Total Outcome Logic: Proving Termination and Nontermination in Programs with Branching

James Li, Noam Zilberstein, and Alexandra Silva

Cornell University

Abstract. While there is a long tradition of reasoning about termination (and nontermination) in the context of program analysis, specialized logics are typically needed to give different termination guarantees. This includes partial correctness, where termination is not guaranteed, and total correctness, where it is guaranteed. We present *Total Outcome Logic*, a single logic which can express the full spectrum of termination conditions and program properties offered by the aforementioned logics. Total Outcome Logic extends termination and incorrectness reasoning across different kinds of branching effects, so that a single metatheory powers this reasoning in different kinds of programs, including nondeterministic and probabilistic. We demonstrate the utility of this approach through a variety of case studies.

1 Introduction

Reasoning about termination has captivated the minds of computer scientists since the inception of the field; the halting problem proved crucial in understanding what is and is not computable. Much later, termination was one of the first *computational effects* considered in the course of the development of formal techniques for program analysis, including program logics.

Hoare Logic initially sought to avoid the termination question by offering *partial correctness* specifications, which hold only *if* the program terminates [16]. Partial correctness was attractive, as it offered a simple way to describe loop behaviors via invariants while still providing *safety*—the property that the program never displays undesirable behaviors [20]. *Total correctness* Hoare Logic was later developed, which additionally guaranteed termination [25]. More recently, other program logics for proving the possibility of nontermination have been explored [32].

Although it is difficult to reason about nontermination—even in deterministic programs—the situation is complicated significantly in the presence of other *computational effects*. As an illustration, consider the combination of nontermination and nondeterminism. This particular mix of effects raises the question of whether unbounded choice is an allowable program construct. More precisely, can we define a reasonable semantics for a language that includes a command $x \coloneqq \bigstar$, which nondeterministically assigns a natural number to the variable x? If so, does following the program terminate?

$$x \coloneqq \bigstar \fatsemi \mathsf{while}\ x > 0\ \mathsf{do}\ x \coloneqq x - 1$$

The answers to these questions are quite subtle. On the one hand, we see that for any choice of n that is assigned to x, the loop will terminate after n iterations, so the program *does* terminate. On the other hand, for any number of iterations m that the loop might perform, the value chosen for x could be $m+1$, meaning that it may not terminate.

We can understand this disconnect from a domain theoretic point of view, as it arises from the fact that the semantics of the sequential composition operator in the programming language above is not Scott continuous, meaning that the behavior of the loop does not stabilize after ω iterations. In fact, Apt and Plotkin [2] proved that there is no continuous semantics that distinguishes nontermination in the presence of unbounded nondeterminism. Operationally, this corresponds to the idea that a machine cannot make infinitely many choices in a finite amount of time, so $x := \bigstar$ cannot exist as a basic program construct [12].

Curiously, the issues with unbounded choice do not arise in conjunction with other computational effects such as probabilistic choice. Termination of probabilistic programs has been studied extensively, and there are many example of probabilistic programs with infinitely many outcomes, which *almost surely terminate*—they terminate with probability 1. For example, one can write a probabilistic program that computes a geometric distribution, where the probability that $x = n$ is $\frac{1}{2^n}$ for any natural number.

In this paper, we seek to develop a *single logic* for reasoning about *termination and nontermination* across *different kinds of branching effects* such as nondeterminism and probabilistic choice. Our formalism follows the weighted programming paradigm, where the possible program end states are assigned weights [4, 37, 40]. Varying the representations of the weights gives interpretations of different effects. For example, Boolean weights indicate whether or not a state is a possible outcome of a nondeterministic program whereas real-valued weights precisely quantify the probabilities of outcomes in a probabilistic program.

Weights are typically elements of a *semiring*, which means that they can be added and multiplied, which is useful when providing semantics to the branching and sequential composition operations of programs, respectively. However, the addition of nontermination complicates the story so that the semiring weights must have additional algebraic properties.

We characterize classes of weights for which nontermination reasoning can be done in a uniform way. Nondeterministic and probabilistic programs are instances of two classes of weightings that we identify in this paper; *conservative* and *indicative*. In a conservative weighting, the weight can be viewed as a resource, with the total amount of weight being split between branches of computation and nontermination, so that the weight of infinite traces can converge to 0 when all the weight is taken up by terminating outcomes. Indicative weightings have an infinite element that cannot be reduced, so that infinite traces cannot occur with weight zero in the limit. Our full list of contributions is as follows:

- ▷ Building on the formalization of Outcome Logic [37], we define a generic weighted program semantics, which can also quantify the weights of infinite

traces (Section 2). We show how the ability of each model to make unbounded choices corresponds to particular properties of the underlying semiring.
- ▷ We extend Outcome Logic with the ability to reason about nontermination to obtain Total Outcome Logic (Section 3). Unlike prior work, this allows us to prove that programs always terminate, or sometimes do not terminate.
- ▷ We show that Total Outcome Logic subsumes Partial and Total Hoare Logic [16, 25] and other logics for incorrectness and nontermination [38, 32] (Section 3.3). We additionally derive proof rules to reason more easily in those logics in Section 4.
- ▷ We present a variety of case studies in Section 5 to demonstrate how to reason about termination and nontermination with Total Outcome Logic.

Omitted proofs are included in the appendix.

2 Weighted Program Semantics

We consider a simple *weighted* programming language in the style of Batz et al. [4] and Zilberstein [37]. We will present the syntax and semantics in this section. While the syntax is not different from the aforementioned works, the semantics will be crafted to take into account potential nontermination.

2.1 Syntax

The syntax is presented in Figure 1. The language includes the usual syntax for the program that does nothing skip, assertions $\mathsf{assume}\ e$, basic (uninterpreted) program actions $a \in \mathsf{Act}$, and sequential composition $\fatsemi$. We also include in the language a nondeterministic choice $C_1 + C_2$, which is a program that nondeterministically chooses one of the branches to execute. One difference worth highlighting is that the $\mathsf{assume}\ e$ command takes a guard e that is not just a usual Boolean expression. In this language guards can be Boolean—the expressions in BExp are built from a basic set of tests Test using the usual Boolean operations—but they can also be *weights* $u \in U$. The intuition behind $\mathsf{assume}\ u$ is that the computation trace in which this is executed is weighed by u. This will become clearer below when we discuss the formal semantics. Using Boolean tests, we can define syntactic sugar for if statements:

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

The syntax for a loop $C^{\langle e_1, e_2 \rangle}$, due to Zilberstein [37], is slightly unusual as it has two guards: e_1 is the guard for the body of the loop C to continue executing, whereas e_2 is the guard for the exit of the loop. In a typical syntax, loops usually only have Boolean guard b and the exit guard is simply the negation of b. This sort of guarded iteration can be encoded in our language as follows:

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

$$
\begin{aligned}
\mathsf{Commands} \ni C &::= \mathsf{skip} \mid \mathsf{assume}\ e \mid a \in \mathsf{Act} \mid C_1 \fatsemi C_2 \mid C_1 + C_2 \mid C^{\langle e_1, e_2 \rangle} \\
\mathsf{Guards} \ni e &::= b \mid u \in U \\
\mathsf{BExp} \ni b &::= \mathsf{true} \mid \mathsf{false} \mid b_1 \vee b_2 \mid b_1 \wedge b_2 \mid \neg b \mid t \in \mathsf{Test}
\end{aligned}
$$

Fig. 1: Syntax of a simple weighted imperative language, parametric on a set of basic program actions Act and a set weights U.

2.2 Semantics as Weighting Functions

In order to provide the semantics of our language we first need to introduce a few algebraic definitions. To make sense of the weights in the context of computation traces, we will need to have structure on the set U that enables us to combine different weights along a computation trace and across computation traces. In other words, how do we give semantics to expressions like: $(\mathsf{assume}\ 3) \fatsemi a \fatsemi (\mathsf{assume}\ 5)$ or $(\mathsf{assume}\ 3) \fatsemi a_1 + (\mathsf{assume}\ 5) \fatsemi a_2$? We will require the set of weights U to have the structure of a *semiring*, which is an algebraic structure that combine two *monoids*.

Definition 1 (Monoid). *A monoid $\langle U, +, \mathbb{0} \rangle$ consists of a carrier set U, an associative binary operation $+ : U \times U \to U$, and an identity element $\mathbb{0}$ ($u + \mathbb{0} = \mathbb{0} + u = u$). If $+ : U \rightharpoonup U$ is partial, then we say the monoid is partial. If $+$ is commutative ($u + v = v + u$), then the monoid is commutative.*

Definition 2 (Semiring). *A semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ is an algebraic structure such that the following holds:*

1. *$\langle U, +, \mathbb{0} \rangle$ is a commutative monoid.*
2. *$\langle U, \cdot, \mathbb{1} \rangle$ is a monoid.*
3. *Distributivity: $u \cdot (v + w) = uv + uw$ and $(u + v) \cdot w = uw + vw$*
4. *Annihilation: $\mathbb{0} \cdot x = x \cdot \mathbb{0} = \mathbb{0}$*

Example 1 (Semirings). The following are examples of semirings that can be used to encode different kinds of computation.

1. The Boolean semiring $\langle \mathbb{B}, \vee, \wedge, 0, 1 \rangle$ uses the Booleans $\mathbb{B} = \{0, 1\}$ as weights, which indicate whether states are possible outcomes. Addition is logical disjunction and multiplication is conjunction.
2. The probabilistic semiring $\langle [0, 1], +, \cdot, 0, 1 \rangle$ uses real-valued probabilities as weights to quantify the likelihoods of outcomes. Addition is standard arithmetic addition, but it is partial since it is undefined if the sum is above 1. Multiplication is standard arithmetic multiplication.
3. The natural number semiring $\langle \mathbb{N}^\infty, +, \cdot, 0, 1 \rangle$ uses natural numbers (plus ∞) as weights, with addition and multiplication given by the standard arithmetic operations.

For more examples, refer to Batz et al. [4] and Zilberstein [37].

$$
\begin{aligned}
[\![\mathsf{skip}]\!]\,(\sigma) &\triangleq \mathsf{unit}(\sigma) \\
[\![C_1 \fatsemi C_2]\!]\,(\sigma) &\triangleq \mathsf{bind}([\![C_1]\!]\,(\sigma), [\![C_2]\!]) \\
[\![C_1 + C_2]\!]\,(\sigma) &\triangleq [\![C_1]\!]\,(\sigma) + [\![C_2]\!]\,(\sigma) \\
[\![a]\!]\,(\sigma) &\triangleq [\![a]\!]_{\mathsf{Act}}\,(\sigma) \\
[\![\mathsf{assume}\ e]\!]\,(\sigma) &\triangleq [\![e]\!]\,(\sigma) \cdot \mathsf{unit}(\sigma) \\
\left[\!\!\left[C^{\langle e_1, e_2\rangle}\right]\!\!\right]\,(\sigma) &\triangleq \mathsf{lfp}\ .\ \left(\Phi_{C^{\langle e_1,e_2\rangle}}\right)(\sigma) \\
\text{where}\quad \Phi_{C^{\langle e_1,e_2\rangle}}\,(\sigma) &\triangleq [\![e_1]\!]\,(\sigma) \cdot \mathsf{bind}([\![C]\!]\,(\sigma), f) + [\![e_2]\!]\,(\sigma) \cdot \mathsf{unit}(\sigma)
\end{aligned}
$$

Fig. 2: Denotational semantics $[\![-]\!] : \Sigma \to \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ of program commands, parametric on a set of program states Σ; an interpretation $[\![a]\!]_{\mathsf{Act}} : \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma)$ for atomic actions $a \in \mathsf{Act}$; and with $\mathsf{Test} \subseteq 2^{\Sigma}$ as the set of primitive tests.

In order to define the semantics we will assume a set of Σ of program states and then assign to each command a map from program states to weighted, possibly nonterminating, traces. More precisely, we will build a denotational object in two steps. First, to encode information about nontermination, we expose divergence as a possible outcome of running a program. We will represent this with the element $\circlearrowleft$, where for any set S, $S_{\circlearrowleft} \triangleq S \cup \{\circlearrowleft\}$. Second, we define weighting functions, which assign a weight to all states $\sigma \in \Sigma$, and to nontermination.

Definition 3 (Weighting Function). *Given a set of program states Σ and a (partial) semiring $\mathcal{A} = \langle X, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$, the set of weighting functions is*

$$
\mathcal{W}_{\mathcal{A}}(\Sigma) \triangleq \left\{ m : \Sigma_{\circlearrowleft} \to X \;\middle|\; |m| \text{ is defined} \right\}
$$

The support $\mathsf{supp}(m) \triangleq \{\sigma \mid m(\sigma) \neq \mathbb{0}\}$ is the set of states to which m assigns nonzero weight and the mass of $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ written as $|m| \triangleq \sum_{\sigma \in \mathsf{supp}(m)} m(\sigma)$ is the sum of the weights of all the elements in the support.

We take a weighting function to represent a configuration in a computation, which may have branched into many different outcomes, including the nontermination outcome $\circlearrowleft$. Captured here are two distinct computational effects: weighted branching and nontermination. We can now assign to each program C a denotation $[\![C]\!] : \Sigma \to \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, as shown in Figure 2[1].

We left several operations underspecified in Figure 2, which we will now explain in detail, from simpler to more complex: 1. semantics of guards and branching; 2. monadic structure unit and bind that provides semantics of skip and sequential composition; 3. fixpoint semantics of loops.

Guards and Branching. Recall that guards e can be Boolean expressions b or weights $u \in U$. We define the semantics of guard expressions e as $[\![e]\!] : \Sigma \to U$:

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

[1] The distinction between $\mathcal{W}_{\mathcal{A}}(\Sigma)$ and $\mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma)$ will be explained when we discuss the semantics of loops

The semantics of Boolean guards $[\![b]\!]_{\mathsf{Test}} : \Sigma \to \{\mathbb{0}, \mathbb{1}\}$ is a simple extension the set element predicate, since primitive tests are sets of program states $\mathsf{Test} \subseteq 2^{\Sigma}$:

$$[\![\mathsf{true}]\!]_{\mathsf{Test}}(\sigma) \triangleq \mathbb{1} \qquad [\![\mathsf{false}]\!]_{\mathsf{Test}}(\sigma) \triangleq \mathbb{0} \qquad [\![t]\!]_{\mathsf{Test}}(\sigma) \triangleq \begin{cases} \mathbb{1} & \text{if } \sigma \in t \\ \mathbb{0} & \text{if } \sigma \notin t \end{cases}$$

$$[\![b_1 \vee b_2]\!]_{\mathsf{Test}}(\sigma) \triangleq [\![b_1]\!]_{\mathsf{Test}}(\sigma) \vee [\![b_2]\!]_{\mathsf{Test}}(\sigma)$$

$$[\![b_1 \wedge b_2]\!]_{\mathsf{Test}}(\sigma) \triangleq [\![b_1]\!]_{\mathsf{Test}}(\sigma) \wedge [\![b_2]\!]_{\mathsf{Test}}(\sigma) \qquad [\![\neg b]\!]_{\mathsf{Test}}(\sigma) \triangleq \neg [\![b]\!]_{\mathsf{Test}}(\sigma)$$

The semantics of $\mathsf{assume}\ e$ uses a product operation $\cdot$ on weighting functions whereas a $+$ operation is used in the definition of $[\![C_1 + C_2]\!]$. These are operations on weighting functions that are lifted from the semiring $+$ and $\cdot$ pointwise:

$$(m_1 + m_2)(\sigma) \triangleq m_1(\sigma) + m_2(\sigma) \qquad (u \cdot m)(\sigma) \triangleq u \cdot m(\sigma)$$

Monadic structure of weighting functions. We next turn to the semantics of sequential composition and skip, which is achieved using a monad [24, 28].

Definition 4 (Monads). *A Kleisli triple $\langle T, \eta, (-)^\dagger \rangle$ in the category **Set** consists of a functor $T : \textbf{\textit{Set}} \to \textbf{\textit{Set}}$, and two morphisms $\eta : \mathsf{Id} \Rightarrow T$ and, for any sets X and Y, $(-)^\dagger : (X \to T(Y)) \to T(X) \to T(Y)$ such that:*

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

If such a triple exists, then the functor T is a monad.

For any semiring $\mathcal{A}$, $\langle \mathcal{W}_{\mathcal{A}}, \eta, (-)^\dagger \rangle$ is a Kleisli triple, with functions η and $(-)^\dagger$ defined as follows:

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

Note how the Kleisli extension $f^\dagger(m)$ discriminates between weight allocated to program states (in Σ) and weight allocated to $\circlearrowleft$ by m. Intuitively, this captures the notion that any program state occupied by a weighted program can be passed along to the next step of the computation, whereas any accrued nontermination persists and must be *carried through*. Although we define the monad operations directly here, $\mathcal{W}_{\mathcal{A}}$ can also be defined via a distributive law [5] between the weighting function monad of Outcome Logic [37] and the $+1$ or *error* monad, which is known to compose with all other monads [22].

Loops. Finally, we will discuss the semantics of iterated computations, which will require $\mathcal{W}_{\mathcal{A}}$ to be a directed complete partial order (dcpo). We will define this order by comparing the weights of each outcome, however we consider the configuration m to get *larger* as the weight of nontermination gets *smaller*.

The partial order we defined can be viewed as an *approximation order* [1], so that the semantic structure gets larger as we learn more information about the

program's behavior after each iteration of the computation. As such, we initially consider the behavior to be nontermination (bottom), and the approximation gets larger as more and more terminating executions are discovered.

We use a *fusion order* where we say that $m_1 \sqsubseteq m_2$ if and only if the weight of each regular state increases and the weight of nontermination decreases. This is similar to the fixpoint maximal trace semantics of Cousot [9] in which finite traces are compared covariantly and infinite traces are compared contravariantly.

Definition 5 (Fusion Order). *We define the fusion order $\sqsubseteq$ on $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ as:*

$$m_1 \sqsubseteq m_2 \quad \textit{iff} \quad m_1(\sigma) \leq m_2(\sigma) \textit{ for all } \sigma \in \Sigma, \textit{ and } m_1(\circlearrowleft) \geq m_2(\circlearrowleft)$$

The above uses the natural order $\leq$ on $\mathcal{A}$: $u \leq v$ iff $u + w = v$ for some w. If $\langle U, \leq \rangle$ is a partial order, then the semiring is said to be *naturally-ordered*. The semiring is *bounded* if there exists a top element $\top$ such that $u \leq \top$ for all $u \in U$. This top element is sometimes required to be an infinity, as defined below.

Definition 6 (Infinity [13]). *An element w in semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ is an infinity if it is additively absorbing:*

$$\forall u \in U.\, u + w = w + u = w$$

w is a strong infinity if it is infinite as well as multiplicatively absorbing for all elements besides 0:

$$\forall x \in U \setminus \{\mathbb{0}\}.\, x \cdot w = w \cdot x = w$$

We also need to define a notion of *bounded weighting* which generalizes the notion of bounded nondeterminism in the modeling of concurrency and fairness, and is crucial for ensuring the standard continuity results for our semantics.

Definition 7 (Bounded Weighting). *We further define the following classes of weighting functions:*

- $\mathcal{W}^{+}_{\mathcal{A}}(\Sigma) \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid \mathsf{supp}(m) \textit{ is finite}\}$
- $\mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma) \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid m(\circlearrowleft) = \sup_{m' \in E(m)} m'(\circlearrowleft)\}$.
- $\mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma) \triangleq \mathcal{W}^{+}_{\mathcal{A}}(\Sigma) \cup \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$

For any particular $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, let $E(m)$ be the set of weighting functions that assign equal weight to all program states in Σ (i.e. that can only differ in the weight on nontermination $\circlearrowleft$):

$$E(m) \triangleq \{m' \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid \forall \sigma \in \Sigma.\, m'(\sigma) = m(\sigma)\}$$

A mapping $\Sigma \to \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma)$ is said to exhibit ***bounded weighting****.*

Our notion approximates the operational behavior captured by bounded nondeterminism, which stipulates that a nondeterministic program can only select between finitely many branches at each step of execution. Drawing from the definition by Back [3] for a powerdomain semantics, this means that running

a nondeterministic program must produce a set of outcomes that is either (i) finite or (ii) exhibits nontermination (such that infinite sets of outcomes must be generated by some nonterminating trace). The sets $\mathcal{W}^{+}_{\mathcal{A}}(\Sigma)$ and $\mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$ are the analogue of Back's two cases, respectively; weightings produced by programs must either:

- ▷ describe a finite set of outcomes (finite support) (that is, be in $\mathcal{W}^{+}_{\mathcal{A}}(\Sigma)$), or;
- ▷ maximize the weight on nontermination $\circlearrowleft$ (that is, be in $\mathcal{W}^{\omega}_{\mathcal{A}}(\circlearrowleft)$).

This can also be seen as a generalization of the Plotkin powerdomain, which includes all finite sets of states and infinite states that contain $\circlearrowleft$ [31, 34]. Here, the choice to *maximize* nontermination lies in our approach to tracking the presence of nontermination in weighting functions. In particular, we support two semantic paradigms and correspondingly require the semiring of weights $\mathcal{A}$ to belong to one of two classes.

1. ***Conservative Weighting***: $\mathcal{A}$ is partial and bounded with *non-idempotent* top element $\top$ such that, for all $x \in X/\{0\}$, $x + \top$ and $\top + x$ are not defined. So, only $\top + 0 = 0 + \top = \top$. In this case, we choose our weighting functions to be drawn from
$$\mathcal{D} \triangleq \{m \in \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma) \mid |m| = \top\}$$
Thus, programs under a *conservative weighting* scheme always produce a weighting function of fixed mass $\top$, and this mass is conserved across sequences of programs. Examples include probabilistic and deterministic programs, for which the mass of weighting functions is fixed at 1.
2. ***Indicative Weighting***: $\mathcal{A}$ is total and bounded with a *strongly infinite* top element $\top$. Since the semiring is total, we can rewrite
$$\mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma) \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid m(\circlearrowleft) = \top\}$$
and limit weighting functions to $\mathcal{D} \triangleq \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma)$. We refer to this as an *indicative weighting* scheme in that the weight on $\circlearrowleft$ serves as an indicator for nontermination; we assign a weight of 0 if the computation does not produce a diverging trace, and assign infinite weight otherwise.

At first, the introduction of these restrictions may seem to fetter the generalizability of our approach. However, we note that any semiring of weights $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ that does not belong to either of the above classes can be extended with a strongly infinite element [13, p. 166]. In particular, we adjoin to $\mathcal{A}$ an element $w \notin U$ to accommodate an *indicative* weighting scheme. Addition and multiplication can be defined such that

- ▷ $w + u = u + w = w$ for all $u \in U$;
- ▷ $w \cdot u = u \cdot w = w$ for all $u \in U/\{\mathbb{0}\}$;
- ▷ $w \cdot 0 = 0 \cdot w = 0$.

Doing so limits our (indicative) semantics to a coarser representation of program behavior, foregoing attempts at tracking finer weights with which a program can

diverge, and instead only maintaining whether nontermination is possible or not. Nevertheless, this is sufficient for most purposes, and in the case for some semirings, necessary to preserve continuity, which is defined below.

Definition 8 (Scott-Continuity). *A semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ with order $\leq$ is Scott-continuous if for any directed set $D \subseteq U$ (where all pairs of elements in D have a supremum):*

$$\sup_{u\in D}(u + v) = (\sup D) + v \quad \sup_{u\in D}(u \cdot v) = (\sup D) \cdot v \quad \sup_{u\in D}(v \cdot u) = v \cdot (\sup D)$$

The semiring is lower Scott-continuous if the same operations preserve infima of all lower-directed subsets).

For Scott Continuous semirings, we can define an infinite sum operation over an index set I as the supremum of sums over all finite subsets of I.

$$\sum_{i\in I} u_i = \sup \left\{ \sum_{i\in J} u_i \mid J \subseteq I \text{ finite} \right\}$$

This construction yields a *complete* semiring, meaning that the sum operator obeys the following desirable laws [19]:

1. For $I = \{i_1, \ldots, i_n\}$ finite, $\sum_{i\in I} u_i = u_{i_1} + \ldots + u_{i_n}$.
2. If $I = \bigcup_{k\in K} J_k$ is a disjoint union of nonempty sets, then $\sum_{k\in K} \sum_{j\in J_k} u_j = \sum_{i\in I} u_i$.
3. 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 $\left(\sum_{i\in I} u_i\right) \cdot v = \sum_{i\in I} u_i \cdot v$.

The above definitions are what we need to fully define the denotational semantics for our language, and in particular the semantics of loops (and the existence of the fixpoint). In particular, we require $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$ to be a naturally ordered, finitary, Scott-continuous, partial semiring bounded by top element $\top$.

3 Total Outcome Logic

We now leverage our nontermination semantics to define *Total* Outcome Logic (TOL), building atop the proof system given by Zilberstein [37] to also reason about the divergence of programs. Like its predecessor, TOL can express the modalities of Dynamic Logic, as well as the partial correctness triples of Hoare Logic and incorrectness triples of Lisbon Logic [29][2]. However, we show that TOL is strictly more expressive than Outcome Logic proper. In particular, by exposing nontermination as a program outcome, TOL is capable of expressing additional properties with termination and non-termination guarantees, including the classical notion of *total correctness*.

[2] The name Lisbon Logic is in fact due to Derek Dreyer, see historical remarks in the related work section of Zilberstein et al. [38]. The same triples first appeared in Möller et al. [29] under the name of *backwards underapproximate triples*.

3.1 Outcome Assertions

First, we define *outcome assertions*, which serve as the pre- and postconditions in TOL. In program logics, it is common to represent pre- and postconditions extensionally as *sets of program states*. Outcome assertions, on the other hand, must operate on a finer level, specifying not only a collection of states but also their weights at the respective point in the computation. We give extensional definitions of outcome assertions $\varphi, \psi \in 2^{\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})}$ as sets of weighting functions. For any $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, we write $m \vDash \varphi$ (read as m *satisfies* φ) to mean $m \in \varphi$.

Below, we introduce useful notation for basic assertions used later in constructing the logic (summarized in Figure 3). First, basic propositional artifacts can be encoded, where $\top$ (always true) is $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, $\bot$ (always false) is the empty set, and basic logical connectives are defined as usual:

$$\top \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \qquad \bot \triangleq \emptyset \qquad \neg\varphi \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})/\varphi$$
$$\varphi \vee \psi \triangleq \varphi \cup \psi \qquad \varphi \wedge \psi \triangleq \varphi \cap \psi \qquad \varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi$$

For a semantic assertion on program states $P \subseteq \Sigma$, we introduce an outcome assertion $P^{(u)}$ to mean that P covers all reachable states with a cumulative weight of u. Similarly, $\Uparrow^{(u)}$ denotes that the nontermination outcome $\circlearrowleft$ occurs with weight $u \cdot \top$.

$$P^{(u)} \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid |m| = u, \mathsf{supp}(m) \subseteq P\} \qquad \Uparrow^{(u)} \triangleq \{u \cdot \top \cdot \eta(\circlearrowleft)\}$$

For simplicity, we write P or $\Uparrow$ for $P^{(1)}$ or $\Uparrow^{(1)}$, respectively. Operations $u \odot \varphi$ and $\varphi \odot u$ scale the outcome assertion φ with weight u on the left or right:

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

Existential quantification with respect to predicate $\phi : T \to 2^{\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})}$ is defined as the union of all $\phi(t)$ for inputs $t \in T$. Then, $m \vDash \exists x : T.\, \phi(x)$ precisely when m satisfies $\phi(t)$ for *some* input t.

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

We write the *outcome conjunction* $\varphi \oplus \psi$ to consist of weighting functions m that can be split into components $m = m_1 + m_2$, with m_1 satisfying φ and m_2 satisfying ψ. We define this more generally for a potentially infinite index set T:

$$\bigoplus_{x \in T} \phi(x) \triangleq \left\{ \sum_{t \in T} m_t \mid \forall t \in T.\, m_t \in \phi(t) \right\}$$

Finally, we assert *identity* to m with $\mathbf{1}_m$, which consists of the singleton $\{m\}$:

$$\top \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \qquad \bot \triangleq \emptyset \qquad \neg\varphi \triangleq \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})/\varphi$$
$$\varphi \vee \psi \triangleq \varphi \cup \psi \qquad \varphi \wedge \psi \triangleq \varphi \cap \psi \qquad \varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi$$

$$P^{(u)} \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid |m| = u, \mathsf{supp}(m) \subseteq P\} \qquad \Uparrow^{(u)} \triangleq \{u \cdot \top \cdot \mathsf{unit}(\circlearrowleft)\}$$
$$u \odot \varphi \triangleq \{u \diamond m \mid m \in \varphi\} \qquad \varphi \odot u \triangleq \{m \cdot u \mid m \in \varphi\}$$
$$\mathbf{1}_m \triangleq \{m\} \qquad \exists x : T.\, \phi(x) \triangleq \bigcup_{t \in T} \phi(t)$$

$$\bigoplus_{x \in T} \phi(x) \triangleq \left\{ \sum_{t \in T} m_t \mid \forall t \in T.\, m_t \in \phi(t) \right\}$$

Fig. 3: Semantic definitions of outcomes assertions as subsets in $2^{\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})}$.

3.2 Outcome Triples

The basic formulae of Total Outcome Logic are *outcome triples*, which take the form $\langle\varphi\rangle\ C\ \langle\psi\rangle$ for a precondition φ, command C, and postcondition ψ.

Definition 9. *The semantics of outcome triples is defined as follows:*

$$\vDash \langle\varphi\rangle\ C\ \langle\psi\rangle \quad \textit{iff} \quad \forall m \in \varphi.\ [\![C]\!]^{\dagger}(m) \vDash \psi$$

Outcome triples are then fundamentally *over-approximate*: the triple holds iff the postcondition *is a superset of* the results attainable from running the program starting in state distributions satisfying the precondition.

3.3 Dynamic Logic, Correctness, and Incorrectness

In modal and Dynamic logics, the $\Box$ and $\Diamond$ operators describe whether an assertion *always* or *sometimes* holds, respectively. These same modalities can be encoded as outcome assertions:

$$\Box P \triangleq \exists(u, v) : U^2.\, P^{(u)} \oplus \Uparrow^{(v)} = \{m \mid \mathsf{supp}(m) \subseteq P_{\circlearrowleft}\}$$
$$\Diamond P \triangleq \exists u : U/\{\emptyset\}.\, P^{(u)} \oplus \top = \{m \mid \mathsf{supp}(m) \cap P \neq \emptyset\}$$

where $\Box$ and $\Diamond$ are De Morgan duals in the sense that: $\Diamond P = \neg\Box\neg P$ and $\Box P = \neg\Diamond\neg P$ (Lemma 9). Just as its interpretation in Dynamic Logic, $\Box P$ makes no guarantees on termination; it claims that it is always the case that P holds *or* that the program has diverged. This termination-insensitivity lends itself nicely to expressing partial correctness of programs, for which we consider only *safety* (that no erroneous states are reached) with no liveness guarantees.

We show that Total Outcome Logic subsumes non-deterministic, partial correctness Hoare Logic. Define: [3]

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

[3] Here, we represent our result domain as $\mathcal{P}(\Sigma_{\circlearrowleft})$ instead of weighting functions $\mathcal{W}_{\mathcal{A}}^{\Sigma_{\circlearrowleft}}$. In the Boolean semiring (nondeterministic setting), these are isomorphic, as each weighting function serves as the characteristic function of some set of states.

In the literature, $[C]Q$ is referred to as the *weakest liberal precondition* and corresponds to a demonic notion of correctness; $\sigma \in [C]Q$ indicates that all terminating outcomes of C starting in σ must fall in Q. On the other hand, $\langle C\rangle Q$ is the *weakest possible precondition* [15], where $\sigma \in \langle C\rangle Q$ indicates that *some* outcome of running C from σ terminates in Q.

We can now draw the connection between Total Outcome Logic triples (using $\Box$) and the partial correctness Hoare triple:

Theorem 1 (Subsumption of Hoare Logic).

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

A triple $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$ in Lisbon Logic [29] states that running program C starting in a state satisfying P *may* terminate in Q. Besides capturing correctness under angelic nondeterminism, Lisbon triples are useful for expressing *incorrectness*. By taking Q to be a class of erroneous states, $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$ checks for the possibility of incorrect behavior. Total Outcome Logic triples (using $\Diamond$) also subsume Lisbon triples.

Theorem 2 (Subsumption of Lisbon Logic).

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

Alternatively, we can define other modalities by giving a termination-sensitive variant $\blacksquare$ of $\Box$:

$$\blacksquare P \triangleq \exists u : U.\, P^{(u)} = \{m \mid \mathsf{supp}(m) \subseteq P\}$$

We take $\blacklozenge$ to be the De Morgan dual:

$$\blacklozenge P \triangleq (\exists (u,v) : U^2/\{0,0\}.\, P^{(u)} \oplus \Uparrow^{(v)} \oplus \top) = \{m \mid \mathsf{supp}(m) \cap P_{\circlearrowleft} \neq \emptyset\}$$

Observe that by having $\blacksquare$ guarantee termination, we define $\blacklozenge$ to permit nontermination. That is, $\Diamond P$ states that P sometimes holds *or* the program sometimes diverges. We also have counterparts to $[C]Q$ and $\langle C\rangle Q$, defined above:

$$[C]^\star Q \triangleq \{\sigma \mid [\![C]\!](\sigma) \subseteq Q\} \quad \langle C\rangle^\star Q \triangleq \{\sigma \mid [\![C]\!](\sigma) \cap Q_{\circlearrowleft} \neq \emptyset\}$$

$[C]^\star Q$ is regarded as the *weakest precondition*, whereas we will call $\langle C\rangle^\star Q$ the *weakest liberal possible precondition*. The $\blacksquare$ modality then allows us to encode total correctness Hoare triples:

Theorem 3 (Subsumption of Total Correctness Hoare Logic).

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

The dual to this is a notion of *partial Lisbon logic* $\langle C\rangle^\star Q$, in which incorrectness is expanded to include both erroneous states and diverging executions, and is similar to what Zhang and Kaminski [35] referred to as partial incorrectness.

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

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

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

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

Fig. 4: Structural rules.

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

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

Fig. 5: Inference rules for non-iterative program commands.

3.4 Proof Theory

The main reasoning apparatus in TOL consists of inference rules for deriving triples, shown in Figures 4 and 5. The bulk of these carry over from Outcome Logic [37], although some must be adapted to account for the extra effect of nontermination in our semantics. We introduce the rules as belonging to three main categories: structural rules, standard command rules, and iteration.

Structural Rules are provided in Figure 4 and hold for arbitrary program commands C, as they capture logical rules that apply to pre- and post-conditions. These include the trivial rules, False and True, which feature the vacuous precondition $\bot$ and the weakest postcondition $\top$, respectively. Div handles nontermination, stating that assertions $\Uparrow^{(u)}$ are preserved across programs.

A given triple can be left-scaled by any weight $u \in U$ using Scale. The rules Disj, Conj, and Choice allow multiple triples to be conjoined via the connectives $\wedge$, $\vee$, and $\oplus$, respectively. Introduction of existential quantification is accomplished through Exists. And finally, the standard Consequence rule is borrowed from Hoare logic to allow for the strengthening of preconditions and weakening of postconditions.

Standard Command Rules are provided in Figure 5 to specify how non-iterative program commands transform outcome assertions. Skip and Seq are

$$\frac{(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty \qquad (\varphi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty \qquad \forall n \in \mathbb{N}. \qquad \varphi_n \vDash \mathsf{true} \qquad \zeta_n \vDash \mathsf{div} \qquad \langle \varphi_n \oplus \zeta_n \rangle \ \mathsf{assume}\ e \fatsemi C \ \langle \varphi_{n+1} \oplus \zeta_{n+1} \rangle \qquad \langle \varphi_n \rangle \ \mathsf{assume}\ e' \ \langle \psi_n \rangle}{\langle \varphi_0 \oplus \zeta_0 \rangle \ C^{\langle e, e' \rangle} \ \langle \psi_\infty \oplus \zeta_\infty \rangle} \ \textsc{Iter}$$

Fig. 6: Inference rule for iteration.

standard for Hoare-style logics. The Assume rule has the premise $\varphi \vDash e = u$, which is defined below:

Definition 10 (Assertion Entailment). *Given an assertion φ, expression e, and weight $u \in U$, we write $\varphi \vDash e = u$ (read as: φ entails $e = u$) iff:*

$$\forall m \in \varphi.\ (\circlearrowleft \notin \mathsf{supp}(m)) \wedge (\forall \sigma \in \mathsf{supp}(m).\ [\![e]\!](\sigma) = u)$$

That is, every weighting function m in φ surely terminates, and e must evaluate to $u \in U$ in any state that m inhabits. For tests $b \in \mathsf{Test}$, we introduce the shorthand $\varphi \vDash b$ to mean $\varphi \vDash b = \mathbb{1}$, which is equivalent to $\varphi \implies \blacksquare b$.

In Plus, the side condition $\varphi \vDash \mathsf{true}$ is equivalent to just a sure-termination guarantee: for $m \in \varphi$, $\circlearrowleft \notin \mathsf{supp}(m)$. If this holds, the triples proved for both branches are combined with outcome conjunction. Sure-termination must be enforced in these cases as the additional effect of nontermination is sequenced differently than for regular weighted states. This is a major difference between Outcome Logic and Total Outcome Logic.

Iteration. Our nontermination semantics culminates in the Iter rule for iteration commands $C^{\langle e, e' \rangle}$ (Figure 6), which also serves to introduce any assertions pertaining to nontermination. The form of this rule corresponds roughly to an unrolling of the loop and describes sequences of assertions that hold between iterations. These assertions come in three sorts:

- $(\varphi_n)_{n\in\mathbb{N}}$ describe ongoing lines of computation, i.e. those that have not yet terminated or been deemed nonterminating. These include program configurations that are passed to the next iteration of the loop and subsequently transformed. Note that we have as premise that $\varphi_n \vDash \mathsf{true}$ for every n, so every $m \in \varphi_n$ describes only weights on program states Σ, but not the weight on $\circlearrowleft$ – we call computations restricted to the states Σ, Σ-computations.
- $(\psi_n)_{n\in\mathbb{N}}$ describe the terminating outcomes that are accrued in iteration n. In particular, observe that ψ_n is obtained by filtering φ_n through command $\mathsf{assume}\ e'$: $\langle \varphi_n \rangle \ \mathsf{assume}\ e' \ \langle \psi_n \rangle$.
- $(\zeta_n)_{n\in\mathbb{N}}$ describe nontermination encountered in the n^{th} iteration (*i.e.,* via nested loops). We write $\zeta_n \vDash \mathsf{div}$ as a premise to enforce this characterization.

Definition 11 (Nonterminating Assertions). *For outcome assertion φ, we write $\varphi \vDash \mathsf{div}$ (read as: φ is nonterminating) iff for all $m \in \varphi$, $\mathsf{supp}(m) \subseteq \{\circlearrowleft\}$.*

The conclusion of the rule has as postcondition two *limiting* assertions ψ_∞ and ζ_∞. Intuitively, the former describes the aggregate of all terminating outcomes accrued by each ψ_n. We represent this formally by writing $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$, which is defined below:

Definition 12 (Converging Assertions). *A family $(\psi_n)_{n\in\mathbb{N}}$ of outcome assertions converges to ψ_∞ (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 $\sum_{n\in\mathbb{N}} m_n \vDash \psi_\infty$.*

On the other hand, ζ_∞ describes the degree of nontermination of the loop in the limit as the number of iterations approaches ∞. This notion is captured by the semantic assertion $(\varphi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty$, which we take to mean:

Definition 13 (Diverging Assertions). *A family $(\psi_n)_{n\in\mathbb{N}}$ of outcome assertions diverges to ψ_∞ (written $(\psi_n)_{n\in\mathbb{N}} \Uparrow \psi_\infty$) iff for all $(m_n)_{n\in\mathbb{N}}$, if $m_n \vDash \psi_n$ for all $n \in \mathbb{N}$, then $(\inf_{n\in\mathbb{N}} |m_n| \cdot \top) \cdot \eta(\circlearrowleft) \vDash \psi_\infty$.*

In other words, the total mass in $\varphi_n \oplus \zeta_n$ in the limit is transfered to the nontermination outcome $\circlearrowleft$. Intuitively, nontermination in the loop has two possible sources: nonterminating weight from nested loops, which is accounted for by ζ_n's, and weight from the loop itself, accounted for by the φ_n's.

This rule allows us to explain the paradox that we introduced in Section 1, in which terminating nondeterministic programs cannot make unboundedly many choices, but almost surely terminating probabilistic programs can. Consider the two programs below, which both increment x in a loop, but the program on the left if interpreted in the Boolean semiring, whereas the one on the right is interpreted in the probabilistic semiring.

$$x \coloneqq 0 \fatsemi (x \coloneqq x+1)^{\langle 1,1\rangle} \qquad\qquad x \coloneqq 0 \fatsemi (x \coloneqq x+1)^{\langle p,1-p\rangle}$$

For the nondeterministic program, we set $\varphi_n \triangleq x = n$, $\psi_n \triangleq x = n$, and $\zeta_n \triangleq \mathsf{true}^{(0)}$ for all $n \in \mathbb{N}$. Letting $\psi_\infty \triangleq \bigoplus_{k\in\mathbb{N}} x = k$, clearly $(x = n)_{n\in\mathbb{N}} \rightsquigarrow \bigoplus_{k\in\mathbb{N}} x = k$, indicating that there is a terminating outcome where $x = k$ for any natural number k. Letting $\zeta_\infty \triangleq \Uparrow$, we also have $(x = n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty$; the weight of continuing to iterate remains to be 1 for all $n \in \mathbb{N}$, so there is a nonterminating trace of weight 1, giving us the final specification.

$$\langle \mathsf{true}\rangle \; x \coloneqq 0 \fatsemi (x \coloneqq x+1)^{\langle 1,1\rangle} \; \langle (\bigoplus_{k\in\mathbb{N}} x = k) \oplus \Uparrow \rangle$$

By contrast, in the probabilistic program we instead set $\varphi_n \triangleq (x = n)^{(p^n)}$ and $\psi_n \triangleq (x = n)^{(p^n\cdot(1-p))}$. The terminating outcomes then form a geometric distribution $\bigoplus_{k\in\mathbb{N}} (x = k)^{(p^k\cdot(1-p))}$. Since the weight captured by the φ_n assertions is decreasing towards 0, the infimum is 0, giving us $\zeta_\infty \triangleq \mathsf{true}^{(0)}$, meaning that the infinite execution occurs with zero probability, and it is therefore omitted from the specification.

$$\langle \mathsf{true}\rangle \; x \coloneqq 0 \fatsemi (x \coloneqq x+1)^{\langle p,1-p\rangle} \; \langle \bigoplus_{k\in\mathbb{N}} (x = k)^{(p^k\cdot(1-p))} \rangle$$

Remark 1 (Accommodating Nontermination). In general, application of rules with some assertion entailment $\varphi \vDash e = u$ as a premise involves decomposing

the precondition into components that describe terminating vs. nonterminating outcomes – the latter of which are handled by Div or some derivative of it.

As an example, take $\varphi = \varphi_t \oplus \varphi_{\circlearrowright}$ where $\varphi_t \vDash \mathsf{true}$ and $\varphi_{\circlearrowright} \vDash \mathsf{div}$. We show a derivation involving Plus across program $C_1 + C_2$, in which we are given premises $\langle \varphi_t \rangle\ C_1\ \langle \psi_{t_1} \rangle$ and $\langle \varphi_t \rangle\ C_2\ \langle \psi_{t_2} \rangle$:

$$\dfrac{\dfrac{\langle \varphi_t \rangle\ C_1\ \langle \psi_{t_1} \rangle \qquad \langle \varphi_t \rangle\ C_2\ \langle \psi_{t_2} \rangle}{\langle \varphi_t \rangle\ C_1 + C_2\ \langle \psi_{t_1} \oplus \psi_{t_2} \rangle}\ \textsc{Plus} \qquad \dfrac{\varphi_{\circlearrowright} \vDash \mathsf{div}}{\langle \varphi_{\circlearrowright} \rangle\ C_1 + C_2\ \langle \varphi_{\circlearrowright} \rangle}\ \textsc{Div}^*}{\langle \varphi_t \oplus \varphi_{\circlearrowright} \rangle\ C_1 + C_2\ \langle \psi_{t_1} \oplus \psi_{t_2} \oplus \varphi_{\circlearrowright} \rangle}\ \textsc{Choice}$$

3.5 Soundness and Relative Completeness

The proof system that we defined for Total Outcome Logic is sound and complete. Soundness means that every triple that is derivable via the inference rules is semantically valid. We state this theorem below, and prove it in the appendix.

Theorem 4 (Soundness). $\vdash \langle \varphi \rangle\ C\ \langle \psi \rangle \implies \vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$.

Relative completeness means that any valid triple can be derived in our proof system, provided an oracle that decides semantic properties such as the implications used in the rule of Consequence and the assertion entailments used in the Assume rule.

Theorem 5 (Relative Completeness). $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle \implies \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$

4 Derived Rules

In this section, we introduce derived rules for common inferences that are made in Total OL proofs.

Divergence. We begin with a Div* rule, which allows the precondition to be any assertion that implies nontermination; it need not quantify the exact weight. In our semantics, we know that for any configuration, a program cannot interfere with existing weight on nontermination $\circlearrowright$. This means that any outcome assertion that only describes nontermination should be preserved across triples. It is useful to expand the Div rule to reflect this:

$$\dfrac{\zeta \vDash \mathsf{div}}{\langle \zeta \rangle\ C\ \langle \zeta \rangle}\ \textsc{Div}^*$$

If Statements and While Loops. Recall from Section 2.1 that we gave syntactic sugar to define if statements and while loops in terms of the branching, iteration, and $\mathsf{assume}\ b$ constructs of our language. Whereas we previously only gave rules for standard branching and iteration, we now give rules for if and while. The if rule is the same as the one from standard Outcome Logic, and

requires the precondition to be split into two parts to satisfy the guard and its negation. The two branches can then be analyzed compositionally.

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

The rule for while loops differs from standard Outcome Logic, as it includes the ability to reason about nontermination. We use three families of assertions: φ_n represents the outcomes where the guard b remains true, ψ_n represents the outcomes where b is false, and ζ_n represents nontermination in nested loops. Compared to the ITER rule that we saw previously, the WHILE rule only has a single premise, as the entailments for φ_n and ψ_n remove the need to derive the behaviors of the ASSUME commands.

$$\frac{\begin{array}{c}(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty \qquad (\varphi_n \oplus \psi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty \qquad \forall n \in \mathbb{N}.\\ \varphi_n \vDash b \qquad \psi_n \vDash \neg b \qquad \zeta_n \vDash \mathsf{div} \qquad \langle \varphi_n \oplus \zeta_n \rangle\ C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \oplus \zeta_{n+1} \rangle\end{array}}{\langle \varphi_0 \oplus \psi_0 \oplus \zeta_0 \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \psi_\infty \oplus \zeta_\infty \rangle}\ \textsc{While}$$

The original Outcome logic can only reason on terminating outcomes, and is thus only able to detect nontermination when no terminating executions are witnessed, i.e. via triples of the form $\langle \varphi \rangle_{\mathrm{OL}}\ C\ \langle \emptyset \cdot \top \rangle_{\mathrm{OL}}$. Whereas this restricts proofs to *possible termination* and *guaranteed nontermination*, TOL now allows us to demonstrate *sure-termination* or *possible nontermination*.

In terms of triples, the former is often bundled with a safety property as a total correctness specification. We can verify total correctness using *variants*, which track the number of iterations a loop makes until reaching a zero-variant – a point of sure-termination where the loop guard fails:

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

Hoare Logic and Lisbon Logic. We will now derive the rules of Lisbon Logic and partial and total correctness Hoare Logic. We will present most of the rules as total correctness triples (of the form $\langle P \rangle\ C\ \langle \blacksquare Q \rangle$), since $\blacksquare Q \Rightarrow \square Q$, and therefore a total correctness triple can be weakened to be a partial correctness one. We first give rules for sequencing total specifications, since the modalities used in pre- and postconditions do not exactly match.

$$\textsc{Seq-Total-Hoare}\quad \frac{\langle P \rangle\ C_1\ \langle \blacksquare Q \rangle \qquad \langle Q \rangle\ C_2\ \langle \blacksquare R \rangle}{\langle P \rangle\ C_1 \fatsemi C_2\ \langle \blacksquare R \rangle} \qquad \textsc{Seq-Lisbon}\quad \frac{\langle P \rangle\ C_1\ \langle \Diamond Q \rangle \qquad \langle Q \rangle\ C_2\ \langle \Diamond R \rangle}{\langle P \rangle\ C_1 \fatsemi C_2\ \langle \Diamond R \rangle}$$

Similarly, we give a rule for if statements in Hoare and Lisbon Logic, which do not require entailments since the premises of the rules directly specify whether

the guard is true or not.

$$\frac{\langle P \wedge b\rangle\ C_1\ \langle \blacksquare Q\rangle \qquad \langle P \wedge \neg b\rangle\ C_2\ \langle \blacksquare Q\rangle}{\langle P\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \blacksquare Q\rangle}\ \textsc{If-Hoare}$$

$$\frac{\langle P \wedge b\rangle\ C_1\ \langle \Diamond Q\rangle \qquad \langle P \wedge \neg b\rangle\ C_2\ \langle \Diamond Q\rangle}{\langle P\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q\rangle}\ \textsc{If-Lisbon}$$

The inference rules for different variants of these logics differs the most when it comes to loops. It is well known that the invariant rule is complete for reasoning about loops in partial correctness Hoare Logic [8]. We derive the invariant rule in our embedded logic.

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

Using the Variant rule that we derived previously, we can also derive the standard variant rule employed in total Hoare Logic. This rule uses an integer valued *ranking* expression R, which represents how close the program is to termination. Each iteration of the loop strictly decreases R, and the program terminates once $R = 0$, so the program must always terminate.

$$\frac{(P \wedge b) \implies R > 0 \qquad \langle P \wedge b \wedge R = n\rangle\ C\ \langle \blacksquare(P \wedge R < n)\rangle}{\langle P \wedge R \geq 0\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \blacksquare(P \wedge \neg b)\rangle}\ \textsc{Hoare-Variant}$$

Nontermination. Bug-finding is often a more practical goal than correctness verification [30]. We can thus shift our sight from proving total correctness to proving the presence of nontermination by establishing a *quasi-invariant* [21]. This is a property preserved by the loop body which ensures that the loop guard holds; the quasi-invariant forces indefinite reiteration, and any computation that satisfies the quasi-invariant once is trapped within the loop. Taken semantically, this aligns closely with the notion of a *recurrent set* [14] of program states, visited infinitely often by the loop.

In a nondeterministic setting, we can define this either angelically or demonically to show that the program *sometimes* or *always* diverges, respectively. With some abuse of notation, we take:

$$\Diamond \Uparrow \triangleq \exists u : U \setminus \{\mathbb{0}\}.\ \Uparrow^{(u)} \oplus \top \qquad \Box \Uparrow \triangleq \exists u : U.\ \Uparrow^{(u)}$$

Then, the following two rules can be derived to prove nontermination.

$$\frac{P \vDash b \qquad \langle P\rangle\ C\ \langle \Diamond P\rangle}{\langle P\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Diamond \Uparrow\rangle}\ \textsc{QInv-Angel} \qquad \frac{P \vDash b \qquad \langle P\rangle\ C\ \langle \Box P\rangle}{\langle P\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Box \Uparrow\rangle}\ \textsc{QInv-Demon}$$

5 Case Studies

We want to emphasize that the power afforded by Total Outcome Logic lies in its subsumption of different types of reasoning, especially pertaining to termination

and nontermination. To demonstrate the breadth of program properties that can be proved in TOL, we deploy it on a few toy examples. In these examples, we use variable assignments $x := E$ as basic actions. These actions are defined in the usual way, as in Zilberstein [37].

5.1 Total Correctness: Quicksort Partition

In the classical formulation of Hoare logic for total correctness, Manna and Pnueli [1974] derive the correctness of the Quicksort partition algorithm, where A is an integer array of size $n+1$ and pivot is the pre-computed pivot value:

$$\textsc{Partition} \triangleq \left\{ \begin{array}{l} i := 0 \fatsemi j := n \fatsemi \\ \mathsf{while}\ i \leq j\ \mathsf{do} \\ \quad \mathsf{if}\ A[i] \leq \mathsf{pivot}\ \mathsf{then}\ i := i+1 \fatsemi \\ \quad \mathsf{else\ if}\ A[j] \geq \mathsf{pivot}\ \mathsf{then}\ j := j-1 \fatsemi \\ \quad \mathsf{else}\ \mathit{swap}(A,i,j) \fatsemi i := i+1 \fatsemi j := j-1 \end{array} \right.$$

Total correctness of Partition can similarly be established in TOL. Again, we can encode a total correctness triple $[P]\ C\ [Q]$ as outcome triple $\langle P\rangle\ C\ \langle \blacksquare Q\rangle$. As the same inference rules available in total Hoare logic can be derived in TOL, our desired proof follows essentially the same steps. First, for a given pivot value pivot, define predicates α and β over integers:

$$\alpha(i) \triangleq \bigwedge_{0 \leq k < i} A[k] \leq \mathsf{pivot} \qquad \beta(j) \triangleq \bigwedge_{j < k \leq n} A[k] \geq \mathsf{pivot}$$

We want to derive $\langle \mathsf{true}\rangle$ Partition $\langle \blacksquare(\alpha(i) \wedge \beta(j) \wedge i > j)\rangle$, which states that any run of the program must terminate having divided the array into two sections: elements $\leq$ pivot and those $\geq$ pivot. We decorate Partition in Figure 8 to sketch the full proof.

The last step of the proof uses Hoare-Variant. Our *invariant* $\alpha(i) \wedge \beta(j)$ enforces two boundaries in the array; every element behind index i must be $\leq$ pivot, while every element after j must be $\geq$ pivot. A natural choice for the *variant* is then the distance between the boundaries $j - i$, which we show must approach one another after each iteration. That is, we need to derive the following across the loop body C:

$$\langle \alpha(i) \wedge \beta(i) \wedge i \leq j \wedge j - i = m\rangle\ C\ \langle \blacksquare(\alpha(i) \wedge \beta(i) \wedge j - i < m)\rangle$$

Clearly, the loop condition $i \leq j$ implies $j - i > 0$. This provides us with all the premises needed to conclude $\langle \alpha(i) \wedge \beta(j)\rangle$ Partition $\langle \blacksquare(\alpha(i) \wedge \beta(j) \wedge i > j)\rangle$.

5.2 Proving Nontermination

Raad et al. [32] advocated the need for formal methods to prove nontermination, as many industrial codebases have bugs that cause programs to diverge and

$$\textsc{Nt} \triangleq \left\{ \begin{array}{l} x := 1 \fatsemi y := 2 \fatsemi \\ \mathsf{while}\ x + y > 1\ \mathsf{do} \\ \quad x := 3 - x \fatsemi \\ \quad y := 3 - y \fatsemi \end{array} \right. \qquad \textsc{MallocDiv} \triangleq \left\{ \begin{array}{l} \mathsf{byte} * p \fatsemi \\ p := (\mathsf{byte}*)\ \mathsf{malloc}(\mathsf{size} + 16) \fatsemi \\ \mathsf{while}\ p = \mathsf{NULL}\ \mathsf{do} \\ \quad p := (\mathsf{byte}*)\ \mathsf{malloc}(\mathsf{size} + 16) \fatsemi \end{array} \right.$$

Fig. 7: Two non-terminating programs.

are difficult to discover with testing. Here, we use some of their examples to demonstrate how TOL can be used for nontermination proving. Consider the Nt program in Figure 7. To show that this program indeed always diverges, we make use of the QInv-Demon rule to derive the triple $\langle \mathsf{true} \rangle \textsc{Nt} \langle \Box \Uparrow \rangle$.

$$\begin{array}{l} \langle \mathsf{true} \rangle \\ x := 1 \fatsemi y := 2 \fatsemi \\ \langle x = 1 \wedge y = 2 \rangle \\ \quad \Longrightarrow \langle x + y = 3 \rangle \\ \qquad x := 3 - x \fatsemi \\ \qquad \langle 3 - x + y = 3 \rangle \qquad //\textsc{Assign} \\ \qquad y := 3 - y \fatsemi \\ \qquad \langle (3 - x) + (3 - y) = 3 \rangle \qquad //\textsc{Assign} \Longrightarrow \langle x + y = 3 \rangle \\ \qquad\quad \Longrightarrow \langle \Box (x + y = 3) \rangle \\ \langle \Box \Uparrow \rangle \end{array}$$

As a second example, we will use the rule QInv-Angel to establish the *possibility* of nontermination in the presence of branching. Consider a loop involving the malloc function in C as in the program MallocDiv in Figure 7.

MallocDiv allocates memory in a loop, iterating until a successful call to malloc and p is set to a non-null value. Crucially, malloc *may* fail each time, giving rise to a divergent execution. We can thus see that the assertion $p = \mathsf{NULL}$ is a (angelic) quasi-invariant for this loop; we have

$$\frac{\langle p = \mathsf{NULL} \rangle\ p := (\mathsf{byte}*)\ \mathsf{malloc}(\mathsf{size} + 16)\ \langle \Diamond (p = \mathsf{NULL}) \rangle}{\langle p = \mathsf{NULL} \rangle\ \mathsf{while}\ p = \mathsf{NULL}\ \mathsf{do}\ \ldots\ \langle \Diamond \Uparrow \rangle}\ \textsc{QInv-Angel}$$

And thus MallocDiv may diverge. Similar proofs of *guaranteed nontermination* can be derived using QInv-Demon.

5.3 Probabilistic Nontermination

Whereas much of the work on probabilistic program analysis focuses on *almost sure termination* [18, 26, 27]—programs that terminate with probability 1—there are many programs simulating physical phenomena, which do not almost surely terminate, but are still important to reason about [17]. An example of

$\langle \mathsf{true} \rangle$
$i := 0 \fatsemi j := n \fatsemi$
$\langle i = 0 \wedge j = n \rangle \implies \langle \alpha(i) \wedge \beta(j) \wedge j - i \geq 0 \rangle$
while $i \leq j$ do
  $\langle \alpha(i) \wedge \beta(j) \wedge i \leq j \wedge j - i = m \rangle \implies \langle \alpha(i) \wedge \beta(j) \wedge j - i = m \rangle$
  if $A[i] \leq$ pivot then
    $\langle \alpha(i) \wedge \beta(j) \wedge A[i] \leq \mathsf{pivot} \wedge j - i = m \rangle \implies \langle \alpha(i+1) \wedge \beta(j) \wedge j - i = m \rangle$
    $i := i + 1 \fatsemi$
    $\langle \alpha(i) \wedge \beta(j) \wedge j - i = m - 1 \rangle$ // Assign
  else if $A[j] \geq$ pivot then
    $\langle \alpha(i) \wedge \beta(j) \wedge A[j] \geq \mathsf{pivot} \wedge j - i = m \rangle \implies \langle \alpha(i) \wedge \beta(j-1) \wedge j - i = m \rangle$
    $j := j - 1 \fatsemi$
    $\langle \alpha(i) \wedge \beta(j) \wedge j - i = m - 1 \rangle$ // Assign
  else
    $\langle \alpha(i) \wedge \beta(j) \wedge A[i] > \mathsf{pivot} \wedge A[j] < \mathsf{pivot} \wedge j - i = m \rangle$
    $swap(A, i, j) \fatsemi$
    $\langle \alpha(i) \wedge \beta(j) \wedge A[j] > \mathsf{pivot} \wedge A[i] < \mathsf{pivot} \wedge j - i = m \rangle$
    $\implies \langle \alpha(i+1) \wedge \beta(j-1) \wedge j - i = m \rangle$
    $i := i + 1 \fatsemi j := j - 1$
    $\langle \alpha(i) \wedge \beta(j) \wedge j - i = m - 2 \rangle$ // Assign
  $\langle \alpha(i) \wedge \beta(j) \wedge j - i < m \rangle$ // If
  $\implies \langle \blacksquare(\alpha(i) \wedge \beta(j) \wedge j - i < m) \rangle$
$\langle \blacksquare\, \alpha(i) \wedge \beta(j) \wedge i > j \rangle$ // Hoare-Variant

Fig. 8: Decorated proof of the Partition program.

such programs is a *tortoise-and-hare* scenario. Here, a tortoise marches forward at a constant speed, and an erratic hare attempts to catch up.

The program below models such a scenario, where t represents the position of the tortoise, h represents the position of the hare, and k is the number of steps that have been taken. The tortoise starts one step ahead of the hare. Each step, the hare either takes a step forward, or leaps forward. However, the hare gets tired over time, so it can only leap a distance of $1 + \frac{1}{2^k}$. The program terminates if the hare catches up to the tortoise. Note that $C_1 +_p C_2$ is syntactic sugar for $(\mathsf{assume}\ p \fatsemi C_1) + (\mathsf{assume}\ 1 - p \fatsemi C_2)$.

$$t := 1 \fatsemi h := 0 \fatsemi k := 0 \fatsemi \quad \mathsf{while}\ h < t\ \mathsf{do}\ \begin{cases} t := t + 1 \fatsemi \\ (h := h + 1) +_{\frac{1}{2}} (h := h + 1 + \frac{1}{2^k}) \fatsemi \\ k := k + 1 \end{cases}$$

Clearly, the hare catches the tortoise with probability $\frac{1}{2}$ if its initial move is a leap forward. If the hare's first move is not a leap forward, then it catches the tortoise with probability 0. We can see this informally, since in the best case, the amount that the hare catches up each round is given by the geometric series $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots$, which converges to 1 in the limit, but that occurs only if the hare *always* leaps forward, an event with probability $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \cdots = 0$.

We will make this formal by analyzing the program with the While rule. For this, we need to define the three families of assertions φ_n, ψ_n and ζ_n. We start with φ_n, which describes the outcomes where execution continues.

$$\varphi_0 \triangleq t = 1 \wedge h = 0 \wedge k = 0$$
$$\varphi_1 \triangleq (t = 2 \wedge h = 1 \wedge k = 1)^{\left(\frac{1}{2}\right)}$$
$$\varphi_n \triangleq (t - h = \frac{1}{2^{n-1}} \wedge k = n)^{\left(\frac{1}{2^n}\right)} \oplus (t - h > \frac{1}{2^{n-1}} \wedge k = n)^{\left(\frac{2^{n-1}-1}{2^n}\right)} \quad \text{if } n \geq 2$$

If $n = 0$, then φ_0 simply describes the start state of the program. If $n = 1$, then φ_n describes the outcome where the hare's first move was a regular step forward. For $n \geq 2$, we only describe two outcomes. With probability $\frac{1}{2^n}$, the hare leaps forward on every step after the first one, and in that case, the difference in position is exactly $\frac{1}{2^n}$. If not, then the hare did not leap on at least one step, in which case the difference in position is strictly greater than $\frac{1}{2^n}$.

The ψ_n assertions describe the terminating outcomes. The program can terminate after the first step, so $\psi_1 \triangleq (t = 2 \wedge h = 2 \wedge k = 1)^{\left(\frac{1}{2}\right)}$. For every other $n \in \mathbb{N}$, we have $\psi_n \triangleq \mathsf{true}^{(0)}$. Finally, $\psi_\infty \triangleq (h = t)^{\left(\frac{1}{2}\right)}$, the only terminating outcome. It is easy to see that $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$. The ζ_n assertions describe nonterminating outcomes. The program only diverges in the limit, so we get $\zeta_n \triangleq \mathsf{true}^{(0)}$ for all $n \in \mathbb{N}$ and $\zeta_\infty \triangleq \Uparrow^{\left(\frac{1}{2}\right)}$. Clearly $(\varphi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty$, since according to φ_n, the probability that $t > h$ after n iterations is $\frac{2^{n-1}-1}{2^n}$, which converges to $\frac{1}{2}$. We now establish the premises of the While rule, ignoring the ζ_n terms since they are vacuous. We start with the case where $n = 0$.

$\langle \varphi_0 \rangle \Leftrightarrow \langle t = 1 \wedge h = 0 \wedge k = 0 \rangle$
$t \coloneqq t + 1$ ⨟
$\langle t = 2 \wedge h = 0 \wedge k = 0 \rangle$
$(h \coloneqq h + 1) +_{\frac{1}{2}} (h \coloneqq h + 1 + \frac{1}{2^k})$ ⨟
$\langle (t = 2 \wedge h = 1 \wedge k = 0)^{\left(\frac{1}{2}\right)} \oplus (t = 2 \wedge h = 2 \wedge k = 0)^{\left(\frac{1}{2}\right)} \rangle$
$k \coloneqq k + 1$
$\langle (t = 2 \wedge h = 1 \wedge k = 1)^{\left(\frac{1}{2}\right)} \oplus (t = 2 \wedge h = 2 \wedge k = 1)^{\left(\frac{1}{2}\right)} \rangle \Leftrightarrow \langle \varphi_1 \oplus \psi_1 \rangle$

Now, we show the case where $n = 1$.

$$
\begin{array}{l}
\langle \varphi_1 \rangle \Leftrightarrow \langle (t = 2 \land h = 1 \land k = 1)^{(\frac{1}{2})} \rangle \\
\quad t \coloneqq t + 1 \fatsemi \\
\langle (t = 3 \land h = 1 \land k = 1)^{(\frac{1}{2})} \rangle \\
\quad (h \coloneqq h + 1) +_{\frac{1}{2}} (h \coloneqq h + 1 + \frac{1}{2^k}) \fatsemi \\
\langle (t = 3 \land h = 2 \land k = 1)^{(\frac{1}{4})} \oplus (t = 3 \land h = 2 + \frac{1}{2} \land k = 1)^{(\frac{1}{4})} \rangle \\
\quad k \coloneqq k + 1 \\
\langle (t = 3 \land h = 2 \land k = 2)^{(\frac{1}{4})} \oplus (t = 3 \land h = 2 + \frac{1}{2} \land k = 2)^{(\frac{1}{4})} \rangle \\
\Rightarrow \langle (t - h = \frac{1}{2} \land k = 2)^{(\frac{1}{4})} \oplus (t - h > \frac{1}{2} \land k = 2)^{(\frac{1}{4})} \rangle \Leftrightarrow \langle \varphi_2 \oplus \psi_2 \rangle
\end{array}
$$

Finally, we show the case where $n \geq 2$.

$$
\begin{array}{l}
\langle \varphi_n \rangle \Leftrightarrow \langle (t - h = \frac{1}{2^{n-1}} \land k = n)^{(\frac{1}{2^n})} \oplus (t - h > \frac{1}{2^{n-1}} \land k = n)^{\left(\frac{2^{n-1}-1}{2^n}\right)} \rangle \\
\quad t \coloneqq t + 1 \fatsemi \\
\langle (t - h = 1 + \frac{1}{2^{n-1}} \land k = n)^{(\frac{1}{2^n})} \oplus (t - h > 1 + \frac{1}{2^{n-1}} \land k = n)^{\left(\frac{2^{n-1}-1}{2^n}\right)} \rangle \\
\quad (h \coloneqq h + 1) +_{\frac{1}{2}} (h \coloneqq h + 1 + \frac{1}{2^k}) \fatsemi \\
\left\langle \begin{array}{l} (t - h = \frac{1}{2^{n-1}} \land k = n)^{\left(\frac{1}{2^{n+1}}\right)} \oplus (t - h = \frac{1}{2^{n-1}} - \frac{1}{2^n} \land k = n)^{\left(\frac{1}{2^{n+1}}\right)} \oplus \\ (t - h > \frac{1}{2^{n-1}} \land k = n)^{\left(\frac{2^{n-1}-1}{2^{n+1}}\right)} \oplus (t - h > \frac{1}{2^{n-1}} - \frac{1}{2^n} \land k = n)^{\left(\frac{2^{n-1}-1}{2^{n+1}}\right)} \end{array} \right\rangle \\
\Rightarrow \left\langle \begin{array}{l} (t - h > \frac{1}{2^n} \land k = n)^{\left(\frac{1}{2^{n+1}}\right)} \oplus (t - h = \frac{1}{2^n} \land k = n)^{\left(\frac{1}{2^{n+1}}\right)} \oplus \\ (t - h > \frac{1}{2^n} \land k = n)^{\left(\frac{2^{n-1}-1}{2^{n+1}}\right)} \oplus (t - h > \frac{1}{2^n} \land k = n)^{\left(\frac{2^{n-1}-1}{2^{n+1}}\right)} \end{array} \right\rangle \\
\Rightarrow \langle (t - h = \frac{1}{2^n} \land k = n)^{\left(\frac{1}{2^{n+1}}\right)} \oplus (t - h > \frac{1}{2^n} \land k = n)^{\left(\frac{2^n-1}{2^{n+1}}\right)} \rangle \\
\quad k \coloneqq k + 1 \\
\langle (t - h = \frac{1}{2^n} \land k = n + 1)^{\left(\frac{1}{2^{n+1}}\right)} \oplus (t - h > \frac{1}{2^n} \land k = n + 1)^{\left(\frac{2^n-1}{2^{n+1}}\right)} \rangle \Leftrightarrow \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle
\end{array}
$$

We complete the proof with a straightforward application of the While rule.

$$
\frac{\forall n \in \mathbb{N}.\quad \langle \varphi_n \rangle\ C_{\mathsf{body}}\ \langle \varphi_{n+1} \oplus \psi_{n+1} \rangle}{\langle t = 1 \land h = 0 \land k = 0 \rangle\ \mathsf{while}\ h < t\ \mathsf{do}\ C_{\mathsf{body}}\ \langle (h = t)^{(\frac{1}{2})} \oplus \Uparrow^{(\frac{1}{2})} \rangle}\ \textsc{While}
$$

This tells us that the program terminates in a state where $h = t$ with probability $\frac{1}{2}$ and it diverges (the hare never catches the tortoise) with probability $\frac{1}{2}$.

6 Related Work

Correctness, Incorrectness, and Nondeterminism. Recent years have seen increased interests in program logics with abilities to reason about nondeterminism in varying ways. This was spurred by Incorrectness Logic [30], which advocated for using angelic nondeterminism to identify programs that *sometimes* have undesirable behavior. Subsequently, new logics such as Outcome Logic [38, 40, 39], Hyper Hoare Logic [11], Exact Separation Logic [23], Local Completeness Logic [6, 7], and Quantitative Hyper Weakest Pre [36] emerged with the ability to reason about both angelic and and demonic nondeterminism in a single framework.

However, none of those logics included the ability to reason about nontermination; while they can be used to prove that programs *sometimes* terminate or *always* diverge, they cannot be used to prove that programs *always* terminate, or *sometimes* diverge. The former is the well-known total correctness property [25]. The latter is useful in real world program analysis, as many codebases have detrimental nontermination bugs that can be found with new static analysis tools [32]. Total Outcome Logic is the only logic that subsumes all of the aforementioned abilities.

In an orthogonal line of work, Cousot [10] described a framework for generating program logics calculationally by composing abstractions. While this framework supports partial and total correctness as well as incorrectness, those abilities do not stem from a single logic, but rather it provides a means for obtaining inference rules for specialized logics.

Weighted Programming. Weighted programming, a paradigm in which each outcome of a program is weighted by an element of a semiring. The initial formulation defined weakest precondition style calculus for deriving the weight of a single outcome. It used an operational semantics based on total semirings, so it could not encoded the bounded models that we have in this paper, such as probabilistic programs. While the operational semantics could identify infinite traces, the semantics was not derived from a fixed point.

Outcome Logic extended the model of weighted programming to partial semirings, to encode more models [37]. Outcome Logic also has the ability to reason about multiple executions in a single derivation, so that it can characterize the weights of many outcomes as once. Quantitative Weakest Hyper Pre [36] is the predicate transformer analogue of Outcome Logic, as weakest preconditions is to Hoare Logic. However, neither Outcome Logic nor Quantitative Weakest Hyper Pre can identify the weight of nonterminating traces.

Powerdomains and Unbounded Nondeterminism. Powerdomains [31, 33] provide a general mechanism for constructing dcpos for powersets. There are three standard powerdomains, all of which have different properties and tradeoffs. These powerdomains arise from lifting a dcpo $\langle X, \leq \rangle$ to a dcpo on $\mathcal{P}(X)$ (the powerset of X) in three different ways using the Hoare, Smyth [33], and Egli-Milner orders below:

$$\begin{array}{lll} S \sqsubseteq_{\mathsf{H}} T & \text{iff} & \forall x \in S.\ \exists y \in T.\ x \leq y \\ S \sqsubseteq_{\mathsf{S}} T & \text{iff} & \forall y \in T.\ \exists x \in S.\ x \leq y \\ S \sqsubseteq_{\mathsf{EM}} T & \text{iff} & S \sqsubseteq_{\mathsf{H}} T \text{ and } S \sqsubseteq_{\mathsf{S}} T \end{array}$$

When applied to a flat poset $\langle \Sigma_{\circlearrowleft}, \leq \rangle$[4], we obtain the following dcpos [34].

$$\begin{array}{rl} \mathsf{Hoare} \triangleq & \langle \{S \cup \{\circlearrowleft\} \mid S \subseteq \Sigma\}, \quad \sqsubseteq_{\mathsf{H}} \rangle \cong \langle \mathcal{P}(\Sigma), \subseteq \rangle \\ \mathsf{Smyth} \triangleq & \langle \mathcal{P}_{\mathsf{fin}}(\Sigma) \cup \{\circlearrowleft\}, \quad \sqsubseteq_{\mathsf{S}} \rangle \\ \mathsf{Plotkin} \triangleq & \langle \mathcal{P}_{\mathsf{fin}}(\Sigma) \cup \{S \subseteq \Sigma \mid \circlearrowleft \in S\}, \sqsubseteq_{\mathsf{EM}} \rangle \end{array}$$

[4] The flat poset has $\circlearrowleft \leq \sigma$ and $\sigma \leq \sigma$ for all $\sigma \in \Sigma_{\circlearrowleft}$, but $\sigma \not\leq \tau$ for all $\sigma \neq \tau$.

The Hoare powerdomain is the only one which allows unrestricted unbounded nondeterminism, however it is not able to distinguish when nontermination may occur (except when there are no terminating outcomes at all), making it a good semantic basis for partial correctness logics. Since $\circlearrowleft$ is always present, the Hoare powerdomain can equivalently be defined as the dcpo on the powerset of Σ ordered by subset inclusion. The semantic model of Outcome Logic [37] can be seen as a generalization of the Hoare powerdomain.

The other two powerdomains do include abilities to distinguish nontermination, and thus do not allow unbounded nondeterminism. This is consistent with the impossibility result of Apt and Plotkin [2] showing that sequential composition cannot be continuous in domains that both distinguish nontermination and also have unbounded branching.

The Smyth powerdomain [33] consists of a distinguished nontermination element, along with all *finite* sets of terminating outcomes. In this construction, if nontermination is possible at all, then we consider the semantics to be $\{\circlearrowleft\}$, so we cannot distinguish a program that sometimes diverges from one that always does. This makes it a good semantic basis for total correctness.

Finally, the Plotkin Powerdomain [31]—derived from the Egli-Milner order—contains all finite subsets of Σ as well as possibly infinite subsets that also contain $\circlearrowleft$. This is reminiscent of our semantic model in this paper, which maximizes the weight of nontermination for infinite sets. Indeed, in the powerset interpretation, the maximum possible weight is 1, with $m(\circlearrowleft) = 1$ meaning that $\circlearrowleft$ is in the set of possible outcomes. As such, our dcpo $\langle \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma), \sqsubseteq \rangle$ can be viewed as a generalization of the Plotkin powerdomain.

7 Conclusion

Recent interest in new program logics for correctness and incorrectness has led to the development of unified reasoning frameworks. However, none of those frameworks included the ability to reason about both termination and nontermination, leaving the picture incomplete. We propose Total Outcome Logic as a logic to derive precise specifications about both terminating and nonterminating traces. Moreover, Total Outcome Logic is generalized over a weighted programming model, meaning that reasoning about termination and nontermination extends to programs with other computational effects, such as probabilistic programs.

It has already been shown that correctness and incorrectness analyses can be unified using Outcome Logic as an underlying theory [40]. Moving from Outcome Logic to Total Outcome Logic we extend the theory to more types of correctness and incorrectness properties. More precisely, TOL enables total correctness specifications (guaranteeing termination), and can be used to witness nontermination, which is of interest in industrial codebases [32]. These improved capabilities make TOL a powerful metatheoretic foundation for program analysis.

Bibliography

[1] Samson Abramsky and Achim Jung. *Domain theory*, page 1–168. Oxford University Press, Inc., USA, 1995. ISBN 019853762X.

[2] K. R. Apt and G. D. Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724–767, aug 1986. ISSN 0004-5411. https://doi.org/10.1145/6490.6494. URL https://doi.org/10.1145/6490.6494.

[3] Ralph-Johan Back. A continuous semantics for unbounded nondeterminism. *Theoretical Computer Science*, 23:187–210, 12 1983. https://doi.org/10.1016/0304-3975(83)90055-5.

[4] Kevin Batz, Adrian Gallus, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Tobias Winkler. Weighted programming, 2022.

[5] Jon Beck. Distributive laws. In B. Eckmann, editor, *Seminar on Triples and Categorical Homology Theory*, pages 119–140, Berlin, Heidelberg, 1969. Springer Berlin Heidelberg. ISBN 978-3-540-36091-9. https://doi.org/10.1007/BFb0083084.

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

[7] Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. A Correctness and Incorrectness Program Logic. *J. ACM*, 70(2), mar 2023. ISSN 0004-5411. https://doi.org/10.1145/3582267.

[8] Stephen A. Cook. Soundness and Completeness of an Axiom System for Program Verification. *SIAM J. Comput.*, 7(1):70–90, feb 1978. ISSN 0097-5397. https://doi.org/10.1137/0207005.

[9] Patrick Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277(1):47–103, 2002. ISSN 0304-3975. https://doi.org/10.1016/S0304-3975(00)00313-3. URL https://www.sciencedirect.com/science/article/pii/S0304397500003133. Static Analysis.

[10] Patrick Cousot. Calculational design of [in]correctness transformational program logics by abstract interpretation. *Proc. ACM Program. Lang.*, 8(POPL), jan 2024. https://doi.org/10.1145/3632849. URL https://doi.org/10.1145/3632849.

[11] Thibault Dardinier and Peter Müller. Hyper hoare logic: (dis-)proving program hyperproperties. *Proc. ACM Program. Lang.*, 8(PLDI), jun 2024. https://doi.org/10.1145/3656437. URL https://doi.org/10.1145/3656437.

[12] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976. ISBN 013215871X.

[13] Jonathan S. Golan. Semirings and their applications. 1999. URL https://api.semanticscholar.org/CorpusID:122727231.

[14] Ashutosh Gupta, Thomas A. Henzinger, Rupak Majumdar, Andrey Rybalchenko, and Ru-Gang Xu. Proving non-termination. *SIGPLAN Not.*, 43 (1):147–158, January 2008. ISSN 0362-1340. https://doi.org/10.1145/1328897.1328459. URL https://doi.org/10.1145/1328897.1328459.

[15] C. A. R. Hoare. Some Properties of Predicate Transformers. *J. ACM*, 25(3): 461–480, Jul 1978. ISSN 0004-5411. https://doi.org/10.1145/322077.322088.

[16] Charles Antony Richard Hoare. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10):576–580, October 1969. ISSN 0001-0782. https://doi.org/10.1145/363235.363259.

[17] Thomas Icard. Beyond almost-sure termination. In Glenn Gunzelmann, Andrew Howes, Thora Tenbrink, and Eddy J. Davelaar, editors, *Proceedings of the 39th Annual Meeting of the Cognitive Science Society, CogSci 2017, London, UK, 16-29 July 2017*. cognitivesciencesociety.org, 2017. URL https://mindmodeling.org/cogsci2017/papers/0430/index.html.

[18] Benjamin Lucien Kaminski. *Advanced weakest precondition calculi for probabilistic programs*. Dissertation, RWTH Aachen University, Aachen, 2019.

[19] Werner Kuich. Algebraic systems and pushdown automata. In *Algebraic foundations in computer science. Essays dedicated to Symeon Bozapalidis on the occasion of his retirement*, pages 228–256. Berlin: Springer, 2011. ISBN 978-3-642-24896-2. https://doi.org/10.1007/978-3-642-24897-9_11.

[20] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977. https://doi.org/10.1109/TSE.1977.229904.

[21] Daniel Larraz, Kaustubh Nimkar, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Proving non-termination using max-smt. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, Cham, 2014. Springer International Publishing.

[22] Christoph Lüth and Neil Ghani. Composing Monads Using Coproducts. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming*, ICFP '02, page 133–144, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581134878. https://doi.org/10.1145/581478.581492.

[23] Petar Maksimović, Caroline Cronjäger, Andreas Lööw, Julian Sutherland, and Philippa Gardner. Exact Separation Logic: Towards Bridging the Gap Between Verification and Bug-Finding. In *37th European Conference on Object-Oriented Programming (ECOOP 2023)*, volume 263 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:27, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-281-5. https://doi.org/10.4230/LIPIcs.ECOOP.2023.19.

[24] Ernest G. Manes. *Algebraic Theories*. Springer New York, 1976. ISBN 9781461298601. https://doi.org/10.1007/978-1-4612-9860-1. URL http://dx.doi.org/10.1007/978-1-4612-9860-1.

[25] Zohar Manna and Amir Pnueli. Axiomatic Approach to Total Correctness of Programs. *Acta Inf.*, 3(3):243–263, sep 1974. ISSN 0001-5903. https://doi.org/10.1007/BF00288637.

[26] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005. ISBN 9780387401157. https://doi.org/10.1007/b138392.

[27] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A New Proof Rule for Almost-Sure Termination. *Proc. ACM Program. Lang.*, 2(POPL), Jan 2018. https://doi.org/10.1145/3158121. URL https://doi.org/10.1145/3158121.

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

[29] Bernhard Möller, Peter O'Hearn, and Tony Hoare. 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*, page 325–343, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-88700-1. https://doi.org/10.1007/978-3-030-88701-8_20.

[30] Peter W. O'Hearn. Incorrectness Logic. *Proc. ACM Program. Lang.*, 4 (POPL), January 2020. https://doi.org/10.1145/3371078.

[31] Gordon Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976. https://doi.org/10.1137/0205035. URL https://doi.org/10.1137/0205035.

[32] Azalea Raad, Julien Vanegue, and Peter O'Hearn. Non-termination proving at scale. *Proc. ACM Program. Lang.*, 8(OOPSLA2), October 2024. https://doi.org/10.1145/3689720. URL https://doi.org/10.1145/3689720.

[33] Michael Smyth. Power domains. *Journal of Computer and System Sciences*, 16(1):23–36, 1978. ISSN 0022-0000. https://doi.org/https://doi.org/10.1016/0022-0000(78)90048-X. URL https://www.sciencedirect.com/science/article/pii/002200007890048X.

[34] Harald Søndergaard and Peter Sestoft. Non-determinism in Functional Languages. *The Computer Journal*, 35(5):514–523, 10 1992. ISSN 0010-4620. https://doi.org/10.1093/comjnl/35.5.514. URL https://doi.org/10.1093/comjnl/35.5.514.

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

[36] Linpeng Zhang, Noam Zilberstein, Benjamin Lucien Kaminski, and Alexandra Silva. Quantitative weakest hyper pre: Unifying correctness and incorrectness hyperproperties via predicate transformers. *Proc. ACM Program. Lang.*, 8(OOPSLA2), oct 2024. https://doi.org/10.1145/3689740. URL https://doi.org/10.1145/3689740.

[37] Noam Zilberstein. A relatively complete program logic for effectful branching, 2024. URL https://arxiv.org/abs/2401.04594.

[38] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc.*

ACM Program. Lang., 7(OOPSLA1), Apr 2023. https://doi.org/10.1145/3586045.

[39] Noam Zilberstein, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. A demonic outcome logic for randomized nondeterminism, 2024. URL https://arxiv.org/abs/2410.22540.

[40] Noam Zilberstein, Angelina Saliling, and Alexandra Silva. Outcome separation logic: Local reasoning for correctness and incorrectness with computational effects. *Proc. ACM Program. Lang.*, 8(OOPSLA1), Apr 2024. https://doi.org/10.1145/3649821.

Appendix

A Domain Properties

Here, we prove a few preliminary results on our semantic domain $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$. The first of these describes how the Kleisli extension $(-)^\dagger : (X \to \mathcal{W}_{\mathcal{A}}(Y)) \to \mathcal{W}_{\mathcal{A}}(X) \to \mathcal{W}_{\mathcal{A}}(Y)$ interacts with $+$, scalar multiplication $\cdot$, and divergence:

Lemma 1. *For any $f, g \in (X \to \mathcal{W}_{\mathcal{A}}(Y))$, $m, m' \in \mathcal{W}_{\mathcal{A}}(X))$, and $u \in U$:*

1. $f^\dagger(m + m') = f^\dagger(m) + f^\dagger(m')$
2. *If* $\mathsf{supp}(m) \subseteq \Sigma$, $(f + g)^\dagger(m) = f^\dagger(m) + g^\dagger(m)$
3. $f^\dagger(u \cdot m) = u \cdot f^\dagger(m)$
4. *If* $\mathsf{supp}(m) \subseteq \Sigma$, $f^\dagger(m \cdot u) = f^\dagger(m) \cdot u$
5. $f^\dagger(\eta(\circlearrowleft)) = \eta(\circlearrowleft)$

Proof. (1), (3), and (4) are left to the reader. We show:

(2) If $\mathsf{supp}(m) \subseteq \Sigma$, then $m(\circlearrowleft) = 0$. We have:

$$\begin{aligned}(f+g)^\dagger(m) &= \left(\sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot (f+g)(\sigma)\right) + m(\circlearrowleft) \cdot \eta(\circlearrowleft) \\ &= \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot (f(\sigma) + g(\sigma)) \\ &= \left(\sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot f(\sigma)\right) + \left(\sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot g(\sigma)\right) \\ &= f^\dagger(m) + g^\dagger(m)\end{aligned}$$

(5) Note that $\mathsf{supp}(\eta(\circlearrowleft)) = \{\circlearrowleft\}$, so $\mathsf{supp}(\eta(\circlearrowleft)) \cap \Sigma = \emptyset$. Then:

$$\begin{aligned}f^\dagger(\eta(\circlearrowleft)) &= \left(\sum_{\sigma \in \mathsf{supp}(\eta(\circlearrowleft)) \cap \Sigma} m(\sigma) \cdot f(\sigma)\right) + \eta(\circlearrowleft)(\circlearrowleft) \cdot \eta(\circlearrowleft) \\ &= \eta(\circlearrowleft)(\circlearrowleft) \cdot \eta(\circlearrowleft) \\ &= \eta(\circlearrowleft)\end{aligned}$$

B Totality of Language Semantics

B.1 $\mathcal{D}$-Closure

To accommodate both a continuous semantics and a (countably) infinite state set, it was necessary to restrict our domain to a subset $\mathcal{D}$ of weighting functions. In particular, we identified two possible classes of semantics with corresponding restrictions on the semiring of weights $\mathcal{A} = \langle X, +, \cdot, 0, 1\rangle$ used:

- ▷ **Conservative**: $\mathcal{A}$ is partial and bounded. Then, we take $\mathcal{D} \triangleq \{m \in \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma) \mid |m| = \top\}$.
- ▷ **Indicative**: $\mathcal{A}$ is bounded with strongly-infinite top element $\top$. For any other semiring, we can adjoin an element $w \notin X$ and extend $+$ and $\cdot$ such that w is a strongly-infinite upper bound:
  - $w + x = x + w = w$ for all $x \in X$;
  - $w \cdot x = x \cdot w = w$ for all $x \in X/\{0\}$;
  - $w \cdot 0 = 0 \cdot w = 0$.

  In this case, take $\mathcal{D} \triangleq \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma)$.

In this section we demonstrate that the program semantics is closed in $\mathcal{D}$ for each of the classes described above.

Conservative

Lemma 2. *For any collection $(x_i \in X)_{i \in I}$ such that $\sum_{i \in I} x_i = \top$, if $(m_i)_{i \in I}$ is a family of weighting functions with $|m_i| = \top$ for each $i \in I$, then $\left|\sum_{i \in I} x_i \cdot m_i\right| = \top$.*

Proof. It holds that

$$\left|\sum_{i \in I} x_i \cdot m_i\right| = \sum_{i \in I} x_i \cdot |m_i| = \sum_{i \in I} x_i \cdot T = \left(\sum_{i \in I} x_i\right) \cdot \top = \top \cdot \top = \top$$

Lemma 3. *For any $m \in \mathcal{D}$ and $f \in \Sigma \to \mathcal{D}$, $\mathsf{bind}(m, f) \in \mathcal{D}$.*

Proof. Recall that $\mathcal{D} \subseteq \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma) = \mathcal{W}^{+}_{\mathcal{A}}(\Sigma) \cup \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$. First, suppose that $m, f(\sigma) \in \mathcal{W}^{+}_{\mathcal{A}}(\Sigma)$ (i.e. m and $f(\sigma)$ have finite support) for all $\sigma \in \mathsf{supp}(m) \cap \Sigma$. Recall:

$$\mathsf{bind}(m, f) = m(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft) + \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot f(\sigma)$$

So,

$$\mathsf{supp}(\mathsf{bind}(m, f)) = (\mathsf{supp}(m) \cap \{\circlearrowleft\}) \cup \bigcup_{\sigma \in \mathsf{supp}(m) \cap \Sigma} \mathsf{supp}(f(\sigma))$$

Since $\mathsf{supp}(m)$ is finite, and $\mathsf{supp}(f(\sigma))$ is finite for each $\sigma \in \mathsf{supp}(m) \cap \Sigma$, the above expression consists of a finite union of finite sets, which is itself finite. Thus, $\mathsf{bind}(m, f) \in \mathcal{W}^{+}_{\mathcal{A}}(\Sigma) \subseteq \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma)$.

Next, we consider the case in which $m \in \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$ or $f(\sigma) \in \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$ for some $\tau \in \mathsf{supp}(m) \cap \Sigma$. We split our analysis across the two weighting schemes defined (and, accordingly, the two definitions for $\mathcal{D}$ provided for each):

- ▷ **Conservative Weighting**: $\mathcal{D} = \{m \in \mathcal{W}^{\infty}_{\mathcal{A}} \mid |m| = \top\}$.

  Since $m, f(\sigma) \in \mathcal{D}$ for every $\sigma \in \Sigma$, it holds that

- $(m(\sigma) \in X)_{\sigma \in \mathsf{supp}(m)}$ is a family of weights such that $\left|\sum_{\sigma \in \mathsf{supp}(m)} m(\sigma)\right| = |m| = \top$;
- $|f(\sigma)| = \top$ for each $\sigma \in \mathsf{supp}(m) \cap \Sigma$;
- $|\top \cdot \mathsf{unit}(\circlearrowleft)| = \top$.

Thus, by lemma 2, $|\mathsf{bind}(m, f)| = \top$.

▷ **Indicative Weighting**: $\mathcal{D} = \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma)$.

If $m \in \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$, then $m(\circlearrowleft) = \sup_{m' \in E(m)} m'(\circlearrowleft) = \top$, indicating the presence of nontermination. As a strong infinity, $\top$ is absorbing and thus propagates through $\mathsf{bind}(m, f)$:

$$\begin{aligned}
\mathsf{bind}(m, f)(\circlearrowleft) &= m(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft)(\circlearrowleft) + \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot f(\sigma)(\circlearrowleft) \\
&= \top \cdot \top \cdot 1 + \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot f(\sigma)(\circlearrowleft) \\
&= \top + \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot f(\sigma)(\circlearrowleft) \\
&= \top = \sup_{m' \in E(\mathsf{bind}(m,f))} m'(\circlearrowleft)
\end{aligned}$$

Similarly, if $f(\tau) \in \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$ for some $\tau \in \mathsf{supp}(m) \cap \Sigma$:

$$\begin{aligned}
\mathsf{bind}(m, f)(\circlearrowleft) &= m(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft)(\circlearrowleft) + \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot f(\sigma)(\circlearrowleft) \\
&= \big(m(\circlearrowleft) \cdot \top \cdot 1\big) + \big(m(\tau) \cdot f(\tau)(\circlearrowleft)\big) + \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma / \{\tau\}} m(\sigma) \cdot f(\sigma)(\circlearrowleft) \\
&= \big(m(\circlearrowleft) \cdot \top \cdot 1\big) + \big(m(\tau) \cdot \top\big) + \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma / \{\tau\}} m(\sigma) \cdot f(\sigma)(\circlearrowleft)
\end{aligned}$$

Since $\tau \in \mathsf{supp}(m)$, we know $m(\tau) \neq 0$, so $m(\tau) \cdot \top = \top$:

$$\begin{aligned}
&= \big(m(\circlearrowleft) \cdot \top \cdot 1\big) + \top + \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma / \{\tau\}} m(\sigma) \cdot f(\sigma)(\circlearrowleft) \\
&= \top = \sup_{m' \in E(\mathsf{bind}(m,f))} m'(\circlearrowleft)
\end{aligned}$$

Therefore, $\mathsf{bind}(m, f) \in \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma) \subseteq \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma) = \mathcal{D}$.

B.2 Fixpoint Existence

Theorem 6 (Scott-Continuity of $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$). *Suppose $\mathcal{A}$ is a partial semiring which is bounded, finitary, Scott-continuous, and lower Scott-continuous. Then, $+ : \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})^2 \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ on weighting functions is Scott-continuous with respect to the **fusion order** $\sqsubseteq$. That is, for any directed subset $D \subseteq \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$,*

$$\sup_{m_1 \in D} (m_1 + m_2) = \sup D + m_2$$

Proof. Let $\sigma \in \Sigma$. Then,

$$\begin{aligned}
\sup_{m_1 \in D} ((m_1 + m_2)(\sigma)) &= \sup_{m_1 \in D} (m_1(\sigma) + m_2(\sigma)) \\
&= \sup_{m_1 \in D} (m_1(\sigma)) + m_2(\sigma) && \text{(Scott-continuity of } \mathcal{A}) \\
&= \big(\sup_{m_1 \in D} m_1(\sigma) \big) + m_2(\sigma)
\end{aligned}$$

Dually,

$$\begin{aligned}
\inf_{m_1 \in D} ((m_1 + m_2)(\circlearrowleft)) &= \inf_{m_1 \in D} (m_1(\circlearrowleft) + m_2(\circlearrowleft)) \\
&= \inf_{m_1 \in D} (m_1(\circlearrowleft) + m_2(\circlearrowleft) && \text{(Lower Scott-continuity of } \mathcal{A}) \\
&= \big(\inf_{m_1 \in D} m_1(\circlearrowleft) \big) + m_2(\circlearrowleft)
\end{aligned}$$

Now, since fusion order extends $\leq$ pointwise on Σ and $\geq$ on $\circlearrowleft$, we have that $\sup_{m_1 \in D}(m_1 + m_2) = \sup D + m_2$.

Lemma 4 (Joint Continuity). *Suppose X is a dcpo. For any function $f : X \times X \to X$ that is Scott-continuous in the variables separately, f must be continuous in the variables jointly as well. I.e. for directed subsets $D_1, D_2 \subseteq X$,*

$$\sup_{x \in D_1, y \in D_2} f(x, y) = f(\sup D_1, \sup D_2)$$

Proof. Since f is continuous separately in the first and second variables:

$$f(\sup D_1, \sup D_2) = \sup_{x \in D_1} f(x, \sup D_2) = \sup_{x \in D_1} \sup_{y \in D_2} f(x, y) = \sup_{x \in D_1, y \in D_2} f(x, y)$$

Lemma 5. *Let $\langle X, +, \cdot, 0, 1 \rangle$ be a finitary (complete), continuous, partial semiring. For any family of Scott-continuous functions $(f_i : X \to X)_{i \in I}$ and directed set $D \subseteq X$:*

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

Proof. Since each f_i is Scott-continuous, we have $\{f_i(x) \mid x \in D\}$ is a directed set. We now proceed by induction on I.

- ▷ Base Case: $I = \{i_1\}$. Then, we only need to show $\sup_{x\in D} f_{i_1}(x) = f_{i_1}(\sup D)$, which follows from the Scott-continuity of f_{i_1}.
- ▷ Successor Case: Suppose the claim holds for all set smaller than I, for I finite. We can partition I into disjoint non-empty parts I_1 and I_2, with $I = I_1 \cup I_2$, $I_1 \cap I_2 = \emptyset$, and $I_1, I_2 \neq \emptyset$. So,

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

  By the induction hypothesis, each of $\lambda x.\ \sum_{i\in I_1} f_i(x)$ and $\lambda x.\ \sum_{i\in I_2} f_i(x)$ are continuous, so $\{\sum_{i\in I_1} f_i(x) \mid x \in D\}$ and $\{\sum_{i\in I_2} f_i(x) \mid x \in D\}$ are directed. We apply joint continuity of $+ : X^2 \to X$ to get:

$$\begin{aligned} &= \sup_{x\in D} \sum_{i\in I_1} f_i(x) + \sup_{x\in D} \sum_{i\in I_2} f_i(x) \\ &= \sum_{i\in I_1} f_i(\sup D) + \sum_{i\in I_2} f_i(\sup D) && \text{(IH)} \\ &= \sum_{i\in I} f_i(\sup D) \end{aligned}$$

- ▷ Limit Case: Suppose the claim holds for all finite indexing sets. Since the semiring is finitary:

$$\begin{aligned} \sup_{x\in D} \sum_{i\in I} f_i(x) &= \sup_{x\in D} \left(\sup_{J\subseteq I,\ J \text{ finite}} \sum_{i\in J} f_i(x) \right) \\ &= \sup_{J\subseteq I,\ J \text{ finite}} \left(\sup_{x\in D} \sum_{i\in J} f_i(x) \right) \\ &= \sup_{J\subseteq I,\ J \text{ finite}} \left(\sum_{i\in J} f_i(\sup D) \right) \\ &= \sum_{i\in I} f_i(\sup D) \end{aligned}$$

Corollary 1. *For any **finite** family of Scott-continuous functions $\left(f_i : \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})\right)_{i\in I}$ and directed set $D \subseteq \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$:*

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

Proof. See the proof of 5 up to the successor case.

Lemma 6. *For any $\sigma \in \Sigma$, the function $g(f) = m(\sigma)\cdot f(\sigma)(\tau)$ is Scott-continuous in the semiring $\mathcal{A}$.*

Proof.

$$\sup_{f \in D} g(f) = \sup_{f \in D} m(\sigma) \cdot f(\sigma)(\tau)$$

By Scott-continuity of the $\cdot$ operator in the semiring:

$$= m(\sigma) \cdot \sup_{f \in D} f(\sigma)(\tau)$$

Since we are using the pointwise* ordering:

$$= m(\sigma) \cdot (\sup D)(\sigma)(\tau) = g(\sup D)$$

Corollary 2. *For $D \subseteq (\Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}))$ directed,*

$$\sup_{f \in D} \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot (\sup D)(\sigma)(\tau) = \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot (\sup D)(\sigma)(\tau)$$

Proof. By Lemma 5.

Lemma 7. *If $m \in \mathcal{D}$ and $f \in (\Sigma \to \mathcal{D})$, $\mathsf{bind}(\llbracket C \rrbracket(\sigma), f)$ is Scott-continuous with respect to its second argument f.*

Proof. Take $D \subseteq \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ directed. Let $\sigma \in \Sigma$ and $\llbracket C \rrbracket(\sigma) \in \mathcal{D}$. Recall that $\mathcal{D} \subseteq \mathcal{W}^{\infty}_{\mathcal{A}}(\Sigma) = \mathcal{W}^{+}_{\mathcal{A}}(\Sigma) \cup \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$, so we can split our analysis across two cases:

▷ Suppose $\llbracket C \rrbracket(\sigma) \in \mathcal{W}^{+}_{\mathcal{A}}(\Sigma)$. Then, $\mathsf{supp}(\llbracket C \rrbracket(\sigma))$ is finite, as is $\mathsf{supp}(\llbracket C \rrbracket(\sigma) \cap \Sigma$.

$$\begin{aligned}
&\sup_{f \in D} \mathsf{bind}(\llbracket C \rrbracket(\sigma), f) \\
&= \sup_{f \in D} \left(\llbracket C \rrbracket(\sigma)(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft) + \sum_{\tau \in \mathsf{supp}(\llbracket C \rrbracket(\sigma) \cap \Sigma)} \llbracket C \rrbracket(\sigma)(\tau) \cdot f(\tau) \right) \\
&= \llbracket C \rrbracket(\sigma)(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft) + \sup_{f \in D} \left(\sum_{\tau \in \mathsf{supp}(\llbracket C \rrbracket(\sigma) \cap \Sigma)} \llbracket C \rrbracket(\sigma)(\tau) \cdot f(\tau) \right)
\end{aligned}$$

Since the above summation is finite (by assumption), we can apply (Corollary):

$$\begin{aligned}
&= \llbracket C \rrbracket(\sigma)(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft) + \sum_{\tau \in \mathsf{supp}(\llbracket C \rrbracket(\sigma)) \cap \Sigma} \llbracket C \rrbracket(\sigma)(\tau) \cdot (\sup D)(\tau) \\
&= \mathsf{bind}(\llbracket C \rrbracket(\sigma), \sup D)
\end{aligned}$$

▷ Otherwise, $[\![C]\!](\sigma) \in \mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$. We know that for any program state $\tau' \in \Sigma$,

$$\sup_{f \in D} \mathsf{bind}([\![C]\!](\sigma), f)(\tau')$$

$$= \sup_{f \in D} \left([\![C]\!](\sigma)(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft)(\tau') + \sum_{\tau \in \mathsf{supp}([\![C]\!](\sigma)) \cap \Sigma} [\![C]\!](\sigma)(\tau) \cdot f(\tau)(\tau') \right)$$

$$= [\![C]\!](\sigma)(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft)(\tau') + \sup_{f \in D} \left(\sum_{\tau \in \mathsf{supp}([\![C]\!](\sigma)) \cap \Sigma} [\![C]\!](\sigma)(\tau) \cdot f(\tau)(\tau') \right)$$

$$= [\![C]\!](\sigma)(\circlearrowleft) \cdot \top \cdot \mathsf{unit}(\circlearrowleft)(\tau') + \sum_{\tau \in \mathsf{supp}([\![C]\!](\sigma)) \cap \Sigma} [\![C]\!](\sigma)(\tau) \cdot (\sup D)(\tau)(\tau')$$

$$= \mathsf{bind}([\![C]\!](\sigma), \sup D)(\tau')$$

In other words, $\sup_{f \in D} \mathsf{bind}([\![C]\!](\sigma), f) \in E(\mathsf{bind}([\![C]\!](\sigma), \sup D))$, so by the definition of $\mathcal{W}^{\omega}_{\mathcal{A}}(\Sigma)$,

$$\sup_{f \in D} \mathsf{bind}([\![C]\!](\sigma), f)(\circlearrowleft) \sqsubseteq \mathsf{bind}([\![C]\!](\sigma), \sup D)(\circlearrowleft)$$

Conversely, $\mathsf{bind}([\![C]\!](\sigma), \sup D) \in E(\sup_{f \in D} \mathsf{bind}([\![C]\!](\sigma), f))$ implies

$$\sup_{f \in D} \mathsf{bind}([\![C]\!](\sigma), f)(\circlearrowleft) \sqsupseteq \mathsf{bind}([\![C]\!](\sigma), \sup D)(\circlearrowleft)$$

Thus, $\sup_{f \in D} \mathsf{bind}([\![C]\!](\sigma), f)(\circlearrowleft) = \mathsf{bind}([\![C]\!](\sigma), \sup D)(\circlearrowleft)$.

Lemma 8. *Let $\Phi_{\langle C, e_1, e_2 \rangle}(f)(\sigma) = [\![e_1]\!](\sigma) \cdot \mathsf{bind}_{\mathsf{nt}}([\![C]\!](\sigma), f) + [\![e_2]\!](\sigma) \cdot \mathsf{unit}(\sigma)$ and suppose that it is a total function, then $\Phi_{C^{\langle e_1, e_2 \rangle}}$ is Scott continuous with respect to the pointwise order: $f_1 \sqsubseteq f_2$ iff $f_1(\sigma) \sqsubseteq f_2(\sigma)$ for all $\sigma \in \Sigma$.*

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

$$\sup_{f \in D} \Phi_{C^{\langle e_1, e_2 \rangle}}(f)(\sigma) = \sup_{f \in D} \left([\![e_1]\!](\sigma) \cdot \mathsf{bind}_{\mathsf{nt}}([\![C]\!](\sigma), f) + [\![e_2]\!](\sigma) \cdot \mathsf{unit}(\sigma) \right)$$

By the continuity of $+$ and $\cdot$ in $\mathcal{W}_{\mathcal{A}}$:

$$= [\![e_1]\!](\sigma) \cdot \left(\sup_{f \in D} \mathsf{bind}_{\mathsf{nt}}([\![C]\!](\sigma), f) \right) + [\![e_2]\!](\sigma) \cdot \mathsf{unit}(\sigma)$$

By Lemma 6 (the previous Lemma):

$$= [\![e_1]\!](\sigma) \cdot \mathsf{bind}_{\mathsf{nt}}([\![C]\!](\sigma), \sup D) + [\![e_2]\!](\sigma) \cdot \mathsf{unit}(\sigma)$$
$$= \Phi_{C^{\langle e_1, e_2 \rangle}}(\sup D)(\sigma)$$

As this holds for all $\sigma \in \Sigma$, we also have that

$$\sup_{f \in D} \Phi_{C^{\langle e_1, e_2 \rangle}}(f) = \Phi_{\langle C, e_1, e_2 \rangle}(\sup D)$$

C Subsumption of Program Logics

Here, we prove the results of Section 3.3 on the subsumption of classical program logics. For this, recall that we limit ourselves to a nondeterministic interpretation of TOL, instantiated on the Boolean semiring. (See: .)

First, we show that the encodings of modalities $\Box$ and $\Diamond$ of Dynamic Logic in TOL are DeMorgan duals:

Lemma 9 (Modal Duality).

$$\Diamond P = \neg\Box\neg P \quad \textit{and} \quad \Box P = \neg\Diamond\neg P$$

Proof.

$$\begin{aligned}\neg\Box\neg P &= \neg\{m \mid \mathsf{supp}(m) \subseteq (\neg P)_{\circlearrowleft}\}\\ &= \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})/\{m \mid \mathsf{supp}(m) \subseteq (\neg P)_{\circlearrowleft}\}\\ &= \{m \mid \mathsf{supp}(m) \not\subseteq (\neg P)_{\circlearrowleft}\}\\ &= \{m \mid \mathsf{supp}(m) \cap P \neq \emptyset\} \quad = \Diamond P\end{aligned}$$

$$\begin{aligned}\neg\Diamond\neg P &= \neg\{m \mid \mathsf{supp}(m) \cap \neg P \neq \emptyset\}\\ &= \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})/\neg\{m \mid \mathsf{supp}(m) \cap \neg P \neq \emptyset\}\\ &= \{m \mid \mathsf{supp}(m) \cap \neg P = \emptyset\}\\ &= \{m \mid \mathsf{supp}(m) \subseteq P_{\circlearrowleft}\} \quad = \Box P\end{aligned}$$

Moreover, the alternative modalities $\blacksquare$ and $\blacklozenge$ exhibit the same duality. Proof of this is nearly identical to the above, and so is omitted.

Theorems 1, 2, 3 state that the triples of Hoare, Lisbon, and Total Hoare logic can be encoded in TOL using the above modal operators.

Theorem 1 (Subsumption of Hoare Logic).

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

Proof. It is a well-known result that $P \subseteq [C]Q$ iff $\{P\}\ C\ \{Q\}$. We show the first equivalence: $\langle P\rangle\ C\ \langle \Box Q\rangle$ iff $P \subseteq [C]Q$.

$\Longrightarrow$: Suppose $\sigma \in P$. Then, $\eta(\sigma) \vDash P$. We assume $\langle P\rangle\ C\ \langle Q\rangle$, so

$$[\![C]\!]^{\dagger}(\eta(\sigma)) \vDash \Box Q \;=\; \exists(u,v) \in U^2.\, Q^{(u)} \oplus \Uparrow^{(v)} \;=\; \{m \mid \mathsf{supp}(m) \subseteq Q_{\circlearrowleft}\}$$

That is, $[\![C]\!](\sigma) \subseteq Q_{\circlearrowleft}$. By definition $\sigma \in [C]Q$.

$\Longleftarrow$: We assume $P \subseteq [C]Q$. Suppose $m \vDash P$, which means $|m| = 1$, $\mathsf{supp}(m) \subseteq P \subseteq [C]Q$. Note that $\circlearrowleft \notin \mathsf{supp}(m)$. Then, since $\mathsf{supp}([\![C]\!](\sigma)) \subseteq Q_{\circlearrowleft}$ for all $\sigma \in \mathsf{supp}(m)$, we have

$$\mathsf{supp}([\![C]\!]^{\dagger}(m)) = \bigcup_{\sigma \in \mathsf{supp}(m)} \mathsf{supp}([\![C]\!](\sigma)) \quad \subseteq \quad Q_{\circlearrowleft}$$

By definition of $\Box$, $[\![C]\!]^{\dagger}(m) \vDash \Box Q$.

Theorem 2 (Subsumption of Lisbon Logic).

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

Proof. Again, $P\langle C\rangle Q$ iff $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$ is an established result. We show $\langle P\rangle\ C\ \langle \Diamond Q\rangle$ iff $P \subseteq \langle C\rangle Q$.

$\Longrightarrow$: Suppose $\sigma \in P$. Then, $\eta(\sigma) \vDash P$. By $\langle P\rangle\ C\ \langle \Diamond Q\rangle$, we have that

$$[\![C]\!]^\dagger(\eta(\sigma)) \vDash \Diamond Q \;=\; \exists u : U/\{\mathbb{0}\}.\, Q^{(u)} \oplus \top \;=\; \{m \mid \mathsf{supp}(m) \cap Q \neq \emptyset\}$$

This means $\mathsf{supp}([\![C]\!]\,(\sigma) \cap Q \neq \emptyset)$, so $\sigma \in \langle C\rangle Q$ by definition.

$\Longleftarrow$: Assume $P \subseteq \langle Q\rangle$. Suppose $m \vDash P^{(1)}$, i.e. $|m| = 1$, $\mathsf{supp}(m) \subseteq P \subseteq \langle C\rangle Q$. Note that $\circlearrowleft \notin \mathsf{supp}(m)$. So, for any $\sigma \in \mathsf{supp}(m)$, $\mathsf{supp}([\![C]\!]\,(\sigma)) \cap Q \neq \emptyset$. It holds that

$$\begin{aligned}\mathsf{supp}\left([\![C]\!]^\dagger(m)\right) \cap Q &= \left(\bigcup_{\sigma\in\mathsf{supp}(m)} \mathsf{supp}([\![C]\!]\,(\sigma))\right) \cap Q \\ &= \bigcup_{\sigma\in\mathsf{supp}(m)} \mathsf{supp}([\![C]\!]\,(\sigma)) \cap Q \quad \neq \emptyset\end{aligned}$$

By definition, $[\![C]\!]^\dagger(m) \vDash \Diamond Q$.

Theorem 3 (Subsumption of Total Hoare Logic).

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

Proof. We show $\vDash \langle P\rangle\ C\ \langle \blacksquare Q\rangle$ iff $P \subseteq [C]^\star Q$.

$\Longrightarrow$: Suppose $\sigma \in P$. Then, $\eta(\sigma) \vDash P$ and since $\langle P\rangle\ C\ \langle \blacksquare Q\rangle$,

$$[\![C]\!]^\dagger(\eta(\sigma)) \vDash \blacksquare Q = \exists u : U/\{\mathbb{0}\}.\, Q^{(u)} = \{m \mid \mathsf{supp}(m) \subseteq Q\}$$

That is, $\mathsf{supp}([\![C]\!]\,(\sigma)) \subseteq Q$, so we have $\sigma \in [C]^\star Q$.

$\Longleftarrow$: Suppose $m \vDash P$, i.e. $|m| = 1$ and $\mathsf{supp}(m) \subseteq P \subseteq [C]^\star Q$. Note that $\circlearrowleft \notin \mathsf{supp}(m)$, and

$$\mathsf{supp}\left([\![C]\!]^\dagger(m)\right) = \bigcup_{\sigma\in\mathsf{supp}(m)} \mathsf{supp}([\![C]\!]\,(\sigma))$$

For each $\sigma \in \mathsf{supp}(m)$, $\mathsf{supp}([\![C]\!]\,(\sigma)) \subseteq Q$. This gives us that $[\![C]\!]^\dagger(m) \vDash \blacksquare Q$, as desired.

D Soundness and Relative Completeness

First, we prove some preliminary results on the semantics of iteration. Recall that the semantics for a command $C^{\langle e,e'\rangle}$ is given in terms of the characteristic function $\Phi_{\langle C,e,e'\rangle} : (\Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})) \to \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$:

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

The following Lemma states that applications of $\Phi_{\langle C,e,e'\rangle}$ starting on $\bot$ are equivalent to *unrolling* $C^{\langle e,e'\rangle}$ a corresponding number of times.

Lemma 10. *For all $n \in \mathbb{N}$, $\sigma \in \Sigma$, and $\tau \in \Sigma_{\circlearrowleft}$,* [5]

$$\Phi^{n+1}_{\langle C,e,e'\rangle}(\bot)(\sigma)(\tau) = \begin{cases} \sum_{k=0}^{n} [\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!](\sigma)(\tau) & \text{if } \tau \in \Sigma \\ \bot^\dagger([\![(\mathsf{assume}\ e \fatsemi C)^{n+1}]\!](\sigma))(\circlearrowleft) & \text{if } \tau = \circlearrowleft \end{cases}$$

Note: For a command C, we define $C^0 \triangleq \mathsf{skip}$ and $C^{n+1} \triangleq C^n \fatsemi C$.

Proof. We proceed by induction on n.

▷ $n = 0$. We have that

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

Now, if $\tau \in \Sigma$, then $\bot^\dagger([\![\mathsf{assume}\ e \fatsemi C)]\!](\sigma))(\tau) = \mathbb{0}$, since $\mathsf{supp}(\bot^\dagger([\![\mathsf{assume}\ e \fatsemi C]\!](\sigma)) = \{\circlearrowleft\}$. On the other hand, $[\![\mathsf{assume}\ e']\!](\sigma)(\circlearrowleft) = \mathbb{0}$. This gives us the two cases, as desired.

▷ *Inductive step.* Suppose the claim holds for n. First,

$$\begin{aligned} \Phi^{n+2}_{\langle C,e,e'\rangle}(\bot)(\sigma) &= [\![e]\!](\sigma) \cdot \big(\Phi^{n+1}_{\langle C,e,e'\rangle}(\bot)\big)^\dagger\big([\![C]\!](\sigma)\big) + [\![e']\!](\sigma) + \eta(\sigma) \\ &= [\![e]\!](\sigma) \cdot \left(\sum_{\tau' \in \mathsf{supp}([\![C]\!](\sigma))} [\![C]\!](\sigma)(\tau') \cdot \Phi^{n+1}_{\langle C,e,e'\rangle}(\bot)(\tau')\right) + [\![e']\!](\sigma) \cdot \eta(\sigma) \end{aligned}$$

Now, for $\tau \in \Sigma$, the induction hypothesis gives us:

$$\Phi^{n+2}_{\langle C,e,e'\rangle}(\bot)(\sigma)(\tau)$$

$$= [\![e]\!](\sigma) \cdot \left(\sum_{\tau' \in \mathsf{supp}[\![C]\!](\sigma)} [\![C]\!](\sigma)(\tau') \cdot \sum_{k=0}^{n} \Big[\!\Big[(\mathsf{assume}\ e \fatsemi C)^k \fatsemi \mathsf{assume}\ e'\Big]\!\Big](\tau')(\tau)\right) + [\![e']\!](\sigma) \cdot \eta(\sigma)(\tau)$$

$$= \left(\sum_{\tau' \in \mathsf{supp}([\![C]\!](\sigma))} \sum_{k=0}^{n} [\![e]\!](\sigma) \cdot [\![C]\!](\sigma)(\tau') \cdot \Big[\!\Big[(\mathsf{assume}\ e \fatsemi C)^k \fatsemi \mathsf{assume}\ e'\Big]\!\Big](\tau')(\tau)\right) + [\![e']\!](\sigma) \cdot \eta(\sigma)(\tau)$$

[5] Our semantics must capture two computational effects – weighted execution and nontermination – each of which compose differently when programs are sequenced. As a result, the unrolling is expressed differently depending on whether we want the final weight on a program state in Σ or the final weight of the nontermination outcome. In the case of the former, we aggregate the weight collected for traces terminating within the first n iterations. In the latter, we push any weight of traces not yet terminated (including any divergent weight from potential inner loops) onto $\circlearrowleft$.

$$= \left(\sum_{k=0}^{n} \sum_{\tau' \in \mathsf{supp}([\![C]\!](\sigma))} [\![e]\!](\sigma) \cdot [\![C]\!](\sigma)(\tau') \cdot \left[\!\left[(\mathsf{assume}\ e \fatsemi C)^k \fatsemi \mathsf{assume}\ e' \right]\!\right](\tau')(\tau) \right) + [\![e']\!](\sigma) \cdot \eta(\sigma)(\tau)$$

$$= \left(\sum_{k=1}^{n+1} \left[\!\left[(\mathsf{assume}\ e \fatsemi C)^k \fatsemi \mathsf{assume}\ e' \right]\!\right](\sigma)(\tau) \right) + [\![\mathsf{assume}\ e']\!](\sigma)(\tau)$$

$$= \sum_{k=0}^{n+1} \left[\!\left[(\mathsf{assume}\ e \fatsemi C)^k \fatsemi \mathsf{assume}\ e' \right]\!\right](\sigma)(\tau)$$

Applying the other case of the induction hypothesis:

$$\Phi^{n+2}_{\langle C,e,e' \rangle}(\bot)(\sigma)(\lightning)$$

$$= [\![e]\!](\sigma) \cdot \left(\sum_{\tau' \in \mathsf{supp}([\![C]\!](\sigma))} [\![C]\!](\sigma)(\tau') \cdot \bot^\dagger([\![(\mathsf{assume}\ e \fatsemi C)^{n+1}]\!](\tau'))(\lightning) \right) + [\![e']\!](\sigma) \cdot \eta(\sigma)(\lightning)$$

$$= \bot^\dagger([\![(\mathsf{assume}\ e \fatsemi C)^{n+2}]\!](\sigma))(\lightning) + [\![e']\!](\sigma) \cdot \mathbb{0}$$

$$= \bot^\dagger([\![(\mathsf{assume}\ e \fatsemi C)^{n+2}]\!](\sigma))(\lightning)$$

We can now reformulate the semantics of $C^{\langle e,e' \rangle}$ by unrolling the command iteration-by-iteration:

Lemma 11 (Unrolling). *For all $\sigma \in \Sigma$ and $\tau \in \Sigma_\lightning$,*

$$\left[\!\left[C^{\langle e,e' \rangle} \right]\!\right](\sigma)(\tau) = \begin{cases} \sum_{n \in \mathbb{N}} [\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!](\sigma)(\tau) & \text{if } \tau \in \Sigma \\ \inf_{n \in \mathbb{N}} \bot^\dagger([\![(\mathsf{assume}\ e \fatsemi C)^n]\!](\sigma))(\tau) & \text{if } \tau = \lightning \end{cases}$$

Proof. By the program semantics (fig. 5) and Kleene's fixed point theorem,

$$\left[\!\left[C^{\langle e,e' \rangle} \right]\!\right](\sigma) = \mathsf{lfp}\ f.\, \Phi_{\langle C,e,e' \rangle}(f)(\sigma) = \sup_{n \in N} \Phi^n_{\langle C,e,e' \rangle}(\bot)(\sigma)$$

Now, suppose $\tau \in \Sigma$. Since $\Phi^0_{\langle C,e,e' \rangle}(\bot) = \bot$, or the bottom of the order on $(\Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_\lightning))$, we can rewrite the supremum as:

$$\sup_{n \in \mathbb{N}} \Phi^n_{\langle C,e,e' \rangle}(\bot)(\sigma)(\tau) = \sup_{n \in \mathbb{N}} \Phi^{n+1}_{\langle C,e,e' \rangle}(\bot)(\sigma)(\tau)$$

By lemma 10:

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

By the definition of infinite sums:

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

Finally, we can obtain the remaining case using Lemma 10:

$$\begin{aligned}\sup_{n\in\mathbb{N}} \Phi^n_{\langle C,e,e'\rangle}(\sigma)(\circlearrowleft) &= \sup_{n\in\mathbb{N}} \left(\bot^\dagger([\![(\mathsf{assume}\ e\ ;\ C)^n)]\!])(\circlearrowleft)\right) \\ &= \inf_{n\in\mathbb{N}} \bot^\dagger([\![(\mathsf{assume}\ e\ ;\ C)^n)]\!])(\circlearrowleft)\end{aligned}$$

Theorem 7 (Soundness).

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

Proof. The triple $\langle\varphi\rangle\ C\ \langle\psi\rangle$ is proved using inference rules given in fig. 5 and fig. 4. If the last step in this proof makes use of an axiom, then we are done since we took all axioms in Ω to be semantically valid. Otherwise, we proceed by induction on the derivation of $\Omega \vdash \langle\varphi\rangle\ C\ \langle\psi\rangle$.

- ▷ False. We need to show $\vDash \langle\bot\rangle\ C\ \langle\varphi\rangle$. Suppose $m \vDash \bot$. But this is impossible, so the claim holds vacuously.
- ▷ True. We need to show $\vDash \langle\varphi\rangle\ C\ \langle\top\rangle$. Suppose $m \vDash \varphi$. It is trivial that $[\![C]\!]^\dagger(m) \vDash \top$, so we are done.
- ▷ Div. We want to demonstrate $\vDash \langle\Uparrow^{(u)}\rangle\ C\ \langle\Uparrow^{(u)}\rangle$. Suppose $m \vDash \Uparrow^{(u)}$. Then, $m = u \cdot \eta(\circlearrowleft)$ by definition. We have

$$\begin{aligned}[\![C]\!]^\dagger(m) &= \sum_{\sigma\in\mathsf{supp}(m)\cap\Sigma} \big(u\cdot\eta(\circlearrowleft)\big)(\sigma)\cdot[\![C]\!](\sigma) + \big(u\cdot\eta(\circlearrowleft)\big)(\circlearrowleft)\cdot\eta(\circlearrowleft) \\ &= \sum_{\sigma\in\mathsf{supp}(m)\cap\Sigma} \mathbb{0}\cdot[\![C]\!](\sigma) + u\cdot\eta(\circlearrowleft) = u\cdot\eta(\circlearrowleft)\end{aligned}$$

  So, $[\![C]\!]^\dagger(m) \vDash \Uparrow^{(u)}$.
- ▷ Scale. By the induction hypothesis, we have $\vDash \langle\varphi\rangle\ C\ \langle\psi\rangle$. We need to show $\vDash \langle u\odot\varphi\rangle\ C\ \langle u\odot\psi\rangle$. Suppose $m \vDash u\odot\varphi$. By definition, $m = u\cdot m'$ where $m' \vDash \varphi$. Using (iii) of Lemma 1, it follows that

$$[\![C]\!]^\dagger(m) = [\![C]\!]^\dagger(u\cdot m') = u\cdot[\![C]\!]^\dagger(m')$$

  Since $[\![C]\!]^\dagger(m') \vDash \psi$, we can conclude that $[\![C]\!]^\dagger(m) \vDash u\odot\psi$.
- ▷ Disj. By the induction hypothesis, $\vDash \langle\varphi_1\rangle\ C\ \langle\psi_1\rangle$ and $\vDash \langle\varphi_2\rangle\ C\ \langle\psi_2\rangle$. We want 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, take $m \vDash \varphi_1$. The induction hypothesis gives us $[\![C]\!]^\dagger(m) \vDash \psi_1$, which can be weakened to $[\![C]\!]^\dagger(m) \vDash \psi_1\vee\psi_2$. The case for $m \vDash \varphi_2$ is symmetric.
- ▷ Conj. We have $\vDash \langle\varphi_1\rangle\ C\ \langle\psi_1\rangle$ and $\vDash \langle\varphi_2\rangle\ C\ \langle\psi_2\rangle$ by the induction hypothesis and we want to show $\vDash \langle\varphi_1\wedge\varphi_2\rangle\ C\ \langle\psi_1\wedge\psi_2\rangle$. Supposing $m \vDash \varphi_1\wedge\varphi_2$, it holds that $m \vDash \varphi_1$ and $m \vDash \varphi_2$. Then, $[\![C]\!]^\dagger(m) \vDash \psi_1$ and $[\![C]\!]^\dagger(m) \vDash \psi_2$. By definition, $[\![C]\!]^\dagger(m) \vDash \psi_1\wedge\psi_2$, as desired.

- ▷ Choice. By the induction hypothesis, we have that for all $t \in T$, $\vDash \langle \phi(t) \rangle\ C\ \langle \phi'(t) \rangle$. We now show that $\langle \bigoplus_{t \in T} \phi(t) \rangle\ C\ \langle \bigoplus_{t \in T} \phi'(t) \rangle$. Suppose $m \vDash \bigoplus_{t \in T} \phi(t)$. By definition, this means $m = \sum_{t \in T} m_t$ such that for all $t \in T$, $m_t \vDash \phi(t)$. Since $\sum$ commutes with $(-)^\dagger$ in the first argument by Lemma 1,

$$[\![C]\!]^\dagger(m) = [\![C]\!]^\dagger\left(\sum_{t \in T} m_t\right) = \sum_{t \in T} [\![C]\!]^\dagger(m_t)$$

  For each m_t, $[\![C]\!]^\dagger(m_t) \vDash \phi'(t)$, giving us $[\![C]\!]^\dagger(m) \vDash \bigoplus_{t \in T} \phi'(t)$.
- ▷ Exists. By the induction hypothesis, we are given $\forall t \in T.\ \vDash \langle \phi(t) \rangle\ C\ \langle \phi'(t) \rangle$. We need to show $\vDash \langle \exists t : T.\ \phi(t) \rangle\ C\ \langle \exists t : T.\ \phi'(t) \rangle$. Suppose $m \vDash \exists t : T.\ \phi(t)$. By definition, this means $m \in \bigcup_{t \in T} \phi(t)$, so there must be some $t \in T$ for which $m \in \phi(t)$. Then, by the induction hypothesis, $[\![C]\!]^\dagger(m) \vDash \phi'(t)$, we have that $[\![C]\!]^\dagger(m) \vDash \exists t : T.\ \phi'(t)$.
- ▷ Consequence. Given to us are $\vDash \varphi' \implies \varphi$ and $\vDash \psi \implies \psi'$, and we have $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$ by the induction hypothesis. We need to show $\vDash \langle \varphi' \rangle\ C\ \langle \psi' \rangle$. Suppose $m \vDash \varphi'$. Then, $m \vDash \varphi$ and $[\![C]\!]^\dagger(m) \vDash \psi$. Finally, this gives us $[\![C]\!]^\dagger(m) \vDash \psi'$ as desired.
- ▷ Skip. We need to show that $\vDash \langle \varphi \rangle\ \mathsf{skip}\ \langle \varphi \rangle$. Suppose that $m \vDash \varphi$. We have that $[\![\mathsf{skip}]\!]^\dagger(m) = m$, so $[\![\mathsf{skip}]\!]^\dagger(m) \vDash \varphi$ as desired.
- ▷ Seq. Given $\Omega \vdash \langle \varphi \rangle\ C_1\ \langle \vartheta \rangle$ and $\Omega \vdash \langle \vartheta \rangle\ C_2\ \langle \psi \rangle$, we need to show $\vDash \langle \varphi \rangle\ C_1 ; C_2\ \langle \psi \rangle$. Suppose $m \vDash \varphi$, so by the induction hypothesis, $[\![C_1]\!]^\dagger(m) \vDash \vartheta$ and $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) \vDash \psi$. Since $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) = ([\![C_1]\!]^\dagger \circ [\![C_2]\!]^\dagger)(m) = [\![C_1 ; C_2]\!]^\dagger(m)$, we are done.
- ▷ Plus. Given $\varphi \vDash \mathbb{1}$, $\Omega \vdash \langle \varphi \rangle\ C_1\ \langle \psi_1 \rangle$, and $\Omega \vdash \langle \varphi \rangle\ C_2\ \langle \psi_2 \rangle$, we need to show $\vDash \langle \varphi \rangle\ C_1 + C_2\ \langle \psi_1 \oplus \psi_2 \rangle$. First, suppose $m \vDash \varphi$. Since $\varphi \vDash \mathbb{1}$, it must be that $\mathsf{supp}(m) \subseteq \Sigma$. By Lemma 1, we know that the $(-)^\dagger$ operator commutes with $+$ in its first argument:

$$[\![C_1 + C_2]\!]^\dagger(m) = [\![C_1]\!]^\dagger(m) + [\![C_2]\!]^\dagger(m)$$

  By the induction hypothesis, $[\![C_1]\!]^\dagger(m) \vDash \psi_1$ and $[\![C_2]\!]^\dagger(m) \vDash \psi_2$, so we have $[\![C_1 + C_2]\!]^\dagger(m) \vDash \psi_1 \oplus \psi_2$.
- ▷ Assume. Suppose $\varphi \vDash e = u$. Recall that this holds iff

$$\forall m \in \varphi.\ m(\circlearrowleft) = 0 \text{ and } \forall \sigma \in \mathsf{supp}(m) \cap \Sigma.\ [\![e]\!](\sigma) = u.$$

  We need to show $\vDash \langle \varphi \rangle\ \mathsf{assume}\ e\ \langle \varphi\ u \rangle$. Take $m \vDash \varphi$. Then,

$$\begin{aligned}
[\![\mathsf{assume}\ e]\!]^\dagger(m) &= \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot [\![\mathsf{assume}\ ]\!](\sigma) + m(\circlearrowleft) \cdot \eta(\circlearrowleft) \\
&= \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot [\![e]\!](\sigma) \cdot \eta(\sigma) + m(\circlearrowleft) \cdot \eta(\circlearrowleft) \\
&= \sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot u \cdot \eta(\sigma) + 0
\end{aligned}$$

$$= \left(\sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot \eta(\sigma) \right) \cdot u$$
$$= m \cdot u$$

By definition, $m \cdot u \vDash \varphi \odot u$, as desired.

▷ ITER. Assuming the premises, we need to show that $\vDash \langle \varphi_0 \oplus \zeta_0 \rangle\ C^{\langle e, e' \rangle}\ \langle \psi_\infty \oplus \zeta_\infty \rangle$. Suppose $m \vDash \varphi_0 \oplus \zeta_0$. By the induction hypothesis, we have that for all $n \in \mathbb{N}$,

$$\vDash \langle \varphi_n \oplus \zeta_n \rangle\ \mathsf{assume}\ e \fatsemi C\ \langle \varphi_{n+1} \oplus \zeta_{n+1} \rangle \qquad \vDash \langle \varphi_n \rangle\ \mathsf{assume}\ e'\ \langle \psi_n \rangle$$

By mathematical induction on n, it is straightforward to show that

$$[\![(\mathsf{assume}\ e \fatsemi C)^n]\!]^\dagger(m) \vDash \varphi_n \oplus \zeta_n \tag{1}$$
$$[\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m) \vDash \psi_n \oplus \zeta_n \tag{2}$$

for all n. Note that $\langle \zeta_n \rangle\ \mathsf{assume}\ e'\ \langle \zeta_n \rangle$ since $\zeta_n \vDash \mathsf{div}$. In particular, we have that for any $m \vDash \zeta_n$, $\mathsf{supp}(m) \subseteq \{\circlearrowleft\}$, so $[\![\mathsf{assume}\ e']\!]^\dagger(m) = m$.
Consider each of these separately:

1. For all n,
$$[\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m) \vDash \psi_n \oplus \zeta_n$$
There exist $p_n, q_n \in \mathcal{W}_{\mathcal{A}}(\Sigma_\circlearrowleft)$, where $p_n \vDash \psi_n$ and $q_n \vDash \zeta_n$, such that $[\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi]\!]^\dagger(m) = p_n + q_n$. Since $(\psi_n)_{n \in \mathbb{N}} \rightsquigarrow \psi_\infty$, it must be that $\sum_{n \in \mathbb{N}} p_n \vDash \psi_\infty$.
2. For all n,
$$[\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m) \vDash \psi_n \oplus \zeta_n$$
Since $(\varphi_n \oplus \zeta_n)_{n \in \mathbb{N}} \Uparrow \zeta_\infty$, we have
$$\left(\inf_{n \in \mathbb{N}} \left| [\![(\mathsf{assume}\ e \fatsemi C)^n]\!]^\dagger(m) \right| \cdot \top \right) \cdot \eta(\circlearrowleft) \vDash \zeta_\infty$$

We can now combine the two parts. By Lemma 11, the unrolling of $\left[\!\left[C^{\langle e, e' \rangle} \right]\!\right]^\dagger$ is

$$\left[\!\left[C^{\langle e, e' \rangle} \right]\!\right]^\dagger = \sum_{n \in \mathbb{N}} \mathsf{true}\ ?\ [\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m) + \left(\inf_{n \in \mathbb{N}} |\, [\![(\mathsf{assume}\ e \fatsemi C)^n]\!]^\dagger(m) | \cdot \top \right) \cdot \eta(\circlearrowleft)$$

Therefore, $\left[\!\left[C^{\langle e, e' \rangle} \right]\!\right]^\dagger(m) \vDash \psi_\infty \oplus \zeta_\infty$.

Definition 14. *For any test $b \in 2^\Sigma$ and $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_\circlearrowleft)$, we define the projection of m onto b to be the following weighting function:*

$$(b\ ?\ m)(\sigma) \triangleq \begin{cases} m(\sigma) & \text{if } [\![b]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \\ 0 & \text{if } [\![b]\!]_{\mathsf{Test}}(\sigma) = \mathbb{0} \text{ or } \sigma = \circlearrowleft \end{cases}$$

It follows that, for any $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_\circlearrowleft)$, $m = b\ ?\ m + \neg b\ ?\ m$.

In the proof of relative completeness, it will be particularly useful to reason on the projection of a given $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ onto $\mathsf{true} = \Sigma$, or the part of m specifying weights *solely* on terminating outcomes. According to the above definition, this is

$$\mathsf{true} \,?\, m = \begin{cases} \mathbb{1} & \text{if } \sigma \in \Sigma \\ \mathbb{0} & \text{if } \sigma = \circlearrowleft \end{cases}$$

This then allows us to decompose any $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ into terminating and non-terminating components: $m = (\mathsf{true} \,?\, m) + m(\circlearrowleft) \cdot \eta(\circlearrowleft)$.

Lemma 12.

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

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

- ▷ $C = \mathsf{skip}$. $[\![\mathsf{skip}]\!]^{\dagger}(m) = m$ for all $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, so $\mathsf{post}(\mathsf{skip}, \varphi) = \{[\![\mathsf{skip}]\!]^{\dagger}(m) \mid m \in \varphi\} = \varphi$. The desired triple is derived by one application of the Skip rule:

$$\frac{}{\langle \varphi \rangle \ C \ \langle \varphi \rangle} \ \textsc{Skip}$$

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

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

  By the induction hypothesis, we have

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

  We complete the proof using the Seq rule:

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

- ▷ $C = C_1 + C_2$. It holds that

$$\begin{aligned} \mathsf{post}(C_1 + C_2, \varphi) &= \left\{ [\![C_1 + C_2]\!]^{\dagger}(m) \mid m \in \varphi \right\} \\ &= \left\{ [\![C_1 + C_2]\!]^{\dagger}\big(\mathsf{true} \,?\, m + m(\circlearrowleft) \cdot \eta(\circlearrowleft)\big) \mid m \in \varphi \right\} \end{aligned}$$

  By Lemma 1 (i), this is equal to:

$$= \left\{ [\![C_1 + C_2]\!]^{\dagger}\big(\mathsf{true} \,?\, m\big) + [\![C_1 + C_2]\!]^{\dagger}\big(m(\circlearrowleft) \cdot \eta(\circlearrowleft)\big) \mid m \in \varphi \right\}$$

Observe that $\mathsf{supp}(\mathsf{true}\,?\,m) \subseteq \Sigma$. Applying (ii) and (iv) of Lemma 1:

$$
\begin{aligned}
&= \left\{ [\![C_1]\!]^\dagger(\mathsf{true}\,?\,m) + [\![C_2]\!]^\dagger(\mathsf{true}\,?\,m) + m(\circlearrowleft)\cdot\eta(\circlearrowleft) \mid m \in \varphi \right\} \\
&= \bigcup_{m\in\varphi} \left\{ [\![C_1]\!]^\dagger(m') + [\![C_2]\!]^\dagger(m') + m(\circlearrowleft)\cdot\eta(\circlearrowleft) \mid m' \in \mathbf{1}_{\mathsf{true}\,?\,m} \right\} \\
&= \exists m : \varphi.\ \mathsf{post}(C_1, \mathsf{true}\,?\,m) \oplus \mathsf{post}(C_2, \mathsf{true}\,?\,m) \ \oplus \mathsf{div}u
\end{aligned}
$$

Then, the following derivation completes the proof:

$$
\dfrac{\dfrac{\dfrac{\mathbf{1_m} \vDash \mathbb{1} \qquad \dfrac{\Omega}{\langle \mathbf{1}_m \rangle\ C_1\ \langle \mathsf{post}(C_1, \mathbf{1}_m)\rangle} \qquad \dfrac{\Omega}{\langle \mathbf{1}_m \rangle\ C_2\ \langle \mathsf{post}(C_2, \mathbf{1}_m)\rangle}}{\langle \mathbf{1}_m \rangle\ C_1 + C_2\ \langle \mathsf{post}(C_1, \mathbf{1}_{\mathsf{true}\,?\,m}) \oplus \mathsf{post}(C_2, \mathbf{1}_{\mathsf{true}\,?\,m})\rangle}\ \textsc{Plus}}{\langle \mathbf{1}_m \rangle\ C_1 + C_2\ \langle \mathsf{post}(C_1, \mathbf{1}_{\mathsf{true}\,?\,m}) \oplus \mathsf{post}(C_2, \mathbf{1}_{\mathsf{true}\,?\,m}) \oplus \Uparrow^{(m(\circlearrowleft))}\rangle}\ \textsc{Choice}}{\langle \varphi \rangle\ C_1 + C_2\ \langle \mathsf{post}(C_1 + C_2, \varphi)\rangle}\ \textsc{Exists}
$$

▷ $C = \mathsf{assume}\ e$, where e must either be a test $b \in \mathsf{Test}$ or weight $u \in U$. First, we see that

$$
\begin{aligned}
\mathsf{post}(\mathsf{assume}\ e, \varphi) &= \left\{ [\![\mathsf{assume}\ e]\!]^\dagger(m) \mid m \in \varphi \right\} \\
&= \left\{ [\![\mathsf{assume}\ e]\!]^\dagger(\mathsf{true}\,?\,m + m(\circlearrowleft)\cdot\eta(\circlearrowleft)) \mid m \in \varphi \right\} \\
&= \left\{ [\![\mathsf{assume}\ e]\!]^\dagger(\mathsf{true}\,?\,m) + m(\circlearrowleft)\cdot\eta(\circlearrowleft) \mid m \in \varphi \right\}
\end{aligned}
$$

Recall that, by Definition 14, $\mathsf{true}\,?\,m = b\,?\,(\mathsf{true}\,?\,m) + \neg b\,?\,(\mathsf{true}\,?\,m)$, so $\mathbf{1}_{\mathsf{true}\,?\,m} = \mathbf{1}_{b\,?\,(\mathsf{true}\,?\,m)} + \mathbf{1}_{\neg b\,?\,(\mathsf{true}\,?\,m)}$. Then, we have that:

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

It should be clear that $b\,?\,(\mathsf{true}\,?\,m) \vDash b = \mathbb{1}$ and $\neg b\,?\,(\mathsf{true}\,?\,m) \vDash b = \mathbb{0}$. The derivation is completed as follows:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{1}_{b\,?\,(\mathsf{true}\,?\,m)} \vDash b = \mathbb{1}}{\langle \mathbf{1}_{b\,?\,(\mathsf{true}\,?\,m)}\rangle\ \mathsf{assume}\ b\ \langle \mathbf{1}_{b\,?\,(\mathsf{true}\,?\,m)} \odot \mathbb{1}\rangle}\ \textsc{Assume} \qquad \dfrac{\mathbf{1}_{\neg b\,?\,(\mathsf{true}\,?\,m)} \vDash b = \mathbb{0}}{\langle \mathbf{1}_{\neg b\,?\,(\mathsf{true}\,?\,m)}\rangle\ \mathsf{assume}\ b\ \langle \mathbf{1}_{b\,?\,(\mathsf{true}\,?\,m)} \odot \mathbb{0}\rangle}\ \textsc{Assume}}{\langle \mathbf{1}_{b\,?\,(\mathsf{true}\,?\,m)} \oplus \mathbf{1}_{\neg b\,?\,(\mathsf{true}\,?\,m)}\rangle\ \mathsf{assume}\ b\ \langle \mathbf{1}_{\neg b\,?\,(\mathsf{true}\,?\,m)}\rangle}\ \textsc{Choice} \qquad \dfrac{}{\langle \Uparrow^{(m(\circlearrowleft))}\rangle\ \mathsf{assume}\ b\ \langle \Uparrow^{(m(\circlearrowleft))}\rangle}\ \textsc{Div}}{\langle \mathbf{1}_{b\,?\,(\mathsf{true}\,?\,m)} \oplus \mathbf{1}_{\neg b\,?\,(\mathsf{true}\,?\,m)} \oplus \Uparrow^{(m(\circlearrowleft))}\rangle\ \mathsf{assume}\ b\ \langle \mathbf{1}_{b\,?\,(\mathsf{true}\,?\,m)} \oplus \Uparrow^{(m(\circlearrowleft))}\rangle}\ \textsc{Choice}}{\langle \varphi \rangle\ \mathsf{assume}\ b\ \langle \mathsf{post}(\mathsf{assume}\ b, \varphi)\rangle}\ \textsc{Exists}
$$

Suppose instead that e is a weight $u \in U$. Since $\mathsf{supp}(\mathsf{true}\ ?\ m) \subseteq \Sigma$, $[\![\mathsf{assume}\ u]\!]^\dagger(\mathsf{true}\ ?\ m) = \mathsf{true}\ ?\ m \cdot u$ by Lemma 1. So,

$$\begin{aligned}\mathsf{post}(\mathsf{assume}\ u, \varphi) &= \{\mathsf{true}\ ?\ m \cdot u + m(\circlearrowleft) \cdot \eta(\circlearrowleft) \mid m \in \varphi\} \\ &= \exists m : \varphi.\ \mathbf{1}_{\mathsf{true}\ ?\ m} \odot u \oplus \Uparrow^{(m(\circlearrowleft))}\end{aligned}$$

We know $\mathbf{1}_{\mathsf{true}\ ?\ m} \vDash u = u$, so the derivation is given by:

$$\textsc{Exists}\ \dfrac{\textsc{Choice}\ \dfrac{\textsc{Assume}\ \dfrac{\mathbf{1}_{\mathsf{true}\ ?\ m} \vDash u = u}{\langle \mathbf{1}_{\mathsf{true}\ ?\ m} \rangle\ \mathsf{assume}\ u\ \langle \mathbf{1}_{\mathsf{true}\ ?\ m} \odot u \rangle} \qquad \textsc{Div}\ \dfrac{}{\langle \Uparrow^{(m(\circlearrowleft))} \rangle\ \mathsf{assume}\ u\ \langle \Uparrow^{(m(\circlearrowleft))} \rangle}}{\langle \mathbf{1}_{\mathsf{true}\ ?\ m} \oplus \Uparrow^{(m(\circlearrowleft))} \rangle\ \mathsf{assume}\ u\ \langle \mathbf{1}_{\mathsf{true}\ ?\ m} \odot u \oplus \Uparrow^{(m(\circlearrowleft))} \rangle}}{\langle \varphi \rangle\ \mathsf{assume}\ u\ \langle \mathsf{post}(\mathsf{assume}\ u, \varphi) \rangle}$$

▷ $C = C^{\langle e, e' \rangle}$. We define

$$\begin{aligned}\varphi_n(m) &\triangleq \mathbf{1}_{\mathsf{true}\ ?\ [\![(\mathsf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)} \\ u_n(m) &\triangleq [\![(\mathsf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)(\circlearrowleft) \\ \psi_n(m) &\triangleq \mathbf{1}_{\mathsf{true}\ ?\ [\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m)} \\ \psi_\infty(m) &\triangleq \mathbf{1}_{\mathsf{true}\ ?\ [\![C^{\langle e, e' \rangle}]\!]^\dagger(m)} \\ v(m) &\triangleq \left[\!\!\left[C^{\langle e, e' \rangle}\right]\!\!\right]^\dagger(m)(\circlearrowleft)\end{aligned}$$

Observe that

$$\begin{aligned}\varphi_n(m) \oplus \Uparrow^{(u_n(m))} &= \mathsf{post}((\mathsf{assume}\ e \fatsemi C)^n, \mathbf{1}_m) \\ \psi_\infty(m) \oplus \Uparrow^{(v(m))} &= \mathsf{post}(C^{\langle e, e' \rangle}, \mathbf{1}_m)\end{aligned}$$

By definition, $\varphi_0(m) = \mathbf{1}_{\mathsf{true}\ ?\ m}$ and $u_0(m) = m(\circlearrowleft)$, so $m = \varphi_0(m) \oplus \Uparrow^{(u_0(m))}$. Then,

$$\varphi = \exists m : \varphi. \varphi_0(m) \oplus \Uparrow^{(u_0(m))} \qquad \mathsf{post}(C^{\langle e, e' \rangle}, \varphi) = \exists m : \varphi. \psi_\infty(m) \oplus \Uparrow^{(v(m))}$$

We proceed by showing that the relevant premises of the ITER rule hold for our definitions above:

- First, we want to show that $\Omega \vdash \langle \varphi_n(m) \oplus \Uparrow^{(u_n(m))} \rangle\ \mathsf{assume}\ e \fatsemi C\ \langle \varphi_{n+1}(m) \oplus \Uparrow^{(u_{n+1}(m))} \rangle$. By the above definitions, it is clear that $\vDash \langle \varphi_n(m) \oplus \Uparrow^{(u_n(m))} \rangle\ \mathsf{assume}\ e \fatsemi C\ \langle \varphi_{n+1}(m) \oplus \Uparrow^{(u_{n+1}(m))} \rangle$. The induction hypothesis tells us that there is a derivation of this triple from Ω.
- Next, we show $\Omega \vdash \langle \varphi_n(m) \rangle\ \mathsf{assume}\ e'\ \langle \psi_n(m) \rangle$. Note that for $\tau \in \Sigma$:

$$\begin{aligned}&\big(\mathsf{true}\ ?\ [\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m)\big)(\tau) \\ &= [\![(\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e']\!]^\dagger(m)(\tau) \\ &= [\![\mathsf{assume}\ e']\!]^\dagger([\![(\mathsf{assume}\ e \fatsemi C)^n]\!]^\dagger(m))(\tau)\end{aligned}$$

$$= \sum_{\sigma \in \mathsf{supp}(\llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m)) \cap \Sigma} \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m)(\sigma) \cdot \llbracket \mathsf{assume}\ e' \rrbracket(\sigma)(\tau)$$

$$= \sum_{\sigma \in \mathsf{supp}(\mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m)) \cap \Sigma} \mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m)(\sigma) \cdot \llbracket \mathsf{assume}\ e' \rrbracket(\sigma)(\tau)$$

$$= \llbracket \mathsf{assume}\ e' \rrbracket^\dagger(\mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m))(\tau)$$

And $\big(\mathsf{true}?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e' \rrbracket^\dagger(m)\big)(\lightning) = \mathbb{0} = \llbracket \mathsf{assume}\ e' \rrbracket^\dagger(\mathsf{true}?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m))$. Thus,

$$\mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e' \rrbracket^\dagger(m) = \llbracket \mathsf{assume}\ e' \rrbracket^\dagger(\mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m))$$

From this, we see that $\psi_n(m) = \mathsf{post}(\mathsf{assume}\ e', \varphi_n(m))$ and thus $\vDash \langle \varphi_n(m) \rangle\ \mathsf{assume}\ e'\ \langle \psi_n(m) \rangle$. By the induction hypothesis, we get $\Omega \vdash \langle \varphi_n(m) \rangle\ \mathsf{assume}\ e'\ \langle \psi_n(m) \rangle$.

- For each $n \in \mathbb{N}$, we defined $\varphi_n = \mathbf{1}_{\mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m)} = \{\mathsf{true}?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m)\}$. Since the weighting function in this singleton is a projection onto Σ, it assigns no weight to nontermination, so $\varphi_n(m) \vDash \mathsf{true}$ by Definition 10 of assertion entailment.
- We now show $(\psi_n(m))_{n \in \mathbb{N}} \rightsquigarrow \psi_\infty(m)$. Take any $(m_n)_{n \in \mathbb{N}}$ such that $m_n \vDash \psi_n(m)$ for each n. Then, $m_n = \mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e' \rrbracket^\dagger(m)$. It holds that for all $\sigma \in \Sigma$,

$$\begin{aligned}\sum_{n \in \mathbb{N}} m_n(\sigma) &= \sum_{n \in \mathbb{N}} \mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e' \rrbracket^\dagger(m)(\sigma) \\ &= \sum_{n \in \mathbb{N}} \llbracket (\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e' \rrbracket^\dagger(m)(\sigma) \\ &= \left\llbracket C^{\langle e, e' \rangle} \right\rrbracket^\dagger(m)(\sigma)\end{aligned}$$

and

$$\sum_{n \in \mathbb{N}} m_n(\lightning) = \sum_{n \in \mathbb{N}} \mathsf{true}\ ?\ \llbracket (\mathsf{assume}\ e \fatsemi C)^n \fatsemi \mathsf{assume}\ e' \rrbracket^\dagger(m)(\lightning) = \mathbb{0}$$

Thus, $\sum_{n \in \mathbb{N}} m_n = \mathsf{true}\ ?\ \left\llbracket C^{\langle e, e' \rangle} \right\rrbracket^\dagger(m)$, giving us that $\sum_{n \in \mathbb{N}} m_n \vDash \psi_\infty(m)$.
- We show that $(\varphi_n(m) \oplus \Uparrow^{(u_n(m))})_{n \in \mathbb{N}} \Uparrow v(m)$. Take any $(m_n)_{n \in \mathbb{N}}$ such that $m_n \vDash \varphi_n(m) \oplus \Uparrow^{(u_n)}$ for each n. Then, $m_n = \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m)$. We have that

$$\begin{aligned}\inf_{n \in \mathbb{N}} |m_n| \cdot \top &= \inf_{n \in \mathbb{N}} \left| \llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m) \right| \cdot \top \\ &= \inf_{n \in \mathbb{N}} \bot^\dagger(\llbracket (\mathsf{assume}\ e \fatsemi C)^n \rrbracket^\dagger(m))(\lightning) \\ &= \left\llbracket C^{\langle e, e' \rangle} \right\rrbracket^\dagger(m)(\lightning) \\ &= v(m)\end{aligned}$$

The derivation $\Omega \vdash \langle\varphi\rangle\ C^{\langle e,e'\rangle}\ \langle \mathsf{post}(C^{\langle e,e'\rangle}, \varphi)\rangle$ is then completed as follows:

$$\dfrac{\dfrac{\dfrac{\Omega}{\forall n.\ \langle\varphi_n(m) \oplus \Uparrow^{(u_n(m))}\rangle\ \mathsf{assume}\ e \fatsemi C\ \langle\varphi_{n+1}(m) \oplus \Uparrow^{(u_{n+1}(m))}\rangle} \quad \dfrac{\Omega}{\langle\varphi_n(m)\rangle\ \mathsf{assume}\ e'\ \langle\psi_n(m)\rangle}}{\langle\varphi_0(m) \oplus \Uparrow^{u_0(m)}\rangle\ C^{\langle e,e'\rangle}\ \langle\psi_\infty(m) \oplus \Uparrow^{(v(m))}\rangle}\ \textsc{Iter}}{\langle\varphi\rangle\ C^{\langle e,e'\rangle}\ \langle \mathsf{post}(C^{\langle e,e'\rangle}, \varphi)\rangle}\ \textsc{Exists}$$

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

E Rule Derivations

Lemma 13. *The following rules are derivable*

$$\textsc{Lemma 13 - Partial}\ \dfrac{\langle P\rangle\ C\ \langle\Box Q\rangle}{\langle\Box P\rangle\ C\ \langle\Box Q\rangle} \qquad \dfrac{\langle P\rangle\ C\ \langle\blacksquare Q\rangle}{\langle\blacksquare P\rangle\ C\ \langle\blacksquare Q\rangle}\ \textsc{Lemma 13 - Total}$$

Proof. Observe that $\Box P$ is equivalent to:

$$\Box P = \exists m : \Box P.\ \mathbf{1}_m = \exists m : \Box P.\ \oplus_{\sigma \in \mathsf{supp}(m)}\ m(\sigma) \odot \mathbf{1}_{\eta(\sigma)}$$

For $\sigma \in \mathsf{supp}(m)$, $\eta(\sigma) \vDash P$, so $\Box P$ implies $\exists m : \Box P.\ \bigoplus_{\sigma \in \mathsf{supp}(m)}\ m(\sigma) \odot P$. Moreover,

$$\begin{aligned}\Box Q &= \exists m : \Box P.\ \Box Q \\ &= \exists m : \Box P.\ \oplus_{\sigma \in \mathsf{supp}(m)}\ \Box Q \quad = \quad \exists m : \Box P.\ \oplus_{\sigma \in \mathsf{supp}(m)}\ m(\sigma) \odot \Box Q\end{aligned}$$

The derivation is as follows:

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

Note that $P \implies \Box P$, so by a simple application of Consequence, we can derive the converse:

$$\dfrac{\langle\Box P\rangle\ C\ \langle\Box Q\rangle}{\langle P\rangle\ C\ \langle\Box Q\rangle}\ \textsc{Consequence}$$

So we see that the two triples $\langle P\rangle\ C\ \langle\Box Q\rangle$ and $\langle\Box P\rangle\ C\ \langle\Box Q\rangle$ are really interchangeable. Very similar derivations can be obtained to show that $\langle P\rangle\ C\ \langle\blacksquare Q\rangle$ and $\langle\blacksquare P\rangle\ C\ \langle\blacksquare Q\rangle$ are equivalent as well.

Lemma 14. *The following rule is derivable:*

$$\frac{\langle P\rangle\ C\ \langle \Diamond Q\rangle}{\langle \Diamond P\rangle\ C\ \langle \Diamond Q\rangle}\ \textsc{Lemma 14}$$

Proof.

Lemma 15. *The following rules are derivable:*

$$\textsc{Seq-Lisbon}\ \frac{\langle P\rangle\ C_1\ \langle \Diamond Q\rangle \qquad \langle Q\rangle\ C_2\ \langle \Diamond R\rangle}{\langle P\rangle\ C_1 \fatsemi C_2\ \langle \Diamond R\rangle} \qquad \textsc{Seq-Total-Hoare}\ \frac{\langle P\rangle\ C_1\ \langle \blacksquare Q\rangle \qquad \langle Q\rangle\ C_2\ \langle \blacksquare R\rangle}{\langle P\rangle\ C_1 \fatsemi C_2\ \langle \blacksquare R\rangle}$$

Proof. We show only the derivation for Seq-Total-Hoare – we derive Seq-Lisbon nearly identically using an application of Lemma 14.

$$\frac{\langle P\rangle\ C_1\ \langle \blacksquare Q\rangle \qquad \dfrac{\langle Q\rangle\ C_2\ \langle \blacksquare R\rangle}{\langle \blacksquare Q\rangle\ C_2\ \langle \blacksquare R\rangle}\ \textsc{Lemma 13 - Total}}{\langle P\rangle\ C_1 \fatsemi C_2\ \langle \blacksquare R\rangle}\ \textsc{Seq}$$

Lemma 16. *The following rule is derivable:*

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

Proof. First, we give a subderivation (1):

$$\frac{\dfrac{\dfrac{\varphi_1 \vDash b}{\langle \varphi_1\rangle\ \mathsf{assume}\ b\ \langle \varphi_1\rangle}\ \textsc{Assume} \qquad \dfrac{\varphi_2 \vDash \neg b}{\langle \varphi_2\rangle\ \mathsf{assume}\ b\ \langle \mathbf{1}_0\rangle}\ \textsc{Assume}}{\langle \varphi_1 \oplus \varphi_2\rangle\ \mathsf{assume}\ b\ \langle \varphi_1\rangle}\ \textsc{Choice} \qquad \langle \varphi_1\rangle\ C\ \langle \psi_1\rangle}{\langle \varphi_1 \oplus \varphi_2\rangle\ \mathsf{assume}\ b \fatsemi C_1\ \langle \psi_1\rangle}\ \textsc{Seq}$$

The proof of (2) is nearly identical. We complete the derivation with:

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

Lemma 17. *The following rule is derivable:*

$$\frac{\langle P \wedge b\rangle\ C_1\ \langle \blacksquare Q\rangle \qquad \langle P \wedge \neg b\rangle\ C_2\ \langle \blacksquare Q\rangle}{\langle P\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \blacksquare Q\rangle}\ \textsc{If-Hoare}$$

Proof. Note that $\blacksquare(P \wedge b) \vDash b$ and $\blacksquare(P \wedge \neg b) \vDash \neg b$. The derivation proceeds as follows:

$$\frac{\dfrac{\dfrac{\textsc{Lemma 13 - Total}\ \langle P \wedge b\rangle\ C_1\ \langle \blacksquare Q\rangle}{\langle \blacksquare(P \wedge b)\rangle\ C_1\ \langle \blacksquare Q\rangle} \qquad \dfrac{\textsc{Lemma 13 - Total}\ \langle P \wedge \neg b\rangle\ C_2\ \langle \blacksquare Q\rangle}{\langle \blacksquare(P \wedge \neg b)\rangle\ C_2\ \langle \blacksquare Q\rangle}}{\langle \blacksquare(P \wedge b) \oplus \blacksquare(P \wedge \neg b)\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \blacksquare Q \oplus \blacksquare Q\rangle}\ \textsc{If}}{\langle P\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \blacksquare Q\rangle}\ \textsc{Consequence}$$

Lemma 18. *The following rule is derivable:*

$$\frac{\langle P \wedge b\rangle\ C_1\ \langle \Diamond Q\rangle \qquad \langle P \wedge \neg b\rangle\ C_2\ \langle \Diamond Q\rangle}{\langle P\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q\rangle}\ \textsc{If-Lisbon}$$

Proof. Subderivation (1):

$$\dfrac{\dfrac{P \wedge b \vDash b \qquad \langle P \wedge b\rangle\ C_1\ \langle \Diamond Q\rangle \qquad \mathbf{1}_0 \vDash \neg b \qquad \dfrac{}{\langle \mathbf{1}_0\rangle\ C_2\ \langle \mathbf{1}_0\rangle}\ \textsc{Lemma}}{\langle P \wedge b\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q\rangle}}{\langle \Diamond(P \wedge b)\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q\rangle}\ \textsc{Lemma 14}$$

Part (2) is symmetrical.

$$\dfrac{\dfrac{\dfrac{\textsc{Lemma 14}}{\langle \Diamond(P \wedge b)\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q\rangle}(1) \qquad \dfrac{\textsc{Lemma 14}}{\langle \Diamond(P \wedge \neg b)\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q\rangle}(2) \qquad \vdots}{\langle \Diamond(P \wedge b) \vee \Diamond(P \wedge \neg b)\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q \vee \Diamond Q\rangle}\ \textsc{Disj}}{\langle P\rangle\ \mathsf{if}\ b\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \Diamond Q\rangle}\ \textsc{Consequence}$$

Lemma 19. *The following rule is derivable:*

$$\frac{\zeta \vDash \mathsf{div}}{\langle \zeta\rangle\ C\ \langle \zeta\rangle}\ \textsc{Div}^*$$

Proof. Since $\zeta \vDash \mathsf{div}$, we know that for all m such that $m \vDash \zeta$, $\mathsf{supp}(m) \subseteq \{\circlearrowleft\}$. Then, these are equivalent: $\zeta = \exists m : \zeta.\ \Uparrow^{(m(\circlearrowleft))}$. The derivation is as follows:

$$\dfrac{\dfrac{\forall m \in \zeta. \qquad \dfrac{}{\langle \Uparrow^{(m(\circlearrowleft))}\rangle\ C\ \langle \Uparrow^{(m(\circlearrowleft))}\rangle}\ \textsc{Div}}{\langle \exists m : \zeta.\ \Uparrow^{(m(\circlearrowleft))}\rangle\ C\ \langle \exists m : \zeta.\ \Uparrow^{(m(\circlearrowleft))}\rangle}\ \textsc{Exists}}{\langle \zeta\rangle\ C\ \langle \zeta\rangle}\ \textsc{Consequence}$$

Lemma 20. *The following rule is derivable:*

$$\frac{\begin{array}{cccc}(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty & (\varphi_n \oplus \psi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty & \forall n \in \mathbb{N}. & \\ \varphi_n \vDash b \qquad \psi_n \vDash \neg b & \zeta_n \vDash \mathsf{div} & \langle \varphi_n \oplus \zeta_n\rangle\ C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \oplus \zeta_{n+1}\rangle & \end{array}}{\langle \varphi_0 \oplus \psi_0 \oplus \zeta_0\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \psi_\infty \oplus \zeta_\infty\rangle}\ \textsc{While}$$

Proof. We will use ITER to derive this rule. First, we give the following sub-derivation (*):

$$\textsc{Choice}\ \frac{\textsc{Assume}\ \dfrac{\varphi_n \vDash b}{\langle \varphi_n \rangle\ \mathsf{assume}\ b\ \langle \varphi_n \rangle} \qquad \textsc{Assume}\ \dfrac{\psi_n \vDash \neg b}{\langle \psi_n \rangle\ \mathsf{assume}\ b\ \langle \mathbf{1}_0 \rangle} \qquad \vdots}{\langle \varphi_n \oplus \psi_n \oplus \zeta_n \rangle\ \mathsf{assume}\ b\ \langle \varphi_n \oplus \zeta_n \rangle} \qquad \dfrac{\zeta_n \vDash \mathsf{div}}{\langle \zeta_n \rangle\ \mathsf{assume}\ b\ \langle \zeta_n \rangle}\ \textsc{Div}^*$$

This gives us (1):

$$\textsc{Seq}\ \frac{\dfrac{(*)}{\langle \varphi_n \oplus \psi_n \oplus \zeta_n \rangle\ \mathsf{assume}\ b\ \langle \varphi_n \oplus \zeta_n \rangle} \qquad \langle \varphi_n \oplus \zeta_n \rangle\ C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \oplus \zeta_{n+1} \rangle}{\langle \varphi_n \oplus \psi_n \oplus \zeta_n \rangle\ \mathsf{assume}\ b \fatsemi C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \oplus \zeta_{n+1} \rangle}$$

And we derive (2) similarly:

$$\textsc{Choice}\ \frac{\textsc{Assume}\ \dfrac{\varphi_n \vDash b}{\langle \varphi_n \rangle\ \mathsf{assume}\ \neg b\ \langle \mathbf{1}_0 \rangle} \qquad \textsc{Assume}\ \dfrac{\psi_n \vDash \neg b}{\langle \psi_n \rangle\ \mathsf{assume}\ \neg b\ \langle \psi_n \rangle}}{\langle \varphi_n \oplus \psi_n \rangle\ \mathsf{assume}\ \neg b\ \langle \psi_n \rangle}$$

Now, note that since $\varphi_n \vDash b$ and $\psi_n \vDash \neg b$, it must be that $\varphi_n \oplus \psi_n \vDash \mathsf{true}$. Recall also that $\mathsf{while}\ b\ \mathsf{do}\ C$ is sugar for $C^{\langle b, \neg b \rangle}$. The full derivation proceeds as follows:

$$\forall n \in \mathbb{N}.\quad \textsc{Iter}\ \frac{\dfrac{(1)}{\langle \varphi_n \oplus \psi_n \oplus \zeta_n \rangle\ \mathsf{assume}\ b \fatsemi C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \oplus \zeta_{n+1} \rangle} \qquad \dfrac{(2)}{\langle \varphi_n \oplus \psi_n \rangle\ \mathsf{assume}\ \neg b\ \langle \psi_n \rangle}\ \textsc{Iter}}{\langle \varphi_0 \oplus \psi_0 \oplus \zeta_0 \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \psi_\infty \oplus \zeta_\infty \rangle}$$

Lemma 21. *The following rule is derivable:*

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

Proof. We will derive this rule using WHILE. For all $n \in \mathbb{N}$, let

$$\varphi_n \triangleq \blacksquare(P \wedge b) \qquad \psi_n = \psi_\infty \triangleq \blacksquare(P \wedge \neg b) \qquad \zeta_n = \zeta_\infty \triangleq \exists u.\ \Uparrow^{(u)}$$

We show that the necessary premises hold:

- ▷ Take any $(m_n)_{n \in \mathbb{N}}$ such that $m_n \vDash \psi_n$ for all n. That is, $m_n \vDash \blacksquare(P \wedge \neg b)$, so $\emptyset \subset \mathsf{supp}(m_n) \subseteq (P \wedge \neg b)$. Then,

$$\emptyset \subset \mathsf{supp}\left(\sum_{n \in \mathbb{N}} m_n\right) = \bigcup_{n \in \mathbb{N}} \mathsf{supp}(m_n) \subseteq (P \wedge \neg b)$$

  By definition, $\sum_{n \in \mathbb{N}} m_n \vDash \blacksquare(P \wedge \neg b) = \psi_\infty$. Thus, $(\psi_n)_{n \in \mathbb{N}} \rightsquigarrow \psi_\infty$.

- ▷ Again, take any $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \varphi_n \oplus \psi_n \oplus \zeta_n$ for all n. Note that $\zeta_\infty = \exists u : U.\ \Uparrow^{(u)} = \{m \mid \mathsf{supp}(m) \subseteq \{\circlearrowleft\}\}$. Thus, it always holds that
$$\left(\inf_{n\in N} |m_n| \cdot \top\right) \cdot \eta(\circlearrowleft) \vDash \zeta_\infty$$
We have $(\varphi_n \oplus \psi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty$.
- ▷ Clearly, $\blacksquare(P \wedge b) \vDash b$, $\blacksquare(P \wedge \neg b) \vDash \neg b$, and $\exists u : U.\ \Uparrow^{(u)} \vDash \mathsf{div}$ for all n.

The derivation is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle P \wedge b\rangle\ C\ \langle \Box P\rangle}{\langle \Box(P \wedge b)\rangle\ C\ \langle \Box P\rangle}\ \textsc{Lemma 13 - Partial}}{\langle \blacksquare(P \wedge b) \oplus \exists u : U.\ \Uparrow^{(u)}\rangle\ C\ \langle \blacksquare(P \wedge b) \oplus \blacksquare(P \wedge \neg b) \oplus \exists u : U.\ \Uparrow^{(u)}\rangle}\ \textsc{Consequence}}{\langle \blacksquare(P \wedge b) \oplus \blacksquare(P \wedge \neg b) \oplus \exists u : U.\ \Uparrow^{(u)}\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \blacksquare(P \wedge \neg b) \oplus \exists u : U.\ \Uparrow^{(u)}\rangle}\ \textsc{While}}{\langle P\rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Box(P \wedge \neg b)\rangle}\ \textsc{Consequence}$$

Lemma 22. *The following rule is derivable:*

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

Proof. Once again, we use While. For all N and n, define

$$\varphi'_n \triangleq \begin{cases} \varphi_{N-n} & \text{if } n < N \\ \mathbb{0} \odot \top & \text{otherwise} \end{cases} \qquad \psi_n \triangleq \begin{cases} \varphi_0 & \text{if } n \in \{N, \infty\} \\ \mathbb{0} \odot \top & \text{otherwise} \end{cases} \qquad \zeta_n = \zeta_\infty \triangleq 1_0$$

We give the necessary premises:

- ▷ Take any $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \psi_n$ for all $n \in \mathbb{N}$. Then, $m_N \vDash \varphi_0$ while $m_n \vDash \mathbb{0} \odot \top$ for $n \neq N$. So $\sum_{n\in\mathbb{N}} m_n = m_N \vDash \varphi_0$. This gives us $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$.
- ▷ Take any $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \varphi_n \oplus \psi_n \oplus \zeta_n$. We know $|m_n| = \mathbb{0}$ for all $n \geq N$, so
$$\left(\inf_{n\in\mathbb{N}} |m_n| \cdot \top\right) \cdot \eta(\circlearrowleft) = \mathbb{0} \cdot \eta(\circlearrowleft) \vDash \mathbb{0} \odot \top$$
We have $(\varphi_n \oplus \psi_n \oplus \zeta_n) \vDash \zeta_\infty$.
- ▷ We have as premise that for all $n < N$, $\varphi_n, (\mathbb{0} \odot \top) \vDash b$, so $\varphi'_n \vDash b$. $\varphi_0, (\mathbb{0} \odot \top) \vDash \neg b$, so $\psi_n \vDash \neg b$. And finally $\zeta_n = \mathbb{0} \odot \top \vDash \mathsf{div}$.

To derive $\langle \varphi'_n\rangle\ C\ \langle \varphi'_{n+1} \oplus \psi_{n+1}\rangle$ for all n, we consider two cases: either $n < N$ or $n \geq N$. We take these subderivations as part (1):

$$\dfrac{\forall m < N.\ \langle \varphi_{m+1}\rangle\ C\ \langle \varphi_m\rangle}{\dfrac{\forall n < N.\ \langle \varphi_{N-n}\rangle\ C\ \langle \varphi_{N-(n+1)}\rangle}{\forall n < N.\ \langle \varphi'_n\rangle\ C\ \langle \varphi'_{n+1} \oplus \psi_{n+1}\rangle}}$$

$$\dfrac{\dfrac{}{\langle \top\rangle\ C\ \langle \top\rangle}\ \textsc{True}}{\dfrac{\langle \mathbb{0} \cdot T\rangle\ C\ \langle \mathbb{0} \odot \top\rangle}{\forall n \geq N.\ \langle \varphi'_n\rangle\ C\ \langle \varphi'_{n+1} \oplus \psi_{n+1}\rangle}}\ \textsc{Scale}$$

The derivation is completed as follows:

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

$$\dfrac{}{\langle \exists N : \mathbb{N}.\ \varphi_N \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \varphi_0 \rangle}$$

Lemma 23. *The following rules are derivable:*

$$\textsc{QInv-Demon}\quad \dfrac{P \vDash b \qquad \langle P \rangle\ C\ \langle \Box P \rangle}{\langle P \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Box \Uparrow \rangle} \qquad \textsc{QInv-Angel}\quad \dfrac{P \vDash b \qquad \langle P \rangle\ C\ \langle \Diamond P \rangle}{\langle P \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Diamond \Uparrow \rangle}$$

Proof. We show the full derivation for QInv-Angel; deriving the demonic version of the rule is very similar.

We use the While rule. For all n, take

$$\varphi_n \triangleq (\exists u : U/\{\mathbb{0}\}.\ P^{(u)}) \oplus \blacksquare b \qquad \psi_n \triangleq \blacksquare(\neg b) \qquad \zeta_n = \exists u : U.\ \Uparrow^{(u)}$$
$$\psi_\infty \triangleq \top \qquad \zeta_\infty \triangleq \exists u : U/\{\mathbb{0}\}.\ \Uparrow^{(u)}$$

Take any family of weighting functions $(m_n)_{n\in\mathbb{N}}$.

- ▷ If $m_n \vDash \psi_n$ for all n, it is trivial that $\sum_{n\in\mathbb{N}} m_n \vDash \top$. So $(\varphi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$.
- ▷ Suppose $m_n \vDash \varphi_n \oplus \psi_n \oplus \zeta_n$. Then, $m_n = p + q$ where $p \vDash \exists u : U/\{0\}.\ P^{(u)} = \varphi_n$, so $|m_n| > |p| > 0$. This means that, for weight $v > 0$,

$$\left(\inf_{n\in\mathbb{N}} |m_n \cdot \top|\right) \cdot \eta(\circlearrowleft) \ = \ v \cdot \eta(\circlearrowleft) \ \vDash \ \exists u : U/\{\mathbb{0}\}.\ \Uparrow^{(u)}$$

  Thus, $(\varphi_n \oplus \psi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty$.

Since $P \vDash b$, we know $\varphi_n = \exists u : U/\{\mathbb{0}\}.\ P^{(u)} \oplus \blacksquare b \vDash b$. Clearly, $\psi_n \vDash \neg b$ and $\zeta_n \vDash \mathsf{div}$.

Note that we can partition $\top = \blacksquare b \oplus \blacksquare(\neg b) \oplus \exists u : U.\ \Uparrow^{(u)}$. So, we have

$$\begin{aligned}\varphi_n \oplus \psi_n \oplus \zeta_n &= (\exists u : U/\{\mathbb{0}\}.\ P^{(u)} \oplus \blacksquare b) \oplus \blacksquare(\neg b) \oplus \exists u : U.\ \Uparrow^{(u)} \\ &= \exists u : U/\{\mathbb{0}\}.\ P^{(u)} \oplus \top \\ &= \Diamond P \\ \psi_\infty \oplus \zeta_\infty &= \exists u : U/\{\mathbb{0}\}.\ \Uparrow^{(u)} \oplus \top \ = \Uparrow^{(*)}\end{aligned}$$

Moreover, $\Diamond P \wedge \Box b = \varphi_n \oplus \zeta_n$. The derivation is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle P \rangle\ C\ \langle \Diamond P \rangle}{\langle \Diamond P \rangle\ C\ \langle \Diamond P \rangle}\ \textsc{Lemma 14}}{\langle \Diamond P \wedge \Box b \rangle\ C\ \langle \Diamond P \rangle}\ \textsc{Consequence}}{\langle \Diamond P \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Uparrow^{(*)} \rangle}\ \textsc{While}}{\langle P \rangle\ \mathsf{while}\ b\ \mathsf{do}\ C\ \langle \Uparrow^{(*)} \rangle}\ \textsc{Consequence}$$