}^E_x[\varphi] && \mathcal{A}^E_x[\ \exists v : T.\ \varphi\ ] \triangleq \exists v : T.\ \mathcal{A}^E_x[\varphi] \\
&\mathcal{A}^E_x[\ \forall\langle\sigma\rangle.\ \varphi\ ] \triangleq \forall\langle\sigma\rangle.\ \mathcal{A}^E_x[\varphi[E[\sigma]/\sigma(x)]] && \mathcal{A}^E_x[\ \exists\langle\sigma\rangle.\ \varphi\ ] \triangleq \exists\langle\sigma\rangle.\ \mathcal{A}^E_x[\varphi[E[\sigma]/\sigma(x)]] \\
&\mathcal{A}^E_x[\ \ \ B\ \ \ ] \triangleq B
\end{aligned}$$

For most of the syntactic assertions, the $\mathcal{A}^E_x[-]$ operation is just propagated recursively. The interesting cases are the state quantifiers. Whenever a state $\sigma$ is quantified (either universally or existentially), then $E[\sigma]$ is syntactically substituted for $\sigma(x)$. In Lemma F.1 we show that $\mathcal{A}^E_x[-]$ is equivalent to the semantic substitution defined in Section 4, meaning that the Assign rule can be written as follows:

$$\frac{}{\langle \mathcal{A}^E_x[\varphi] \rangle\ x := E\ \langle \varphi \rangle}\ \textsc{Assign}$$

As an example, we will show how the Assign rule can be used to derive the specification that we gave above for the insecure program $\ell := h + 1$. We begin by applying the syntactic substitution to our desired postcondition $\neg\text{low}(\ell)$, as follows:

$$\begin{aligned}
\mathcal{A}^{h+1}_\ell[\neg\text{low}(\ell)] &= \mathcal{A}^{h+1}_\ell[\neg(\forall\langle\sigma\rangle.\ \forall\langle\tau\rangle.\ \sigma(\ell) = \tau(\ell))] \\
&= \mathcal{A}^{h+1}_\ell[\exists\langle\sigma\rangle.\ \exists\langle\tau\rangle.\ \sigma(\ell) \neq \tau(\ell))] \\
&= \exists\langle\sigma\rangle.\ \exists\langle\tau\rangle.\ \sigma(h) + 1 \neq \tau(h) + 1 \\
&= \neg\text{low}(h + 1)
\end{aligned}$$

Now, it is relatively easy to see that $\text{low}(\ell) \wedge \neg\text{low}(h) \Rightarrow \neg\text{low}(h) \Rightarrow \neg\text{low}(h + 1)$, so a simple application of Assign and the rule of Consequence completes the proof.

*Assume.* We now discuss a syntactic rule for assume statements. This will again involve defining some syntactic transformations on the postconditions. First, for any test $b$, we let $b[\sigma]$ be the hypertest obtained by replacing all occurrences of variables $x$ with $\sigma(x)$. For example, $(x = y + 1)[\sigma] = (\sigma(x) = \sigma(y) + 1)$. Now, we define $\Pi_b[-]$, which transforms an assertion $\varphi$ such that the test $b$ is true in all the quantified states.

$$\Pi_b[\ \varphi \wedge \psi\ ] \triangleq \Pi_b[\varphi] \wedge \Pi_b[\psi] \qquad \Pi_b[\ \varphi \vee \psi\ ] \triangleq \Pi_b[\varphi] \vee \Pi_b[\psi]$$
$$\Pi_b[\ \forall v\colon T.\ \varphi\ ] \triangleq \forall v\colon T.\ \Pi_b[\varphi] \qquad \Pi_b[\ \exists v\colon T.\ \varphi\ ] \triangleq \exists v\colon T.\ \Pi_b[\varphi]$$
$$\Pi_b[\ \forall\langle\sigma\rangle.\ \varphi\ ] \triangleq \forall\langle\sigma\rangle.\ b[\sigma] \Rightarrow \Pi_b[\varphi] \qquad \Pi_b[\ \exists\langle\sigma\rangle.\ \varphi\ ] \triangleq \exists\langle\sigma\rangle.\ b[\sigma] \wedge \Pi_b[\varphi]$$
$$\Pi_b[\ \ B\ \ ] \triangleq B$$

Once again, all cases except for the state quantifiers simply recursively apply $\Pi_b[-]$. In the case of state quantifiers, we modify quantification to be only over all states where $b$ holds. As we show in Lemma F.5, this means that $\Pi_b[\varphi] \Rightarrow (\varphi \wedge \Box b) \oplus \Box(\neg b)$, so that $\varphi$ will still hold after executing **assume** $b$. This allows us to derive the following Assume HHL rule:

$$\frac{}{\langle \Pi_b[\varphi] \rangle\ \textbf{assume}\ b\ \langle \varphi \rangle}\ \textsc{Assume HHL}$$

*Nondeterministic Assignment.* The final syntactic rules that we will define pertain to nondeterministic assignment. We again define a syntactic transformation on postconditions $\mathcal{H}_x^S[-]$ for programs where $x$ is nondeterministically assigned a value from the set $S$.[5]

$$\mathcal{H}_x^S[\ \varphi \wedge \psi\ ] \triangleq \mathcal{H}_x^S[\varphi] \wedge \mathcal{H}_x^S[\psi] \qquad \mathcal{H}_x^S[\ \varphi \vee \psi\ ] \triangleq \mathcal{H}_x^S[\varphi] \vee \mathcal{H}_x^S[\psi]$$
$$\mathcal{H}_x^S[\ \forall v\colon T.\ \varphi\ ] \triangleq \forall v\colon T.\ \mathcal{H}_x^S[\varphi] \qquad \mathcal{H}_x^S[\ \exists v\colon T.\ \varphi\ ] \triangleq \exists v\colon T.\ \mathcal{H}_x^S[\varphi]$$
$$\mathcal{H}_x^S[\ \forall\langle\sigma\rangle.\ \varphi\ ] \triangleq \forall\langle\sigma\rangle.\ \forall v\colon S.\ \mathcal{H}_x^S[\varphi[v/\sigma(x)]] \qquad \mathcal{H}_x^S[\ \exists\langle\sigma\rangle.\ \varphi\ ] \triangleq \exists\langle\sigma\rangle.\ \exists v\colon s.\ \mathcal{H}_x^S[\varphi[v/\sigma(x)]]$$
$$\mathcal{H}_x^S[\ \ B\ \ ] \triangleq B$$

The operation $\mathcal{H}_x^S[-]$ is similar to $\mathcal{A}_x^E[-]$, except that in the cases for state quantifiers, $\sigma(x)$ is replaced with any (or some) value in the set $S$. So, for example, if we wish to nondeterministically set $x$ to be either 1 or 2, and then assert that $\forall\langle\sigma\rangle.\ \exists\langle\tau\rangle.\ \sigma(x) \neq \tau(x)$, then we get:

$$\mathcal{H}_x^{\{1,2\}}[\forall\langle\sigma\rangle.\ \exists\langle\tau\rangle.\ \sigma(x) \neq \tau(x)] = \forall\langle\sigma\rangle.\ \forall v\colon \{1,2\}.\ \exists\langle\tau\rangle.\ \exists v'\colon \{1,2\}.\ v \neq v'$$

The above assertion is trivially true. We can now use $\mathcal{H}_x^S[-]$ to derive inference rules for nondeterministic assignments. First, we will define syntactic sugar for an additional programming construct that nondeterministically assigns a variable $x$ to be any natural number. We define this using the Kleene star $C^\star$ defined in Section 2.4, which repeats $C$ a nondeterministic number of times.

$$x := \bigstar \qquad \triangleq \qquad x := 0 \fatsemi (x := x + 1)^\star$$

Now, by selecting the appropriate set $S$, we can derive the following havoc rules for nondeterministic assignments. The derivations of these rules are shown in Appendix F.2.

$$\frac{}{\langle \mathcal{H}_x^{\{a,b\}}[\varphi] \rangle\ (x := a) + (x := b)\ \langle \varphi \rangle}\ \textsc{Havoc-2} \qquad \frac{}{\langle \mathcal{H}_x^{\mathbb{N}}[\varphi] \rangle\ x := \bigstar\ \langle \varphi \rangle}\ \textsc{Havoc-N}$$

[5] This varies slightly from the definition of Dardinier and Müller [2024], $\mathcal{H}_x[-]$, in which $x$ is assigned any value, not from a particular set. We can recover their operation as $\mathcal{H}_x[-] = \mathcal{H}_x^{\mathsf{Val}}[-]$.

## 6 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. Existing 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 6.1. We then show how a single proof about that program can be reused in both a partial correctness specification (Section 6.2) and a probabilistic program (Section 6.3). The full derivations are given in Appendix G.

### 6.1 Integer Division

In order to avoid undefined behavior related to division by zero, our expression syntax from Section 4 does not include division. However, we can write a simple procedure to divide two natural numbers $a$ and $b$ using repeated subtraction.

$$\texttt{Div} \triangleq \left\{ \begin{array}{l} q \coloneqq 0 \fatsemi r \coloneqq a \fatsemi \\ \textbf{while}\ r \geq b\ \textbf{do} \\ \quad r \coloneqq r - b \fatsemi \\ \quad q \coloneqq q + 1 \end{array} \right.$$

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 $\varphi_n$ shown below.

$$\varphi_n \triangleq \left\{ \begin{array}{lll} \lceil q + n = \lfloor a \div b \rfloor \wedge r = (a \bmod b) + n \times b \rceil & \text{if} & n \leq \lfloor a \div b \rfloor \\ \bot & \text{if} & n > \lfloor a \div b \rfloor \end{array} \right.$$

Executing the loop body in a state satisfying $\varphi_n$ results in a state satisfying $\varphi_{n-1}$. At the end, $\varphi_0$ 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.

$$\langle \lceil a \geq 0 \wedge b > 0 \rceil \rangle\ \texttt{Div}\ \langle \lceil q = \lfloor a \div b \rfloor \wedge r = a \bmod b \rceil \rangle$$

Note that the **Div** program is deterministic; it does not use branching and 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.

### 6.2 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.

$$\textbf{Collatz} \triangleq \left\{ \begin{array}{l} i \coloneqq 0 \fatsemi \\ \textbf{while } a \neq 1 \textbf{ do} \\ \quad b \coloneqq 2 \fatsemi \textbf{Div} \fatsemi \\ \quad \textbf{if } r = 0 \textbf{ then } a \coloneqq q \textbf{ else } a \coloneqq 3 \times a + 1 \fatsemi \\ \quad i \coloneqq i + 1 \end{array} \right.$$

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.

$$\langle \lceil a = n \wedge n > 0 \rceil \rangle \ \textbf{Collatz} \ \langle \Box (i = S_n) \rangle$$

### 6.3 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 2.10.

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

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.

$$\varphi_n \triangleq \lceil a = n \wedge r = a \bmod 2 \rceil^{(\frac{1}{2^n})} \qquad \psi_n \triangleq \lceil a = n \wedge r = a \bmod 2 \rceil^{(\frac{1}{2^{n+1}})}$$

According to Iter, the final postcondition can be obtained by taking an outcome conjunction of all the $\psi_n$ 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.

$$\langle \lceil \mathsf{true} \rceil \rangle \ a \coloneqq 0 \fatsemi r \coloneqq 0 \fatsemi (a \coloneqq a + 1 \fatsemi b \coloneqq 2 \fatsemi \textbf{Div})^{\langle \frac{1}{2} \rangle} \ \langle \lceil r = 0 \rceil \oplus_{\frac{2}{3}} \lceil r = 1 \rceil \rangle$$

### 6.4 Implications to Program Analysis

Building on the proof reusability demonstrated in this case study, we now explore how Outcome Logic can be used as a unifying foundation for correctness and incorrectness static analysis. Although correctness and incorrectness can have many meanings, we follow the lead of O'Hearn [2020], with the distinction coming down to demonic vs angelic nondeterminism. More precisely, a correctness specification covers *all* traces whereas an incorrectness specification witnesses the *existence* of a single faulty one.

Many real world static analysis systems operate in this way, such as Meta's Infer tool [Calcagno et al. 2015], which was initially developed as a verification engine to prove memory safety in large codebases. It accordingly reports specifications as Hoare Triples $\{P\}\ C\ \{Q\}$, where $P$ specifies the resources that must be available in order for the program $C$ to execute safely in all traces [Calcagno et al. 2009, 2011]. However, it was later discovered that Infer was more effective as a bug finding tool, both because correctness analysis was sometimes computationally intractable and also because the codebases contained bugs [Distefano et al. 2019].

The problem is that Hoare Logic is not sound as a logical foundation for bug finding—it admits false positives [O'Hearn 2020]. More specifically, if an erroneous outcome occurs nondeterministically, then Hoare Logic has no way to witness a trace proving that the bug occurs. For example, if $Q_{\text{ok}}$ represents the desired outcome and $Q_{\text{er}}$ represents the erroneous one, then the Hoare Logic postcondition must be a disjunction of the two: $\{P\}\ C\ \{Q_{\text{ok}} \vee Q_{\text{er}}\}$, but given a specification of that form, it is possible that every execution falls into the $Q_{\text{ok}}$ branch. In practice, postconditions become imprecise due to the use of abstraction [Calcagno et al. 2009], and so Infer often fails to prove the absence of bugs, but that cannot be used as evidence that a bug exists [Raad et al. 2020].

In response, the Infer team developed a new analysis called Pulse, which is based on angelic nondeterminism and therefore has a different logical foundation [Le et al. 2022; Raad et al. 2020]. As we saw in Section 5.1, both forms of nondeterminism can be represented in Outcome Logic, by using different modalities in the postcondition[6].

$$\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle \qquad\qquad \langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle$$

However, these two types of specifications are still incompatible; one cannot be used in a subderivation of the other. Crucially—as is the case with Infer and Pulse—this means that intermediate procedure specifications cannot be shared between correctness and incorrectness analyses. This is unfortunate, as computing those intermediate specifications in a large codebase is costly.

Fortunately, sharable specifications can be expressed in Outcome Logic. The trick is to use specifications of the form $\langle \lceil P \rceil \rangle\ C\ \langle \lceil Q_1 \rceil \oplus \cdots \oplus \lceil Q_n \rceil \rangle$ wherever possible, which both cover all the outcomes (for correctness) and also guarantee reachability of each $\lceil Q_k \rceil$ (for bug finding). Many procedures—even looping ones—fit into this format, *e.g.*, the one we saw in Section 6.1.

On the other hand, some procedures will need to be analyzed using specialized techniques. For example, the use of a loop invariant in Section 6.2 introduced a $\Box$ modality to indicate that there may not be any terminating outcomes. Alternatively, if a bug arises in one of the paths, the analysis can introduce a $\Diamond$ modality in order to retain less information about the other paths. Both of these can be achieved by an application of the rule of consequence.

$$\frac{\langle \lceil P \rceil \rangle\ C\ \langle \lceil Q_1 \rceil \oplus \cdots \oplus \lceil Q_n \rceil \rangle \quad \forall k.\ Q_k \Rightarrow Q}{\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle} \qquad \frac{\langle \lceil P \rceil \rangle\ C\ \langle \lceil Q_1 \rceil \oplus \cdots \oplus \lceil Q_n \rceil \rangle}{\langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q_k \rangle}$$

Zilberstein et al. [2024] designed an algorithm for this type of analysis, showing not only that Outcome Logic models Infer and Pulse, but also that the engines of those tools could be consolidated so as to share procedure specifications in many cases. In this paper, we provided new inference rules, particularly for loops, giving those algorithms more sound reasoning principles to build on.

## 7 Case Study: Graph Problems

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

### 7.1 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

[6] Although Pulse was initially based on Incorrectness Logic [O'Hearn 2020], it was observed by Zilberstein et al. [2024] and Raad et al. [2024] that it can also be modeled using Lisbon Logic

either the $x$ or $y$-axis (or steps in a fixed direction if the destination on one axis is already reached).

$$\textbf{Walk} \triangleq \left\{ \begin{array}{l} \textbf{while } x < N \vee y < M \textbf{ do} \\ \quad \textbf{if } x < N \wedge y < M \textbf{ then} \\ \quad\quad (x \coloneqq x + 1) + (y \coloneqq y + 1) \\ \quad \textbf{else if } x \geq N \textbf{ then} \\ \quad\quad y \coloneqq y + 1 \\ \quad \textbf{else} \\ \quad\quad x \coloneqq x + 1 \end{array} \right.$$

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 2.9) 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 $\varphi_n$ 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 $x = N - n \wedge y = M$ to $x = N \wedge y = M - n$. 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:

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

The loop body moves the program state from $\varphi_{n+1}$ to $\varphi_n$. The outcomes of $\varphi_{n+1}$ 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:

$$\varphi_{N+M} = \bigoplus_{k=N}^{N} \lceil x = N - k \wedge y = N - k \rceil^{\left(\binom{0}{N-N}\right)} = \lceil x = 0 \wedge y = 0 \rceil$$

In addition, the postcondition is:

$$\varphi_0 = \bigoplus_{k=0}^{0} \lceil x = N - k \wedge y = M + k \rceil^{\left(\binom{N+M}{N}\right)} = \lceil x = N \wedge y = M \rceil^{\left(\binom{N+M}{N}\right)}$$

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 H.1.

$$\langle \lceil x = 0 \wedge y = 0 \rceil \rangle \ \textbf{Walk} \ \langle \lceil x = N \wedge y = M \rceil^{\left(\binom{N+M}{N}\right)} \rangle$$

### 7.2 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 $G[i][j] = \text{true}$ 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 $next \leq N$, 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.

$$\mathbf{SP} \triangleq \left\{\begin{array}{l} \textbf{while } pos \neq t \textbf{ do} \\ \quad next := 1 \fatsemi \\ \quad (next := next + 1)^{\langle next < N,\, G[pos][next]\rangle} \fatsemi \\ \quad pos := next \fatsemi \\ \quad \textbf{assume } 1 \end{array}\right.$$

We will interpret this program using the Tropical semiring from Example 2.11, 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: $\mathrm{sp}^t_n(G, s, s')$ indicates whether there is a path of length $n$ from $s$ to $s'$ in $G$ without passing through $t$ and $\mathrm{sp}(G, s, t)$ is the shortest path length from $s$ to $t$. Let $I = \{1, \ldots, N\} \setminus \{t\}$.

$$\mathrm{sp}^t_0(G, s, s') \triangleq (s = s')$$

$$\mathrm{sp}^t_{n+1}(G, s, s') \triangleq \bigvee_{i \in I} \mathrm{sp}^t_n(G, s, i) \wedge G[i][s']$$

$$\mathrm{sp}(G, s, t) \triangleq \inf\{n \in \mathbb{N}^\infty \mid \mathrm{sp}^t_n(G, s, t)\}$$

We analyze the while loop using the While rule, which requires $\varphi_n$ and $\psi_n$ to record the outcomes where the loop guard is true or false, respectively, after $n$ iterations. We define these as follows, where + denotes regular arithmetic addition rather than addition in the tropical semiring.

$$\varphi_n = \bigoplus_{i \in I} \lceil pos = i \rceil^{(\mathrm{sp}^t_n(G,s,i)+n)} \qquad \psi_n = \lceil pos = t \rceil^{(\mathrm{sp}^t_n(G,s,t)+n)} \qquad \psi_\infty = \lceil pos = t \rceil^{(\mathrm{sp}(G,s,t))}$$

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 $\infty$ otherwise. The final postcondition $\psi_\infty$ is the shortest path length to $t$, which is also the infimum of $\mathrm{sp}^t_n(G, s, t) + n$ for all $n$. Using the Iter rule we get the following derivation for the inner loop:

$$\begin{array}{l} \left\langle \bigoplus_{i \in I} \lceil pos = i \wedge next = 1 \rceil^{(\mathrm{sp}^t_n(G,s,i)+n)} \right\rangle \\ \quad (next := next + 1)^{\langle next < N,\, G[pos][next]\rangle} \fatsemi \\ \left\langle \bigoplus_{j=1}^{N} \bigoplus_{i \in I} \lceil pos = i \wedge next = j \rceil^{((\mathrm{sp}^t_n(G,s,i) \wedge G[i][j])+n)} \right\rangle \\ \quad pos := next \fatsemi \textbf{assume } 1 \\ \left\langle \bigoplus_{j=1}^{N} \bigoplus_{i \in I} \lceil pos = j \rceil^{((\mathrm{sp}^t_n(G,s,i) \wedge G[i][j])+n+1)} \right\rangle \implies \\ \left\langle \bigoplus_{j=1}^{N} \lceil pos = j \rceil^{(\mathrm{sp}^t_{n+1}(G,s,j)+n+1)} \right\rangle \end{array}$$

The outcome conjunction over $i \neq t$ (corresponding to the minimum weight path) gives us $pos = j$ with weight $\mathrm{sp}^t_{n+1}(G, s, j) + n + 1$—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 $\mathrm{sp}^t_0(G, s, i) = \text{false}$ 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.

$$\langle \lceil pos = s \rceil \rangle \ \mathbf{SP} \ \langle \lceil pos = t \rceil^{(\mathrm{sp}(G,s,t))} \rangle$$

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

## 8 Related Work

*Correctness, Incorrectness, and Unified Program Logics.* While formal verification has long been the aspiration for static analysis, bug-finding tools are often more practical in real-world engineering settings. This partly comes down to efficiency—bugs can be found without considering all traces—and partly due to the fact that most real world software just is not correct [Distefano et al. 2019].

However, the 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 [2020] 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.

Although Incorrectness Logic successfully serves as a logical foundation for bug-finding tools [Le et al. 2022; Raad et al. 2020], it is semantically incompatible with correctness analysis, making sharing of toolchains difficult or impossible. Attention has therefore turned to unifying correctness and incorrectness theories. This includes Exact Separation Logic, which combines Hoare Logic and Incorrectness Logic to generate specifications that are valid for both, but that also precludes under- or over-approximation via the rule of consequence [Maksimović et al. 2023]. Local Completeness Logic combines Incorrectness Logic with an over-approximate abstract domain to similar effect; it also precludes dropping paths [Bruni et al. 2021, 2023].

In other recent work, Cousot [2024] constructs proof systems for nondeterministic program logics using fixpoint abstraction, providing a means to compare their semantics through the lens of abstract interpretation. Our goal in this paper is orthogonal; we capture many different types of specifications with a *single* semantics and proof theory, so as to reuse proof fragments as much as possible. We also sought to include several kinds of effects rather than just nondeterminism.

*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 [Harel et al. 2001; Pratt 1976] and Hoare's [1978] calculus of *possible correctness*. The idea was briefly mentioned in the Incorrectness Logic literature [Le et al. 2022; Möller et al. 2021; O'Hearn 2020], but using Lisbon Logic as a foundation of incorrectness analysis was not fully explored until the introduction of Outcome Logic [Zilberstein et al. 2023], which subsumes both Lisbon Logic and Hoare Logic, as we saw in Section 5.1. The metatheory of Lisbon Logic has subsequently been explored more deeply [Ascari et al. 2023; Raad et al. 2024]. Hyper Hoare Logic also generalizes Hoare and Lisbon Logics [Dardinier and Müller 2024], and is semantically equivalent to the Boolean instance of OL (Example 2.7), but does not support effects other than nondeterminism.

Outcome Logic initially used a model based on both a monad and a monoid, with looping defined via the Kleene star $C^\star$ [Zilberstein et al. 2023]. The semantics of $C^\star$ had to be justified for each instance, however it is not compatible with probabilistic computation (see Footnote 1), so an ad-hoc semantics was used in the probabilistic case. Moreover, only the Induction rule was provided for reasoning about $C^\star$, amounting 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. As we showed in Sections 6 and 7, Iter 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 separation logic instance [Zilberstein et al. 2024]. The model was refined to use semirings, and the programming 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 2.9). Rather than giving inference rules, the paper provided a symbolic execution algorithm, which also only supported loops via bounded unrolling.

This article extends the prior conference papers 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 loop, deriving additional inference rules for Hoare and Lisbon Logic embeddings, 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 [Floyd 1967a; Hoare 1969], but total correctness (requiring termination) was later introduced too [Manna and Pnueli 1974]. More recently, automated tools were developed to prove (non)termination in real-world software [Berdine et al. 2006; Brockschmidt et al. 2013; Cook et al. 2014, 2006a,b; Raad et al. 2024].

Nondeterminism also showed up in early variants of Hoare Logic, stemming from Dijkstra's Guarded Command Language (GCL) [Dijkstra 1975] and Dynamic Logic [Harel et al. 2001; Pratt 1976]; it is useful for modeling backtracking algorithms [Floyd 1967b] and opaque aspects of program evaluation such as user input and concurrent scheduling. While Hoare Logic has traditionally used demonic nondeterminism [Broy and Wirsing 1981], other program logics have recently arisen to deal with nondeterminism in different ways, particularly for incorrectness [Ascari et al. 2023; Dardinier and Müller 2024; de Vries and Koutavas 2011; Möller et al. 2021; O'Hearn 2020; Raad et al. 2024; Zilberstein et al. 2023].

Beginning with the seminal work of Kozen [Kozen 1979, 1983], the study of probabilistic programs has a rich history. This eventually led to the development of probabilistic Hoare Logic variants [Barthe et al. 2018; den Hartog 2002, 1999; Rand and Zdancewic 2015] that enable reasoning about programs in terms of likelihoods and expected values. Doing so requires pre- and postconditions to be predicates on probability distributions.

Outcome Logic provides abilities to reason about those effects 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 6.

*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 [1978] 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.

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 post$(C, P)$ from Definition 3.5. Implications for an *expressive* language quickly become undecidable, as they must encode Peano arithmetic [Apt 1981; Lipton 1977]. 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 [Barthe et al. 2018; den Hartog 1999]. den Hartog [2002] devised a relatively complete probabilistic logic with syntactic assertions, but the programming language does not include loops and is therefore considerably simplified; it is unclear if this approach would extend to looping programs. In addition Batz et al. [2021] created an expressive language for expectation-based reasoning, however the language only has constructs to describe states and expected values; it does not contain a construct like $\oplus$ to express properties involving of multiple traces of the program. As we saw in the discussion following Definition 3.5, the ability to describe multiple executions makes the expressivity question more complex.

Several program logics (including our own) use semantic assertions, which are trivially expressive [Ascari et al. 2023; Barthe et al. 2018; Calcagno et al. 2007; Cousot et al. 2012; Dardinier and Müller 2024; Jung et al. 2018, 2015; Kaminski 2019; O'Hearn 2020; Raad et al. 2024; Yang 2001]. This includes logics that are mechanized within proof assistants [Barthe et al. 2018; Dardinier and Müller 2024; Jung et al. 2018, 2015], so we do not see the extensional nature of our approach as a barrier to mechanization in the future.

*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 [Kozen 1983] and weakest pre-expectation calculi [Kaminski 2019; Morgan et al. 1996] are used to reason about randomized programs in terms of expected values. This idea has been extended to non-probabilistic quantitative properties too [Batz et al. 2022; Zhang and Kaminski 2022; Zhang et al. 2024].

Weighted programming [Batz et al. 2022] generalizes pre-expectation reasoning using semirings to model branch weights, much like 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 6.3, we derive a single OL triple that gives the probabilities of two outcomes, whereas weighted programming (or weakest pre-expectations) would need to compute the weight of each branch individually. In addition, Batz et al. [2022] only support total semirings, so they cannot analyze standard probabilistic programs. Weighted programming was extended to handle hyperproperties by Zhang et al. [2024], which can be seen as a weakest precondition calculus for Outcome Logic.

## 9 Conclusion, Limitations, and Future Work

In this article, we have presented a proof system for Outcome Logic, which is sufficient for reasoning about programs with branching effects. That is, effects that can be semantically encoded by assigning weights to a collection of final states. However, there remains room for future development of Outcome Logic in order to support additional kinds of effects and other capabilities. In this section, we describe the limitations of the present formalization, and how those limitations could be addressed in future work.

*Total Correctness and Nontermination.* Although Outcome Logic provides some ability to reason about nontermination—*i.e.,* it can be used to prove that a program *sometimes* or *never* terminates—it cannot be used for total correctness [Manna and Pnueli 1974], as the semantics only tracks terminating traces, not the existence of infinite ones. Given some specification $\langle \varphi \rangle\ C\ \langle \psi \rangle$, it is therefore impossible to know whether additional nonterminating behavior outside of $\psi$ is possible.

The challenge in tracking infinite traces is that the semantics can easily become non-continuous, making it more difficult to prove the existence of fixed points used to define the semantics of loops. More precisely, Apt and Plotkin [1986] showed that no continuous semantics is possible that both distinguishes nontermination and also supports unbounded nondeterminism. That means that, at least in the nondeterministic interpretation (Example 2.7), a primitive action of the following form, which assigns $x$ to a nondeterministically chosen natural number, must be precluded.

$$[\![x := \bigstar]\!]\,(s) \quad \triangleq \quad \sum_{n \in \mathbb{N}} \eta(s[x := n]) \quad \cong \quad \{s[x := n] \mid n \in \mathbb{N}\}$$

Note that in the present formulation, we could define the above construct as syntactic sugar, as we did in Section 5.2. The program below has the exact same semantics as above, since our model in Section 2 does not record the fact that it also has an infinite (nonterminating) execution.

$$x := \bigstar \quad \triangleq \quad x := 0 \fatsemi (x := x + 1)^\star$$

However, in a total correctness version, the semantics of those programs would not be the same, corresponding to Dijkstra's [1976] observation that a program cannot make infinitely many choices in a finite amount of time, and giving an operational argument for why unbounded nondeterminism should not be allowed. However, the story is more murky when generalizing to the semiring weighted model that we have presented in this paper. As we just saw, unbounded choice is not valid in the nondeterministic model, but it is valid in the probabilistic model (Example 2.10). For example, the following probabilistic program makes infinitely many choices, but it also *almost surely terminates* (there is an infinite trace, but it occurs with probability 0, so we do not consider it a possible outcome).

$$x := 0 \fatsemi (x := x + 1)^{\langle \frac{1}{2} \rangle}$$

So, in the total correctness setting, the continuity of program operators depends on the semiring in a non-obvious way, which cannot be explained by existing work on powerdomains [Plotkin 1976; Smyth 1978; Søndergaard and Sestoft 1992]. In the future, we hope to develop a semantic model for Outcome Logic that explains this discrepancy in terms of properties of the semirings.

*Mixing Effects.* While Outcome Logic can be used to analyze programs that have various types of branching, it cannot yet handle programs that display multiple kinds of branching in tandem. Of particular interest is the combination of probabilistic choice and nondeterminism, which would be useful to analyze randomized distributed systems (where nondeterminism arises due to concurrent scheduling). Unfortunately, the typical powerset monad representation of nondeterminism (Example 2.7) does not compose well with probability distributions (Example 2.10) [Varacca and Winskel 2006], and similar restrictions apply to other combinations of semirings [Parlant 2020; Zwart 2020; Zwart and Marsden 2019].

However, it was recently shown that probability distributions do compose well with multisets [Jacobs 2021; Kozen and Silva 2023]. In Section 7.1, we showed how the Outcome Logic instance based on multisets (Example 2.9) can be used for quantitative analysis, however that example did not correspond to a canonical model of computation. By contrast, a logic using that same multiset instance for probabilistic nondeterminism would model a very pertinent combination of effects, motivating the study of these more exotic forms of nondeterminism in this paper.

*Mutable State, Separation Logic, and Concurrency.* As an extension to our Outcome Logic formulation in this paper, it would be possible to add atomic commands for dynamically allocating and mutating heap pointers, and separation logic [Reynolds 2002] style inference rules to reason about mutation. Separation logic introduces new logical primitives such as the points-to predicate $x \mapsto E$—stating that the $x$ is a pointer whose address maps to the value of $E$ on the heap—and the separating conjunction $P * Q$—stating that $P$ and $Q$ hold in disjoint regions of the heap, guaranteeing that if $x \mapsto E_1 * y \mapsto E_2$ holds, then $x$ and $y$ do not alias each other.

This idea was already explored in Outcome Separation Logic (OSL) [Zilberstein et al. 2024], but questions relating to unbounded looping and completeness were not addressed in that work. In separation logic, it is typical to provide *small axioms* [O'Hearn et al. 2001], which specify the behavior of mutation locally. For example, the following Write rule specifies the behavior of mutation in a singleton heap. Global reasoning is then done via the Frame rule [Yang and O'Hearn 2002], which states that a proof done in a local footprint is also valid in a larger heap.

$$\frac{}{\{x \mapsto -\}\ [x] \leftarrow E\ \{x \mapsto E\}}\textsc{Write} \qquad \frac{\{P\}\ C\ \{Q\}}{\{P * F\}\ C\ \{Q * F\}}\textsc{Frame}$$

Yang [2001] showed that the small axioms along with the Frame rule are complete for standard separation logic; any specification can be derived using them. By contrast, Zilberstein et al. [2024] showed that the small axioms and Frame rule are sound in OSL, but it is not known whether they are complete. In fact, there is some evidence to the contrary, since the resources specified outside the footprint of the local mutation may take on many different values, and therefore the Frame rule would need to be applied for each branch of the computation. It would be interesting to reformulate OSL with the looping constructs of this paper and explore the notion of completeness in that setting.

Going further, it would also be interesting to explore a concurrent variant of OSL. Along with the aforementioned challenges of reasoning about weighted branching in tandem with the nondeterministic behavior of the concurrent scheduler, this would also require a more powerful version of separation than that offered by OSL. The OSL separating conjunction $\varphi \circledast P$ is asymmetric; $\varphi$ is an unrestricted outcome assertion, whereas $P$ can only be a basic assertion, like those described in Section 3.1. However, concurrent separation logic [O'Hearn 2004] typically requires the entire state space to be separated in order to compositionally analyze concurrent threads, so a symmetric separating conjunction $\varphi \circledast \psi$ would be needed.

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 [Zilberstein et al. 2023, 2024] captures those properties in a unified way, but until now the proof theory and connections to other logics have not been thoroughly explored.

This article expands on the prior Outcome Logic work and provides a comprehensive account of the OL metatheory by presenting a relatively complete proof system for Outcome Logic and a significant number of derived rules. This shows that programs with branching effects are not only *semantically* similar, but also share common reasoning principles. Specialized techniques (*i.e.,* analyzing loops with variants or invariants) are particular modes of use of our more general logic, and are compatible with each other rather than requiring distinct semantics.

## References

Samson Abramsky and Achim Jung. 1995. *Domain theory.* Oxford University Press, Inc., USA, 1–168.

Krzysztof Apt and Gordon Plotkin. 1986. Countable nondeterminism and random assignment. *J. ACM* 33, 4 (aug 1986), 724–767. https://doi.org/10.1145/6490.6494

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

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

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

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

Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2021. Relatively complete verification of probabilistic programs: an expressive language for expectation-based reasoning. *Proc. ACM Program. Lang.* 5, POPL, Article 39 (jan 2021), 30 pages. https://doi.org/10.1145/3434320

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

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

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

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

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

Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. 2015. Moving Fast with Software Verification. In *NASA Formal Methods*. Springer International Publishing, Cham, 3–11. https://doi.org/10.1007/978-3-319-17524-9_1

Cristiano Calcagno, Dino Distefano, Peter O'Hearn, and Hongseok Yang. 2009. Compositional Shape Analysis by Means of Bi-Abduction. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Savannah, GA, USA) *(POPL '09)*. Association for Computing Machinery, New York, NY, USA, 289–300. https://doi.org/10.1145/1480881.1480917

Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. 2011. Compositional Shape Analysis by Means of Bi-Abduction. *J. ACM* 58, 6, Article 26 (Dec 2011), 66 pages. https://doi.org/10.1145/2049697.2049700

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

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

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

Byron Cook, Andreas Podelski, and Andrey Rybalchenko. 2006b. 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

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

Patrick Cousot. 2024. Calculational Design of [In]Correctness Transformational Program Logics by Abstract Interpretation. *Proc. ACM Program. Lang.* 8, POPL, Article 7 (jan 2024), 34 pages. https://doi.org/10.1145/3632849

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

Thibault Dardinier and Peter Müller. 2024. Hyper Hoare Logic: (Dis-)Proving Program Hyperproperties. *Proc. ACM Program. Lang.* 8, PLDI, Article 207 (jun 2024), 25 pages. https://doi.org/10.1145/3656437

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

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

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.

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

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

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

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

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

Robert W. Floyd. 1967a. 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.

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

David Harel, Dexter Kozen, and Jerzy Tiuryn. 2001. Dynamic logic. *SIGACT News* 32, 1 (2001), 66–69. https://doi.org/10.1145/568438.568456

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

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

Bart Jacobs. 2021. From Multisets over Distributions to Distributions over Multisets. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science* (Rome, Italy) *(LICS '21).* Association for Computing Machinery, New York, NY, USA, Article 39, 13 pages. https://doi.org/10.1109/LICS52264.2021.9470678

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

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

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

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

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

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

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

Dexter Kozen and Alexandra Silva. 2023. Multisets and Distributions. arXiv:2301.10812 [cs.LO]

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.

Werner Kuich. 2011. Algebraic systems and pushdown automata. In *Algebraic foundations in computer science. Essays dedicated to Symeon Bozapalidis on the occasion of his retirement.* Berlin: Springer, 228–256. https://doi.org/10.1007/978-3-642-24897-9_11

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

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

Andreas Lööw, Daniele Nantes-Sobrinho, Sacha-Élie Ayoun, Caroline Cronjäger, Petar Maksimović, and Philippa Gardner. 2024. Compositional Symbolic Execution for Correctness and Incorrectness Reasoning. In *38th European Conference on Object-Oriented Programming (ECOOP 2024) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 313)*, Jonathan

Aldrich and Guido Salvaneschi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 25:1–25:28. https://doi.org/10.4230/LIPIcs.ECOOP.2024.25

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

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

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

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

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

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

Peter W. O'Hearn. 2004. Resources, Concurrency and Local Reasoning. In *CONCUR 2004 - Concurrency Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 49–67. https://doi.org/10.1016/j.tcs.2006.12.035

Peter W. O'Hearn. 2020. Incorrectness Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 10 (Jan. 2020), 32 pages. https://doi.org/10.1145/3371078

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

Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. 2001. Local Reasoning about Programs That Alter Data Structures. In *Proceedings of the 15th International Workshop on Computer Science Logic (CSL '01)*. Springer-Verlag, Berlin, Heidelberg, 1–19. https://doi.org/10.1007/3-540-44802-0_1

Louis Parlant. 2020. *Monad Composition via Preservation of Algebras*. Ph.D. Dissertation. University College London. https://discovery.ucl.ac.uk/id/eprint/10112228/

Gordon Plotkin. 1976. A Powerdomain Construction. *SIAM J. Comput.* 5, 3 (1976), 452–487. https://doi.org/10.1137/0205035 arXiv:https://doi.org/10.1137/0205035

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

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

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

Azalea Raad, Julien Vanegue, and Peter O'Hearn. 2024. Non-termination Proving at Scale. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 280 (Oct 2024), 29 pages. https://doi.org/10.1145/3689720

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).

J.C. Reynolds. 2002. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*. 55–74. https://doi.org/10.1109/LICS.2002.1029817

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

Michael Smyth. 1978. Power domains. *J. Comput. System Sci.* 16, 1 (1978), 23–36. https://doi.org/10.1016/0022-0000(78)90048-X

Harald Søndergaard and Peter Sestoft. 1992. Non-determinism in Functional Languages. *Comput. J.* 35, 5 (10 1992), 514–523. https://doi.org/10.1093/comjnl/35.5.514 arXiv:https://academic.oup.com/comjnl/article-pdf/35/5/514/1125580/35-5-514.pdf

Daniele Varacca and Glynn Winskel. 2006. Distributing probability over non-determinism. *Mathematical Structures in Computer Science* 16, 1 (2006), 87–113. https://doi.org/10.1017/S0960129505005074

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

Hongseok Yang and Peter O'Hearn. 2002. A Semantic Basis for Local Reasoning. In *Foundations of Software Science and Computation Structures*. Springer Berlin Heidelberg, Berlin, Heidelberg, 402–416. https://doi.org/10.1007/3-540-45931-6_28

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

Linpeng Zhang, Noam Zilberstein, Benjamin Lucien Kaminski, and Alexandra Silva. 2024. Quantitative Weakest Hyper Pre: Unifying Correctness and Incorrectness Hyperproperties via Predicate Transformers. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 300 (oct 2024), 30 pages. https://doi.org/10.1145/3689740

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

Noam Zilberstein, Angelina Saliling, and Alexandra Silva. 2024. Outcome Separation Logic: Local Reasoning for Correctness and Incorrectness with Computational Effects. *Proc. ACM Program. Lang.* 8, OOPSLA1 (Apr 2024). https://doi.org/10.1145/3649821

Maaike Zwart. 2020. *On the Non-Compositionality of Monads via Distributive Laws*. Ph.D. Dissertation. University of Oxford. https://ora.ox.ac.uk/objects/uuid:b2222b14-3895-4c87-91f4-13a8d046febb

Maaike Zwart and Dan Marsden. 2019. No-Go Theorems for Distributive Laws. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–13. https://doi.org/10.1109/lics.2019.8785707

# Appendix

## A Totality of Language Semantics

### 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.

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

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

$$
\begin{aligned}
[\![b]\!]\,(\sigma) &\triangleq [\![b]\!]_{\mathsf{Test}}\,(\sigma) \\
[\![u]\!]\,(\sigma) &\triangleq u
\end{aligned}
$$

### A.2 Fixed Point Existence

For all the proofs in this section, we assume that the operations $+$, $\cdot$, and $\sum$ belong to a Scott continuous, naturally ordered, partial semiring with a top element (as described in Section 2.3).

Lemma A.1. *Let $\langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ be a continuous, naturally ordered, partial semiring. For any family of Scott continuous functions $(f_i : X \to \mathcal{W}(Y))_{i \in I}$ 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. The proof proceeds by transfinite induction on the size of $I$.

- **Base case:** $I = \emptyset$, so clearly:

$$\sup_{x \in D} \sum_{i \in \emptyset} f_i(x) = \sup_{x \in D} \mathbb{0} = \mathbb{0} = \sum_{i \in \emptyset} \sup_{x \in D} f_i(x)$$

- **Successor Case:** Suppose the claim holds for sets of size $\alpha$, and let $|I| = \alpha + 1$. We can partition $I$ into $I' \cup \{i\}$ where $i \in I$ is an arbitrary element and $I' = I \setminus \{i\}$ so that $|I'| = \alpha$. Now, we have that:

$$\sup_{x \in D} \sum_{j \in I} f_j(x) = \sup_{x \in D} (\sum_{j \in I'} f_j(x)) + f_i(x)$$

Now, note that since all the $f_j$ functions are Scott continuous, they must also be monotone, and addition is also monotone. Therefore the following equality holds [Abramsky and Jung 1995, Proposition 2.1.12].

$$= \sup_{x \in D} \sup_{y \in D} (\sum_{j \in I'} f_j(x)) + f_i(y)$$

By continuity of the semiring:

$$= (\sup_{x \in D} \sum_{j \in I'} f_j(x)) + (\sup_{y \in D} f_i(y))$$

By continuity of $f_i$ and the induction hypothesis:

$$= \sum_{j \in I'} f_j(\sup D) + f_i(\sup D)$$
$$= \sum_{i \in I} f_i(\sup D)$$

▹ **Limit case:** suppose that the claim holds for all finite index sets. Now, given the definition of the sum operator:

$$\sup_{x \in D} \sum_{i \in I} f_i(x) = \sup_{x \in D} \sup_{J \subseteq_{\text{fin}} I} \sum_{j \in J} f_j(x)$$

The finite subsets of $I$ are a directed set and clearly the inner sum is monotone in $x$ and $J$, so we can rearrange the suprema [Abramsky and Jung 1995, Proposition 2.1.12].

$$= \sup_{J \subseteq_{\text{fin}} I} \sup_{x \in D} \sum_{j \in J} f_j(x)$$

By the induction hypothesis:

$$= \sup_{J \subseteq_{\text{fin}} I} \sum_{j \in J} f_j(\sup D)$$
$$= \sum_{i \in I} f_i(\sup D)$$

□

Lemma A.2. *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.3. *For any $m \in \mathcal{W}(X)$ and $f : X \to \mathcal{W}(Y)$, we get that $f^\dagger : \mathcal{W}(X) \to \mathcal{W}(Y)$ is a total function.*

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

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

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 $|m| =$

$\sum_{x \in \mathrm{supp}(m)} m(x)$ must be defined. By Lemma A.2, the sum in the definition of $(-)^\dagger$ is therefore defined. Now, we need to show that $|f^\dagger(m)|$ is defined:

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

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.2. □

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

Lemma A.4. *$(-)^\dagger \colon (X \to \mathcal{W}(Y)) \to (\mathcal{W}(X) \to \mathcal{W}(Y))$ 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. For any $m$, we have:

$$\begin{aligned} (\sup_{f \in D} f^\dagger)(m) &= \sup_{f \in D} f^\dagger(m) \\ &= \sup_{f \in D} \sum_{x \in \mathrm{supp}(m)} m(x) \cdot f(x) \end{aligned}$$

By Lemma A.1, using the property we just proved.

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

□

Lemma A.5. *Let $\Phi_{\langle C,e,e' \rangle}(f)(\sigma) = [\![e]\!](\sigma) \cdot f^\dagger([\![C]\!](\sigma)) + [\![e']\!](\sigma) \cdot \eta(\sigma)$ and suppose that it is a total function, then $\Phi_{\langle C,e,e' \rangle}$ is Scott continuous with respect to the pointwise order: $f_1 \sqsubseteq^\bullet f_2$ 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:

$$\begin{aligned} &\sup_{f \in D} \Phi_{\langle C,e,e' \rangle}(f)(\sigma) \\ &= \sup_{f \in D} \left( [\![e]\!](\sigma) \cdot f^\dagger([\![C]\!](\sigma)) + [\![e']\!](\sigma) \cdot \eta(\sigma) \right) \end{aligned}$$

By the continuity of + and ·, we can move the supremum up to the $(-)^\dagger$, which is the only term that depends on $f$.

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

By Lemma A.4.

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

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

Now, given Lemma A.5 and the Kleene fixed point theorem, we know that the least fixed point is defined and is equal to:

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

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

If a partial semiring is used to interpret the language semantics, then 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.6 (Compatibility).* The expressions $e_1$ and $e_2$ are compatible in semiring $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ if $[\![e_1]\!](\sigma) + [\![e_2]\!](\sigma)$ is defined for any $\sigma \in \Sigma$.

The nondeterministic (Examples 2.7 and 2.9) and tropical (Example 2.11) 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.7. *If $e_1$ and $e_2$ are compatible, then $[\![e_1]\!](\sigma) \cdot m_1 + [\![e_2]\!](\sigma) \cdot m_2$ is defined for any $m_1$ and $m_2$.*

Proof. Since $e_1$ and $e_2$ are compatible, then $[\![e_1]\!](\sigma) + [\![e_2]\!](\sigma)$ is defined. By Lemma A.2, that also means that $[\![e_1]\!](\sigma) \cdot |m_1| + [\![e_2]\!](\sigma) \cdot |m_2|$ is defined too. Now, we have:

$$\begin{aligned}
&[\![e_1]\!](\sigma) \cdot |m_1| + [\![e_2]\!](\sigma) \cdot |m_2| \\
&= [\![e_1]\!](\sigma) \cdot \sum_{\tau \in \mathrm{supp}(m_1)} m_1(\tau) + [\![e_2]\!](\sigma) \cdot \sum_{\tau \in \mathrm{supp}(m_2)} m_2(\tau) \\
&= \sum_{\tau \in \mathrm{supp}(m_1)} [\![e_1]\!](\sigma) \cdot m_1(\tau) + \sum_{\tau \in \mathrm{supp}(m_2)} [\![e_2]\!](\sigma) \cdot m_2(\tau) \\
&= \sum_{\tau \in \mathrm{supp}(m_1) \cup \mathrm{supp}(m_2)} [\![e_1]\!](\sigma) \cdot m_1(\tau) + [\![e_2]\!](\sigma) \cdot m_2(\tau) \\
&= |\, [\![e_1]\!](\sigma) \cdot m_1 + [\![e_2]\!](\sigma) \cdot m_2|
\end{aligned}$$

Therefore $[\![e_1]\!](\sigma) \cdot m_1 + [\![e_2]\!](\sigma) \cdot m_2$ must be well-defined. □

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

Lemma A.8. *If $e_1$ and $e_2$ are compatible and $[\![C_1]\!]$ and $[\![C_2]\!]$ are total functions, then:*

$$[\![(\textbf{assume}\ e_1 \fatsemi C_1) + (\textbf{assume}\ e_2 \fatsemi C_2)]\!]$$

*is a total function.*

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

$$\begin{aligned}
[\![(\textbf{assume}\ e_1 \fatsemi C_1) + (\textbf{assume}\ e_2 \fatsemi C_2)]\!](\sigma) &= [\![C_1]\!]^\dagger([\![\textbf{assume}\ e_1]\!](\sigma)) + [\![C_2]\!]^\dagger([\![\textbf{assume}\ e_2]\!](\sigma)) \\
&= [\![C_1]\!]^\dagger([\![e_1]\!](\sigma) \cdot \eta(\sigma)) + [\![C_2]\!]^\dagger([\![e_2]\!](\sigma) \cdot \eta(\sigma)) \\
&= [\![e_1]\!](\sigma) \cdot [\![C_1]\!]^\dagger(\eta(\sigma)) + [\![e_2]\!](\sigma) \cdot [\![C_2]\!]^\dagger(\eta(\sigma)) \\
&= [\![e_1]\!](\sigma) \cdot [\![C_1]\!](\sigma) + [\![e_2]\!](\sigma) \cdot [\![C_2]\!](\sigma)
\end{aligned}$$

By Lemma A.7, 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.9. *If $e$ and $e'$ are compatible and $[\![C]\!]$ is a total function, then $[\![C^{\langle e,\, e'\rangle}]\!]$ is a total function.*

Proof. Let $\Phi_{\langle C,e,e'\rangle}(f)(\sigma) = [\![e]\!](\sigma) \cdot f^\dagger([\![C]\!](\sigma)) + [\![e']\!](\sigma) \cdot \eta(\sigma)$. Since $e$ and $e'$ are compatible, it follows from Lemma A.7 that $\Phi_{\langle C,e,e'\rangle}$ is a total function. By Lemma A.5, we therefore also know that $[\![C^{\langle e,\, e'\rangle}]\!]$ 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 $\sigma$, one of $[\![b]\!](\sigma)$ or $[\![\neg b]\!](\sigma)$ 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 2.4 are well-defined in all interpretations.

In the probabilistic interpretation (Example 2.10), 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 Soundness and Completeness of Outcome Logic

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

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

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

Note that this can generally be considered equivalent to $\varphi \Rightarrow \Box(e = u)$, however we did not assume that equality is in the set of primitive tests. 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:

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

We now present the soundness proof, following the sketch from Section 3.4. The first results pertain to the semantics of iteration. We start by recalling the characteristic function:

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

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

Lemma B.2. *For all $n \in \mathbb{N}$:*

$$\Phi^{n+1}_{\langle C,e,e' \rangle}(\lambda x.\mathbb{0}) = \sum_{k=0}^{n} \left[\!\left[(\mathbf{assume}\ e \fatsemi C)^k \fatsemi \mathbf{assume}\ e'\right]\!\right]$$

Proof. By mathematical induction on $n$.

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

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

- Inductive step, suppose the claim holds for $n$. Now, for all $\sigma \in \Sigma$:

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

By the induction hypothesis

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

□

Lemma 3.4. *The following equation holds:*

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

Proof. First, by the Kleene fixed point theorem and the program semantics (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 B.2:

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

By the definition of infinite sums:

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

□

Theorem 3.3 (Soundness).

$$\Omega \vdash \langle\varphi\rangle\ C\ \langle\psi\rangle \qquad \Longrightarrow \qquad \vDash \langle\varphi\rangle\ C\ \langle\psi\rangle$$

Proof. The triple $\langle\varphi\rangle\ C\ \langle\psi\rangle$ is proven using inference rules from Figure 3, or by applying an axiom in $\Omega$. If the last step is using an axiom, then the proof is trivial since we assumed that all the axioms in $\Omega$ 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 $\vDash \langle\varphi\rangle\ \mathbf{skip}\ \langle\varphi\rangle$. Suppose that $m \vDash \varphi$. Since $[\![\mathbf{skip}]\!]^\dagger(m) = m$, then clearly $[\![\mathbf{skip}]\!]^\dagger(m) \vDash \varphi$.
- 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 $\Omega$. 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 $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) = ([\![C_2]\!]^\dagger \circ [\![C_1]\!])^\dagger(m) = [\![C_1 \fatsemi C_2]\!]^\dagger(m)$, we are done.
- Plus. Given $\Omega \vdash \langle\varphi\rangle\ C_1\ \langle\psi_1\rangle$ and $\Omega \vdash \langle\varphi\rangle\ C_2\ \langle\psi_2\rangle$, we need to show $\vDash \langle\varphi\rangle\ C_1 + C_2\ \langle\psi_1 \oplus \psi_2\rangle$. 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 2.4 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 $\vDash \langle\varphi\rangle\ \mathbf{assume}\ e\ \langle\varphi \odot u\rangle$. Suppose $m \vDash \varphi$. Since $\varphi \vDash e = u$, then $[\![e]\!](\sigma) = u$ for all $\sigma \in \mathrm{supp}(m)$. This means that:

$$\begin{aligned}
[\![\mathbf{assume}\ e]\!]^\dagger(m) &= \sum_{\sigma\in\mathrm{supp}(m)} m(\sigma) \cdot [\![\mathbf{assume}\ e]\!](\sigma) \\
&= \sum_{\sigma\in\mathrm{supp}(m)} m(\sigma) \cdot [\![e]\!](\sigma) \cdot \eta(\sigma) \\
&= \sum_{\sigma\in\mathrm{supp}(m)} m(\sigma) \cdot u \cdot \eta(\sigma)
\end{aligned}$$

$$= (\sum_{\sigma \in \mathrm{supp}(m)} m(\sigma) \cdot \eta(\sigma)) \cdot u$$
$$= m \cdot u$$

And by definition, $m \cdot u \vDash \varphi \odot u$, so we are done.

- ▹ Iter. We know that $\vDash \langle \varphi_n \rangle$ **assume** $e \fatsemi C \langle \varphi_{n+1} \rangle$ and that $\vDash \langle \varphi_n \rangle$ **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 $n \in \mathbb{N}$:

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

and

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

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:

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

Finally, by Lemma 3.4 we get that $[\![C^{\langle e,\, e' \rangle}]\!]^\dagger(m) \vDash \psi_\infty$.

- ▹ 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 $\vDash \langle u \odot \varphi \rangle\ C\ \langle u \odot \psi \rangle$. Suppose $m \vDash u \odot \varphi$. So 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 $\odot$, we get that $u \cdot [\![C]\!]^\dagger(m) \vDash u \odot \psi$.
- ▹ 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 $m \vDash \varphi_1$. 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, $\vDash \langle \phi(t) \rangle\ C\ \langle \phi'(t) \rangle$ for all $t \in T$, and we need to show that $\vDash \langle \bigoplus_{x \in T} \phi(x) \rangle\ C\ \langle \bigoplus_{x \in T} \phi'(x) \rangle$. Suppose $m \vDash \bigoplus_{x \in T} \phi(x)$, so for each $t \in T$ there is an $m_t$ such that $m_t \in \phi(t)$ and $m = \sum_{t \in T} m_t$. By the induction hypothesis, we get that $[\![C]\!]^\dagger(m_t) \in \phi'(t)$ for all $t \in T$. Now, we have:

$$\begin{aligned}
\sum_{t \in T} [\![C]\!]^\dagger(m_t) &= \sum_{t \in T} \sum_{\sigma \in \mathrm{supp}(m_t)} m_t(\sigma) \cdot [\![C]\!](\sigma) \\
&= \sum_{\sigma \in \mathrm{supp}(m)} (\sum_{t \in T} m_t(\sigma)) \cdot [\![C]\!](\sigma) \\
&= \sum_{\sigma \in \mathrm{supp}(m)} m(\sigma) \cdot [\![C]\!](\sigma) \\
&= [\![C]\!]^\dagger(m)
\end{aligned}$$

Therefore, we get that $[\![C]\!]^\dagger(m) \in \bigoplus_{x \in T} \phi'(x)$.

- ▹ 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 $\varphi' \implies \varphi$ and $\psi \implies \psi'$ 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 3.6.

$$\Omega \vdash \langle \varphi \rangle\ C\ \langle \mathsf{post}(C, \varphi) \rangle$$

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

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

$$\frac{}{\langle \varphi \rangle\ \mathbf{skip}\ \langle \varphi \rangle}\ \textsc{Skip}$$

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

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

  Now, by the induction hypothesis, we know that:

$$\begin{aligned}
&\Omega \vdash \langle \varphi \rangle\ C_1\ \langle \mathsf{post}(C_1, \varphi) \rangle \\
&\Omega \vdash \langle \mathsf{post}(C_1, \varphi) \rangle\ C_2\ \langle \mathsf{post}(C_1 \fatsemi C_2, \varphi) \rangle
\end{aligned}$$

  Now, we complete the derivation as follows:

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

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

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

We now complete the derivation as follows:

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

▹ $C = \mathbf{assume}\ e$, 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:

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

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

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

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

$$\dfrac{\varphi \vDash u = u}{\langle \varphi \rangle\ \mathbf{assume}\ u\ \langle \varphi \odot u\rangle}\ \textsc{Assume}$$

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

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

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 $m_n = [\![(\mathbf{assume}\ e \fatsemi C)^n \fatsemi \mathbf{assume}\ e']\!]^\dagger(m)$. Therefore by Lemma 3.4 we get that $\sum_{n \in \mathbb{N}} m_n = [\![C^{\langle e,\, e' \rangle}]\!]^\dagger(m)$, and therefore $\sum_{n \in \mathbb{N}} m_n \vDash \psi_\infty(m)$. We now complete the derivation as follows:

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

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

□

## C Variables and State

We now give additional definitions and proofs from Section 4. 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.

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

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 C.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 $C$:

- $C = \textbf{skip}$. 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 $\Box P \oplus \Box P \Leftrightarrow \Box P$.
- $C = \textbf{assume}\ e$. Since $\textbf{assume}\ e$ 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.

- $C = x := E$. We know that $x \notin \text{free}(P)$, so for all $s \in P$ and $v \in \text{Val}$, 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 $s[x \mapsto [\![E]\!]_{\text{Exp}}(s)] \in P$, which also means that:

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

Now suppose that $s \in P$, then clearly $s[x \mapsto [\![E]\!]_{\text{Exp}}(s)] \in P$, 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 4.1 (Soundness and Completeness).

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

Proof.

($\Rightarrow$) Suppose $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$. By Theorem 3.7, we already know that this triple is derivable for all commands other than assignment so it suffices to show the case where $C = x := E$. Now suppose $\vDash \langle \varphi \rangle\ x := E\ \langle \psi \rangle$. 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:

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

($\Leftarrow$) 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 3.3.

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

□

## D Subsumption of Program Logics

In this section, we provide proofs for the theorems in Section 5.1. Note that the following two theorems assume a nondeterministic interpretation of Outcome Logic, using the semiring defined in Example 2.7.

Theorem 5.1 (Subsumption of Hoare Logic).

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

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 [Pratt 1976].

($\Rightarrow$) Suppose $\sigma \in P$, then $\eta(\sigma) \vDash \lceil P \rceil$ and since $\vDash \langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle$ we get that $[\![C]\!]^\dagger(\eta(\sigma)) \vDash \Box Q$, which is equivalent to $[\![C]\!](\sigma) \vDash \exists u : U. \lceil Q \rceil^{(u)}$. This means that $\text{supp}([\![C]\!](\sigma)) \subseteq Q$, 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 $|m| = \mathbb{1}$ 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:

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

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

□

Theorem 5.2 (Subsumption of Lisbon Logic).

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

Proof. We only prove that $\vDash \langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle$ iff $P \subseteq \langle C \rangle Q$, since $P \subseteq \langle C \rangle Q$ iff $\vDash \{\!|P|\!\}\ C\ \{\!|Q|\!\}$ follows by definition [Möller et al. 2021; Zilberstein et al. 2023].

($\Rightarrow$) Suppose $\sigma \in P$, then $\eta(\sigma) \vDash \lceil P \rceil$ and since $\vDash \langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle$ we get that $\llbracket C \rrbracket^\dagger(\eta(\sigma)) \vDash \Diamond Q$, which is equivalent to saying that there exists a $\tau \in \mathrm{supp}(\llbracket C \rrbracket(\sigma))$ 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 $m \vDash \lceil P \rceil$, so $|m| = \mathbb{1}$ 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 $\mathrm{supp}(\llbracket C \rrbracket^\dagger(m)) = \bigcup_{\sigma \in \mathrm{supp}(m)} \mathrm{supp}(\llbracket C \rrbracket(\sigma))$, 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 $\vDash \langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle$.

□

## E Derived Rules

### E.1 Sequencing in Hoare and Lisbon Logic

We first prove the results about sequencing Hoare Logic encodings.

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

$$\frac{\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle}{\langle \Box P \rangle\ C\ \langle \Box Q \rangle}$$

Proof. We first establish two logical consequences. First, we have:

$$\begin{aligned} \Box P &\implies \exists m : \Box P.\ \mathbf{1}_m \\ &\implies \exists m : \Box P.\ \textstyle\bigoplus_{\sigma \in \mathrm{supp}(m)} m(\sigma) \odot \mathbf{1}_{\eta(\sigma)} \\ &\implies \exists m : \Box P.\ \textstyle\bigoplus_{\sigma \in \mathrm{supp}(m)} m(\sigma) \odot \lceil P \rceil \end{aligned}$$

And also:

$$\begin{aligned} \exists m : \Box P.\ \textstyle\bigoplus_{\sigma \in \mathrm{supp}(m)} m(\sigma) \odot \Box Q &\implies \exists m : \Box P.\ \textstyle\bigoplus_{\sigma \in \mathrm{supp}(m)} \Box Q \\ &\implies \exists m : \Box P.\ \Box Q \\ &\implies \Box Q \end{aligned}$$

We now complete the derivation.

$$\dfrac{\dfrac{\forall m \in \Box P.\quad \dfrac{\forall \sigma \in \mathsf{supp}(m).\quad \dfrac{\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle}{\langle m(\sigma) \odot \lceil P \rceil \rangle\ C\ \langle m(\sigma) \odot \Box Q \rangle}\ \textsc{Scale}}{\langle \bigoplus_{\sigma \in \mathsf{supp}(m)} m(\sigma) \odot \lceil P \rceil \rangle\ C\ \langle \bigoplus_{\sigma \in \mathsf{supp}(m)} m(\sigma) \odot \Box Q \rangle}\ \textsc{Choice}}{\langle \exists m : \Box P.\ \bigoplus_{\sigma \in \mathsf{supp}(m)} m(\sigma) \odot \lceil P \rceil \rangle\ C\ \langle \exists m : \Box P.\ \bigoplus_{\sigma \in \mathsf{supp}(m)} m(\sigma) \odot \Box Q \rangle}\ \textsc{Exists}}{\langle \Box P \rangle\ C\ \langle \Box Q \rangle}\ \textsc{Consequence}$$

□

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

$$\dfrac{\langle \lceil P \rceil \rangle\ C_1\ \langle \Box Q \rangle \qquad \langle \lceil Q \rceil \rangle\ C_2\ \langle \Box R \rangle}{\langle \lceil P \rceil \rangle\ C_1 \fatsemi C_2\ \langle \Box R \rangle}$$

Proof.

$$\dfrac{\langle \lceil P \rceil \rangle\ C_1\ \langle \Box Q \rangle \qquad \dfrac{\langle \lceil Q \rceil \rangle\ C_2\ \langle \Box R \rangle}{\langle \Box Q \rangle\ C_2\ \langle \Box R \rangle}\ \textsc{Lemma E.1}}{\langle \lceil P \rceil \rangle\ C_1 \fatsemi C_2\ \langle \Box R \rangle}\ \textsc{Seq}$$

□

Now, we turn to Lisbon Logic

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

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

Proof. First, for each $m \in \Diamond P$, let $\sigma_m$ be an arbitrary state such that $\sigma_m \in \mathsf{supp}(m)$ and $\sigma_m \in P$. This state must exists given that $m \in \Diamond P$. We also let $u_m = m(\sigma_m)$ and note that $u_m \neq \mathbb{0}$. We therefore have the following consequences:

$$\begin{aligned} \Diamond P &\implies \exists m : \Diamond P.\ \mathbf{1}_m \\ &\implies \exists m : \Diamond P.\ (u_m \odot \mathbf{1}_{\eta(\sigma_m)}) \oplus \top \\ &\implies \exists m : \Diamond P.\ (u_m \odot \lceil P \rceil) \oplus \top \end{aligned}$$

$$\begin{aligned} \exists m : \Diamond P.\ (u_m \odot \Diamond Q) \oplus \top &\implies \exists m : \Diamond P.\ \Diamond Q \oplus \top \\ &\implies \exists m : \Diamond P.\ \Diamond Q \\ &\implies \Diamond Q \end{aligned}$$

We now complete the derivation as follows:

$$\dfrac{\dfrac{\forall m \in \Diamond P.\quad \dfrac{\dfrac{\langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle}{\langle u_m \odot \lceil P \rceil \rangle\ C\ \langle u_m \odot \Diamond Q \rangle}\ \textsc{Scale} \qquad \dfrac{}{\langle \top \rangle\ C\ \langle \top \rangle}\ \textsc{True}}{\langle (u_m \odot \lceil P \rceil) \oplus \top \rangle\ C\ \langle (u_m \odot \Diamond Q) \oplus \top \rangle}\ \textsc{Choice}}{\langle \exists m : \Diamond P.\ (u_m \odot \lceil P \rceil) \oplus \top \rangle\ C\ \langle \exists m : \Diamond P.\ (u_m \odot \Diamond Q) \oplus \top \rangle}\ \textsc{Exists}}{\langle \Diamond P \rangle\ C\ \langle \Diamond Q \rangle}\ \textsc{Consequence}$$

□

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

$$\frac{\langle \lceil P \rceil \rangle\ C_1\ \langle \diamond Q \rangle \qquad \langle \lceil Q \rceil \rangle\ C_2\ \langle \diamond R \rangle}{\langle \lceil P \rceil \rangle\ C_1 \fatsemi C_2\ \langle \diamond R \rangle}$$

Proof.

$$\dfrac{\langle \lceil P \rceil \rangle\ C_1\ \langle \diamond Q \rangle \qquad \dfrac{\langle \lceil Q \rceil \rangle\ C_2\ \langle \diamond R \rangle}{\langle \diamond Q \rangle\ C_2\ \langle \diamond R \rangle}\ \textsc{Lemma E.3}}{\langle \lceil P \rceil \rangle\ C_1 \fatsemi C_2\ \langle \diamond R \rangle}\ \textsc{Seq}$$

□

## E.2 If Statements and While Loops

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

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

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:

$$\dfrac{\dfrac{\dfrac{\varphi_1 \vDash b = \mathbb{1}}{\langle \varphi_1 \rangle\ \mathbf{assume}\ b\ \langle \varphi_1 \odot \mathbb{1} \rangle}\ \textsc{Assume} \qquad \dfrac{\varphi_2 \vDash b = \mathbb{0}}{\langle \varphi_2 \rangle\ \mathbf{assume}\ b\ \langle \varphi_2 \odot \mathbb{0} \rangle}\ \textsc{Assume}}{\langle \varphi_1 \oplus \varphi_2 \rangle\ \mathbf{assume}\ b\ \langle \varphi_1 \rangle}\ \textsc{Split} \qquad \langle \varphi_1 \rangle\ C_1\ \langle \psi_1 \rangle}{\langle \varphi_1 \oplus \varphi_2 \rangle\ \mathbf{assume}\ b \fatsemi C_1\ \langle \psi_1 \rangle}\ \textsc{Seq}$$

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

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

□

Lemma E.6. *The following inference is derivable:*

$$\frac{\varphi \vDash b \qquad \langle \varphi \rangle\ C_1\ \langle \psi \rangle}{\langle \varphi \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \psi \rangle}\ \textit{If1}$$

Proof.

$$\dfrac{\dfrac{\varphi \vDash b \qquad \langle \varphi \rangle\ C_1\ \langle \psi \rangle \qquad \mathbb{0} \odot \top \vDash \neg b \qquad \dfrac{\dfrac{}{\langle \top \rangle\ C_2\ \langle \top \rangle}\ \textsc{True}}{\langle \mathbb{0} \odot \top \rangle\ C_2\ \langle \mathbb{0} \odot \top \rangle}\ \textsc{Scale}}{\langle \varphi \oplus (\mathbb{0} \odot \top) \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \psi \oplus (\mathbb{0} \odot \top) \rangle}\ \textsc{If}}{\langle \varphi \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \psi \rangle}\ \textsc{Consequence}$$

□

Lemma E.7. *The following inference is derivable:*

$$\frac{\varphi \vDash \neg b \qquad \langle \varphi \rangle\ C_2\ \langle \psi \rangle}{\langle \varphi \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \psi \rangle}\ \textit{If2}$$

Proof.

$$\dfrac{\dfrac{\mathbb{0} \odot \top \vDash b \qquad \dfrac{\dfrac{}{\langle \top \rangle\ C_1\ \langle \top \rangle}\ \textsc{True}}{\langle \mathbb{0} \odot \top \rangle\ C_1\ \langle \mathbb{0} \odot \top \rangle}\ \textsc{Scale} \qquad \varphi \vDash \neg b \qquad \langle \varphi \rangle\ C_2\ \langle \psi \rangle}{\langle \varphi \oplus (\mathbb{0} \odot \top) \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \psi \oplus (\mathbb{0} \odot \top) \rangle}\ \textsc{If}}{\langle \varphi \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \psi \rangle}\ \textsc{Consequence}$$

□

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

$$\frac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \Box Q \rangle \qquad \langle \lceil P \wedge \neg b \rceil \rangle\ C_2\ \langle \Box Q \rangle}{\langle \lceil P \rceil \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Box Q \rangle}\ \textit{If (Hoare)}$$

Proof. The derivation is shown below:

$$\dfrac{\dfrac{\Box(P \wedge b) \vDash b \qquad \dfrac{\dfrac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \Box Q \rangle}{\langle \Box(P \wedge b) \rangle\ C_1\ \langle \Box Q \rangle}\ \textsc{Lemma E.1} \qquad \Box(P \wedge \neg b) \vDash \neg b \qquad \dfrac{\langle \lceil P \wedge \neg b \rceil \rangle\ C_1\ \langle \Box Q \rangle}{\langle \Box(P \wedge \neg b) \rangle\ C_2\ \langle \Box Q \rangle}\ \textsc{Lemma E.1}}{\langle \Box(P \wedge b) \oplus \Box(P \wedge \neg b) \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Box Q \oplus \Box Q \rangle}\ \textsc{If}}{\langle \lceil P \rceil \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Box Q \rangle}\ \textsc{Consequence}$$

□

Lemma E.9 (Lisbon Logic If Rule). *The following inference is derivable.*

$$\frac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \Diamond Q \rangle \qquad \langle \lceil P \wedge \neg b \rceil \rangle\ C_2\ \langle \Diamond Q \rangle}{\langle \lceil P \rceil \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Diamond Q \rangle}\ \textit{If (Lisbon)}$$

Proof. The derivation is shown below:

$$\dfrac{\dfrac{\dfrac{\lceil P \wedge b \rceil \vDash b \qquad \langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \Diamond Q \rangle}{\langle \lceil P \wedge b \rceil \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{If1}}{\langle \Diamond(P \wedge b) \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{Lemma E.3} \qquad \dfrac{\dfrac{\lceil P \wedge \neg b \rceil \vDash \neg b \qquad \langle \lceil P \wedge \neg b \rceil \rangle\ C_2\ \langle \Diamond Q \rangle}{\langle \lceil P \wedge \neg b \rceil \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{If2}}{\langle \Diamond(P \wedge \neg b) \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{Lemma E.3}}{\langle \Diamond(P \wedge b) \vee \Diamond(P \wedge \neg b) \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Diamond Q \vee \Diamond Q \rangle}\ \textsc{Disj}}{\langle \lceil P \rceil \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{Consequence}$$

□

Lemma E.10 (While Rule). *The following inference is derivable.*

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

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

$$\dfrac{\dfrac{\dfrac{\varphi_n \vDash b}{\langle \varphi_n \rangle\ \mathbf{assume}\ b\ \langle \varphi_n \rangle}\ \textsc{Assume} \qquad \dfrac{\psi_n \vDash \neg b}{\langle \psi_n \rangle\ \mathbf{assume}\ b\ \langle \psi_n \odot \mathbb{0} \rangle}\ \textsc{Assume}}{\langle \varphi_n \oplus \psi_n \rangle\ \mathbf{assume}\ b\ \langle \varphi_n \rangle}\ \textsc{Split} \qquad \langle \varphi_n \rangle\ C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle}{\langle \varphi_n \oplus \psi_n \rangle\ \mathbf{assume}\ b \fatsemi C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle}\ \textsc{Seq}$$

Now part (2):

$$\dfrac{\dfrac{\varphi_n \vDash b}{\langle \varphi_n \rangle\ \mathbf{assume}\ \neg b\ \langle \varphi_n \odot \mathbb{0} \rangle}\ \textsc{Assume} \qquad \dfrac{\psi_n \vDash \neg b}{\langle \psi_n \rangle\ \mathbf{assume}\ \neg b\ \langle \psi_n \rangle}\ \textsc{Assume}}{\langle \varphi_n \oplus \psi_n \rangle\ \mathbf{assume}\ \neg b\ \langle \psi_n \rangle}\ \textsc{Split}$$

Finally, we complete the derivation as follows:

$$\dfrac{\forall n \in \mathbb{N}.\quad \dfrac{(1)}{\langle \varphi_n \oplus \psi_n \rangle\ \mathbf{assume}\ b \fatsemi C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle} \qquad \dfrac{(2)}{\langle \varphi_n \oplus \psi_n \rangle\ \mathbf{assume}\ \neg b\ \langle \psi_n \rangle}}{\langle \varphi_0 \oplus \psi_0 \rangle\ \mathbf{while}\ b\ \mathbf{do}\ C\ \langle \psi_\infty \rangle}\ \textsc{Iter}$$

□

## E.3 Loop Invariants

Lemma E.11 (Loop Invariant Rule). *The following inference is derivable.*

$$\frac{\langle \lceil P \wedge b \rceil \rangle\ C\ \langle \Box P \rangle}{\langle \lceil P \rceil \rangle\ \mathbf{while}\ b\ \mathbf{do}\ C\ \langle \Box(P \wedge \neg b) \rangle}\ \textit{Invariant}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle \lceil P \wedge b \rceil \rangle\ C\ \langle \Box P \rangle}{\langle \Box(P \wedge b) \rangle\ C\ \langle \Box P \rangle}\ \textsc{Lemma E.1}}{\langle \Box(P \wedge b) \rangle\ C\ \langle \Box(P \wedge b) \oplus \Box(P \wedge \neg b) \rangle}\ \textsc{Consequence}}{\langle \Box(P \wedge b) \oplus \Box(P \wedge \neg b) \rangle\ \mathbf{while}\ b\ \mathbf{do}\ C\ \langle \Box(P \wedge \neg b) \rangle}\ \textsc{While}}{\langle \lceil P \rceil \rangle\ \mathbf{while}\ b\ \mathbf{do}\ C\ \langle \Box(P \wedge \neg b) \rangle}\ \textsc{Consequence}$$

□

## E.4 Loop Variants

Lemma E.12. *The following inference is derivable.*

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

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

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

It is easy to see that $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$. Each $\psi_n$ except for $\psi_N$ and $\psi_\infty$ 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 $\mathbb{0} \odot \top \vDash e = u$ 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).

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

Finally, we complete the derivation.

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

□

Lemma E.13 (Lisbon Logic Loop Variants). *The following inference is derivable.*

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

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

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

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 $\sigma \in \mathrm{supp}(m_N)$ such that $\sigma \in P_0$. By definition $(\sum_{n\in\mathbb{N}} m_n)(\sigma) \geq m_N(\sigma) > \mathbb{0}$, 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 $\varphi_N = \Diamond P_0 = \Diamond P_{N-(N-1)-1}$).

$$\dfrac{\dfrac{\dfrac{\dfrac{\lceil P_{N-n} \rceil \vDash b}{\langle \lceil P_{N-n} \rceil \rangle\ \textbf{assume}\ b\ \langle \lceil P_{N-n} \rceil \rangle}\ \textsc{Assume} \qquad \langle \lceil P_{N-n} \rceil \rangle\ C\ \langle \Diamond P_{N-n-1} \rangle}{\langle \lceil P_{N-n} \rceil \rangle\ \textbf{assume}\ b \fatsemi C\ \langle \Diamond P_{N-n-1} \rangle}\ \textsc{Seq}}{\langle \Diamond P_{N-n} \rangle\ \textbf{assume}\ b \fatsemi C\ \langle \Diamond P_{N-n-1} \rangle}\ \textsc{Lemma E.3}}{\langle \varphi_n \rangle\ \textbf{assume}\ b \fatsemi C\ \langle \varphi_{n+1} \rangle}$$

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

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

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, $n \neq N$, so $\psi_n = \top$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\lceil P_0 \rceil \vDash \neg b}{\langle \lceil P_0 \rceil \rangle\ \mathbf{assume}\ \neg b\ \langle \lceil P_0 \rceil \rangle}\ \textsc{Assume}}{\langle \lceil P_0 \rceil \rangle\ \mathbf{assume}\ \neg b\ \langle \Diamond P_0 \rangle}\ \textsc{Consequence}}{\langle \Diamond P_0 \rangle\ \mathbf{assume}\ \neg b\ \langle \Diamond P_0 \rangle}\ \textsc{Lemma E.3}}{\langle \varphi_n \rangle\ \mathbf{assume}\ \neg b\ \langle \psi_n \rangle} \qquad \dfrac{\dfrac{}{\langle \varphi_n \rangle\ \mathbf{assume}\ \neg b\ \langle \top \rangle}\ \textsc{True}}{\langle \varphi_n \rangle\ \mathbf{assume}\ \neg b\ \langle \psi_n \rangle}$$

Finally, we complete the derivation using the Iter rule.

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

□

## F Hyper Hoare Logic

In this section, we present the omitted proof related the derived rules in Hyper Hoare Logic from Section 5.2. First, we give the inductive definition of negation for syntactic assertions below:

$$\begin{aligned}
\neg(\varphi \wedge \psi) &\triangleq \neg\varphi \vee \neg\psi \\
\neg(\varphi \vee \psi) &\triangleq \neg\varphi \wedge \neg\psi \\
\neg(\forall x\colon \mathsf{Val}.\varphi) &\triangleq \exists x\colon \mathsf{Val}.\neg\varphi \\
\neg(\exists \langle\sigma\rangle.\varphi) &\triangleq \forall x\colon \mathsf{Val}.\neg\varphi \\
\neg(\forall \langle\sigma\rangle.\varphi) &\triangleq \exists \langle\sigma\rangle.\neg\varphi \\
\neg(\exists \langle\sigma\rangle.\varphi) &\triangleq \forall \langle\sigma\rangle.\neg\varphi \\
\neg(B) &\triangleq \neg B
\end{aligned}$$

### F.1 Variable Assignment

For convenience, we repeat the definition of the syntactic variable assignment transformation.

$$\begin{aligned}
\mathcal{A}^E_x[\varphi \wedge \psi] &\triangleq \mathcal{A}^E_x[\varphi] \wedge \mathcal{A}^E_x[\psi] \\
\mathcal{A}^E_x[\varphi \vee \psi] &\triangleq \mathcal{A}^E_x[\varphi] \vee \mathcal{A}^E_x[\psi] \\
\mathcal{A}^E_x[\forall y : T.\varphi] &\triangleq \forall y : T.\mathcal{A}^E_x[\varphi] \\
\mathcal{A}^E_x[\exists y : T.\varphi] &\triangleq \exists y :\ T.\mathcal{A}^E_x[\varphi] \\
\mathcal{A}^E_x[\forall \langle\sigma\rangle.\varphi] &\triangleq \forall \langle\sigma\rangle.\mathcal{A}^E_x[\varphi[E[\sigma]/\sigma(x)]]
\end{aligned}$$

$$\mathcal{A}_x^E[\exists\langle\sigma\rangle.\varphi] \triangleq \exists\langle\sigma\rangle.\mathcal{A}_x^E[\varphi[E[\sigma]/\sigma(x)]]$$
$$\mathcal{A}_x^E[B] \triangleq B$$

Lemma F.1.

$$\mathcal{A}_x^E[\varphi] = \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \varphi\}$$

Proof. By induction on the structure of $\varphi$.

- $\varphi = \varphi_1 \wedge \varphi_2$.

$\mathcal{A}_x^E[\varphi_1 \wedge \varphi_2] = \mathcal{A}_x^E[\varphi_1] \wedge \mathcal{A}_x^E[\varphi_2]$

By the induction hypothesis

$$= \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \varphi_1\} \wedge \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \varphi_2\}$$
$$= \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \varphi_1 \wedge \varphi_2\}$$

- $\varphi = \varphi_1 \vee \varphi_2$.

$\mathcal{A}_x^E[\varphi_1 \vee \varphi_2] = \mathcal{A}_x^E[\varphi_1] \vee \mathcal{A}_x^E[\varphi_2]$

By the induction hypothesis

$$= \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \varphi_1\} \vee \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \varphi_2\}$$
$$= \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \varphi_1 \vee \varphi_2\}$$

- $\varphi = \forall y\colon T.\, \psi$.

$$\begin{aligned}\mathcal{A}_x^E[\forall y\colon T.\, \psi] &= \forall y\colon T.\, \mathcal{A}_x^E[\psi] \\ &= \bigcup_{v \in T} \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \psi[v/y]\} \\ &= \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \bigcup_{v \in T} \psi[v/y]\} \\ &= \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \forall y\colon T.\, \psi\}\end{aligned}$$

- $\varphi = \exists y\colon T.\, \psi$.

$$\begin{aligned}\mathcal{A}_x^E[\exists y\colon T.\, \psi] &= \exists y\colon T.\, \mathcal{A}_x^E[\psi] \\ &= \bigcap_{v \in T} \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \psi[v/y]\} \\ &= \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \bigcap_{v \in T} \psi[v/y]\} \\ &= \{m \in \mathcal{W}(\mathcal{S}) \mid [\![x \coloneqq E]\!]^\dagger(m) \in \exists y\colon T.\, \psi\}\end{aligned}$$

- $\forall\langle\sigma\rangle.\varphi$.

$$\mathcal{A}_x^E[\forall\langle\sigma\rangle.\varphi] = \forall\langle\sigma\rangle.\mathcal{A}_x^E[\varphi[E[\sigma]/\sigma(x)]]$$

By the induction hypothesis.

$$= \forall\langle\sigma\rangle.\{m \mid [\![x \coloneqq E]\!]^\dagger(m) \in \varphi[E[\sigma]/\sigma(x)]\}$$
$$= \{m \mid \forall t \in \mathrm{supp}(m).\ [\![x \coloneqq E]\!]^\dagger(m) \in \varphi[E[t]/\sigma(x)][t/\sigma]\}$$

Since $m$ and $[\![x := E]\!]^\dagger(m)$ differ only in the values of $x$, we can instead quantify over the post-states, which are updated such that $t(x) = E[t]$.

$$= \{m \mid [\![x := E]\!]^\dagger(m) \in \{m' \mid \forall t \in \mathrm{supp}(m').\ m' \in \varphi[t/\sigma]\}$$
$$= \{m \mid [\![x := E]\!]^\dagger(m) \in \forall\langle\sigma\rangle.\varphi\}$$

□

## F.2 Nondeterministic Assignment

$$\begin{aligned}
\mathcal{H}_x^S[\varphi \wedge \psi] &\triangleq \mathcal{H}_x^S[\varphi] \wedge \mathcal{H}_x^S[\psi] \\
\mathcal{H}_x^S[\varphi \vee \psi] &\triangleq \mathcal{H}_x^S[\varphi] \vee \mathcal{H}_x^S[\psi] \\
\mathcal{H}_x^S[\forall y\colon T.\varphi] &\triangleq \forall y\colon T.\mathcal{H}_x^S[\varphi] \\
\mathcal{H}_x^S[\exists y\colon T.\varphi] &\triangleq \exists y\colon T.\mathcal{H}_x^S[\varphi] \\
\mathcal{H}_x^S[\forall\langle\sigma\rangle.\varphi] &\triangleq \forall\langle\sigma\rangle.\ \forall v\colon S.\ \mathcal{H}_x^S[\varphi[v/\sigma(x)]] \\
\mathcal{H}_x^S[\exists\langle\sigma\rangle.\varphi] &\triangleq \exists\langle\sigma\rangle.\ \exists v\colon S.\ \mathcal{H}_x^S[\varphi[v/\sigma(x)]] \\
\mathcal{H}_x^S[B] &\triangleq B
\end{aligned}$$

Lemma F.2. *If $m \in \mathcal{H}_x^S[\varphi]$, then $\bigoplus_{v\in S} \mathbf{1}_{[\![x:=v]\!]^\dagger(m)} \Rightarrow \varphi$*

Proof. Suppose that $m' \vDash \bigoplus_{v\in S} \mathbf{1}_{[\![x:=v]\!]^\dagger(m)}$. The proof proceeds by induction on the structure of $\varphi$.

- $\varphi = \varphi_1 \wedge \varphi_2$. We know that $m \in \mathcal{H}_x^S[\varphi_i]$ for each $i \in \{1, 2\}$, so by the induction hypothesis $m' \vDash \varphi_i$ and therefore $m' \vDash \varphi_1 \wedge \varphi_2$.
- $\varphi = \varphi_1 \vee \varphi_2$. Without loss of generality, suppose that $m \in \mathcal{H}_x^S[\varphi_1]$, so by the induction hypothesis $m' \vDash \varphi_1$ and therefore we can weaken the assertion to conclude that $m' \vDash \varphi_1 \vee \varphi_2$.
- $\varphi = \forall y : T.\ \psi$. We know that $m \in \mathcal{H}_x^S[\psi[v/y]]$ for all $v \in T$, therefore by the induction hypothesis, $m' \vDash \varphi[v/y]$ for all $v \in T$. This means that $m' \vDash \forall y\colon T.\ \varphi$.
- $\varphi = \exists y : T.\ \psi$. We know that $m \in \mathcal{H}_x^S[\psi[v/y]]$ for some $v \in T$, therefore by the induction hypothesis, $m' \vDash \varphi[v/y]$. This means that $m' \vDash \exists y\colon T.\ \varphi$.
- $\varphi = \forall\langle\sigma\rangle.\ \psi$. We know that $m \in \mathcal{H}_x^S[\psi[v/\sigma(x)][s/\sigma]]$ for all $s \in \mathrm{supp}(m)$ and $v \in S$. Therefore, by the induction hypothesis, $m' \vDash \psi[v/\sigma(x)][s/\sigma]$. Now, note that $\psi[v/\sigma(x)][s/\sigma] = \psi[s[x := v]/\sigma]$, so clearly $m' \vDash \psi[s[x := v]/\sigma]$ for each $s \in \mathrm{supp}(m)$ and $v \in S$. In addition, $\mathrm{supp}(m') = \{s[x := v] \mid s \in \mathrm{supp}(m), v \in S\}$, and so $m' \vDash \psi[t/\sigma]$ for each $t \in \mathrm{supp}(m')$, therefore $m' \vDash \forall\langle\sigma\rangle.\ \psi$.
- $\varphi = \exists\langle\sigma\rangle.\ \psi$. We know that $m \in \mathcal{H}_x^S[\psi[v/\sigma(x)][s/\sigma]]$ for some $s \in \mathrm{supp}(m)$ and $v \in S$. Therefore, by the induction hypothesis, $m' \vDash \psi[v/\sigma(x)][s/\sigma]$. Now, note that $\psi[v/\sigma(x)][s/\sigma] = \psi[s[x := v]/\sigma]$, so clearly $m' \vDash \psi[s[x := v]/\sigma]$ for some $s \in \mathrm{supp}(m)$ and $v \in S$. In addition, $\mathrm{supp}(m') = \{s[x := v] \mid s \in \mathrm{supp}(m), v \in S\}$, and so $m' \vDash \psi[t/\sigma]$ for some $t \in \mathrm{supp}(m')$, therefore $m' \vDash \exists\langle\sigma\rangle.\ \psi$.
- $\varphi = B$. If $m \in \mathcal{H}_x^S[B] = B$, then $B$ must be a tautology, so it must be the case that $m' \vDash B$.

□

Lemma F.3. *The following rule is derivable*

$$\frac{}{\langle \mathcal{H}_x^{\{a,b\}}[\varphi] \rangle\ (x := a) + (x := b)\ \langle \varphi \rangle}$$

Proof.

$$\dfrac{\dfrac{\dfrac{}{\langle \mathbf{1}_m \rangle\ x := a\ \langle \mathbf{1}_{[\![x=a]\!]^\dagger(m)} \rangle}\ \textsc{Assign} \qquad \dfrac{}{\langle \mathbf{1}_m \rangle\ x := b\ \langle \mathbf{1}_{[\![x=b]\!]^\dagger(m)} \rangle}\ \textsc{Assign}}{\langle \mathbf{1}_m \rangle\ (x := a) + (x := b)\ \langle \bigoplus_{v \in \{a,b\}} \mathbf{1}_{[\![x=v]\!]^\dagger(m)} \rangle}\ \textsc{Plus}}{\forall m \in \mathcal{H}_x^{\{a,b\}}[\varphi].\ \langle \mathbf{1}_m \rangle\ (x := a) + (x := b)\ \langle \varphi \rangle}\ \textsc{Consequence}}{\langle \mathcal{H}_x^{\{a,b\}}[\varphi] \rangle\ (x := a) + (x := b)\ \langle \varphi \rangle}\ \textsc{Exists}$$

□

Lemma F.4. *The following rule is derivable*

$$\dfrac{}{\langle \mathcal{H}_x^{\mathbb{N}}[\varphi] \rangle\ x := \bigstar\ \langle \varphi \rangle}$$

Proof. Recall that $x := \bigstar$ is syntactic sugar for $x := 0 \fatsemi (x := x+1)^\star$ and $(x := x+1)^\star$ is syntactic sugar for $(x := x+1)^{\langle \mathbb{1}, \mathbb{1} \rangle}$. We will derive this rule using the Iter rule. The first step is to select the assertion families:

$$\varphi_n \triangleq \psi_n \triangleq \mathbf{1}_{[\![x:=n]\!]^\dagger(m)} \qquad \psi_\infty \triangleq \bigoplus_{k \in \mathbb{N}} \mathbf{1}_{[\![x:=k]\!]^\dagger(m)}$$

So clearly $(\psi_n)_{n \in \mathbb{N}} \rightsquigarrow \psi_\infty$. We now complete derivation (1) below:

$$\dfrac{\dfrac{\mathbf{1}_{[\![x:=n]\!]^\dagger(m)} \vDash \mathbb{1} = \mathbb{1}}{\langle \mathbf{1}_{[\![x:=n]\!]^\dagger(m)} \rangle\ \mathbf{assume}\ \mathbb{1}\ \langle \mathbf{1}_{[\![x:=n]\!]^\dagger(m)} \rangle}\ \textsc{Assume} \qquad \dfrac{}{\langle \mathbf{1}_{[\![x:=n]\!]^\dagger(m)} \rangle\ x := x+1\ \langle \mathbf{1}_{[\![x:=n+1]\!]^\dagger(m)} \rangle}\ \textsc{Assign}}{\langle \mathbf{1}_{[\![x:=n]\!]^\dagger(m)} \rangle\ \mathbf{assume}\ \mathbb{1} \fatsemi x := x+1\ \langle \mathbf{1}_{[\![x:=n+1]\!]^\dagger(m)} \rangle}\ \textsc{Seq}}{\langle \mathbf{1}_{[\![x:=n]\!]^\dagger(m)} \rangle\ (x := x+1)^\star\ \langle \bigoplus_{k \in \mathbb{N}} \mathbf{1}_{[\![x:=k]\!]^\dagger(m)} \rangle}\ \textsc{Iter}$$

Using (1) above, we complete the derivation as follows:

$$\dfrac{\dfrac{\dfrac{}{\langle \mathbf{1}_m \rangle\ x := 0\ \langle \mathbf{1}_{[\![x:=0]\!]^\dagger(m)} \rangle}\ \textsc{Assign} \qquad (1)}{\langle \mathbf{1}_m \rangle\ x := \bigstar\ \langle \bigoplus_{v \in \mathbb{N}} \mathbf{1}_{[\![x=v]\!]^\dagger(m)} \rangle}\ \textsc{Seq}}{\forall m \in \mathcal{H}_x^{\mathbb{N}}[\varphi].\ \langle \mathbf{1}_m \rangle\ x := \bigstar\ \langle \varphi \rangle}\ \textsc{Consequence}}{\langle \mathcal{H}_x^{\mathbb{N}}[\varphi] \rangle\ x := \bigstar\ \langle \varphi \rangle}\ \textsc{Exists}$$

□

### F.3 Assume

$$\begin{aligned}
\Pi_b[\varphi \wedge \psi] &\triangleq \Pi_b[\varphi] \wedge \Pi_b[\psi] \\
\Pi_b[\varphi \vee \psi] &\triangleq \Pi_b[\varphi] \vee \Pi_b[\psi] \\
\Pi_b[\forall x\colon T.\varphi] &\triangleq \forall x\colon T.\Pi_b[\varphi] \\
\Pi_b[\exists x\colon T.\varphi] &\triangleq \exists x\colon T.\Pi_b[\varphi] \\
\Pi_b[\forall \langle \sigma \rangle.\varphi] &\triangleq \forall \langle \sigma \rangle.\ \neg b[\sigma] \vee \Pi_b[\varphi] \\
\Pi_b[\exists \langle \sigma \rangle.\varphi] &\triangleq \exists \langle \sigma \rangle.\ b[\sigma] \wedge \Pi_b[\varphi] \\
\Pi_b[B] &\triangleq B
\end{aligned}$$

Lemma F.5. *For any assertion $\varphi$ (from the syntax in Section 5.2) and test $b$, $\vdash \langle \Pi_b[\varphi] \rangle$ **assume** $b$ $\langle \varphi \rangle$*

Proof. We first establish that for any $m$, if $m \vDash \Pi_b[\varphi]$, then $m \vDash (\varphi \wedge \Box b) \oplus \Box(\neg b)$ by induction on the structure of $\varphi$.

- $\varphi = \varphi_1 \wedge \varphi_2$. We know that $m \vDash \Pi_b[\varphi_i]$ for each $i \in \{1, 2\}$. By the induction hypothesis, we get that $m \vDash (\varphi_i \wedge \Box b) \oplus \Box(\neg b)$, so $m \vDash ((\varphi_1 \wedge \Box b) \oplus \Box(\neg b)) \wedge ((\varphi_2 \wedge \Box b) \oplus \Box(\neg b))$. Given that $b$ and $\neg b$ are disjoint, we can simplify this to $m \vDash (\varphi_1 \wedge \varphi_2 \wedge \Box b) \oplus \Box(\neg b)$.
- $\varphi = \varphi_1 \vee \varphi_2$. Without loss of generality, suppose $m \vDash \Pi_b[\varphi_1]$. By the induction hypothesis, we get that $m \vDash (\varphi_1 \wedge \Box b) \oplus \Box(\neg b)$. We can weaken this to $m \vDash ((\varphi_1 \vee \varphi_2) \wedge \Box b) \oplus \Box(\neg b)$.
- $\varphi = \forall x\colon T.\varphi$. Similar to the case for conjunctions above.
- $\varphi = \exists x\colon T.\varphi$. Similar to the case for disjunctions above.
- $\varphi = \forall\langle\sigma\rangle.\varphi$. We know that $m \vDash \neg b[s] \vee \Pi_b[\varphi[s/\sigma]]$ for all $s \in \mathrm{supp}(m)$. So, for each $s$, we know that either $[\![b]\!]_{\mathrm{Test}}(s) = \mathrm{false}$, of $m \vDash \Pi_b[\varphi[s/\sigma]]$, in which case $m \vDash (\varphi[s/\sigma] \wedge \Box b) \oplus (\Box(\neg b))$ by the induction hypothesis. Since this is true for all $s$, we get $m \vDash ((\forall\langle\sigma\rangle.\varphi) \wedge \Box b) \oplus (\Box(\neg b))$.
- $\varphi = \exists\langle\sigma\rangle.\varphi$. We know that $m \vDash b[s] \wedge \Pi_b[\varphi[s/\sigma]]$ for some $s \in \mathrm{supp}(m)$. So, by the induction hypothesis $m \vDash (\varphi[s/\sigma] \wedge \Box b) \oplus \Box(\neg b)$, which we can weaken to $m \vDash ((\exists\langle\sigma\rangle.\varphi) \wedge \Box b) \oplus \Box(\neg b)$
- $\varphi = B$. Since $m \vDash B$, then $B$ must not contain any free variables, and therefore $B$ is a tautology. So we have:

$$m \vDash \top \quad \Leftrightarrow \quad m \vDash \Box b \oplus \Box(\neg b) \quad \Leftrightarrow \quad m \vDash (B \wedge \Box b) \oplus \Box(\neg b)$$

We complete the derivation as follows:

$$\dfrac{\dfrac{\dfrac{\varphi \wedge \Box b \vDash b}{\langle \varphi \wedge \Box b \rangle \textbf{ assume } b \langle \varphi \wedge \Box b \rangle}\textsc{Assume} \qquad \dfrac{\Box(\neg b) \vDash b = \mathbb{0}}{\langle \Box(\neg b) \rangle \textbf{ assume } b \langle (\Box(\neg b)) \odot \mathbb{0} \rangle}\textsc{Assume}}{\langle (\varphi \wedge \Box b) \oplus \Box(\neg b) \rangle \textbf{ assume } b \langle (\varphi \wedge \Box b) \oplus (\Box(\neg b)) \odot \mathbb{0} \rangle}\textsc{Choice}}{\langle \Pi_b[\varphi] \rangle \textbf{ assume } b \langle \varphi \rangle}\textsc{Consequence}$$

□

## G Reusing Proof Fragments

### G.1 Integer Division

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

$$\textbf{Div} \triangleq \begin{cases} q := 0 \fatsemi r := a \fatsemi \\ \textbf{while } r \geq b \textbf{ do} \\ \quad r := r - b \fatsemi \\ \quad q := q + 1 \end{cases}$$

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

$$\varphi_n \triangleq \begin{cases} \lceil q + n = \lfloor a \div b \rfloor \wedge r = (a \bmod b) + n \times b \rceil & \text{if} \quad n \leq \lfloor a \div b \rfloor \\ \bot & \text{if} \quad n > \lfloor a \div b \rfloor \end{cases}$$

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 $n > \lfloor a \div b \rfloor$, $\varphi_n = \bot$ and $\bot \vDash r \geq b$ 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 $\Longrightarrow$. In the application of the Variant rule,

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

Fig. 5. Derivation for the DIV program.

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.

### G.2 The Collatz Conjecture

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

$$
\textbf{Collatz} \triangleq \left\{
\begin{array}{l}
i \coloneqq 0 \fatsemi \\
\textbf{while}\ a \neq 1\ \textbf{do} \\
\quad b \coloneqq 2 \fatsemi \textbf{Div} \fatsemi \\
\quad \textbf{if}\ r = 0\ \textbf{then}\ a \coloneqq q\ \textbf{else}\ a \coloneqq 3 \times a + 1 \fatsemi \\
\quad i \coloneqq i + 1
\end{array}
\right.
$$

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:

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

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 $\forall k < i + 1.f^k(n) \neq 1$.

$\langle \lceil a = n \wedge n > 0 \rceil \rangle$
$i \coloneqq 0$ ⨟
$\langle \lceil a = n \wedge n > 0 \wedge i = 0 \rceil \rangle \implies$
$\langle \lceil a = f^i(n) \wedge \forall k < i. f^k(n) \neq 1 \rceil \rangle$
**while** $a \neq 1$ **do**
$\langle \lceil a = f^i(n) \wedge \forall k < i. f^k(n) \neq 1 \wedge a \neq 1 \rceil \rangle \implies$
$\langle \lceil a = f^i(n) \wedge \forall k < i+1. f^k(n) \neq 1 \rceil \rangle$
$b \coloneqq 2$ ⨟
$\langle \lceil a = f^i(n) \wedge \forall k < i+1. f^k(n) \neq 1 \wedge b = 2 \rceil \rangle$
**Div** ⨟
$\langle \lceil a = f^i(n) \wedge \forall k < i+1. f^k(n) \neq 1 \wedge b = 2 \wedge q = \lfloor a \div b \rfloor \wedge r = (a \bmod b) \rceil \rangle \implies$
$\langle \lceil a = f^i(n) \wedge \forall k < i+1. f^k(n) \neq 1 \wedge q = \lfloor f^i(n) \div 2 \rfloor \wedge r = (f^i(n) \bmod 2) \rceil \rangle$
**if** $r = 0$ **then**
$\langle \lceil a = f^i(n) \wedge \forall k < i+1. f^k(n) \neq 1 \wedge q = \lfloor f^i(n) \div 2 \rfloor \wedge r = (f^i(n) \bmod 2) \wedge r = 0 \rceil \rangle \implies$
$\langle \lceil \forall k < i+1. f^k(n) \neq 1 \wedge q = \lfloor f^i(n) \div 2 \rfloor \wedge (f^i(n) \bmod 2) = 0 \rceil \rangle$
$a \coloneqq q$
$\langle \lceil \forall k < i+1. f^k(n) \neq 1 \wedge a = \lfloor f^i(n) \div 2 \rfloor \wedge (f^i(n) \bmod 2) = 0 \rceil \rangle \implies$
$\langle \Box(a = f^{i+1}(n) \wedge \forall k < i+1. f^k(n) \neq 1) \rangle$
**else**
$\langle \lceil a = f^i(n) \wedge \forall k < i+1. f^k(n) \neq 1 \wedge q = \lfloor f^i(n) \div 2 \rfloor \wedge r = (f^i(n) \bmod 2) \wedge r \neq 0 \rceil \rangle \implies$
$\langle \lceil a = f^i(n) \wedge \forall k < i+1. f^k(n) \neq 1 \wedge (f^i(n) \bmod 2) = 1 \rceil \rangle$
$a \coloneqq 3 \times a + 1$ ⨟
$\langle \lceil a = 3 \times f^i(n) + 1 \wedge \forall k < i+1. f^k(n) \neq 1 \wedge (f^i(n) \bmod 2) = 1 \rceil \rangle \implies$
$\langle \Box(a = f^{i+1}(n) \wedge \forall k < i+1. f^k(n) \neq 1) \rangle$
$\langle \Box(a = f^{i+1}(n) \wedge \forall k < i+1. f^k(n) \neq 1) \rangle$
$i \coloneqq i + 1$
$\langle \Box(a = f^i(n) \wedge \forall k < i. f^k(n) \neq 1) \rangle$
$\langle \Box(a = f^i(n) \wedge \forall k < i. f^k(n) \neq 1 \wedge a = 1) \rangle \implies$
$\langle \Box(i = S_n) \rangle$

Fig. 6. Derivation for the COLLATZ program.

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 $\lceil P \rceil \wedge \Box Q \Leftrightarrow \lceil P \wedge Q \rceil$.

When it comes time to analyze the if statement, we use the If (Hoare) rule (Lemma E.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, $S_n$.

$$
\begin{array}{l}
\langle \lceil \mathsf{true} \rceil \rangle \\
\quad a := 0 \fatsemi \\
\langle \lceil a = 0 \rceil \rangle \\
\quad r := 0 \fatsemi \\
\langle \lceil a = 0 \wedge r = 0 \rceil \rangle \implies \\
\langle \lceil a = 0 \wedge r = 0 \bmod 2 \rceil^{(1)} \rangle \implies \\
\langle \varphi_0 \rangle \\
\left( \begin{array}{l} a := a + 1 \fatsemi \\ b := 2 \fatsemi \\ \mathbf{Div} \end{array} \right)^{\langle \frac{1}{2} \rangle} \\
\langle \psi_\infty \rangle \implies \\
\langle \lceil r = 0 \rceil \oplus_{\frac{2}{3}} \lceil r = 1 \rceil \rangle
\end{array}
\qquad
\begin{array}{l}
\langle \varphi_n \rangle \implies \\
\langle \lceil a = n \rceil^{(\frac{1}{2^n})} \rangle \\
\quad \mathbf{assume}\ \tfrac{1}{2} \fatsemi \\
\langle \lceil a = n \rceil^{(\frac{1}{2^{n+1}})} \rangle \\
\quad a := a + 1 \fatsemi \\
\langle \lceil a = n + 1 \rceil^{(\frac{1}{2^{n+1}})} \rangle \\
\quad b := 2 \fatsemi \\
\langle \lceil a = n + 1 \wedge b = 2 \rceil^{(\frac{1}{2^{n+1}})} \rangle \\
\quad \mathbf{Div} \\
\langle \lceil a = n + 1 \wedge r = a \bmod 2 \rceil^{(\frac{1}{2^{n+1}})} \rangle \implies \\
\langle \varphi_{n+1} \rangle
\end{array}
$$

Fig. 7. Left: derivation for the main body of the probabilistic looping program. Right: derivation of the probabilistic loop.

### G.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 $\frac{1}{3}$.

$$a := 0 \fatsemi r := 0 \fatsemi (a := a + 1 \fatsemi b := 2 \fatsemi \mathbf{Div})^{\langle \frac{1}{2} \rangle}$$

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

$$\varphi_n \triangleq \lceil a = n \wedge r = a \bmod 2 \rceil^{(\frac{1}{2^n})} \qquad \psi_n \triangleq \lceil a = n \wedge r = a \bmod 2 \rceil^{(\frac{1}{2^{n+1}})}$$

Additionally, let $\psi_\infty = \lceil r = 0 \rceil \oplus_{\frac{2}{3}} \lceil r = 1 \rceil$. 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 $m_n \vDash \lceil r = 0 \rceil^{(\frac{1}{2^{n+1}})}$ for all even $n$ and $m_n \vDash \lceil r = 1 \rceil^{(\frac{1}{2^{n+1}})}$ for all odd $n$. In other words, the cummulative probability mass for each $m_n$ where $n$ is even is:

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

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 $\sum_{n \in \mathbb{N}} m_n \vDash \lceil r = 0 \rceil \oplus_{\frac{2}{3}} \lceil r = 1 \rceil$. Having shown that, we complete the derivation, shown in Figure 7. Two proof obligations are generated by applying the Iter rule, the first is proven in **??**. 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 $\langle \varphi_n \rangle\ \mathbf{assume}\ \frac{1}{2}\ \langle \psi_n \rangle$, which is easily dispatched using the Assume rule.

$$
\begin{array}{l}
\langle \lceil x = 0 \wedge y = 0 \rceil \rangle \implies \\
\langle \varphi_{N+M} \rangle \implies \\
\langle \exists n : \mathbb{N}.\ \varphi_n \rangle \\
\textbf{while } x < N \vee y < M \textbf{ do} \\
\quad \langle \varphi_{n+1} \rangle \\
\quad \textbf{if } x < N \wedge y < M \textbf{ then} \\
\qquad \langle \bigoplus_{k=\max(1,n+1-M)}^{\min(N,n)} \lceil x = N-k \wedge y = M-(n+1-k) \rceil^{\binom{N+M-(n+1)}{N-k}} \rangle \\
\qquad (x := x+1) + (y := y+1) \\
\qquad \langle \bigoplus_{k=\max(1,n+1-M)}^{\min(N,n)} \lceil x = N-k+1 \wedge y = M-(n+1-k) \rceil^{\binom{N+M-(n+1)}{N-k}} \oplus \lceil x = N-k \wedge y = M-(n-k) \rceil^{\binom{N+M-(n+1)}{N-k}} \rangle \\
\quad \textbf{else if } x \geq N \textbf{ then} \\
\qquad \langle \bigoplus_{k=\max(0,n+1-M)}^{0} \lceil x = N-k \wedge y = M-(n+1-k) \rceil^{\binom{N+M-(n+1)}{N-k}} \rangle \\
\qquad y := y+1 \\
\qquad \langle \bigoplus_{k=\max(0,n+1-M)}^{0} \lceil x = N-k \wedge y = M-(n-k) \rceil^{\binom{N+M-(n+1)}{N-k}} \rangle \\
\quad \textbf{else} \\
\qquad \langle \bigoplus_{k=n+1}^{\min(N,n+1)} \lceil x = N-k \wedge y = M-(n+1-k) \rceil^{\binom{N+M-(n+1)}{N-k}} \rangle \\
\qquad x := x+1 \\
\qquad \langle \bigoplus_{k=n+1}^{\min(N,n+1)} \lceil x = N-k+1 \wedge y = M-(n+1-k) \rceil^{\binom{N+M-(n+1)}{N-k}} \rangle \\
\quad \langle \varphi_n \rangle \\
\langle \lceil x = N \wedge y = M \rceil^{\binom{N+M}{N}} \rangle
\end{array}
$$

Fig. 8. Random walk proof

## H Graph Problems and Quantitative Analysis

### H.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 $(N, M)$.

$$
\textbf{Walk} \triangleq \left\{
\begin{array}{l}
\textbf{while } x < N \vee y < M \textbf{ do} \\
\quad \textbf{if } x < N \wedge y < M \textbf{ then} \\
\qquad (x := x+1) + (y := y+1) \\
\quad \textbf{else if } x \geq N \textbf{ then} \\
\qquad y := y+1 \\
\quad \textbf{else} \\
\qquad x := x+1
\end{array}
\right.
$$

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 7.1.

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

Recall that $n$ indicates how many steps $(x, y)$ is from $(N, M)$, so $\varphi_{N+M}$ is the precondition and $\varphi_0$ 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 $\varphi_{n+1}$ into three components, satisfying $n < N \wedge y < M$, $n \geq N$, and $y \geq M$, respectively. The assertion $x \geq N$ is only possible if we have already taken at least $N$ steps, or in other words, if $n+1 \leq (N+M) - N = M$. Letting $k$ range from $\max(0, n+1-M)$ to 0 therefore gives us a single term $k = 0$ when $n+1 \leq M$ and an empty conjunction otherwise. A similar argument holds when $y \geq M$. 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 $x \neq N$ and $y \neq M$.

Let $P(n,k) = (x = N-k \wedge y = M-(n-k))$. 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.

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

Now, we can combine the conjunctions with like terms.

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

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

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

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

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

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:

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

And by recombining the terms, we get:

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

Which is precisely $\varphi_n$. According to the Variant rule, the final postcondition is just $\varphi_0$.

### H.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 2.11).

$$\mathbf{SP} \triangleq \left\{ \begin{array}{l} \textbf{while } pos \neq t \textbf{ do} \\ \quad next \coloneqq 1 \fatsemi \\ \quad (next \coloneqq next + 1)^{\langle next < N,\, G[pos][next] \rangle} \fatsemi \\ \quad pos \coloneqq next \fatsemi \\ \quad \textbf{assume } 1 \end{array} \right.$$

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 $\varphi_n$ represents the outcomes where the guard remains true after exactly $n$ iterations and $\psi_n$ represents the outcomes where the loop guard is false after $n$ iterations. Let $I = \{1, \ldots, N\} \setminus \{t\}$.

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

$$
\begin{array}{l}
\langle \lceil pos = s \rceil \rangle \implies \\
\langle \bigoplus_{i=1}^{N} \lceil pos = i \rceil^{(\mathrm{sp}_0^t(G,s,i))} \rangle \implies \\
\langle \varphi_0 \oplus \psi_0 \rangle \\
\textbf{while } pos \neq t \textbf{ do} \\
\quad \langle \varphi_n \rangle \implies \\
\quad \langle \bigoplus_{i \in I} \lceil pos = i \rceil^{(\mathrm{sp}_n^t(G,s,i)+n)} \rangle \\
\quad next \coloneqq 1 \fatsemi \\
\quad \langle \bigoplus_{i \in I} \lceil pos = i \wedge next = 1 \rceil^{(\mathrm{sp}_n^t(G,s,i)+n)} \rangle \\
\quad (next \coloneqq next + 1)^{\langle next < N,\ G[pos][next] \rangle} \fatsemi \\
\quad \langle \bigoplus_{j=1}^{N} \bigoplus_{i \in I} \lceil pos = i \wedge next = j \rceil^{((\mathrm{sp}_n^t(G,s,i) \wedge G[i][j])+n)} \rangle \implies \\
\quad \langle \bigoplus_{j=1}^{N} \bigoplus_{i \in I} \lceil next = j \rceil^{((\mathrm{sp}_n^t(G,s,i) \wedge G[i][j])+n)} \rangle \\
\quad pos \coloneqq next \fatsemi \\
\quad \langle \bigoplus_{j=1}^{N} \bigoplus_{i \in I} \lceil pos = j \rceil^{((\mathrm{sp}_n^t(G,s,i) \wedge G[i][j])+n)} \rangle \\
\quad \textbf{assume } 1 \\
\quad \langle \bigoplus_{j=1}^{N} \bigoplus_{i \in I} \lceil pos = j \rceil^{((\mathrm{sp}_n^t(G,s,i) \wedge G[i][j])+n+1)} \rangle \implies \\
\quad \langle \bigoplus_{j=1}^{N} \lceil pos = j \rceil^{(\mathrm{sp}_{n+1}^t(G,s,j)+n+1)} \rangle \implies \\
\quad \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle \\
\langle \psi_\infty \rangle \implies \\
\langle \lceil pos = t \rceil^{(\mathrm{sp}(G,s,t))} \rangle
\end{array}
$$

Fig. 9. Shortest path proof

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

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 $|m_n| = \mathrm{sp}_n^t(G,s,t) + n$ 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 $m_n(\sigma) = \mathrm{sp}_n^t(G,s,t) + n$, and since $\mathrm{sp}_n^t(G,s,t)$ is Boolean valued and true $= 0$ and false $= \infty$, then $m_n(\sigma)$ is either $n$ or $\infty$.

By definition, the minimum $n$ for which $\mathrm{sp}_n^t(G,s,t) = \mathrm{true}$ is $\mathrm{sp}(G,s,t)$, so for all $n < \mathrm{sp}_n^t(G,s,t)$, it must be the case that $|m_n| = \infty$ and for all $n \geq \mathrm{sp}_n^t(G,s,t)$, it must be the case that $|m_n| = n$. Now, $|\sum_{n \in \mathbb{N}} m_n| = \min_{n \in \mathbb{N}} |m_n| = \mathrm{sp}_n^t(G,s,t)$, 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.

$$\vartheta_j \triangleq \begin{cases} \bigoplus_{i \in I} \lceil pos = i \wedge next = j \rceil^{(\mathrm{sp}_n^t(G,s,i)+n)} & \text{if} \quad j < N \\ \top \odot \mathbb{0} & \text{if} \quad j \geq N \end{cases}$$

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

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

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

$$\begin{array}{l} \langle \vartheta_j \rangle \implies \\ \langle \bigoplus_{i \in I} \lceil pos = i \wedge next = j \rceil^{(\mathrm{sp}^t_n(G,s,i)+n)} \rangle \\ \textbf{assume}\ next < N \fatsemi \\ \langle \bigoplus_{i \in I} \lceil pos = i \wedge next = j \rceil^{(\mathrm{sp}^t_n(G,s,i)+n)} \rangle \\ next := next + 1 \\ \langle \bigoplus_{i \in I} \lceil pos = i \wedge next = j+1 \rceil^{(\mathrm{sp}^t_n(G,s,i)+n)} \rangle \implies \\ \langle \vartheta_{j+1} \rangle \end{array}$$

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

$$\langle \vartheta_j \rangle\ \textbf{assume}\ next < N \fatsemi next := next + 1\ \langle \top \odot \mathbb{0} \rangle$$

For the second proof obligation, we must show that:

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

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

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

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