Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning

NOAM ZILBERSTEIN, Cornell University, USA
DEREK DREYER, MPI-SWS, Germany
ALEXANDRA SILVA, Cornell University, USA

Program logics for bug-finding (such as the recently introduced Incorrectness Logic) have framed correctness and incorrectness as dual concepts requiring different logical foundations. In this paper, we argue that a single unified theory can be used for both correctness and incorrectness reasoning. We present Outcome Logic (OL), a novel generalization of Hoare Logic that is both *monadic* (to capture computational effects) and *monoidal* (to reason about outcomes and reachability). OL is guaranteed to find true bugs, while retaining *correctness* reasoning abilities as well. To formalize the applicability of OL to both correctness and incorrectness, we prove that any false OL specification can be disproven in OL itself. We also use our framework to reason about new types of incorrectness in nondeterministic and probabilistic programs. Given these advances, we advocate for OL as a new foundational theory of correctness and incorrectness.

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

Additional Key Words and Phrases: Program Logics, Hoare Logic, Incorrectness Reasoning

ACM Reference Format:
Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc. ACM Program. Lang.* 7, OOPSLA1 (February 2023), 51 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

"Program correctness and incorrectness are two sides of the same coin." – O'Hearn [2019]

1 INTRODUCTION

Developing formal methods to prove program *correctness*—the *absence* of bugs—has been a holy grail in program logic and static analysis research for many decades. However, seeing as many static analyses deployed in practice are *bug-finding* tools, O'Hearn [2019] recently advocated for the development of formal methods for proving program *incorrectness*; we need expressive, efficient, and compositional ways to reliably identify the *presence* of bugs as well.

The aforementioned paper of O'Hearn [2019] proposed Incorrectness Logic (IL) as a logical foundation for reasoning about program incorrectness. IL is inspired by—and in a precise technical sense *dual to*—Hoare Logic. Like Hoare Logic, IL specifications are *compositional*, given in terms of preconditions P and postconditions Q. Hoare Triples $\{P\}\ C\ \{Q\}$ stipulate that the result of running the program C on any state satisfying P will be a state that satisfies Q. Incorrectness Triples

Authors' addresses: Noam Zilberstein, noamz@cs.cornell.edu, Cornell University, USA; Derek Dreyer, dreyer@mpi-sws.org, MPI-SWS, Saarland Informatics Campus, Germany; Alexandra Silva, alexandra.silva@cornell.edu, Cornell University, USA.

2475-1421/2023/2-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

$[P]\ C\ [Q]$ *go in reverse*—all states satisfying Q must be reachable from some state satisfying P.

$$\begin{array}{lllllllll} \textbf{Hoare Logic:} & \vDash \{P\}\ C\ \{Q\} & \text{iff} & \forall \sigma \vDash P. & \forall \tau. & \tau \in [\![C]\!](\sigma) & \Rightarrow & \tau \vDash Q \\ \textbf{Incorrectness Logic:} & \vDash [P]\ C\ [Q] & \text{iff} & \forall \tau \vDash Q. & \exists \sigma. & \tau \in [\![C]\!](\sigma) & \text{and} & \sigma \vDash P \end{array}$$

Practically speaking, IL differs from Hoare Logic in two key ways. First, whereas Hoare Logic has no false negatives (*i.e.,* a verified program behaves correctly in *all* executions), IL has *no false positives*: any bug found using IL is in fact reachable by *some* execution of the program. Second, whereas Hoare Logic is over-approximate, IL is *under-approximate*: to prove that a program is incorrect, one only needs to specify (in the postcondition) a *subset* of the possible outcomes, which helps to ensure the efficiency of large-scale analyses. Subsequent work [Raad et al. 2020, 2022; Le et al. 2022] has focused on extending IL to account for a variety of program errors (*e.g.,* memory errors, memory leaks, data races, and deadlocks) and on using the resulting Incorrectness Separation Logics (ISLs) to explain and inform the development of bug-catching static analyses.

Despite these exciting advances, we argue that the foundations of incorrectness reasoning are still far from settled—and worthy of reconsideration. IL achieves *true positives* (reachability of end-states) and *under-approximation* through the same mechanism: quantification over all states that satisfy the postcondition. However, this conflation of concepts leads to several problems:

- The semantics of IL can only encompass types of incorrectness that are under-approximate. As we will see in Section 2.2, there are other types of incorrectness that cannot be expressed in IL. For example, in nondeterministic programs, IL can be used to show the *reachability* of bad states, but it cannot prove *non-reachability* of good states.
- IL is not amenable to probabilistic execution models and therefore is not a good fit for reasoning about incorrectness in randomized programs (Section 7.2).
- IL cannot easily describe what conditions are *sufficient* to trigger a bug (Section 6.6), meaning that analyses based on IL must implement extra algorithmic checks to determine whether a bug is worth reporting [Le et al. 2022].

Our key insight is that reachability and under-approximation are separate concepts that can (and should) be handled independently. But once reachability is separated from under-approximation, the resulting program logic no longer applies only to bug-finding. In this paper, we show how the full spectrum of correctness and incorrectness reasoning can be achieved with a unified foundation: a generalization of "good old" Hoare Logic that we call ***Outcome Logic (OL)***. In addition to consolidating the foundations of incorrectness with traditional correctness reasoning, OL overcomes all the aforementioned drawbacks of IL.

In OL, assertions are no longer predicates over program states, but rather predicates on an *outcome monoid*, whose elements can be, for instance, *sets of program states* or *probability distributions on program states*. The monoidal structure enables us to model a new *outcome conjunction*, $P \oplus Q$, asserting that the predicates P and Q each hold in *reachable* executions (or hold in subdistributions on program executions). We can also under-approximate by joining a predicate with $\top$, the trivial outcome: $P \oplus \top$ states that P only partially covers the program outcomes. OL offers several advantages as a unifying foundation for correctness and incorrectness:

Generality. OL unifies program analysis across two dimensions. First, since any untrue OL spec can be disproven in OL (Theorem 5.1), correctness and incorrectness reasoning are possible in a single program logic. Second, OL uses a monadic semantics which allows it to be instantiated for different evaluation models such as nondeterminism, erroneous termination, and probabilistic choice, thereby unifying correctness and incorrectness reasoning across execution models.

Beyond Reachability. Until now, the study of incorrectness has revolved primarily around *reachability* of crash states. We prove that OL handles a broader characterization of incorrectness than IL in nondeterministic programs (Theorem 5.6), as well as probabilistic incorrectness (Theorem 5.10).

Manifest Errors. In order to improve fix rates in automated bug finding tools, Le et al. [2022] only report bugs that occur regardless of context. These bugs—called *manifest errors*—are not straightforward to characterize using Incorrectness Logic: an auxiliary algorithm is needed to check whether some bug is truly a manifest error. In contrast, manifest errors are trivial to characterize in OL—Le et al.'s [2022] original definition can be expressed as an OL triple (Lemma 6.7).

The contributions of the paper are as follows:

- We provide an overview of the semantics of IL and explain what is needed in order to characterize broader classes of errors (Section 2). We show how reasoning about outcomes can account for reachability of end-states and enable under-approximation (when desired).
- We define Outcome Logic formally (Section 3 and Section 4), parametric on a monad and an assertion logic. We define syntax and semantics of the logic, using Bunched Implications (BI) formulae for pre- and postconditions, and provide inference rules to reason about validity.
- We show that OL is suitable for both correctness and incorrectness reasoning by proving that false OL triples can be disproven within OL (Section 5). As a corollary, OL can disprove Hoare triples, which was one motivation for IL (Corollary 5.7). We go further and show three kinds of incorrectness that can be captured in OL, only one of which is expressible in IL (Section 5.1).
- We exemplify how OL can be instantiated to find memory errors (Section 6) and probabilistic bugs (Section 7). We argue that the latter use case is not feasible in IL (Section 7.2).

Finally, we conclude in Section 8 and Section 9 by discussing related work and next steps.

2 OVERVIEW: A LANDSCAPE OF TRIPLES

The study of incorrectness has made apparent the need for new program logics that guarantee true positives and support under-approximate reasoning, since standard Hoare Logic—which does not enjoy those properties—is incapable of proving the presence of bugs. Concretely, in a valid Hoare Triple, denoted $\vDash \{P\}\ C\ \{Q\}$, running the program C in any state satisfying the precondition P will result in a state satisfying the postcondition Q (the formal definition is given in Figure 1). Suppose we wanted to use such a triple to prove that the program $x := \mathsf{malloc}() \fatsemi [x] \leftarrow 1$ has a bug (malloc may nondeterministically return null, causing the program to crash with a segmentation fault when the subsequent command attempts to store the value 1 at the location pointed to by x). We might be tempted to specify the triple as follows:

$$\{\mathsf{true}\}\ x := \mathsf{malloc}() \fatsemi [x] \leftarrow 1\ \{(\mathsf{ok} : x \mapsto 1) \vee (\mathsf{er} : x = \mathsf{null})\} \tag{1}$$

Here, the assertion $(\mathsf{ok} : p)$ means that the program terminated successfully in a state satisfying p and $(\mathsf{er} : q)$ means that it crashed in a state satisfying q. However, this is not quite right. According to the semantics of Hoare Logic, every possible end state must be covered by the postcondition, hence the need to use a disjunction to indicate that two outcomes are possible. But since we do not know that every state described by the postcondition is *reachable*, it is possible that every program trace ends up satisfying the first disjunct $(\mathsf{ok} : x \mapsto 1)$ and the error state is never reached.

Incorrectness Logic offers a solution to this problem. In a valid Incorrectness Triple, $\vDash [P]\ C\ [Q]$, every state satisfying Q is reachable by running C in some state satisfying P. So, simply switching the triple type in the above example *does* give us a witness that the error is possible.

$$[\mathsf{true}]\ x := \mathsf{malloc}() \fatsemi [x] \leftarrow 1\ [(\mathsf{ok} : x \mapsto 1) \vee (\mathsf{er} : x = \mathsf{null})] \tag{2}$$

Triple Name	Syntax		Semantics					
Hoare Logic	$\vDash \{P\}\ C\ \{Q\}$	iff	$\forall \sigma \vDash P.$	$\forall \tau.$	$\tau \in [\![C]\!](\sigma)$	$\Rightarrow$	$\tau \vDash Q$	
Incorrectness Logic (IL) / Reverse Hoare Logic (RHL)	$\vDash [P]\ C\ [Q]$	iff	$\forall \tau \vDash Q.$	$\exists \sigma.$	$\tau \in [\![C]\!](\sigma)$	and	$\sigma \vDash P$	
Outcome Logic (OL)	$\vDash \langle P \rangle\ C\ \langle Q \rangle$	iff	$\forall m.$	$m \vDash P$	$\Longrightarrow$	$[\![C]\!]^\dagger(m) \vDash Q$		

Fig. 1. Semantics of triples where P and Q are logical formulae, C is a program, Σ is the set of all program states, $\sigma, \tau \in \Sigma$, and $[\![C]\!] : \Sigma \to 2^\Sigma$ is the reachable states function. In the last line of the table, M is a monad, $m \in M\Sigma$ and $[\![C]\!]^\dagger : M\Sigma \to M\Sigma$ is the monadic lifting of $[\![\cdot]\!] : \Sigma \to M\Sigma$.

Though the conclusion remains a disjunction, the semantics of the triple (Figure 1) ensures that *every* state in the disjunction is reachable. Moreover, we can under-approximate by *dropping disjuncts* from the postcondition and use the simpler specification:

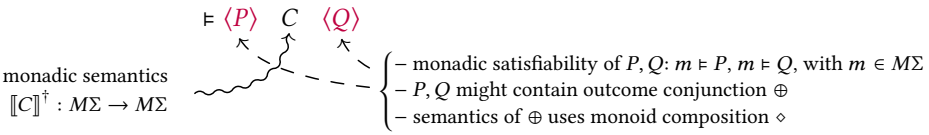
$$[\text{true}]\ x := \text{malloc}()\ ;\ [x] \leftarrow 1\ [\text{er} : x = \text{null}] \tag{3}$$

This more parsimonious specification still witnesses the error while also helping to ensure efficiency of large-scale automated analyses, which must keep descriptions at each program point small.

The duality between Hoare Logic and Incorrectness Logic appears sensible. Hoare Logic has *no false negatives*—a program is only correct if we account for all the possible outcomes. Incorrectness Logic has *no false positives*—an error is only worth reporting if it is truly reachable. However, we argue in this paper that incorrectness reasoning and Hoare Logic are *not* in fact at odds: an approach to incorrectness that is more similar to Hoare Logic is not only possible but in fact advantageous for several reasons, including the ability to express when an error will be *manifest* and the ability to reason about additional varieties of incorrectness.

2.1 Unifying Correctness and Incorrectness

Our first insight is that the inability to prove the existence of bugs is not inherent in the semantics of Hoare Logic. Rather, it is the result of an assertion logic that is not expressive enough to reason about reachability. Triple (1) shows how the usual logical disjunction is inadequate in reaching this goal. To remedy this, we use a logic with extra algebraic structure on outcomes, reminiscent of the use of a resource logic in separation logic [Reynolds 2002]. In this case, resources are program outcomes rather than heap locations. Program outcomes do not necessarily need to be the usual traces in a (non-)deterministic execution model, but can also arise from programs with alternative execution models such as probabilistic computation. To model different types of computations in a uniform way, we use an execution model parametric on a monad. We call this new logic Outcome Logic (OL), with triples denoted by $\langle P \rangle\ C\ \langle Q \rangle$ (defined formally in Figure 1). Let us schematically point out the generalizations in these new triples:



OL triples follow the spirit of Hoare Logic—first quantifying over elements satisfying the precondition and then stipulating that the result of running the program on such an element must satisfy the postcondition. The difference is that in OL triples, the pre- and postconditions are satisfied by a monoidal collection of outcomes $m \in M\Sigma$ rather than individual program states $\sigma \in \Sigma$. This allows us to introduce a new connective in the logic—the *outcome conjunction* $\oplus$—which models program

outcomes as resources. Consider the postcondition in triple (2) if we replace $\vee$ by $\oplus$:

$$(\mathsf{ok} : x \mapsto 1) \vee (\mathsf{er} : x = \mathsf{null}) \quad \text{vs.} \quad (\mathsf{ok} : x \mapsto 1) \oplus (\mathsf{er} : x = \mathsf{null})$$

A program state satisfies the first formula just by satisfying one of the disjuncts, whereas the second one requires a collection of states that can be split to witness satisfaction of both. This ability to split outcomes emerges as a requirement that $M\Sigma$ is a (partial commutative) monoid. Given two outcomes $m_1, m_2 \in M\Sigma$, there is an operation $\diamond$ that enables us to combine them $m_1 \diamond m_2 \in M\Sigma$. The satisfiability of $\oplus$ is then defined using $\diamond$ to split the monoidal state:

$$m \vDash P \oplus Q \qquad \text{iff} \qquad \exists m_1, m_2 \in M\Sigma.\quad m = m_1 \diamond m_2 \quad \text{and} \quad m_1 \vDash P \quad \text{and} \quad m_2 \vDash Q$$

Consider instantiating the above to the powerset monad that associates a set A with the set of its subsets $\mathcal{P}A$. Given a semantic function $[\![C]\!] : \Sigma \to \mathcal{P}\Sigma$ that maps individual start states σ to the set of final states reachable by executing C, we can give a monadic semantics $[\![C]\!]^\dagger(S) = \bigcup_{\sigma \in S} [\![C]\!]\,(\sigma)$ where S is a *set* of start states.[1] The monoid composition $\diamond$ on $\mathcal{P}A$ is given by set union, which is used compositionally to define satisfiability of $\oplus$ as follows: $S \vDash P \oplus Q$ iff $S_1 \vDash P$ and $S_2 \vDash Q$ such that $S = S_1 \cup S_2$. Given some satisfaction relation for individual program states $\vDash_\Sigma \subseteq \Sigma \times \mathsf{Prop}$, we then define satisfaction of atomic assertions as follows:

$$S \vDash P \qquad \text{iff} \qquad S \neq \emptyset \quad \text{and} \quad \forall \sigma \in S.\ \sigma \vDash_\Sigma P$$

The extra restriction $S \neq \emptyset$ witnesses that P is reachable (and not vacuously satisfied). Putting this all together, we instantiate the generic OL triples (Figure 1) to the powerset monad:

$$\vDash \langle P \rangle\ C\ \langle Q \rangle \qquad \text{iff} \qquad \forall S \in \mathcal{P}\Sigma.\quad S \vDash P \quad \Rightarrow \quad [\![C]\!]^\dagger(S) \vDash Q$$

Now, we can revisit the example in triple (2) in OL using $\oplus$ instead of $\vee$:

$$\langle \mathsf{ok} : \mathsf{true} \rangle\ x := \mathsf{malloc}() \fatsemi [x] \leftarrow 1\ \langle (\mathsf{ok} : x \mapsto 1) \oplus (\mathsf{er} : x = \mathsf{null}) \rangle \tag{4}$$

This specification *does* witness the bug—for any start state there is at least one end state that satisfies each of the outcomes. However, we are still recording extra, non-erroneous outcomes, which is problematic for a large scale analysis algorithm. Following the example in triple (3), we would like to specify the bug above in a way that mentions only the relevant outcome in the postcondition. We can achieve this by simply weakening the postcondition. According to the semantics above, the following implications hold:

$$S \vDash P \oplus Q \quad \Rightarrow \quad S \vDash P \oplus \top \qquad \text{and} \qquad S \vDash P \oplus Q \quad \Rightarrow \quad S \vDash \top \oplus Q$$

So in a sense, we can *drop outcomes* by converting them to $\top$. For notational convenience, we define the following under-approximate triple:

$$\vDash^{\downarrow} \langle P \rangle\ C\ \langle Q \rangle \qquad \text{iff} \qquad \vDash \langle P \rangle\ C\ \langle Q \oplus \top \rangle$$

Using this shorthand, the following simpler specification is also valid:

$$\vDash^{\downarrow} \langle \mathsf{ok} : \mathsf{true} \rangle\ x := \mathsf{malloc}() \fatsemi [x] \leftarrow 1\ \langle \mathsf{er} : x = \mathsf{null} \rangle \tag{5}$$

This example demonstrates that OL is suitable for reasoning about crash errors, just like IL. However our goal is not simply to cover the same use cases as IL, but rather to go further. Next, we will show in Section 2.2 that there are bugs expressible in OL that cannot be expressed in IL. In Section 2.3 we will also explain why the semantics of OL are a better fit for characterizing an important class of bugs known as manifest errors.

[1] The $[\![-]\!]^\dagger$ function is formally the monadic (or Kleisli) extension of $[\![-]\!]$; we will define this formally in Section 3.

2.2 A Broader Characterization of Correctness and Incorrectness

In the semantics of Incorrectness Logic, the notions of reachability and under-approximation are conflated: both are a consequence of the fact that IL quantifies over the states that satisfy the postcondition. However, reachability and under-approximation are separate concepts and OL allows us to reason about each independently. Reachability is expressed with the outcome conjunction $\oplus$ and under-approximation is achieved by dropping outcomes. Separating reachability and under-approximation is useful for *both* correctness *and* incorrectness reasoning.

To see this, we will first investigate *correctness* properties that rely on reachability. Before the introduction of Incorrectness Logic by O'Hearn [2019], de Vries and Koutavas [2011] devised a semantically equivalent logic, which they called Reverse Hoare Logic. The goal of this work was to prove *correctness* specifications that involved multiple possible end states, all of which must be reachable. As we saw in Example 1, Hoare Logic cannot express such specifications. So, de Vries and Koutavas [2011] proposed the Reverse Hoare Triple, which—like Incorrectness Triples—guarantees that every state described by the postcondition is reachable.

The motivating example for Reverse Hoare Logic was a nondeterministic shuffle function. Consider the following specification, where $\Pi(a)$ is the set of permutations of a:

$$[\mathsf{true}]\ b := \mathsf{shuffle}(a)\ [b \in \Pi(a)]$$

This specification states that *every* permutation of the list is a possible output of shuffle; however, it is not a complete correctness specification. It does not rule out the possibility that the output is not a permutation of the input ($b \notin \Pi(a)$). The semantics of Reverse Hoare Logic is motivated by *reachability*, but—like Incorrectness Logic—it achieves reachability in a manner that is inextricably linked to under-approximation, which is undesirable for correctness reasoning.

de Vries and Koutavas [2011] note this, stating that a complete specification for shuffle would require both Hoare Logic *and* Reverse Hoare Logic, but also that it would be worthwhile to study logics that can "express both the reachability of good states and the non-reachability of bad states" [de Vries and Koutavas 2011, §8]. OL does just that—the full correctness of the shuffle program can be captured using a single OL triple that guarantees reachability *without* under-approximating:

$$\langle \mathsf{true} \rangle\ b := \mathsf{shuffle}(a)\ \langle \bigoplus_{\pi \in \Pi(a)} (b = \pi) \rangle \tag{6}$$

The OL specification above states *not only* that all the permutations are reachable, *but also* that they are the only possible outcomes. So, OL allows us to express a correctness property in a single triple that otherwise would have required *both* a Hoare Triple *and* a Reverse Hoare Triple.

We now turn to consider incorrectness reasoning. Given that the above OL triple is a complete correctness specification, we are interested to know what it would mean for shuffle to be incorrect. In other words, what would it take to *disprove* the specification of shuffle? There are two ways that the triple could be false: either one particular permutation $\pi \in \Pi(a)$ is not reachable or the output b is not a permutation of a. Both bugs can be expressed as OL triples:

$$\exists \pi \in \Pi(a).\ \langle \mathsf{true} \rangle\ b := \mathsf{shuffle}(a)\ \langle b \neq \pi \rangle \qquad\qquad \langle \mathsf{true} \rangle\ b := \mathsf{shuffle}(a)\ \langle (b \notin \Pi(a)) \oplus \top \rangle$$

These triples both denote *true bugs* since the validity of either triple implies that specification (6) is false. In fact, these are the *only* ways that specification (6) can be false. This follows from a more general result called Falsification, which we prove in Theorem 5.6:

$$\not\vDash \langle P \rangle\ C\ \langle \bigoplus_{i=1}^{n} Q_i \rangle \quad \text{iff} \quad \exists P' \Rightarrow P.\ \ \exists i.\ \vDash \langle P' \rangle\ C\ \langle \neg Q_i \rangle \quad \text{or} \quad \vDash \langle P' \rangle\ C\ \langle (\bigwedge_{i=1}^{n} \neg Q_i) \oplus \top \rangle$$

Intuitively, a nondeterministic program is incorrect iff either one of the desired outcomes never occurs or some undesirable outcome sometimes occurs.[2] Incorrectness Logic can only characterize the latter type of incorrectness, whereas OL accounts for both and is thus strictly more expressive in the nondeterministic setting. An analogous result holds for probabilistic programs (Section 5.2), whereas IL is not suitable for reasoning about probabilistic incorrectness at all (Section 7.2).

2.3 Semantic Characterizations of Bugs

In addition to enabling us to witness a larger class of incorrectness than IL (unreachable states and probabilistic incorrectness), OL also provides a more *intuitive* way to reason about the type of bugs that IL was designed for: reachability of unsafe states.

Recalling the crash error in Section 2.1, both IL triples and OL triples soundly characterize the bug, as they both witness a trace that reaches the crash. The Incorrectness Triple (3) states that any failing execution where x is null is reachable from some starting state. In other words, true is a *necessary* condition to reach a segmentation fault. However, true is trivially a necessary condition, so this triple does not tell us much about what will trigger the bug in practice. By contrast, the OL triple (5) states that true is a *sufficient* condition, which gives us more information—the bug can *always* occur no matter what the starting state is.

The latter semantics has a close correspondence to a class of bugs, known as *manifest errors* [Le et al. 2022], which occur regardless of how the enclosing procedure is used and are of particular interest in automated bug-finding tools. Le et al. [2022, Def. 3.2] give a formal characterization of manifest errors, but it is not a natural fit for Incorrectness Logic: determining whether an IL triple is a manifest error requires an auxiliary algorithmic check. Though Le et al. [2022] note that there are connections between manifest errors and under-approximate variants of Hoare Logic, we go further in proving that their original definition of a manifest error is semantically equivalent to an OL triple of the form $\vDash^{\downarrow} \langle \mathsf{ok} : \mathsf{true} \rangle\ C\ \langle \mathsf{er} : q * \mathsf{true} \rangle$ (Lemma 6.7). Manifest errors are therefore trivial to characterize in OL by a simple syntactic inspection. This suggests that OL is semantically closer to the way in which programmers naturally characterize bugs.

In addition to being an intuitive foundation for incorrectness, OL unifies program analysis across two dimensions. First, it unifies correctness and incorrectness reasoning within a single program logic, and second, it does so across execution models (*e.g.,* nondeterministic and probabilistic). In the remainder of the paper, we will formalize the ideas that have been exemplified thus far. We formalize the OL model in Section 3 and Section 4, prove the applicability of OL to nondeterministic and probabilistic correctness and incorrectness in Section 5, and show how OL can be used in nondeterministic and probabilistic domains in Section 6 and Section 7, respectively. Given these advantages, we argue that OL offers a promising alternative foundation for incorrectness reasoning.

3 A MODULAR PROGRAMMING LANGUAGE

We start by defining a programming language, inspired by Dijkstra's guarded command language [Dijkstra 1975], see Figure 2. The syntax includes $\mathbb{0}$, which represents divergence, $\mathbb{1}$, acting as skip, sequential composition $C_1 \fatsemi C_2$, choice $C_1 + C_2$, iteration $C^\star$, and parametrizable atomic commands c. At first sight this looks like a standard imperative language (with nondeterministic choice). However, we will interpret the syntax in a semantic model that is parametric on a monad and a partial commutative monoid. The former enables a generic semantics of sequential composition, whereas the latter provides a generic interpretation of choice.

[2] In general, there is also a third option: the program diverges (has no outcomes). See Theorem 5.6.

$$[\![C]\!] : \Sigma \rightharpoonup M\Sigma$$

$$\begin{array}{lll} C ::= \mathbb{0} & & [\![\mathbb{0}]\!](\sigma) = \varnothing \\ \mid \mathbb{1} & & [\![\mathbb{1}]\!](\sigma) = \mathsf{unit}(\sigma) \\ \mid C_1 \fatsemi C_2 & & [\![C_1 \fatsemi C_2]\!](\sigma) = \mathsf{bind}([\![C_1]\!](\sigma), [\![C_2]\!]) \\ \mid C_1 + C_2 & & [\![C_1 + C_2]\!](\sigma) = [\![C_1]\!](\sigma) \diamond [\![C_2]\!](\sigma) \\ \mid C^\star & & [\![C^\star]\!](\sigma) = \mathsf{lfp}(\lambda f.\lambda\sigma.f^\dagger([\![C]\!](\sigma))) \diamond \mathsf{unit}(\sigma))(\sigma) \\ \mid c & & [\![c]\!](\sigma) = [\![c]\!]_{\mathsf{atom}}(\sigma) \end{array}$$

Fig. 2. Syntax and Semantics of the Command Language parameterized by an execution model $\langle M, \mathsf{bind}, \mathsf{unit}, \diamond, \varnothing\rangle$ and a language of atomic commands with semantics $[\![c]\!]_{\mathsf{atom}} : \Sigma \to M\Sigma$

Before we define the semantic model we need to recall the definition of a monad and partial commutative monoid. We assume familiarity with basic category theory (categories, functors, natural transformations), see [Pierce 1991] for an introduction.

Definition 3.1 (Monad). A monad is a triple $\langle M, \mathsf{bind}, \mathsf{unit}\rangle$ in which M is a functor on a category C, $\mathsf{unit} : \mathsf{Id} \Rightarrow M$ is a natural transformation, and $\mathsf{bind} : MA \times (A \to MB) \to MB$ satisfies:

(1) $\mathsf{bind}(m, \mathsf{unit}) = m$
(2) $\forall a \in A.\ \mathsf{bind}(\mathsf{unit}(a), f) = f(a)$
(3) $\mathsf{bind}(\mathsf{bind}(m, f))(g) = \mathsf{bind}(m, \lambda a.\mathsf{bind}(f(a), g))$

Typical examples of monads include powerset, error, and distribution monads (defined in Section 5 and Section 6). Given a function $f : A \to MB$, its monadic extension $f^\dagger : MA \to MB$ is defined as $f^\dagger(m) = \mathsf{bind}(m, f)$.

Definition 3.2 (PCM). A partial commutative monoid (PCM) is a triple $\langle S, \diamond, \varnothing\rangle$ consisting of a set S and a partial binary operation $\diamond : S \to S \rightharpoonup S$ that is associative, commutative, and has unit $\varnothing$.

A typical example of a PCM, used in probabilistic reasoning, is $\langle [0, 1], +, 0\rangle$ (+ is partial, it is undefined when the addition is out-of-bounds). We are now ready to define the execution model we need to provide semantics to our language.

Definition 3.3 (Execution Model). An Execution Model is a structure $\langle M, \mathsf{bind}, \mathsf{unit}, \diamond, \varnothing\rangle$ such that $\langle M, \mathsf{bind}, \mathsf{unit}\rangle$ is a monad in the category of sets, and for any set A, $\langle MA, \diamond, \varnothing\rangle$ is a PCM that preserves the monad bind: $\mathsf{bind}(m_1 \diamond m_2, k) = \mathsf{bind}(m_1, k) \diamond \mathsf{bind}(m_2, k)$ and $\mathsf{bind}(\varnothing, k) = \varnothing$.

In Figure 2 we present the semantics of the language. The monad operations are used to provide semantics to $\mathbb{1}$ and sequential composition $\fatsemi$ whereas the monoid operation is used in the semantics of choice and iteration. Note that in general the semantics of the language is a partial function since $\diamond$ is partial. The partiality of $\diamond$ is necessary in order to express a probabilistic semantics, since two probability distributions can only be combined if their cumulative probability mass is at most 1. For the languages we will work with in this paper, there are simple syntactic checks to ensure totality of the semantics. In the probabilistic case, this involves ensuring that all uses of + and $\star$ are guarded. We show that the semantics is total for the execution models of interest in Appendix A.

Example 3.4 (State and Guarded Commands). The base language introduced in the previous section is parametric over a set of program states Σ. In this example, we describe a specific type of program state, the semantics of commands over those states, and a mechanism to define the typical control flow operators (if and while). First, we assume some syntax of program expressions $e \in \mathsf{Exp}$

which includes variables $x \in \mathsf{Var}$ as well as the typical Boolean and arithmetic operators. Atomic commands come from the following syntax.

$$c ::= \mathsf{assume}\ e \mid x := e \qquad (x \in \mathsf{Var}, e \in \mathsf{Exp})$$

The command assume e does nothing if e is true and eliminates the current outcome if not; $x := e$ is variable assignment. A program stack is a mapping from variables to values $\mathcal{S} = \{s : \mathsf{Var} \to \mathsf{Val}\}$ where program values $\mathsf{Val} = \mathbb{Z} + \mathbb{B}$ are integers ($\mathbb{Z}$) or Booleans ($\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$). Expressions are evaluated to values given a stack using $[\![e]\!]_{\mathsf{Exp}} : \mathcal{S} \to \mathsf{Val}$. The semantics of atomic commands $[\![c]\!] : \mathcal{S} \to M\mathcal{S}$, parametric on an execution model, is defined below.

$$[\![\mathsf{assume}\ e]\!](s) = \begin{cases} \mathsf{unit}(s) & \text{if } [\![e]\!]_{\mathsf{Exp}}(s) = \mathsf{true} \\ \varnothing & \text{if } [\![e]\!]_{\mathsf{Exp}}(s) = \mathsf{false} \end{cases} \qquad [\![x := e]\!](s) = \mathsf{unit}(s[x \mapsto [\![e]\!]_{\mathsf{Exp}}(s)])$$

While a language instantiated with the atomic commands described above is still nondeterministic, we can use assume to define the usual (deterministic) control flow operators as syntactic sugar.

$$\begin{aligned} \mathsf{if}\ e\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 &= (\mathsf{assume}\ e \fatsemi C_1) + (\mathsf{assume}\ \neg e \fatsemi C_2) & \mathsf{skip} &= \mathbb{1} & C^0 &= \mathbb{1} \\ \mathsf{while}\ e\ \mathsf{do}\ C &= (\mathsf{assume}\ e \fatsemi C)^\star \fatsemi \mathsf{assume}\ \neg e & \mathsf{for}\ N\ \mathsf{do}\ C &= C^N & C^{k+1} &= C \fatsemi C^k \end{aligned}$$

In fact, when paired with a nondeterministic evaluation model, this language is equivalent to Dijkstra's [1975] Guarded Command Language (GCL) by a straightforward syntactic translation.

4 OUTCOME LOGIC

In this section, we formally define Outcome Logic (OL). We first define the logic of outcome assertions which will act as the basis for writing pre- and postconditions in OL. Next, we give the semantics of OL triples, which is parametric on an execution model, atomic command semantics, and an assertion logic. Finally, we give proof rules that are sound for all OL instances.

4.1 A Logic for Monoidal Assertions: Modeling the Outcome Conjunction

We now give a formal account of the outcome assertion logic that was briefly described in Section 2.1. The outcome assertion logic is an instance of the Logic of Bunched Implications (BI) [O'Hearn and Pym 1999], a substructural logic that is used to reason about *resource* usage. Separation logic [Reynolds 2002] and its extensions [O'Hearn 2004] are the most well-known applications of BI. In our case, the relevant resources are *program outcomes* rather than heap locations.

We use the formulation of BI due to Docherty [2019]. While Docherty [2019] gives a thorough account of the BI proof theory, we are mainly interested in the semantics for the purposes of this paper. The syntax and semantics are given in Figure 3 with logical negation $\neg\varphi$ being defined as $\varphi \Rightarrow \bot$. The semantics is parametric on a BI frame $\langle X, \diamond, \preccurlyeq, \varnothing\rangle$ where $\langle X, \diamond, \varnothing\rangle$ is a PCM and $\preccurlyeq \subseteq X \times X$ is a preorder, and a satisfaction relation for basic assertions $\vDash_{\mathsf{atom}} \subseteq X \times \mathsf{Prop}$.

The two non-standard additions are the *outcome conjunction* $\oplus$, a connective to join outcomes, and $\top^\oplus$, an assertion to specify that there are no outcomes. These intended meanings are reflected in the semantics: $\top^\oplus$ is only satisfied by the monoid unit $\varnothing$, whereas $\varphi \oplus \psi$ is satisfied by m iff m can be partitioned into $m_1 \diamond m_2$ to satisfy each outcome formula separately. We will focus on *classical* interpretations of BI where the preorder $\preccurlyeq$ is equality.[3]

Definition 4.1 (Outcome Assertion Logic). Given an execution model $\langle M, \mathsf{bind}, \mathsf{unit}, \diamond, \varnothing\rangle$ and a satisfaction relation for atomic assertions $\vDash_{\mathsf{atom}} \subseteq M\Sigma \times \mathsf{Prop}$, an Outcome Assertion Logic is an

[3]Intuitionistic interpretations of BI with non-trivial preorders can be used as an alternative way to encode under-approximate program logics. This idea is explored in Appendix B.1.

$\varphi ::=$	$\top$	$m \vDash \top$	always	
$\mid$	$\bot$	$m \vDash \bot$	never	
$\mid$	$\top^{\oplus}$	$m \vDash \top^{\oplus}$	iff	$m = \varnothing$
$\mid$	$\varphi \wedge \psi$	$m \vDash \varphi \wedge \psi$	iff	$m \vDash \varphi$ and $m \vDash \psi$
$\mid$	$\varphi \oplus \psi$	$m \vDash \varphi \oplus \psi$	iff	$\exists m_1, m_2.\ m_1 \diamond m_2 \preccurlyeq m$ and $m_1 \vDash \varphi$ and $m_2 \vDash \psi$
$\mid$	$\varphi \Rightarrow \psi$	$m \vDash \varphi \Rightarrow \psi$	iff	$\forall m'.$ if $m \preccurlyeq m'$ and $m' \vDash \varphi$ then $m' \vDash \psi$
$\mid$	P	$m \vDash P$	iff	$P \in \mathsf{Prop}$ and $m \vDash_{\mathsf{atom}} P$

Fig. 3. Syntax and semantics of BI given a BI frame $\langle X, \diamond, \preccurlyeq, \varnothing \rangle$ and satisfaction relation $\vDash_{\mathsf{atom}} \subseteq X \times \mathsf{Prop}$

instance of BI based on the BI frame $\langle M\Sigma, \diamond, =, \varnothing \rangle$. Informally, we refer to BI assertions φ, ψ as outcome assertions and the atomic assertions $P, Q \in \mathsf{Prop}$ as individual outcomes.

Remark 1 (Notation for Assertions). For the remainder of the paper, lowercase Greek metavariables φ, ψ refer to (syntactic) outcome assertions (Definition 4.1), uppercase Latin metavariables P, Q refer to atomic assertions (individual outcomes), and lowercase Latin metavariables p, q refer to assertions on individual program states.

Example 4.2 (Outcomes). We mentioned one example of a PCM in Section 2: X can be sets of program states and the monoid operation $\diamond$ is set union. Another example is probability (sub)distributions over a set and $\diamond$ is +. This monoid operation is partial; adding two subdistributions is only possible if the mass associated with a point (and the entire distribution) remains in $[0, 1]$.

As discussed in Section 2, under-approximation and the ability to drop outcomes is an important part of incorrectness reasoning as it allows large scale analyses to only track pertinent information. We use the following shorthand to express under-approximate outcome assertions.

Definition 4.3 (Under-Approximate Outcome Assertions). Given an outcome assertion logic with satisfaction relation $\vDash \subseteq M\Sigma \times \mathsf{Prop}$, we define an under-approximate satisfaction relation $\vDash^{\downarrow} \subseteq M\Sigma \times \mathsf{Prop}$ as $m \vDash^{\downarrow} \varphi$ iff $m \vDash \varphi \oplus \top$.

Intuitively, $\varphi \oplus \top$ corresponds to under-approximation since it states that φ only covers a subset of the outcomes (with the rest being unconstrained, since they are covered by $\top$). Defining under-approximation in this way allows us to reason about correctness and incorrectness within a single program logic. It also enables us to drop outcomes simply by weakening; it is always possible to weaken an outcome to $\top$, so $m \vDash P \oplus Q$ implies that $m \vDash P \oplus \top$. Equivalently, $m \vDash^{\downarrow} P \oplus Q$ implies that $m \vDash^{\downarrow} P$. These facts are proven in Appendix B. A similar formulation would be possible using an intuitionistic interpretation of BI (where, roughly speaking, we take the preorder to be $m_1 \preccurlyeq m_2$ iff $\exists m.\ m_1 \diamond m = m_2$). We prove this correspondence in Appendix B.1.

4.2 Outcome Triples

We now have all the ingredients needed to define the validity of the program logic.

Definition 4.4 (Outcome Triples). The parameters needed to instantiate OL are:

(1) An execution model: $\langle M, \mathsf{bind}, \mathsf{unit}, \diamond, \varnothing \rangle$
(2) A set of program states Σ and semantics of atomic commands: $[\![c]\!]_{\mathsf{atom}} : \Sigma \to M\Sigma$
(3) A syntax of atomic assertions Prop and satisfaction relation: $\vDash_{\mathsf{atom}} \subseteq M\Sigma \times \mathsf{Prop}$

Now, let $[\![-]\!] : \Sigma \to M\Sigma$ be the semantics of the language in Figure 2 with parameters (1) and (2) and $\vDash$ be the outcome assertion satisfaction relation (Definition 4.1) with parameters (1) and (3). For any program C (Figure 2), and outcome assertions φ and ψ:

$$\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \text{iff} \qquad \forall m \in M\Sigma.\quad m \vDash \varphi \quad \Longrightarrow \quad [\![C]\!]^{\dagger}(m) \vDash \psi$$

OL is a generalization of Hoare Logic—the triples first quantify over elements satisfying the precondition and then stipulate that the result of running the program on those elements satisfies the postcondition. The difference is that now the pre- and postconditions are expressed as outcome assertions and thus satisfied by a monoidal collection $m \in M\Sigma$, which can account for execution models such as nondeterminism and probability distributions.

Using outcome assertions for pre- and postconditions adds significant expressive power. We already saw in Section 2 how Outcome Logic allows us to reason about reachability and under-approximation. We can also encode other useful concepts such as partial correctness—the post-condition holds *if* the program terminates—by taking a disjunction with $\top^{\oplus}$ to express that the program may diverge[4]. For convenience, we define the following notation where the left triple encodes under-approximation and the right triple encodes partial correctness.

$$\vDash^{\downarrow} \langle \varphi \rangle\ C\ \langle \psi \rangle \quad \text{iff} \quad \vDash \langle \varphi \rangle\ C\ \langle \psi \oplus \top \rangle \qquad\qquad \vDash_{\text{pc}} \langle \varphi \rangle\ C\ \langle \psi \rangle \quad \text{iff} \quad \vDash \langle \varphi \rangle\ C\ \langle \psi \vee \top^{\oplus} \rangle$$

In fact, the right triple corresponds exactly to standard Hoare Logic (Figure 1) if we instantiate OL using the powerset semantics (Definition 5.3) and limit the pre- and post-conditions to be atomic assertions. This result is stated below and proven in Appendix C.

Theorem 4.5 (Subsumption of Hoare Triples). $\vDash \{P\}\ C\ \{Q\}$ *iff* $\vDash_{\text{pc}} \langle P \rangle\ C\ \langle Q \rangle$

While capturing many logics in one framework is interesting and demonstrates the versatility of Outcome Triples, our primary goal is to investigate the roles that these program logics can play for expressing correctness and incorrectness properties. We justify OL as a theoretical basis for correctness and incorrectness reasoning in Section 5 and give examples for how OL can be applied to nondeterministic and probabilistic programs in Section 6 and Section 7.

4.3 Proof Systems

Now that we have formalized the *validity* of Outcome triples (denoted $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$), we can construct proof systems for this family of logics. We write $\vdash \langle \varphi \rangle\ C\ \langle \psi \rangle$ to mean that the triple $\langle \varphi \rangle\ C\ \langle \psi \rangle$ is *derivable* from a set of inference rules. Each set of inference rules that we define throughout the paper will be *sound* with respect to a certain OL instance.

Global rules. Some generic rules that are valid for any OL instance are shown at the top of Figure 4. Most of the rules including Zero, One, and Seq are standard. The Rule of Consequence allows the strengthening and weakening of pre- and post-conditions respectively using any semantically valid BI implication. The Split rule allows us to analyze the program C with two different pre/postcondition pairs and join the results using an outcome conjunction.

Rules for nondeterministic programs. In the middle of Figure 4 we see two rules that are only valid in nondeterministic languages where the semantics is based on the powerset monad. The Plus rule characterizes nondeterministic choice by joining the outcomes from analyzing each branch using an outcome conjunction. Repeated uses of the Induction rule allow us to unroll an iterated command for a finite number of iterations.

Rules for guarded programs. Finally, at the bottom of Figure 4 is a collection of rules for expression-based languages that have the syntax introduced in Example 3.4. We write $P \vDash e$ to mean that P *entails* e. Formally, if $P \vDash e$ and $Q \vDash \neg e$ and $m \vDash P \oplus Q$, then $[\![\text{assume } e]\!]^{\dagger}(m) \vDash P$. Substitutions $P[e/x]$ must be defined for basic assertions and satisfy $m \vDash P[e/x]$ implies $[\![x := e]\!]^{\dagger}(m) \vDash P$.

[4]Disjunctions are defined $\varphi \vee \psi$ iff $\neg(\neg\varphi \wedge \neg\psi)$, a standard encoding in classical logic.

Generic Rules

$$\frac{}{\langle\varphi\rangle\ \mathbb{0}\ \langle\top^{\oplus}\rangle}\textsc{Zero} \qquad \frac{}{\langle\varphi\rangle\ \mathbb{1}\ \langle\varphi\rangle}\textsc{One} \qquad \frac{\langle\varphi\rangle\ C_1\ \langle\psi\rangle \qquad \langle\psi\rangle\ C_2\ \langle\vartheta\rangle}{\langle\varphi\rangle\ C_1 \fatsemi C_2\ \langle\vartheta\rangle}\textsc{Seq} \qquad \frac{\forall i \in \mathbb{N}.\ \langle\varphi_i\rangle\ C\ \langle\varphi_{i+1}\rangle}{\langle\varphi_0\rangle\ \text{for } N \text{ do } C\ \langle\varphi_N\rangle}\textsc{For}$$

$$\frac{\langle\varphi_1\rangle\ C\ \langle\psi_1\rangle \quad \langle\varphi_2\rangle\ C\ \langle\psi_2\rangle}{\langle\varphi_1 \oplus \varphi_2\rangle\ C\ \langle\psi_1 \oplus \psi_2\rangle}\textsc{Split} \qquad \frac{\varphi' \Rightarrow \varphi \quad \langle\varphi\rangle\ C\ \langle\psi\rangle \quad \psi \Rightarrow \psi'}{\langle\varphi'\rangle\ C\ \langle\psi'\rangle}\textsc{Consequence}$$

$$\frac{}{\langle\top^{\oplus}\rangle\ C\ \langle\top^{\oplus}\rangle}\textsc{Empty} \qquad \frac{}{\langle\varphi\rangle\ C\ \langle\top\rangle}\textsc{True} \qquad \frac{}{\langle\bot\rangle\ C\ \langle\varphi\rangle}\textsc{False}$$

Nondeterministic Rules

$$\frac{\langle\varphi\rangle\ C_1\ \langle\psi_1\rangle \qquad \langle\varphi\rangle\ C_2\ \langle\psi_2\rangle}{\langle\varphi\rangle\ C_1 + C_2\ \langle\psi_1 \oplus \psi_2\rangle}\textsc{Plus} \qquad \frac{\langle\varphi\rangle\ \mathbb{1} + C \fatsemi C^{\star}\ \langle\psi\rangle}{\langle\varphi\rangle\ C^{\star}\ \langle\psi\rangle}\textsc{Induction}$$

Expression-Based Rules

$$\frac{}{\langle P[e/x]\rangle\ x := e\ \langle P\rangle}\textsc{Assign} \qquad \frac{P_1 \vDash e \qquad P_2 \vDash \neg e}{\langle P_1 \oplus P_2\rangle\ \text{assume } e\ \langle P_1\rangle}\textsc{Assume}$$

$$\frac{P_1 \vDash e \qquad \langle P_1\rangle\ C_1\ \langle Q_1\rangle \qquad P_2 \vDash \neg e \qquad \langle P_2\rangle\ C_2\ \langle Q_2\rangle}{\langle P_1 \oplus P_2\rangle\ \text{if } e \text{ then } C_1 \text{ else } C_2\ \langle Q_1 \oplus Q_2\rangle}\textsc{If (Multi-Outcome)}$$

Fig. 4. Inference rules that are valid for a variety of OL instantiations. The metavariables φ, ψ refer to arbitrary outcome assertions and P, Q refer to atomic (single-outcome) assertions.

The Assign rule uses *weakest-precondition* style backwards substitution. Assume uses expression entailment to annihilate the outcome where the guard is false. Similarly, If (Multi-Outcome) uses entailment to map entire outcomes to the true or false branches of an if statement, respectively.

All the rules in Figure 4 are sound (see Appendix F for details of the proof).

Theorem 4.6 (Soundness of Proof System). *If* $\vdash \langle P\rangle\ C\ \langle Q\rangle$ *then* $\vDash \langle P\rangle\ C\ \langle Q\rangle$

Note that it is not possible to have generic loop-invariant based iteration rules that are valid for *all* instances of Outcome Logic. This is because loop invariants assume a *partial* correctness specification; they do not guarantee termination. Outcome Logic—in some instantiations—guarantees reachability of end states and therefore must witness a *terminating* program execution. This is in line with the Backwards Variant rule from Incorrectness Logic [O'Hearn 2019, Fig.2], the While rule from Reverse Hoare Logic [de Vries and Koutavas 2011, Fig.2], and Loop Variants from Total Hoare Logic [Apt 1981]. Such a rule for GCL is available in Appendix G.

5 MODELING CORRECTNESS AND INCORRECTNESS VIA OUTCOMES

Incorrectness Logic was motivated in large part by its ability to *disprove* correctness specifications (*i.e.,* Hoare Triples) [Möller et al. 2021, Thm 4.1]. In this section, we prove that OL can also disprove correctness specifications. Not only can OL disprove the same specifications as IL (Corollary 5.7), but it can also express strictly *more* types of incorrectness. Theorem 5.6 shows three classes of bugs that can be characterized in OL for nondeterministic programs, only one of which is expressible in IL. Section 5.2 shows that OL can express probabilistic incorrectness too, whereas IL cannot.

Our first result is stated in terms of *semantic* triples in which the pre- and postconditions are *semantic* assertions (which we denote with uppercase Greek metavariables $\Phi, \Psi \in 2^{M\Sigma}$) rather than the *syntactic* assertions $\varphi, \psi \in$ Prop we have seen thus far. The advantage of this approach is

that we can show the power of the OL model without worrying about the expressiveness of the syntactic assertion language. (As a point of reference, the formal development of Incorrectness Logic is purely semantic [O'Hearn 2019; Möller et al. 2021; Le et al. 2022], as was the metatheory for separation logic [Calcagno et al. 2007; Yang 2001].)

The following Falsification theorem states that any false OL triple can be disproven within OL. Since we already know that OL subsumes Hoare Logic (Theorem 4.5), it follows that any correctness property that is expressible in Hoare Logic can be disproven using OL. We use $\vDash_S \langle \Phi \rangle\ C\ \langle \Psi \rangle$ to denote a valid semantic OL triple, that is: if $m \in \Phi$, then $[\![C]\!]^\dagger(m) \in \Psi$. The assertion $\mathrm{sat}(\Phi)$ means that Φ is satisfiable, in other words $\Phi \neq \emptyset$.

Theorem 5.1 (Semantic Falsification). *For any OL instance and any program C and semantic assertions Φ, Ψ:*

$$\nvDash_S \langle \Phi \rangle\ C\ \langle \Psi \rangle \qquad \textit{iff} \qquad \exists \Phi'.\ \textit{such that}\ \ \Phi' \Rightarrow \Phi,\ \ \mathrm{sat}(\Phi'),\ \ \textit{and}\ \ \vDash_S \langle \Phi' \rangle\ C\ \langle \neg \Psi \rangle$$

Proof. We provide a proof sketch here, whereas the full proof is given Appendix D.1. If $\nvDash_S \langle \Phi \rangle\ C\ \langle \Psi \rangle$, then there must be an $m \in \Phi$ such that $[\![C]\!]^\dagger(m) \notin \Psi$. Choosing $\Phi' = \{m\}$ gives us $\vDash_S \langle \Phi' \rangle\ C\ \langle \neg \Psi \rangle$. For the reverse direction, we know from $\mathrm{sat}(\Phi')$ that there is an $m \in \Phi'$ and from $\Phi' \Rightarrow \Phi$, we know that $m \in \Phi$ and from $\vDash_S \langle \Phi \rangle\ C\ \langle \neg \Psi \rangle$, we know that $[\![C]\!]^\dagger(m) \notin \Psi$, so $\nvDash_S \langle \Phi \rangle\ C\ \langle \Psi \rangle$. □

The full proof of this theorem and formulation of semantic triples are given in Appendix D.1. While this result shows the power of the OL *model*, we also seek to answer whether the outcome assertion syntax given in Definition 4.1 can express the pre- and postconditions needed to disprove other triples. We answer this question in the affirmative, although the forward direction of the result has to be proven separately for nondeterministic and probabilistic models. While the semantic proof above applies to *any* OL instance, the syntactic versions that we present in Section 5.1 and Section 5.2 rely on additional properties of the specific OL instance. Despite the added complexity, we deem this worthwhile since syntactic descriptions give us a characterizations that can be used in the design of automated bug-finding tools.

The reverse direction of Theorem 5.1 corresponds to O'Hearn's [2019] Principle of Denial, though the original Principle of Denial used two triple types (IL and Hoare) and now we only need to use one (OL). We can prove a syntactic version of The Principle of Denial for OL, which can be thought of as a generalization of the *true positives* property, since it tells us when an OL triple (denoting a bug) disproves another OL triple (denoting correctness).

Theorem 5.2 (Principle of Denial). *For any OL instance and any program C and syntactic assertions φ, φ', and ψ:*

$$\textit{If}\quad \varphi' \Rightarrow \varphi,\quad \mathrm{sat}(\varphi'),\quad \textit{and}\quad \vDash \langle \varphi' \rangle\ C\ \langle \neg \psi \rangle\quad \textit{then}\quad \nvDash \langle \varphi \rangle\ C\ \langle \psi \rangle$$

The proof of this theorem is a consequence of Theorem 5.1, together with a result stating how to translate syntactic triples to equivalent semantic ones (Lemma D.1).

Proving a syntactic version of the forward direction of Theorem 5.1 is more complicated—it requires us to witness the existence of a syntactic assertion corresponding to Φ'. The way in which this assertion is constructed depends on several properties of the OL instance. One additional requirement is that the program C must terminate after finitely many steps, otherwise the precondition may not be finitely expressible. This is a common issue when generating preconditions and as a result many developments choose to work with semantic assertions rather than syntactic ones [Kaminski 2019]. The IL falsification results are also only given semantically [O'Hearn 2019; Möller et al. 2021], which avoids infinitary assertions in loop cases.

In the following sections, we will investigate falsification in both nondeterministic and probabilistic OL instances. In doing so, we will provide more specific falsification theorems which both deal

with syntactic assertions and more precisely characterize the ways in which particular programs can be incorrect. While we have just seen that we can obtain a falsification witness for correctness specifications $\langle \varphi \rangle\ C\ \langle \psi \rangle$ by negating the postcondition, proving a triple with postcondition $\neg\psi$ may not be convenient. For example, if ψ is a sequence of outcomes $Q_1 \oplus \cdots \oplus Q_n$, then it is not immediately clear what $\neg\psi$ expresses. We therefore provide more intuitive assertions for canonical types of incorrectness encountered in programs.

5.1 Falsification in Nondeterministic Programs

In this section, we explore falsification for nondeterministic programs. The first step is to formally define a nondeterministic instance of OL by defining an evaluation model and BI frame.

Definition 5.3 (Nondeterministic Evaluation Model). A nondeterministic evaluation model based on program states $\sigma \in \Sigma$ is $\langle \mathbb{2}^{\Sigma}, \mathsf{bind}, \mathsf{unit}, \cup, \emptyset \rangle$ where $\langle \mathbb{2}^{(-)}, \mathsf{bind}, \mathsf{unit} \rangle$ is the powerset monad:

$$\mathsf{bind}(S, k) \triangleq \bigcup_{x \in S} k(x) \qquad\qquad \mathsf{unit}(x) \triangleq \{x\}$$

Definition 5.4 (Nondeterministic Outcome Assertions). Given some satisfaction relation on program states $\vDash_{\Sigma} \subseteq \Sigma \times \mathsf{Prop}$, we create an instance of the outcome assertion logic (Definition 4.1) with the BI frame $\langle \mathbb{2}^{\Sigma}, \cup, =, \emptyset \rangle$ such that atomic assertions come from Prop and are satisfied as follows:

$$S \vDash P \qquad \text{iff} \qquad S \neq \emptyset \quad \text{and} \quad \forall \sigma \in S.\ \sigma \vDash_{\Sigma} P$$

We impose one additional requirement, that the atomic assertions $Q \in \mathsf{Prop}$ can be logically negated[5], which we will denote $\overline{Q}$. Now, we return to the question of how to falsify a sequence of nondeterministic outcomes $Q_1 \oplus \cdots \oplus Q_n$. Lemma 5.5 shows that there are exactly three ways that this assertion can be false.

Lemma 5.5 (Falsifying Assertions). *For any $S \in \mathbb{2}^{\Sigma}$ and atomic assertions $Q_1, \ldots, Q_n$,*

$$S \nvDash Q_1 \oplus \cdots \oplus Q_n \qquad \textit{iff} \qquad \exists i.\ S \vDash \overline{Q}_i \quad \textit{or} \quad S \vDash (\overline{Q}_1 \wedge \cdots \wedge \overline{Q}_n) \oplus \top \quad \textit{or} \quad S \vDash \top^{\oplus}$$

If we take $Q_1 \oplus \cdots \oplus Q_n$ to represent a desirable set of program outcomes, then Lemma 5.5 tells us that said program can be wrong in exactly three ways. Either there is some desirable outcome (Q_i) that the program never reaches, there is some undesirable outcome ($\overline{Q}_1 \wedge \cdots \wedge \overline{Q}_n$) that the program sometimes reaches, or there is an input that causes it to diverge ($\top^{\oplus}$). Now, following from this result, we can state what it means to falsify a nondeterministic specification:

Theorem 5.6 (Nondeterministic Falsification). *For any OL instance based on the nondeterministic evaluation model (Definition 5.3) and outcome assertions (Definition 5.4), $\nvDash \langle \varphi \rangle\ C\ \langle \bigoplus_{i=1}^{n} Q_i \rangle$ iff:*

$$\exists \varphi' \Rightarrow \varphi.\ \mathsf{sat}(\varphi') \ \textit{ and } \ \exists i.\ \vDash \langle \varphi' \rangle\ C\ \langle \overline{Q}_i \rangle \ \textit{ or } \ \vDash^{\downarrow} \langle \varphi' \rangle\ C\ \langle \bigwedge_{i=1}^{n} \overline{Q}_i \rangle \ \textit{ or } \ \vDash \langle \varphi' \rangle\ C\ \langle \top^{\oplus} \rangle$$

The type of bugs expressible in Incorrectness Logic are a special case of Theorem 5.6. Since IL is under-approximate, it can only express the second kind of bug (reachability of a bad outcome), not the first (non-reachability of a good outcome), or last (divergence). IL was motivated by its ability to disprove Hoare Triples—since Hoare Triples are a special case of OL (Theorem 4.5), Theorem 5.6 suggests that OL can disprove Hoare Triples as well. We make this correspondence explicit in the following Corollary where, compared to Theorem 5.6, the first two cases collapse since there is only a single outcome and the divergence case no longer represents a bug since the Hoare Triple is a partial correctness specification.

[5] Crucially, $\overline{Q}$ is not the same as $\neg Q$ (where $\neg$ is from BI) since $S \vDash \neg Q$ iff $S = \emptyset$ or $\exists \sigma \in S. \sigma \nvDash_{\Sigma} Q$ whereas $S \vDash \overline{Q}$ iff $S \neq \emptyset$ and $\forall \sigma \in S.\ \sigma \nvDash_{\Sigma} Q$.

Corollary 5.7 (Hoare Logic Falsification).

$$\not\vDash \{P\}\ C\ \{Q\} \qquad \textit{iff} \qquad \exists \varphi \Rightarrow P.\ \ \mathsf{sat}(\varphi) \quad \textit{and} \quad \vDash^{\downarrow} \langle \varphi \rangle\ C\ \langle \overline{Q} \rangle$$

So, although we do show that OL *semantically* subsumes Incorrectness Logic, it does have the ability to express the same bugs as IL. OL can also disprove more complex correctness properties, such as that of the shuffle function that we saw in Section 2.2. As we will now see, another OL instance is capable of disproving probabilistic properties too.

5.2 Falsification in Probabilistic Programs

Before we can define falsification in a probabilistic setting, we must establish some preliminary definitions. Probabilistic programs use an execution model based on probability (sub)distributions. A (sub)distribution $\mu \in \mathcal{D}X$ over a set X is a function mapping elements $x \in X$ to probabilities in $[0, 1] \subset \mathbb{R}$. The support of a distribution is the set of elements having nonzero probability $\mathsf{supp}(\mu) = \{x \mid \mu(x) > 0\}$ and the mass of a distribution is $|\mu| = \sum_{x \in \mathsf{supp}(\mu)} \mu(x)$. A valid distribution must have mass at most 1. The empty distribution $\varnothing$ maps everything to probability 0 and distributions can be summed pointwise $\mu_1 + \mu_2 = \lambda x.\mu_1(x) + \mu_2(x)$ if $|\mu_1| + |\mu_2| \leq 1$. For any countable set X, $\langle \mathcal{D}X, +, \varnothing \rangle$ is a PCM. In addition, distributions can be weighted by scalars $p \cdot \mu = \lambda x.p \cdot \mu(x)$ if $p \cdot |\mu| \leq 1$ (this is always defined if $p \leq 1$). The Dirac distribution δ_x assigns probability 1 to x and 0 to everything else. We complete the definition of a probabilistic execution model:

Definition 5.8 (Probabilistic Evaluation Model). A probabilistic evaluation model based on program states Σ is defined as $\langle \mathcal{D}\Sigma, \mathsf{bind}, \mathsf{unit}, +, \varnothing \rangle$ where $\langle \mathcal{D}, \mathsf{bind}, \mathsf{unit} \rangle$ is the Giry [1982] monad:

$$\mathsf{bind}(\mu, k) = \sum_{x \in \mathsf{supp}(\mu)} \mu(x) \cdot k(x) \qquad\qquad \mathsf{unit}(x) = \delta_x$$

We can make our imperative language probabilistic by adding a command $x \xleftarrow{\$} \eta$ for sampling from finitely supported probability distributions $\eta \in \mathcal{D}\mathsf{Val}$ over program values. This command is intended to be added to an existing language such as GCL (Example 3.4) or mGCL (Section 6.4). The program semantics and atomic assertions are based on distributions over program states $\mu \in \mathcal{D}\Sigma$. The semantics for the sampling command is defined in terms of variable assignment. This allows us to abstract over the type of program states.

$$c ::= x \xleftarrow{\$} \eta \qquad\qquad [\![x \xleftarrow{\$} \eta]\!]\,(\sigma) = \mathsf{bind}(\eta, \lambda v.\ [\![x := v]\!]\,(\sigma))$$

Definition 5.9 (Probabilistic Outcome Assertions). Given some satisfaction relation on program states $\vDash_\Sigma\ \subseteq \Sigma \times \mathsf{Prop}$, we instantiate the outcome assertion logic (Definition 4.1) with the BI frame $\langle \mathcal{D}\Sigma, +, =, \varnothing \rangle$ such that atomic assertions have the form $\mathbb{P}[A] = p$ where $p \in [0, 1]$, $A \in \mathsf{Prop}$, and:

$$\mu \vDash \mathbb{P}[A] = p \qquad \text{iff} \qquad |\mu| = p \quad \text{and} \quad \forall \sigma \in \mathsf{supp}(\mu).\ \sigma \vDash_\Sigma A$$

Intuitively, the assertion $\mathbb{P}[A] = p$ states that the outcome A occurs with probability p. As a shorthand for under-approximate assertions, we also define $\mathbb{P}[A] \geq p$ to be $(\mathbb{P}[A] = p) \oplus \top$ (see Lemma B.5 for a semantic justification).

We will now investigate falsification of probabilistic assertions of the form $\bigoplus_{i=1}^{n} (\mathbb{P}[A_i] = p_i)$. In general, any such sequence can be falsified by specifying the precise probabilities of all combinations of the outcomes A_i. In the special case where $n = 2$, $\mu \nvDash (\mathbb{P}[A] = p \oplus \mathbb{P}[B] = q)$ iff:

$$\mu \quad \vDash \quad \mathbb{P}[A \wedge B] = p_1 \quad \oplus \quad \mathbb{P}[A \wedge \neg B] = p_2 \quad \oplus \quad \mathbb{P}[\neg A \wedge B] = p_3 \quad \oplus \quad \mathbb{P}[\neg A \wedge \neg B] = p_4$$

Such that $p_4 > 0$ or $p_2 > p$ or $p_3 > q$ or $p_1 + p_2 + p_3 \neq p + q$. The more general version of this result with n outcomes is proven in Lemmas D.9 and D.12. Following from this result, it takes 2^n outcomes to disprove an assertion with n outcomes, which is infeasible for large n. However, there

are several special cases that require many fewer outcomes. For example, if all the A_is are pairwise disjoint, then falsification can be achieved with just $n+1$ outcomes. Below, we use $\vec{q}$ to denote a vector of probabilities $q_1, \ldots, q_n$.

Theorem 5.10 (Disjoint Falsification). *First, let $A_0 = \bigwedge_{i=1}^{n} \neg A_i$. If all the events are disjoint (for all $i \neq j$, $A_i \wedge A_j$ iff* false*), then:*

$$\not\vDash \langle \varphi \rangle\ C\ \langle \bigoplus_{i=1}^{n} (\mathbb{P}[A_i] = p_i) \rangle \qquad \text{iff} \qquad \exists \vec{q}, \varphi' \Rightarrow \varphi. \quad \vDash \langle \varphi' \rangle\ C\ \langle \bigoplus_{i=0}^{n} (\mathbb{P}[A_i] = q_i) \rangle$$

Such that $\mathsf{sat}(\varphi')$ *and* $q_0 \neq 0$ *or for some* i $q_i \neq p_i$.

Many specifications fall into this disjointness case. The primary way in which proofs split into multiple probabilistic outcomes is via sampling, which always splits the postcondition into disjoint outcomes since each outcome corresponds to the sampled variable x taking on a unique value.

The correctness of some probabilistic programs are specified using lower bounds. For example, we may want to specify that some good outcome occurs *with high probability*. These assertions can also be falsified using a lower bound.

Theorem 5.11 (Principle of Denial for Lower Bounds).

$$\text{If} \quad \exists \varphi' \Rightarrow \varphi. \quad \mathsf{sat}(\varphi'), \quad \vDash \langle \varphi' \rangle\ C\ \langle \mathbb{P}[\neg A] \geq q \rangle \quad \text{then} \quad \not\vDash \langle \varphi \rangle\ C\ \langle \mathbb{P}[A] \geq p \rangle \quad (\text{where } q > 1 - p)$$

Note that this implication only goes one way, since the original specification $\mathbb{P}[A] \geq p$ could be satisfied by a sub-distribution μ where $|\mu| < 1$ and therefore $(\mathbb{P}_\mu[A] \not\geq p) \not\Rightarrow (\mathbb{P}_\mu[\neg A] > 1 - p)$. There are many more special cases for probabilistic falsification, but the relevant cases for the purposes of this paper fall into the categories discussed above.

6 OUTCOME LOGIC FOR MEMORY ERRORS

In this section we specialize OL to prove the existence of memory errors in nondeterministic programs. The program logic is constructed in four layers. First, at its core, there is an assertion logic for describing heaps in the style of separation logic (Section 6.1). On top of that, we build an assertion logic with the capability of describing error states and multiple outcomes (Section 6.2). Then, we define the execution model using a monad combining both errors and nondeterminism (Section 6.3). Finally, we provide proof rules for this multi-layered logic (Section 6.4).

We use this logic in Section 6.5 to reason about memory errors in the style of Incorrectness Separation Logic [Raad et al. 2020]. We also discuss why the semantics of Outcome Logic is a good fit for this type of bug finding by examining manifest errors in more depth (Section 6.6).

6.1 Heap Assertions

First, we create a syntax of logical assertions to describe the heap in the style of Separation Logic [Reynolds 2002]. In order to describe why a program crashed, we need *negative heap assertions* in addition to the standard points-to predicates. These assertions, denoted $x \not\mapsto$, state that the pointer x is invalidated [Raad et al. 2020]. The syntax for the heap assertion logic is below.

$$p \in \mathsf{SL} ::= \mathbf{emp} \mid \exists x.p \mid p \wedge q \mid p \vee q \mid p \Rightarrow q \mid p * q \mid p \mathrel{-\!\!*} q \mid e \mid e_1 \mapsto e_2 \mid e \mapsto - \mid e \not\mapsto \qquad (7)$$

In this syntax $e \in \mathsf{Exp}$ is a program expression which includes true and false. We add logical negation $\neg p$ as shorthand for $p \Rightarrow \text{false}$. These assertions are satisfied by a stack and heap pair $(s, h) \in \mathcal{S} \times \mathcal{H}$. Stacks are defined as before (Example 3.4) and heaps are partial functions from positive natural numbers (addresses) to program values or bottom $\mathcal{H} = \{h \mid h \colon \mathbb{N}^+ \rightharpoonup \mathsf{Val} + \{\bot\}\}$[6].

[6]Note that $\ell \notin \mathrm{dom}(h)$ indicates that we have no information about the pointer ℓ whereas $h(\ell) = \bot$ indicates that ℓ is deallocated. This is why h is both partial *and* includes $\bot$ in the co-domain.

The constant null is equal to 0, so it is not a valid address and therefore for any heap h, $\mathsf{null} \notin \mathsf{dom}(h)$. The semantics of SL is defined in Appendix E.1 and is similar to that of Raad et al. [2020].

6.2 Reasoning about Errors

While most formulations of Hoare Logic focus only on safe states, descriptions of error states are a fundamental part of Incorrectness Logic [O'Hearn 2019]. Reasoning about errors is built into the semantics of incorrectness triples and the underlying programming languages. In the style of Incorrectness Logic, we use $(\mathsf{ok} : p)$ and $(\mathsf{er} : p)$ to indicate whether or not the program terminated successfully. In our formulation, however, these are regular assertions rather than part of the triples themselves. This makes our assertion logic more expressive because we can describe programs that have multiple outcomes—some of which are successful and some erroneous—in a single triple. The semantics of programs that may crash is also encoded as a monadic effect.

Definition 6.1 (Assertion logic with errors). Given a set of error states E, a set of program states Σ, and the relations $\vDash_E \subseteq E \times \mathsf{Prop}_E$ and $\vDash_\Sigma \subseteq \Sigma \times \mathsf{Prop}_\Sigma$, we construct a new assertion logic with semantics $\vDash \subseteq (E + \Sigma) \times (\mathsf{Prop}_\Sigma \times \mathsf{Prop}_E)$ defined below:

$$\mathbb{i}_L(e) \vDash (p, q) \quad \text{iff} \quad e \vDash_E q \qquad\qquad \mathbb{i}_R(\sigma) \vDash (p, q) \quad \text{iff} \quad \sigma \vDash_\Sigma p$$

In the above, let $\mathbb{i}_L : E \longrightarrow E + \Sigma$ and $\mathbb{i}_R : \Sigma \longrightarrow E + \Sigma$ be the left and right injections, respectively. We also add syntactic sugar $(\mathsf{ok} : p) \triangleq (p, \mathsf{false})$ and $(\mathsf{er} : q) \triangleq (\mathsf{false}, q)$, so in general the assertion (p, q) can be thought of as $(\mathsf{ok} : p) \vee (\mathsf{er} : q)$. Additional logical operations ($\neg$, $\wedge$, and $\vee$) are defined in Appendix E.2. We now combine errors with separation logic to instantiate the following:

Definition 6.2 (Separation Logic with Errors). We define an assertion logic as follows:

- The syntax of basic assertions Prop is given in Definition 6.1 with $\mathsf{Prop}_E = \mathsf{Prop}_\Sigma = \mathsf{SL}$, the heap assertion logic (7). So, Prop has the syntax $(\mathsf{ok} : p)$ and $(\mathsf{er} : q)$ where $p, q \in \mathsf{SL}$.
- Σ, the set of program states, is given by $\mathcal{S} \times \mathcal{H}$.
- The satisfaction relation is also given in Definition 6.1 with $E = \Sigma$, so $\vDash \subseteq (\Sigma + \Sigma) \times \mathsf{Prop}$.

6.3 Execution Model

The execution model supports both nondeterminism and errors. We achieve this by combining the powerset monad (Definition 5.3) with an error monad. We begin by defining the error monad, which is based on taking a coproduct with a set E of errors. In order to use errors in conjunction with another effect (*i.e.,* nondeterminism), we define a monad transformer [Liang et al. 1995]. This is valid since the error monad composes with all other monads [Lüth and Ghani 2002].

Definition 6.3 (Execution model with errors). Given some execution model $\langle M, \mathsf{bind}_M, \mathsf{unit}_M, \diamond, \varnothing \rangle$, we define a new execution model $\langle M(E + -), \mathsf{bind}_{\mathsf{er}}, \mathsf{unit}_{\mathsf{er}}, \diamond, \varnothing \rangle$ such that:

$$\mathsf{bind}_{\mathsf{er}}(m, k) = \mathsf{bind}_M \left(m, \lambda x. \begin{cases} k(y) & \text{if } x = \mathbb{i}_R(y) \\ \mathsf{unit}_M(x) & \text{if } x = \mathbb{i}_L(y) \end{cases} \right) \qquad \mathsf{unit}_{\mathsf{er}}(x) = \mathsf{unit}_M(\mathbb{i}_R(x))$$

Note the monoid definition ($\diamond$ and $\varnothing$) remains the same as the original execution model. For example, if the outer monad is powerset, we still use set union and empty set in the same way—errors only exist within a single outcome.

Example 6.4 (Execution model for nondeterminism and errors). We are particularly interested in the above definition when M is the powerset monad, *i.e.,* $\mathcal{P}^{(-)}$. This results in an execution model $\langle \mathcal{P}^{E+-}, \mathsf{bind}, \mathsf{unit}, \diamond, \varnothing \rangle$ where the operations are derived as follows:

$$\mathsf{bind}(S, k) = \{\mathbb{i}_L(x) \mid \mathbb{i}_L(x) \in S\} \cup \bigcup_{\mathbb{i}_R(x) \in S} k(x) \qquad \mathsf{unit}(x) = \{\mathbb{i}_R(x)\}$$

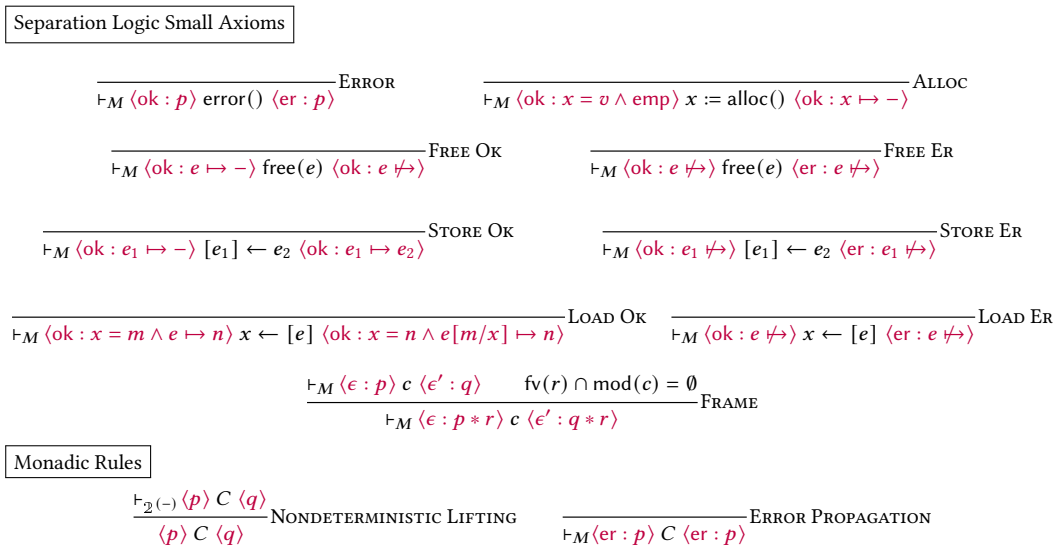


Fig. 5. Proof rules for Outcome-Based Separation Logic, the OL instantiation of Definition 6.5. The first group is inspired by O'Hearn et al.'s [2001] small axioms with additional rules added for unsafe states. The second group deal with the monadic execution model.

We now turn to defining atomic commands for manipulating the heap in a language called the Guarded Command Language with Memory (mGCL). The syntax for mGCL is given below and the semantics is in Appendix E.3. Note that mGCL commands are deterministic and can therefore be interpreted in both nondeterministic and probabilistic evaluation models.

$$c \in \mathsf{mGCL} ::= \mathsf{assume}\ e \mid x := e \mid x := \mathsf{alloc}() \mid \mathsf{free}(e) \mid x \leftarrow [e] \mid [e_1] \leftarrow e_2 \mid \mathsf{error}()$$

Assume and assignment are the same as in GCL (Example 3.4). The usual heap operations for allocation (alloc), deallocation (free), loads ($x \leftarrow [e]$), and stores ($[e_1] \leftarrow e_2$) are also included along with an error command that immediately fails. We also define $x := \mathsf{malloc}()$ as syntactic sugar for $(x := \mathsf{alloc}()) + (x := \mathsf{null})$, which is valid in nondeterministic evaluation models.

Definition 6.5 (Outcome-Based Separation Logic). We instantiate OL (Definition 4.4) with:

(1) The execution model is from Example 6.4 with $E = \mathcal{S} \times \mathcal{H}$.
(2) The language of atomic commands is mGCL.
(3) The assertion logic is the one given in Definition 5.4 using Definition 6.2 for basic assertions.

Note that although the execution model has been augmented with errors, the nondeterministic falsification result (Theorem 5.6) still holds for Outcome-Based Separation Logic.

6.4 Proof Rules for Memory Errors

Now that we have defined the semantics of OL triples that can express properties about memory and errors, let us turn to the proof theory. In this section, we will define proof rules for Outcome-Based Separation Logic (Definition 6.5), which we will use in subsequent sections to prove that programs crash due to memory errors. We will define these proof rules in a way that is *generic* with respect to the execution model, leveraging the fact that the atomic mGCL commands are deterministic, and thus can be given specifications that hold good under multiple different execution models (*e.g.,* nondeterminism or probabilistic computation).

Concretely, let us observe that the semantics of mGCL is based on the composition of two monads: an outer monad M (*e.g.,* powerset), and the error monad $E + -$. Since the atomic commands of mGCL are deterministic, however, their semantics is agnostic to the choice of the outer monad M, and can be specified axiomatically without needing to talk explicitly about (multiple) outcomes. Hence, we define a new type of triple that is capable of making assertions about errors (using er and ok), but says nothing about outcomes (using $\oplus$):

Definition 6.6 (Liftable Triples). Consider an OL instance based on the composition of two monads $M = M_1 \circ M_2$, and so $[\![C]\!] : \Sigma \to (M_1 \circ M_2)\Sigma$ (note that any monad can be decomposed in this way, by taking $M_2 = \mathsf{Id}$). One such example is the execution model from Example 6.4 where $M_1 = \mathbb{2}^{(-)}$ and $M_2 = E + -$. The validity of an OL triple that is *liftable* into the monad M_1 is defined as follows:

$$\vDash_{M_1} \langle p \rangle\ C\ \langle q \rangle \quad \text{iff} \quad \forall \sigma \in M_2\Sigma.\quad \sigma \vDash p \implies \exists \tau \in M_2\Sigma.\ [\![C]\!]^\dagger(\mathsf{unit}_{M_1}(\sigma)) = \mathsf{unit}_{M_1}(\tau) \text{ and } \tau \vDash q$$

Intuitively, this triple says that C is deterministic; if we run it on any individual state that satisfies p, then the result will be an individual state satisfying q. In the case of Example 6.4, this means that p and q describe elements of $E + \Sigma$; they can describe error states (using er and ok), but cannot use $\oplus$. Similarly, we write $\vdash_{M_1} \langle p \rangle\ C\ \langle q \rangle$ to denote a liftable derivation, which is sound with respect to the above semantics and can be lifted into the monad M_1.

Figure 5 contains the proof rules for Outcome-Based Separation Logic (Definition 6.5). The first group of rules is very close to the standard separation logic proof system originally due to O'Hearn et al. [2001], with the addition of rules to reason about unsafe states inspired by Raad et al. [2020]. These rules are liftable into any monad M (since errors compose with all other monads). The Lifting proof rule states that if some triple is liftable into the powerset monad (where p and q are satisfied by individual states), then we can obtain a new triple where p and q are satisfied by sets of states (as in Definition 5.4). All of the small axioms above can be lifted in this way.

In order to use the proof rules for conditionals and assignment from Figure 4, we also define expression entailment and substitution. Both operations are only defined for ok assertions.

$$(\mathsf{ok} : p) \vDash e \quad \text{iff} \quad p \Rightarrow e \qquad\qquad (\mathsf{ok} : p)[e/x] \triangleq \mathsf{ok} : p[e/x]$$

This means that, for example, the Assign rule only allows us to prove $\langle \mathsf{ok} : p[e/x] \rangle\ x := e\ \langle \mathsf{ok} : p \rangle$. If an error has occurred, we instead use the Error Propagation rule to propagate the error forward through the proof (*i.e.,* $\langle \mathsf{er} : p \rangle\ x := e\ \langle \mathsf{er} : p \rangle$), since the program will never recover from the crash.

6.5 Proof of a Bug

We now demonstrate that the OL proof system shown in Figure 5 is effective for bug-finding. The program in Figure 6 has a possible use-after-free error. This program first appeared as a motivating example for ISL [Raad et al. 2020]. It models a common error in C++ when using the `std::vector` library. A call to `push_back` can reallocate the vector's underlying memory buffer, in which case pointers to that buffer become invalid.

As in Raad et al. [2020], we model the vector as a single heap location, and the `push_back` function nondeterministically chooses to either reallocate the buffer or do nothing. A subsequent memory access may then fail, as seen in the main function. Since our language does not have procedures, we model these as macros and prove the existence of the bug with all the code inlined. The proof mostly makes use of standard separation logic proof rules and is quite similar to the ISL version [Raad et al. 2020] especially in the use of negative heap assertion after the call to free. Under-approximation is achieved using the rule of consequence to drop one of the outcomes.

Correctness for this program would be given by the postcondition $(\mathsf{ok} : v \mapsto x * x \mapsto 1)$. As Theorem 5.6 showed, we can disprove it by showing that an undesirable outcome will sometimes

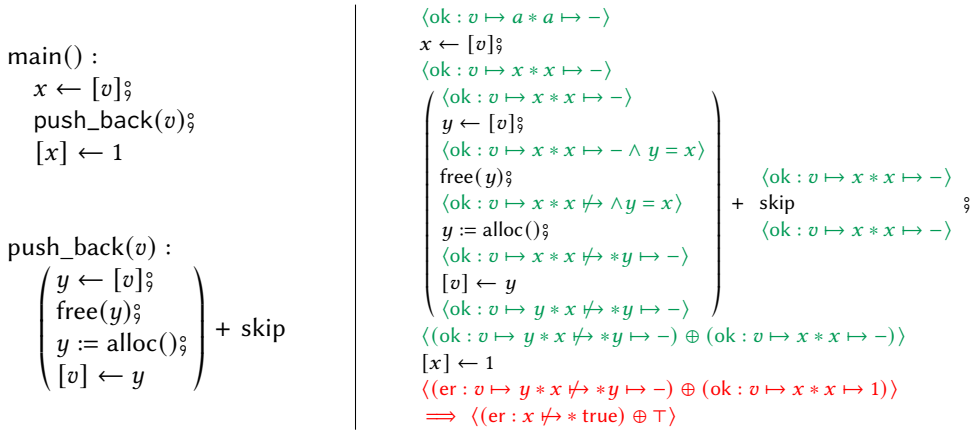


Fig. 6. Program with a possible use-after-free error (left) and proof sketch (right)

occur. In this case, that undesirable outcome is $(\mathsf{er} : x \not\mapsto *\mathsf{true})$. Clearly, $(\mathsf{er} : x \not\mapsto *\mathsf{true}) \oplus \top$ implies $\neg(\mathsf{ok} : v \mapsto x * x \mapsto 1)$, so the specification in Figure 6 disproves the correctness specification.

6.6 Manifest Errors

Le et al. [2022] showed empirically that the fix rates of bug-finding tools can be improved by reporting only those bugs that occur regardless of context. These errors are known as *manifest errors*, as demonstrated in the examples below.

$$\vDash \langle \mathsf{ok} : x \not\mapsto \rangle\ [x] \leftarrow 1\ \langle \mathsf{er} : x \not\mapsto \rangle \qquad \vDash \langle \mathsf{ok} : \mathsf{true} \rangle\ x := \mathsf{malloc}()\ ⨟\ [x] \leftarrow 1\ \langle \mathsf{er} : x = \mathsf{null} \rangle$$

The left program has a *latent* error since it is only triggered if the pointer is already deallocated, therefore it would not be reported. The right program has a *manifest* error since it is possible no matter the context in which the program is invoked. Le et al. [2022, Def. 3.2] give the following definition for manifest errors:

$$\vDash [\mathsf{ok} : p]\ C\ [\mathsf{er} : q] \quad \text{is a manifest error} \qquad \text{iff} \qquad \forall \sigma.\ \ \exists \tau \in [\![C]\!](\sigma).\ \ \tau \vDash (\mathsf{er} : q * \mathsf{true})$$

First note that the precondition p does not appear in the formal definition. This indicates that IL preconditions do not meaningfully describe the conditions *sufficient* to reach an end state. In addition, the universal quantification over the precondition resembles Hoare Logic more closely than Incorrectness Logic (which quantified over the postcondition). As stated in the following lemma, the formal definition of a manifest error can be expressed as an OL triple.

Lemma 6.7 (Manifest Error Characterization).

$$\vDash [p]\ C\ [\mathsf{er} : q] \quad \textit{is a manifest error} \qquad \textit{iff} \qquad \vDash^{\downarrow} \langle \mathsf{ok} : \mathsf{true} \rangle\ C\ \langle \mathsf{er} : q * \mathsf{true} \rangle$$

Following from this result, determining whether a program has a manifest error is equivalent to proving an OL triple of the form above. Characterizing a manifest error using IL is much harder. Le et al. [2022] provide an algorithm to do so, which involves several satisfiability checks (which are NP-hard). The difficulty in characterizing manifest errors suggests that under-approximation in IL is too powerful. To see this, we compare the standard If rule from OL to One-Sided If—a hallmark of IL which allows the analysis to only consider one branch of an if statement.

$$\frac{\langle \mathsf{ok} : p \wedge e \rangle\ C_1\ \langle \epsilon : q \rangle \qquad \langle \mathsf{ok} : p \wedge \neg e \rangle\ C_2\ \langle \epsilon : q \rangle}{\langle \mathsf{ok} : p \rangle\ \mathsf{if}\ e\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \langle \epsilon : q \rangle}\ \textsc{If} \qquad \frac{[p \wedge e]\ C_1\ [\epsilon : q]}{[p]\ \mathsf{if}\ e\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ [\epsilon : q]}\ \textsc{One-Sided If}$$

$$\frac{\vdash_{\mathcal{D}} \langle A \rangle\ C\ \langle B \rangle}{\langle \mathbb{P}[A] = p \rangle\ C\ \langle \mathbb{P}[B] = p \rangle}\ \textsc{Lifting} \qquad \frac{\forall v \in \mathrm{supp}(\eta).\ \langle A \rangle\ x := v\ \langle B_v \rangle}{\langle \mathbb{P}[A] = p \rangle\ x \xleftarrow{\$} \eta\ \langle \bigoplus_{v \in \mathrm{supp}(\eta)} (\mathbb{P}[B_v] = p \cdot \eta(v)) \rangle}\ \textsc{Sample}$$

Fig. 7. Probabilistic proof rules.

One-Sided If generates imprecise preconditions since the precondition of the premise ($p \wedge e$) is stronger than the precondition of the conclusion (p). OL, on the other hand, requires the precondition to be precise enough to force the execution down a specific logical path, otherwise both paths must be considered as seen in the If rule. As such, OL enables under-approximation in *just the right ways*; only outcomes that result from nondeterministic choice can be dropped.

Le et al.'s [2022] discussion of manifest errors suggests that *sufficient* preconditions are important; we need to know what happens when we run the program on *any* state satisfying the precondition. Interestingly, there is no analogous motivation for covering the whole postcondition (as IL does). Reachability is important, but we only have to reach *some* error state, not *all* of them.

7 PROBABILISTIC INCORRECTNESS

Randomization is a powerful tool that is seeing increased adoption in mainstream software development as it is essential for machine learning and security applications. The study of probabilistic programming has a rich history [Kozen 1979, 1983], but there is little prior work on proving that probabilistic programs are *incorrect*. In Section 5.2, we gave a theoretical result showing that probabilistic specifications in OL can be disproven. In this section, we provide a proof system for probabilistic OL and use it to prove incorrectness in a particular example program.

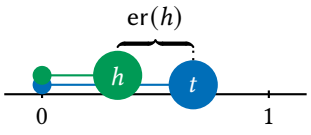
We work with the probabilistic OL instance using the evaluation model from Definition 5.8 and the outcome assertions in Definition 5.9. The basic commands are assignment and assume from GCL (Example 3.4) with probabilistic sampling added ($x \xleftarrow{\$} \eta$). There are only two proof rules for the probabilistic language, given in Figure 7. The Lifting rule allows us to lift a derivation (e.g. for variable assignment) into a probabilistic setting. This is sound, since every state in the support must transition from A to B, thus $\mathbb{P}[A]$ before running C is equal to $\mathbb{P}[B]$ after. The Sample rule splits the postcondition into a separate outcome for each value in the support of η. Lemma F.6 proves the soundness of these rules. The rules for conditional branching in Figure 4 can be used in probabilistic proofs by defining expression entailment ($\mathbb{P}[A] = p) \vDash e$ iff $A \vDash e$. Assign can also be used; substitution propagates inside the probabilistic assertion $(\mathbb{P}[A] = p)[e/x] = (\mathbb{P}[A[e/x]] = p)$. Note that the conditional rules require us to know the probability that the guard is true or false upfront. This is standard for probabilistic Hoare Logics [Barthe et al. 2018; den Hartog 2002].

Absent are rules for while loops. Looping rules in probabilistic languages are complex since invariants cannot be used when probabilities change across iterations. Such proof rules are certainly expressible in our model, but are out of scope for this paper. For examples of how this is done, see Barthe et al. [2018]; den Hartog [2002]; Rand and Zdancewic [2015].

7.1 Error Bounds for Machine Learning

Randomization is often used in approximation algorithms where computing the exact solution to a problem is difficult. In these applications, some amount of error is acceptable as long as it is likely to be small. One such application is supervised learning algorithms, which produce a *hypothesis* from a set of labelled examples. The examples are members of some set X and are drawn randomly from some probability distribution $\eta \in \mathcal{D}X$. The hypothesis is a function $h : X \to \mathbb{B}$ which guesses whether new data points are positive or negative examples.

Consider the simple learning problem in which we want to learn a point $t \in [0, 1] \subset \mathbb{R}$ on the unit interval. Since we require distributions used in programs to be finite, we can approximate



$h := -1^{\S}_{\S}$
for N do
  $x \xleftarrow{\$} [0,1]^{\S}_{\S}$
  if $\mathcal{L}(x) \wedge x > h$ then
    $h := x$
  else
    skip

$\langle \mathbb{P}[\text{true}] = 1 \rangle \implies \langle \mathbb{P}[\text{er}(-1) > \varepsilon] \geq (1-\varepsilon)^0 \rangle$
$h := -1^{\S}_{\S}$
$\langle \mathbb{P}[\text{er}(h) > \varepsilon] \geq (1-\varepsilon)^0 \rangle$
for N do
  $\langle \mathbb{P}[\text{er}(h) > \varepsilon] \geq (1-\varepsilon)^i \rangle$
  $x \xleftarrow{\$} [0,1]^{\S}_{\S}$
  $\langle \bigoplus_{q \in [0,1]} \mathbb{P}[\text{er}(h) > \varepsilon \wedge x = q] \geq \Delta \cdot (1-\varepsilon)^i \rangle$
  if $\mathcal{L}(x) \wedge x > h$ then
    $\langle \bigoplus_{q \in (h,t]} \mathbb{P}[\text{er}(h) > \varepsilon \wedge x = q] \geq \Delta \cdot (1-\varepsilon)^i \rangle$
    $\implies \langle \bigoplus_{x \in (h,t-\varepsilon]} \mathbb{P}[\text{er}(x) > \varepsilon] \geq \Delta \cdot (1-\varepsilon)^i \rangle$
    $\implies \langle \mathbb{P}[\text{er}(x) > \varepsilon] \geq (t-\varepsilon-h) \cdot (1-\varepsilon)^i \rangle$
    $h := x$
    $\langle \mathbb{P}[\text{er}(h) > \varepsilon] \geq (t-\varepsilon-h) \cdot (1-\varepsilon)^i \rangle$
  else
    $\langle \bigoplus_{q \in [0,h] \cup (t,1]} \mathbb{P}[\text{er}(h) > \varepsilon \wedge x = q] \geq \Delta \cdot (1-\varepsilon)^i \rangle$
    skip$^{\S}_{\S}$
    $\langle \bigoplus_{q \in [0,h] \cup (t,1]} \mathbb{P}[\text{er}(h) > \varepsilon \wedge x = q] \geq \Delta \cdot (1-\varepsilon)^i \rangle$
    $\implies \langle \mathbb{P}[\text{er}(h) > \varepsilon] \geq (h+1-t) \cdot (1-\varepsilon)^i \rangle$
  $\langle \mathbb{P}[\text{er}(h) > \varepsilon] \geq (t-\varepsilon-h) \cdot (1-\varepsilon)^i \oplus \mathbb{P}[\text{er}(h) > \varepsilon] \geq (h+1-t) \cdot (1-\varepsilon)^i \rangle$
  $\implies \langle \mathbb{P}[\text{er}(h) > \varepsilon] \geq (1-\varepsilon)^{i+1} \rangle$
$\langle \mathbb{P}[\text{er}(h) > \varepsilon] \geq (1-\varepsilon)^N \rangle$

Fig. 8. The interval learning problem: a diagram of the learning problem (top left), a program implementing interval learning (bottom left), and a proof sketch (right).

$[0,1]$ as $\{k \cdot \Delta \mid 0 \leq k \leq \frac{1}{\Delta}\}$ for some finite *step size* Δ. Anything in the interval $[0,t]$ is considered a positive example, and anything greater than t is a negative example. This concept is illustrated at the top of Figure 8 and the program below–expressed in a probabilistic extension of GCL–learns this concept by repeatedly sampling examples and refining the hypothesis h after each round. The resulting hypothesis is always equal to the largest positive example that the algorithm has seen. Therefore it will always classify negative examples correctly and only make mistakes on positive examples between h and t.

The *labelling* oracle $\mathcal{L}(x) = x \leq t$ gives the true label of any point on the interval. Let $\text{er}(h) = t - h$ be the error of the hypothesis (the total probability mass between h and t). The goal is to determine the probability that h has error greater than ε after N iterations. Practically speaking, this simulates training the model on a dataset of size N. Intuitively, the error will be less than ε if the algorithm ever samples an example in the interval $[t-\varepsilon, t]$. The chance of getting a hit in this range increases greatly with the number of examples seen. While this problem may seem contrived, it is a 1-dimensional version of the *Rectangle Learning Problem* which is known to have practical applications and the proof ideas are extensible to other learnable concepts [Kearns and Vazirani 1994].

To prove that this program is *correct*, we want to say that the resulting hypothesis has small error with high probability. Choosing an error bound ε and a confidence parameter δ, we say that the program is correct if at the end $\mathbb{P}[\text{er}(h) \leq \varepsilon] \geq 1 - \delta$. Now, we can look to Theorem 5.11 to determine how to *disprove* the correctness specification. We need to show that the probability of the opposite happening ($\text{er}(h) > \varepsilon$) is higher than δ. Based on the derivation in Figure 8, we conclude that the program is incorrect if $(1-\varepsilon)^N > \delta$. Suppose we had a dataset of size $N = 100$ and desired at most 1% error ($\varepsilon = 0.01$) with 90% likelihood ($\delta = 0.1$). Then the postcondition tells us that the error is higher than 1% with probability at least 37%. Clearly $37\% > \delta$, so the program is incorrect; we need a larger dataset in order to get a better result.

7.2 Probabilistic Incorrectness Logic

It is natural to ask whether a similar result could be achieved using a probabilistic variant of Incorrectness Logic. However, such a program logic is cumbersome and produces poor characterizations

of errors. To show this, we begin by examining the semantics of a probabilistic IL triple.

$$\vDash [P]\ C\ [Q] \qquad \text{iff} \qquad \forall \mu \vDash Q.\quad \exists \mu'.\quad \mu \sqsubseteq [\![C]\!]^{\dagger}(\mu') \quad \text{and} \quad \mu' \vDash P$$

This definition differs from standard Incorrectness Logic in two ways. First, assertions are satisfied by *distributions* over program states $\mu \in \mathcal{D}\Sigma$ rather than individual program states $\sigma \in \Sigma$. This is necessary in order to make the assertion logic quantitative. Second, under-approximation is achieved using the sub-distribution relation $\sqsubseteq$ instead of set inclusion[7]. As is typical with Incorrectness Logic, this definition stipulates that *any* subdistribution satisfying the postcondition must be reachable by an execution of the program. While in non-probabilistic cases it can already be hard to fully characterize a valid end-state, even more information is needed in the probabilistic case.

To demonstrate this, consider the interval learning program from Figure 8. The postcondition of this triple is $\mathbb{P}[\mathsf{er}(h) > \varepsilon] \geq (1-\varepsilon)^N$. This is not a valid postcondition for an incorrectness triple because it does not adequately describe the resulting distribution of the program. That is, there are many distributions satisfying this assertion that could not result from running the program. In one such distribution, $h = -1$ with probability 1. So, lower bounds are not suitable for use in Incorrectness Logic because a distribution can be invented where the probability is arbitrarily large, rendering it unreachable. But changing the inequality to an equality to obtain $\mathbb{P}[\mathsf{er}(h) > \varepsilon] = (1-\varepsilon)^N$ does not solve the problem. This assertion can be satisfied by a distribution where $h = -1$ with probability $(1-\varepsilon)^N$, which is also unreachable. In order for an assertion to properly characterize the output distribution, it has to specify all the possible values of h. Such an assertion is given below:

$$\bigoplus_{x=0}^{\varepsilon} \left(\mathbb{P}[\mathsf{er}(h) = x] = (1-x)^N - (1-(x+\Delta))^N \right)$$

The original assertion was easy to understand, we immediately knew the probability of having a large error. By contrast, the added information needed for the Incorrectness Logic version actually obscures the result. It is not useful to know the probability of each value of h, we only care about bounding the probability that $\mathsf{er}(h) > \varepsilon$. In general, Probabilistic Incorrectness Logic requires us to specify the *entire joint distribution* over all the program variables which is certainly undesirable and often infeasible.

Many techniques in probabilistic program analysis summarize the output distribution in alternative ways. This includes using expected values [Morgan et al. 1996; Kaminski 2019] and probabilistic independence [Barthe et al. 2019]. If those techniques are used to express correctness, it makes sense that similar ideas would be desirable for incorrectness. However, techniques that *summarize* a distribution are incompatible with Incorrectness Logic since they do not specify the output distribution in a sufficient level of detail. Based on these findings, we conclude that developing probabilistic variants of Incorrectness Logic is not a promising research direction. In fact, the differences between correctness and incorrectness are often quite blurred in probabilistic examples. Since some amount of error is typically expected, it is not possible to reason about correctness *without* reasoning about incorrectness. It is therefore sensible that a unified theory captures both.

8 RELATED WORK

Incorrectness reasoning and program analysis. In motivating Incorrectness Logic (IL), O'Hearn [2019] posed the twin challenges of *sound* and *scalable* incorrectness reasoning: program logics for incorrectness must guarantee *true positive* bugs, while also supporting *under-approximation* in order to scale to large codebases. Outcome Logic (OL) takes inspiration from those challenges,

[7] This order is defined pointwise: $\mu_1 \sqsubseteq \mu_2$ iff $\forall x. \mu_1(x) \leq \mu_2(x)$.

but offers a solution that is closer to traditional Hoare Logic [Hoare 1969] and, as such, is also compatible with correctness reasoning.

Outcome Logic was also inspired in part by *Lisbon triples*, which were first described in a published article by Möller et al. [2021, §5] under the name *backwards under-approximate triples*.[8] The semantics of Lisbon triples is based on Hoare's [1978] calculus of *possible correctness*: for any initial state satisfying the precondition, there exists *some* trace of execution leading to a final state satisfying the postcondition. As such, Lisbon triples describe true positives (behaviors that are witnessed by an actual trace, assuming the pre is satisfiable). As recounted by O'Hearn [2019, §7], Lisbon triples predate Incorrectness Logic; Derek Dreyer and Ralf Jung suggested them as a foundation for incorrectness reasoning during a discussion with Peter O'Hearn and Jules Villard that took place at POPL'19 in Lisbon (hence the name "Lisbon Triples").

Shortly thereafter, O'Hearn developed the semantics of IL triples. His major motivation for developing IL (instead of further exploring Lisbon triples) was the goal of finding a logical foundation for *scalable* bug-catching static analysis tools (such as Pulse-X [Le et al. 2022]), and one key to scalability is the ability to discard program paths (aka "drop disjuncts") during analysis. More concretely, the analysis accumulates a disjunction of assertions which symbolically represents the set of possible states at each program point. If this set gets too large, then it is important to be able to drop some of the disjuncts in order to save memory and computation time. Thanks to its reverse rule of consequence—which supports *strengthening* of the postcondition—IL provides a sound logical foundation for dropping disjuncts, whereas Lisbon triples do not.

One can see OL as a generalization of Lisbon triples which supports discarding of program paths in a different way than IL does: namely, via the *outcome conjunction* connective, which lets one reason about multiple executions at the same time.[9] Specifically, "disjuncts" arise in a program analysis when the program makes a *choice* to branch based on either a logic condition (*e.g.,* an if statement or while loop) or a computational effect (*e.g.,* nondeterminism or randomization). In IL, both types of choice are encoded by standard disjunction. In OL, on the other hand, we distinguish these two forms of choice by using disjunction ($\vee$) for the former and outcome conjunction ($\oplus$) for the latter. This leads to a different approach for supporting discarding of program paths, but one which we believe can serve as an alternative logical foundation for practical static analyses.

Let us first consider the case of choices arising from computational effects. Incorrectness Logic includes a Choice rule that allows analyses to drop one branch of a nondeterministic choice. An analogous derived rule is also sound in Outcome Logic (see Appendix B.2); both are shown below.

$$\frac{[P]\ C_1\ [Q]}{[P]\ C_1 + C_2\ [Q]}\textsc{Choice (IL)} \qquad \frac{\langle P\rangle\ C_1\ \langle Q\rangle}{\langle P\rangle\ C_1 + C_2\ \langle Q \oplus \top\rangle}\textsc{Under-Approx (OL)}$$

Given that nondeterministic variants of OL provide reachability guarantees, it may appear surprising that a conclusion about $C_1 + C_2$ can be made without showing that C_2 terminates. However, the assertion $\top$ encompasses all outcomes (including nontermination), so this inference is valid. Note that there are also symmetric versions of these rules where the C_2 branch is instead taken.

Let us now consider the case of choices arising from logical conditions, where the differences between OL and IL are more pronounced. Consider the following program, which will only fail in the case that b is true.

$$\langle \mathsf{ok} : x \mapsto - \rangle\ \mathsf{if}\ b\ \mathsf{then}\ \mathsf{free}(x)\ \mathsf{else}\ \mathsf{skip}\ ;\ [x] \leftarrow 1\ \langle (\mathsf{ok} : x \mapsto 1) \vee (\mathsf{er} : x \not\mapsto)\rangle$$

The semantics of OL does not permit us to simply drop one of the disjuncts in the postcondition. If we want to only explore the program path in which the error occurs, then we need to push

[8] Though Le et al. [2022, §3.2] also mention backwards under-approximate triples, their potential has gone largely unexplored.
[9] In Appendix C, we show that Lisbon triples are in fact a special case of OL.

information about the logical condition b backwards into the precondition.

$$\langle \mathsf{ok} : x \mapsto - \wedge b \rangle \text{ if } b \text{ then } \mathsf{free}(x) \text{ else skip} \fatsemi [x] \leftarrow 1 \langle \mathsf{er} : x \not\mapsto \rangle$$

This is in contrast to Incorrectness Logic, in which we *can* drop disjuncts, *but in return* we need to ensure that every state described by the postcondition is reachable. More precisely, $(\mathsf{er} : x \not\mapsto)$ is not a strong enough IL postcondition for the aforementioned program because it includes the unreachable state in which $x \not\mapsto$, but b is false. In IL, one must therefore specify the bug as follows:

$$[x \mapsto -] \text{ if } b \text{ then } \mathsf{free}(x) \text{ else skip} \fatsemi [x] \leftarrow 1 [\mathsf{er} : x \not\mapsto \wedge b]$$

So, in either case we must record the same amount of information about the logical condition b. The difference is whether this information appears in the pre- or postcondition. As we discussed in Section 6.6, there are advantages to having a more precise precondition (as OL does): it enables us to easily determine how to trigger a bug and characterize manifest errors. Conversely, the precise postconditions required by IL make it difficult to design abstract domains, suggesting that IL is not compatible with popular analysis techniques like abstract interpretation [Ascari et al. 2022].

Furthermore, in order to generate more useful bug reports and error traces for the user, practical static analysis tools like Pulse-X [Le et al. 2022] *do* in any case push logical conditions backwards to the pre-condition using a technique called bi-abduction [Calcagno et al. 2011]. This suggests that while the theories of OL and IL differ substantially, it may be possible to build practical static analysis tools atop OL in a similar manner to IL-based tools like Pulse-X. We plan to investigate this further in future work.

Unifying correctness and incorrectness. Parallel efforts have been made to unify correctness and incorrectness reasoning within a single program logic. Bruni et al. [2021, 2023] created a program logic based on Incorrectness Logic, but with limits on the rule of consequence such that an over-approximation of the reachable states can always be recovered from the postcondition. Similarly, Exact Separation Logic (ESL) [Maksimović et al. 2022] combines the semantics of IL and Hoare Logic in triples that exactly describe the reachable states.

Both of these logics are capable of proving correctness properties as well as finding true bugs. But they achieve this by compromising the ability to use the rule of consequence, which is crucial to scalable analysis algorithms. Analyses based on Hoare Logic use consequences to abstract the postcondition, reducing the information overhead and aiding in finding loop invariants. Analyses based on IL use consequences to drop disjuncts and consider fewer program paths. Since neither type of consequence is valid in the logics of Bruni et al. [2021] and ESL, it remains unclear whether those theories can feasibly serve as the foundation of practical tools. By contrast, Outcome Logic enjoys the full power of the rule of consequence and can also drop nondeterministic paths.

There has also been work to connect the theories of correctness and incorrectness algebraically using Kleene Algebra with Tests (KAT) [Kozen 1997], an equational theory for reasoning about program equivalence. Möller et al. [2021]; Zhang et al. [2022] showed that both Hoare Logic and IL can be embedded in variants of KAT and used this insight to formalize connections between the two types of specifications. While this provides an algebraic theory powerful enough to capture Hoare Logic and IL, this connection does not go as deep as the unification offered by OL and does not provide a clear path to shared analyses for both program verification and bug finding.

Since our paper was conditionally accepted to OOPSLA, a very closely related paper has appeared on arXiv, which presents a program logic, called Hyper Hoare Logic, for proving and disproving program hyper-properties (properties relating multiple program traces) [Dardinier and Müller 2023]. It achieves this using the same underlying semantics as Outcome Logic instantiated to the powerset monad. Their work shows the applicability of the OL model beyond the usage scenarios that we envisioned in this paper.

Separation logic and Iris. While both Separation Logic [Reynolds 2002] and Outcome Logic employ Bunched Implications [O'Hearn and Pym 1999] as a fundamental part of their metatheories, the way in which BI is used in each case is substantially different.

In Separation Logic and its extensions such as Iris [Jung et al. 2015, 2018], the value of the BI resource monoid is neatly demonstrated by the Frame Rule, which enables local reasoning by adding assertions about unused resources to the pre- and post-conditions of some smaller proof derivation. In this way, framing allows us to talk about the same program execution with additional (unused) resources. By contrast, the outcome conjunction deals with assertions about *different* program executions.

The Frame Rule is in general unsound with respect to the outcome conjunction. To demonstrate this, we use the same counterexample that Reynolds [2002] used to demonstrate that the Rule of Constancy is unsound in separation logic:

$$\frac{\langle x \mapsto - \rangle\ [x] \leftarrow 4\ \langle x \mapsto 4 \rangle}{\langle x \mapsto - \oplus y \mapsto 3 \rangle\ [x] \leftarrow 4\ \langle x \mapsto 4 \oplus y \mapsto 3 \rangle}\textsc{Frame}$$

It is easy to see that this is an invalid inference. The outcome conjunction does not preclude that x and y are aliased, in which case it must be that $y \mapsto 4$ in the postcondition. Instead, we have the Split rule (Figure 4), which allows us to analyze a program separately for each outcome in the precondition and then compose the resulting outcomes in the postcondition.

This example shows that, although both separation logic and OL use BI, the two logics are modeling two very different aspects of the program (resource usage vs. program outcomes, respectively), and the resulting program logics are therefore different.

OL and separation logic are not mutually exclusive. In Section 6, we saw how separation logic can be embedded in OL. In addition, we believe that combining OL with Iris is a very interesting direction for future research: Iris offers advanced mechanisms to reason modularly about concurrency, and OL offers a way to extend Hoare Logic in a way that is amenable to both correctness and incorrectness reasoning. Combining the two would result in a program logic capable of proving the existence of bugs in concurrent programs (while a concurrent version of Incorrectness Logic already exists [Raad et al. 2022], it is not built atop Iris and does not support the full capabilities offered by Iris).

In a concurrent version of Outcome Logic, outcomes would model possible interleavings of concurrent branches. In an assertion of the form $P \oplus \top$, the predicate P could describe an undesirable outcome that occurs in *some* of those interleavings (*i.e.,* a bug), which is not currently possible to express in Iris.

Probabilistic and quantitative program analysis. Probabilistic variants of Hoare Logic [Barthe et al. 2018; den Hartog 2002; Rand and Zdancewic 2015; Tassarotti and Harper 2019] were a major source of inspiration for the design of Outcome Logic. Whereas pre- and postconditions of standard Hoare Logic describe individual program states, probabilistic variants of Hoare Logic use assertions that describe *distributions* over program states. These logics also include connectives similar to the outcome conjunction, but specialized to probability distributions. In Outcome Logic, we generalize from probability distributions to support a wider variety of PCMs.

Starting with the seminal work of Kozen [1979, 1983], expected values have been a favorite choice for probabilistic program analysis. Morgan et al. [1996]'s weakest-pre-expectation (wpe) calculus computes expected values of program expressions with an approach similar to Dijkstra's [1976] Weakest Precondition calculus. Many extensions to wpe have arisen, including to handle nondeterminism, runtimes [Kaminski 2019], and Separation Logic [Batz et al. 2019]. This line of work has not intersected with Incorrectness Logic since the semantics of weakest-pre is incompatible with IL, although Batz et al. [2019] hinted at the nuanced interaction between correctness and

incorrectness in quantitative settings with their "faulty garbage collector" example. We hope that our new perspective—using Hoare Logic for incorrectness—will encourage the use of wpe calculi for bug-finding.

Zhang and Kaminski [2022] developed a Quantitative Strongest Post (QSP) calculus and noted its connections to IL, which was originally characterized by O'Hearn [2019] in terms of Dijkstra's [1976] strongest-post. QSP is an interesting foundation for studying the Galois Connections between types of quantitative program specifications, although the goals are somewhat orthogonal to our own in that we sought to *unify* correctness and incorrectness rather than explore dualities.

9 CONCLUSION

Formal methods for incorrectness remain a young field. The foundational work of O'Hearn [2019] has already led to several program logics for proving the existence of bugs such as memory errors, memory leaks, data races, and deadlocks [Raad et al. 2020, 2022; Le et al. 2022]. However, as with any new field there are growing pains—manifest errors and probabilistic programs are an awkward fit in the original formulation of IL. This has inspired us to pursue a new theory incorporating O'Hearn's [2019] core tenets of incorrectness—true positives and under-approximation—while also accounting for more evaluation models and different types of incorrectness. Outcome Logic achieves just that, with the added benefit of unifying the theories of correctness and incorrectness in a single program logic. Our Falsification Theorem (Theorem 5.1) shows that any OL triple can be disproven within the logic—this means that if there is a bug invalidating a correctness specification, it can be found. OL also offers a cleaner characterization of manifest errors, suggesting it may be semantically closer to the way that programmers reason about bugs.

In this paper, we introduced OL as a theoretical basis for incorrectness reasoning, but in the future we plan to further explore its practical potential as well. Incorrectness Logic has been shown to scale well as an underlying theory for bug-finding in large part due to its ability to *drop disjuncts* [Raad et al. 2020; Le et al. 2022]; analysis algorithms accumulate a disjunction of possible outcomes as they move forward through a program, and due to the semantics of IL, these disjuncts can be soundly pruned to keep the search space small. Hoare Logics (including OL) cannot drop disjuncts. However, as we saw in Section 2 and Section 4, OL *can* drop *outcomes*, which we believe is sufficient to make the algorithm scale to large codebases (although this remains to be demonstrated). Furthermore, since OL triples can be used both for correctness and incorrectness reasoning, we plan to develop a bi-abductive [Calcagno et al. 2011] algorithm to infer procedure summaries that can be used by *both* correctness verification and bug-finding analyses.

When O'Hearn [2019] remarked that "program correctness and incorrectness are two sides of the same coin," he was expressing that just as programmers spend significant mental energy debugging (reasoning about *incorrectness*), we in the formal methods community must invent sound reasoning principles for incorrectness. We take this idea one step further, suggesting that program correctness and incorrectness are two *usages* of the same *program logic*. We hope that this unifying perspective will continue to invigorate the field of incorrectness reasoning and invite the reuse of tools and techniques that have already been successfully deployed for correctness reasoning.

ACKNOWLEDGMENTS

We thank Peter O'Hearn, Josh Berdine, Azalea Raad, Jules Villard, Quang Loc Le, and Julien Vanegue for their helpful feedback. This work has been supported in part by the Defense Advanced Research Projects Agency under Contract HR001120C0107.

REFERENCES

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

Flavio Ascari, Roberto Bruni, and Roberta Gori. 2022. Limits and difficulties in the design of under-approximation abstract domains. In *Foundations of Software Science and Computation Structures*, Patricia Bouyer and Lutz Schröder (Eds.). Springer International Publishing, Cham, 21–39. https://doi.org/10.1007/978-3-030-99253-8_2

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

Gilles Barthe, Justin Hsu, and Kevin Liao. 2019. A Probabilistic Separation Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 55 (Dec. 2019), 30 pages. https://doi.org/10.1145/3371123

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

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

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2023. A Correctness and Incorrectness Program Logic. *J. ACM* (feb 2023). https://doi.org/10.1145/3582267 Just Accepted.

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

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

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

Edsko de Vries and Vasileios Koutavas. 2011. Reverse Hoare Logic. In *Software Engineering and Formal Methods*, Gilles Barthe, Alberto Pardo, and Gerardo Schneider (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 155–171. https://doi.org/10.1007/978-3-642-24690-6_12

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

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

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

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

Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. 2016. Probabilistic NetKAT. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*. 282–309. https://doi.org/10.1007/978-3-662-49498-1_12

Michèle Giry. 1982. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, B. Banaschewski (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 68–85. https://doi.org/10.1007/BFb0092872

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

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

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

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

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

Michael J. Kearns and Umesh V. Vazirani. 1994. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, USA.

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

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

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

Dexter Kozen. 2000. On Hoare Logic and Kleene Algebra with Tests. *ACM Trans. Comput. Logic* 1, 1 (Jul 2000), 60–76. https://doi.org/10.1145/343369.343378

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

Sheng Liang, Paul Hudak, and Mark Jones. 1995. Monad Transformers and Modular Interpreters. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL '95)*. Association for Computing Machinery, New York, NY, USA, 333–343. https://doi.org/10.1145/199448.199528

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

Petar Maksimović, Caroline Cronjäger, Julian Sutherland, Andreas Lööw, Sacha-Élie Ayoun, and Philippa Gardner. 2022. Exact Separation Logic. https://doi.org/10.48550/ARXIV.2208.07200

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

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

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

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

Peter W. O'Hearn and David J. Pym. 1999. The Logic of Bunched Implications. *The Bulletin of Symbolic Logic* 5, 2 (1999), 215–244. http://www.jstor.org/stable/421090

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

Benjamin C. Pierce. 1991. *Basic Category Theory for Computer Scientists*. MIT Press. https://doi.org/10.7551/mitpress/1524.001.0001

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

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

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

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

Joseph Tassarotti and Robert Harper. 2019. A Separation Logic for Concurrent Randomized Programs. *Proc. ACM Program. Lang.* 3, POPL, Article 64 (Jan 2019), 30 pages. https://doi.org/10.1145/3290377

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

Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. 2022. On Incorrectness Logic and Kleene Algebra with Top and Tests. *Proc. ACM Program. Lang.* 6, POPL, Article 29 (jan 2022), 30 pages. https://doi.org/10.1145/3498690

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

A TOTALITY OF LANGUAGE SEMANTICS

As mentioned in Section 3, the semantics of the language in Figure 2 can be made total in all the execution models that we use (nondeterministic and probabilistic), despite depending on the partial monoid operator ($\diamond$). In this section we discuss restrictions that must be placed on probabilistic languages in order to make the semantics total and also establish the existence of the least fixed point used in the semantics of $C^\star$.

Regardless of the execution model, proving the fixed point existence requires us to prove that the semantic map $[\![-]\!]^\dagger$ is continuous with respect to some partial order. We remark that a preorder can be generically defined in terms of the monoid operation $m_1 \sqsubseteq m_2$ iff there exists m such that $m_1 \diamond m = m_2$. In both the nondeterministic and probabilistic case, this relation is also anti-symmetric, therefore it is a partial order. In fact, in the case of the powerset monad, $\sqsubseteq$ is equivalent to $\subseteq$.

We also introduce the notion of syntactic validity for a program C. For example, the use of expressions must be well-typed. That is, if assume e appears in the program, then e must be boolean valued, *i.e.*, $\forall \sigma.\, [\![e]\!]\,(\sigma) \in \mathbb{B}$.

A.1 Nondeterministic Languages

Since the monoid operation for nondeterminsitic languages is set union (a total function), we can allow unrestricted access to $C_1 + C_2$ and $C^\star$. Therefore, to ensure totality, we must only prove that the least fixed point exists.

Lemma A.1 (Fixed point existence). *For any semantics of atomic commands, the function $F(f)(\sigma) = f^\dagger([\![C]\!]\,(\sigma)) \diamond \mathsf{unit}(\sigma)$ has a least fixed point when specialized to the powerset monad.*

Proof. We first note that in the lemma statement $f : \Sigma \to 2^\Sigma$ and $[\![C]\!] : \Sigma \to 2^\Sigma$. We also define the point-wise partial order $f_1 \sqsubseteq f_2$ iff $\forall x. f_1(x) \subseteq f_2(x)$. Clearly, the function $\lambda x.\emptyset$ is the bottom of this order. This also means that for any non-empty chain $f_1 \sqsubseteq f_2 \sqsubseteq \ldots$ it must be that $\bigsqcup_i f_i = \lambda x. \bigcup_i f_i(x)$. We now show that F is Scott continuous:

$$
\begin{aligned}
F\left(\bigsqcup_i f_i\right) &= \lambda\sigma.\left(\bigsqcup_i f_i\right)^\dagger([\![C]\!]\,(\sigma)) \cup \{\sigma\} \\
&= \lambda\sigma.\left(\bigcup_{\tau \in [\![C]\!](\sigma)} \left(\bigsqcup_i f_i\right)(\tau)\right) \cup \{\sigma\} \\
&= \lambda\sigma.\bigcup_{\tau \in [\![C]\!](\sigma)} \left(\bigcup_i f_i(\tau) \cup \{\sigma\}\right) \\
&= \lambda\sigma.\bigcup_i \left(f_i^\dagger([\![C]\!]\,(\sigma)) \cup \{\sigma\}\right) \\
&= \lambda\sigma.\bigcup_i F(f_i)(\sigma) \\
&= \bigsqcup_i F(f_i)
\end{aligned}
$$

Therefore, by the Kleene Fixed Point Theorem, $\mathsf{lfp}(F) = \bigsqcup_{n \in \mathbb{N}} F^n(\lambda x.\emptyset)$. □

A.2 Probabilistic Languages

In probabilistic languages, we can ensure totality using simple syntactic checks. That is, we syntactically limit programs to not use $C_1 + C_2$ and $C^\star$, but rather the guarded versions as shown in Example 3.4. In addition, we establish that bind is total. Since bind is implemented as a sum, we

must ensure that the cumulative probability mass of the summands does not exceed 1. This is easy to see:

$$|\mathsf{bind}_{\mathcal{D}}(\mu, f)| = \left| \sum_{\sigma \in \mathsf{supp}(\mu)} \mu(\sigma) \cdot f(\sigma) \right| = \sum_{\sigma \in \mathsf{supp}(\sigma)} \mu(\sigma) \cdot |f(\sigma)| \leq \sum_{\sigma \in \mathsf{supp}(\sigma)} \mu(\sigma) = |\mu|$$

Since $f : \Sigma \to \mathcal{D}\Sigma$, then for any $\sigma \in \Sigma$, $|f(\sigma)| \leq 1$. Therefore, we have shown that $|\mathsf{bind}(\mu, f)| \leq |\mu|$ (bind is contractive) and since it cannot add probability mass it must be total.

Lemma A.2 (Totality of Probabilistic Language Semantics). *The function $[\![C]\!] : \Sigma \to \mathcal{D}(\Sigma)$ is total subject to the syntactic restrictions on C described above.*

Proof. The proof is by induction on C. All of the cases except if statements and while loops trivially follow from the definition of $[\![-]\!]$.

- If. First note that $[\![\text{if } e \text{ then } C_1 \text{ else } C_2]\!](\sigma) = [\![(\text{assume } e \fatsemi C_1) + (\text{assume } \neg e \fatsemi C_2)]\!](\sigma) = [\![\text{assume } e \fatsemi C_1]\!](\sigma) + [\![\text{assume } \neg e \fatsemi C_2]\!](\sigma)$. Now, we do case analysis on the value of $[\![e]\!]_{\mathsf{Exp}}(\sigma)$. If $[\![e]\!]_{\mathsf{Exp}}(\sigma) = \text{true}$, then $[\![\text{assume } e]\!](\sigma) = \delta_\sigma$ and $[\![\text{assume } \neg e]\!](\sigma) = \varnothing$. Therefore, we know that $[\![\text{assume } e \fatsemi C_1]\!](\sigma) = [\![C_1]\!](\sigma)$ and $[\![\text{assume } \neg e \fatsemi C_2]\!](\sigma) = \varnothing$. By the induction hypothesis $[\![C_1]\!](\sigma)$ is defined and so $[\![C_1]\!](\sigma) + \varnothing$ must also be defined. The case where $[\![e]\!]_{\mathsf{Exp}}(\sigma) = \text{false}$ is symmetrical.
- While. We begin by proposing an alternate semantics for (guarded) while loops:

  $$[\![\text{while } e \text{ do } C]\!](\sigma) = \mathsf{lfp}(F)(\sigma) \quad \text{where} \quad F(f)(\sigma) = f^\dagger([\![\text{assume } e \fatsemi C]\!](\sigma)) \diamond [\![\text{assume } \neg e]\!](\sigma)$$

  In this semantics, we push the assume $\neg e$ in to the fixed point computation which allows $\diamond$ to be defined. In the nondeterminism case where $\diamond$ is total, this semantics is equivalent to the one defined in Figure 2. Now, note that when using the partial order described at the beginning of this section, the supremum of two distributions (if it exists) is $\mu_1 \sqcup \mu_2 = \lambda\sigma.\max(\mu_1(\sigma), \mu_2(\sigma))$. We can therefore see that addition distributes over the supremum:

  $$\begin{aligned}(\mu_1 \sqcup \mu_2) + \mu &= (\lambda\sigma.\max(\mu_1(\sigma), \mu_2(\sigma))) + \mu \\ &= \lambda\sigma.\max(\mu_1(\sigma), \mu_2(\sigma))) + \mu(\sigma) \\ &= \lambda\sigma.\max((\mu_1 + \mu)(\sigma), (\mu_2 + \mu)(\sigma))) \\ &= (\mu_1 + \mu) \sqcup (\mu_2 + \mu)\end{aligned}$$

  We now proceed to prove that F is Scott continuous. We use the same point-wise order that we saw in Lemma A.1, $f_1 \sqsubseteq f_2$ iff $\forall x. f_1(x) \sqsubseteq f_2(x)$.

  $$\begin{aligned}F\left(\bigsqcup_i f_i\right) &= \lambda\sigma.\left(\bigsqcup_i f_i\right)^\dagger([\![\text{assume } e \fatsemi C]\!](\sigma)) + [\![\text{assume } \neg e]\!](\sigma) \\ &= \lambda\sigma.\sum_{\tau \in [\![\text{assume } e \fatsemi C]\!](\sigma)} (\textstyle\bigsqcup_i f_i(\tau)) + [\![\text{assume } \neg e]\!](\sigma) \\ &= \lambda\sigma.\bigsqcup_i (f_i^\dagger([\![\text{assume } e \fatsemi C]\!](\sigma)) + [\![\text{assume } \neg e]\!](\sigma))\end{aligned}$$

  Note that this sum is always defined since one of $[\![\text{assume } e]\!](\sigma)$ or $[\![\text{assume } \neg e]\!](\sigma)$ must be $\varnothing$.

  $$\begin{aligned}&= \lambda\sigma.\bigsqcup_i F(f_i)(\sigma) \\ &= \bigsqcup_i F(f_i)\end{aligned}$$

  Therefore, by the Kleene Fixed Point Theorem, $\mathsf{lfp}(F) = \bigsqcup_{n \in \mathbb{N}} F^n(\lambda x.\varnothing)$.

□

B UNDER-APPROXIMATION

In Definition 4.3, we defined under-approximate outcome assertions $m \vDash^{\downarrow} \varphi$ to be syntactic sugar for $m \vDash \varphi \oplus \top$. In order to motivate this choice, we prove the following results, which show that this definition of under-approximation corresponds to *dropping outcomes*.

Lemma B.1 (Dropping Outcomes). *In any BI frame, the following implications hold: $\varphi \oplus \psi \Rightarrow \varphi \oplus \top$ and $\varphi \oplus \psi \Rightarrow \top \oplus \psi$.*

Proof. Suppose that $m \vDash \varphi \oplus \psi$. Then there exists m_1 and m_2 such that $m \succcurlyeq m_1 \diamond m_2$ and $m_1 \vDash \varphi$ and $m_2 \vDash \psi$. Clearly, also $m_2 \vDash \top$, so $m \vDash \varphi \oplus \top$. The second implication is symmetric. □

Lemma B.2 (Dropping Outcomes (Under-Approximate)). *In any BI frame, if $m \vDash^{\downarrow} \varphi \oplus \psi$, then $m \vDash \varphi$.*

Proof. Since $m \vDash^{\downarrow} \varphi \oplus \psi$, then $m \vDash \varphi \oplus \psi \oplus \top$. This means that $m_1 \vDash \varphi$, $m_2 \vDash \psi$ and $m_3 \vDash \top$ such that $m_1 \diamond m_2 \diamond m_3 \preccurlyeq m$. Clearly, $m_2 \vDash \top$ as well. Recombining these, we get $m \vDash \varphi \oplus \top \oplus \top$ which is equivalent to $m \vDash \varphi \oplus \top$, or just $m \vDash^{\downarrow} \varphi$. □

B.1 Alternative Formulation using Intuitionistic BI

In Section 4.1 we defined a single variant of the outcome logic using classical BI with under-approximate assertions as syntactic sugar. A different development is possible using an intuitionistic interpretation of BI with a preorder defined in terms of the monoid composition:

$$m_1 \preccurlyeq m_2 \qquad \text{iff} \qquad \begin{cases} m_2 = \varnothing & \text{if} \quad m_1 = \varnothing \\ \exists m.\ m_1 \diamond m = m_2 & \text{if} \quad m_1 \neq \varnothing \end{cases}$$

Note that the first case ensures that $\varnothing$ is only related to itself, which is necessary to ensure that $m \vDash \top^{\oplus}$ iff $m = \varnothing$. Atomic assertions in intuititionistic BI interpretations must respect the persistence property: if $m \vDash P$ and $m' \succcurlyeq m$, then $m' \vDash P$ (this is also referred to as monotonicity in Kripke semantics). We will now show that the under-approximate satisfaction relation $\vDash^{\downarrow}$is valid as an intuitionistic satisfaction relation for atomic propositions.

Lemma B.3 (Under-Approximate Satisfaction is Persistent). *For any $m, m' \in M\Sigma$ and atomic assertion P, if $m \vDash^{\downarrow} P$ and $m' \succcurlyeq m$, then $m' \vDash^{\downarrow} P$.*

Proof. If $m = \varnothing$, then m' is also $\varnothing$ and so clearly $m' \vDash^{\downarrow} P$. Now suppose that $m \neq \varnothing$. Since $m \vDash^{\downarrow} P$, then $m \vDash P \oplus \top$. Since $m' \succcurlyeq m$ and $m \neq \varnothing$, there is some m'' such that $m \diamond m'' = m'$. Clearly, $m'' \vDash \top$, so $m' \vDash (P \oplus \top) \oplus \top$. This means that $m' \vDash P \oplus \top$, or in other words $m' \vDash^{\downarrow} P$. □

If we combine the under-approximate satisfaction relation with the basic assertions for nondeterministic and probabilistic evaluation models (Definitions 5.4 and 5.9), we get a sensible semantics. As the following two lemmas show, under-approximation in the nondeterministic case corresponds to existential quantification and in the probabilistic case it corresponds to lower bounds.

Lemma B.4. *In the powerset interpretation of BI, $S \vDash^{\downarrow} P$ iff $\exists \sigma \in S.\ \sigma \vDash_{\Sigma} P$.*

Proof.

(⇒) Suppose that $S \vDash^{\downarrow} P$, so $S \vDash P \oplus \top$, or in other words $S_1 \vDash P$ and $S_2 \vDash \top$ such that $S = S_1 \cup S_2$. Further, this means that $S_1 \neq \emptyset$ and $\forall \sigma \in S_1.\ \sigma \vDash_{\Sigma} P$. Since we know S_1 is nonempty, then there exists $\sigma \in S_1.\ \sigma \vDash_{\Sigma} P$ and since $S_1 \subseteq S$, then $\sigma \in S$ as well, so $\exists \sigma \in S.\ \sigma \vDash_{\Sigma} P$.

(⇐) Suppose that $\exists \sigma \in S.\ \sigma \vDash_{\Sigma} P$. Now, let $T = \{\sigma\}$, so clearly $T \vDash P$ and $S \vDash \top$ and $T \cup S = S$. Therefore, $S \vDash P \oplus \top$ and so $S \vDash^{\downarrow} P$.

□

Lemma B.5. *In the distribution interpretation of BI, $\mu \vDash^{\downarrow} (\mathbb{P}[A] = p)$ iff $\mathbb{P}_\mu[A] \geq p$, where:*

$$\mathbb{P}_\mu[A] \triangleq \sum \{\mu(\sigma) \mid \sigma \in \mathsf{supp}(\mu), \sigma \vDash_\Sigma A\}$$

Proof.

($\Rightarrow$) Assume that $\mu \vDash^{\downarrow} (\mathbb{P}[A] = p)$, so $\mu \vDash (\mathbb{P}[A] = p) \oplus \top$. Therefore $\mu_1 \vDash (\mathbb{P}[A] = p)$ and $\mu_2 \vDash \top$ such that $\mu_1 + \mu_2 = \mu$. This tells us that $\mathbb{P}_{\mu_1}[A] = p$. When we add μ_2 to μ_1 to get μ, the probability of A can only increase, so $\mathbb{P}_\mu[A] \geq p$.

($\Leftarrow$) Assume that $\mathbb{P}_\mu[A] \geq p$. That means there must be a sub-distribution μ_1 of μ such that $|\mu_1| = p$ and $\forall \sigma \in \mathsf{supp}(\mu_1).\ \sigma \vDash_\Sigma A$. Let the other part of the distribution be μ_2 (so $\mu = \mu_1 + \mu_2$). Now, by construction, $\mu_1 \vDash (\mathbb{P}[A] = p)$ and $\mu_2 \vDash \top$, so $\mu \vDash (\mathbb{P}[A] = p) \oplus \top$, or equivalently $\mu \vDash^{\downarrow} (\mathbb{P}[A] = p)$.

□

B.2 Derived Under-approximate Proof Rules

In this section, we provide derived inference rules that aid in reasoning about programs in an under-approximate manner. The first set of rules under-approximate nondeterministic choice by only exploring one of the paths and using the trivial post-condition $\top$ for the other path. Note that the unexplored path may diverge, $\top$ is a valid postcondition no matter what behavior it has.

Lemma B.6. *The following proof rules for under-approximating program paths can be derived for any nondeterministic Outcome Logic instance.*

$$\frac{\langle \varphi \rangle\ C_1\ \langle \psi \rangle}{\langle \varphi \rangle\ C_1 + C_2\ \langle \psi \oplus \top \rangle}\ \textit{Under-Approx Left} \qquad \frac{\langle \varphi \rangle\ C_2\ \langle \psi \rangle}{\langle \varphi \rangle\ C_1 + C_2\ \langle \psi \oplus \top \rangle}\ \textit{Under-Approx Right}$$

Proof. We show the derivation for Under-Approx Left below. The derivation of Under-Approx Right is symmetric.

$$\frac{\langle \varphi \rangle\ C_1\ \langle \psi \rangle \qquad \dfrac{}{\langle \varphi \rangle\ C_1\ \langle \top \rangle}\ \textsc{True}}{\langle \varphi \rangle\ C_1 + C_2\ \langle \psi \oplus \top \rangle}\ \textsc{Plus}$$

□

Once an under-approximate $- \oplus \top$ predicate has been introduced, it is also convenient to have inference rules that propagate it forward. The following two derived rules can be used to sequence under-approximate derivations together.

Lemma B.7. *The following inference rules are derivable for any Outcome Logic instance.*

$$\frac{\langle \varphi \rangle\ C\ \langle \psi \rangle}{\langle \varphi \oplus \top \rangle\ C\ \langle \psi \oplus \top \rangle}\ \textit{Under-Approx Prop} \qquad \frac{\langle \varphi \rangle\ C_1\ \langle \psi \oplus \top \rangle \qquad \langle \psi \rangle\ C_2\ \langle \vartheta \rangle}{\langle \varphi \rangle\ C_1 \fatsemi C_2\ \langle \vartheta \oplus \top \rangle}\ \textit{Under-Approx Seq}$$

Proof. These rules are derived as follows:

$$\frac{\langle \varphi \rangle\ C\ \langle \psi \rangle \qquad \dfrac{}{\langle \top \rangle\ C\ \langle \top \rangle}\ \textsc{True}}{\langle \varphi \oplus \top \rangle\ C\ \langle \psi \oplus \top \rangle}\ \textsc{Split} \qquad \frac{\langle \varphi \rangle\ C_1\ \langle \psi \oplus \top \rangle \qquad \dfrac{\langle \psi \rangle\ C_2\ \langle \vartheta \rangle}{\langle \psi \oplus \top \rangle\ C_2\ \langle \vartheta \oplus \top \rangle}\ \textsc{Under-Approx Prop}}{\langle \varphi \rangle\ C_1 \fatsemi C_2\ \langle \vartheta \oplus \top \rangle}\ \textsc{Seq}$$

□

C EQUIVALENCE OF TRIPLES

In this section, we show that the nondeterministic instance of OL subsumes Hoare Logic [Hoare 1969] and the Backward Under-Approximate Triples of Möller et al. [2021] that were mentioned briefly in Section 8. We assume we have a nondeterministic program semantics over program states $[\![C]\!] : \Sigma \to 2^\Sigma$ and an assertion logic where propositions $P, Q \in \text{Prop}$ are satisfied by program states, so $\vDash_\Sigma \subseteq \Sigma \times \text{Prop}$. Both of the aforementioned triple semantics are defined below where under-approximate triples use the notation $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$ due to Le et al. [2022]:

$$\vDash \{P\}\ C\ \{Q\} \quad \text{iff} \quad \forall \sigma \in \Sigma.\ \ \sigma \vDash_\Sigma P \ \Rightarrow\ \forall \tau \in [\![C]\!](\sigma).\ \ \tau \vDash_\Sigma Q$$
$$\vDash \{\!|P|\!\}\ C\ \{\!|Q|\!\} \quad \text{iff} \quad \forall \sigma \in \Sigma.\ \ \sigma \vDash_\Sigma P \ \Rightarrow\ \exists \tau \in [\![C]\!](\sigma).\ \ \tau \vDash_\Sigma Q$$

Now, we will work with nondeterministic instances of OL using the evaluation model from Definition 5.3 and the logic of atomic assertions from Definition 5.4. We let BI disjunctions $\varphi \vee \psi$ be syntactic sugar for $\neg(\neg\varphi \wedge \neg\psi)$ (this encoding is typical in classical logics). We now prove our first result, that Hoare Triples are subsumed by OL. As we mentioned in Section 4.2, since Hoare Triples are partial correctness specification, we have to use the postcondition $Q \vee \top^\oplus$ to express that Q holds *if* the program terminates[10].

Theorem 4.5 (Subsumption of Hoare Triples). $\vDash \{P\}\ C\ \{Q\}$ *iff* $\vDash_{\text{pc}} \langle P\rangle\ C\ \langle Q\rangle$

Proof.

($\Rightarrow$) Suppose that $\vDash \{P\}\ C\ \{Q\}$ or in other words, for any $\sigma \vDash_\Sigma P$ and $\tau \in [\![C]\!](\sigma)$, $\tau \vDash_\Sigma Q$. Now suppose that $S \vDash P$, or in other words, $S \neq \emptyset$ and $\forall \sigma \in S.\ \sigma \vDash_\Sigma P$. Since $\vDash \{P\}\ C\ \{Q\}$, then for any $\tau \in [\![C]\!](\sigma)$, $\tau \vDash_\Sigma Q$. Now, we know that $[\![C]\!]^\dagger(S) = \text{bind}(S, [\![C]\!]) = \bigcup_{\sigma \in S} [\![C]\!](\sigma)$. This means that for every $\tau \in [\![C]\!]^\dagger(S)$, $\tau \vDash_\Sigma Q$. So, if $[\![C]\!]^\dagger(S) \neq \emptyset$, then $[\![C]\!]^\dagger(S) \vDash Q$. If $[\![C]\!]^\dagger(S) = \emptyset$, then $[\![C]\!]^\dagger(S) \vDash \top^\oplus$. Therefore $[\![C]\!]^\dagger(S) \vDash Q \vee \top^\oplus$ and $\vDash \langle P\rangle\ C\ \langle Q \vee \top^\oplus\rangle$.

($\Leftarrow$) Suppose that $\vDash \langle P\rangle\ C\ \langle Q \vee \top^\oplus\rangle$ and so if $S \vDash P$, then $[\![C]\!]^\dagger(S) \vDash Q \vee \top^\oplus$. Now suppose that $\sigma \vDash_\Sigma P$. Then trivially $\{\sigma\} \vDash P$, so we can use our assumption to conclude that $[\![C]\!]^\dagger(\{\sigma\}) \vDash Q \vee \top^\oplus$. This implies that $\forall \tau \in [\![C]\!](\sigma).\ \tau \vDash_\Sigma Q$ (in the case where $[\![C]\!]^\dagger(\{\sigma\}) \vDash \top^\oplus$, then $[\![C]\!](\sigma) = \emptyset$, so it holds vacuously).

□

Now, we will prove that OL triples subsume Backwards Under-Approximate Triples as well. This time, we use the under-approximate variant of OL which transforms the postcondition Q into $Q \oplus \top$. This corresponds to existential quantification as we proved in Lemma B.4.

Theorem C.1 (Subsumption of Under-Approximate Triples). $\vDash \{\!|P|\!\}\ C\ \{\!|Q|\!\}$ *iff* $\vDash^\downarrow \langle P\rangle\ C\ \langle Q\rangle$

Proof.

($\Rightarrow$) Suppose that $\vDash \{\!|P|\!\}\ C\ \{\!|Q|\!\}$ or in other words, for any $\sigma \vDash_\Sigma P$, there exists a $\tau \in [\![C]\!](\sigma)$ such that $\tau \vDash_\Sigma Q$. Now suppose that $S \vDash P$, or in other words $S \neq \emptyset$ and $\forall \sigma \in S.\ \sigma \vDash_\Sigma P$. Pick one such $\sigma \in S$ (there must be at least one since $S \neq \emptyset$). Since $\vDash \{\!|P|\!\}\ C\ \{\!|Q|\!\}$, then $\exists \tau \in [\![C]\!](\sigma)$ such that $\tau \vDash_\Sigma Q$. Since $\sigma \in S$, then $S = \{\sigma\} \cup S$ and therefore by linearity, $[\![C]\!]^\dagger(S) = [\![C]\!]^\dagger(\{\sigma\} \cup S) = [\![C]\!]^\dagger(\{\sigma\}) \cup [\![C]\!]^\dagger(S) = [\![C]\!](\sigma) \cup [\![C]\!]^\dagger(S)$ and since $\tau \in [\![C]\!](\sigma)$ then $\tau \in [\![C]\!]^\dagger(S)$ and so $\exists \tau \in [\![C]\!]^\dagger(S).\ \tau \vDash_\Sigma Q$. By Lemma B.4, we can therefore conclude that $[\![C]\!]^\dagger(S) \vDash Q \oplus \top$.

[10] Equivalent ways of expressing this include $\neg Q \Rightarrow \top^\oplus$ (if Q is false, then the program must diverge) or $\neg\top^\oplus \Rightarrow Q$ (if the program terminates, then Q holds). Alternatively, if we modified the semantics of atomic assertions (Definition 5.4) to be $S \vDash P$ iff $\forall \sigma \in S.\ \sigma \vDash_\Sigma P$ (without requiring that $S \neq \emptyset$), then we would have a more direct correspondence: $\vDash \{P\}\ C\ \{Q\}$ iff $\vDash \langle P\rangle\ C\ \langle Q\rangle$, but then $P \oplus Q$ would behave more like $P \vee Q$, not guaranteeing reachability.

($\Leftarrow$) Suppose that $\vDash \langle P \rangle\ C\ \langle Q \oplus \top \rangle$ and so if $S \vDash P$, then $[\![C]\!]^\dagger(S) \vDash Q \oplus \top$. Now suppose that $\sigma \vDash_\Sigma P$. Then trivially $\{\sigma\} \vDash P$, so we can use our assumption to conclude that $[\![C]\!]^\dagger(\{\sigma\}) \vDash Q \oplus \top$. Now, by Lemma B.4, there is some $\tau \in [\![C]\!]\ (\sigma)$ such that $\tau \vDash_\Sigma Q$.

□

The combination of Theorem C.1 and Corollary 5.7 suggest that Backwards Under-Approximate triples can disprove any Hoare Triple as well (if the precondition φ from Theorem C.1 can be expressed as a basic assertion). Möller et al. [2021] also stated this fact, although the proof was omitted.

D FALSIFICATION

In this section we, prove the falsification results from Section 5 of the main text. These theorems are inspired by that of Möller et al. [2021, Theorem 4.1], who proved that if some Hoare triple is false $\nvDash \{P\}\ C\ \{Q\}$, then there is some other Incorrectness triple $\vDash [P']\ C\ [Q']$ that disproves it.

$$\nvDash \{P\}\ C\ \{Q\} \qquad \text{iff} \qquad \exists P', Q'.\quad P' \Rightarrow P \quad \text{and} \quad Q' \not\Rightarrow Q \quad \text{and} \quad \vDash [P']\ C\ [Q']$$

The proof given by Möller et al. [2021] is *semantic*; it does not witness the construction of P' and Q' as *syntactic* assertions. We give any analogous result in Appendix D.1. Theorem 5.1 proves that any false OL triple (with semantic assertions) can be disproven by another OL triple.

While this result shows the strength of the OL model, we are also interested to know if our *syntactic* assertion logic is powerful enough to express the pre- and postconditions needed to disprove other triples. We answer this question in the affirmative, although the result is less general. While the semantic proof in Appendix D.1 applies to *any* OL instance, the syntactic proofs rely on some additional properties of the particular evaluation model. We lay out the requirements for a falsifiable instance of OL in Appendix D.2 and prove that the nondeterministic and probabilistic instances are falsifiable in Appendices D.3 and D.4 respectively.

D.1 Falsification Proof with Semantic Assertions

We first introduce the notion of a semantic OL triple. A semantic assertion is simply a set of satisfying models. We will use the uppercase greek metavariables to denote semantic assertions $\Phi, \Psi \in 2^{M\Sigma}$. The semantic interpretation of a syntactic assertion is the set of models that satisfies it $(\!|\varphi|\!) \triangleq \{m \mid m \in M\Sigma, m \vDash \varphi\}$. Logical implication $\Phi \Rightarrow \Psi$ is given by set inclusion $\Phi \subseteq \Psi$. Note that $\Phi \Rightarrow \Psi$ is a proposition, *not* a semantic assertion (*i.e.,* it is not a set). Negation is given by $\neg\Phi = 2^{M\Sigma} \setminus \Phi$ and we say that an assertion is satisfiable $\mathrm{sat}(\Phi)$ iff $\Phi \neq \emptyset$. This gives us the following expected properties:

$$(\!|\varphi|\!) \Rightarrow (\!|\psi|\!) \quad \text{iff} \quad \varphi \Rightarrow \psi \qquad\qquad m \in \neg\Phi \quad \text{iff} \quad m \notin \Phi \qquad\qquad \mathrm{sat}((\!|\varphi|\!)) \quad \text{iff} \quad \exists m \in M\Sigma.\ m \vDash \varphi$$

We also define the notion of a semantic OL triple as follows:

$$\vDash_S \langle \Phi \rangle\ C\ \langle \Psi \rangle \qquad \text{iff} \qquad \forall m \in M\Sigma.\quad m \in \Phi \quad \Longrightarrow \quad [\![C]\!]^\dagger(m) \in \Psi$$

The correspondence between semantic and syntactic triples is given by the following lemma.

Lemma D.1 (Equivalence of Semantic and Syntactic triples). *If* $\Phi = (\!|\varphi|\!)$ *and* $\Psi = (\!|\psi|\!)$, *then:*

$$\vDash_S \langle \Phi \rangle\ C\ \langle \Psi \rangle \qquad \textit{iff} \qquad \vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$$

Proof.

($\Rightarrow$) Suppose that $m \vDash \varphi$, then $m \in (\!|\varphi|\!) = \Phi$, so using $\vDash_S \langle \Phi \rangle\ C\ \langle \Psi \rangle$, we know that $[\![C]\!]^\dagger(m) \in \Psi = (\!|\psi|\!)$. This means that $[\![C]\!]^\dagger(m) \vDash \psi$, so $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$.

($\Leftarrow$) Suppose that $m \in \Phi = (\!|\varphi|\!)$, then it must be that $m \vDash \varphi$, so using $\vDash \langle\varphi\rangle\ C\ \langle\psi\rangle$, we can conclude that $[\![C]\!]^\dagger(m) \vDash \psi$. This means that $[\![C]\!]^\dagger(m) \in (\!|\psi|\!) = \Psi$, so therefore $\vDash_S \langle\Phi\rangle\ C\ \langle\Psi\rangle$.

□

We can now prove the Semantic Falsification theorem and the Principle of Denial, which were introduced in Section 5.

Theorem 5.1 (Semantic Falsification). *For any OL instance and any program C and semantic assertions Φ, Ψ:*

$$\nvDash_S \langle\Phi\rangle\ C\ \langle\Psi\rangle \quad \textit{iff} \quad \exists\Phi'.\ \textit{such that}\ \ \Phi' \Rrightarrow \Phi,\ \ \mathrm{sat}(\Phi'),\ \ \textit{and}\ \ \vDash_S \langle\Phi'\rangle\ C\ \langle\neg\Psi\rangle$$

Proof.

($\Rightarrow$) Assume that $\nvDash_S \langle\Phi\rangle\ C\ \langle\Psi\rangle$, so that means that there is some $m \in M\Sigma$ such that $m \in \Phi$ and $[\![C]\!]^\dagger(m) \notin \Psi$. By definition, this also means that $[\![C]\!]^\dagger(m) \in \neg\Psi$. Now, let $\Phi' = \{m\}$, so clearly $\Phi' \Rrightarrow \Phi$ (since $\Phi' \subseteq \Phi$) and $\mathrm{sat}(\Phi')$. To see that $\vDash_S \langle\Phi'\rangle\ C\ \langle\neg\Psi\rangle$, suppose that $m' \in \Phi'$. By construction, it must be that $m' = m$, so therefore $[\![C]\!]^\dagger(m') \in \neg\Psi$ (since we already know that $[\![C]\!]^\dagger(m) \in \neg\Psi$.

($\Leftarrow$) Assume that there is some Φ' such that $\Phi' \Rrightarrow \Phi$, $\mathrm{sat}(\Phi')$, and $\vDash_S \langle\Phi'\rangle\ C\ \langle\neg\Psi\rangle$. Then, there must be some $m \in \Phi'$ and so $m \in \Phi$ as well. Since $\vDash_S \langle\Phi'\rangle\ C\ \langle\neg\Psi\rangle$, then $[\![C]\!]^\dagger(m) \in \neg\Psi$ and so $[\![C]\!]^\dagger(m) \notin \Psi$. We therefore know that $m \in \Phi$ and $[\![C]\!]^\dagger(m) \notin \Psi$, so $\nvDash_S \langle\Phi\rangle\ C\ \langle\Psi\rangle$.

□

Theorem 5.2 (Principle of Denial). *For any OL instance and any program C and syntactic assertions φ, φ', and ψ:*

$$\textit{If}\quad \varphi' \Rrightarrow \varphi,\quad \mathrm{sat}(\varphi'),\quad \textit{and}\quad \vDash \langle\varphi'\rangle\ C\ \langle\neg\psi\rangle\quad \textit{then}\quad \nvDash \langle\varphi\rangle\ C\ \langle\psi\rangle$$

Proof. Let $\Phi' = (\!|\varphi'|\!)$, $\Phi = (\!|\varphi|\!)$, and $\Psi = (\!|\psi|\!)$. From our assumptions, we can conclude that $\Phi' \Rrightarrow \Phi$ and $\mathrm{sat}(\Phi')$ and by Lemma D.1 we can conclude that $\vDash_S \langle\Phi'\rangle\ C\ \langle\neg\Psi\rangle$. Therefore by Theorem 5.1, this implies that $\nvDash_S \langle\Phi\rangle\ C\ \langle\Psi\rangle$. Using Lemma D.1 again, we conclude that $\nvDash \langle\varphi\rangle\ C\ \langle\psi\rangle$.

□

D.2 Falsification Proof with Syntactic Assertions

The syntactic version of the forward direction of the Falsification Theorem imposes more specific constraints on the assertions and execution model. We first lay out the general strategy for the proof, and then provide the formal details.

If we start with $\nvDash \langle\varphi\rangle\ C\ \langle\psi\rangle$, then we know that there exists some m such that $m \vDash \varphi$ and $[\![C]\!]^\dagger(m) \nvDash \psi$ and this implies that $[\![C]\!]^\dagger(m) \vDash \neg\psi$ since we are working with classical interpretations of BI. Now, we have a single program execution starting at φ and ending at $\neg\psi$, and we would like to extrapolate a valid OL triple from this (possibly with a precondition stronger than φ since the bad outcome $\neg\psi$ may only occur under some more specific constraints).

We are going to do this by induction on the program C. However, in cases that involve choice (*e.g.*, $C = C_1 + C_2$), we need to be able to split the postcondition into the components corresponding to the two choices (C_1 or C_2). This is possible, but only if the postcondition contains no implications. Logical negation is an implication ($\neg\psi$ is shorthand for $\psi \Rrightarrow \bot$), therefore we need a different postcondition ψ' that implies $\neg\psi$, but is syntactically valid. The precise form of ψ' will depend on the BI instance.

In addition, the program C must terminate after finitely many steps, otherwise the precondition that we generate may be infinitely large. Possible ways around this include using a fixed point

logic, however we are not aware of any versions of BI that have a fixed point operator. Instead, we will assume going forward that every program terminates after finitely many steps.

In order to make the argument formal, we first introduce the notion of a falsifiable OL instance which adds the constraints needed to split assertions and extrapolate triples. Next, we prove a couple of intermediate lemmas before giving the main result. In the next sections, we will instantiate this result to the nondeterministic and probabilistic evaluation models.

Definition D.2 (Falsifiable OL). An instance of OL is falsifiable if it has the following properties:

(1) The PCM operation has the properties:
   (a) If $m_1 \diamond m_2 = \varnothing$, then $m_1 = m_2 = \varnothing$
   (b) If $m_1 \diamond m_2 = n_1 \diamond n_1$, then there exist s_1, s_2, t_1, t_2 such that $s_1 \diamond s_2 = n_1$, $t_1 \diamond t_2 = n_2$, $s_1 \diamond t_1 = m_1$ and $s_2 \diamond t_2 = m_2$.
(2) Atomic assertions P are splittable, that is if $m_1 \diamond m_2 \vDash P$, then there exist φ_1 and φ_2 such that $m_1 \vDash \varphi_1$ and $m_2 \vDash \varphi_2$ and $\varphi_1 \oplus \varphi_2 \Rightarrow P$.
(3) Sequences of outcomes are falsifiable, $m \nvDash Q_1 \oplus \cdots \oplus Q_n$, iff $\exists \psi$ containing no implications such that $m \vDash \psi$ and $\psi \Rightarrow \neg \bigoplus_{i=1}^{n} Q_j$.
(4) Atomic commands have trace extrapolation, if $[\![c]\!]^\dagger(m) \vDash \psi$, then there exists φ such that $m \vDash \varphi$ and $\vDash \langle \varphi \rangle\ c\ \langle \psi \rangle$ (where φ and ψ have no implications).

Lemma D.3 (Splitting). *For any BI assertion φ that contains no implications and where the BI frame comes from a falsifiable OL instance, if $m_1 \diamond m_2 \vDash \varphi$, then there exist φ_1 and φ_2 such that $m_1 \vDash \varphi_1$ and $m_2 \vDash \varphi_2$ and $\varphi_1 \oplus \varphi_2 \Rightarrow \varphi$.*

Proof. By induction on the structure of φ (Figure 3).

- $\varphi = \top$. Clearly $m_1 \vDash \top$ and $m_2 \vDash \top$ and $\top \oplus \top \Rightarrow \top$.
- $\varphi = \bot$. Vacuous since $m_1 \diamond m_2 \vDash \bot$ is impossible.
- $\varphi = \top^\oplus$. If $m_1 \diamond m_2 \vDash \top^\oplus$, then it must be the case that $m_1 = m_2 = \varnothing$ (by property (1a) of Definition D.2). So, $m_1 \vDash \top^\oplus$ and $m_2 \vDash \top^\oplus$ and $\top^\oplus \oplus \top^\oplus \Rightarrow \top^\oplus$.
- $\varphi = \psi' \wedge \psi$. We know that $m_1 \diamond m_2 \vDash \psi' \wedge \psi$, so $m_1 \diamond m_2 \vDash \psi'$ and $m_1 \diamond m_2 \vDash \psi$. By the induction hypotheses, There are φ_1, φ_2, ψ_1, and ψ_2 such that $m_1 \vDash \varphi_1$ and $m_2 \vDash \varphi_2$ and $m_1 \vDash \psi_1$ and $m_2 \vDash \psi_2$ and $\varphi_1 \oplus \varphi_2 \Rightarrow \psi'$ and $\psi_1 \oplus \psi_2 \Rightarrow \psi$. Therefore, $m_1 \vDash \varphi_1 \wedge \psi_1$ and $m_2 \vDash \varphi_2 \wedge \psi_2$. Now, suppose $m' \vDash (\varphi_1 \wedge \psi_1) \oplus (\varphi_2 \wedge \psi_2)$. Then $m'_1 \vDash \varphi_1$, $m'_1 \vDash \psi_1$, $m'_2 \vDash \varphi_2$, and $m'_2 \vDash \psi_2$ such that $m'_1 \diamond m'_2 = m'$. So, $m' \vDash \varphi_1 \oplus \varphi_2$ and $m' \vDash \psi_1 \oplus \psi_2$ and by the implications from the induction hypotheses, $m' \vDash \psi'$ and $m' \vDash \psi$, so $m' \vDash \psi' \wedge \psi$.
- $\varphi = \psi' \oplus \psi$. We know that $m_1 \diamond m_2 \vDash \psi' \oplus \psi$, so $n_1 \vDash \psi'$ and $n_2 \vDash \psi$ such that $n_1 \diamond n_2 = m_1 \diamond m_2$. By property (1b) of Definition D.2, there must be s_1, s_2, t_1 and t_2 such that $s_1 \diamond s_2 = n_1$, $t_1 \diamond t_2 = n_2$, $s_1 \diamond t_1 = m_1$ and $s_2 \diamond t_2 = m_2$. So, $s_1 \diamond s_2 \vDash \psi'$ and $t_1 \diamond t_2 \vDash \psi$, and by the induction hypothesis, $s_1 \vDash \varphi_1$, $s_2 \vdash \varphi_2$, $t_1 \vDash \psi_1$ and $t_2 \vDash \psi_2$ such that $\varphi_1 \oplus \varphi_2 \Rightarrow \psi'$ and $\psi_1 \oplus \psi_2 \Rightarrow \psi$. Recombining terms, we get that $m_1 \vDash \varphi_1 \oplus \psi_1$ and $m_2 \vDash \varphi_2 \oplus \psi_2$ and it is easy to see that $\varphi_1 \oplus \varphi_2 \oplus \psi_1 \oplus \psi_2 \Rightarrow \psi' \oplus \psi$.
- $\varphi = \psi' \Rightarrow \psi$. Vacuous since we assumed φ has no implications.
- $\varphi = P$. By property (2) from Definition D.2.

□

Lemma D.4 (Trace Extrapolation). *For any falsifiable OL instance, if there exists m such that $[\![C]\!]^\dagger(m) \vDash \psi$ (where ψ contains no implications), then there exists φ (also with no implications) such that $m \vDash \varphi$ and $\vDash \langle \varphi \rangle\ C\ \langle \psi \rangle$.*

Proof. By induction on the structure of the program C (see Figure 2).

- $C = \mathbb{0}$. Assume that $[\![\mathbb{0}]\!]^\dagger(m) \vDash \psi$. Since $[\![\mathbb{0}]\!]^\dagger(m) = \varnothing$, then this assumption gives us $\varnothing \vDash \psi$. We can then take $\varphi = \top$ and derive $\vDash \langle \varphi \rangle\ \mathbb{0}\ \langle \psi \rangle$: for any $m' \vDash \top$ we have $[\![\mathbb{0}]\!]^\dagger(m') \vDash \psi$ since we know $\varnothing \vDash \psi$ and $[\![\mathbb{0}]\!]^\dagger(m') = \varnothing$, for any m'. We also clearly have $m \vDash \top$.
- $C = \mathbb{1}$. Assume $[\![\mathbb{1}]\!]^\dagger(m) \vDash \psi$. Since $[\![\mathbb{1}]\!]^\dagger(m) = m$, then this assumption gives us $m \vDash \psi$. We can take $\varphi = \psi$ and immediately derive $m \vDash \varphi$. We can then also derive $\vDash \langle \varphi \rangle\ \mathbb{1}\ \langle \psi \rangle$: for any $m' \vDash \varphi$ we have $[\![\mathbb{1}]\!]^\dagger(m') = m' \vDash \psi$ since we know $\varphi = \psi$.
- $C = C_1 + C_2$. Assume $[\![C_1 + C_2]\!]^\dagger(m) \vDash \psi$. We know that $[\![C_1 + C_2]\!]^\dagger(m) = [\![C_1]\!]^\dagger(m) \diamond [\![C_2]\!]^\dagger(m)$, so $[\![C_1]\!]^\dagger(m) \diamond [\![C_2]\!]^\dagger(m) \vDash \psi$. By Lemma D.3, we know that there exist ψ_1, ψ_2 such that $[\![C_1]\!]^\dagger(m) \vDash \psi_1$ and $[\![C_2]\!]^\dagger(m) \vDash \psi_2$ and $\psi_1 \oplus \psi_2 \Rrightarrow \psi$. By induction, there exist φ_i such that $\vDash \langle \varphi_i \rangle\ C_i\ \langle \psi_i \rangle$ for $i \in \{1, 2\}$ and $m \vDash \varphi_i$. Now, we pick the precondition $\varphi = \varphi_1 \wedge \varphi_2$ (so $m \vDash \varphi$). It remains to argue that $\vDash \langle \varphi \rangle\ C_1 + C_2\ \langle \psi \rangle$: for any $m' \vDash \varphi_1 \wedge \varphi_2$, we know that $m' \vDash \varphi_i$ and using the fact that $\vDash \langle \varphi_i \rangle\ C_i\ \langle \psi_i \rangle$, we conclude that $[\![C_i]\!]^\dagger(m') \vDash \psi_i$ (for $i = 1, 2$). Hence, $[\![C_1 + C_2]\!]^\dagger(m') \vDash \psi_1 \oplus \psi_2$ and since $\psi_1 \oplus \psi_2 \Rrightarrow \psi$ we can conclude $[\![C_1 + C_2]\!]^\dagger(m') \vDash \psi$.
- $C = C_1 \fatsemi C_2$. Assume $[\![C_1 \fatsemi C_2]\!]^\dagger(m) \vDash \psi$. We know that $[\![C_1 \fatsemi C_2]\!]^\dagger(m) = [\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m))$, so $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) \vDash \psi$. By the induction hypothesis, we conclude that there exists ϑ such that $\vDash \langle \vartheta \rangle\ C_2\ \langle \psi \rangle$ and $[\![C_1]\!]^\dagger(m) \vDash \vartheta$. By induction again, we get that $\vDash \langle \varphi \rangle\ C_1\ \langle \vartheta \rangle$ such that $m \vDash \varphi$. Now, to show that $\vDash \langle \varphi \rangle\ C_1 \fatsemi C_2\ \langle \psi \rangle$, suppose that $m' \vDash \varphi$, then we know that $[\![C_1]\!]^\dagger(m') \vDash \vartheta$ (from $\vDash \langle \varphi \rangle\ C_1\ \langle \vartheta \rangle$), and we know that $[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m')) \vDash \psi$ (from $\vDash \langle \vartheta \rangle\ C_2\ \langle \psi \rangle$), so therefore $\vDash \langle \varphi \rangle\ C_1 \fatsemi C_2\ \langle \psi \rangle$.
- $C = C^\star$. We will first show that for any n, there is a φ such that $\vDash \langle \varphi \rangle\ C^n\ \langle \psi \rangle$ and $m \vDash \varphi$. The proof is by induction on n. The case where $n = 0$ follows from the $\mathbb{1}$ case above. Now, by the induction hypothesis we know that there is some ϑ such that $\vDash \langle \vartheta \rangle\ C^n\ \langle \psi \rangle$ and $[\![C]\!]^\dagger(m) \vDash \vartheta$. By the previous induction hypothesis, we know that $\vDash \langle \varphi \rangle\ C\ \langle \vartheta \rangle$, such that $m \vDash \varphi$. So, combining these results, we get that $\vDash \langle \varphi \rangle\ C^{n+1}\ \langle \psi \rangle$.
  Now, since we assumed that the program terminates after finitely many steps, there must be some n such that $[\![C^\star]\!]^\dagger(m) = [\![\mathbb{1}]\!]^\dagger(m) \diamond [\![C]\!]^\dagger(m) \diamond \cdots \diamond [\![C^n]\!]^\dagger(m)$. By repeatedly using Lemma D.3, we can split ψ into $\psi_0, \ldots, \psi_n$ such that $[\![C^k]\!]^\dagger(m) \vDash \psi_k$ (for each $k \in \{0, \ldots, n\}$) and $\psi_0 \oplus \cdots \oplus \psi_n \Rrightarrow \psi$. By the inductive proof above, for each k, there is a φ_k such that $\vDash \langle \varphi_k \rangle\ C^k\ \langle \psi_k \rangle$ and $m \vDash \varphi_k$. Now, let $\varphi = \varphi_0 \wedge \cdots \wedge \varphi_n$, so clearly $m \vDash \varphi$. We can conclude that $\vDash \langle \varphi \rangle\ C^\star\ \langle \psi_0 \oplus \cdots \oplus \psi_n \rangle$ by an argument analogous to the $C = C_1 + C_2$ case. Finally, since $\psi_0 \oplus \cdots \oplus \psi_n \Rrightarrow \psi$, we can weaken the postcondition to obtain $\vDash \langle \varphi \rangle\ C^\star\ \langle \psi \rangle$.
- $C = c$. By property (4) of Definition D.2.

□

Theorem D.5 (Falsification). *For any falsifiable OL instance,*

$$\nvDash \langle \varphi \rangle\ C\ \langle \bigoplus_{i=1}^{n} Q_i \rangle \quad \textit{iff} \quad \exists \varphi' \Rrightarrow \varphi \quad \textit{and} \quad \exists \psi \Rrightarrow \neg \bigoplus_{i=1}^{n} Q_i \quad \textit{such that} \quad \vDash \langle \varphi' \rangle\ C\ \langle \psi \rangle$$

Where ψ has no implications and $\mathrm{sat}(\varphi')$.

Proof.

(⇒) Since $\nvDash \langle \varphi \rangle\ C\ \langle \bigoplus_{i=1}^{n} Q_i \rangle$, then there is an m such that $m \vDash \varphi$ and $[\![C]\!]^\dagger(m) \nvDash \bigoplus_{i=1}^{n} Q_i$. By property (3) of Definition D.2, we know that there exists a ψ with no implications such that $\psi \Rrightarrow \neg \bigoplus_{i=1}^{n} Q_i$ and $[\![C]\!]^\dagger(m) \vDash \psi$. We can now use Lemma D.4 to conclude that there is a ϑ such that $\vDash \langle \vartheta \rangle\ C\ \langle \psi \rangle$ and $m \vDash \vartheta$. Now, let $\varphi' = \vartheta \wedge \varphi$ (φ' is satisfiable since $m \vDash \vartheta$ and $m \vDash \varphi$).

Clearly also $\varphi' \Rightarrow \varphi$. It just remains to show that $\vDash \langle \varphi' \rangle\ C\ \langle \psi \rangle$: for any $m' \vDash \varphi'$, then $m' \vDash \vartheta$ and so $[\![C]\!]^\dagger(m') \vDash \psi$ (since $\vDash \langle \vartheta \rangle\ C\ \langle \psi \rangle$).

($\Leftarrow$) Assume that $\varphi' \Rightarrow \varphi$ and $\psi \Rightarrow \neg \bigoplus_{i-1}^{n} Q_i$ and $\vDash \langle \varphi' \rangle\ C\ \langle \psi \rangle$. By weakening, we can also conclude that $\vDash \langle \varphi' \rangle\ C\ \langle \neg \bigoplus_{i-1}^{n} Q_i \rangle$. By Theorem 5.2, we can conclude that $\nvDash \langle \varphi \rangle\ C\ \langle \bigoplus_{i-1}^{n} Q_i \rangle$.

□

Remark 2. The restrictions laid out in Definition D.2 are quite specific, but we will see in the following sections that they are naturally satisfied in both the nondeterministic and probabilistic models. While the Trace Extrapolation (Lemma D.4) property may seem unconventional, it can be thought of as a more specialized version of a weakest precondition transformer [Dijkstra 1976]. Indeed, if we had such a predicate transformer, we would know that $[\![C]\!]^\dagger(m) \vDash \psi$ implies that $m \vDash \mathsf{wp}(C, \psi)$ and that $\vDash \langle \mathsf{wp}(C, \psi) \rangle\ C\ \langle \psi \rangle$ is a valid triple. Unfortunately, weakest preconditions have complex interactions with choice mechanisms such as $C_1 + C_2$ and $x \xleftarrow{\$} \eta$ (refer to Kaminski [2019] for a more in-depth discussion of wp and choice). The question of whether a weakest precondition exists for OL remains open. We plan to explore this more in future work, but for now we use the more specialized Trace Extrapolation property to complete the falsification proof.

Before moving on to the evaluation-model-specific falsification results, we prove a useful lemma about trace extrapolation for pure commands.

Lemma D.6 (Trace Extrapolation for Pure Commands). *For any OL instance where trace extrapolation holds for pure commands c and basic assertions P, Q, that is:*

$$[\![c]\!]^\dagger(m) \vDash Q \quad \Rightarrow \quad \exists P.\ \ m \vDash P \ \ \text{and} \ \ \vDash \langle P \rangle\ c\ \langle Q \rangle$$

Then trace extrapolation holds for pure commands and any assertions that do not have implications:

$$[\![c]\!]^\dagger(m) \vDash \psi \quad \Rightarrow \quad \exists \varphi \text{ with no implications.}\ \ m \vDash \varphi \ \ \text{and} \ \ \vDash \langle \varphi \rangle\ c\ \langle \psi \rangle$$

Proof. By induction on the structure of ψ:

- $\psi = \top$. Let $\varphi = \top$. Clearly $m \vDash \top$ and $\vDash \langle \top \rangle\ c\ \langle \top \rangle$: suppose $m' \vDash \top$, then clearly $[\![c]\!]^\dagger(m') \vDash \top$.
- $\psi = \bot$. Vacuous since $[\![c]\!]^\dagger(m) \vDash \bot$ is impossible.
- $\psi = \psi_1 \wedge \psi_2$. Assume $[\![c]\!]^\dagger(m) \vDash \psi_1 \wedge \psi_2$. By induction, we know that there is a φ_i such that $m \vDash \varphi_i$ and $\vDash \langle \varphi_i \rangle\ c\ \langle \psi_i \rangle$ for $i \in \{1, 2\}$. Now, let $\varphi = \varphi_1 \wedge \varphi_2$, so clearly $m \vDash \varphi$. We now show that $\vDash \langle \varphi \rangle\ c\ \langle \psi_1 \wedge \psi_2 \rangle$: suppose $m' \vDash \varphi$, then $m' \vDash \varphi_i$ for $i \in \{1, 2\}$. Since $\vDash \langle \varphi_i \rangle\ c\ \langle \psi_i \rangle$, then $[\![c]\!]^\dagger(m') \vDash \psi_i$, therefore $[\![c]\!]^\dagger(m') \vDash \psi_1 \wedge \psi_2$.
- $\psi = \psi_1 \oplus \psi_2$. Assume $[\![c]\!]^\dagger(m) \vDash \psi_1 \oplus \psi_2$. Since c is pure, it cannot split into multiple outcomes, therefore the fact that $[\![c]\!]^\dagger(m)$ has multiple outcomes means that m also must have multiple outcomes, so there must be m_1, m_2 such that $m = m_1 \diamond m_2$ and $[\![c]\!]^\dagger(m_1) \vDash \psi_1$ and $[\![c]\!]^\dagger(m_2) \vDash \psi_2$. By induction, we know that there is a φ_i such that $m_i \vDash \varphi_i$ and $\vDash \langle \varphi_i \rangle\ c\ \langle \psi_i \rangle$ for $i \in \{1, 2\}$. Now, let $\varphi = \varphi_1 \oplus \varphi_2$, so clearly $m \vDash \varphi$. Now we show that $\vDash \langle \varphi \rangle\ c\ \langle \psi_1 \oplus \psi_2 \rangle$: suppose $m' \vDash \varphi$, so $m'_1 \vDash \varphi_1$ and $m'_2 \vDash \varphi_2$ such that $m'_1 \diamond m'_2 = m'$. For each $i \in \{1, 2\}$, we know that $\vDash \langle \varphi_i \rangle\ c\ \langle \psi_i \rangle$ for $i \in \{1, 2\}$ and so $[\![c]\!]^\dagger(m'_i) \vDash \psi_i$. Combining these, we get $[\![c]\!]^\dagger(m') \vDash \psi_1 \oplus \psi_2$.
- $\psi = \psi_1 \Rightarrow \psi_2$. Vacuous since we assumed that ψ has no implications.
- $\psi = Q$. By assumption.

□

D.3 Nondeterministic Falsification

This section contains proofs for the falsification results in Section 5.1. The goal is to show that nondeterministic instances of OL are falsifiable by showing that Definition D.2 holds for instances

of OL using the nondeterministic evaluation model and outcome logic. We first prove that assertions can be falsified, then we prove trace extrapolation, and then we prove Definition D.2.

Lemma 5.5 (Falsifying Assertions). *For any $S \in 2^\Sigma$ and atomic assertions $Q_1, \ldots, Q_n$,*

$$S \nvDash Q_1 \oplus \cdots \oplus Q_n \qquad \textit{iff} \qquad \exists i.\ S \vDash \overline{Q}_i \quad \textit{or} \quad S \vDash (\overline{Q}_1 \wedge \cdots \wedge \overline{Q}_n) \oplus \top \quad \textit{or} \quad S \vDash \top^\oplus$$

Proof.

($\Rightarrow$) Suppose that $S \nvDash Q_1 \oplus \cdots \oplus Q_n$, so for all $S_1, \ldots, S_n$, if $S = \bigcup_{i=1}^n S_i$, there exists some i such that $S_i \nvDash Q_i$. Now, for each i, let $T_i = \{\sigma \mid \sigma \in S, \sigma \vDash_\Sigma Q_i\}$, so by construction $T_i \subseteq S$ and therefore $\bigcup_{i=1}^n T_i \subseteq S$. If $S \neq \bigcup_{i=1}^n T_i$, then $S \supset \bigcup_{i=1}^n T_i$, so there must be some $\tau \in S$ such that for all i, $\tau \notin T_i$ and so for all i, $\tau \nvDash_\Sigma Q_i$, or in other words, $\tau \vDash_\Sigma \overline{Q}_i$. So, $S \vDash (\overline{Q}_1 \wedge \cdots \wedge \overline{Q}_n) \oplus \top$. Otherwise, it must be the case that $S = \bigcup_{i=1}^n T_i$ and we therefore know that there exists some i such that $T_i \nvDash Q_i$. By construction, $\sigma \vDash_\Sigma Q_i$ for every $\sigma \in T_i$, so it must be that $T_i = \emptyset$. This means that $\sigma \nvDash_\Sigma Q_i$ for every $\sigma \in S$, so $S \vDash \overline{Q}_i$. Or, if $S = \emptyset$, then $S \vDash \top^\oplus$.

($\Leftarrow$) There are three cases:

- Suppose there is some i such that $S \vDash \overline{Q}_i$. This means that $S \neq \emptyset$ and $\forall \sigma \in S.\ \sigma \nvDash_\Sigma Q_i$. Since there are no states satisfying Q_i, then there is no $S_i \subseteq S$ such that $S_i \vDash Q_i$ and therefore $S \nvDash Q_1 \oplus \cdots \oplus Q_n$.
- Suppose that $S \vDash (\overline{Q}_1 \wedge \cdots \wedge \overline{Q}_n) \oplus \top$, so by Lemma B.4, there is some state $\sigma \in S$ such that σ does not satisfy any Q_i. Therefore $S \nvDash Q_1 \oplus \cdots \oplus Q_n$: there is now way to break S into n parts each of which satisfying a Q_i because none of those sets can contain σ.
- Suppose $S \vDash \top^\oplus$ and so $S = \emptyset$. Then clearly $S \nvDash Q_1 \oplus \cdots \oplus Q_n$: we cannot witness each Q_i because there are no states at all.

□

Lemma D.7 (Nondeterministic Trace Extrapolation). *If $[\![c]\!]^\dagger(S) \vDash \psi$ and ψ has no implications, then there is some φ such that $S \vDash \varphi$ and $\vDash \langle\varphi\rangle\ c\ \langle\psi\rangle$.*

Proof. By cases on the structure of c.

▸ $c = (\text{assume } e)$. We know that $[\![\text{assume } e]\!]^\dagger(S) \vDash \psi$. Let $S_1 = [\![\text{assume } e]\!]^\dagger(S) = \{\sigma \mid \sigma \in S, \sigma \vDash e\}$ and $S_2 = [\![\text{assume } \neg e]\!]^\dagger(S) = \{\sigma \mid \sigma \in S, \sigma \nvDash e\}$. Clearly $S_1 \cup S_2 = S$, since the two assume statements partition S into two parts. We now define φ as follows:

$$\varphi = \varphi_1 \oplus \varphi_2 \qquad \text{where} \qquad \varphi_1 = \begin{cases} \psi \wedge e & \text{if } S_1 \neq \emptyset \\ \psi \wedge \top^\oplus & \text{if } S_1 = \emptyset \end{cases} \qquad \varphi_2 = \begin{cases} \overline{e} & \text{if } S_2 \neq \emptyset \\ \top^\oplus & \text{if } S_2 = \emptyset \end{cases}$$

We now show that $S_1 \vDash \varphi_1$: we already know that $S_1 \vDash \psi$ by assumption. If $S_1 \neq \emptyset$, then it must satisfy e, since by construction all the states in S_1 satisfy e. If not, then $S_1 = \emptyset \vDash \top^\oplus$. A similar argument shows that $S_2 \vDash \varphi_2$. Given this, $S \vDash \varphi$.
It remains to show that $\vDash \langle\varphi\rangle$ assume e $\langle\psi\rangle$. Suppose $T \vDash \varphi$, so $T_1 \vDash \varphi_1$ and $T_2 \vDash \varphi_2$ such that $T_1 \cup T_2 = T$. Since all the states satisfying e from T are in T_1, then $[\![\text{assume } e]\!]^\dagger(T) = T_1$. Since $T_1 \vDash \varphi_1$, then $T_1 \vDash \psi$.

▸ c is a pure command. It suffices to show that the property holds for basic assertions Q, we can then use Lemma D.6 to complete the proof.
Suppose that $[\![c]\!]^\dagger(S) \vDash Q$. This means that $[\![c]\!]^\dagger(S) \neq \emptyset$ and $\forall \sigma\ [\![c]\!]^\dagger(S).\ \sigma \vDash Q$. For pure commands, there are well known weakest precondition predicate transformations that satisfy $\tau \vDash Q$ iff $\sigma \vDash \text{wp}(c, Q)$ such that $\{\tau\} = [\![c]\!]\ (\sigma)$. This includes the rules for variable assignment ($\text{wp}(x := v, Q) = Q[v/a]$) as well as the backwards reasoning rules for Separation Logic given by Reynolds [2002]. So, it must be the case that $S \vDash \text{wp}(c, Q)$: since c is pure, it cannot change the magnitude of the set, so $S \neq \emptyset$. In addition, since all the states in the output set satisfy Q,

then the states in S must all satisfy $\mathrm{wp}(c, Q)$. Finally, we conclude that $\vDash \langle \mathrm{wp}(c,Q)\rangle\ c\ \langle Q\rangle$: suppose that $T \vDash \mathrm{wp}(c,Q)$, then $T \neq \emptyset$ and $\forall \sigma \in T.\ \sigma \vDash \mathrm{wp}(c,Q)$. By the properties of weakest preconditions, we know that $\tau \vDash Q$ if $\{\tau\} = [\![c]\!](\sigma)$, so everything in $[\![c]\!]^\dagger(T)$ must satisfy Q. Additionally, c is pure and cannot change the size of T, so $[\![c]\!]^\dagger(T) \neq \emptyset$. This means that $[\![c]\!]^\dagger(T) \vDash Q$.

□

Lemma D.8. *The nondeterministic instance of OL is falsifiable*

Proof.

(1) Properties of the PCM $\langle 2^{2^\Sigma}, \cup, \emptyset\rangle$:
   (a) If $S \cup T = \emptyset$, then it must be the case that $S = T = \emptyset$.
   (b) Suppose that $S_1 \cup S_2 = T_1 \cup T_2$. Now, let $U_1 = S_1 \cap T_1$, $U_2 = S_2 \cap T_1$, $V_1 = S_1 \cap T_2$, and $V_2 = S_2 \cap T_2$. It is easy to see that $U_1 \cup U_2 = T_1$ and $V_1 \cup V_2 = T_2$ and $U_1 \cup V_1 = S_1$ and $U_2 \cup V_2 = S_2$.
(2) Basic assertion splitting: If $S_1 \cup S_2 \vDash P$, then there are three options. If $S_1 = \emptyset$, then $S_1 \vDash \top^\oplus$ and $S_2 \vDash P$ and clearly $\top^\oplus \oplus P \Rightarrow P$. The case where $S_2 = \emptyset$ is symmetrical. Finally, if both S_1 and S_2 are nonempty, then $S_1 \vDash P$ and $S_2 \vDash P$ and $P \oplus P \Rightarrow P$.
(3) Assertion falsification: Follows from Lemma 5.5.
(4) Trace extrapolation: By Lemma D.7.

□

The following theorem is a more specific version of Theorem D.5 where we include a more specific postcondition (following from Lemma 5.5) instead of existentially quantifying the postcondition.

Theorem 5.6 (Nondeterministic Falsification). *For any OL instance based on the nondeterministic evaluation model (Definition 5.3) and outcome assertions (Definition 5.4),* $\nvDash \langle\varphi\rangle\ C\ \langle \bigoplus_{i=1}^n Q_i\rangle$ *iff:*

$$\exists \varphi' \Rightarrow \varphi.\ \mathrm{sat}(\varphi') \quad \textit{and} \quad \exists i.\ \vDash \langle\varphi'\rangle\ C\ \langle \overline{Q}_i\rangle \ \textit{or}\ \vDash^{\downarrow} \langle\varphi'\rangle\ C\ \langle \bigwedge_{i=1}^n \overline{Q}_i\rangle \ \textit{or}\ \vDash \langle\varphi'\rangle\ C\ \langle \top^\oplus\rangle$$

Proof. Follows directly from Lemmas 5.5 and D.8 and theorem D.5. □

Now, we show that OL can disprove any Hoare Triple, which means that it fully subsumes the use case of Incorrectness Logic.

Corollary 5.7 (Hoare Logic Falsification).

$$\nvDash \{P\}\ C\ \{Q\} \qquad \textit{iff} \qquad \exists \varphi \Rightarrow P.\quad \mathrm{sat}(\varphi) \quad \textit{and} \quad \vDash^{\downarrow} \langle\varphi\rangle\ C\ \langle \overline{Q}\rangle$$

Proof.

(⇒) Assume $\nvDash \{P\}\ C\ \{Q\}$. From Theorem 4.5, we get that $\nvDash \langle P\rangle\ C\ \langle Q \vee \top^\oplus\rangle$. This means that there is some S such that $S \vDash P$ and $[\![C]\!]^\dagger(S) \nvDash Q \vee \top^\oplus$, which implies that $[\![C]\!]^\dagger(S) \nvDash Q$ and $[\![C]\!]^\dagger(S) \nvDash \top^\oplus$, which implies that $[\![C]\!]^\dagger(S) \vDash \overline{Q} \oplus \top$. Now, we can use Lemma D.4 to conclude that there is an assertion ϑ such that $S \vDash \vartheta$ and $\vDash \langle\vartheta\rangle\ C\ \langle \overline{Q} \oplus \top\rangle$. Now let $\varphi = \vartheta \wedge P$, so clearly $S \vDash \varphi$ and since $\varphi \Rightarrow \vartheta$, then $\vDash \langle\varphi\rangle\ C\ \langle \overline{Q} \oplus \top\rangle$, or equivalently, $\vDash^{\downarrow} \langle\varphi\rangle\ C\ \langle \overline{Q}\rangle$.

(⇐) Since $\mathrm{sat}(\varphi)$, there is some $S \vDash \varphi$ and since $\varphi \Rightarrow P$, then also $S \vDash P$. From $\vDash^{\downarrow} \langle\varphi\rangle\ C\ \langle \overline{Q}\rangle$, we know that there is a $\tau \in [\![C]\!]^\dagger(S)$ such that $\tau \nvDash_\Sigma Q$. There must also be some $\sigma \in S$ such that $\tau \in [\![C]\!](\sigma)$, and since $\sigma \in S$ and $S \vDash P$, then $\sigma \vDash_\Sigma P$. So, we have now shown that $\sigma \vDash_\Sigma P$ and $\tau \in [\![C]\!](\sigma)$, and $\tau \nvDash_\Sigma Q$, therefore $\nvDash \{P\}\ C\ \{Q\}$.

□

D.4 Probabilistic Falsification

In this section, we prove the claims from Section 5.2 pertaining to the falsifiability of probabilistic OL triples. The goal is to prove that probabilistic instances of OL uphold Definition D.2. We begin by showing that sequences of assertions can be falsified (Lemma D.9), then we we show Trace Extrapolation (Lemma D.11), and finally we show that probabilistic OL is falsifiable (Lemma D.12) implying that Theorem D.5 holds.

Lemma D.9 (Falsifying Probabilistic Assertions).

$$\mu \not\vDash \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i) \qquad \textit{iff} \qquad \exists \psi \textit{ (with no implications).} \quad \mu \vDash \psi \quad \textit{and} \quad \psi \Rightarrow \neg \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i)$$

Proof.

($\Rightarrow$) We begin by defining ψ as follows:

$$\psi \triangleq \bigoplus_{x \in \{0,1\}^n}(\mathbb{P}[B_x] = \mathbb{P}_\mu[B_x]) \qquad \text{where} \qquad B_x \triangleq \bigwedge_{i=1}^{n} \text{xor}(A_i, x_i)$$

In the above, x_i denotes the i^{th} bit of the string x, so if $x_i = 0$, then the i^{th} conjunct of B_x is A_i and if $x_i = 1$, then it is $\neg A_i$. Now, for any $\sigma \in \text{supp}(\mu)$, there must be exactly one B_x such that $\sigma \vDash_\Sigma B_x$. This is because for each A_i, either $\sigma \vDash A_i$ or $\sigma \vDash \neg A_i$, and so a unique B_x corresponds to these choices. That means that μ can be partitioned by its support into sub-distributions μ_x such that $\forall \sigma \in \text{supp}(\mu_x).\ \sigma \vDash_\Sigma B_x$ and $\mu = \sum_{x \in \{0,1\}^n} \mu_x$. Additionally, $|\mu_x| = \mathbb{P}_\mu[B_x]$ since $\text{supp}(\mu_x)$ contains exactly those states that satisfy B_x. Therefore, for each x, $\mu_x \vDash (\mathbb{P}[B_x] = \mathbb{P}_\mu[B_x])$ and so $\mu \vDash \psi$.
Now we must show that $\psi \Rightarrow \neg \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i)$. Suppose that $\eta \vDash \psi$. This means that $\forall x \in \{0,1\}^n$ there is an η_x such that $\eta_x \vDash (\mathbb{P}[B_x] = \mathbb{P}_\mu[B_x])$ and $\eta = \sum_{x \in \{0,1\}^n} \eta_x$. This also implies that for all x, $|\eta_x| = |\mu_x|$.
For the sake of contradiction, suppose $\eta \vDash \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i)$. In order for this to be true, then for each i, we would need the following:

$$\sum_{x \in \{0,1\}^n, x_i = 0} \alpha_{x,i} \cdot \eta_x \vDash \mathbb{P}[A_i] = p_i$$

Where each $\alpha_{x,i}$ is a coefficient between 0 and 1 such that for all x, $\sum_{i=1}^{n} \alpha_{x,i} = 1$. Essentially, this distributes the probability mass of each η_x among all the A_i assertions that it is compatible with. Now, since for each x, $|\eta_x| = |\mu_x|$ and if $x_i = 0$, then every $\sigma \in \text{supp}(\mu_x)$ must satisfy A_i, we also have the following:

$$\sum_{x \in \{0,1\}^n, x_i = 0} \alpha_{x,i} \cdot \mu_x \vDash \mathbb{P}[A_i] = p_i$$

And this implies that $\mu \vDash \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i)$, which is a contradiction, therefore it must be the case that $\eta \vDash \neg \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i)$.

($\Leftarrow$) Suppose that there is some ψ such that $\mu \vDash \psi$ and $\psi \Rightarrow \neg \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i)$. By modus ponens, $\mu \vDash \neg \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i)$ and therefore $\mu \not\vDash \bigoplus_{i=1}^{n}(\mathbb{P}[A_i] = p_i)$.

□

The following lemma is needed for trace extrapolation.

Lemma D.10 (Assertion Scaling). *For any scalar $\alpha \neq 0$ and assertion φ, there exists a ψ such that for any μ if $\alpha \cdot |\mu| \leq 1$, then $\mu \vDash \varphi$ iff $\alpha \cdot \mu \vDash \psi$*

Proof. By induction on the structure of φ.

- $\varphi = \top$. Let $\psi = \top$. Clearly $\mu \vDash \top$ iff $\alpha \cdot \mu \vDash \top$ since both are always true.
- $\varphi = \bot$. Let $\psi = \bot$. Clearly $\mu \vDash \bot$ iff $\alpha \cdot \mu \vDash \bot$ since both are always false.
- $\varphi = \top^{\oplus}$. Let $\psi = \top^{\oplus}$. Clearly $\mu \vDash \top^{\oplus}$ iff $\alpha \cdot \mu \vDash \top^{\oplus}$ since both are true iff $\mu = \varnothing$.
- $\varphi = \varphi_1 \wedge \varphi_2$. By the induction hypothesis, there exist ψ_1 and ψ_2 such that for any μ, $\mu \vDash \varphi_i$ iff $\alpha \cdot \mu \vDash \psi_i$ for $i \in \{1, 2\}$. Now, let $\psi = \psi_1 \wedge \psi_2$, so clearly $\mu \vDash \varphi_1 \wedge \varphi_2$ iff $\alpha \cdot \mu \vDash \psi_1 \wedge \psi_2$.
- $\varphi = \varphi_1 \oplus \varphi_2$. By the induction hypothesis, there exist ψ_1 and ψ_2 such that for any μ, $\mu \vDash \varphi_i$ iff $\alpha \cdot \mu \vDash \psi_i$ for $i \in \{1, 2\}$. Now, let $\psi = \psi_1 \oplus \psi_2$. It must be that $\mu \vDash \varphi_1 \oplus \varphi_2$ iff $\mu_1 \vDash \varphi_1$ and $\mu_2 \vDash \varphi_2$ such that $\mu = \mu_1 + \mu_2$ iff $\alpha \cdot \mu_1 \vDash \psi_1$ and $\alpha \cdot \mu_2 \vDash \psi_2$ such that $\alpha \cdot \mu = \alpha \cdot \mu_1 + \alpha \cdot \mu_2$ iff $\alpha \cdot \mu \vDash \psi_1 \oplus \psi_2$.
- $\varphi = \varphi_1 \Rightarrow \varphi_2$. By the induction hypothesis, there exist ψ_1 and ψ_2 such that for any μ, $\mu \vDash \varphi_i$ iff $\alpha \cdot \mu \vDash \psi_i$ for $i \in \{1, 2\}$. Now, let $\psi = \psi_1 \Rightarrow \psi_2$, so clearly $\mu \vDash \varphi_1 \Rightarrow \varphi_2$ iff $\alpha \cdot \mu \vDash \psi_1 \Rightarrow \psi_2$.
- $\varphi = (\mathbb{P}[A] = p)$. Let $\psi = (\mathbb{P}[A] = \alpha \cdot p)$. It is easy to see that $\mu \vDash (\mathbb{P}[A] = p)$ iff $\alpha \cdot \mu \vDash (\mathbb{P}[A] = \alpha \cdot p)$ since $|\mu| = p$ iff $\alpha \cdot |\mu| = \alpha \cdot p$.

□

Lemma D.11 (Probabilistic Trace Extrapolation). *If $[\![c]\!]^{\dagger}(\mu) \vDash \psi$ and ψ has no implications, then there is some φ such that $\mu \vDash \varphi$ and $\vDash \langle \varphi \rangle\ c\ \langle \psi \rangle$.*

Proof. By cases on the structure of c.

- $c = (x \xleftarrow{\$} \eta)$. First, we know that:

$$[\![x \xleftarrow{\$} \eta]\!]^{\dagger}(\mu) = \sum_{\sigma \in \mathrm{supp}(\mu)} \mu(\sigma) \cdot \sum_{v \in \mathrm{supp}(\eta)} \eta(v) \cdot [\![x := v]\!](\sigma) = \sum_{v \in \mathrm{supp}(\eta)} [\![x := v]\!]^{\dagger}(\eta(v) \cdot \mu)$$

  So, we can apply Lemma D.3 many times to get a ψ_v for each v such that $[\![x := v]\!]^{\dagger}(\eta(v) \cdot \mu) \vDash \psi_v$ and $(\bigoplus_{v \in \mathrm{supp}(\eta)} \psi_v) \Rightarrow \psi$. Now, since $x := v$ is pure, we can use the next case of this proof to conclude that there is a φ_v such that $\eta(v) \cdot \mu \vDash \varphi_v$ and $\vDash \langle \varphi_v \rangle\ x := v\ \langle \psi_v \rangle$. Using Lemma D.10 (with $\alpha = \frac{1}{\eta(v)}$) we can get a φ'_v such that $\mu' \vDash \varphi'_v$ iff $\eta(v) \cdot \mu' \vDash \varphi_v$ (and therefore $\mu \vDash \varphi'_v$). Now, let $\varphi = \bigwedge_{v \in \mathrm{supp}(\eta)} \varphi'_v$, so clearly $\mu \vDash \varphi$. We can also show that $\vDash \langle \varphi \rangle\ x \xleftarrow{\$} \eta\ \langle \psi \rangle$:
  Suppose that $\mu' \vDash \varphi$. Then, $\mu' \vDash \varphi'_v$ for each v. This also means that $\eta(v) \cdot \mu' \vDash \varphi_v$. Now, using $\vDash \langle \varphi_v \rangle\ x := v\ \langle \psi_v \rangle$, we know that $[\![x := v]\!]^{\dagger}(\eta(v) \cdot \mu') \vDash \psi_v$. Combining these, we get $[\![x \xleftarrow{\$} \eta]\!]^{\dagger}(\mu') \vDash \bigoplus_{v \in \mathrm{supp}(\eta)} \psi_v$. This implies that $[\![x \xleftarrow{\$} \eta]\!]^{\dagger}(\mu') \vDash \psi$.
- $c = (\text{assume } e)$. We know that $[\![\text{assume } e]\!]^{\dagger}(\mu) \vDash \psi$. Now, let $\varphi = (\psi \wedge \mathbb{P}[e] = p) \oplus (\mathbb{P}[\neg e] = |\mu| - p)$ where $p = |\,[\![\text{assume } e]\!]^{\dagger}(\mu)|$.
  Let $\mu_1 = [\![\text{assume } e]\!]^{\dagger}(\mu)$ and $\mu_2 = [\![\text{assume } \neg e]\!]^{\dagger}(\mu)$. Clearly $\mu_1 + \mu_2 = \mu$, since the two assume statements partition the support of μ into two parts. It is also the case that $\mu_1 \vDash (\psi \wedge \mathbb{P}[e] = p)$ since we took $\mu_1 \vDash \psi$ as an assumption and $\mathbb{P}_{\mu_1}[e] = |\,[\![\text{assume } e]\!]^{\dagger}(\mu)|$ by construction. Similarly, $\mu_2 \vDash (\mathbb{P}[\neg e] = |\mu| - p)$ since μ_2 contains all the states where e is false by construction and must have mass equal to $|\mu| - |\mu_1|$. Therefore, $\mu \vDash \varphi$.
  We now show that $\vDash \langle \varphi \rangle\ \text{assume } e\ \langle \psi \rangle$. Suppose that $\mu' \vDash \varphi$. Therefore, $\mu_1 \vDash \psi \wedge \mathbb{P}[e] = p$ and $\mu_2 \vDash \mathbb{P}[\neg e] = (1 - p)$ such that $\mu' = \mu_1 + \mu_2$. It must be the case that $[\![\text{assume } e]\!]^{\dagger}(\mu') = \mu_1$, since $\neg e$ holds for every state in the support of μ_2. We already know that $\mu_1 \vDash \psi$, so $[\![\text{assume } e]\!]^{\dagger}(\mu') \vDash \psi$.
- c is a pure command. It suffices to show that the property holds for basic assertions $\mathbb{P}[A] = p$, we can then use Lemma D.6 to complete the proof.
  Suppose that $[\![c]\!]^{\dagger}(\mu) \vDash (\mathbb{P}[A] = p)$. This means that $|\,[\![c]\!]^{\dagger}(\mu)| = p$ and $\forall \sigma \in \mathrm{supp}([\![c]\!]^{\dagger}(\mu)).\ \sigma \vDash A$. For pure commands, there are well known weakest precondition predicate transformations

that satisfy $\tau \vDash A$ iff $\sigma \vDash \mathsf{wp}(c, A)$ such that $\mathsf{unit}(\tau) = [\![c]\!](\sigma)$. This includes the rules for variable assignment ($\mathsf{wp}(x := v, A) = A[v/a]$) as well as the backwards reasoning rules for Separation Logic given by Reynolds [2002]. So, it must be the case that $\mu \vDash \mathbb{P}[\mathsf{wp}(c, A)] = p$: since c is pure, it cannot change the mass of the distribution, so $|\mu| = |[\![c]\!]^\dagger(\mu)| = p$. In addition, since all the states in the output distribution satisfy A, then the states in μ must all satisfy $\mathsf{wp}(c, A)$. Finally, we conclude that $\vDash \langle \mathbb{P}[\mathsf{wp}(c, A)] = p \rangle\ c\ \langle \mathbb{P}[A] = p \rangle$: suppose that $\mu' \vDash \mathbb{P}[\mathsf{wp}(c, A)] = p$, then $|\mu'| = p$ and $\forall \sigma \in \mathsf{supp}(\mu').\ \sigma \vDash \mathsf{wp}(c, A)$. By the properties of weakest preconditions, we know that $\tau \vDash A$ if $\mathsf{unit}(\tau) = [\![c]\!](\sigma)$, so everything in the support of $[\![c]\!]^\dagger(\mu')$ must satisfy A. Additionally, c is pure and cannot change the mass of the distribution, so $|[\![c]\!]^\dagger(\mu')| = |\mu'| = p$. This means that $[\![c]\!]^\dagger(\mu') \vDash \mathbb{P}[A] = p$.

□

Lemma D.12. *The probabilistic instance of OL is falsifiable*

Proof.

(1) Properties of the PCM $\langle \mathcal{D}\Sigma, +, \varnothing \rangle$:
   (a) If $\mu_1 + \mu_2 = \varnothing$, then it must be the case that $\mu_1 = \mu_2 = \varnothing$ since the monoid operation + can only add probability mass, not remove it.
   (b) Suppose that $\mu_1 + \mu_2 = \eta_1 + \eta_2$. We now define the following:

$$\alpha_1 \triangleq \lambda x.\, \min(\mu_1(x), \eta_1(x)) \qquad \alpha_2 \triangleq \lambda x. \eta_1(x) - \alpha_1(x)$$
$$\beta_2 \triangleq \lambda x.\, \min(\mu_2(x), \eta_2(x)) \qquad \beta_1 \triangleq \lambda x. \eta_2(x) - \beta_2(x)$$

   By construction $\alpha_1 + \alpha_2 = \eta_1$ and $\beta_1 + \beta_2 = \eta_2$. We now show that for any x, $\alpha_1(x) + \beta_1(x) = \mu_1(x)$:

$$\begin{aligned}
\alpha_1(x) + \beta_1(x) &= \min(\mu_1(x), \eta_1(x)) + \eta_2(x) - \beta_2(x) \\
&= \min(\mu_1(x), \eta_1(x)) + \eta_2(x) - \min(\mu_2(x), \eta_2(x)) \\
&= \min(\mu_1(x), \eta_1(x)) + \max(\eta_2(x) - \mu_2(x), 0) \\
&= \min(\mu_1(x), \eta_1(x)) + \max(\mu_1(x) - \eta_1(x), 0)
\end{aligned}$$

   So, if $\mu_1(x) \leq \eta_1(x)$, then this equals $\mu_1(x) + 0$, otherwise it is $\eta_1(x) + \mu_1(x) - \eta_1(x)$.

$$= \mu_1(x)$$

   It is also true that $\alpha_2(x) + \beta_2(x) = \mu_2(x)$ by a symmetric argument.
(2) Basic assertion splitting: We know that $\mu_1 \diamond \mu_2 \vDash \mathbb{P}[A] = p$, so $|\mu_1| + |\mu_2| = p$ and all the states in both supports satisfy A. That means that $\mu_1 \vDash (\mathbb{P}[A] = |\mu_1|)$ and $\mu_2 \vDash (\mathbb{P}[A] = |\mu_2|)$ and $(\mathbb{P}[A] = |\mu_1|) \oplus (\mathbb{P}[A] = |\mu_2|) \Rightarrow (\mathbb{P}[A] = p)$.
(3) Assertion falsification: Follows from Lemma D.9.
(4) Trace extrapolation: By Lemma D.11.

□

While we have already shown that probabilistic OL is falsifiable, the result in Lemma D.12 gives us a falsifying postcondition that is exponentially large. If the original specification had n outcomes in the postcondition, then the specification that disproves it will have 2^n outcomes. We now show that in the common case where the outcomes are disjoint, the incorrectness specification only needs $n + 1$ outcomes.

Theorem 5.10 (Disjoint Falsification). *First, let* $A_0 = \bigwedge_{i=1}^{n} \neg A_i$. *If all the events are disjoint (for all* $i \neq j$, $A_i \wedge A_j$ *iff* false*), then:*

$$\not\vDash \langle \varphi \rangle\ C\ \langle \bigoplus_{i=1}^{n} (\mathbb{P}[A_i] = p_i) \rangle \qquad \textit{iff} \qquad \exists \vec{q}, \varphi' \Rightarrow \varphi. \quad \vDash \langle \varphi' \rangle\ C\ \langle \bigoplus_{i=0}^{n} (\mathbb{P}[A_i] = q_i) \rangle$$

Such that $\mathrm{sat}(\varphi')$ *and* $q_0 \neq 0$ *or for some* i $q_i \neq p_i$.

Proof. In general, if all the B_js are disjoint, then $\mu \vDash \bigoplus_{j=1}^{m} (\mathbb{P}[B_j] = r_j)$ iff for each j, $\mathbb{P}_\mu[B_j] = r_j$ and $|\mu| = \sum_{j=1}^{m} r_j$. This is easy to see, since the disjointness condition partitions the support of μ. It will now suffice to prove the following claim, the remainder of the proof then follows from Theorem 5.1. Claim: $\mu \not\vDash \bigoplus_{i=1}^{n} (\mathbb{P}[A_i] = p_i)$ iff $\exists \vec{q}.\ \mu \vDash \bigoplus_{i=0}^{n} (\mathbb{P}[A_i] = q_i)$. Due to disjointness, this is equivalent to saying that there is some i such that $\mathbb{P}_\mu[A_i] \neq p_i$ or $|\mu| \neq \sum_{i=1}^{n} p_i$ iff there exist $\vec{q}$ such that for each i, $\mathbb{P}_\mu[A_i] = q_i$ and $|\mu| = \sum_{i=0}^{n} q_i$ and either $q_0 \neq 0$ or there is some i such that $q_i \neq p_i$.

($\Rightarrow$) Let each $q_i = \mathbb{P}_\mu[A_i]$, with the addition of A_0, the A_is form a tautology, so they account for all the states in μ and therefore $\sum_{i=0}^{n} q_i = |\mu|$. By assumption, either $\mathbb{P}_\mu[A_i] \neq p_i$ or $|\mu| \neq \sum_{i=1}^{n} p_i$. If $\mathbb{P}_\mu[A_i] \neq p_i$, then clearly $p_i \neq q_i$. If every $\mathbb{P}_\mu[A_i] = p_i$, then it must be that $|\mu| \neq \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} q_i$, and so it must be that $q_0 \neq 0$.

($\Leftarrow$) Suppose that every $\mathbb{P}[A_i] = q_i$ and $\sum_{i=0}^{n} q_i = |\mu|$ and either $p_i \neq q_i$ for some i or $q_0 \neq 0$. If there is an i such that $p_i \neq q_i$, then clearly $\mathbb{P}[A_i] \neq p_i$. If each $p_i = q_i$, then it must be that $q_0 \neq 0$, and then $\sum_{i=1}^{n} p_i = (\sum_{i=0}^{n} q_i) - q_0 = |\mu| - q_0 \neq |\mu|$.

□

Going further, some specifications can be disproven using a single lower bound:

Theorem 5.11 (Principle of Denial for Lower Bounds).

$$\textit{If} \quad \exists \varphi' \Rightarrow \varphi. \quad \mathrm{sat}(\varphi'), \quad \vDash \langle \varphi' \rangle\ C\ \langle \mathbb{P}[\neg A] \geq q \rangle \quad \textit{then} \quad \not\vDash \langle \varphi \rangle\ C\ \langle \mathbb{P}[A] \geq p \rangle \quad (\textit{where } q > 1 - p)$$

Proof. We first show that $(\mathbb{P}[\neg A] \geq q) \Rightarrow \neg(\mathbb{P}[A] \geq p)$. Suppose that $\mu \vDash \mathbb{P}[\neg A] \geq q$, so by Lemma B.5, $\mathbb{P}_\mu[\neg A] \geq q$ and therefore $\mathbb{P}_\mu[\neg A] > 1 - p$. This also means that $\mathbb{P}_\mu[A] < |\mu| - (1 - p)$ and since $|\mu| \leq 1$, then $\mathbb{P}_\mu[A] < 1 - (1 - p) = p$. It follows that $\mathbb{P}_\mu[A] \not\geq p$.

Now, given the implication that we just proved, we can conclude that $\vDash \langle \varphi' \rangle\ C\ \langle \neg(\mathbb{P}[A] \geq p) \rangle$. Therefore, the original claim holds by Theorem 5.2. □

E SEPARATION LOGIC

In this section we define the semantics of the assertion logic and atomic commands defined in Section 6.4.

E.1 Semantics of the Assertion Logic

Recall the syntax for separation logic.

$$p \in \mathsf{SL} ::= \mathbf{emp} \mid \exists x.p \mid p \wedge q \mid p \vee q \mid p \Rightarrow q \mid p * q \mid p \mathrel{-\!\!*} q \mid e \mid e_1 \mapsto e_2 \mid e \mapsto - \mid e \not\mapsto$$

First we define the disjoint union of two heaps $\uplus : \mathcal{H} \to \mathcal{H} \rightharpoonup \mathcal{H}$ as follows:

$$h_1 \uplus h_2 \quad \triangleq \quad \lambda \ell. \begin{cases} h_1(\ell) & \text{if } \ell \in \mathrm{dom}(h_1) \\ h_2(\ell) & \text{if } \ell \in \mathrm{dom}(h_2) \end{cases} \qquad \text{if} \qquad \mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$$

The satisfaction relation $\vDash \subseteq (\mathcal{S} \times \mathcal{H}) \times \mathsf{SL}$ is defined as follows.

$$
\begin{array}{lll}
(s,h) \vDash \mathbf{emp} & \text{iff} & \mathsf{dom}(h) = \emptyset \\
(s,h) \vDash \exists x.p & \text{iff} & (s,h) \vDash p[v/x] \text{ for some } v \\
(s,h) \vDash p \wedge q & \text{iff} & (s,h) \vDash p \text{ and } (s,h) \vDash q \\
(s,h) \vDash p \vee q & \text{iff} & (s,h) \vDash p \text{ or } (s,h) \vDash q \\
(s,h) \vDash p \Rightarrow q & \text{iff} & \text{if } (s,h) \vDash p \text{ then } (s,h) \vDash q \\
(s,h) \vDash p * q & \text{iff} & \exists h_1, h_2 \text{ such that } h = h_1 \uplus h_2 \text{ and } (s,h_1) \vDash p \text{ and } (s,h_2) \vDash q \\
(s,h) \vDash p \mathrel{-\!\!*} q & \text{iff} & \forall h_1, h_2 \text{ such that } h_2 = h \uplus h_1 \text{ if } (s,h_1) \vDash p \text{ then } (s,h_2) \vDash q \\
(s,h) \vDash e & \text{iff} & [\![e]\!](s) = \mathsf{true} \\
(s,h) \vDash e_1 \mapsto e_2 & \text{iff} & [\![e_1]\!](s) = \ell \text{ and } \mathsf{dom}(h) = \{\ell\} \text{ and } h(\ell) = [\![e_2]\!](s) \\
(s,h) \vDash e \mapsto - & \text{iff} & [\![e]\!](s) = \ell \text{ and } \mathsf{dom}(h) = \{\ell\} \text{ and } h(\ell) \neq \bot \\
(s,h) \vDash e \not\mapsto & \text{iff} & [\![e]\!](s) = \mathsf{null} \text{ and } \mathsf{dom}(h) = \emptyset \text{ or} \\
 & & [\![e]\!](s) = \ell \text{ and } \mathsf{dom}(h) = \{\ell\} \text{ and } h(\ell) = \bot
\end{array}
$$

Note that this is a *classical* interpretation of separation logic where the points-to predicate $x \mapsto v$ is satisfied only by a singleton heap. We can add the *intuitionistic* points-to predicate $x \hookrightarrow v$ as syntactic sugar for $x \mapsto v * \mathsf{true}$ which is satisfied by *any* stack–heap pair (s,h) where $h([\![x]\!](s)) = [\![v]\!](s)$. The difference between $x \mapsto v$ and $x \hookrightarrow v$ is very similar to the difference between over- and under-approximate versions of outcomes that we saw in Section 4, where we defined under-approximation $m \vDash^{\downarrow} P$ to be $m \vDash P \oplus \top$.

E.2 Logical Operations on Errors

Let $\vDash_A \subseteq A \times \mathsf{Prop}_A$ and $\vDash_B \subseteq A \times \mathsf{Prop}_B$ be two logical satisfaction relations in which the assertion syntaxes (Prop_A and Prop_B) contain the usual logical constructs true, false, $\wedge$, $\vee$, and $\neg$. In addition, let $\mathsf{Prop} = \mathsf{Prop}_A \times \mathsf{Prop}_B$ and $\vDash \subseteq (B + A) \times \mathsf{Prop}$ is the satisfaction relation from Definition 6.1. We now add the following logical operations:

$$
\begin{array}{lcllcl}
\mathsf{true} & \triangleq & (\mathsf{true}, \mathsf{true}) & (p,q) \wedge (p',q') & \triangleq & (p \wedge p', q \wedge q') \\
\mathsf{false} & \triangleq & (\mathsf{false}, \mathsf{false}) & (p,q) \vee (p',q') & \triangleq & (p \vee p', q \vee q') \\
\neg(p,q) & \triangleq & (\neg p, \neg q) & & &
\end{array}
$$

To provide some justification for these definitions, we prove the following sanity checks.

Lemma E.1 (Sanity checks for logical operations). *The following statements hold for all m, p, p', q, and q'.*

- *True:* $m \vDash \mathsf{true}$
- *False:* $m \nvDash \mathsf{false}$
- *Conjunction:* $m \vDash (p,q) \wedge (p',q')$ *iff* $m \vDash (p,q)$ *and* $m \vDash (p',q')$
- *Disjunction:* $m \vDash (p,q) \vee (p',q')$ *iff* $m \vDash (p,q)$ *or* $m \vDash (p',q')$
- *Negation:* $m \vDash \neg(p,q)$ *iff* $m \nvDash (p,q)$
- *Sugar Syntax:* $(\mathsf{ok} : p) \vee (\mathsf{er} : q)$ *iff* (p,q)

Proof. We prove each case assuming that $m = \mathbb{i}_L(b)$. The cases where $m = \mathbb{i}_R(a)$ are symmetric.

- True: $\mathbb{i}_L(b) \vDash (\mathsf{true}, \mathsf{true})$ since $b \vDash \mathsf{true}$
- False: $\mathbb{i}_L(b) \nvDash (\mathsf{false}, \mathsf{false})$ since $b \nvDash \mathsf{false}$
- Conjunction: $\mathbb{i}_L(b) \vDash (p \wedge p', q \wedge q')$ iff $b \vDash q \wedge q'$ iff $b \vDash q$ and $b \vDash q'$ iff $\mathbb{i}_L(b) \vDash (p,q)$ and $\mathbb{i}_L(b) \vDash (p',q')$.
- Disjunction: $\mathbb{i}_L(b) \vDash (p \vee p', q \vee q')$ iff $b \vDash q \vee q'$ iff $b \vDash q$ or $b \vDash q'$ iff $\mathbb{i}_L(b) \vDash (p,q)$ or $\mathbb{i}_L(b) \vDash (p',q')$.
- Negation: $\mathbb{i}_L(b) \vDash (\neg p, \neg q)$ iff $b \vDash \neg q$ iff $b \nvDash q$ iff $\mathbb{i}_L(b) \nvDash (p,q)$

▷ Sugar Syntax: $(\mathsf{ok} : p) \vee (\mathsf{er} : q) = (p, \mathsf{false}) \vee (\mathsf{false}, q) = (p \vee \mathsf{false}, \mathsf{false} \vee q)$ iff (p, q).

□

E.3 Semantics of Programs

Recall the syntax of the atomic mGCL commands.

$$c \in \mathsf{mGCL} ::= \mathsf{assume}\ e \mid x := e \mid x := \mathsf{alloc}() \mid \mathsf{free}(e) \mid x \leftarrow [e] \mid [e_1] \leftarrow e_2 \mid \mathsf{error}()$$

The semantics $[\![c]\!] : \mathcal{S} \times \mathcal{H} \to M((\mathcal{S} \times \mathcal{H}) + (\mathcal{S} \times \mathcal{H}))$ is given below, parameterized by any monad M. Note that often the semantics of $\mathsf{alloc}()$ is nondeterministic and, in particular, it might reallocate some location ℓ such that $h(\ell) = \bot$. We have chosen to make the semantics fully deterministic so as to allow mGCL to be embedded into, for example, a probabilistic evaluation context.

$$[\![\mathsf{assume}\ e]\!](s,h) = \begin{cases} \mathsf{unit}_{\mathsf{er}}((s,h)) & \text{if } [\![e]\!](s) \neq 0 \\ \varnothing & \text{otherwise} \end{cases}$$

$$[\![x := e]\!](s,h) = \mathsf{unit}_{\mathsf{er}}((s[x \mapsto [\![e]\!](s)], h))$$

$$[\![x := \mathsf{alloc}()]\!](s,h) = \mathsf{unit}_{\mathsf{er}}((s[x \mapsto \ell], h[\ell \mapsto \mathsf{null}])) \text{ where } \ell = \max(\mathsf{dom}(h)) + 1$$

$$[\![\mathsf{free}(e)]\!](s,h) = \begin{cases} \mathsf{unit}_{\mathsf{er}}(s, h[\ell \mapsto \bot]) & \text{if } [\![e]\!](\sigma) = \ell \text{ and } \ell \in \mathsf{dom}(h) \text{ and } h(\ell) \neq \bot \\ \mathsf{unit}_M(\mathbb{i}_L((s,h))) & \text{if } [\![e]\!](\sigma) = \ell \text{ and } h(\ell) = \bot \\ \varnothing & \text{if } [\![e]\!](\sigma) \notin \mathsf{dom}(h) \end{cases}$$

$$[\![[e_1] \leftarrow e_2]\!](s,h) = \begin{cases} \mathsf{unit}_{\mathsf{er}}(s, h[\ell \mapsto [\![e_2]\!](\sigma)]) & \text{if } [\![e_1]\!](\sigma) = \ell \text{ and } \ell \in \mathsf{dom}(h) \text{ and } h(\ell) \neq \bot \\ \mathsf{unit}_M(\mathbb{i}_L((s,h))) & \text{if } [\![e_1]\!](\sigma) = \ell \text{ and } h(\ell) = \bot \\ \varnothing & \text{if } [\![e_1]\!](\sigma) \notin \mathsf{dom}(h) \end{cases}$$

$$[\![x \leftarrow [e]]\!](s,h) = \begin{cases} \mathsf{unit}_{\mathsf{er}}(s[x \mapsto h(\ell)], h) & \text{if } [\![e]\!](s) = \ell \text{ and } \ell \in \mathsf{dom}(h) \text{ and } h(\ell) \neq \bot \\ \mathsf{unit}_M(\mathbb{i}_L((s,h))) & \text{if } [\![e]\!](s) = \ell \text{ and } h(\ell) = \bot \\ \varnothing & \text{if } [\![e]\!](\sigma) \notin \mathsf{dom}(h) \end{cases}$$

$$[\![\mathsf{error}()]\!](s,h) = \mathsf{unit}_M(\mathbb{i}_L((s,h)))$$

We can define the usual semantics of $\mathsf{alloc}()$ if we specialize M to the powerset monad.

$$[\![x := \mathsf{alloc}()]\!](s,h) = \{\mathbb{i}_R((s[x \mapsto \ell], h[\ell \mapsto v])) \mid \ell \in \mathbb{N}^+, v \in \mathsf{Val}, \ell \notin \mathsf{dom}(h) \vee h(\ell) = \bot\}$$

E.4 Manifest Errors

Lemma 6.7 (Manifest Error Characterization).

$\vDash [p]\ C\ [\mathsf{er} : q]$ *is a manifest error* *iff* $\vDash^{\downarrow} \langle \mathsf{ok} : \mathsf{true}\rangle\ C\ \langle \mathsf{er} : q * \mathsf{true}\rangle$

Proof. First, recall that by definition, $\vDash [p]\ C\ [\mathsf{er} : q]$ is a manifest error iff $\forall \sigma.\ \exists \tau \in [\![C]\!](\sigma).\ \tau \vDash (\mathsf{er} : q * \mathsf{true})$.

($\Rightarrow$) Suppose that $S \vDash (\mathsf{ok} : \mathsf{true})$. This means that there must be a σ such that $\mathbb{i}_R(\sigma) \in S$. By the definition of manifest errors, we know that $\exists \tau \in [\![C]\!](\sigma)$ such that $\tau \vDash (\mathsf{er} : q * \mathsf{true})$. Now, since $[\![C]\!](\sigma) = [\![C]\!]^\dagger(\{\mathbb{i}_R(\sigma)\})$ and $\{\mathbb{i}_R(\sigma)\} \subseteq S$, then $\tau \in [\![C]\!]^\dagger(S)$ and so $[\![C]\!]^\dagger(S) \vDash^{\downarrow} (\mathsf{er} : q * \mathsf{true})$. Therefore, $\vDash^{\downarrow} \langle \mathsf{ok} : \mathsf{true}\rangle\ C\ \langle \mathsf{er} : q * \mathsf{true}\rangle$.

($\Leftarrow$) Let σ be any program state. From $\vDash^{\downarrow} \langle \mathsf{ok} : \mathsf{true}\rangle\ C\ \langle \mathsf{er} : q * \mathsf{true}\rangle$, we know that $[\![C]\!]^\dagger(\{\mathbb{i}_R(\sigma)\}) \vDash^{\downarrow} (\mathsf{er} : q * \mathsf{true})$. So, by Lemma B.4 there must be some $\tau \in [\![C]\!](\sigma)$ such that $\tau \vDash (\mathsf{er} : q * \mathsf{true})$ (since $[\![C]\!]^\dagger(\{\mathbb{i}_R(\sigma)\}) = [\![C]\!](\sigma)$).

□

E.5 The Compound Frame Rule

The Frame rule from Figure 5 can be used within a single outcome by applying Lifting rule. This is not very useful given that we would often want to add frames into large, compositional proofs.

To achieve this, we can add a *compound* frame rule that states that if we can frame some separation logic assertion r into every outcome in the precondition, then we can also frame it into every outcome in the postcondition:

$$\frac{\{\bigoplus_{i=1}^{n}(\epsilon_i : p_i)\}\ C\ \{\bigoplus_{j=1}^{m}(\epsilon'_j : q_j)\} \qquad \mathsf{fv}(r) \cap \mathsf{mod}(C) = \emptyset}{\{\bigoplus_{i=1}^{n}(\epsilon_i : p_i * r)\}\ C\ \{\bigoplus_{j=1}^{m}(\epsilon'_j : q_j * r)\}}\ \textsc{Compound Frame}$$

Proving that this rule is sound is a straightforward modification of the standard soundness proof for the Frame rule from separation logic. The only difference is that the pre- and postconditions are satisfied by sets of states. Unpacking this set allows us to use the induction hypothesis and then we can frame r back in to each outcome of the postcondition given that $\mathsf{fv}(r) \cap \mathsf{mod}(C) = \emptyset$.

F SOUNDNESS PROOFS

Lemma F.1 (Soundness of generic rules in Figure 4). *If* $\vdash \langle P \rangle\ C\ \langle Q \rangle$ *then* $\vDash \langle P \rangle\ C\ \langle Q \rangle$.

Proof. By induction on the derivation $\vdash \langle P \rangle\ C\ \langle Q \rangle$.

- Zero. Suppose that $m \vDash \varphi$. We know that $[\![\mathbb{0}]\!]^\dagger\ (m) = \varnothing$ and $\varnothing \vDash \top^\oplus$, therefore $\vDash \langle \varphi \rangle\ \mathbb{0}\ \langle \top^\oplus \rangle$
- One. Suppose that $m \vDash \varphi$. The know that $[\![\mathbb{1}]\!]^\dagger\ (m) = m$ and we assumed that $m \vDash \varphi$, so $\vDash \langle \varphi \rangle\ \mathbb{1}\ \langle \varphi \rangle$
- Seq. Suppose that $m \vDash \varphi$. By induction, we know that $[\![C_1]\!]^\dagger\ (m) \vDash \psi$. By induction again, we know that $[\![C_2]\!]^\dagger\ ([\![C_1]\!]^\dagger\ (m)) \vDash \vartheta$. In addition:

$$\begin{aligned}
[\![C_2]\!]^\dagger([\![C_1]\!]^\dagger(m)) &= \mathsf{bind}([\![C_1]\!]^\dagger(m), [\![C_2]\!]) \\
&= \mathsf{bind}(\mathsf{bind}(m, [\![C_1]\!]), [\![C_2]\!]) \\
&= \mathsf{bind}(m, \lambda\sigma.\mathsf{bind}([\![C_1]\!]\ (\sigma), [\![C_2]\!])) \\
&= \mathsf{bind}(m, [\![C_1 \fatsemi C_2]\!]) \\
&= [\![C_1 \fatsemi C_2]\!]^\dagger(m)
\end{aligned}$$

  So, $[\![C_1 \fatsemi C_2]\!]^\dagger(m) \vDash \vartheta$ and therefore $\vDash \langle \varphi \rangle\ C_1 \fatsemi C_2\ \langle \vartheta \rangle$
- For. Since for N *do* C is syntactic sugar for C^n (or, equivalently, $C \fatsemi \cdots \fatsemi C$), this rule can be derived by induction on N using the Seq use.
- Split. Suppose $m \vDash \varphi_1 \oplus \varphi_2$, then there exists m_1 and m_2 such that $m_1 \diamond m_2 = m$ and $m_1 \vDash \varphi_1$ and $m_2 \vDash \varphi_2$. By induction, we know that $[\![C]\!]^\dagger(m_1) \vDash \psi_1$ and $[\![C]\!]^\dagger(m_2) \vDash \psi_2$. By linearity, we know that $[\![C]\!]^\dagger(m_1) \diamond [\![C]\!]^\dagger(m_2) = [\![C]\!]^\dagger(m_1 \diamond m_2) = [\![C]\!]^\dagger(m)$. Note that this does not necessarily mean that $[\![C]\!]^\dagger(m)$ is defined, but if we limit C to be syntactically valid (as described in Appendix A), then it must be defined and so $[\![C]\!]\ (m) \vDash \psi_1 \oplus \psi_2$
- Consequence. Suppose that $m \vDash \varphi'$. By the assumption that $\varphi' \Rightarrow \varphi$, this means that $m \vDash \varphi$. By induction, we know that $[\![C]\!]^\dagger(m) \vDash \psi$ and so $[\![C]\!]^\dagger(m) \vDash \psi'$ (since $\psi \Rightarrow \psi'$) and therefore $\vDash \langle \varphi' \rangle\ C\ \langle \psi' \rangle$
- Empty. Suppose that $m \vDash \top^\oplus$, then $m = \varnothing$. We also know that $\mathsf{bind}(\varnothing, f) = \varnothing$ for any f, so $[\![C]\!]^\dagger(\varnothing) = \varnothing$, therefore $[\![C]\!]^\dagger(m) \vDash \top^\oplus$.
- True. Suppose that $m \vDash \varphi$. It is trivially true that $[\![C]\!]^\dagger(m) \vDash \top$.
- False. The premise that $m \vDash \bot$ is impossible, therefore this case vacuously holds.

□

Lemma F.2 (Soundness of nondeterministic rules in Figure 4). *If* $\vdash \langle P \rangle\ C\ \langle Q \rangle$ *then* $\vDash \langle P \rangle\ C\ \langle Q \rangle$.

Proof. By induction on the derivation $\vdash \langle P \rangle\ C\ \langle Q \rangle$.

- Plus. Suppose that $m \vDash \varphi$. By induction, we know that $[\![C_1]\!]^\dagger(m) \vDash \psi_1$ and $[\![C_2]\!]^\dagger(m) \vDash \psi_2$. By the definition of $[\![-]\!]$ we also know that $[\![C_1]\!]^\dagger(m) \cup [\![C_2]\!]^\dagger(m) = [\![C_1 + C_2]\!]^\dagger(m)$ and therefore $[\![C_1 + C_2]\!]^\dagger(m) \vDash \psi_1 \oplus \psi_2$.
- Induction. Suppose that $m \vDash \varphi$. We know by induction that $[\![\mathbb{1} + C \fatsemi C^\star]\!]^\dagger(m) \vDash \psi$. Let $F = \lambda f.\lambda\sigma.f^\dagger([\![C]\!](\sigma)) \cup \mathsf{unit}(\sigma)$ and note that:

$$\begin{aligned}
[\![\mathbb{1} + C \fatsemi C^\star]\!]^\dagger(m) &= [\![\mathbb{1}]\!]^\dagger(m) \cup [\![C \fatsemi C^\star]\!]^\dagger(m) \\
&= m \cup [\![C^\star]\!]^\dagger([\![C]\!]^\dagger(m))) \\
&= m \cup \bigcup_{n \in \mathbb{N}} F^n(\lambda x.\varnothing)^\dagger([\![C]\!]^\dagger(m)) \\
&= F(\lambda x.\varnothing)^\dagger(m) \cup \bigcup_{n \geq 1} F^n(\lambda x.\varnothing)^\dagger(m) \\
&= [\![C^\star]\!]^\dagger(m)
\end{aligned}$$

  So, $[\![C^\star]\!]^\dagger(m) \vDash \psi$.

□

Lemma F.3 (Soundness of expression-based rules in Figure 4). *If* $\vdash \langle P \rangle\ C\ \langle Q \rangle$ *then* $\vDash \langle P \rangle\ C\ \langle Q \rangle$.

Proof. By induction on the derivation $\vdash \langle P \rangle\ C\ \langle Q \rangle$.

- Assume. Suppose that $m \vDash P_1 \oplus P_2$. Since $P_1 \vDash e$ and $P_2 \vDash \neg e$, we know by the definition of expression entailment that that $[\![\mathsf{assume}\ e]\!]^\dagger(m) \vDash P$.
- Assign. Suppose that $m \vDash P[e/x]$. By the required properties of substitution, we know that $[\![x := e]\!]^\dagger(m) \vDash P$.
- If. Suppose that $m \vDash P_1 \oplus P_2$. Now observe that:

$$\begin{aligned}
[\![\mathsf{if}\ e\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2]\!]^\dagger(m) &= [\![(\mathsf{assume}\ e \fatsemi C_1) + (\mathsf{assume}\ \neg e \fatsemi C_2)]\!]^\dagger(m) \\
&= [\![\mathsf{assume}\ e \fatsemi C_1]\!]^\dagger(m) \diamond [\![\mathsf{assume}\ \neg e \fatsemi C_2]\!]^\dagger(m) \\
&= [\![C_1]\!]^\dagger([\![\mathsf{assume}\ e]\!]^\dagger(m)) \diamond [\![C_2]\!]^\dagger([\![\mathsf{assume}\ \neg e]\!]^\dagger(m))
\end{aligned}$$

  Now, let $m_1 = [\![\mathsf{assume}\ e]\!]^\dagger(m)$ and $m_2 = [\![\mathsf{assume}\ \neg e]\!]^\dagger(m)$. Since we know that $P_1 \vDash e$ and $P_2 \vDash \neg e$ and $m \vDash P_1 \oplus P_2$, then $m_1 \vDash P_1$ and $m_2 \vDash P_2$ (by the required properties of expression entailment).

$$= [\![C_1]\!]^\dagger(m_1) \diamond [\![C_2]\!]^\dagger(m_2)$$

  By the induction hypotheses, we also know that $[\![C_1]\!]^\dagger(m_1) \vDash Q_1$ and $[\![C_2]\!]^\dagger(m_2) \vDash Q_2$, therefore $[\![C_1]\!]^\dagger(m_1) \diamond [\![C_2]\!]^\dagger(m_2) \vDash Q_1 \oplus Q_2$. Note that this composition with $\diamond$ is valid in all the execution models we have presented since $\cup$ is total and we have already shown that $+$ on distributions is defined in the semantics of if statements.

□

Lemma F.4 (Soundess of Nondeterministic Lifting Rule). *The following inference rule is sound.*

$$\frac{\vdash_{2^{(-)}} \langle p \rangle\ C\ \langle q \rangle}{\langle p \rangle\ C\ \langle q \rangle}\ \textit{Nondeterministic Lift}$$

Proof. By induction on the derivation $\vdash \langle p \rangle\ C\ \langle q \rangle$. Suppose that $S \vDash p$, so that means that $S \neq \emptyset$ and $\forall \sigma \in S.\ \sigma \vDash p$. We know by induction that for any $\sigma \vDash p$, there is some τ such that $[\![C]\!]^\dagger(\{\sigma\}) = \{\tau\}$ and $\tau \vDash q$. We also know that $[\![C]\!]^\dagger(S) = \bigcup_{\sigma \in S} [\![C]\!]^\dagger(\{\sigma\})$ and since each for each σ, there is a τ such that $[\![C]\!]^\dagger(\{\sigma\}) = \{\tau\}$, then $\forall \tau \in [\![C]\!]^\dagger(S)$, $\tau \vDash q$ and so $[\![C]\!]^\dagger(S) \vDash q$ □

Lemma F.5 (Soundness of Error Propagation). *The following inference rule is sound:*

$$\frac{}{\vdash_M \langle er : p \rangle\ C\ \langle er : p \rangle}\ \textit{Error Propagation}$$

Proof. Suppose that $m \vDash (\mathsf{er} : p)$, and so there must be some σ such that $m = \mathbb{i}_L(\sigma)$ and $\mathbb{i}_L(\sigma) \vDash (\mathsf{er} : p)$. Now, we have:

$$\begin{aligned}
[\![C]\!]^\dagger(\mathsf{unit}_M(\mathbb{i}_L(\sigma))) &= \mathsf{bind}_M\left(\mathsf{unit}_M(\mathbb{i}_L(\sigma)), \lambda x. \begin{cases} [\![C]\!](y) & \text{if } x = \mathbb{i}_R(y) \\ \mathsf{unit}_M(x) & \text{if } x = \mathbb{i}_L(y) \end{cases}\right) \\
&= \left(\lambda x. \begin{cases} [\![C]\!](y) & \text{if } x = \mathbb{i}_R(y) \\ \mathsf{unit}_M(x) & \text{if } x = \mathbb{i}_L(y) \end{cases}\right)(\mathbb{i}_L(\sigma)) \\
&= \mathsf{unit}_M(\mathbb{i}_L(\sigma))
\end{aligned}$$

And since we already know that $\mathbb{i}_L(\sigma) \vDash (\mathsf{er} : p)$, we are done. □

Lemma F.6 (Soundness of Probabilistic Proof System). *The inference rules at the top of Figure 7 are sound.*

Proof. By induction on the derivation $\vdash \langle P \rangle\ C\ \langle Q \rangle$

- ▹ Lifting. Suppose that $\mu \vDash (\mathbb{P}[A] = p)$, so for every $\sigma \in \mathrm{supp}(\mu)$, $\sigma \vDash A$ and $|\mu| = p$. We know by induction that for any σ there is some τ_σ such that $[\![C]\!](\sigma) = \delta_{\tau_\sigma}$ and $\tau_\sigma \vDash B$. So, $\mu' = [\![C]\!]^\dagger(\mu) = \sum_{\sigma \in \mathrm{supp}(\mu)} \mu(\sigma) \cdot [\![C]\!](\sigma) = \sum_{\sigma \in \mathrm{supp}(\mu)} \mu(\sigma) \cdot \delta_{\tau_\sigma}$. Therefore $|\mu'| = |\mu|$ and $\forall \tau \in \mathrm{supp}(\mu')$, $\tau \vDash B$, so $\mu' \vDash (\mathbb{P}[B] = p)$.
- ▹ Sample. First, observe that:

$$\begin{aligned}
[\![x \xleftarrow{\$} \eta]\!]^\dagger(\mu) &= \mathsf{bind}(\mu, \lambda\sigma.\mathsf{bind}(\eta, \lambda v.\, [\![x := v]\!](\sigma))) \\
&= \sum_{\sigma \in \mathrm{supp}(\mu)} \mu(\sigma) \cdot \sum_{v \in \mathrm{supp}(\eta)} \eta(v) \cdot [\![x := v]\!](\sigma) \\
&= \sum_{v \in \mathrm{supp}(\eta)} \eta(v) \cdot \sum_{\sigma \in \mathrm{supp}(\mu)} \mu(\sigma) \cdot [\![x := v]\!](\sigma) \\
&= \sum_{v \in \mathrm{supp}(\eta)} \eta(v) \cdot [\![x := v]\!]^\dagger(\mu)
\end{aligned}$$

  Now, by the same argument that we used in the lifting cases, since $\mu \vDash (\mathbb{P}[A] = p)$ and $\langle A \rangle\ x := v\ \langle B_v \rangle$, then $[\![x := v]\!]^\dagger(\mu) \vDash (\mathbb{P}[B_v] = p)$. Therefore, we we can also weight the distribution to obtain $\eta(v) \cdot [\![x := v]\!]^\dagger(\mu) \vDash (\mathbb{P}[B_v] = \eta(v) \cdot p)$. Now, the sum over $v \in \mathrm{supp}(\eta)$ corresponds exactly to an outcome conjunction, so we have:

$$\sum_{v \in \mathrm{supp}(\eta)} \eta(v) \cdot [\![x := v]\!]^\dagger(\mu) \quad \vDash \quad \bigoplus_{v \in \mathrm{supp}(\eta)} (\mathbb{P}[B_v] = \eta(v) \cdot p)$$

□

Lemma F.7 (Correctness of Expression Entailment). *If $m \vDash P \oplus Q$ and $P \vDash e$ and $Q \vDash \neg e$, then $[\![\mathsf{assume}\ e]\!]^\dagger(m) \vDash P$ for both the nondeterministic and probabilistic interpretations of expression entailment*

Proof.

- Nondeterminism. First note that $[\![\text{assume } e]\!]^\dagger(S) = \{\sigma \mid \sigma \in S, [\![e]\!]_{\text{Exp}}(\sigma) = \text{true}\}$. In addition, $m \vDash P \oplus Q$ means that there are nonempty sets S_1 and S_2 such that $S_1 \cup S_2 = S$ and $S_1 \vDash P$ and $S_2 \vDash Q$. Depending on which atomic assertions we are using, P is either some assertion p or $(\mathsf{ok} : p)$, in either case, we know from $P \vDash e$ that $p \Rightarrow e$. We know that every state in S_1 satisfies p (and therefore also e), so $[\![\text{assume } e]\!]^\dagger(S_1) = S_1$. By a similar argument, $[\![\text{assume } e]\!]^\dagger(S_2) = \emptyset$. Therefore $[\![\text{assume } e]\!]^\dagger(S) = S_1 \cup \emptyset = S_1$ and we already know that $S_1 \vDash P$.
- Probabilistic. The semantics of assume are similar in this case; states are filtered from the support that do not agree with e and the distribution is otherwise left unchanged. Now suppose that $\mu \vDash (\mathbb{P}[A] = p) \oplus (\mathbb{P}[B] = q)$ and therefore $\mu_1 \vDash (\mathbb{P}[A] = p)$ and $\mu_2 \vDash (\mathbb{P}[B] = q)$ such that $\mu_1 + \mu_2 = \mu$. All states in $\text{supp}(\mu_1)$ satisfy A (and therefore also e since $A \vDash e$), so $[\![\text{assume } e]\!]^\dagger(\mu_1) = \mu_1$. The opposite is true for μ_2, so $[\![\text{assume } e]\!]^\dagger(\mu_2) = \varnothing$. Therefore $[\![\text{assume } e]\!]^\dagger(\mu) = \mu_1 + \varnothing = \mu_1$ and we already know that $\mu_1 \vDash (\mathbb{P}[A] = p)$.

□

G ADDITIONAL RULES FOR CONDITIONALS AND LOOPS

As mentioned in Section 4.3, fully generic looping rules for OL that work with all instances of the logic are not possible because different instances have different constraints when it comes to termination. However, it is possible to create an *under-approximate* rule that unrolls a loop for a bounded number of iterations:

$$\frac{\forall i.\ P_i \vDash e \qquad \forall i.\ Q_i \vDash \neg e \qquad \forall i.\ \langle P_i \rangle\ C\ \langle P_{i+1} \oplus Q_{i+1} \rangle}{\langle P_0 \oplus Q_0 \rangle\ \text{while } e \text{ do } C\ \langle (\bigoplus_{i=0}^{n} Q_i) \oplus \top \rangle}\ \textsc{Bounded Unrolling}$$

In this rule, P_i is the outcome of running C i times with the guard remaining true and similarly Q_i is the outcome of running C i times with the guard becoming false. The true components (P_i) are passed forward into the next iteration whereas the false components that cause the loop to exit (Q_i) are joined to the postcondition.

This rule avoids the termination question entirely by only looking at finite executions, the remainder of outcomes are covered by $\top$. Similar to the conditional rules seen in Figure 4, it requires you to separate assertions into components that are "true" and "false" with respect to the loop guard e. This may not be possible in nondeterministic settings, as the loop body C may only produce one outcome. It is, however, suitable for probabilistic applications so long as the probability of the loop guard is known (probabilistic assertions can always be split by probability mass).

For nondeterministic proof systems where we may not be able to split assertions into multiple outcomes, we can create specialized loops rules. Such a rule for the separation logic proof system is given below:

$$\frac{\forall i > 0.\ (p_i \Rightarrow e) \wedge \epsilon_i = \mathsf{ok} \qquad \forall i \in \mathbb{N}.\ \langle \epsilon_{i+1} : p_{i+1} \rangle\ C\ \langle \epsilon_i : p_i \rangle \qquad (p_0 \Rightarrow \neg e) \vee (\epsilon_0 = \mathsf{er})}{\langle \epsilon_n : p_n \rangle\ \text{while } e \text{ do } C\ \langle \epsilon_0 : p_0 \rangle}\ \textsc{While}$$

This rule is very similar to the rule for loops from Total Hoare Logic [Apt 1981] with the addition that the postcondition may not imply that e is false if the program has crashed. Similarly, we can formulate the familiar conditional rule that operates within a single outcome:

$$\frac{\langle \mathsf{ok} : p \wedge e \rangle\ C_1\ \langle Q \rangle \qquad \langle \mathsf{ok} : p \wedge \neg e \rangle\ C_2\ \langle Q \rangle}{\langle \mathsf{ok} : p \rangle\ \text{if } e \text{ then } C_1 \text{ else } C_2\ \langle Q \rangle}\ \textsc{If (Single Outcome)}$$