TOTAL OUTCOME LOGIC: UNIFIED REASONING FOR NONTERMINATION AND BRANCHING EFFECTS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

James Li,

Based on Joint work with Noam Zilberstein and Alexandra Silva

August 2025

ABSTRACT

Since their inception in the form of Hoare Logic, program logics have evolved far beyond simple imperative programs, expanding to handle memory alongside additional computational effects such as nondeterminism and probabilistic branching. Orthogonally, more recent research has taken a turn toward incorrectness logics, which prove the presence of bugs rather than their absence, offering greater tractability for industrial adoption.

However, correctness and incorrectness reasoning have remained fundamentally incompatible, forcing practitioners to use separate frameworks. This gap was filled by the recently developed Outcome Logic [41]. By decomposing reasoning on the reachability of separate program outcomes, Outcome Logic was able to unify both paradigms as well as generalize across different branching effects.

Our work extends Outcome Logic to address one of its blind spots: namely, reasoning about program (non)termination. While specialized logics exist for termination criteria like partial correctness (termination not assured) and total correctness (guaranteed termination), Outcome Logic can only express partial correctness and its dual—guaranteed divergence. In particular, the semantics of Outcome Logic captures only terminating outcomes, precluding reasoning on nonterminating behavior.

We expand the syntax, semantics, and proof rules of Outcome Logic to express the full spectrum of termination conditions of programs. As we will see, the mixture of nontermination and nondeterminism requires a nontrivial reworking of the program semantics. Our resulting framework is shown to be

sound and relatively complete, as well as expressive enough to subsume recently proposed taxonomies of modern program logics. As a proof of concept, we provide case studies demonstrating the ability to perform different kinds of termination analyses in our logic.

BIOGRAPHICAL SKETCH

James has been a Cornellian for six years. He graduated with a BA in Computer Science and Mathematics in 2023. Inspired by the programming languages community, he remained in Ithaca to do his MS in Computer Science, advised by Professor Alexandra Silva.

Cornell has instilled in him a love of teaching. He has served on course staff for the Computer Science department for six semesters and worked as an ESL tutor. Outside of academics, James was a member of Cornell Lion Dance, and likes to read philosophy and learn foreign languages. He takes joy in gardening, cooking, and the smell of old books.

To she, who makes me jump with joy.

ACKNOWLEDGEMENTS

There are many to thank for this thesis. More than anyone, I owe my growth as a researcher to my advisor Alexandra and my collaborator Noam. Alexandra has been an unceasingly supportive and welcoming figure in the PL community at Cornell, and always reassuring. Noam has guided me through many of the ideas and rough patches of this paper, and I am constantly surprised by his insight. His contributions to this project, and to my understanding of program logics in general, are manifold. I am grateful for his patience and advice, and I continue to have much to learn from him.

I would like to thank Harold Hodes, who painstakingly combed through drafts of this thesis and met with me to propose revisions. The presentation of this paper benefited greatly from our discussions. More generally, I cherish the numerous faculty in computer science and philosophy that have stoked my curiosity, and with whom I have had the pleasure of taking classes and teaching.

For all the joyous whimsy, I thank my MS cohort. For growing in tandem with my thesis and grounding me back to the earth, I thank my garden. For every day with you, I thank my partner, Anna Grace.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Hoare Logic

The challenge of program correctness was recognized as early as 1949 by Alan Turing in his conference paper titled "Checking a Large Routine" [27]. Foreseeing that programmers would have to contend with bugs, he proposed a systematic way of manually proving a program correct by means of annotating program fragments with assertions "which can be checked individually, and from which the correctness of the whole program follows". This approach, merely a vision when put forth by Turing, was formalized by Floyd and Hoare, who advanced a semantics for simple imperative languages, the proof obligations for verifying different commands, and finally a formal deductive framework for dispatching proofs.

Hoare Logic emerged from this branch of work as the first program logic. The basic formulae of Hoare Logic are *program triples* of the form:

$$\{P\}\ C\ \{Q\}$$

P and Q are assertions describing the state of the program before and after executing the program C, respectively. This forms a partial correctness specification: the triple is valid iff whenever C is executed from a state satisfying P and terminates, the resulting state satisfies Q.

As a deductive system, Hoare Logic provides a set of inference rules for deriving such triples given some triples as premises. For instance, take the SE-

QUENCE rule, which enables us to compose triples across a sequence of two programs:

$$\frac{\{P\}\ C_1\ \{R\} \qquad \{R\}\ C_2\ \{Q\}}{\{P\}\ C_1 \fatsemi C_2\ \{Q\}}\ \textsc{Sequence}$$

Other rules include those for reasoning over if-statements, as well as a rule for verifying loops by way of invariants. Thus, Hoare Logic enabled a principled system for proving correctness of programs that was compositional—building larger proofs from smaller fragments.

1.2 Incorrectness Logic

Program logics gradually phased into larger efforts to automate reasoning on programs. Unfortunately, proving correctness proved mostly intractable for large industrial codebases, requiring extensive annotations and complete verification across all execution paths. Facebook's Infer tool demonstrated a more pragmatic approach, using separation logic to detect memory bugs rather than prove complete correctness. This success inspired the development of Incorrectness logic [32], a logic geared specifically for bug-finding. It does so first in altering the interpretation of its triples. Now,

$$[P]\ C\ [Q]$$

is valid iff all states in Q are reachable by executing C from some state in P. Intuitively, Q represents a class of error states whose reachability we want to establish.

This logic is *underapproximate*; the triple above stipulates that Q be a subset of states reachable from P by way of C. This is dual to overapproximate triples such as in Hoare Logic, for which the postcondition Q is instead a *superset* of those reachable states.

1.3 Outcome Logic

While correctness and incorrectness are two sides of the same coin, their logics involve incompatible interpretations of triples (overapproximation vs. underapproximation) as well as distinct proof rules.

Outcome Logic, from which our present work proceeds, captures both notions in a single theory. Outcome Logic generalizes Hoare Logic to programs that branch, *e.g.*, by nondeterministic or probabilistic choice, preserving the ability to reason about correctness. One of its main innovations is in lifting program reasoning to algebraic representations of choice—for instance, sets of states or probability distributions—which enables it to express the reachability of outcomes. As a result, Outcome Logic can also encode underapproximate triples, as well as broader notions of incorrectness and *manifest* errors, for which Incorrectess Logic is unsuited. Outcome Logic thus generalizes across two dimensions of program properties: (1) correctness vs. incorrectness and (2) branching effects.

Another fundamental dimension of program analysis concerns termination. There is a long tradition of reasoning about whether a program halts in program analysis. Hoare triples do not guarantee termination, and thus are *partial*

correctness specifications. Shortly after Hoare, [3] added this termination assurance to triples and extended Hoare Logic to *total correctness*. Automated proofs of termination have since emerged as a subject of research [22, 25]. Conversely, principles for reasoning about nontermination have been studied in [23], [28] and most recently in a logic called UNTER [45].

The semantics of Outcome Logic tracks only the terminating outcomes of a program, remaining agnostic to any nonterminating executions. Consequently, it is restricted to partial correctness; Outcome Logic can prove whether a program sometimes terminates or always diverges (by witnessing no terminating outcomes), but it cannot prove whether a program always terminates or sometimes diverges. This limitation motivates our work: we extend Outcome Logic to handle the full spectrum of termination properties. We propose a program semantics for branching and nontermination, powering a framework that can express total correctness, possible divergence, and other termination criteria while preserving Outcome Logic's unified approach to correctness and incorrectness reasoning.

Remark. We omitted details of the theory of Outcome Logic in this introduction; since our project is an extension of Outcome Logic, our framework shares much of the same technical machinery, which will be presented in later sections. For further reading on the applications and case studies of Outcome Logic, see [41, 46].

CHAPTER 2

WEIGHTED COMMAND LANGUAGE

Weighted programming [35], [46] is a paradigm which generalizes branching effects such as nondeterminism and probabilistic choice by assigning each branch of execution with a *weight* $u \in U$. We adopt the weighted programming language from Outcome Logic [46] as shown in Figure 2.1.

2.1 Syntax

Our syntax for programs consists of typical imperative commands such as **`skip`**, *i.e.,* the program which does nothing, assertions **`assume`** e, and sequential composition $\fatsemi$. Primitive (uninterpreted) program commands are supplied as *actions* $a \in \mathsf{Act}$. We also include in the language nondeterministic choice $C_1 + C_2$, which nondeterministically chooses to execute one of the branches C_1 or C_2. It is worth highlighting that the guard e of the **`assume`** e command is not restricted to just Boolean expressions. The 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 **`assume`** e

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

Figure 2.1: Syntax of a simple weighted imperative language, parametric on a set of basic program actions Act and a set of weights U.

is that the computation trace in which this is executed is weighed by u. This is formalized later in our semantics.

Using choice + and **assume** with Boolean tests, we can define syntactic sugar for if statements:

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

The syntax for loops $C^{\langle e_1, e_2 \rangle}$ involves 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. This generalizes while loops found in typical imperative languages, which have one Boolean guard b (with the exit guard being the negation $\neg b$) and which can be encoded in our language as:

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

2.2 Nontermination Semantics for Weighted Branching

A few algebraic definitions are needed to define the semantics of our language. Informally, we can think of each computation trace of a weighted program as labeled with some weight $u \in U$. Since programs are compositional, we require additional structure on U to specify precisely how weights combine, both along a single trace and across different traces. For instance, how do we formalize the meaning of commands like: **assume** $3 \fatsemi a \fatsemi$ **assume** 5 or (**assume** $3 \fatsemi a_1$) + (**assume** $3 \fatsemi a_2$)? Our approach, drawn from [35] and [46], requires the set of weights U to have the structure of a semiring, which is an algebraic structure that combines two *monoids*.

Definition 2.2.1 (Monoid). A *monoid* $\langle U, +, \mathbb{0} \rangle$ consists of a carrier set U, an associa-

tive 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 $\langle U, +, \mathbb{0} \rangle$ is a *partial* monoid. If $+$ is commutative ($u + v = v + u$), then the monoid is *commutative*.

Definition 2.2.2 (Semiring). A (partial) *semiring* $\langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ is an algebraic structure such that:

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

Example 2.2.1 (Semirings). The following semirings encode different kinds of computation.

1. The Boolean semiring $\langle \mathbb{B}, \vee, \wedge, 0, 1 \rangle$ models nondeterminism by using Boolean $\mathbb{B} = \{0, 1\}$ weights to indicate whether an outcome is possible (1) or impossible (0). Addition is disjunction and multiplication is conjunction.

   Restricting $\vee$ to be partial such that $1 \vee 1$ is undefined allows us to model deterministic computation (intuitively, by allowing only one trace at a time).
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 standard arithmetic operations. This also serves as a model of nondeterminism, in which we count the traces leading to each outcome, *i.e.,* tracking multisets of outcomes.

For more examples, refer to [35] and [46].

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

Definition 2.2.3 (Weighting Function). Given a set of program states Σ and a (partial) semiring $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$, the set of weighting functions is

$$\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \triangleq \left\{ m : \Sigma_{\circlearrowleft} \to U \;\middle|\; |m| \text{ is defined and } \mathsf{supp}(m) \text{ is countable} \right\}$$

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 cumulative weight in m.

Intuitively, a weighting function is a snapshot of a weighted computation across all its possible traces; it describes how the computation has been distributed across different possible outcomes (including divergence), with each outcome assigned a weight reflecting its probability, cost, or other quantitative measure in the current branching configuration. This allows us to captures two distinct computational effects in a single object: weighted branching and nontermination.

We can now assign to each program command C a denotation $[\![C]\!] : \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ as shown in Figure 2.2. We leave several operations underspecified,

$$
\begin{aligned}
[\![\texttt{skip}]\!](\sigma) &\triangleq \eta(\sigma) & [\![a]\!](\sigma) &\triangleq [\![a]\!]_{\mathsf{Act}}(\sigma) \\
[\![C_1 \mathbin{;} C_2]\!](\sigma) &\triangleq [\![C_2]\!]^\dagger([\![C_1]\!](\sigma)) & [\![C_1 + C_2]\!](\sigma) &\triangleq [\![C_1]\!](\sigma) + [\![C_2]\!](\sigma) \\
[\![\texttt{assume}\ e]\!](\sigma) &\triangleq [\![e]\!](\sigma) \cdot \eta(\sigma) & \left[\!\left[C^{\langle e_1, e_2\rangle}\right]\!\right](\sigma) &\triangleq \mathsf{lfp}\left(\Phi_{C^{\langle e_1,e_2\rangle}}\right)(\sigma)
\end{aligned}
$$

$$
\text{where} \quad \Phi_{C^{\langle e_1,e_2\rangle}}(f)(\sigma) \triangleq [\![e_1]\!](\sigma) \cdot f^\dagger([\![C]\!](\sigma)) + [\![e_2]\!](\sigma) \cdot \eta(\sigma)
$$

Figure 2.2: Denotational semantics $[\![C]\!] : \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_\cup)$ of programs, parametric on a set of program states Σ; an interpretation $[\![a]\!]_{\mathsf{Act}} : \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_\cup)$ for atomic actions $a \in \mathsf{Act}$; and $\mathsf{Test} \subseteq 2^\Sigma$, the set of primitive tests.

which we now explain in detail, from simpler to more complex: 1. semantics of guards and branching; 2. monadic structure η and $(-)^\dagger$ that provide semantics of **skip** and sequential composition; and 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 $[\![e]\!] : \Sigma \to U$, where $[\![b]\!](\sigma) \triangleq [\![b]\!]_{\mathsf{Test}}(\sigma)$ and $[\![u]\!](\sigma) \triangleq u$. The semantics of Boolean guards $[\![b]\!]_{\mathsf{Test}} : \Sigma \to \{\mathbb{0}, \mathbb{1}\}$ is a simple extension of the set-element predicate (where $\mathbb{0}, \mathbb{1}$ are the semiring identities),

since primitive tests are sets of program states $\mathsf{Test} \subseteq 2^{\Sigma}$.

$$[\![\mathsf{true}]\!]_{\mathsf{Test}}(\sigma) \triangleq \mathbb{1} \qquad [\![\mathsf{false}]\!]_{\mathsf{Test}}(\sigma) \triangleq \mathbb{0}$$

$$[\![b_1 \vee b_2]\!]_{\mathsf{Test}}(\sigma) \triangleq \begin{cases} \mathbb{1} & \text{if } [\![b_1]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \text{ or } [\![b_2]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \\ \mathbb{0} & \text{otherwise} \end{cases}$$

$$[\![b_1 \wedge b_2]\!]_{\mathsf{Test}}(\sigma) \triangleq \begin{cases} \mathbb{1} & \text{if } [\![b_1]\!]_{\mathsf{Test}}(\sigma) = [\![b_2]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \\ \mathbb{0} & \text{otherwise} \end{cases}$$

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

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

Monadic structure of weighting functions.

Monads [5, 14] provide a mathematical framework for defining the denotational semantics of computational effects. Our semantics of **`skip`** and sequential composition relies on adding monadic structure to $\mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$.

Definition 2.2.4 (Monads). A *Kleisli triple* $\langle T, \eta, (-)^{\dagger} \rangle$ in the category **Set** consists of a functor $T : \mathbf{Set} \to \mathbf{Set}$, a natural transformation $\eta : \mathsf{Id} \Rightarrow T$, and, for any sets X and Y, a map $(-)^{\dagger} : (X \to T(Y)) \to T(X) \to T(Y)$ such that $\eta_X^{\dagger} = \mathsf{id}_X$, $f^{\dagger} \circ \eta = f$, and $f^{\dagger} \circ g^{\dagger} = (f^{\dagger} \circ g)^{\dagger}$. If such a triple exists, then the functor T is a monad.

We can view $\mathcal{W}_{\mathcal{A}}(-_{\cup})$ as a functor which augments any supplied set of program states Σ with the additional structure of weighting functions $\mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$, encoding both branching and nontermination. We supply $\eta_{\Sigma} : \Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$ and

$(-)^\dagger : (\Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_\circlearrowleft)) \to (\mathcal{W}_{\mathcal{A}}(\Sigma_\circlearrowleft) \to \mathcal{W}_{\mathcal{A}}(\Sigma_\circlearrowleft))$ as monadic operations for *lifting* states to weighting functions and for *sequencing effects*, respectively. More generally:

Definition 2.2.5. For any (partial) semiring $\mathcal{A}$, $\langle \mathcal{W}^{\circlearrowleft}_{\mathcal{A}}, \eta, (-)^\dagger \rangle$ is a Kleisli triple, where

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

Note that, in our semantics, we are only concerned with monadic structure with respect to Σ, and so we write η to mean η_Σ in particular.

We write $\delta_\circlearrowleft$ to mean the weighting function assigning a weight of 1 to $\circlearrowleft$ and 0 to everything else:

$$\delta_\circlearrowleft(\sigma) = \begin{cases} 0 & \text{if } \sigma \in \Sigma \\ 1 & \text{if } \sigma = \circlearrowleft \end{cases}$$

Observe that the *Kleisli extension* $f^\dagger(m)$ discriminates between weight allocated to program states (in Σ) and weight allocated to $\circlearrowleft$ by m. This anticipates how we model sequencing programs C_1 and C_2 (see Figure 2.2). Intuitively, the terminal outcomes of C_1 are fed as input to C_2 whereas nonterminating traces of C_1 never reach C_2—they must be "carried through" the second command. Although we define the monad operations directly here, $\mathcal{W}_{\mathcal{A}}(-_\circlearrowleft)$ can also be defined via a distributive law [1] between the weighting function monad of Outcome Logic [46] and the +1 or *error* monad, which is known to compose with all other monads [20].

Loops.

The semantics of iterated computations $C^{\langle e_1, e_2 \rangle}$ is defined as the least fixpoint of the functional $\Phi_{C^{\langle e_1, e_2 \rangle}}$ capturing the recursive structure of the loop. In line with the Scott-Strachey tradition of denotational semantics, demonstrating the existence of this fixpoint requires that $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ be equipped with a partial order $\sqsubseteq$, and that $(\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}), \sqsubseteq)$ be *directed-complete*:

Definition 2.2.6 (Directed Set). Let P be a partially-ordered set (poset). Then a subset $D \subseteq P$ is directed iff it is nonempty, and every pair of elements $x, y \in D$ have an upper bound in D.

Definition 2.2.7 (Directed-Complete Partial Order). A poset P is a directed-complete partial order (dcpo) if every directed subset $D \subseteq P$ has a supremum.

We use a *fusion order*: $m_1 \sqsubseteq m_2$ if and only if the weight of each program state increases and the weight of nontermination decreases. This is similar to the fixpoint maximal trace semantics of Cousot [19] in which finite traces are compared covariantly and infinite traces are compared contravariantly.

Definition 2.2.8 (Fusion Order). We define the *fusion order* $\sqsubseteq$ on $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ as:

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

Above, $\leq$ is the *natural order* 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 *naturally-ordered*. The semiring is *bounded* if there exists a top element $\top_{\mathcal{A}}$ such that $u \leq \top_{\mathcal{A}}$ for all $u \in U$. This top element is sometimes required to be an infinity, as defined below.

Definition 2.2.9 (Infinity [17]). An element w in semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ is an *infinity* if it is additively absorbing: $u + w = w + u = w$ for all $u \in U$. In addition, w is a

strong infinity if it is infinite as well as multiplicatively absorbing: $x \cdot w = w \cdot x = w$ for all $x \neq \mathbb{0}$.

In domain theory, it is common to view $\sqsubseteq$ as an *information ordering* [16]—$m_1 \sqsubseteq m_2$ means that m_2 contains more information about the program's behavior than m_1. This induces a notion of *approximation* that allows us to model loops; at the beginning of a loop, we initially consider the behavior to be nontermination (bottom), and the approximation becomes more complete as more terminating executions are discovered.

Guaranteeing a fixpoint also requires our semantics to be *Scott-continuous*, meaning that semantic functions preserve suprema of directed sets. In particular, it requires we show that:

Lemma. *For all iteration commands* $C^{\langle e,e' \rangle}$ *and directed* $D \subseteq \left(\Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})\right)$,

$$\sup_{f \in D} \Phi_{C^{\langle e,e' \rangle}}(f)(\sigma) = \Phi_{C^{\langle e,e' \rangle}}(\sup D)(\sigma)$$

Note that the order on functions $\left(\Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})\right)$ is taken to be pointwise with respect to the fusion order.

Unfortunately, this does not hold in all semirings of weights, *e.g.*, when $\mathcal{A}$ is the Boolean semiring and models nondeterministic computation. There are well-known barriers to continuity in a denotational semantics with both nondeterminism and nontermination, tracing back to an impossibility result by [12]. The difficulties arise when our semantic domain admits *unbounded nondeterminism*—when we can model programs that produce a infinite number of possible outcomes *without ever diverging*.

To see why the discontinuity arises, consider the program $x := \star$, which

nondeterministically assigns x to a natural number. Suppose we compose this program with a loop:

$$x := \star \fatsemi \textbf{while } x > 0 \textbf{ do } x := x - 1$$

Does the above program terminate? For each individual branch, x is set to some $n \in \mathbb{N}$ and the loop terminates in n iterations. The problem is that we cannot make this argument *compositionally*. Indeed, a nonterminating trace exists in every finite approximation of the loop. This means that standard techniques for proving termination such as loop variants and ranking functions will not suffice.

Curiously, the issues with unbounded choice do not arise when choice is probabilistic. Termination of probabilistic programs has been studied extensively [21, 31, 30, 39, 48], and there are many examples 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 all $n \geq 1$. The probability of termination after n iterations is $1 - \frac{1}{2^n}$, which clearly converges to 1 using a simple compositional proof.

For the purposes of ensuring continuity of loops in our semantics, we categorize these types of choice into two classes. Probabilistic programs are *conservative*—probability mass is treated like a resource, with the total amount of mass (1) being split between branches of computation and nontermination, so the weight of infinite traces can converge to 0 in the limit. In a nondeterministic setting, the weight on nontermination is *indicative*—there is an infinite amount of total *weight*, and spawning new branches does not consume anything. In conservative weightings—like probabilistic programs—it is possible to have com-

positional inference rules for termination with unbounded choice, whereas in indicative weightings it is not possible.

Restrictions on the Domain. Rather than proving the full continuity of our semantics—which is impossible due to unbounded nondeterminism—we prove a conditional continuity result that holds when $[\![C]\!]$ maps into a restricted subset $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ within $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$.

We can formalize the two types of choice reflected in nondeterministic vs. probabilistic branching. In particular, classify our nontermination semantics as either:

1. ***Conservative***. For this semantics, $\mathcal{A}$ is partial and bounded with (additively) *non-idempotent* top element $\top$ such that $x + \top = \top + x = \top$ implies $x = 0$.

   We restrict our semantics to:

   $$\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid m(\circlearrowleft) = \sup_{m' \in \mathsf{Equiv}_{\Sigma}(m)} m'(\circlearrowleft)\}$$

   For any particular $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, let $\mathsf{Equiv}_{\Sigma}(m)$ 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 $\circlearrowleft$):

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

   Programs under a *conservative weighting* scheme always produce a weighting function of fixed mass $\top$, which is conserved across sequences of programs. Examples include probabilistic and deterministic programs, for which the mass of weighting functions is fixed at 1. In fact, tt can be shown that under our listed properties, $\top$ of a conservative semiring must always be the additive identity (Lemma A.0.4).

2. ***Indicative***. Here, $\mathcal{A}$ is total and bounded with a *strongly infinite* top element $\top$. We prove continuity on the restricted domain:

$$\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma) \triangleq \{m \in \mathcal{W}^{\circlearrowleft}_{\mathcal{A}}(\Sigma) \mid \mathsf{supp}(m) \text{ is finite} \vee m(\circlearrowleft) = \top\}$$

   Note that this restriction generalizes the notion of *bounded nondeterminism*. Operationally, it captures the behavior that programs can only select between finitely many branches at each step of execution. That is, $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ comprises weighting functions that can be generated via such programs. It also echoes [11]'s definition of bounded nondeterminism for powerdomains, which requires that nondeterministic programs either (i) produce a finite set of results or (ii) diverge. Note that this implies that infinite sets of outcomes must be generated by one or more nonterminating trace.

   This can also be seen as a generalization of the Plotkin powerdomain, which includes all finite sets of program states, as well as all infinite state sets that contain a bottom element representing nontermination [6, 15].

   We refer to this type of semantics as *indicative* since the weight on $\circlearrowleft$ serves merely as an indicator for nontermination; we assign a weight of $\mathbb{0}$ if the computation does not produce a diverging trace, and assign a weight of $\top$ otherwise.

The introduction of these restrictions admittedly makes the framework less parsimonious, but we argue that they do not significantly limit the generalizability our approach. By construction, they capture the major types of branching of concern in program analysis: nondeterminism, probability, and determinism.

The restriction of conservative semantics, when instantiated for probabilistic choice, is simply the requirement that our restricted domain correspond

to probability distributions (where total mass is 1). In Section B.2, we define well-formed programs for which this semantics is total.

We also 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 [17, p. 166]. In particular, we adjoin to $\mathcal{A}$ an element $\infty \notin U$ to accommodate an *indicative* weighting scheme. Addition and multiplication can be defined such that $\infty + u = u + \infty = \infty$ for all $u \in U$; $\infty \cdot u = u \cdot \infty = \infty$ for all $u \in U \setminus \{\mathbb{0}\}$; and $\infty \cdot \mathbb{0} = \mathbb{0} \cdot \infty = \mathbb{0}$. 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, even necessary to preserve continuity.

This coarseness frequently arises from the nature of nontermination. Take the natural number semiring $\langle \mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1 \rangle$, which models nondeterminism via multisets (by tracking the number of occurrences of possible outcomes). The number of diverging traces of a program is not always well-defined; a program can periodically branch and prune its branches, such that this number oscillates forever. A weight of ∞ on $\circlearrowleft$ thus serves to merely indicate possibility, rather than encode quantity.

The final condition we need to impose to achieve a continuous semantics is Scott-continuity of the semiring $\mathcal{A}$ itself:

Definition 2.2.10 (Scott-Continuity). A semiring $\langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ with order $\leq$ is *Scott-continuous* if it is a dcpo and, for any directed set $D \subseteq U$ (where all pairs of

elements in D have a supremum):

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

The semiring is *lower Scott-continuous* if it is a dcpo with respect to the dual order $\geq$ and the same operations preserve infima.

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 [24]:

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. In particular, we require $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ to be a naturally ordered, complete, Scott-continuous, (partial) semiring bounded by top element $\top_{\mathcal{A}}$.

CHAPTER 3

TOTAL OUTCOME LOGIC

Our nontermination semantics now enables us to extend Outcome Logic to prove termination guarantees, possible nontermination, as well as more fine-grained results about the specific weight with which a program diverges. In this section, we present the theory of *Total* Outcome Logic (TOL), which extends the system by [46] to handle such (non)termination criteria——conditions that form a major axis of program properties targeted by existing program logics such as *total correctness* in Total Hoare Logic [3].

3.1 Outcome Assertions

First, we redefine *outcome assertions*, which will serve as pre- and postconditions of the triples in TOL . In Hoare Logic [2], pre- and post-conditions are predicates on program states Σ, which correspond to subsets of states 2^{Σ}. By contrast, assertions in Outcome Logic are lifted to be predicates on *weighting functions*. This renders them much more powerful, as they characterize not just individual program states but entire branching configurations of (weighted) nondeterministic programs; outcome assertions describe which outcomes are (im)possible and with what distribution of weights they can occur. Mirroring the approach by [46], we define outcome assertions semantically: assertions $\varphi, \psi \in 2^{\mathcal{W}_{\mathcal{A}(\Sigma_\cup)}}$ are *subsets* of weighting functions. Satisfaction $m \vDash \varphi$ (read as 'm satisfies φ') is defined as set membership $m \in \varphi$.

Below, we introduce useful propositional notation for assertions. Truth con-

stants $\top, \bot$ and the standard logical connectives are defined as:

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

$$\varphi \vee \psi \triangleq \varphi \cup \psi \qquad \varphi \wedge \psi \triangleq \varphi \cap \psi \qquad \varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi$$

Given a predicate on program states $P \subseteq \Sigma$, we define liftings to indicate that P over-approximates, under-approximates, or exactly characterizes the support of a weighting function with weight u. Similarly, $\Uparrow^{(u)}$ denotes that the nontermination outcome $\circlearrowleft$ occurs with weight $u \cdot \top_{\mathcal{A}}$.

$$\lceil P \rceil^{(u)} \triangleq \{m \mid |m| = u, \mathsf{supp}(m) \subseteq P\} \qquad \lfloor P \rfloor^{(u)} \triangleq \{m \mid |m| = u, \mathsf{supp}(m) \supseteq P\}$$

$$[P]^{(u)} \triangleq \{m \mid |m| = u, \mathsf{supp}(m) = P\} \qquad \Uparrow^{(u)} \triangleq \{u \cdot \delta_{\circlearrowleft}\}$$

We omit the superscript when the weight is exactly $\mathbb{1}$, *i.e.*, $\lceil P \rceil = \lceil P \rceil^{(\mathbb{1})}$. Operations $u \odot \varphi$ and $\varphi \odot u$ scale the outcome assertion φ with weight u on the left or right. 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.

$$u \odot \varphi \triangleq \{u \cdot m \mid m \in \varphi\} \qquad \varphi \odot u \triangleq \{m \cdot u \mid m \in \varphi\} \qquad \exists x : T.\, \phi(x) \triangleq \bigcup_{t \in T} \phi(t)$$

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 over an 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 $\{m\}$.

3.2 Triples

The formulae of TOL and Outcome Logic are *outcome triples* of the form $\langle \varphi \rangle\ C\ \langle \psi \rangle$, for precondition φ, program command C, and postcondition ψ.

Definition 3.2.1 (Outcome Triple). An outcome triple is *valid* iff:

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

This resembles the usual *overapproximate* interpretation of triples in Hoare Logic [2]: we say that $\langle \varphi \rangle\ C\ \langle \psi \rangle$ holds if ψ is a superset of the set of weighting functions that may result from running C starting in a configuration in φ. However, one of the main insights of Outcome Logic is that a broader class of reachability conditions can be expressed by lifting assertions to weighting functions and partitioning the distribution of results using $\oplus$. As such, Outcome Logic can capture both program correctness and incorrectness [41, 46], and as can TOL (rather trivially, which we touch on in Chapter 4).

3.3 Deductions

Inference rules for deriving triples in TOL are given in Figures 3.1 and 3.2. Many of these rules carry over from [46], but some must be adapted to account for the additional effect of nontermination. We present them in three categories: structural rules, non-iterative command rules, and a rule for loop commands.

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

$$\frac{\langle \varphi \rangle\ C\ \langle \psi \rangle}{\langle u \odot \varphi \rangle\ C\ \langle u \odot \psi \rangle}\ \textsc{Scale} \qquad \frac{\langle \varphi_1 \rangle\ C\ \langle \psi_1 \rangle \qquad \langle \varphi_2 \rangle\ C\ \langle \psi_2 \rangle}{\langle \varphi_1 \vee \varphi_2 \rangle\ C\ \langle \psi_1 \vee \psi_2 \rangle}\ \textsc{Disj}$$

$$\frac{\langle \varphi_1 \rangle\ C\ \langle \psi_1 \rangle \qquad \langle \varphi_2 \rangle\ C\ \langle \psi_2 \rangle}{\langle \varphi_1 \wedge \varphi_2 \rangle\ C\ \langle \psi_1 \wedge \psi_2 \rangle}\ \textsc{Conj} \qquad \frac{\left\{ \langle \phi(t) \rangle\ C\ \langle \phi'(t) \rangle \right\}_{t \in T}}{\langle \bigoplus_{x \in T} \phi(x) \rangle\ C\ \langle \bigoplus_{x \in T} \phi'(x) \rangle}\ \textsc{Choice}$$

$$\frac{\left\{ \langle \phi(t) \rangle\ C\ \langle \phi'(t) \rangle \right\}_{t \in T}}{\langle \exists x : T.\ \phi(x) \rangle\ C\ \langle \exists x : T.\ \phi'(x) \rangle}\ \textsc{Exists}$$

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

Figure 3.1: Structural rules.

3.3.1 Structural Rules.

(Figure 3.1). Structural rules hold for arbitrary program commands C and vary according to the logical structure of pre- and post-conditions. These are unchanged from Outcome Logic [46] except for Div.

We briefly introduce the original rules first. Included are the axioms False and True, which hold for a vacuous precondition $\bot$ and universal postcondition $\top$, respectively. 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. We also have the standard Consequence rule, which allows for strengthening preconditions and weak-

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

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

Figure 3.2: Inference rules for non-iterative program commands.

ening postconditions.

The axiom Div handles nontermination captured by the precondition, stating that assertions $\Uparrow^{(u)}$ can be pushed through triples. The intuition is that traces diverging prior to a command C never reach execution of C, and are thus unaffected by it.

3.3.2 Non-iterative Command Rules.

(Figure 3.2). These rules specify how outcome assertions are propagated through *non-iterative* program commands. Skip and Seq are standard for Hoare-style logics and are identical in Outcome Logic [46]. The Assume rule is conditioned by the *assertion entailment* $\varphi \vDash e = u$, defined below:

Definition 3.3.1 (Assertion Entailment). Given an assertion φ, expression e, and weight $u \in U$, we write $\varphi \vDash e = u$ (φ *entails* $e = u$) iff for all $m \vDash \varphi$, $\circlearrowleft \notin \mathsf{supp}(m)$ and $[\![e]\!](\sigma) = u$ for all $\sigma \in \mathsf{supp}(m)$.

That is, we take it to mean that for every weighting function $m \vDash \varphi$ and any $\sigma \in \mathsf{supp}(m)$, e evaluates to u in state σ. For tests $b \in \mathsf{Test}$, we introduce the shorthand $\varphi \vDash b$ to mean $\varphi \vDash b = \mathbb{1}$. In Plus, the side condition $\varphi \vDash \mathsf{true}$ is equivalent

to a termination guarantee, since $[\![\mathsf{true}]\!](\sigma) = \mathbb{1}$ always holds. Assuming this is satisfied, the triples proved for both branches can be combined with outcome conjunction.

Termination must be enforced in these cases since nonterminating traces are sequenced differently than terminating ones, as apparent in the definition of $(-)^{\dagger}$. Similar issues arise in instances of OL with exceptions [41, 47] and probabilistic nondeterminism [50].

3.3.3 Iteration Rule.

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

where the premises A_n and B_n are defined for each $n \in \mathbb{N}$ as follows:

$$A_n \triangleq \langle \varphi_n \oplus \zeta_n \rangle\ \mathtt{assume}\ e \fatsemi C\ \langle \varphi_{n+1} + \zeta_{n+1} \rangle$$
$$B_n \triangleq \langle \varphi_n \rangle\ \mathtt{assume}\ e'\ \langle \psi_n \rangle$$

Figure 3.3: Inference rule for iteration commands $C^{\langle e,e' \rangle}$.

Definition 3.3.2 (Divergence Entailment). For outcome assertion φ, we write $\varphi \vDash \mathsf{div}$ (read as: φ entails divergence) iff for all $m \in \varphi$, $\mathsf{supp}(m) \subseteq \{\circlearrowleft\}$.

Our main addition to the proof theory of Outcome Logic is in the Iter rule, which enables reasoning over iteration commands $C^{\langle e,e' \rangle}$ and, here, is expanded to reason about (non)termination. Its premises correspond to an infinite unrolling of the loop, with the assertions $(\varphi_n, \psi_n, \zeta_n)_{n\in\mathbb{N}}$ describing different computation traces at each loop iteration. Informally, we can imagine stepping through

the execution tree of $C^{\langle e,e'\rangle}$, advancing one iteration at a time and grouping each computation trace into one of three states: (1) **In-progress**, (2) **Terminated**, or (3) **Diverging**. Each family of assertions corresponds to one of these states:

(1) φ_n describes ongoing traces that reach iteration n and will continue to step to the next iteration. This excludes traces that escape the loop or diverge before ever reaching the nth iteration. The side condition $\varphi_n \vDash \mathsf{true}$ is, again, equivalent to a termination guarantee: for all $m \vDash \varphi_n$, $\circlearrowleft \notin \mathsf{supp}(m)$. We say that such weighting functions are *convergent*.

(2) ψ_n describes traces that terminate immediately following the nth iteration of the loop, *i.e.,* that escape the loop through the exit guard e'.

(3) ζ_n describes traces that diverge prior to reaching iteration n. This is reflected in the condition $\zeta_n \vDash \mathsf{div}$, which means that for all $m \vDash \zeta_n$, $\mathsf{supp}(m) \subseteq \{\circlearrowleft\}$, *i.e.,* all mass is constrained to nontermination.

These assertions are related according to the premises:

$$A_n \triangleq \langle \varphi_n \oplus \zeta_n \rangle \ \mathbf{assume}\ e \fatsemi C \ \langle \varphi_{n+1} \oplus \zeta_{n+1} \rangle \qquad B_n \triangleq \langle \varphi_n \rangle \ \mathbf{assume}\ e' \ \langle \psi_n \rangle$$

That is, A_n and B_n characterize the two branches at the nth iteration of the loop: a trace may either (A) pass through the loop guard e and execute the loop body, or (B) pass through the exit guard e' and terminate.

Note that ζ_{n+1} accounts for both nonterminating traces captured by ζ_n, as well as any traces that diverge while executing the loop body C during iteration n. In our language, it follows that ζ_n can only grow across iterations due to nested loops in C.

The postcondition of the conclusion consists of two assertions ψ_∞ and ζ_∞ which encode the *limiting* behavior of the loop. The former aggregates all the terminating traces caught by each ψ_n, and thus captures the loop's terminating outcomes. More formally, we write $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$, defined to mean:

Definition 3.3.3 (Converging Assertions). A family $(\psi_n)_{n\in\mathbb{N}}$ of assertions converges to ψ_∞ (written $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$) iff for any $(m_n)_{n\in\mathbb{N}}$, 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_∞ captures the nonterminating traces of the loop, which consist of:

(i) traces that diverge at some specific iteration n (the limit of accumulated nontermination captured by each ζ_n);

(ii) traces that continue indefinitely through all iterations without exiting the loop (the limit of continuing traces from each φ_n).

This notion is formalized by the condition $(\varphi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty$, which we define as follows:

Definition 3.3.4 (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 $(\inf_{n\in\mathbb{N}} |m_n| \cdot \top_{\mathcal{A}}) \cdot \delta_\circlearrowleft \vDash \psi_\infty$.

Remark 3.3.1 (Accommodating Nontermination). In general, applying 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 $\varphi = \varphi_t \oplus \varphi_\circlearrowleft$ where $\varphi_t \vDash \mathsf{true}$ and $\varphi_\circlearrowleft \vDash \mathsf{div}$. 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 $\langle \varphi_t \rangle\ C_2\ \langle \psi_{t_2} \rangle$:

$$\dfrac{\dfrac{\varphi_t \vDash \mathsf{true} \qquad \langle \varphi_t \rangle\ C_1\ \langle \psi_{t_1} \rangle \qquad \langle \varphi_t \rangle\ C_2\ \langle \psi_{t_2} \rangle}{\langle \varphi_t \rangle\ C_1 + C_2\ \langle \psi_{t_1} \oplus \psi_{t_2} \rangle}\ \textsc{Plus} \qquad \dfrac{\varphi_{\circlearrowleft} \vDash \mathsf{div}}{\langle \varphi_{\circlearrowleft} \rangle\ C_1 + C_2\ \langle \varphi_{\circlearrowleft} \rangle}\ \textsc{Div}^{*\,1}}{\langle \varphi_t \oplus \varphi_{\circlearrowleft} \rangle\ C_1 + C_2\ \langle \psi_{t_1} \oplus \psi_{t_2} \oplus \varphi_{\circlearrowleft} \rangle}\ \textsc{Choice}$$

We write $\Gamma \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$ to mean that the triple $\langle \varphi \rangle\ C\ \langle \psi \rangle$ is *derivable* using these inference rules (Figures 3.1 to 3.3) given a collection of axioms about atomic actions $\Gamma = \{\ldots, \langle \varphi' \rangle\ a\ \langle \psi' \rangle, \ldots\}$.

3.3.4 Soundness and Relative Completeness

We now establish the soundness and (relative) completeness of TOL .

Soundness concerns the *correctness* of our proof system. It guarantees that if a TOL triple is derivable, then it is also semantically valid (according to Definition 3.2.1). Let Ω consist of all triples $\langle \varphi \rangle\ a\ \langle \psi \rangle$ such that $a \in \mathsf{Act}$, and $\vDash \langle \varphi \rangle\ a\ \langle \psi \rangle$ (all true triples about atomic actions). Then, soundness of TOL can be stated formally as:

Theorem 3.3.1 (Soundness)**.**

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

The proof of soundness is given in Chapter C and proceeds by induction on the structure of the derivation $\Omega \vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$.

On the other hand, completeness deals with our system's scope of applicability. Formally, it is the converse of soundness—stating that any semanti-

cally valid triple can be derived using our rules. Unfortunately, no sufficiently expressive program logic is absolutely complete. In particular, systems over Turing-complete program languages can encode the halting problem: *e.g.*, take the Hoare triple $\{\mathsf{true}\}\ C\ \{\mathsf{false}\}$, which states that the program C never terminates [10]. In general, valid triples of this form cannot be derived due to the undecidability of halting.

As per the tradition for Hoare-style logics, we weaken the property to [8]'s notion of *relative* completeness, which states that TOL is complete provided an oracle to decide semantic properties such as the implications used in the rule of CONSEQUENCE and the assertion entailments used in the ASSUME rule.

The standard approach to proving relative completeness is to first show that triples of the form $\langle\varphi\rangle\ C\ \langle\mathsf{post}(C,\varphi)\rangle$ can be derived, where $\mathsf{post}(C,\varphi)$ is the *strongest postcondition* that makes $\langle\varphi\rangle\ C\ \langle\psi\rangle$ true:

Definition 3.3.5 (Strongest Postcondition). $\mathsf{post}(C,\varphi)$ consists of all weighting functions reachable from φ by executing command C:

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

Lemma 3.3.1. $\Omega \vdash \langle\varphi\rangle\ C\ \langle\mathsf{post}(C,\varphi)\rangle$

The proof of this intermediate lemma is also provided in Chapter C, and proceeds by induction of the structure of the program. Given this result, we can conclude relative completeness:

Theorem 3.3.2 (Relative Completeness)**.**

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

Proof. We show that $\mathsf{post}(C, \varphi) \implies \psi$. Suppose that $m \vDash \mathsf{post}(C, \varphi)$. Then by definition, there must exist some m' such that $m = [\![C]\!]^{\dagger}(m')$. Since $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$, we know that $m \vDash \psi$ as well. Then, we can apply CONSEQUENCE to derive the desired triple:

$$\dfrac{\dfrac{\Omega}{\langle \varphi \rangle\ C\ \langle \mathsf{post}(C, \varphi) \rangle}\ \textsc{Lemma 3.3.1} \qquad \mathsf{post}(C, \varphi) \implies \psi}{\langle \varphi \rangle\ C\ \langle \psi \rangle}\ \textsc{Consequence}$$

□

CHAPTER 4

UNIFICATION OF A LANDSCAPE OF TRIPLES

Outcome Logic unified reasoning about correctness and incorrectness, but without a semantics to explicitly handle nontermination, it could not express total correctness or the possibility of diverging traces. In this chapter, we demonstrate how such program properties can be encoded in TOL , filling the gap left by its predecessor.

To position TOL as a foundational theory for modern program logics, we also situate our contributions within the current ecosystem of such logics. Recent work has produced taxonomies of program logics that classify different interpretations of triples $\{P\}\ C\ \{Q\}$ across frameworks. These varying semantics correspond directly to different classes of program properties: *e.g.,* Total Hoare Logic extends standard triples to express total correctness, while Incorrectness Logic modifies triples to be underapproximate, enabling bug detection. Building on Outcome Logic's generality, we show how TOL subsumes the logics identified in these taxonomies, spanning the corresponding spectrum of program properties.

4.1 The Verscht and Kaminski [49] Taxonomy

We turn our focus to the taxonomy of program logics developed by Verscht and Kaminski [49], which systematically classifies 16 different logics for nondeterministic, looping programs. This taxonomy organizes logics along three main axes: (1) correctness vs. incorrectness, (2) partiality vs. totality, and (3) angelic vs. demonic nondeterminism. The logics are related through weakening, con-

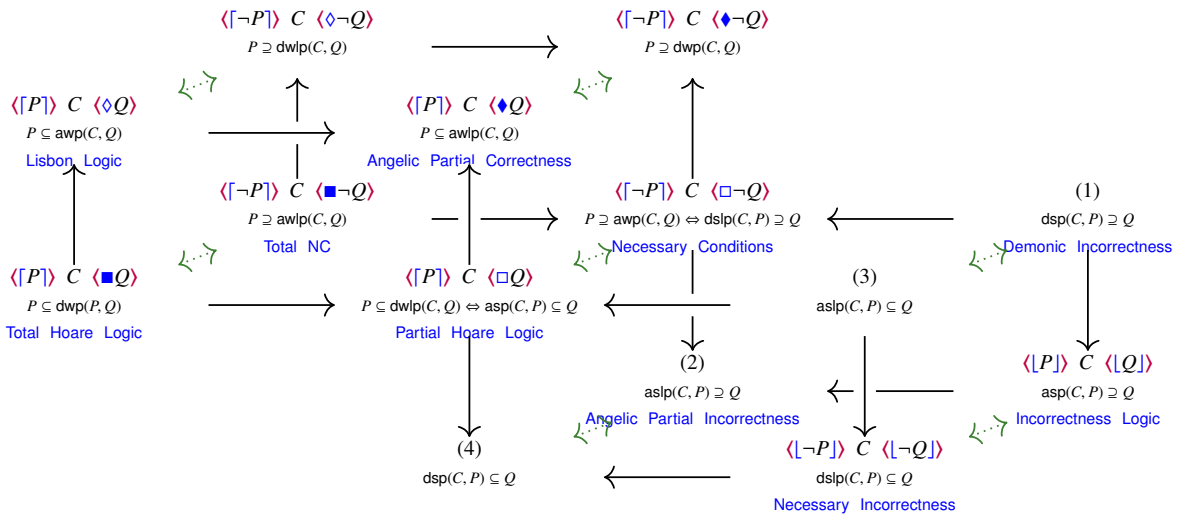


Figure 4.1: Adaptation of the Verscht and Kaminski [49] Taxonomy. Standard arrows represent implications and green dotted arrows represent contrapositives. The numbered logics are represented as: (1) $\langle\lceil\neg P\rceil\rangle\ C\ \langle\Box\neg Q\rangle \wedge \langle\lfloor P\rfloor\rangle\ C\ \langle\lfloor Q\rfloor\rangle$, (2) $\langle\lceil\neg P\rceil\rangle\ C\ \langle\Box\neg Q\rangle \vee \langle\lfloor P\rfloor\rangle\ C\ \langle\lfloor Q\rfloor\rangle$, (3) $\langle\lceil P\rceil\rangle\ C\ \langle\Box Q\rangle \wedge \langle\lfloor\neg P\rfloor\rangle\ C\ \langle\lfloor\neg Q\rfloor\rangle$, and (4) $\langle\lceil P\rceil\rangle\ C\ \langle\Box Q\rangle \vee \langle\lfloor\neg P\rfloor\rangle\ C\ \langle\lfloor\neg Q\rfloor\rangle$. [2]

traposition, and Galois connections, with some of these connections giving rise to entirely novel triple semantics. We display the full taxonomy in Figure 4.1, reformatted as a double cube.

In particular, the different semantics of triples are formulated in terms of predicate transformers——variants of weakest preconditions and strongest postconditions. We demonstrate how TOL subsumes all of these logics, focusing primarily on the major existing frameworks with the most practical interest.

4.1.1 Weakest Precondition Logics

The upper cube in Figure 4.1 is comprised of logics defined by different flavors of *weakest precondition* transformers. Originally due to [4], these transformers initially had two variants: the weakest precondition of a program C relative to

postcondition Q is the set of states that always terminate in Q after C is run, whereas the weakest *liberal* precondition is the set of start states that end in Q *if* the program terminates. Both of these transformers treat nondeterminism demonically, so [49] refer to them as the demonic weakest precondition (dwp) and demonic weakest liberal precondition (dwlp), respectively.

$$\mathsf{dwp}(C, Q) \triangleq \{\sigma \mid \mathsf{supp}([\![C]\!](\sigma)) \subseteq Q\}$$

$$\mathsf{dwlp}(C, Q) \triangleq \{\sigma \mid \mathsf{supp}([\![C]\!](\sigma)) \subseteq Q_{\cup}\}$$

It is well known that total and partial correctness Hoare Logic [2, 3] can be framed in terms of these transformers. That is, the total correctness Hoare Triple $\{P\}\ C\ \{Q\}$ is equivalent to $P \subseteq \mathsf{dwp}(C, Q)$ and the partial correctness triple $\{P\}\ C\ \{Q\}$ is equivalent to $P \subseteq \mathsf{dwlp}(C, Q)$. Corresponding angelic versions of these transformers can be defined as so:

$$\mathsf{awp}(C, Q) \triangleq \{\sigma \mid \mathsf{supp}([\![C]\!](\sigma)) \cap Q \neq \emptyset\}$$

$$\mathsf{awlp}(C, Q) \triangleq \{\sigma \mid \mathsf{supp}([\![C]\!](\sigma)) \cap Q_{\cup} \neq \emptyset\}$$

The angelic weakest precondition (awp), originally due to [9] under the name *weakest possible precondition*, is the set of states that reach Q via *some* trace in C. The resulting logic $P \subseteq \mathsf{awp}(C, Q)$—known as Lisbon Logic [41], Backwards Under-Approximation [33], and Sufficient Incorrectness Logic [38]—is useful for proving incorrectness, *i.e.*, that certain bugs are reachable. The angelic weakest liberal precondition is less well studied. It is the set of states from which the program either reaches the postcondition or diverges, and was introduced by [49] in order to complete the front face of the upper cube in Figure 4.1. A special instance of awlp is featured in UNTer, a logic for proving possible nontermination [45]. In particular, the condition $P \subseteq \mathsf{awlp}(C, \mathsf{false})$ is equivalent to the existence of a diverging trace of program C initially satisfying P. We encode

these logics in TOL using the following modalities; recall that here we assume that $\mathsf{supp}(m) \neq \emptyset$.

$$\Box P \triangleq \exists(u,v) : U^2 \setminus \{(\mathbb{0},\mathbb{0})\}.\ \lceil P \rceil^{(u)} \oplus \Uparrow^{(v)} \qquad = \{m \mid \mathsf{supp}(m) \subseteq P_{\circlearrowleft}\}$$

$$\Diamond P \triangleq \exists u : U \setminus \{\mathbb{0}\}.\ \lceil P \rceil^{(u)} \oplus \top \qquad = \{m \mid \mathsf{supp}(m) \cap P \neq \emptyset\}$$

$$\blacksquare P \triangleq \exists u : U \setminus \{\mathbb{0}\}.\ \lceil P \rceil^{(u)} \qquad = \{m \mid \mathsf{supp}(m) \subseteq P\}$$

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

The $\Box$ and $\Diamond$ modalities are inspired by Dynamic Logic [7, 18] and correspond to dwlp and awp, respectively. That is, $\Box Q$ states that Q covers all the terminating states in some weighting function m, and $\Diamond Q$ states that m contains some state in Q. As was shown by [46], $\Box$ and $\Diamond$ can be used to encode partial correctness Hoare Logic and Lisbon Logic:

Theorem 4.1.1 (Subsumption of Hoare and Lisbon Logic)**.**

$$(i) \quad P \subseteq \mathsf{dwlp}(C,Q) \iff \vDash \langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle$$

$$(ii) \quad P \subseteq \mathsf{awp}(C,Q) \iff \vDash \langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle$$

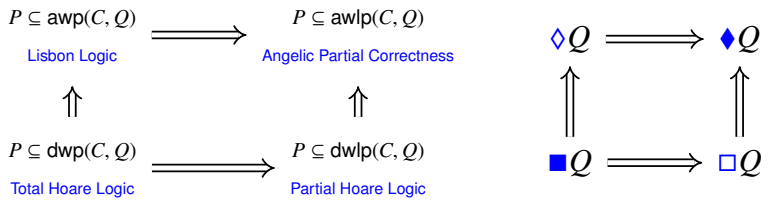
The $\blacksquare$ and $\blacklozenge$ modalities must directly witness a particular (zero and nonzero) amount of nontermination mass, and were therefore not expressible in standard Outcome Logic [41, 46, 47]. More precisely $\blacksquare Q$ states not only that Q covers all the reachable states, but also that $\circlearrowleft$ has weight zero. Finally, $\blacklozenge Q$ states that either Q or $\circlearrowleft$ occur with nonzero weight. These new modalities can be used to express the final logics:

Theorem 4.1.2 (Subsumption of Total Hoare Logic and Angelic Partial Correctness)**.**

$$(i) \quad P \subseteq \mathsf{dwp}(C,Q) \iff \vDash \langle \lceil P \rceil \rangle\ C\ \langle \blacksquare Q \rangle$$

$$(ii) \quad P \subseteq \mathsf{awlp}(C,Q) \iff \langle \lceil P \rceil \rangle\ C\ \langle \blacklozenge Q \rangle$$

The front face of the upper cube in Figure 4.1 is now fully accounted for. Logics in this quadrant are related by arrows, indicating that one triple can be logically weakened to another. A corresponding commuting square holds for modalities:



Contrapositive Logics.

The back face of the upper cube is obtained by taking contrapositives of the four aforementioned logics, *i.e.*, by negating the pre- and postconditions. Because of the de Morgan style dualities between the modalities and predicate transformers (*e.g.*, $\Box P$ iff $\neg\Diamond\neg P$ and $\blacksquare P$ iff $\neg\blacklozenge\neg P$), these logics can also be expressed by flipping the $\subseteq$ to be a $\supseteq$.

For instance, the contrapositive of Hoare Logic is given by $\neg P \subseteq \mathsf{dwlp}(C, \neg Q)$, which is equivalent to $P \supseteq \mathsf{dwlp}(C, Q)$. This interpretation of triples was named *Necessary Conditions* (NC) by Cousot et al. [26]. Indeed, in an NC triple $\{P\}\ C\ \{Q\}$, P is a necessary condition to reach Q, which means that all states outside of P must not reach Q. The remaining contrapositive logics of the upper cube have not been studied. Note that in subsuming the triples of one logic, TOL also trivially subsumes its contrapositive.

4.1.2 Strongest Postcondition Logics

The lower cube in Figure 4.1 is framed in terms of strongest postconditions [13], which give the set of states reachable from the precondition. Verscht and Kaminski refer to the classical strongest postcondition as the *angelic* strongest postcondition (awp), as the states must only be reachable from *some* start state, not *all* start states. We restate Definition 3.3.5 here:

$$\mathsf{asp}(C, P) \triangleq \{\tau \mid \exists \sigma \in P.\ \tau \in \mathsf{supp}([\![C]\!](\sigma))\}$$

There is a well known Galois connection between dwlp and asp, meaning that partial Hoare Logic can be framed in terms of both:

$$P \subseteq \mathsf{dwlp}(C, Q) \iff \mathsf{asp}(C, P) \subseteq Q$$

Then, there is the demonic strongest *liberal* postcondition (dslp), which is dual to asp in the sense that $\mathsf{asp}(C, P) = \overline{\mathsf{dslp}(C, \neg P)}$. Here, liberality corresponds to unreachability of the postcondition rather than nontermination from the precondition, so $\mathsf{dslp}(C, P)$ contains states that are unreachable from outside of P:

$$\mathsf{dslp}(C, P) \triangleq \{\tau \mid \forall \sigma \notin P.\ \tau \notin \mathsf{supp}([\![C]\!](\sigma))\}$$

Cousot and [37] observed another Galois connection between awp and dslp, meaning that NC also has two characterizations:

$$P \supseteq \mathsf{awp}(C, Q) \iff \mathsf{dslp}(C, P) \supseteq Q$$

For completeness, we include the remaining two transformers, although they do not have obvious intuitions or use cases, and as [49, §3.2.4] point out, they cannot be defined inductively. As such, we do not define them directly, but rather represent them equivalently as combinations of other transformers that

we have seen.

$$\mathsf{aslp}(C, P) = \mathsf{asp}(C, P) \cup \mathsf{dslp}(C, P)$$

$$\mathsf{dsp}(C, P) = \mathsf{asp}(C, P) \cap \mathsf{dslp}(C, P)$$

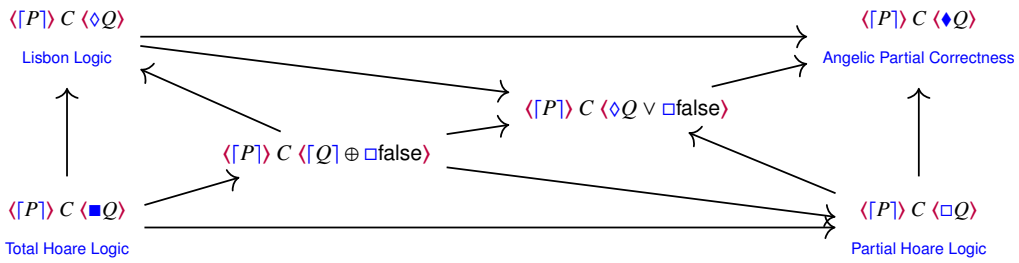
We can now describe the logics in the lower cube of Figure 4.1. Due to the aforementioned Galois connections, the top left edge overlaps with the upper cube, and describes partial Hoare Logic and NC. The only other logic in the cube that has been studied is Incorrectness Logic (IL), in the back right, which is obtained by swapping the $\subseteq$ for a $\supseteq$ in the asp definition of Hoare Logic, *i.e.*, an IL postcondition under-approximates the reachable states whereas a Hoare Logic postcondition over-approximates [32]. As was shown by [43, Proposition 6], IL can be encoded using under-approximate assertions $\lfloor P \rfloor$, *i.e.*, $m \vDash \lfloor P \rfloor$ iff $\mathsf{supp}(m) \supseteq P$.

The remaining logics in the lower cube are not particularly well-understood. Indeed, [49] showed that these logics have no inductive calculi, and accordingly we encode them in TOL using two triples. For example, Demonic Incorrectness states both that P is a necessary condition to reach Q ($\langle \lceil \neg P \rceil \rangle\ C\ \langle \Box \neg Q \rangle$), and also that every state in Q is reachable from P ($\langle \lfloor P \rfloor \rangle\ C\ \langle \lfloor Q \rfloor \rangle$), *i.e.*, it is similar to standard incorrectness logic, but with the additional stipulation that P covers *all* the states that could reach the bug. No concrete applications have been found for these logics; as of now, they exist only to neatly complete the cube.

4.1.3 In-Between Logics

Although Figure 4.1 appears comprehensive, more logics can be formed by taking unions and intersections of the aforementioned predicate transformers. Fo-

cusing on the front face of the upper cube, [49] form new logics by taking the conjunction and disjunction of Lisbon and Partial Hoare Logic. These *in-between* logics can be encoded in TOL as follows:



The first new logic offers triples of the form $\langle\lceil P\rceil\rangle\ C\ \langle\lceil Q\rceil \oplus \Box\mathsf{false}\rangle$—$Q$ is reachable and is the only terminating outcome, but nontermination is possible too. This is the precise meaning of $\langle\lceil P\rceil\rangle\ C\ \langle\lceil Q\rceil\rangle$ in standard Outcome Logic [46], as its semantic model could not identify the existence of nonterminating traces. This logic has a clear application in that it unifies partial correctness and incorrectness reasoning within a single logic [41].

The second new logic has the form $\langle\lceil P\rceil\rangle\ C\ \langle\Diamond Q \vee \Box\mathsf{false}\rangle$, meaning that either Q is reachable or C never terminates. The applications for this second logic are less clear, being weaker than both Hoare Logic and Lisbon Logic. Still, expressing it in TOL at least gives us compositional rules to derive specifications, whereas Verscht and Kaminski must derive it using two separate calculi. More broadly, the existence of these in-between logics show that Figure 4.1 does not tell the complete story; *how many other logics are in-between?* In Section 4.2, we explore this question further and see how TOL is more expressive than any one of the predicate transformers that we have seen thus far.

4.2 A New Taxonomy for Correctness and Incorrectness

Practical Program Logics

First, we ought to ground our discussions of the previous taxonomies in practical applications. We have shown that TOL has the expressive power to subsume all the logics in both the Verscht and Kaminski [49] taxonomy of Hoare-like logics and the Cousot [42] cube. However, although the prior two sections have explored more than 16 different program logics, only a few of them have found practical use. The front face of the upper cube in Figure 4.1—containing (partial and total) Hoare Logic and Lisbon Logic—is well understood, and serves as the foundation of decades of program analysis research. Incorrectness Logic has served as a semantic justification for real-world static analysis systems [**isl**, 36], although it was more recently shown that Lisbon Logic also models those systems [47, 45] and has some advantages in its ability to identify the cause of a bug [41, 38].

The remaining logics are not readily applicable to real static analysis problems. Their underlying concepts—demonic strongest postconditions (being reachable from *all* start states) and liberal postconditions (being unreachable from any start state)—neatly form Galois connections and complete the cubes, but defy intuitions and do not map to any of the properties about programs that have arisen in the past several decades of static analysis research. Even worse, some of these properties cannot be described using inductive rules [49].

These taxonomies are obtained by taking strict binaries on what is actually a spectrum of program properties. This is captured by the metatheory of TOL ,

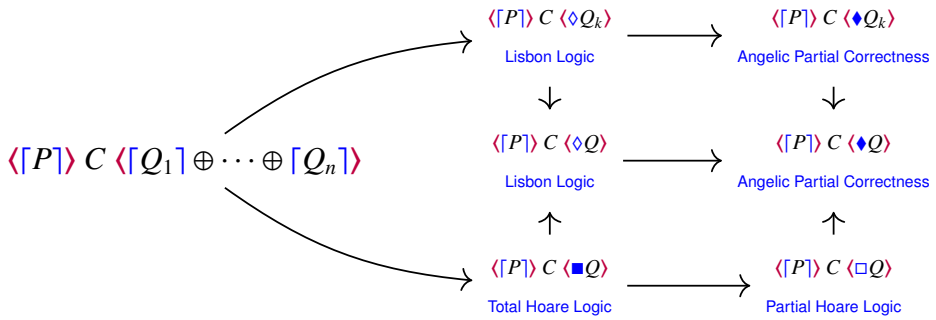
which lifts reasoning to the level of (weighted) sets of outcomes. As a result, we can encode finer reachability and termination requirements as outcome assertions rather than building them into the interpretation of our triples. This lends TOL far greater versatility than simply instantiating the 16+ different logics, as we are no longer confined to the corners of the cube, but can navigate freely in the volume within.

Importantly, this enables shared program analyses in TOL for different properties. While Figure 4.1 suggests that Total Hoare Logic is the *strongest* logic in that upper front square, this gives only a limited, and perhaps misleading, picture as to the compatibility of the logics in that square.

Total Hoare triples give over-approximate correctness specifications, while Lisbon triples target the existence of a more select set of outcomes—potential bugs, for instance. More precisely, suppose that Q_{ok} represents some collection of good states and Q_{er} represents bugs states. It may be the case that the bug described by Q_{er} *sometimes* occurs and so the Lisbon triple $\langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q_{\mathsf{er}} \rangle$ is valid. But since the bug does not *always* occur, total Hoare Logic can only use a weaker postcondition $\langle \lceil P \rceil \rangle\ C\ \langle \blacksquare(Q_{\mathsf{ok}} \vee Q_{\mathsf{er}}) \rangle$. These two specification are incompatible; there is no weakening relation between them. Similar issues arise between the other logics when postconditions differ. Some logics are wholly incompatible—*i.e.*, one type of triple cannot be used in a subderivation of the other whatsoever, such as between partial correctness $\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle$ and Lisbon Logic $\langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle$—rendering them isolated and disconnected for the purpose of analysis.

In TOL , we can instead perform the bulk of our program analysis on finer-grained intermediate specifications, where we track *reachable* outcomes of inter-

est with $\oplus$ in the form of assertions: $\lceil Q_1 \rceil \oplus \cdots \oplus \lceil Q_n \rceil$. Then, letting $Q = Q_1 \vee \cdots \vee Q_n$ and $1 \leq k \leq n$, we can weaken these to our desired property with an application of CONSEQUENCE, as shown below.



(4.1)

The commuting square from the front of the upper cube in Figure 4.1 still appears, but above it there is also a Lisbon triple with a stronger postcondition $Q_k \Rightarrow Q$, which is incomparable to the total Hoare triple below it. Yet, both specifications stem from a common TOL triple, shown on the left of the figure. This is significant because helper functions in a codebase can be given specifications in TOL and then be repurposed for total correctness, partial correctness, and/or incorrectness.

With this in mind, we present our full taxonomy in Figure 4.2, which is a superset of the one above. The full taxonomy includes more of the recently introduced logics for (in)correctness, and (non)termination, including Incorrectness Logic [**il**]. All of these logics are arranged on the front face of the diagram, and common TOL specifications are placed behind.

Recall that neither Figure 4.1 nor **??** provide ways to compare Hoare Logic and Incorrectness Logic. The key to doing so is Exact Logic, which expresses both kinds of triples together [40]. Exact Logic is encoded in TOL using exact predicates $[P]$, stating that the reachable states are exactly P. Whereas, the original Exact Logic was based on partial correctness, we also include a new Total

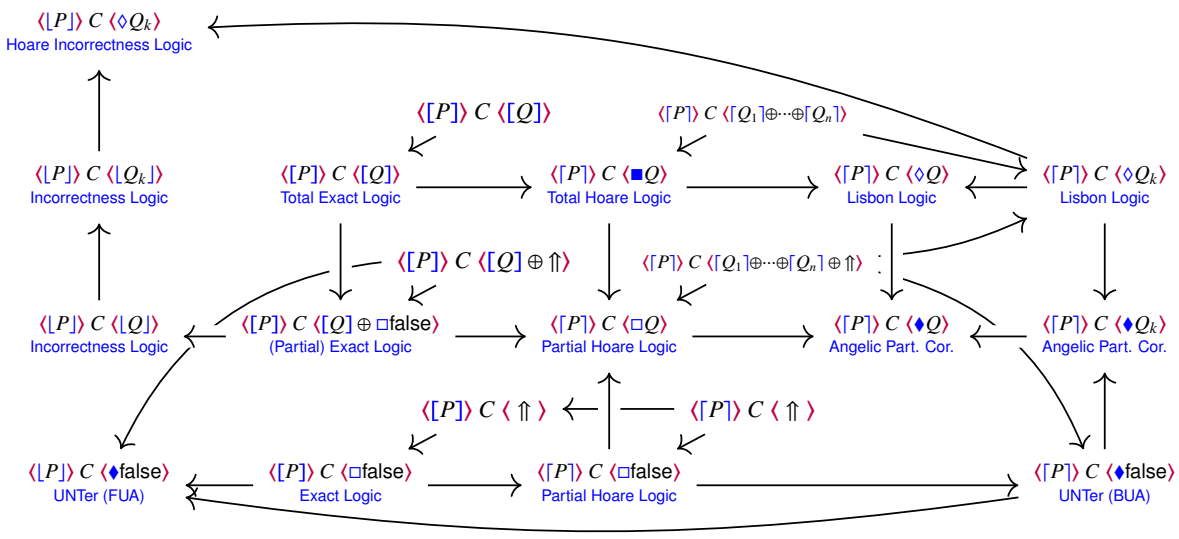


Figure 4.2: A new taxonomy of correctness and incorrectness logics, where $Q = Q_1 \vee \cdots \vee Q_n$ and $1 \leq k \leq n$.

Exact Logic, which additionally guarantees termination.

But despite our ability to encode Incorrectness and Exact Logics in TOL, there are advantages to forgoing them and instead using variations of Lisbon Logic for bug-finding [41, 38, 45]. Whereas Incorrectness Logic witnesses *some* erroneous trace starting from the precondition, Lisbon Logic ensures that the bug can be witnessed from *all* start states. This is an important property in program analysis, making it easier to find *manifest errors*—bugs that can occur regardless of context [36]. Exact Logic was developed for symbolic execution [44], but is not an ideal foundation for other kinds static analysis as it has no weakening rule, and therefore cannot perform reasoning in abstract domains to make use of, *e.g.*, loop invariants [47, 34]. Case in point: there is no weakening of the exact triple $\langle [P] \rangle\ C\ \langle [Q] \rangle$ to use the more precise postcondition $[Q_k]$.

By contrast, TOL offers more versatility to navigate between correctness and incorrectness specifications. With this in mind, we turn our attention to the por-

tion of the taxonomy containing Partial and Total Hoare Logic, along with everything to the right. This is an expanded version of (4.1) above, where the unifying power of TOL shines. In the back there are three forms of TOL triples. Programs in the first row always terminate, programs in the second row sometimes terminate, and programs in the third row never terminate. A single TOL triple can subsume partial correctness, total correctness, Lisbon Logic, and nontermination, all of which have been successfully deployed in industrial static analysis tools. TOL unifies all these kinds of correctness and incorrectness, making it an ideal logical foundation. In next sections, we will develop reasoning principles for TOL and illustrate their versatility through a few simple case studies.

CHAPTER 5

DERIVED RULES

The vision behind Outcome Logic and TOL is that of a unified theoretical foundation for program analysis—one which captures the different forms of reasoning scattered across various program logics. In the previous chapter, we show how TOL is able to express the specifications of many types of logics. It remains to show that TOL can also subsume their proof theories to derive similar or equivalent results. The focus and generality of TOL also drive some of its base rules to be rather fine-grained, abstract, and impractical. The ITER rule, in particular, is infinitary (like in Outcome Logic [46]) and requires establishing complicated semantic properties of assertions. In this chapter, we derive new rules for common patterns and convenient reasoning principles arising from logics we have previously seen.

5.0.1 If Statements, While Loops, and Divergence.

Recall from Section 2.1 that we gave syntactic sugar to define if statements and while loops in terms of the branching, iteration, and **`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.

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

The rule for while loops differs from standard Outcome Logic, as it includes nontermination information. We use three families of assertions: φ_n represents the outcomes where 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.

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

Outcome Logic can only reason about terminating outcomes, and is thus only able to detect nontermination when no terminating executions are witnessed, *i.e.*, via triples of the form $\langle \varphi \rangle_{\mathrm{OL}}\ C\ \langle \mathbb{0} \odot \top \rangle_{\mathrm{OL}}$. Whereas this restricts proofs to *possible termination* and *guaranteed nontermination*, TOL now allows us to demonstrate *sure-termination* or *possible nontermination*.

In Total Hoare triples [3], the former is 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 guaranteed termination at which the loop guard fails.

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

In addition, a program cannot interfere with the existing weight on nontermination $\circlearrowleft$. 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.

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

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 $\langle \lceil P \rceil \rangle\ C\ \langle \blacksquare Q \rangle$), 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.

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

$$\frac{\langle \lceil P \rceil \rangle\ C_1\ \langle \lozenge Q \rangle \quad \langle \lceil Q \rceil \rangle\ C_2\ \langle \lozenge R \rangle}{\langle \lceil P \rceil \rangle\ C_1 \fatsemi C_2\ \langle \lozenge R \rangle}\ \textsc{Seq-Lisbon}$$

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.

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

$$\frac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \lozenge Q \rangle \qquad \langle \lceil P \wedge \neg b \rceil \rangle\ C_2\ \langle \lozenge Q \rangle}{\langle \lceil P \rceil \rangle\ \mathtt{if}\ b\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2\ \langle \lozenge Q \rangle}\ \textsc{If-Lisbon}$$

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.

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

We can also derive the standard variant rule employed in total Hoare Logic. The rule uses an integer-valued *ranking* expression R, which represents how close the program is to termination. In the premise, we are given that each iteration of the loop strictly decreases R, and the loop guard fails once R reaches 0. This allows us to conclude that the loop surely terminates:

$$\frac{P \wedge b \implies R > 0 \qquad \forall n \in \mathbb{N}.\ \langle\lceil P \wedge b \wedge R = n\rceil\rangle\ C\ \langle\blacksquare(P \wedge R < n)\rangle}{\langle\lceil P \wedge R \leq N\rceil\rangle\ \mathtt{while}\ b\ \mathtt{do}\ C\ \langle\blacksquare(P \wedge \neg b)\rangle}\ \textsc{Hoare-Variant}$$

Nontermination.

Bug-finding is often a more practical goal than correctness verification [32]. We can thus shift our sight from proving total correctness to proving the presence of nontermination by establishing a *quasi-invariant* [28]. 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* [23] 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. Using the modalities from Chapter 4, we get:

$$\blacklozenge\mathsf{false} = \exists u : U \setminus \{\mathbb{0}\}.\ \Uparrow^{(u)} \oplus \top \qquad\qquad \square\mathsf{false} = \exists u : U.\ \Uparrow^{(u)}$$

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

$$\frac{\lceil P \rceil \vDash b \qquad \langle \lceil P \rceil \rangle\ C\ \langle \Diamond P \rangle}{\langle \lceil P \rceil \rangle\ \texttt{while}\ b\ \texttt{do}\ C\ \langle \blacklozenge\mathsf{false} \rangle}\ \textsc{QInv-Angel} \qquad \frac{\lceil P \rceil \vDash b \qquad \langle \lceil P \rceil \rangle\ C\ \langle \square P \rangle}{\langle \lceil P \rceil \rangle\ \texttt{while}\ b\ \texttt{do}\ C\ \langle \square\mathsf{false} \rangle}\ \textsc{QInv-Demon}$$

5.1 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 proven in TOL , we deploy it on a few examples. In these examples, we use variable assignments $x := E$ as basic actions. These actions are defined in the usual way, as in [46].

5.1.1 Total Correctness: Quicksort Partition

In the classical formulation of Hoare logic for total correctness, [3] derive the correctness of the Quicksort partition algorithm, where A is an integer array of

size $n+1$ and p is the pre-computed pivot value:

$$\textsc{Partition} \triangleq \begin{cases} i := 0 \fatsemi j := n \fatsemi \\ \textbf{while}\ i \leq j\ \textbf{do} \\ \quad \textbf{if}\ A[i] \leq p\ \textbf{then}\ i := i+1 \fatsemi \\ \quad \textbf{else if}\ A[j] \geq p\ \textbf{then}\ j := j-1 \fatsemi \\ \quad \textbf{else}\ \textbf{swap}(A,i,j) \fatsemi i := i+1 \fatsemi j := j-1 \end{cases}$$

Total correctness of PARTITION can similarly be established in TOL . Again, we can encode a total correctness triple $[P]\ C\ [Q]$ as outcome triple $\langle \lceil P \rceil \rangle\ C\ \langle \blacksquare Q \rangle$. 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 p, define predicates α and β over integers:

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

We want to derive $\langle \lceil \mathsf{true} \rceil \rangle\ \textsc{Partition}\ \langle \blacksquare(\alpha(i) \wedge \beta(j) \wedge i > j) \rangle$, which states that any run of the program must terminate having divided the array into two sections: elements $\leq p$ and those $\geq p$. We decorate PARTITION in Figure 5.2 in Section 5.2.2 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 $\leq p$, while every element after j must be $\geq p$. 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:

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

$$\textsc{Nt} \triangleq \begin{cases} x := 1 \fatsemi y := 2 \fatsemi \\ \textbf{while}\ \ x + y > 1\ \ \textbf{do} \\ \quad x := 3 - x \fatsemi \\ \quad y := 3 - y \fatsemi \end{cases}$$

$$\textsc{MallocDiv} \triangleq \begin{cases} \textbf{byte} * p \fatsemi \\ p := (\textbf{byte}*)\,\textbf{malloc}(\textbf{size} + 16) \fatsemi \\ \textbf{while}\ \ p\ \ \textbf{= NULL do} \\ \quad p := (\textbf{byte}*)\,\textbf{malloc}(\textbf{size} + 16) \fatsemi \end{cases}$$

Figure 5.1: Two non-terminating programs.

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

5.2 Additional Case Studies

5.2.1 Proving Nontermination

[45] advocated the need for formal methods to prove nontermination, as many industrial codebases have bugs that cause programs to diverge and 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 5.1. To show that this program indeed always diverges, we make use of

the QINV-DEMON rule to derive the triple $\langle\lceil\mathsf{true}\rceil\rangle \mathrm{NT} \langle\Box\mathsf{false}\rangle$.

$\langle\lceil\mathsf{true}\rceil\rangle$

$x := 1 \fatsemi y := 2\fatsemi$

$\langle\lceil x = 1 \wedge y = 2\rceil\rangle$

$\Longrightarrow \langle\lceil x + y = 3\rceil\rangle$

$x := 3 - x\fatsemi$

$\langle\lceil 3 - x + y = 3\rceil\rangle$ //ASSIGN

$y := 3 - y\fatsemi$

$\langle\lceil (3 - x) + (3 - y) = 3\rceil\rangle$ //ASSIGN

$\Longrightarrow \langle\lceil x + y = 3\rceil\rangle$

$\Longrightarrow \langle\Box(x + y = 3)\rangle$

$\langle\Box\mathsf{false}\rangle$

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 5.1.

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

$$\frac{\langle\lceil p = \texttt{NULL}\rceil\rangle\ p := (\texttt{byte}*)\,\texttt{malloc}(\texttt{size} + 16)\ \langle\Diamond(p = \mathsf{NULL})\rangle}{\langle\lceil p = \texttt{NULL}\rceil\rangle\ \texttt{while}\ p = \texttt{NULL}\ \texttt{do}\ \ldots\ \langle\blacklozenge\mathsf{false}\rangle}\ \text{QINV-ANGEL}$$

And thus MALLOCDIV may diverge. Similar proofs of *guaranteed nontermination* can be derived using QINV-DEMON.

$\langle\lceil \text{true} \rceil\rangle$
$i := 0 \fatsemi j := n \fatsemi$
$\langle\lceil i = 0 \wedge j = n \rceil\rangle \implies \langle\lceil \alpha(i) \wedge \beta(j) \wedge j - i \geq 0 \rceil\rangle$
while $i \leq j$ **do**
  $\langle\lceil \alpha(i) \wedge \beta(j) \wedge i \leq j \wedge j - i = m \rceil\rangle \implies \langle\lceil \alpha(i) \wedge \beta(j) \wedge j - i = m \rceil\rangle$
  if $A[i] \leq p$ **then**
    $\langle\lceil \alpha(i) \wedge \beta(j) \wedge A[i] \leq p \wedge j - i = m \rceil\rangle \implies \langle\lceil \alpha(i+1) \wedge \beta(j) \wedge j - i = m \rceil\rangle$
    $i := i + 1 \fatsemi$
    $\langle\lceil \alpha(i) \wedge \beta(j) \wedge j - i = m - 1 \rceil\rangle$ // ASSIGN
  else if $A[j] \geq p$ **then**
    $\langle\lceil \alpha(i) \wedge \beta(j) \wedge A[j] \geq p \wedge j - i = m \rceil\rangle \implies \langle\lceil \alpha(i) \wedge \beta(j-1) \wedge j - i = m \rceil\rangle$
    $j := j - 1 \fatsemi$
    $\langle\lceil \alpha(i) \wedge \beta(j) \wedge j - i = m - 1 \rceil\rangle$ // ASSIGN
  else
    $\langle\lceil \alpha(i) \wedge \beta(j) \wedge A[i] > p \wedge A[j] < p \wedge j - i = m \rceil\rangle$
    swap$(A, i, j) \fatsemi$
    $\langle\lceil \alpha(i) \wedge \beta(j) \wedge A[j] > p \wedge A[i] < p \wedge j - i = m \rceil\rangle$
    $\implies \langle\lceil \alpha(i+1) \wedge \beta(j-1) \wedge j - i = m \rceil\rangle$
    $i := i + 1 \fatsemi j := j - 1$
    $\langle\lceil \alpha(i) \wedge \beta(j) \wedge j - i = m - 2 \rceil\rangle$ // ASSIGN
  $\langle\lceil \alpha(i) \wedge \beta(j) \wedge j - i < m \rceil\rangle$ // 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

Figure 5.2: Decorated proof of the PARTITION program.

5.2.2 Full Proof of Quicksort Partition

5.2.3 Probabilistic Nontermination

Whereas much of the work on probabilistic verification focuses on *almost sure termination* [31, 21, 30]—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 [29]. An example of such programs is a scenario where a tortoise marches forward at a constant speed, and an erratic hare attempts to catch up. [1]

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

$$
\begin{aligned}
&t \coloneqq 1 \fatsemi h \coloneqq 0 \fatsemi k \coloneqq 0 \fatsemi \\
&\textbf{while}\ h < t\ \textbf{do} \\
&\quad C_{\mathsf{body}} \triangleq \begin{cases} t \coloneqq t + 1 \fatsemi \\ (h \coloneqq h + 1) +_{\frac{1}{2}} (h \coloneqq h + 1 + \frac{1}{2^k}) \fatsemi \\ k \coloneqq k + 1 \end{cases}
\end{aligned}
$$

The hare catches the tortoise with probability $\frac{1}{2}$, when its initial move is a leap forward. If the hare's first move is not a leap, then it catches the tortoise with probability 0. We can see this informally, since in the best case, the hare catches up each round by $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots$, which converges to 1, 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 using 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. 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

[1] Example and analysis is due to Noam Zilberstein.

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

Figure 5.3: Derivation of the $n \geq 2$ case.

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 $\frac{1}{2^n}$.

$$
\begin{array}{ll}
\varphi_0 \triangleq \lceil t = 1 \wedge h = 0 \wedge k = 0 \rceil & \varphi_1 \triangleq \lceil t = 2 \wedge h = 1 \wedge k = 1 \rceil^{(\frac{1}{2})} \\
\varphi_n \triangleq \lceil t - h = \dfrac{1}{2^{n-1}} \wedge k = n \rceil^{(\frac{1}{2^n})} \oplus \lceil t - h > \dfrac{1}{2^{n-1}} \wedge k = n \rceil^{\left(\frac{2^{n-1}-1}{2^n}\right)} & \text{if } n \geq 2
\end{array}
$$

The ψ_n assertions describe the terminating outcomes. The program can terminate after the first step, so $\psi_1 \triangleq \lceil t = 2 \wedge h = 2 \wedge k = 1 \rceil^{(\frac{1}{2})}$, but $\psi_n \triangleq \lceil \mathsf{true} \rceil^{(0)}$ for every other $n \neq 1$. Finally, $(\psi_n)_{n \in \mathbb{N}} \rightsquigarrow \psi_\infty \triangleq \lceil h = t \rceil^{(\frac{1}{2})}$, the only terminating outcome. The ζ_n assertions describe nonterminating outcomes. The program only diverges in the limit, so we get $\zeta_n \triangleq \lceil \mathsf{true} \rceil^{(0)}$ for all $n \in \mathbb{N}$ and $\zeta_\infty \triangleq \Uparrow^{(\frac{1}{2})}$. 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 show the cases

where $n = 0$ and $n = 1$ below on the left and right, respectively.

$$\langle \varphi_0 \rangle \Leftrightarrow \langle \lceil t = 1 \wedge h = 0 \wedge k = 0 \rceil \rangle$$
$$t := t + 1 \fatsemi$$
$$\langle \lceil t = 2 \wedge h = 0 \wedge k = 0 \rceil \rangle$$
$$(h := h + 1) +_{\frac{1}{2}} (h := h + 1 + \tfrac{1}{2^k}) \fatsemi$$
$$\langle \bigoplus\nolimits_{i \in \{1,2\}} \lceil t = 2 \wedge h = i \wedge k = 0 \rceil^{(\frac{1}{2})} \rangle$$
$$k := k + 1$$
$$\langle \bigoplus\nolimits_{i \in \{1,2\}} \lceil t = 2 \wedge h = i \wedge k = 1 \rceil^{(\frac{1}{2})} \rangle$$
$$\Leftrightarrow \langle \varphi_1 \oplus \psi_1 \rangle$$

$$\langle \varphi_1 \rangle \Leftrightarrow \langle \lceil t = 2 \wedge h = 1 \wedge k = 1 \rceil^{(\frac{1}{2})} \rangle$$
$$t := t + 1 \fatsemi$$
$$\langle \lceil t = 3 \wedge h = 1 \wedge k = 1 \rceil^{(\frac{1}{2})} \rangle$$
$$(h := h + 1) +_{\frac{1}{2}} (h := h + 1 + \tfrac{1}{2^k}) \fatsemi$$
$$\langle \lceil t = 3 \wedge h = 2 \wedge k = 1 \rceil^{(\frac{1}{4})} \oplus \lceil t = 3 \wedge h = 2 + \tfrac{1}{2} \wedge k = 1 \rceil^{(\frac{1}{4})} \rangle$$
$$k := k + 1$$
$$\langle \lceil t = 3 \wedge h = 2 \wedge k = 2 \rceil^{(\frac{1}{4})} \oplus \lceil t = 3 \wedge h = 2 + \tfrac{1}{2} \wedge k = 2 \rceil^{(\frac{1}{4})} \rangle$$
$$\Rightarrow \langle \lceil t - h = \tfrac{1}{2} \wedge k = 2 \rceil^{(\frac{1}{4})} \oplus \lceil t - h > \tfrac{1}{2} \wedge k = 2 \rceil^{(\frac{1}{4})} \rangle$$
$$\Leftrightarrow \langle \varphi_2 \oplus \psi_2 \rangle$$

Finally, we show the case where $n \geq 2$ in Figure 5.3. We complete the proof with a straightforward application of the WHILE rule.

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

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 $\frac{1}{2}$.

CHAPTER 6
CONCLUDING REMARKS

Nontermination Semantics for Branching We have presented a sound and relatively complete extension of Outcome Logic that can reason about termination and nontermination. The resulting logic can prove total correctness as well as possible divergence, allowing it to subsume an additional dimension of existing frameworks.

This power was built off of our nontermination semantics for weighted programs. A few comments should be made about this, as this is where the author feels a bulk of any future work or improvements ought to be directed. The mixture of branching and nontermination is difficult to formalize using purely domain theory and the theory of the semiring. Our continuity results for the semantics would have benefited greatly from more structure on the domain—structure that we felt reluctant to add in fear of losing generalizability. For instance, probabilistic semantics typically have the full power of measure theory and functional analysis behind them, but our goal in this project was not to delve deep on one particular type of program. This tension resulted in the classification of indicative vs. conservative semantics, which was undoubtedly a strained effort, as the latter category only really captures deterministic and probabilistic programs.

The crux of guaranteeing a fully Scott-continuous semantics for nontermination and branching (at least using the fusion order $\sqsubseteq$) boils down to showing a minimax equality:

Lemma. *Let X be a dcpo. For an infinite family of lower Scott-continuous functions*

$(f_i : X \to U)_{i \in I}$ and directed $D \subseteq X$ (with respect to the dual order):

$$\sup_{J \subseteq I} \inf_{x \in D} \sum_{i \in J} f_i(x) = \inf_{x \in D} \sup_{J \subseteq I} \sum_{i \in J} f_i(x)$$

The left is straightforwardly $\leq$ the right, but the reverse direction does not hold in general. In fact, for the Boolean semiring, this fails, which is related to the difficulties of unbounded nondeterminism and nontermination [12]. We conjecture this could be proved for certain semirings provided it be equipped with additional structure, and a more clever semantics could side-step it altogether.

Reusing Proof Fragments The promise of providing a single deductive framework for a variety of program properties is the ability to reuse proof fragments. By placing correctness, incorrectness, branching, and now termination on the same foundation, Outcome Logic enables us to use common specifications that can be lowered to prove each of the aforementioned targets. This offers a more parsimonious basis for program analysis, with potential gains in efficiency. Further case studies in TOL would be ideal to highlight this.

BIBLIOGRAPHY

[1] Jon Beck. "Distributive laws". In: *Seminar on Triples and Categorical Homology Theory*. Ed. by B. Eckmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1969, pp. 119–140. ISBN: 978-3-540-36091-9. DOI: 10.1007/BFb0083084.

[2] Charles Antony Richard Hoare. "An Axiomatic Basis for Computer Programming". In: *Commun. ACM* 12.10 (Oct. 1969), 576–580. ISSN: 0001-0782. DOI: 10.1145/363235.363259.

[3] Zohar Manna and Amir Pnueli. "Axiomatic Approach to Total Correctness of Programs". In: *Acta Inf.* 3.3 (1974), 243–263. ISSN: 0001-5903. DOI: 10.1007/BF00288637.

[4] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976, pp. I–XVII, 1–217. ISBN: 013215871X.

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

[6] Gordon Plotkin. "A Powerdomain Construction". In: *SIAM Journal on Computing* 5.3 (1976), pp. 452–487. DOI: 10.1137/0205035. eprint: https://doi.org/10.1137/0205035. URL: https://doi.org/10.1137/0205035.

[7] Vaughan R. Pratt. "Semantical Considerations on Floyd-Hoare Logic". In: *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. 1976, pp. 109–121. DOI: 10.1109/SFCS.1976.27.

[8] Stephen A. Cook. “Soundness and Completeness of an Axiom System for Program Verification”. In: *SIAM J. Comput.* 7.1 (Feb. 1978), 70–90. ISSN: 0097-5397. DOI: 10.1137/0207005. URL: https://doi.org/10.1137/0207005.

[9] C. A. R. Hoare. “Some Properties of Predicate Transformers”. In: *J. ACM* 25.3 (1978), 461–480. ISSN: 0004-5411. DOI: 10.1145/322077.322088.

[10] Krzysztof R. Apt. “Ten Years of Hoare’s Logic: A Survey—Part I”. In: *ACM Trans. Program. Lang. Syst.* 3.4 (Oct. 1981), 431–483. ISSN: 0164-0925. DOI: 10.1145/357146.357150. URL: https://doi.org/10.1145/357146.357150.

[11] Ralph-Johan Back. “A Continuous Semantics for Unbounded Nondeterminism.” In: *Theoretical Computer Science* 23 (Dec. 1983), 187–210. DOI: 10.1016/0304-3975(83)90055-5.

[12] K. R. Apt and G. D. Plotkin. “Countable nondeterminism and random assignment”. In: *J. ACM* 33.4 (1986), 724–767. ISSN: 0004-5411. DOI: 10.1145/6490.6494. URL: https://doi.org/10.1145/6490.6494.

[13] Edsger W. Dijkstra and Carel S. Schölten. “The strongest postcondition”. In: *Predicate Calculus and Program Semantics*. New York, NY: Springer New York, 1990, pp. 209–215. ISBN: 978-1-4612-3228-5. DOI: 10.1007/978-1-4612-3228-5_12. URL: https://doi.org/10.1007/978-1-4612-3228-5_12.

[14] Eugenio Moggi. “Notions of computation and monads”. In: *Information and Computation* 93.1 (1991), pp. 55–92. ISSN: 0890-5401. DOI: 10.1016/0890-5401(91)90052-4.

[15] Harald Søndergaard and Peter Sestoft. "Non-determinism in Functional Languages". In: *The Computer Journal* 35.5 (Oct. 1992), pp. 514–523. ISSN: 0010-4620. DOI: 10.1093/comjnl/35.5.514. eprint: https://academic.oup.com/comjnl/article-pdf/35/5/514/1125580/35-5-514.pdf. URL: https://doi.org/10.1093/comjnl/35.5.514.

[16] Samson Abramsky and Achim Jung. "Domain theory". In: *Handbook of Logic in Computer Science (Vol. 3): Semantic Structures*. USA: Oxford University Press, Inc., 1995, 1–168. ISBN: 019853762X.

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

[18] David Harel, Dexter Kozen, and Jerzy Tiuryn. "Dynamic logic". In: *SIGACT News* 32.1 (2001), pp. 66–69. DOI: 10.1145/568438.568456. URL: https://doi.org/10.1145/568438.568456.

[19] Patrick Cousot. "Constructive design of a hierarchy of semantics of a transition system by abstract interpretation". In: *Theoretical Computer Science* 277.1 (2002). Static Analysis, pp. 47–103. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(00)00313-3. URL: https://www.sciencedirect.com/science/article/pii/S0304397500003133.

[20] Christoph Lüth and Neil Ghani. "Composing Monads Using Coproducts". In: *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming*. ICFP '02. Pittsburgh, PA, USA: Association for Computing Machinery, 2002, 133–144. ISBN: 1581134878. DOI: 10.1145/581478.581492.

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

[22] Josh Berdine et al. "Automatic Termination Proofs for Programs with Shape-Shifting Heaps". In: *Computer Aided Verification*. Ed. by Thomas Ball and Robert B. Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 386–400. ISBN: 978-3-540-37411-4.

[23] Ashutosh Gupta et al. "Proving non-termination". In: *SIGPLAN Not.* 43.1 (Jan. 2008), 147–158. ISSN: 0362-1340. DOI: 10.1145/1328897.1328459. URL: https://doi.org/10.1145/1328897.1328459.

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

[25] Marc Brockschmidt, Byron Cook, and Carsten Fuhs. "Better Termination Proving through Cooperation". In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 413–429. ISBN: 978-3-642-39799-8.

[26] Patrick Cousot et al. "Automatic Inference of Necessary Preconditions". In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 128–148. ISBN: 978-3-642-35873-9.

[27] C.B. Jones. "Checking a Large Routine". In: *Alan Turing: His Work and Impact* (May 2013), pp. 455–463. DOI: 10.1016/B978-0-12-386980-7.50019-8.

[28] Daniel Larraz et al. “Proving Non-termination Using Max-SMT”. In: *Computer Aided Verification*. Ed. by Armin Biere and Roderick Bloem. Cham: Springer International Publishing, 2014.

[29] Thomas Icard. “Beyond Almost-Sure Termination”. In: *Proceedings of the 39th Annual Meeting of the Cognitive Science Society, CogSci 2017, London, UK, 16-29 July 2017*. Ed. by Glenn Gunzelmann et al. cognitivesciencesociety.org, 2017. URL: https://mindmodeling.org/cogsci2017/papers/0430/index.html.

[30] Annabelle McIver et al. “A New Proof Rule for Almost-Sure Termination”. In: *Proc. ACM Program. Lang.* 2.POPL (2018). DOI: 10.1145/3158121. URL: https://doi.org/10.1145/3158121.

[31] Benjamin Lucien Kaminski. “Advanced weakest precondition calculi for probabilistic programs”. Dissertation. Aachen: RWTH Aachen University, 2019, 1 Online–Ressource (xiv, 363 Seiten) : Illustrationen, Diagramme. DOI: 10.18154/RWTH-2019-01829.

[32] Peter W. O’Hearn. “Incorrectness Logic”. In: *Proc. ACM Program. Lang.* 4.POPL (Jan. 2020). DOI: 10.1145/3371078.

[33] 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*. Marseille, France: Springer-Verlag, 2021, 325–343. ISBN: 978-3-030-88700-1. DOI: 10.1007/978-3-030-88701-8_20.

[34] Flavio Ascari, Roberto Bruni, and Roberta Gori. “Limits and difficulties in the design of under-approximation abstract domains”. In: *Foundations of*

Software Science and Computation Structures. Cham: Springer International Publishing, 2022, pp. 21–39. ISBN: 978-3-030-99253-8. DOI: 10.1007/978-3-030-99253-8_2.

[35] Kevin Batz et al. *Weighted Programming*. 2022. arXiv: 2202.07577 [cs.PL].

[36] Quang Loc Le et al. "Finding Real Bugs in Big Programs with Incorrectness Logic". In: *Proc. ACM Program. Lang.* 6.OOPSLA1 (2022). DOI: 10.1145/3527325.

[37] Linpeng Zhang and Benjamin Lucien Kaminski. "Quantitative strongest post: a calculus for reasoning about the flow of quantitative information". In: *Proc. ACM Program. Lang.* 6.OOPSLA1 (2022). DOI: 10.1145/3527331. URL: https://doi.org/10.1145/3527331.

[38] Flavio Ascari et al. *Sufficient Incorrectness Logic: SIL and Separation SIL*. 2023. arXiv: 2310.18156 [cs.LO].

[39] Shenghua Feng et al. "Lower Bounds for Possibly Divergent Probabilistic Programs". In: *Proc. ACM Program. Lang.* 7.OOPSLA1 (Apr. 2023). DOI: 10.1145/3586051. URL: https://doi.org/10.1145/3586051.

[40] Petar Maksimović et al. "Exact Separation Logic: Towards Bridging the Gap Between Verification and Bug-Finding". In: *37th European Conference on Object-Oriented Programming (ECOOP 2023)*. Vol. 263. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 19:1–19:27. ISBN: 978-3-95977-281-5. DOI: 10.4230/LIPIcs.ECOOP.2023.19.

[41] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. "Outcome Logic: A Unifying Foundation of Correctness and Incorrectness Reasoning". In:

Proc. ACM Program. Lang. 7.OOPSLA1 (2023). DOI: 10.1145/3586045. URL: https://doi.org/10.1145/3586045.

[42] Patrick Cousot. "Calculational Design of [In]Correctness Transformational Program Logics by Abstract Interpretation". In: *Proc. ACM Program. Lang.* 8.POPL (2024). DOI: 10.1145/3632849. URL: https://doi.org/10.1145/3632849.

[43] Thibault Dardinier and Peter Müller. "Hyper Hoare Logic: (Dis-)Proving Program Hyperproperties". In: *Proc. ACM Program. Lang.* 8.PLDI (2024). DOI: 10.1145/3656437. URL: https://doi.org/10.1145/3656437.

[44] Andreas Lööw et al. "Compositional Symbolic Execution for Correctness and Incorrectness Reasoning". In: *38th European Conference on Object-Oriented Programming (ECOOP 2024)*. Ed. by Jonathan Aldrich and Guido Salvaneschi. Vol. 313. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 25:1–25:28. ISBN: 978-3-95977-341-6. DOI: 10.4230/LIPIcs.ECOOP.2024.25. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2024.25.

[45] Azalea Raad, Julien Vanegue, and Peter O'Hearn. "Non-termination Proving at Scale". In: *Proc. ACM Program. Lang.* 8.OOPSLA2 (Oct. 2024). DOI: 10.1145/3689720. URL: https://doi.org/10.1145/3689720.

[46] Noam Zilberstein. *A Relatively Complete Program Logic for Effectful Branching*. 2024. arXiv: 2401.04594 [cs.LO]. URL: https://arxiv.org/abs/2401.04594.

[47] Noam Zilberstein, Angelina Saliling, and Alexandra Silva. "Outcome Separation Logic: Local Reasoning for Correctness and Incorrectness with Computational Effects". In: *Proc. ACM Program. Lang.* 8.OOPSLA1 (2024). DOI: 10.1145/3649821.

[48] Rupak Majumdar and V.R. Sathiyanarayana. "Sound and Complete Proof Rules for Probabilistic Termination". In: *Proc. ACM Program. Lang.* 9.POPL (Jan. 2025). DOI: 10.1145/3704899. URL: https://doi.org/10.1145/3704899.

[49] Lena Verscht and Benjamin Lucien Kaminski. "A Taxonomy of Hoare-Like Logics: Towards a Holistic View using Predicate Transformers and Kleene Algebras with Top and Tests". In: *Proc. ACM Program. Lang.* 9.POPL (Jan. 2025). DOI: 10.1145/3704896. URL: https://doi.org/10.1145/3704896.

[50] Noam Zilberstein et al. "A Demonic Outcome Logic for Randomized Nondeterminism". In: *Proc. ACM Program. Lang.* 9.POPL (2025). DOI: 10.1145/3704855. URL: https://doi.org/10.1145/3704855.

APPENDIX A

SEMANTIC DOMAIN: LEMMAS AND PROPERTIES

Theorem A.0.1. *$(\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft}), \sqsubseteq)$ is a directed-complete partial order.*

Lemma A.0.1. *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. $f^{\dagger}(u \cdot m) = u \cdot f^{\dagger}(m)$ *and* $f^{\dagger}(m \cdot u) = f^{\dagger}(m) \cdot u$
3. *If* $\mathsf{supp}(m) \subseteq \Sigma$, $(f + g)^{\dagger}(m) = f^{\dagger}(m) + g^{\dagger}(m)$
4. *If* $\mathsf{supp}(m) \subseteq \Sigma$, $(f \cdot u)^{\dagger}(m) = f^{\dagger}(m) \cdot u$
5. $f^{\dagger}(\delta_{\circlearrowleft}) = \delta_{\circlearrowleft}$

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

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

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

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

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

□

Lemma A.0.2. *If $\sum_{i\in I} u_i$ is defined, then for any $(v_i)_{i\in I}$, $\sum_{i\in I} u_i \cdot v_i$ is defined.*

Proof. For the top element $\top$ of U, $v_i \leq \top$ for all $i \in I$. By definition of $\leq$, for each $i \in I$, there is a v_i' such that $v_i + v_i' = \top$. Since multiplication is total, we know that $left(\sum_{i\in I} u_i) \cdot v$ is defined. Note too that:

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

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

Lemma A.0.3. *For any $m \in \mathcal{W}_{\mathcal{A}}(X_{\cup})$ and $f : (X \to \mathcal{W}_{\mathcal{A}}(Y_{\cup}))$, we get that $f^{\dagger} : \mathcal{W}(X) \to \mathcal{W}(Y)$ is a total function.*

Proof. The definition of $(-)^{\dagger}$ is:

$$f^{\dagger}(m)(y) = \sum_{x\in \mathsf{supp}(m)\cap X} m(x) \cdot f(x)(y) + m(\cup) \cdot \delta_{\cup}$$

To show that this is well-defined, we need to show that the sum exists and that the resulting weighting function has a well-defined mass. First, we know that since $m \in \mathcal{W}_{\mathcal{A}}(X_{\cup})$, the mass $|m| = \sum_{\sigma\in \mathsf{supp}(m)} m(\sigma)$ is defined. By Lemma A.0.2, the sum in the definition of $(-)^{\dagger}$ must be well-defined.

It remains to show that $|f^{\dagger}(m)|$ is defined:

$$\begin{aligned}\left|f^{\dagger}(m)\right| &= \sum_{y\in \mathsf{supp}(f^{\dagger}(m))} \\ &= \left(\sum_{y\in \bigcup_{a\in \mathsf{supp}(m)\cap X} \mathsf{supp}(f(a))} \sum_{x\in \mathsf{supp}(m\cap X)} m(x) \cdot f(x)(y)\right) + m(\cup) \cdot \delta_{\cup}\end{aligned}$$

By commutativity and associativity:

$$= \left(\sum_{x \in \mathsf{supp}(m) \cap \Sigma} m(x) \cdot \sum_{y \in \mathsf{supp}(f(x))} f(x)(y) \right) + m(\circlearrowleft) \cdot \delta_x$$
$$= \left(\sum_{x \in \mathsf{supp}(m) \cap \Sigma} m(x) \cdot |f(x)| \right) + m(\circlearrowleft) \cdot \delta_x$$

Since $f(x) \in \mathcal{W}_{\mathcal{A}}(Y)$ for all $x \in X$, $|f(x)|$ must be defined. Since $|m|$ is defined, the entire sum must be as well, by Lemma A.0.2. □

Lemma A.0.4. *Let* $\langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ *be a conservative semiring,* i.e., *a partial semiring bounded by nonidempotent top element:* $u + \top = \top \implies u = 0$. *Then* $\top = 1$.

Proof. We cannot have $\top < 1$. Suppose $\top > 1$. Then, there exists $z \neq 0 \in U$ such that $1 + z = \top$. But then

$$\top \cdot \top = \top \cdot (1 + z) = \top + \top \cdot z \geq \top$$

Since addition is partial with bound $\top$, we need $\top + \top \cdot z \leq \top$. By definition, $\top + \top \cdot z \geq \top$. This forces $\top + \top \cdot z = \top$.

Since $\top$ is nonidempotent, $\top \cdot z = 0$. But $z \neq 0$ and $\top \neq 0$, so this violates the semiring property that 0 is the unique annihilator. □

APPENDIX B

TOTALITY OF LANGUAGE SEMANTICS

B.1 Fixed-Point Existence

Recall that the semantics of iteration commands $C^{\langle e_1, e_2 \rangle}$ is given by least fixed points: $\left[\!\left[C^{\langle e_1, e_2 \rangle}\right]\!\right](\sigma) \triangleq \mathsf{lfp}\left(\Phi_{C^{\langle e_1, e_2 \rangle}}\right)(\sigma)$, where

$$\Phi_{C^{\langle e_1, e_2 \rangle}} : \left(\Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})\right) \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft}),$$
$$\triangleq [\![e_1]\!](\sigma) \cdot f^{\dagger}([\![C]\!](\sigma)) + [\![e_2]\!](\sigma) \cdot \eta(\sigma)$$

is the functional encoding the recursive structure of the loop.

In this section, we prove the existence of this fixpoint using the usual Kleene fixpoint construction, with respect to the fusion order $\sqsubseteq$:

$$\mathsf{lfp}\left(\Phi_{C^{\langle e_1, e_2 \rangle}}\right) = \sup_{n \in \mathbb{N}} \Phi^n_{C^{\langle e_1, e_2 \rangle}}(\bot_f)$$

We denote $\bot_f$ as the bottomost element of $\left(\Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})\right)$, on which $\sqsubseteq$ is lifted pointwise:

$$\bot_f(\sigma)(\tau) \triangleq \top \cdot \delta_{\circlearrowleft} = \begin{cases} \mathbb{0} & \text{if } \tau \in \Sigma \\ \top & \text{if } \tau = \circlearrowleft \end{cases}$$

B.1.1 Continuity of Semantics

To apply Kleene's theorem, we need to show that $\Phi_{C^{\langle e_1, e_2 \rangle}}$ is Scott-continuous. We build up to this by proving the continuity of the semiring operations, and by extension, operations on weighting functions.

For all the results in this section, we assume $+$, $\cdot$, and $\sum$ are operations belonging to a naturally-ordered, fully (Scott-)continuous (partial) semiring $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ bounded with a top element $\top_{\mathcal{A}}$ (as described in Section 2.2).

Lemma B.1.1. *Let X, Y be dcpos. For monotonic $f : X^2 \to Y$ and directed $D \subseteq X$,*

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

Proof. We first show $\sup_{x \in D} f(x,x) \leq \sup_{x,y \in D} f(x,y)$, which follows immediately from the inclusion:

$$\{f(x,x) \mid x \in X\} \quad \subseteq \quad \{f(x,y) \mid x, y \in X\}$$

Now for the reverse inequality: since D is directed, for any $x, y \in D$, there exists an upper bound $z \in D$ such that $x \leq z$ and $y \leq z$. By monotonicity of f,

$$f(x,y) \leq f(z,z)$$

Since this holds for all x, y, we know that each $f(x,y)$ is bounded from above by some $f(z,z)$ and thus by $\sup_{x \in D} f(x,x)$. Therefore,

$$\sup_{x,y \in D} f(x,y) \leq \sup_{x \in D} f(x,x)$$

□

Lemma B.1.2. *Let X be a dcpo. For any family of Scott-continuous functions $(f_i : X \to U)_{i \in I}$ 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. We proceed by transfinite induction on the cardinality of I.

- **Base Case:** $I = \emptyset$, so the sum is vacuous:

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

- **Successor Case:** Suppose the claim holds for all sets of size n, where $n \in \mathbb{N}$. Let $|I| = n + 1$. Then we can partition $I = I' \cup \{j\}$ for arbitrary $j \in I$ such that $|I'| = n$. It holds that

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

Binary addition + and the functions f_i are monotonic by definition of Scott-continuity. By the induction hypothesis, $\sum_{i \in I'} f_i$ is monotonic as well. We can thus apply Lemma B.1.1 to take the supremum separately over the two summands:

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

By Scott-continuity of +:

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

Finally, by the IH and Scott-continuity of f_j:

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

- **Limit Case:** Suppose the claim holds for all finite index sets. Let $|I| \geq \aleph_0$

be a limit cardinal. By definition of the sum operator:

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

□

We now try to prove the same result with respect to the dual order $\geq$. As it turns out, the furthest we can get without additional structure on the semiring is showing lower-continuity for finite sums. Proving this for infinite sums requires a minimax equality which, generally, does not hold (see Chapter 6).

Lemma B.1.3. *Let X be a dcpo. For any* finite *family of lower Scott-continuous functions $(f_i : X \to U)_{i\in I}$ and $D \subseteq X$ directed (in the dual order):*

$$\inf_{x\in D}\sum_{i\in I} f_i(x) = \sum_{i\in I} f_i(\inf D)$$

Proof. See the proof of Lemma B.1.2 up to the successor case. □

Finally, we lift these two results from the semiring to weighting functions.

Lemma B.1.4. *Let X be a dcpo. For any* finite *family of Scott-continuous functions $(f_i : X \to \mathcal{W}_{\mathcal{A}}(\Sigma_\cup))$ and directed set $D \subseteq X$:*

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

Proof. Follows from Lemmas B.1.2 and B.1.3, since $\subseteq$ on $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ is defined pointwise with the order flipped for nonterminating weight. □

Theorem B.1.1 (Scott-Continuity of $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$)**.** *Scalar multiplication $\cdot$ and pointwise addition $+$ on weighting functions are Scott-continuous with respect to fusion order $\sqsubseteq$. That is, for any directed subset $D \subseteq \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$,*

$$\sup_{m \in D} u \cdot m = u \cdot \sup D \qquad \sup_{m \in D}(m \cdot u) = \sup D \cdot u \qquad \sup_{m_1 \in D}(m_1 + m_2) = \sup D + m_2$$

Proof. This follows by pointwise analysis of weighting functions and the continuity of $\mathcal{A}$. We show the proof for $+$ to illustrate how the fusion order treats terminal and nonterminal outcomes dually. First, consider $\sigma \in \Sigma$:

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

Dually, for $\circlearrowleft$:

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

The proofs for $\cdot$ follow similarly. □

Lemma B.1.5. *The restricted Kleisli extension $(-)^\dagger : \left(\Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})\right) \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ is Scott-continuous.*

Proof. Let $D \subseteq \left(\Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})\right)$ be a directed set. For any weighting function $m \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, we want to show that

$$\sup_{f \in D} f^\dagger(m) = (\sup D)^\dagger(m)$$

This requires exploiting the structure of our restricted subdomain $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$. We break into cases, depending on whether our semantics is *conservative* or *indicative* (Section 2.2).

Case 1: Indicative Semantics. We have that m falls in

$$\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid \mathsf{supp}(m) \text{ is finite} \vee m(\circlearrowleft) = \top\}$$

If $\mathsf{supp}(m)$ is finite, then by Lemma B.1.2 and the continuity of $+, \cdot$:

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

Otherwise, if $m(\circlearrowleft) = \top$, we can leverage the absorptivity of $\top$:

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

Case 2: Conservative Semantics. Then, m is in

$$\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \triangleq \{m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft}) \mid m(\circlearrowleft) = \sup_{m' \in \mathsf{Equiv}(m)} m'(\circlearrowleft)\}$$

where $\mathsf{Equiv}(m) = \{m' \mid \forall \tau \in \Sigma.\, m'(\tau) = m(\tau)\}$. For all $\tau \in \Sigma$:

$$\left(\sup_{f \in D} f^\dagger(m)\right)(\tau) = \sup_{f \in D}\left(\sum_{\sigma \in \mathsf{supp}(m) \cap \Sigma} m(\sigma) \cdot f(\sigma)(\tau) + m(\circlearrowleft) \cdot \delta_x(\tau)\right)$$

Since $\delta_x(\tau) = 0$:

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

By Lemma B.1.2 and continuity of $\cdot$:

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

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

$$= (\sup D)^\dagger(m)(\tau)$$

By completeness of $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, we know that $\sup^\dagger_{f \in D}(m)$ and $(\sup D)^\dagger(m)$ are members of $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$. The above equality implies that $\sup^\dagger_{f \in D}(m) \in \mathsf{Equiv}\left((\sup D)^\dagger(m)\right)$ and vice versa. It follows that:

$$\left(\sup^\dagger_{f \in D}(m)\right)(\circlearrowleft) \leq \left((\sup D)^\dagger(m)\right)(\circlearrowleft)$$

as does the reverse inequality:

$$\left(\sup^\dagger_{f \in D}(m)\right)(\circlearrowleft) \geq \left((\sup D)^\dagger(m)\right)(\circlearrowleft)$$

Therefore, $\sup^\dagger_{f \in D}(m) = (\sup D)^\dagger(m)$. □

Lemma B.1.6. *Suppose that the loop functional $\Phi_{C^{\langle e,e' \rangle}}$ maps into the restricted domain:*

$$\Phi_{C^{\langle e_1,e_2 \rangle}} : (\Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})) \to \Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$$

Then $\Phi_{C^{\langle e_1,e_2 \rangle}}$ is Scott-continuous.

Proof. For any directed set $D \subseteq (\Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup}))$ and $\sigma \in \Sigma$,

$$\sup_{f \in D} \Phi_{C^{\langle e_1, e_2 \rangle}}(f)(\sigma) = \sup_{f \in D} \Big([\![e_1]\!](\sigma) \cdot f^{\dagger}([\![C]\!](\sigma)) + [\![e_2]\!] \cdot \eta(\sigma) \Big)$$

By continuity of + and $\cdot$, we can move the supremum up to the $(-)^{\dagger}$:

$$= [\![e_1]\!](\sigma) \cdot \left(\sup_{f \in D} f^{\dagger}([\![C]\!](\sigma)) \right) + [\![e_2]\!] \cdot \eta(\sigma)$$

Then, by Lemma B.1.5,

$$= [\![e_1]\!](\sigma) \cdot (\sup D)^{\dagger}([\![C]\!](\sigma)) + [\![e_2]\!] \cdot \eta(\sigma)$$

□

Given Lemma B.1.6, we can apply Kleene's fixed point theorem, concluding that the least fixed point of $\Phi_{C^{\langle e, e' \rangle}}$ exists and is equal to:

$$\mathsf{lfp}\ (\Phi_{C^{\langle e, e' \rangle}}) = \sup_{n \in \mathbb{N}} \Phi^{n}_{C^{\langle e, e' \rangle}}(\bot_f)$$

Thus the semantics of iteration is well-defined, assuming $\Phi_{C^{\langle e, e' \rangle}}$ is total.

B.2 Well-Defined Programs

In the previous section, we established the existence of a least fixpoint for loops, given that the loop functional $\Phi_{C^{\langle e, e' \rangle}}$ be (1) total and (2) restricted to $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$. The former may not hold if a partial semiring is used to interpret the language semantics. For the latter, we need to show that our semantics is closed in $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$.

We specify syntactic restraints on branching $C_1 + C_2$, weighting commands **`assume`** e, and iteration $C^{\langle e, e' \rangle}$ that ensure both (1) and (2).

B.2.1 Syntactic Sugar

Compatibility

We reuse the notion of *compatible* expressions introduced for past iterations of Outcome Logic. Many of these results are repeated from [47].

Definition B.2.1 (Compatibility). Expressions e_1 and e_2 are *compatible* in semiring $\mathcal{A} = \langle U, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ if $[\![e_1]\!](\sigma) + [\![e_2]\!](\sigma)$ is defined for all $\sigma \in \Sigma$.

Lemma B.2.1. *If e_1 and e_2 are compatible, then $[\![e_1]\!](\sigma) \cdot m_1 + [\![e_2]\!](\sigma) \cdot m_2$ is defined for any $m_1, m_2 \in \mathcal{W}_{\mathcal{A}}(\Sigma_\cup)$.*

Proof. Since e_1 and e_2 are compatible, then $[\![e_1]\!](\sigma) + [\![e_2]\!](\sigma)$ is defined. By Lemma A.0.2, we have that $[\![e_1]\!](\sigma) \cdot |m_1| + [\![e_2]\!](\sigma) \cdot |m_2|$ is defined as well. Then:

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

Therefore, $[\![e_1]\!](\sigma) + [\![e_2]\!](\sigma)$ is well-defined. □

Lemma B.2.2. *If e_1 and e_2 are compatible and $[\![C_1]\!]$ and $[\![C_2]\!]$ are total functions, then:*

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

is a total function.

Proof.

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

By Section B.2.1, this sum is defined and thus the semantics is valid. □

Normalization

For conservative semirings, we've provided restrictions on branching to ensure totality of the semantics. We also want restrictions that ensure the semantics is closed in our restricted domain $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$, which guarantees that the semantics of loops is defined (Section B.1).

We define the notion of normalization as a strengthening of compatibility.

Definition B.2.2. Expressions e_1 and e_2 are *normalized* in a conservative semiring (Section 2.2) if $[\![e_1]\!](\sigma) + [\![e_2]\!](\sigma) = \top$ for all $\sigma \in \Sigma$.

Lemma B.2.3. *Suppose u_1, u_2 are normalized. Then, for $m_1, m_2 \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$,*

$$(u_1 \cdot m_1 + u_2 \cdot m_2) \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$$

Proof. Let $m = u_1 \cdot m_1 + u_2 \cdot m_2$.

$$\begin{aligned}
|m| &= |u_1 \cdot m_1 + u_2 \cdot m_2| \\
&= u_1 \cdot |m_1| + u_2 \cdot |m_2| \\
&= u_1 \cdot \sup_{q_1 \in \mathsf{Equiv}(m_1)} |q_1| + u_2 \cdot \sup_{q_2 \in \mathsf{Equiv}(m_2)} |q_2| \\
&= u_1 \cdot \top + u_2 \cdot \top \\
&= (u_1 + u_2) \cdot \top \\
&= \top \cdot \top = \top \\
&= \sup_{q \in \mathsf{Equiv}(m)} |q|
\end{aligned}$$

If we expand both ends of this equality, we get:

$$\sum_{\sigma \in \mathsf{supp}(m)} m(\sigma) = \sup_{q \in \mathsf{Equiv}(m)} \sum_{\sigma \in \mathsf{supp}(q)} q(\sigma)$$

Since $m(\sigma) = q(\sigma)$ for all $\sigma \in \Sigma$, the supremum is attained when $m(\circlearrowleft) = \sup_{q \in \mathsf{Equiv}(m)} q(\circlearrowleft)$. □

The above result can be extended to general sums.

Definition B.2.3. We say that program exhibits *normalized choice* if all branching $C_1 + C_2$ and weighting **assume** e is of the form:

$$\textbf{assume}\ e_1 \fatsemi C_1 + \textbf{assume}\ e_2 \fatsemi C_2$$

where e_1 and e_2 are normalized.

B.2.2 Closure

For indicative semantics, $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ does not form a dcpo with respect to general directed sets. Instead, because we limit to bounded nondeterminism, we need

to generalize the Egli-Milner order on powerdomains [12] to restrict directed sets.

Definition B.2.4. In the indicative setting, a directed set D is an *EG-directed* set if for all $m_1, m_2 \in D$, $m_1 = m_2$ or $m_1 \sqsubseteq m_2 \implies m_1(\circlearrowleft) = \top$.

Lemma B.2.4. *In the indicative setting, $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ equipped with the fusion-order is a poset that preserves the suprema of* EG-directed *sets.*

Proof. We have that $\sqsubseteq$ is a partial order. Let $D \subseteq \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ be an *EG-directed* set. Let

$$(\sup D)(\sigma) = \begin{cases} \sup_{m \in D} m(\sigma) & \text{if } \sigma \in \Sigma \\ \inf_{m \in D} m(\circlearrowleft) & \text{if } \sigma = \circlearrowleft \end{cases}$$

Since $\mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ is Scott-continuous, this exists. It remains to show that $\sup D \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$. We argue that there are two possibilities:

(1) For all $m \in D$, $\mathsf{supp}(m)$ is finite; or

(2) For all $m \in D$, $m(\circlearrowleft) = \top$

Suppose otherwise. Then there must exist $m_1 \in D$ such that $\mathsf{supp}(m_1)$ is infinite and $m_2 \in D$ such that $m_2(\circlearrowleft) \neq \top$. Since D is directed, there must exist $m \in D$ greater than both. That is, $m(\sigma) \geq m_1(\sigma)$ for all $\sigma \in \Sigma$, so $\mathsf{supp}(m)$ is infinite. And $m(\circlearrowleft) \leq m_2(\circlearrowleft) < \top$. But $m \notin \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$.

Thus, we have two cases. In the case of (2), it is trivial that $(\sup D)(\circlearrowleft) = \inf_{m \in D} m(\circlearrowleft) = \top$. So, $\sup D \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$.

In this case of (1), either $\mathsf{supp}(\sup D)$ is finite or infinite. If finite, then we are done. If infinite, then for any m, $\sup D \neq m$ for any m. By the definition of

EG-directed sets, since $m \sqsubseteq \sup D$, $m(\circlearrowleft) = \top$. Because this holds for all $m \in D$, $(\sup D)(\circlearrowleft) = \top$ as well. □

We now move onto the closure results for our semantics. We start with indicative semantics, in which interpretations of programs must be limited to bounded nondeterminism.

Lemma B.2.5. *Suppose the semiring $\mathcal{A}$ is indicative. If for all $a \in \mathsf{Act}$, $[\![a]\!] : \Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$, then $[\![C]\!] : \Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$ for all programs C.*

Proof. By induction on the structure of C. That our programs preserve bounded nondeterminism, supposing atomic actions do, should be largely straightforward. We only include sketches of the cases for $C_1 \fatsemi C_2$ and $C^{\langle e,e' \rangle}$:

- $C_1 \fatsemi C_2$.

$$[\![C_1 \fatsemi C_2]\!](\sigma) = [\![C_2]\!]^{\dagger}([\![C_1]\!](\sigma))$$
$$= \sum_{\tau \in \mathsf{supp}([\![C_1]\!](\sigma)) \cap \Sigma} [\![C_1]\!](\sigma)(\tau) \cdot [\![C_2]\!](\tau) + [\![C_1]\!](\sigma)(\circlearrowleft) \cdot \delta_{\circlearrowleft}$$

  By the IH, for every σ, τ, $[\![C_1]\!](\sigma)$ and $[\![C_2]\!](\tau)$ both exhibit bounded nondeterminism. If all have finite support, then the above weighting function must have finite support as well.

  Recall that $\top$ is multiplicatively and additively absorbing. If $[\![C_1]\!](\sigma)$ diverges with weight $\top$ for some σ, then $[\![C_1]\!](\sigma)(\circlearrowleft) \cdot \delta_{\circlearrowleft}(\circlearrowleft) = \top$, and so on for the entire expression. If $[\![C_2]\!](\tau)(\circlearrowleft) = \top$, a similar argument applies.

▹ $C^{\langle e,e'\rangle}$. By Kleene's theorem:

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

$\Phi_{C^{\langle e,e'\rangle}}(f) = [\![e]\!](\sigma) \cdot f^\dagger([\![C]\!](\sigma)) + [\![e']\!](\sigma) \cdot \eta(\sigma)$. $\mathcal{W}^\square_{\mathcal{A}}(\Sigma_\cup)$ is closed under these operations (as has been shown or by straightforward computation).

By mathematical induction, we can show that for all $n \in \mathbb{N}$:

$$\Phi^n_{C^{\langle e,e'\rangle}}(\bot_f)(\sigma) \in \mathcal{W}^\square_{\mathcal{A}}(\Sigma_\cup)$$

It is also easy to verify that $(\Phi_C^{\langle e,e'\rangle})^n(\bot_f)$ form an EG-directed set. We leave these to the reader.

Since $\mathcal{W}^\square_{\mathcal{A}}(\Sigma_\cup)$ preserves suprema of EG-directed sets and $\Phi_{C^{\langle e,e'\rangle}}$ is Scott-continuous, the supremum $\sup_{n\in\mathbb{N}} \Phi^n_{C^{\langle e,e'\rangle}}(\bot_f)(\sigma)$ must exist in $\mathcal{W}^\square_{\mathcal{A}}(\Sigma_\cup)$.

□

Now, we prove the analogue for a conservative semantics. This assumes the additional syntactic constraints that ensure normalization in programs.

Lemma B.2.6. *In the conservative setting, $\mathcal{W}^\square_{\mathcal{A}}(\Sigma_\cup)$ equipped with the fusion order $\sqsubseteq$ forms a dcpo.*

Proof. We have that $\sqsubseteq$ is a partial order. Take $D \subseteq \mathcal{W}^\square_{\mathcal{A}}(\Sigma_\cup)$. We argue that D must form a chain, *i.e.,* for any $m_1, m_2 \in D$, either $m_1 \sqsubseteq m_2$ or vice versa.

Suppose by way of contradiction that there exists $m_1, m_2 \in D$ incomparable. Consider any outcomes $\sigma, \tau \in \Sigma_\cup$.

- If $\sigma, \tau \in \Sigma$, then (without loss of generality) $m_1(\sigma) > m_2(\sigma)$ and $m_1(\tau) < m_2(\tau)$. Since D is directed, there must be an upper bound m such that $m(\sigma) \geq m_1(\sigma)$ and $m(\sigma) \geq m_2(\tau)$. As we have seen in the proof of Lemma B.2.3, since $m_1, m_2 \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$, $|m_1| = |m_2| = \top$. But this means $|m| > |m_1| = \top$. This is impossible by the nonidempotency of $+$.
- WLOG, if $\tau = \cup$, then $m_1(\sigma) \geq m_2(\sigma)$ and $m_1(\cup) > m_2(\cup)$. A similar argument follows: If $|m_1| = \top$ then it $|m_2| \neq \top$. But $|m_2| = \top$ since $m_2 \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$.

Since D is in fact a chain, then $|\sup D| = \sup_{m \in D} |m \in D| = \top$. Therefore, $(\sup D) \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$. □

Lemma B.2.7. *Suppose the semiring $\mathcal{A}$ is conservative. If for all $a \in \mathsf{Act}$, $[\![a]\!] : \Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$, and C exhibits normalized choice, then $[\![C]\!] : \Sigma \to \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$ for all programs C.*

Proof. By induction on the structure of C.

- **`skip`**. We know that $[\![\texttt{skip}]\!](\sigma) = \eta(\sigma)$. Suppose $m \in \mathsf{Equiv}([\![\texttt{skip}]\!](\sigma))$. Since $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$, $|m| = m(\sigma) + m(\cup)$ must be defined.

  By Lemma A.0.4, $1 = \top$. We have

  $$|m| = m(\sigma) + m(\cup) = 1 + m(\cup) \leq 1$$

  Since 1 is nonidempotent, $m(\cup) = 0$. Therefore,

  $$[\![\texttt{skip}]\!](\sigma)(\cup) = 0 = \sup_{m \in \mathsf{Equiv}([\![\texttt{skip}]\!](\sigma))} m(\cup)$$

▹ **assume** $u_1 \fatsemi C_1 + \textbf{assume}\ u_2 \fatsemi C_2$. By the IH, we have that $[\![C_1]\!], [\![C_2]\!]$ map into $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$. By assumption $u_1 + u_2$ are normalized, so by Lemma B.2.3,

$$[\![\textbf{assume}\ u_1 \fatsemi C_1 + \textbf{assume}\ u_2 \fatsemi C_2]\!] = u_1 \cdot [\![C_1]\!] + u_2 \cdot [\![C_2]\!] \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$$

▹ $C_1 \fatsemi C_2$.

$$\begin{aligned}[\![C_1 \fatsemi C_2]\!](\sigma) &= [\![C_2]\!]^{\dagger}([\![C_1]\!](\sigma)) \\ &= \sum_{\tau \in \mathsf{supp}([\![C_1]\!](\sigma)) \cap \Sigma} [\![C_1]\!](\sigma)(\tau) \cdot [\![C_2]\!](\tau) + [\![C_1]\!](\sigma)(\circlearrowleft) + \delta_{\cup}\end{aligned}$$

By the IH, we have that $[\![C_2]\!](\tau) \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$ for all τ. It is also straightforward that $\delta_{\cup} \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$. Note that since $[\![C_1]\!](\sigma) \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$ (again, by IH), the mass $|[\![C_1]\!](\sigma)| = \sum_{\tau \in \Sigma_{\cup}} [\![C_1]\!](\sigma)(\tau)$ must be normalized. Thus, we can apply *Lemma B.2.3*, concluding that $[\![C_1 \fatsemi C_2]\!](\sigma) \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$.

$C^{\langle e, e' \rangle}$. By Kleene's theorem:

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

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

In the previous cases, we showed that $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$ is closed under $(-)^{\dagger}$, and that $\eta(\sigma) \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$. Since e and e' are assumed to be normalized, $\Phi_{C^{\langle e, e' \rangle}}(f)$ maps to $\mathcal{W}^{\Sigma_{\cup}}_{\mathcal{A}}$ by Lemma B.2.3.

Then by mathematical induction, we can show that for all $n \in \mathbb{N}$:

$$\Phi^n_{C^{\langle e, e' \rangle}}(\bot_f)(\sigma) \in \mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$$

Since $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$ is a dcpo and $\Phi_{C^{\langle e, e' \rangle}}$ is Scott-continuous, the supremum $\sup_{n \in \mathbb{N}} \Phi^n_{C^{\langle e, e' \rangle}}$ must exist in $\mathcal{W}^{\square}_{\mathcal{A}}(\Sigma_{\cup})$. □

Lemma B.2.8. *If e and e' are normalized and $[\![C]\!]$ is a total function (with C exhibiting normalized choice), then $\left[\!\left[C^{\langle e,e'\rangle}\right]\!\right]$ is a total function.*

Proof. Let $\Phi_{C^{\langle e,e'\rangle}}(f)(\sigma) = [\![e]\!](\sigma) \cdot f^\dagger([\![C]\!](\sigma)) + [\![e']\!](\sigma) \cdot \eta(\sigma)$. By Lemma B.2.7, $[\![C]\!](\sigma) \in \mathcal{W}'^{\square}_{\mathcal{A}}(\Sigma_\cup)$, and assuming f is restricted to $\mathcal{W}'^{\square}_{\mathcal{A}}(\Sigma_\cup)$, so is $f^\dagger(\sigma)$. We know also from the proof of Lemma B.2.7 that $\eta(\sigma) \in \mathcal{W}'^{\square}_{\mathcal{A}}(\Sigma_\cup)$. Since e and e' are normalized, by Lemma B.2.3, $\Phi_{C^{\langle e,e'\rangle}}(f)(\sigma) \in \mathcal{W}'^{\square}_{\mathcal{A}}(\Sigma_\cup)$.

By Lemma B.1.6, we know that $\left[\!\left[C^{\langle e,e'\rangle}\right]\!\right]$ is total. □

APPENDIX C

SOUNDNESS AND COMPLETENESS OF TOL

Here, we provide omitted proofs for establishing soundness and relative completeness of TOL . We start by introducing some helpful preliminary definitions.

Definition C.0.1. For any test $b \in 2^{\Sigma}$ and $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$, we define the *projection* of m onto b to be the following weighting function:

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

In our proofs, it will be useful to decompose a weighting function $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$ into two—one part which solely describes weights on program states Σ, and another which solely describes the degree of nontermination. The former is represented by the projection $\mathsf{true}\,?\,m$ and the latter by $\mathsf{div}\,?\,m$, which we define below.

Definition C.0.2. For $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$, the nonterminating component $(\mathsf{div}\,?\,m) \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$ of m is:

$$(\mathsf{div}\,?\,m)(\sigma) \triangleq m(\cup) \cdot \delta_{\cup}(\sigma) = \begin{cases} \mathbb{0} & \text{if } \sigma \in \Sigma \\ m(\cup) & \text{if } \sigma = \cup \end{cases}$$

Note that $(\mathsf{div}\,?\,m) \vDash \Uparrow^{(m(\cup))}$.

Lemma C.0.1. *For any $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$ and $b \in$ Test:*

$$m = \mathsf{true}\,?\,m + \mathsf{div}\,?\,m = b\,?\,m + \neg b\,?\,m + \mathsf{div}\,?\,m$$

Proof. Immediate from the definitions. □

It follows straightforwardly that if $\mathsf{supp}(m) \subseteq \Sigma$, then $m = \mathsf{true}\,?\,(m)$. Likewise, if $\mathsf{supp}(m) \subseteq \{\circlearrowleft\}$, $m = \mathsf{div}?m$. Projection commutes with the following operations:

Lemma C.0.2. *For any $b \in \mathit{Test}$ and $m, m_i \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\circlearrowleft})$,*

(i) $\sum_{i\in I}(b\,?\,m_i) = b\,?\,\sum_{i\in I} m_i$

(ii) $u \cdot (b\,?\,m) = b\,?\,u \cdot m$ *and* $(b\,?\,m) \cdot u = b\,?\,m \cdot u$

Proof. We show that $(u \cdot \sum_{i\in I}(b\,?\,m_i) \cdot v)\,(\sigma) = (b\,?\,u \cdot \sum_{i\in I} m_i \cdot v)\,(\sigma)$. From the definition of projection, it is straightforward to see that these are equal to

$$\begin{cases} u \cdot \sum_{i\in I} m_i(\sigma) \cdot v & \text{if } [\![b]\!]_{\mathsf{Test}}(\sigma) = \mathbb{1} \\ \mathbb{0} & \text{if } [\![b]\!]_{\mathsf{Test}}(\sigma) = \mathbb{0} \text{ or } \sigma = \circlearrowleft \end{cases}$$

□

C.1 Soundness

We proceed by proving results on the semantics of iteration. In Section B.1.1, we expressed the denotation of $C^{\langle e,e'\rangle}$ as the Kleene fixed point:

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

Our first Lemma reformulates the Kleene iterates $\Phi^n_{C^{\langle e,e'\rangle}}$ in terms sequences of unrolled commands.

Lemma C.1.1. *For all $n \in \mathbb{N}$, $\sigma \in \Sigma$, and $\tau \in \Sigma_{\circlearrowleft}$,*

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

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

Proof. We proceed by induction on n.

- *Base Case*: $n = 0$. We have that:

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

  Suppose $\tau \in \Sigma$. Since $\bot_f = \lambda\rho.\ \top \cdot \delta_{\circlearrowleft}$, it follows that $\bot_f^\dagger\Big([\![\textbf{assume } e \fatsemi C)]\!](\sigma)\Big)(\tau) = \mathbb{0}$. Then,

$$\Phi_{C^{\langle e,e'\rangle}}(\bot_f)(\sigma)(\tau) = [\![\textbf{assume } e']\!](\sigma)(\tau)$$

  On the other hand, if $\tau = \circlearrowleft$, then $[\![\textbf{assume } e']\!](\sigma)(\tau) = \mathbb{0}$. In this case:

$$\Phi_{C^{\langle e,e'\rangle}}(\bot_f)(\sigma)(\circlearrowleft) = \bot_f^\dagger\Big([\![\textbf{assume } e \fatsemi C]\!](\sigma)\Big)(\circlearrowleft)$$

- *Inductive step*: Suppose the claim holds for n. First, for any $\sigma \in \Sigma$, $\tau \in \Sigma_{\circlearrowleft}$:

$$\Phi^{n+2}_{\langle C,e,e'\rangle}(\bot_f)(\sigma)(\tau) = [\![e]\!](\sigma) \cdot \Big(\Phi^{n+1}_{\langle C,e,e'\rangle}(\bot_f)\Big)^\dagger\Big([\![C]\!](\sigma)\Big)(\tau) + [\![e']\!](\sigma) + \eta(\sigma)(\tau)$$

  Suppose $\tau \in \Sigma$. We use the IH to expand $\Phi^{n+1}_{C^{\langle e,e'\rangle}}(\bot_f)$:

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

Otherwise, $\tau = \circlearrowleft$. We apply the other case of the IH:

$$\begin{aligned}\Phi^{n+2}_{\langle C,e,e'\rangle}(\bot_f)(\sigma)(\tau) &= [\![e]\!](\sigma) \cdot \left(\bot_f^\dagger\left([\!\!\left[(\textbf{assume}\ e \fatsemi C)^{n+1}\right]\!\!]\right)\right)^\dagger\left([\![C]\!](\sigma)\right)(\circlearrowleft) \\ &\quad + [\![e']\!](\sigma) \cdot \eta(\sigma)(\circlearrowleft) \\ &= \bot_f^\dagger([\!\!\left[(\textbf{assume}\ e \fatsemi C)^{n+2}\right]\!\!]) + [\![e']\!] \cdot \mathbb{0} \\ &= \bot_f^\dagger([\!\!\left[(\textbf{assume}\ e \fatsemi C)^{n+2}\right]\!\!])\end{aligned}$$

□

Remark C.1.1. The two different cases in the above result are due to differences in how the effects of branching and nontermination are composed. In particular, if we want the weight on a terminal program state in Σ, then we aggregate the weight collected from terminating traces in the first n iterations. We push the weight of traces that have not yet terminated at iteration n onto $\circlearrowleft$.

The next Lemma applies this unrolling to $[\![C^{\langle e,e'\rangle}]\!]$, expressing the denotation in terms of infinite sequences of commands.

Lemma C.1.2 (Unrolling). *For all $\sigma \in \Sigma$ and $\tau \in \Sigma_\circlearrowleft$,*

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

Proof. By Kleene's fixed point theorem (Section B.1.1),

$$[\![C^{\langle e,e'\rangle}]\!](\sigma)(\tau) = \left(\sup_{n\in N} \Phi^n_{\langle C,e,e'\rangle}(\bot_f)(\sigma)\right)(\tau)$$

Case 1: $\tau \in \Sigma$. Since $\Phi^0_{\langle C,e,e'\rangle}(\bot_f) = \bot_f$, *i.e.*, the bottom element of $\left(\Sigma \to \mathcal{W}_{\mathcal{A}}(\Sigma_\circlearrowleft)\right)$, we can rewrite this supremum as:

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

By Lemma C.1.1:

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

By the definition of infinite sums:

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

Case 2: $\tau = \circlearrowleft$. By Lemma C.1.1:

$$\left(\sup_{n\in\mathbb{N}} \Phi^n_{\langle C,e,e'\rangle}(\sigma)\right)(\circlearrowleft) = \left(\sup_{n\in\mathbb{N}} \bot^{\dagger}_f([\![(\textbf{assume}\ e \fatsemi C)^n)]\!]\right)(\circlearrowleft)$$

Recall that $\sqsubseteq$ is defined pointwise, except the order is reversed for $\circlearrowleft$ (see Definition 2.2.8). Therefore, the supremum of weighting functions is equivalent to taking the infimum of the weights on $\circlearrowleft$:

$$= \inf_{n\in\mathbb{N}} \left(\bot^{\dagger}_f([\![(\textbf{assume}\ e \fatsemi C)^n)]\!])(\circlearrowleft)\right)$$

□

Equivalently, we can express this unrolling in terms of projections:

Corollary C.1.1.

$$[\![C^{\langle e,e'\rangle}]\!](\sigma) = \mathsf{true}\ ?\ \left(\sum_{n\in N} [\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!](\sigma)\right) + \mathsf{div}\ ?\ \left(\inf_{n\in\mathbb{N}} \bot^{\dagger}_f([\![(\textbf{assume}\ e \fatsemi C)^n]\!])(\sigma)\right)$$

Proof. By application of Lemma C.0.1. □

Theorem C.1.1 (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. 3.2, fig. 3.1, and fig. 3.3. 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, $m = u \cdot \delta_{\circlearrowleft}$ by definition. We have

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

  So, $[\![C]\!]^\dagger(m) \vDash \Uparrow^{(u)}$.
- ▹ SCALE. By the IH, 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 (ii) of Lemma A.0.1, 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 IH, $\vDash \langle\varphi_1\rangle\ C\ \langle\psi_1\rangle$ 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 IH 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 IH 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 IH, we have that $\vDash \langle \phi(t) \rangle\ C\ \langle \phi'(t) \rangle$ for all $t \in T$. 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 $t \in T$, $m_t \vDash \phi(t)$. By Lemma A.0.1 (i), we can move $\sum$ outside the extension like so:

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

  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 IH, 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 IH, $[\![C]\!]^\dagger(m) \vDash \phi'(t)$, 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 IH. 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 $\vDash \langle \varphi \rangle\ \mathtt{skip}\ \langle \varphi \rangle$. Suppose that $m \vDash \varphi$. We have that $[\![\mathtt{skip}]\!]^\dagger(m) = m$, so $[\![\mathtt{skip}]\!]^\dagger(m) \vDash \varphi$ 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 $\vDash \langle \varphi \rangle\ C_1 \fatsemi C_2\ \langle \psi \rangle$. Suppose $m \vDash \varphi$, so by the IH, $[\![C_1]\!]^\dagger(m) \vDash \vartheta$ and $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) \vDash \psi$. Since $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) = ([\![C_1]\!]^\dagger \circ [\![C_2]\!]^\dagger)(m) = [\![C_1 \fatsemi C_2]\!]^\dagger(m)$, we are done.

- PLUS. Given $\varphi \vDash \mathsf{true}$, $\Omega \vdash \langle \varphi \rangle\ C_1\ \langle \psi_1 \rangle$, 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 $\varphi \vDash \mathsf{true}$, it must

be that $\mathsf{supp}(m) \subseteq \Sigma$. By Lemma A.0.1 (iii), we know that the $(-)^\dagger$ operator commutes with $+$:

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

By the IH, $[\![C_1]\!]^\dagger(m) \vDash \psi_1$ 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 $\vDash \langle \varphi \rangle$ **assume** e $\langle \varphi \odot u \rangle$. Take $m \vDash \varphi$. Then,

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

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

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

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

It is then straightforward to show, by mathematical induction on n:

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

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

for all n. It follows that:

(i) Since for all n, $[\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m) \vDash \psi_n \oplus \zeta_n$, there exist $p_n, q_n \in \mathcal{W}_{\mathcal{A}}(\Sigma_\cup)$ such that

$$[\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m) = p_n + q_n$$

for $p_n \vDash \psi_n$ and $q_n \vDash \zeta_n$.

The ITER rule has side conditions $\psi_n \vDash \mathsf{true}$ and $\zeta_n \vDash \mathsf{div}$—that is, ψ_n characterizes terminating outcomes while ζ_n characterizes nontermination. Then, the weighting function $p_n \vDash \psi_n$ is precisely the terminating component of $[\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m)$.

Since $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$, it holds that:

$$\psi_\infty \ \dashv\ \sum_{n\in\mathbb{N}} p_n = \sum_{n\in\mathbb{N}} \mathsf{true}\ ?\ \left([\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m)\right)$$

By **??**, the projection commutes with the sum:

$$= \mathsf{true}\ ?\ \left(\sum_{n\in\mathbb{N}} [\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m)\right)$$

(ii) Since $[\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m) \vDash \psi_n \oplus \zeta_n$ for all n and $(\varphi_n \oplus \zeta_n)_{n\in\mathbb{N}} \Uparrow \zeta_\infty$, we have

$$\zeta_\infty \ \dashv\ \left(\inf_{n\in\mathbb{N}} \left| [\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)\right| \cdot \top\right) \cdot \delta_\cup$$

Since $\sqsubseteq$ is defined pointwise on $\mathcal{W}_{\mathcal{A}}(\Sigma_\cup)$, but reverses the order on $\cup$:

$$= \sup_{n\in\mathbb{N}} \left(\left| [\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)\right| \cdot \top \cdot \delta_\cup\right)$$

$\bot_f = \top \cdot \delta_\cup$, $\bot_f^\dagger(m) = |m| \cdot \top \cdot \delta_\cup$ for any weighting function m, so:

$$= \sup_{n\in\mathbb{N}} \bot_f^\dagger\left([\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger\ (m)\right)$$

The support of this weighting function must be a subset of $\{\cup\}$, by definition of $\bot_f$. Thus:

$$= \mathsf{div}\ ?\ \left(\sup_{n\in\mathbb{N}} \bot_f^\dagger\left([\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger\ (m)\right)\right)$$

We now combine these two results. By Corollary C.1.1, the unrolling of $\left[\!\left[C^{\langle e,e'\rangle}\right]\!\right]^{\dagger}(m)$ is the sum of the above components:

$$\left[\!\left[C^{\langle e,e'\rangle}\right]\!\right]^{\dagger}(m) = \mathsf{true}\ ?\ \left(\sum_{n\in\mathbb{N}} [\![(\mathbf{assume}\ e \fatsemi C)^n \fatsemi \mathbf{assume}\ e']\!]^{\dagger}(m)\right) + \mathsf{div}\ ?\ \left(\sup_{n\in\mathbb{N}} \bot_f^{\dagger}([\![(\mathbf{assume}\ e \fatsemi C)^n]\!]^{\dagger}(m))\right)$$

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

□

C.2 Relative Completeness

We now prove Lemma 3.3.1, the key lemma underlying our relative completeness result from Section 3.3.4.

Lemma 3.3.1. $\Omega \vdash \langle\varphi\rangle\ C\ \langle\mathsf{post}(C,\varphi)\rangle$

Proof. We proceed by induction on the structure of the program C.

- $C = \mathbf{skip}$. $[\![\mathbf{skip}]\!]^{\dagger}(m) = m$ for all $m \in \mathcal{W}_{\mathcal{A}}(\Sigma_{\cup})$, so $\mathsf{post}(\mathbf{skip},\varphi) = \varphi$. 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 IH, we have

$$\Omega \vdash \langle \varphi \rangle\ C_1\ \langle \mathsf{post}(C_1, \varphi) \rangle$$

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

The derivation is completed with the SEQ rule:

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

▹ $C = C_1 + C_2$.

$$\mathsf{post}(C_1 + C_2, \varphi) = \left\{ [\![C_1 + C_2]\!]^\dagger(m) \mid m \in \varphi \right\}$$

By Lemma C.0.1, we can split decompose each m into terminating and nonterminating components:

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

By Lemma A.0.1 (i), this is equal to:

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

Observe that $\mathsf{supp}(\mathsf{true}\ ?\ m) \subseteq \Sigma$. Applying (iii) and (iv) of Lemma A.0.1:

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

$$= \bigcup_{m \in \varphi} \left\{ [\![C_1]\!]^\dagger(m') + [\![C_2]\!]^\dagger(m') + \mathsf{div}\ ?\ m \mid m' \in \mathbf{1}(\mathsf{true}\ ?\ m) \right\}$$

$$= \exists m : \varphi.\ \mathsf{post}(C_1, \mathsf{true}\ ?\ m) \oplus \mathsf{post}(C_2, \mathsf{true}\ ?\ m) \oplus \Uparrow^{(m(\circlearrowleft))}$$

We split the derivation into parts. First, note that $\mathbf{1}(\mathsf{true}\ ?\ m) \vDash \mathsf{true}$ for any m. Then, we have the subderivation $(*)$:

$$\dfrac{\dfrac{\Omega}{\langle \mathbf{1}(\mathsf{true}\ ?\ m) \rangle\ C_1\ \langle \mathsf{post}(C_1, \mathbf{1}(\mathsf{true}\ ?\ m)) \rangle} \qquad \dfrac{\Omega}{\langle \mathbf{1}(\mathsf{true}\ ?\ m) \rangle\ C_2\ \langle \mathsf{post}(C_2, \mathbf{1}(\mathsf{true}\ ?\ m)) \rangle} \vdots}{\langle \mathbf{1}(\mathsf{true}\ ?\ m) \rangle\ C_1 + C_2\ \langle \mathsf{post}(C_1, \mathbf{1}(\mathsf{true}\ ?\ m)) \oplus \mathsf{post}(C_2, \mathbf{1}(\mathsf{true}\ ?\ m)) \rangle}\ \textsc{Plus}$$

The full derivation is completed as follows:

$$\dfrac{\dfrac{\dfrac{(*)}{\langle \mathbf{1}(\mathsf{true}\ ?\ m) \rangle\ C_1 + C_2\ \langle \ldots \rangle}\ \textsc{Plus} \qquad \dfrac{}{\langle \Uparrow^{(m(\circlearrowleft))} \rangle\ C_1 + C_2\ \langle \Uparrow^{(m(\circlearrowleft))} \rangle}\ \textsc{Div}}{\forall m \in \varphi.\ \langle \mathbf{1}(m) \rangle\ C_1 + C_2\ \langle \mathsf{post}(C_1, \mathbf{1}(\mathsf{true}\ ?\ m)) \oplus \mathsf{post}(C_2, \mathbf{1}(\mathsf{true}\ ?\ m)) \oplus \Uparrow^{(m(\circlearrowleft))} \rangle}\ \textsc{Choice}}{\langle \varphi \rangle\ C_1 + C_2\ \langle \mathsf{post}(C_1 + C_2, \varphi) \rangle}\ \textsc{Exists}$$

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

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

Suppose that e is a test $b \in \mathsf{Test}$. By Lemma C.0.1, $\mathsf{true}\ ?\ m = b\ ?\ m + \neg b\ ?\ m$, so:

$$\mathbf{1}(\mathsf{true}\ ?\ m) = \mathbf{1}(b\ ?\ m) \oplus \mathbf{1}(\neg b\ ?\ m)$$

Then, we have that:

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

By definition, $b\ ?\ m \vDash b = \mathbb{1}$ and $\neg b\ ?\ m \vDash b = \mathbb{0}$. Then, we have $(*)$:

$$\dfrac{\dfrac{}{\mathbf{1}(b\ ?\ m) \vDash b = \mathbb{1}}\ \textsc{Assume} \qquad \dfrac{}{\mathbf{1}(\neg b\ ?\ m) \vDash b = \mathbb{0}}\ \textsc{Assume}}{\langle \mathbf{1}(b\ ?\ m) \oplus \mathbf{1}(\neg b\ ?\ m) \rangle\ \mathbf{assume}\ b\ \langle \mathbf{1}(\neg b\ ?\ m) \rangle}\ \textsc{Choice}$$

The derivation is completed as follows:

$$\dfrac{\forall m \in \varphi. \qquad \dfrac{\dfrac{}{\langle \Uparrow^{(m(\circlearrowleft))} \rangle\ \mathbf{assume}\ b\ \langle \Uparrow^{(m(\circlearrowleft))} \rangle}\ \textsc{Div} \qquad \vdots \qquad (*)}{\langle \mathbf{1}(b\ ?\ m) \oplus \mathbf{1}(\neg b\ ?\ m) \oplus \Uparrow^{(m(\circlearrowleft))} \rangle\ \mathbf{assume}\ b\ \langle \mathbf{1}(b\ ?\ m) \oplus \Uparrow^{(m(\circlearrowleft))} \rangle}\ \textsc{Choice}}{\langle \varphi \rangle\ \mathbf{assume}\ b\ \langle \mathsf{post}(\mathbf{assume}\ b, \varphi) \rangle}\ \textsc{Exists}$$

Suppose instead that e is a weight $u \in U$. Since $\mathsf{supp}(\mathsf{true}\ ?\ m) \subseteq \Sigma$, then by *Lemma A.0.1*(*iv*):

$$\begin{aligned}[\![\mathbf{assume}\ u]\!]^{\dagger}(\mathsf{true}\ ?\ m) &= \left(\lambda\sigma.\ u \cdot \eta(\sigma)\right)^{\dagger}(\mathsf{true}\ ?\ m) \\ &= \left([\lambda\sigma.\ \eta(\sigma)] \cdot u\right)^{\dagger}(\mathsf{true}\ ?\ m) \\ &= (\mathsf{true}\ ?\ m) \cdot u\end{aligned}$$

Consequently:

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

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

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

▹ $C = C^{\langle e, e' \rangle}$. Note that $\varphi = \exists m.\ \varphi.\ \mathbf{1}(m)$. To use the ITER rule, we define the corresponding families of assertions, parametric on a weighting function m:

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

Observe that

$$\begin{aligned}\varphi_n(m) \oplus \zeta_n(m) &= \mathbf{1}([\![(\mathbf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)) = \mathsf{post}((\mathbf{assume}\ e \fatsemi C)^n, \mathbf{1}(m)) \\ \psi_\infty(m) \oplus \zeta_\infty(m) &= \mathbf{1}\left(\left[\!\left[C^{\langle e, e' \rangle}\right]\!\right]^\dagger(m)\right) = \mathsf{post}(C^{\langle e, e' \rangle}, \mathbf{1}(m))\end{aligned}$$

Next, we show that the premises of ITER hold for these definitions:

- $\Big(\psi_n(m)\Big)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty(m)$.

  Consider a family $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \psi_n(m)$ for all n. By definition of $\psi_n(m)$, m_n can only be one weighting function: $[\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m)$. Then,

$$\sum_{n\in\mathbb{N}} m_n = \sum_{n\in\mathbb{N}} \mathsf{true}\ ?\ \Big([\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m)\Big)$$

  By Lemma C.0.2:

$$= \mathsf{true}\ ?\ \left(\sum_{n\in\mathbb{N}} [\![(\textbf{assume}\ e \fatsemi C)^n \fatsemi \textbf{assume}\ e']\!]^\dagger(m)\right)$$

  By Corollary C.1.1, this is precisely the stateful component of $\Big[\!\Big[C^{\langle e,e'\rangle}\Big]\!\Big]^\dagger(m)$:

$$= \mathsf{true}\ ?\ \Big[\!\Big[C^{\langle e,e'\rangle}\Big]\!\Big]^\dagger(m) \vDash \mathbf{1}\Big(\Big[\!\Big[C^{\langle e,e'\rangle}\Big]\!\Big]^\dagger(m)\Big) = \psi_\infty(m)$$

- $\Big(\varphi_n(m) \oplus \zeta_n(m)\Big)_{n\in\mathbb{N}} \Uparrow \zeta_\infty(m)$.

  Take a family $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \varphi_n(m) \oplus \zeta_n(m)$ for all n. Then, $m_n \vDash \mathbf{1}([\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger(m))$, so $m_n = [\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)$. We have that:

$$(\inf_{n\in\mathbb{N}} |m_n| \cdot \top) \cdot \delta_\cup = \left(\inf_{n\in\mathbb{N}} \Big|[\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)\Big| \cdot \top\right) \cdot \delta_\cup$$

  As in the ITER case of Soundness, this is equal to:

$$= \sup_{n\in\mathbb{N}} \bot_f^\dagger\Big([\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)\Big)$$
$$= \mathsf{div}\ ?\ \left(\sup_{n\in\mathbb{N}} \bot_f^\dagger\Big([\![(\textbf{assume}\ e \fatsemi C)^n]\!]^\dagger(m)\Big)\right)$$

  By Corollary C.1.1, this is equal to $\mathsf{div}\ ?\ \Big[\!\Big[C^{\langle e,e'\rangle}\Big]\!\Big]^\dagger(m)$, which certainly satisfies $\Uparrow^{\left(\left[\!\left[C^{\langle e,e'\rangle}\right]\!\right]^\dagger(m)\right)} = \zeta_\infty(m)$.

- $\Omega \vdash \langle \varphi_n(m) \oplus \zeta_n(m) \rangle$ **assume** e ⨟ C $\langle \varphi_{n+1}(m) \oplus \zeta_{n+1}(m) \rangle$.

$$\begin{aligned}
\varphi_{n+1}(m) \oplus \zeta_{n+1}(m) &= \left\{ [\![(\textbf{assume } e \fatsemi C)^{n+1}]\!]^\dagger(m) \right\} \\
&= \left\{ [\![\textbf{assume } e \fatsemi C]\!]^\dagger([\![(\textbf{assume } e \fatsemi C)^n]\!]^\dagger(m)) \right\} \\
&= \left\{ [\![\textbf{assume } e \fatsemi C]\!]^\dagger(m') \mid m' \in \varphi_n(m) \oplus \zeta_n(m) \right\} \\
&= \mathsf{post}(\textbf{assume } e \fatsemi C, \varphi_n(m) \oplus \zeta_n(m))
\end{aligned}$$

  By the IH, this triple is derivable.

- $\Omega \vdash \langle \varphi_n(m) \rangle$ **assume** e' $\langle \psi_n(m) \rangle$.

$$\begin{aligned}
\varphi_n(m) &= \left\{ \mathsf{true} \ ? \ [\![(\textbf{assume } e \fatsemi C)^n \fatsemi \textbf{assume } e']\!]^\dagger(m) \right\} \\
&= \left\{ [\![\textbf{assume } e']\!]^\dagger(\mathsf{true} \ ? \ (\textbf{assume } e \fatsemi C)^n) \right\} \\
&= \left\{ [\![\textbf{assume } e']\!]^\dagger(m') \mid m' \in \varphi_n(m) \right\} \\
&= \mathsf{post}(\textbf{assume } e', \varphi_n(m))
\end{aligned}$$

  By the IH, this triple is derivable.

- For all $n \in \mathbb{N}$, it is clear that $\varphi_n(m), \psi_n(m) \vDash \mathsf{true}$ and $\zeta_n(m) \vDash \mathsf{div}$.

By definition, $\varphi_0(m) = \mathbf{1}(\mathsf{true} \ ? \ m)$ and $\zeta_0(m) = \Uparrow^{(m(\circlearrowleft))}$, so

$$\varphi_0(m) \oplus \zeta_0(m) = \mathbf{1}\Big(\mathsf{true} \ ? \ m + \mathsf{div} \ ? \ m\Big) = \mathbf{1}(m)$$

Then,

$$\varphi = \exists m : \varphi. \ \varphi_0(m) \oplus \zeta_0(m)$$

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

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

$$\dfrac{\dfrac{\forall n.\ \langle \varphi_n(m) \oplus \zeta_n(m) \rangle\ \mathbf{assume}\ e \fatsemi C\ \langle \varphi_{n+1}(m) \oplus \zeta_{n+1}(m) \rangle \qquad \Omega \qquad \vdots \qquad \dfrac{\Omega}{\langle \varphi_n(m) \rangle\ \mathbf{assume}\ e'\ \langle \psi_n(m) \rangle}}{\langle \varphi_0(m) \oplus \zeta_0(m) \rangle\ C^{\langle e,e' \rangle}\ \langle \psi_\infty(m) \oplus \zeta_\infty(m) \rangle}\ \textsc{Iter}}{\langle \varphi \rangle\ C^{\langle e,e' \rangle}\ \langle \mathsf{post}(C^{\langle e,e' \rangle}, \varphi) \rangle}\ \textsc{Exists}$$

▹ $C = a$. 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 $\vDash \langle \varphi \rangle\ a\ \langle \mathsf{post}(a, \varphi) \rangle$.

□

APPENDIX D

RULE DERIVATIONS IN TOL

Provided here are the derivations of rules introduced in Chapter 5.

Lemma D.0.1. *The following rules are derivable*

$$\textsc{Lemma D.0.1 - Partial}\ \frac{\langle \lceil P \rceil \rangle\ C\ \langle \Box Q \rangle}{\langle \Box P \rangle\ C\ \langle \Box Q \rangle} \qquad \frac{\langle \lceil P \rceil \rangle\ C\ \langle \blacksquare Q \rangle}{\langle \blacksquare P \rangle\ C\ \langle \blacksquare Q \rangle}\ \textsc{Lemma D.0.1 - Total}$$

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

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

□

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

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

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

Lemma D.0.2. *The following rule is derivable:*

$$\frac{\langle \lceil P \rceil \rangle\ C\ \langle \Diamond Q \rangle}{\langle \Diamond P \rangle\ C\ \langle \Diamond Q \rangle}\ \textsc{Lemma D.0.2}$$

Proof. For each $m \in \Diamond P$, we can fix a state $\sigma_m \in \mathsf{supp}(m)$ such that $\sigma_m \in \lceil P \rceil$ as well. This must exist by definition of $\Diamond P$. Let $u_m = m(\sigma)$ be its assigned weight. Note that $u_m \neq \mathbb{0}$. Then, we have as consequences:

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

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

The derivation is as follows:

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

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

□

Lemma D.0.3. *The following rules are derivable:*

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

Proof. We show only the derivation for Seq-Total-Hoare – we derive Seq-Lisbon nearly identically using an application of Lemma D.0.2.

$$\dfrac{\langle \lceil P \rceil \rangle\ C_1\ \langle \blacksquare Q \rangle \qquad \dfrac{\langle \lceil Q \rceil \rangle\ C_2\ \langle \blacksquare R \rangle}{\blacksquare Q \rangle\ C_2\ \langle \blacksquare R \rangle}\ \textsc{Lemma D.0.1 - Total}}{\langle \lceil P \rceil \rangle\ C_1 \fatsemi C_2\ \langle \blacksquare R \rangle}\ \textsc{Seq}$$

□

Lemma D.0.4. *The following rule is derivable:*

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

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

$$\dfrac{\dfrac{\dfrac{\varphi_1 \vDash b}{\langle \varphi_1 \rangle\ \mathbf{assume}\ b\ \langle \varphi_1 \rangle}\ \textsc{Assume} \quad \dfrac{\varphi_2 \vDash \neg b}{\langle \varphi_2 \rangle\ \mathbf{assume}\ b\ \langle \mathbb{0} \odot \varphi_2 \rangle}\ \textsc{Assume}}{\langle \varphi_1 \oplus \varphi_2 \rangle\ \mathbf{assume}\ b\ \langle \varphi_1 \rangle}\ \textsc{Choice} \quad \langle \varphi_1 \rangle\ C\ \langle \psi_1 \rangle}{\langle \varphi_1 \oplus \varphi_2 \rangle\ \mathbf{assume}\ b \fatsemi C_1\ \langle \psi_1 \rangle}\ \textsc{Seq}$$

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

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

□

Lemma D.0.5. *The following rule is derivable:*

$$\dfrac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \blacksquare Q \rangle \qquad \langle \lceil P \wedge \neg b \rceil \rangle\ C_2\ \langle \blacksquare Q \rangle}{\langle \lceil P \rceil \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \blacksquare Q \rangle}\ \textsc{If-Hoare}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \blacksquare Q \rangle}{\langle \blacksquare(P \wedge b) \rangle\ C_1\ \langle \blacksquare Q \rangle}\ \textsc{Lemma D.0.1} \quad \dfrac{\langle \lceil P \wedge \neg b \rceil \rangle\ C_2\ \langle \blacksquare Q \rangle}{\langle \blacksquare(P \wedge \neg b) \rangle\ C_2\ \langle \blacksquare Q \rangle}\ \textsc{Lemma D.0.1}}{\langle \blacksquare(P \wedge b) \oplus \blacksquare(P \wedge \neg b) \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \blacksquare Q \oplus \blacksquare Q \rangle}\ \textsc{If}}{\langle \lceil P \rceil \rangle\ \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \langle \blacksquare Q \rangle}\ \textsc{Consequence}$$

□

Lemma D.0.6. *The following rule is derivable:*

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

Proof. Subderivation (1):

$$\lceil P \wedge b \rceil \vDash b$$

$$\dfrac{\langle \lceil P \wedge b \rceil \rangle\ C_1\ \langle \Diamond Q \rangle \qquad \mathbb{0} \odot \top \vDash \neg b \qquad \dfrac{}{\langle \top \rangle\ C_2\ \langle \top \rangle}\ \textsc{True}}{\langle \lceil P \wedge b \rceil \rangle\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{If}$$

$$\frac{\langle \lceil P \wedge b \rceil \rangle\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \Diamond Q \rangle}{\langle \Diamond (P \wedge b) \rangle\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{Lemma D.0.2}$$

Part (2) is symmetrical.

$$\dfrac{\dfrac{\dfrac{(1)}{\langle \Diamond (P \wedge b) \rangle\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{Lemma D.0.2} \qquad \dfrac{(2)}{\langle \Diamond (P \wedge \neg b) \rangle\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{Lemma D.0.2}}{\langle \Diamond (P \wedge b) \vee \Diamond (P \wedge \neg b) \rangle\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \Diamond Q \vee \Diamond Q \rangle}\ \textsc{Disj}}{\langle \lceil P \rceil \rangle\ \texttt{if}\ b\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \langle \Diamond Q \rangle}\ \textsc{Consequence}$$

□

Lemma D.0.7. *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 $m \vDash \zeta$, $\mathsf{supp}(m) \subseteq \{\circlearrowright\}$. Then, these are equivalent: $\zeta = \exists m : \zeta.\ \Uparrow^{(m(\circlearrowright))}$. The derivation is as follows:

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

□

Lemma D.0.8. *The following rule is derivable:*

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

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

$$\dfrac{\textsc{Assume}\ \dfrac{\varphi_n \vDash b}{\langle \varphi_n \rangle\ \textbf{assume}\ b\ \langle \varphi_n \rangle} \qquad \dfrac{\psi_n \vDash \neg b}{\langle \psi_n \rangle\ \textbf{assume}\ b\ \langle \mathbb{0} \cdot \top \rangle}\ \textsc{Assume} \qquad \dfrac{\dfrac{\zeta_n \vDash \mathsf{div}}{\langle \zeta_n \rangle\ \textbf{assume}\ b\ \langle \zeta_n \rangle}\ \textsc{Div*} \qquad \vdots}{\langle \varphi_n \oplus \psi_n \oplus \zeta_n \rangle\ \textbf{assume}\ b\ \langle \varphi_n \oplus \zeta_n \rangle}\ \textsc{Choice}$$

This gives us (1):

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

And we derive (2) similarly:

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

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

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

□

Lemma D.0.9. *The following rule is derivable:*

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

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

$$\left(\inf_{n\in N} |m_n| \cdot \top\right) \cdot \delta_{\circlearrowleft} \vDash \zeta_\infty$$

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:

$$\dfrac{\dfrac{\dfrac{\langle \lceil P \wedge b \rceil \rangle\ C\ \langle \square P \rangle}{\langle \square(P \wedge b) \rangle\ C\ \langle \square P \rangle}\ \textsc{Lemma D.0.1 - Partial}}{\langle \blacksquare(P \wedge b) \oplus \exists u : U.\ \Uparrow^{(u)} \rangle\ C\ \langle \blacksquare(P \wedge b) \oplus \blacksquare(P \wedge \neg b) \oplus \exists u : U.\ \Uparrow^{(u)} \rangle}\ \textsc{Consequence}}{\langle \blacksquare(P \wedge b) \oplus \blacksquare(P \wedge \neg b) \oplus \exists u : U.\ \Uparrow^{(u)} \rangle\ \mathbf{while}\ b\ \mathbf{do}\ C\ \langle \blacksquare(P \wedge \neg b) \oplus \exists u : U.\ \Uparrow^{(u)} \rangle}\ \textsc{While}}{\langle \lceil P \rceil \rangle\ \mathbf{while}\ b\ \mathbf{do}\ C\ \langle \square(P \wedge \neg b) \rangle}\ \textsc{Consequence}$$

□

Lemma D.0.10. *The following rule is derivable:*

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

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 for the rule:

▹ 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

$$\left(\inf_{n\in\mathbb{N}} |m_n| \cdot \top\right) \cdot \delta_{\cup} = \mathbb{0} \cdot \delta_{\cup} \vDash \mathbb{0} \odot \top$$

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):

$$\dfrac{\dfrac{\forall m < N.\ \langle \varphi_{m+1} \rangle\ C\ \langle \varphi_m \rangle}{\forall n < N.\ \langle \varphi_{N-n} \rangle\ C\ \langle \varphi_{N-(n+1)} \rangle}}{\forall n < N.\ \langle \varphi'_n \rangle\ C\ \langle \varphi'_{n+1} \oplus \psi_{n+1} \rangle} \qquad \dfrac{\dfrac{}{\langle \top \rangle\ C\ \langle \top \rangle}\ \textsc{True}}{\langle \mathbb{0} \cdot T \rangle\ C\ \langle \mathbb{0} \odot \top \rangle}\ \textsc{Scale}}{\forall n \geq N.\ \langle \varphi'_n \rangle\ C\ \langle \varphi'_{n+1} \oplus \psi_{n+1} \rangle}$$

The derivation is completed as follows:

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

□

Lemma D.0.11. *The following rule is derivable:*

$$\textsc{Hoare-Variant}\quad \dfrac{(P \wedge b) \Rightarrow R > 0 \qquad \forall n \in \mathbb{N}.\ \langle \lceil P \wedge b \wedge R = n \rceil \rangle\ C\ \langle \blacksquare (P \wedge R < n) \rangle}{\langle \lceil P \wedge R \leq N \rceil \rangle\ \mathbf{while}\ b\ \mathbf{do}\ C\ \langle \blacksquare (P \wedge \neg b) \rangle}$$

Proof. We use WHILE and fix families of assertions φ_n, ψ_n and ζ_n for the rule.

For all $n \in \mathbb{N}$, let $\zeta_n \triangleq \mathbb{0} \odot \top$ and

$$\varphi_n \triangleq \begin{cases} \blacksquare(P \wedge R \leq (N-n) \wedge b) & \text{if } n \leq N \\ \mathbb{0} \odot \top & \text{if } n > N \end{cases} \qquad \psi_n \triangleq \begin{cases} \blacksquare(P \wedge R \leq (N-n) \wedge \neg b) & \text{if } n \leq N \\ \mathbb{0} \odot \top & \text{if } n > N \end{cases}$$

Let $\psi_\infty \triangleq \blacksquare(P \wedge \neg b)$ and $\zeta_\infty \triangleq \mathbb{0} \odot \top$.

We clearly have that $\varphi_n \vDash b$ and $\psi_n \vDash \neg b$. Next, we show that the necessary convergence/divergence conditions hold:

- Take any $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \psi_n$ for all n. Then $m_n \vDash \blacksquare(P \wedge \neg b)$, so $\sum_{n\in\mathbb{N}} m_n \vDash \blacksquare(P \wedge \neg b)$. This gives us $(\psi_n)_{n\in\mathbb{N}} \rightsquigarrow \psi_\infty$.
- Take $(m_n)_{n\in\mathbb{N}}$ such that $m_n \vDash \varphi_n \oplus \psi_n \oplus \zeta_n$ for all n. Then for $m > N$, $m_n \vDash \mathbb{0} \cdot \top$. So, $(\inf_{n\in\mathbb{N}} |m_n| \cdot \top \cdot \delta_\cup = 0 \cdot \delta_\cup)$. Thus, $(\varphi_n \oplus \psi_n \oplus \zeta_n) \Uparrow \zeta_\infty$.

Observe that for $n > N$, $\varphi_n \oplus \psi_n \oplus \zeta_n = \lceil P \wedge R \leq (N-n) \rceil$. In part (1) below, we derive the triple $\langle \varphi_n \oplus \zeta_n \rangle\ C\ \langle \varphi_{n+1} \oplus \psi_{n+1} \oplus \zeta_{n+1} \rangle$:

$$\dfrac{\dfrac{\dfrac{}{\langle \lceil P \wedge b \wedge R = m \rceil \rangle\ C\ \langle \blacksquare(P \wedge R < m) \rangle}\ \text{LEMMA D.0.1 - TOTAL}}{\langle \blacksquare(P \wedge b \wedge R = m) \rangle\ C\ \langle \blacksquare(P \wedge R < m) \rangle}\ \text{CONSEQUENCE}}{\dfrac{\forall m \leq N - n. \quad \langle \blacksquare(P \wedge b \wedge R = m) \rangle\ C\ \langle \blacksquare(P \wedge R \leq (N-(n+1))) \rangle}{\langle \blacksquare(P \wedge b \wedge R \leq (N-n)) \rangle\ C\ \langle \blacksquare(P \wedge R \leq (N-(n+1))) \rangle}\ \text{EXISTS}}$$

We then complete the derivation with an application of WHILE and CONSE-

QUENCE:

$$\dfrac{\dfrac{(1)}{\langle\blacksquare(P \wedge b \wedge R \leq (N-n))\rangle\ C\ \langle\blacksquare(P \wedge R \leq (N-(n+1)))\rangle}\ \textsc{While}}{\langle\blacksquare(P \wedge R \leq N)\rangle\ \texttt{while}\ b\ \texttt{do}\ C\ \langle\blacksquare(P \wedge \neg b)\rangle}\ \textsc{Consequence}$$

$$\langle\lceil P \wedge R \leq N\rceil\rangle\ \texttt{while}\ b\ \texttt{do}\ C\ \langle\blacksquare(P \wedge \neg b)\rangle$$

□

Lemma D.0.12. *The following rules are derivable:*

$$\dfrac{\lceil P\rceil \vDash b \qquad \langle\lceil P\rceil\rangle\ C\ \langle\square P\rangle}{\langle\lceil P\rceil\rangle\ \texttt{while}\ b\ \texttt{do}\ C\ \langle\square\mathsf{false}\rangle}\ \textsc{QInv-Demon}$$

$$\dfrac{\lceil P\rceil \vDash b \qquad \langle\lceil P\rceil\rangle\ C\ \langle\lozenge P\rangle}{\langle\lceil P\rceil\rangle\ \texttt{while}\ b\ \texttt{do}\ C\ \langle\blacklozenge\mathsf{false}\rangle}\ \textsc{QInv-Angel}$$

Proof. We show the full derivation for QINV-ANGEL.

We use the WHILE rule. For all n, take

$$\varphi_n \triangleq (\exists u : U \setminus \{\mathbb{0}\}.\ \lceil P\rceil^{(u)}) \oplus \blacksquare b \qquad \psi_n \triangleq \blacksquare(\neg b) \qquad \zeta_n = \exists u : U.\ \Uparrow^{(u)}$$

$$\psi_\infty \triangleq \top \qquad \zeta_\infty \triangleq \exists u : U \setminus \{\mathbb{0}\}.\ \Uparrow^{(u)}$$

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 $p \vDash \exists u : U \setminus \{\mathbb{0}\}.\ \lceil P\rceil^{(u)} = \varphi_n$, so $|m_n| > |p| > 0$. This means that, for weight $v > 0$,

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

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

Since $\lceil P \rceil \vDash b$, we know $\varphi_n = \exists u : U \setminus \{\mathbb{0}\}.\ \lceil P \rceil^{(u)} \oplus \blacksquare b \vDash b$. 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

$$\begin{aligned}
\varphi_n \oplus \psi_n \oplus \zeta_n &= (\exists u : U \setminus \{\mathbb{0}\}.\ \lceil P \rceil^{(u)} \oplus \blacksquare b) \oplus \blacksquare(\neg b) \oplus \exists u : U.\ \Uparrow^{(u)} \\
&= \exists u : U \setminus \{\mathbb{0}\}.\ \lceil P \rceil^{(u)} \oplus \top \\
&= \blacklozenge P \\
\psi_\infty \oplus \zeta_\infty &= \exists u : U \setminus \{\mathbb{0}\}.\ \Uparrow^{(u)} \oplus \top \quad = \quad \blacklozenge \mathsf{false}
\end{aligned}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle \lceil P \rceil \rangle\ C\ \langle \blacklozenge P \rangle}{\langle \blacklozenge P \rangle\ C\ \langle \blacklozenge P \rangle}\ \textsc{Lemma D.0.2}}{\langle \blacklozenge P \wedge \square b \rangle\ C\ \langle \blacklozenge P \rangle}\ \textsc{Consequence}}{\langle \blacklozenge P \rangle\ \texttt{while}\ b\ \texttt{do}\ C\ \langle \blacklozenge \mathsf{false} \rangle}\ \textsc{While}}{\langle \lceil P \rceil \rangle\ \texttt{while}\ b\ \texttt{do}\ C\ \langle \blacklozenge \mathsf{false} \rangle}\ \textsc{Consequence}$$

The derivation for QInv-Demon is similar, except we take:

$$\phi_n \triangleq \square P \qquad \psi_n \triangleq 0 \odot \top \qquad \psi_\infty \triangleq 0 \odot \top$$

□