Monoidal Hoare 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 Monoidal Hoare Logic (MHL), a novel generalization of Hoare Logic that is both monadic (to capture computational effects) and monoidal (to reason about outcomes and reachability). MHL is guaranteed to find true bugs, while retaining correctness reasoning abilities as well. To formalize the applicability of MHL to both correctness and incorrectness, we prove that any false MHL specification can be disproven in MHL itself. We also use our framework to reason about new types of incorrectness in non-deterministic and probabilistic programs. Given these advances, we advocate for MHL as a new foundational theory of correctness and incorrectness.

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

Hoare Logic: $\vdash \{P\} C \{Q\}$ iff $\forall \tau \vdash P. \forall \sigma. \tau \in [C] (\sigma) \Rightarrow \tau \vdash Q$

Incorrectness Logic: $\vdash [P] C [Q]$ iff $\forall \tau \vdash Q. \exists \sigma. \tau \in [C] (\sigma)$ and $\sigma \vdash P$

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.

2022. XXXX-XXXX/2022/10-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

, Vol. 1, No. 1, Article . Publication date: October 2022.
Despite these exciting advances, we argue that the foundations of incorrectness reasoning are still far from settled—and worthy of reconsideration. IL achieves \textit{true positives} (reachability of end-states) and \textit{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 §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 \textit{reachability} of bad states, but it cannot prove \textit{non-reachability} of good states.
- IL is not amenable to probabilistic execution models and therefore cannot be used to reason about incorrectness in randomized programs.
- IL cannot easily describe what conditions are \textit{sufficient} to trigger a bug, 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 in a unified framework: a generalization of “good old” Hoare Logic that we call \textbf{Monoidal Hoare Logic (MHL)}. In addition to consolidating the foundations of incorrectness with traditional correctness reasoning, MHL overcomes all the aforementioned drawbacks of IL.

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

MHL offers several advantages as a unifying foundation for correctness and incorrectness:

\textbf{Generality.} MHL unifies program analysis across two dimensions. First, since any untrue MHL spec can be disproven in MHL (Theorem 5.1), correctness and incorrectness reasoning are possible in a single program logic. Second, MHL 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.

\textbf{Beyond Reachability.} Until now, the study of incorrectness has revolved primarily around \textit{reachability} of crash states. We prove that MHL handles a broader characterization of incorrectness than IL in nondeterministic programs (Theorem 5.4), as well as probabilistic incorrectness (Theorem 5.6).

\textbf{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 \textit{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 MHL—Le et al.’s [2022] original definition can be expressed as an MHL triple (Lemma 6.6).

In a nutshell, 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 (§2). We show how reasoning about outcomes can account for reachability of end-states and enable under-approximation (when desired).
Fig. 1. Semantics of triples where \( P \) and \( Q \) are logical formulae, \( C \) is a program, \( \Sigma \) is the set of all program states, \( \sigma, \tau \in \Sigma \), and \( \llbracket C \rrbracket : \Sigma \rightarrow 2^\Sigma \) is the reachable states function. In the last line of the table, \( M \) is a monad, \( m \in M\Sigma \) and \( \llbracket C \rrbracket : M\Sigma \rightarrow M\Sigma \) is the monadic lifting of \( \llbracket \cdot \rrbracket : \Sigma \rightarrow M\Sigma \).

- We define Monoidal Hoare Logic formally (§3 and §4), parametric on a monad and an outcome assertion logic. We define syntax and semantics of the logic, using Bunched Implications (BI) formulae for pre- and postconditions. We provide inference rules to reason about validity.
- We show that MHL is suitable for both correctness and incorrectness reasoning by proving that false MHL triples can be disproven within MHL (§5). As a corollary, MHL can also disprove standard Hoare triples, which was one of the motivations for IL (Corollary 5.5). We go further and show three kinds of incorrectness that can be captured in MHL, only one of which is expressible in IL (§5.1).
- We exemplify how MHL can be instantiated to find memory errors (§6) and probabilistic bugs (§7). We argue that the latter use case is not possible in IL (§7.2).

Finally, we conclude in §8 and §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 \( x := \text{malloc}() ; [x] \leftarrow 1 \) has a bug (malloc may nondeterministically return null, causing the program to crash with a segmentation fault). We might be tempted to specify the triple:

\[
\{\text{true}\} \ x := \text{malloc}() ; [x] \leftarrow 1 \ ( (\text{ok} : x \mapsto 1) \lor (\text{er} : x = \text{null}) )
\]

Here, the assertion \( \text{(ok : p)} \) states that the program terminated successfully in a state satisfying \( p \) and \( \text{(er : q)} \) states 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 \( \text{(ok : x \mapsto 1)} \) 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.

\[
[\text{true}] \ x := \text{malloc}() ; [x] \leftarrow 1 \ ( (\text{ok} : x \mapsto 1) \lor (\text{er} : x = \text{null}) )
\]

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:

\[
[\text{true}] \ x := \text{malloc}() ; [x] \leftarrow 1 \ [\text{er} : x = \text{null}]
\]
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 Monoidal Hoare Logic (MHL), 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:

\[
\text{monadic semantics } \quad \boxplus \quad \text{semantics of } \boxplus \text{ uses monoid composition } \diamond
\]

MHL triples follow the spirit of Hoare Logic—first quantifying over elements satisfying the pre-condition and then stipulating that the result of running the program on such an element must satisfy the postcondition. The difference is that in MHL triples, the pre- and postconditions are satisfied by a monoidal collection of outcomes \( m \in MS \) rather than individual program states \( \sigma \in \Sigma \). This allows us to introduce a new connective in the logic—the outcome conjunction \( \boxplus \)—which models program outcomes as resources. Consider the postcondition in triple (2) if we replace \( \lor \) by \( \boxplus \):

\[
(\text{ok} : x \mapsto 1) \lor (\text{er} : x = \text{null}) \quad \text{vs.} \quad (\text{ok} : x \mapsto 1) \oplus (\text{er} : x = \text{null})
\]

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 \( MS \) is a (partial commutative) monoid. Given two outcomes \( m_1, m_2 \in MS \), there is an operation \( \diamond \) that enables us to combine them \( m_1 \diamond m_2 \in MS \). The satisfiability of \( \oplus \) is then defined using \( \diamond \) to split the monoidal state:

\[
m \models P \oplus Q \quad \text{iff} \quad \exists m_1, m_2 \in MS. \quad m = m_1 \diamond m_2 \quad \text{and} \quad m_1 \models P \quad \text{and} \quad m_2 \models Q
\]

Consider instantiating the above to the powerset monad that associates a set \( A \) with the set of its subsets \( 2^A \). Given a semantic function \( \llbracket C \rrbracket : \Sigma \rightarrow 2^\Sigma \) that maps individual start states \( \sigma \) to the set of final states reachable by executing \( C \), we can give a monadic semantics \( \llbracket C \rrbracket^- (S) = \bigcup_{\sigma \in S} \llbracket C \rrbracket^- (\sigma) \) where \( S \) is a set of start states.\(^1\) The monoid composition \( \diamond \) on \( 2^A \) is given by set union, which is used compositionally to define satisfiability of \( \oplus \) as follows: \( S \models P \oplus Q \iff S_1 \models P \) and \( S_2 \models Q \) such

\(^1\) The \([\cdot]^-\) function is formally the monadic (or Kleisli) extension of \([\cdot]^-\); we will define this formally in §3.
that $S = S_1 \cup S_2$. Given some satisfaction relation for individual program states $\vdash_{\Sigma} \subseteq \Sigma \times \text{Prop}$, we then define satisfaction of atomic assertions as follows:

$$S \vdash P \quad \text{iff} \quad S \neq \emptyset \quad \text{and} \quad \forall \sigma \in S. \sigma \vdash_{\Sigma} P$$

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

$$\vdash \langle P \rangle C \langle Q \rangle \quad \text{iff} \quad \forall S \in 2^\Sigma. S \vdash P \Rightarrow [C]^\downarrow(S) \nvdash Q$$

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

$$\langle \text{ok : true} \rangle x := \text{malloc()} \; ; \; [x] \leftarrow 1 \; \langle (\text{ok : } x \mapsto 1) \oplus (\text{er : } x = \text{null}) \rangle$$

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:

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

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

$$\vdash^\downarrow \langle P \rangle C \langle Q \rangle \quad \text{iff} \quad \vdash \langle P \rangle C \langle Q \oplus \top \rangle$$

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

$$\vdash^\downarrow \langle \text{ok : true} \rangle x := \text{malloc()} \; ; \; [x] \leftarrow 1 \; \langle \text{er : } x = \text{null} \rangle$$

This example demonstrates that MHL 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. We will show in §2.2 that there are bugs expressible in MHL that cannot be expressed in IL. In §2.3 we will also explain why the semantics of MHL are a better fit for characterizing an important class of bugs known as manifest errors.

### 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 MHL 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 first will 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 $a$:

$$[\text{true}] b := \text{shuffle}(a) \; [b \in \Pi(a)]$$
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 typically 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]. MHL does just that—the full correctness of the shuffle program can be captured using a single MHL triple that guarantees reachability *without* under-approximating:

\[
\langle \text{true} \rangle \ b := \text{shuffle} \ (a) \bigoplus_{\pi \in \Pi(a)} \langle b = \pi \rangle
\]

The MHL specification above states *not only* that all the permutations are reachable, *but also* that they are the only possible outcomes. So, MHL 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 MHL 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 MHL triples:

\[
\exists \pi \in \Pi(a). \langle \text{true} \rangle \ b := \text{shuffle} \ (a) \ (b \neq \pi) \quad \langle \text{true} \rangle \ b := \text{shuffle} \ (a) \ (b \notin \Pi(a)) \oplus \top
\]

In addition, both of these triples 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 5.4:

\[
\forall P \ C \ \bigoplus_{i=1}^{n} Q_i \iff \exists P' \Rightarrow P. \ \exists i. \ V. \langle P' \rangle \ C \ (\neg Q_i) \text{ or } \forall P' \ C \ \bigoplus_{i=1}^{n} (\neg Q_i) \oplus \top
\]

Intuitively, a nondeterministic program is incorrect iff either one of the desired outcomes never occurs or some undesirable outcome sometimes occurs. In general, there is also a third option: the program diverges (has no outcomes). See Theorem 5.4.

**2.3 The Semantics of Incorrectness**

In addition to enabling us to witness a larger class of incorrectness than IL (unreachable states and probabilistic incorrectness), MHL 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 §2.1, both IL triples and MHL 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 MHL 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.
We start by defining a programming language, inspired by Dijkstra’s guarded command language 
we will interpret the syntax in a semantic model that is parametric on a monad and a partial

\[ \langle M, \text{bind}, \text{unit} \rangle \]

Typical examples of monads include powerset, error, and distribution monads (defined in §5 and

\[ \langle S, \circ, \emptyset \rangle \]

\[ \text{bind}(m, \text{unit}) = m \]

(2) \[ \forall a \in A. \text{bind}(\text{unit}(a), f) = f(a) \]

(3) \[ \text{bind}(\text{bind}(m, f))(g) = \text{bind}(m, \lambda a. \text{bind}(f(a), g)) \]

Typical examples of monads include powerset, error, and distribution monads (defined in §5 and

\[ \langle S, \circ, \emptyset \rangle \]

A partial commutative monoid (PCM) is a triple

\[ \langle S, \circ, \emptyset \rangle \]

\[ \text{bind}(m, \circ n) = \text{bind}(m, \circ n) \]

A typical example of a PCM, used in probabilistic reasoning, is \[ \langle [0, 1], +, 0 \rangle \] (the + is partial when

\[ \text{bind}(\emptyset, k) = \emptyset \]

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 0, which represents divergence, 1, acting

\[ \langle M, \text{bind}, \text{unit} \rangle \]

unit: \[ \text{Id} \Rightarrow M \] is a natural transformation, and \[ \text{bind}: MA \times (A \rightarrow MB) \rightarrow MB \] satisfies:

(1) \[ \text{bind}(m, \text{unit}) = m \]

(2) \[ \forall a \in A. \text{bind}(\text{unit}(a), f) = f(a) \]

(3) \[ \text{bind}(\text{bind}(m, f))(g) = \text{bind}(m, \lambda a. \text{bind}(f(a), g)) \]

Typical examples of monads include powerset, error, and distribution monads (defined in §5 and

\[ \langle S, \circ, \emptyset \rangle \]

A partial commutative monoid (PCM) is a triple

\[ \langle S, \circ, \emptyset \rangle \]

\[ \text{bind}(m, \circ n) = \text{bind}(m, \circ n) \]

A typical example of a PCM, used in probabilistic reasoning, is \[ \langle [0, 1], +, 0 \rangle \] (the + is partial when

\[ \text{bind}(\emptyset, k) = \emptyset \]

3 A MODULAR PROGRAMMING LANGUAGE

We start by defining a programming language, inspired by Dijkstra’s guarded command lan-

\[ \langle M, \text{bind}, \text{unit} \rangle \]

unit: \[ \text{Id} \Rightarrow M \] is a natural transformation, and \[ \text{bind}: MA \times (A \rightarrow MB) \rightarrow MB \] satisfies:

(1) \[ \text{bind}(m, \text{unit}) = m \]

(2) \[ \forall a \in A. \text{bind}(\text{unit}(a), f) = f(a) \]

(3) \[ \text{bind}(\text{bind}(m, f))(g) = \text{bind}(m, \lambda a. \text{bind}(f(a), g)) \]

Typical examples of monads include powerset, error, and distribution monads (defined in §5 and

\[ \langle S, \circ, \emptyset \rangle \]

A partial commutative monoid (PCM) is a triple

\[ \langle S, \circ, \emptyset \rangle \]

\[ \text{bind}(m, \circ n) = \text{bind}(m, \circ n) \]

A typical example of a PCM, used in probabilistic reasoning, is \[ \langle [0, 1], +, 0 \rangle \] (the + is partial when

\[ \text{bind}(\emptyset, k) = \emptyset \]

3 A MODULAR PROGRAMMING LANGUAGE

We start by defining a programming language, inspired by Dijkstra’s guarded command lan-

\[ \langle M, \text{bind}, \text{unit} \rangle \]

unit: \[ \text{Id} \Rightarrow M \] is a natural transformation, and \[ \text{bind}: MA \times (A \rightarrow MB) \rightarrow MB \] satisfies:

(1) \[ \text{bind}(m, \text{unit}) = m \]

(2) \[ \forall a \in A. \text{bind}(\text{unit}(a), f) = f(a) \]

(3) \[ \text{bind}(\text{bind}(m, f))(g) = \text{bind}(m, \lambda a. \text{bind}(f(a), g)) \]

Typical examples of monads include powerset, error, and distribution monads (defined in §5 and

\[ \langle S, \circ, \emptyset \rangle \]

A partial commutative monoid (PCM) is a triple

\[ \langle S, \circ, \emptyset \rangle \]

\[ \text{bind}(m, \circ n) = \text{bind}(m, \circ n) \]

A typical example of a PCM, used in probabilistic reasoning, is \[ \langle [0, 1], +, 0 \rangle \] (the + is partial when

\[ \text{bind}(\emptyset, k) = \emptyset \]
In Figure 2 we present the semantics of the language. The monad operations are used to provide semantics to $\mathcal{I}$ and sequential composition $;$ 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 $\circ$ is partial. The partiality of $\circ$ 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 3.1 (State and Guarded Commands). The base language introduced in the previous section is parametric over a set of program states $\Sigma$. 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 $e \in \text{Exp}$ which includes variables $x \in \text{Var}$ as well as the typical Boolean and arithmetic operators. Atomic commands come from the following syntax.

$$c ::= \begin{array}{ll} & e \\ \text{x} & : & e & \quad (x \in \text{Var}, e \in \text{Exp}) \end{array}$$

The command $c$ does nothing if $e$ is true and eliminates the current outcome if not. The syntax $x := e$ is standard variable assignment. A program stack is a mapping from variable names to values $S = \{s \mid s : \text{Var} \rightarrow \text{Val}\}$ where $\text{Val} = \mathbb{Z} + \mathbb{B}$ and $\mathbb{B} = \{\text{true}, \text{false}\}$. Given a stack, we can evaluate expressions to values using $[e]_{\text{Exp}} : S \rightarrow \text{Val}$. The semantics of these atomic commands $[c] : \Sigma \rightarrow MS$ is defined below.

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

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.

$$\begin{array}{ll} & \text{if } e \text{ then } C_1 \text{ else } C_2 = \text{assume } e ; C_1 + \text{assume } \neg e ; C_2 \quad \text{skip} = \mathcal{I} \quad \text{end} = \mathcal{I} \\ & \text{while } e \text{ do } C = \text{assume } e ; C^\star ; \text{assume } \neg e \quad \text{for } N \text{ do } C = C^N \quad C^{k+1} = C \circ C^k \end{array}$$

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

4 MONOIDAL HOARE LOGIC

In this section, we formally define Monoidal Hoare Logic (MHL). We first define the logic of outcome assertions which will act as the basis for writing pre- and postconditions in MHL. Next, we give
We will now give a formal account of the outcome logic that was briefly described in §2.1. (Definition 4), uppercase metavariables $P \lesssim$ (PCM) and program logics. This idea is explored in Appendix B.1. 3

We mentioned one example of a PCM in §2: Example 4.1 (Outcomes). The semantic 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.

4.1 A Logic for Monoidic Assertions: Modeling the Outcome Conjunction

We will now give a formal account of the outcome logic that was briefly described in §2.1. The outcome 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.

Our outcome logic is based on the formulation of BI due to Docherty [2019]. The syntax and semantics are given in Figure 3 with logical negation $\neg \phi$ being defined as $\phi \Rightarrow \bot$. To define the semantics of the logic we use a BI frame $\langle X, \circ, \ll, \emptyset \rangle$ where $\langle X, \circ, \emptyset \rangle$ is a partial commutative monoid (PCM) and $\ll \subseteq X \times X$ is a preorder on $X$. The semantics is also parametric on a satisfaction relation for basic assertions $\models_{atom} \subseteq X \times \text{Prop}$.

The two non-standard additions are the outcome conjunction $\circ$, a connective to specify multiple outcomes, and $\top^{\circ}$, an assertion to specify that there are no outcomes. These intended meanings are reflected in the semantics: $\top^{\circ}$ is only satisfied by the monoid unit $\emptyset$ whereas $\phi \circ \psi$ is satisfied by a monoid element $m$ iff $m$ can be partitioned into $m_1 \circ m_2$ to satisfy each outcome formula separately. We will focus on classical interpretations of BI where the preorder $\ll$ is equality.

Definition 4 (Outcome Logic). Given an execution model $\langle M, \text{bind}, \text{unit}, \circ, \emptyset \rangle$ and a satisfaction relation for atomic assertions $\models_{atom} \subseteq M \Sigma \times \text{Prop}$, an Outcome Logic is an instance of BI based on the BI frame $\langle M \Sigma, \circ, =, \emptyset \rangle$. Informally, we refer to the atomic assertions $P, Q \in \text{Prop}$ as outcomes.

As a notational note, for the remainder of the paper uppercase Greek metavariables $\Phi, \Psi$ denote semantic assertions. Greek metavariables $\phi, \psi$ refer to syntactic assertions in the outcome logic (Definition 4), uppercase metavariables $P, Q$ refer to atomic assertions (individual outcomes) and lowercase metavariables $p, q$ refer to assertions that are satisfied by individual program states.

Example 4.1 (Outcomes). We mentioned one example of a PCM in §2: $X$ can be sets of program states and the monoid operation $\circ$ is set union. Another example is probability (sub)distributions over a set and $\circ$ is $\cdot$. 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 §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.

\footnote{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.}
**Definition 5 (Under-Approximate Outcome Logic).** Given an outcome logic with satisfaction relation \( \models \subseteq MS \times \text{Prop} \), we define an under-approximate satisfaction relation \( \models^\downarrow \subseteq MS \times \text{Prop} \) as 
\[
m \models^\downarrow \varphi \iff m \models \varphi \oplus \top.
\]

Intuitively, \( \varphi \oplus \top \) corresponds to under-approximation since it is equivalent to saying that \( \varphi \) 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 \( m \models P \oplus Q \) implies that \( m \models P \oplus \top \). Equivalently, \( m \models^\downarrow P \oplus Q \) implies that \( m \models^\uparrow P \). 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 \( m_1 \preceq m_2 \iff \exists m. m_1 \circ m = m_2 \)). We prove this correspondence in Appendix B.1.

### 4.2 Monoidal Hoare Triples

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

**Definition 6 (Monoidal Hoare Triples).** The parameters needed to instantiate MHL are:

1. An execution model: \( \langle M, \text{bind}, \text{unit}, \circ, \Box \rangle \)
2. A set of program states \( \Sigma \) and semantics of atomic commands: \[ [\cdot]_\text{atom} : \Sigma \rightarrow MS \]
3. A syntax of atomic assertions \( \Sigma \) and satisfaction relation: \[ \models_\text{atom} \subseteq MS \times \text{Prop} \]

Now, let \[ [-] : \Sigma \rightarrow MS \] be the semantics of the language in Figure 2 with parameters (1) and (2) and \( \models \) be the Outcome Logic satisfaction relation (Definition 4) with parameters (1) and (3). For any program \( C \) (Figure 2), and Outcome Logic assertions \( \varphi \) and \( \psi \):

\[
\models (\varphi) C \langle \psi \rangle \iff \forall m \in MS. \ m \models \varphi \implies [C]^\uparrow (m) \models \psi
\]

MHL 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 post-conditions are expressed in Outcome Logic and thus satisfied by a monoidal collection \( m \in MS \), which can account for execution models such as nondeterminism and probability distributions.

Using Outcome Logic for pre- and post-assertions adds significant expressive power. We already saw in §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 postcondition holds if the program terminates—by taking a disjunction with \( \top \) 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.

\[
\models^\downarrow (\varphi) C \langle \psi \rangle \iff \models (\varphi) C \langle \psi \oplus \top \rangle \quad \models_{\text{pc}} (\varphi) C \langle \psi \rangle \iff \models (\varphi) C \langle \psi \lor \top \rangle
\]

In fact, the right triple corresponds exactly to standard Hoare Logic (Figure 1) if we instantiate MHL using the powerset semantics (Definition 7) and limit the pre- and post-conditions to be atomic assertions. This result is stated below and proven in Appendix C.

**Theorem 4.2 (Subsumption of Hoare Triples).** \( \models \{ P \} C \{ Q \} \iff \models_{\text{pc}} \{ P \} C \{ Q \} \)

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

\(^4\)Disjunctions are defined \( \varphi \lor \psi \) iff \( \neg (\neg \varphi \land \neg \psi) \), a standard encoding in classical logic.
All the rules in Figure 4 are sound (see Appendix F for details of the proof). The entailment to map entire outcomes to the true or false branches of an if statement, respectively. If (Multi-Outcome) entailment to annihilate the outcome where the guard is false. Similarly, substitutions based languages that have the syntax introduced in Example 3.1. We write \( \vdash \langle \varphi \rangle C \langle \psi \rangle \) refer to arbitrary assertions in the outcome logic and \( P, Q \) refer to atomic (single-outcome) assertions.

### 4.3 Proof Systems

Now that we have formalized the validity of Monoidal Hoare 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 MHL instance.

**Global rules.** Some generic rules that are valid for any MHL instance are shown at the top of Figure 4. Most of the rules including Zero, One, Seq, and Consequence are standard. 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 non-deterministic 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 3.1. We write \( P \vdash e \) to mean that \( P \) entails \( e \). Formally, if \( P \not\vdash e \) and \( Q \vdash \lnot e \) and \( m \vdash P \lor Q \), then \( \lbrack \text{assume } e \rbrack(m) \not\vdash P \). Syntactic substitutions \( P[e/x] \) must be defined and satisfy \( m \vdash P[e/x] \) implies \( \lbrack x := e \rbrack(m) \not\vdash P \).

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) use 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 4.3 (Soundness of Proof System).** If \( \vdash \langle P \rangle C \langle Q \rangle \) then \( \not\vdash \langle P \rangle C \langle Q \rangle \)
Noam Zilberstein, Derek Dreyer, and Alexandra Silva

Note that it is not possible to have generic loop-invariant based iteration rules that are valid for all instances of Monoidal Hoare Logic. This is because loop invariants assume a partial correctness specification; they do not guarantee termination. Monoidal Hoare 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 MHL AS A LOGIC FOR CORRECTNESS AND INCORRECTNESS

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 MHL can also disprove correctness specifications. Not only can MHL disprove the same specifications as IL (Corollary 5.5), but it can also express strictly more types of incorrectness. Theorem 5.4 shows three classes of bugs that can be characterized in MHL for nondeterministic programs, only one of which is expressible in IL. §5.2 shows that MHL can handle 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 $\Phi, \Psi \in 2^\mathbb{M}$ rather than the syntactic assertions $\varphi, \psi \in \text{Prop}$ we have seen thus far. The advantage of this approach is that we can show the power of the MHL model without worrying about the expressiveness of the assertion language. The 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 Falsification theorem states that any false MHL triple can be disproven within MHL. Since we already know that MHL subsumes Hoare Logic (Theorem 4.2), this result tells us that any correctness property that is expressible in Hoare Logic can be disproven using MHL.

**Theorem 5.1 (Semantic Falsification).** For any MHL instance and any program $C$ and semantic assertions $\Phi, \Psi$:

$$^\ast_S \langle \Phi \rangle C \langle \Psi \rangle \iff \exists \Phi' . \text{such that } \Phi' \Rightarrow \Phi , \text{sat}(\Phi') , \text{and } ^\ast_S \langle \Phi' \rangle C \langle \neg \Psi \rangle$$

The proof of this theorem, as well as the full formulation of semantic triples, is given in Appendix D.1. This result shows the power of the MHL model, but we also seek to answer whether the Outcome Logic syntax 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 non-deterministic and probabilistic models. While the semantic proof above applies to any MHL instance, the syntactic versions that we present in §5.1 and §5.2 rely on additional properties of the specific MHL instance. Having a syntactic description of the precondition gives us a characterization 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 (MHL). We can prove a syntactic version of The Principle of Denial for MHL, which can be thought of as a generalization of the true positives property, since it tells us when an MHL triple (denoting a bug) disproves another MHL triple (denoting correctness).

**Theorem 5.2 (Principle of Denial).** For any MHL instance and any program $C$ and syntactic assertions $\varphi, \varphi', \text{and } \psi$:

$$\text{If } \varphi' \Rightarrow \varphi , \text{sat}(\varphi') , \text{and } \vDash \langle \varphi' \rangle C \langle \neg \psi \rangle \text{ then } \not\vDash \langle \varphi \rangle C \langle \psi \rangle$$

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 \( \Phi \). The way in which this assertion is constructed depends on several properties of the MHL 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 MHL 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 \( \psi \) is a sequence of outcomes \( Q_1 \oplus \cdots \oplus Q_n \), then it is not immediately clear what \( \neg \psi \) expresses.

### 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 MHL by defining an evaluation model and BI frame.

**Definition 7 (Nondeterministic Evaluation Model).** A nondeterministic evaluation model based on program states \( \sigma \in \Sigma \) is \( \langle 2^\Sigma, \text{bind}, \text{unit}, \cup, \emptyset \rangle \) where \( \langle 2^{\langle - \rangle}, \text{bind}, \text{unit} \rangle \) is the powerset monad:

\[
\text{bind}(S, k) \triangleq \bigcup_{x \in S} k(x) \quad \text{unit}(x) \triangleq \{x\}
\]

**Definition 8 (Nondeterministic Outcome Logic).** Given some satisfaction relation on program states \( \models \subseteq \Sigma \times \text{Prop} \), we create an instance of the outcome logic (Definition 4) with the BI frame \( \langle 2^\Sigma, \cup, =, \emptyset \rangle \) such that atomic assertions come from \( \text{Prop} \) and are satisfied as follows:

\[
S \models P \iff S \neq \emptyset \quad \text{and} \quad \forall \sigma \in S. \sigma \models \Sigma P
\]

We impose one additional requirement, that the atomic assertions \( Q \in \text{Prop} \) can be logically negated, 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 5.3 shows that there are exactly three ways that this assertion can be false.

**Lemma 5.3 (Falsifying Assertions).** For any \( S \in 2^\Sigma \) and atomic assertions \( Q_1, \ldots, Q_n \),

\[
S \not\models Q_1 \oplus \cdots \oplus Q_n \quad \text{iff} \quad \exists i. \ S \not\models \overline{Q}_i \quad \text{or} \quad S \not\models (\overline{Q}_1 \wedge \cdots \wedge \overline{Q}_n) \oplus \top \quad \text{or} \quad S \not\models \top^\oplus
\]

If we take \( Q_1 \oplus \cdots \oplus Q_n \) to represent a desirable set of program outcomes, then Lemma 5.3 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 5.4 (Nondeterministic Falsification).** For any MHL instance based on the nondeterministic evaluation model (Definition 7) and outcome logic (Definition 8), \( \langle \varphi \rangle \ C \langle \psi \rangle \) if:

\[
\exists \varphi' \Rightarrow \varphi. \ \text{sat}(\varphi') \quad \text{and} \quad \exists i. \ \not\models \langle \varphi' \rangle \ C \langle \overline{Q}_i \rangle \quad \text{or} \quad \not\models \langle \varphi' \rangle \ C \langle \bigwedge_{i=1}^n \overline{Q}_i \rangle \quad \text{or} \quad \not\models \langle \varphi' \rangle \ C \langle \top^\oplus \rangle
\]

\(^5\)Crucially, \( \overline{Q} \) is not the same as \( \neg Q \) (where \( \neg \) is from BI) since \( S \models \neg Q \iff \exists \sigma \in S. \sigma \not\models Q \) whereas \( S \models Q \iff \forall \sigma \in S. \sigma \models Q \).
The type of bugs expressible in Incorrectness Logic are a special case of Theorem 5.4. 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 MHL (Theorem 4.2), Theorem 5.4 suggests that MHL can disprove Hoare Triples as well. We make this correspondence explicit in the following Corollary where, compared to Theorem 5.4, 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.

**Corollary 5.5 (Hoare Logic Falsification).**

\[ \forall \{P\} C \{Q\} \quad \text{iff} \quad \exists \varphi \Rightarrow P. \; \text{sat} (\varphi) \quad \text{and} \quad \downarrow \langle \varphi \rangle C \langle \overline{Q} \rangle \]

So, although MHL does not *semantically* subsume Incorrectness Logic, it does subsume IL’s bug-finding abilities. MHL can also disprove more complex correctness properties, such as that of bug-finding abilities. MHL can also disprove more complex correctness properties, such as that of the shuffle function that we saw in §2.2. As we will now see, another MHL 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 \( \mu \in D X \) 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 \( \text{supp}(\mu) = \{ x \mid \mu(x) > 0 \} \) and the mass of a distribution is defined as \( |\mu| = \sum_{x \in \text{supp}(\mu)} \mu(x) \). A valid distribution must have mass at most 1. The empty distribution \( \emptyset \) maps everything to probability 0 and distributions can be summed \( \mu_1 + \mu_2 = \lambda x. \mu_1(x) + \mu_2(x) \) if \( |\mu_1| + |\mu_2| \leq 1 \). For any countable set \( X \), \( \langle DX, +, \emptyset \rangle \) 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 \( \delta_x \) assigns probability 1 to \( x \) and 0 to everything else. We complete the definition of a probabilistic execution model:

**Definition 9 (Probabilistic Evaluation Model).** A probabilistic evaluation model based on program states \( \Sigma \) is defined as \( \langle D \Sigma, \text{bind}, \text{unit}, +, \emptyset \rangle \) where \( \langle D, \text{bind}, \text{unit} \rangle \) is the Giry [1982] monad:

\[
\text{bind}(\mu, k) = \sum_{x \in \text{supp}(\mu)} \mu(x) \cdot k(x) \quad \text{unit}(x) = \delta_x
\]

We can make our imperative language probabilistic by adding a command for sampling from finitely supported probability distributions \( \eta \in D \text{Val} \) over program values \( x \triangleleft \eta \). This command is intended to be added to an existing language such as GCL (Example 3.1) or HeapLang (§6.4). The program semantics and atomic assertions are based on distributions over program states \( \mu \in D \Sigma \). The semantics for the sampling command is defined in terms of variable assignment. This allows us to abstract over the type of program states (i.e. \( S \) or \( S \times H \)).

\[
c := x \triangleleft \eta \quad [x \triangleleft \eta] (\sigma) = \text{bind}(\eta, \lambda v. [x := v] (\sigma))
\]

**Definition 10 (Probabilistic Outcome Logic).** Given some satisfaction relation on program states \( \forall \Sigma \subseteq \Sigma \times \text{Prop} \), we instantiate Outcome Logic (Definition 4) with the BI frame \( \langle D \Sigma, +, =, \emptyset \rangle \) such that atomic assertions have the form \( \mathbb{P}[A] = p \) where \( p \in [0,1] \), \( A \in \text{Prop} \), and:

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

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 $A_i$. The special case where $n = 2$ is shown below.

$$\mu \not\models \mathbb{P}[A] = p \oplus \mathbb{P}[B] = q \iff \mu \not\models \mathbb{P}[A \land B] = p_1 \oplus \mathbb{P}[\neg A \land B] = p_2 \oplus \mathbb{P}[\neg A \land \neg B] = p_3 \oplus \mathbb{P}[\neg A \land \neg B] = p_4$$

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 falsify 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 $A_i$s are pairwise disjoint, then falsification can be achieved with just $n + 1$ outcomes.

**Theorem 5.6 (Disjoint Falsification).** First, let $A_0 = \bigwedge_{i=1}^n \neg A_i$. If all the events are disjoint (for all $i \neq j$, $A_i \land A_j$ iff false), then:

$$\mu \not\models \langle \varphi \rangle C \langle \bigoplus_{i=1}^n (\mathbb{P}[A_i] = p_i) \rangle \iff \exists q_i \in [0,1] \text{ s.t. } q_i \neq p_i.$$  

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

Most 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 5.7 (Principle of Denial for Lower Bounds).**

If $\exists \varphi \Rightarrow \varphi$. sat($\varphi'$), $\models \langle \varphi' \rangle C \langle \mathbb{P}[\neg A] \geq q \rangle$ then $\models \langle \varphi \rangle C \langle \mathbb{P}[A] \geq p \rangle$ (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 $\mu$ where $|\mu| < 1$ and therefore $(\mathbb{P}_\mu[A] \not\geq p) \not\models (\mathbb{P}_\mu[\neg A] > 1 - p)$. There are many more special cases for probabilistic falsification, but the ones that we care about for the purposes of this paper fall into the categories discussed above.

### 6 MHL FOR MEMORY ERRORS

In this section we specialize MHL to prove the existence of memory errors in non-deterministic 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 (§6.1). On top of that, we build an assertion logic with the capability of describing error states and multiple outcomes (§6.2). Then, we define the execution model using a monad combining both errors and non-determinism (§6.3). Finally, we provide proof rules for this multi-layered logic (§6.4).

We use this logic in §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 Monoidal Hoare Logic is a good fit for this type of bug finding by examining manifest errors in more depth (§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 $x \not\rightarrow$, state that the pointer $x$ is invalidated [Raad et al. 2020]. The syntax for the heap assertion logic is below.

$$p \in \text{SL} ::= \text{emp} \mid \exists x. p \mid p \land q \mid p \lor q \mid p \Rightarrow q \mid p \not\rightarrow q \mid e \mid e_1 \leftrightarrow e_2 \mid e \not\rightarrow \not\rightarrow \mid e \not\rightarrow (7)$$
In this syntax $e \in \Exp$ 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 3.1) and
heaps are partial functions from positive natural numbers (addresses) to program values or bottom
$\mathcal{H} = \{h \mid h : \mathbb{N}^+ \rightarrow \text{Val} + \{\bot\}\}$. The constant null is equal to 0, so it is not a valid
address and therefore for any heap $h$, $n \notin \text{dom}(h)$.

The semantics of $\mathcal{SL}$ 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 $(\text{ok} : p)$ and $(\text{er} : q)$ 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 the relations $\equiv_E \subseteq E \times \text{Prop}_E$ and $\equiv_\Sigma \subseteq \Sigma \times \text{Prop}_\Sigma$, we
construct a new assertion logic with semantics $\equiv \subseteq (E + \Sigma) \times (\text{Prop}_\Sigma \times \text{Prop}_E)$ defined below:

$$i_L(e) \equiv (p, q) \iff e \in_E q \quad i_R(\sigma) \equiv (p, q) \iff \sigma \in_\Sigma p$$

In the above, let $i_L : E \rightarrow E + A$ and $i_R : A \rightarrow E + A$ be the left and right injections, respectively.
We also add syntactic sugar $(\text{ok} : p) \triangleq (p, \text{false})$ and $(\text{er} : q) \triangleq (\text{false}, q)$, so in general the assertion
$(p, q)$ can be thought of as $(\text{ok} : p) \lor (\text{er} : q)$. Additional logical operations ($\neg, \land,$ and $\lor$) 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 $\text{Prop}$ is given in Definition 6.1 with $\text{Prop}_E = \text{Prop}_\Sigma = \mathcal{SL}$, the
  heap assertion logic (7). So, $\text{Prop}$ has the syntax $(\text{ok} : p)$ and $(\text{er} : q)$ where $p, q \in \Sigma$.
- $\Sigma$, 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 $\equiv \subseteq (\Sigma + \Sigma) \times \text{Prop}$.

6.3 Execution Model

The execution model supports both non-determinism and errors. We achieve this by combining the
powerset monad (Definition 7) 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 $(M, \text{bind}_M, \text{unit}_M, \circ, \emptyset)$, we define a new execution model $(M(E + \cdot), \text{bind}_e, \text{unit}_e, \circ, \emptyset)$ such that:

$$\text{bind}_e(m, k) = \text{bind}_M(m, \lambda x. \begin{cases} k(y) & \text{if } x = i_R(y) \\ \text{unit}_M(x) & \text{if } x = i_L(y) \end{cases}) \quad \text{unit}_e(x) = \text{unit}_M(i_R(x))$$

Note the monoid definition ($\circ$ and $\emptyset$) 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.

\footnote{Note that $\ell \notin \text{dom}(h)$ indicates that we have no information about the pointer $\ell$ whereas $h(\ell) = \bot$ indicates that $\ell$ is
deallocated. This is why $h$ is both partial and includes $\bot$ in the co-domain.}
We first need to define the notion of a pure command, which does not display any computational effects. We now turn to defining a language of atomic commands for manipulating the heap. The syntax for this language is given below and the semantics is in Appendix E.3. We are particularly interested in the powerset monad. This results in an execution model 

\[
\langle \text{ok} : p \rangle \text{error()} \quad \langle \text{er} : p \rangle
\]

\[
\langle \text{ok} : e \mapsto e \rangle \text{free}(e) \quad \langle \text{ok} : e \mapsto e \rangle
\]

\[
\langle \text{ok} : e \mapsto e \rangle \left[ e_1 \leftarrow e_2 \langle \text{ok} : e \mapsto e \rangle \right]
\]

\[
\langle \text{ok} : x = x' \land e \mapsto x'' \rangle \quad \langle \text{ok} : x = x' \land e[x/x'] \mapsto x'' \rangle
\]

\[
\text{pure}_{\text{C}}(p) \quad \text{pure}_{\text{set}}(C) \quad \text{pure}_{\text{A}}(C)
\]

\[
\text{alloc} \quad \text{dealloc} \quad \text{load} \quad \text{store}
\]

\[
\text{Frame} \quad \text{Lifting} \quad \text{Disjunction} \quad \text{Error Propagation}
\]

Monoidal Hoare Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning

Fig. 5. Proof rules for Under-approximate Monadic Separation Logic, the MHL instantiation of Definition 6.5. The first group is inspired by Reynolds [2002]'s original proof system with additional rules added for unsafe states. The second group deal with the monadic execution model.

Example 6.4 (Execution model for non-determinism and errors). We are particularly interested in the powerset monad. This results in an execution model 

\[
\langle 2^{E^+}, \text{bind}, \text{unit}, \oslash, \varnothing \rangle
\]

where the operations are derived as follows:

\[
\text{bind}(S, k) = \{i_L(x) \mid i_L(x) \in S \} \cup \bigcup_{i_R(x) \in S} k(x)
\]

\[
\text{unit}(x) = \{i_R(x)\}
\]

We now turn to defining a language of atomic commands for manipulating the heap. The syntax for this language is given below and the semantics is in Appendix E.3.

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

Assume and assignment are the same as in GCL (Example 3.1). 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 := malloc() as syntactic sugar for (x := alloc()) + (x := null), which is valid in nondeterministic evaluation models.

Definition 6.5 (Outcome-Based Separation Logic). We instantiate MHL (Definition 6) with:

1. The execution model is from Example 6.4 with \( E = S \times H \).
2. The language of atomic commands is HeapLang.
3. The assertion logic is the one given in Definition 8 using Definition 6.2 for basic assertions.

Note that although the execution model has been augmented with errors, the nondeterministic falsification result (Theorem 5.4) still holds for Outcome-Based Separation Logic.

6.4 Proof Rules for Memory Errors

We first need to define the notion of a pure command, which does not display any computational effects. Pure commands cannot use nondeterministic choice or errors.
We now demonstrate that the MHL proof system shown in Figure 5 is effective for bug-finding. 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 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 one in the library. A call to push_back can reallocate the vector’s underlying memory buffer, in which case pointers to that buffer become invalid.

We give proof rules in Figure 5 for Outcome-Based Separation Logic (Definition 6.5). We will use these proof rules in subsequent sections to prove memory errors. The first group of rules is very close to the standard separation logic proof system originally due to Reynolds [2002], with the addition of rules to reason about unsafe states inspired by Raad et al. [2020]. These rules are pure (see Definition 11)—they do not assume a particular evaluation model and can therefore be lifted for use in either a nondeterministic or probabilistic proof system. The lifting proof rule states that if some triple is derivable for a pure program (where p and q are satisfied by individual states), then we can lift it such that p and q are satisfied by sets of states.

The proof rules from Figure 4 for conditionals can also be used in this proof system if expression entailment is defined. This will only be defined for ok assertions where (ok : p) ⊨ e iff p ⇒ e. Similarly, substitution is also only defined for ok assertions: (ok : p)[e/x] = ok : p[e/x]. This means that the Assign rule only allows us to prove ⊢ (ok : p[e/x]) x := e (ok : p). If an error has occurred, we instead use the Error Propagation rule to propagate the error forward through the proof, since the program will never recover from the crash. The rule shown in Figure 5 restricts C to be pure so that soundness can be established regardless of the execution model. It is straightforward to derive a similar rule for any C using lifting rules.

### 6.5 Proof of a Bug

We now demonstrate that the MHL 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 is given by postcondition (ok : v ↦ x * x ↦ 1). As Theorem 5.4 showed, we can disprove it by showing that an undesirable outcome will sometimes occur. In this case, that undesirable outcome is (er : x ↦ ). Clearly, (er : x ↦ ) ⇒ ¬(ok : v ↦ x * x ↦ 1), so the specification in Figure 6 succeeds in falsifying 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.

\[ \equiv \langle \text{ok} : x \not\rightarrow \rangle [x] \leftarrow 1 \langle \text{er} : x \not\rightarrow \rangle \quad \equiv \langle \text{ok} : \text{true} \rangle x := \text{malloc}() ; [x] \leftarrow 1 \langle \text{er} : x = \text{null} \rangle \]

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
we need to know what happens when we run the program on any programming has a rich history [Kozen 1979, 1983], but there is little prior work on proving that development as it is essential for machine learning and security applications. The study of probabilistic Randomization is a powerful tool that is seeing increased adoption in mainstream software development, but there is little prior work on proving that.

Reachability is important, but we only have to reach just the right ways in both paths must be considered as seen in the. If precondition to be precise enough to force the execution down a specific logical path, otherwise more specific than the precondition of the conclusion (\( p \)). One-Sided If generates imprecise preconditions since the precondition of the premise (\( p \)) is more specific than the precondition of the conclusion (\( \neg e \)). MHL, 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, MHL 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.
Consider the simple learning problem in which we want to learn a point \( x \in \mathbb{R} \) from some probability distribution \( \eta \). 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 \( h : X \rightarrow \mathbb{B} \) which guesses whether new data points are positive or negative examples.

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 \( h : X \rightarrow \mathbb{B} \) which guesses whether new data points are positive or negative examples.

Consider the simple learning problem in which we want to learn a point \( t \in [0, 1] \subset \mathbb{R} \). Since we require distributions used in programs to be finite, we can approximate \( [0, 1] \) as \( \{ k \cdot \Delta \mid 0 \leq k \leq \frac{1}{\Delta} \} \) for some finite step size \( \Delta \). 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 \( L(x) = x \leq t \) 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 \( \varepsilon \) after \( N \) iterations. Practically speaking, this simulates training the model on a dataset of size \( N \). Intuitively, the error will be less than \( \varepsilon \) 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
we need a larger dataset in order to get a better result. This definition differs from standard Incorrectness Logic in two ways. First, assertions are satisfied using the sub-distribution relation instead of set inclusion. Second, under-approximation is achieved by distributions over program states $\mu \in \mathcal{D} \mathcal{X}$ 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 $\mathbb{P}[\text{er}(h) > \varepsilon] \geq (1 - \varepsilon)^N$. This is not a valid postcondition for an incorrectness triple because it does not adequately describe the resulting distribution of the program. That

\[ \mathbb{P}[\text{true}] = 1 \implies \mathbb{P}[\text{er}(-1) > \varepsilon] \geq (1 - \varepsilon)^N \]

\[ h := -1 \]

\[ \mathbb{P}[\text{er}(h) > \varepsilon] \geq (1 - \varepsilon)^N \]

for $N$ do

\[ \mathbb{P}[\text{er}(h) > \varepsilon] \geq (1 - \varepsilon)^N \]

$x \leftarrow [0, 1]$

if $L(x) \land x > h$ then

$h := x$

else

skip

\[ \mathbb{P}[\text{er}(h) > \varepsilon] \geq (1 - \varepsilon)^N \]

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

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

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 $\varepsilon$ and a confidence parameter $\delta$, 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 5.7 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 $\delta$. 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.

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

iff

$$\forall \mu \models Q.\ \exists \mu'.\ \mu \sqsubseteq \mathbb{C}^1(\mu')$$

and

$$\mu' \models P$$

This definition differs from standard Incorrectness Logic in two ways. First, assertions are satisfied by distributions over program states $\mu \in \mathcal{D} \mathcal{X}$ 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. 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 $\mathbb{P}[\text{er}(h) > \varepsilon] \geq (1 - \varepsilon)^N$. This is not a valid postcondition for an incorrectness triple because it does not adequately describe the resulting distribution of the program.
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
\[
P[\text{er}(h) > \varepsilon] = (1 - \varepsilon)^N
\]
does not solve the problem. This assertion can be satisfied by a distribution where \( h = -1 \) with probability \((1 - \varepsilon)^N\) and \( h = t \) with probability \(1 - (1 - \varepsilon)^N\), 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:

\[
\bigoplus_{x=0}^{\ell} \left( P[\text{er}(h) = x] = (1 - x)^N - (1 - (x + \Delta))^N \right)
\]

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 of having a large error. 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, no technique that summarizes a distribution can possibly be used with incorrectness logic since it does 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.** This paper is inspired by past work on Incorrectness Logic [O’Hearn 2019; Raad et al. 2020, 2022]. Our approach is based on forwards-running (Hoare-style) program logics rather than the backwards-style that has been typical for incorrectness reasoning thus far.

Möller et al. [2021, §5] mentioned the possibility of using an under-approximate variant of Hoare Triples in lieu of the original IL triples and referred to them as backwards under-approximate triples with semantics based on a calculus of possible correctness due to Hoare [1978, §5.3]. These triples were in fact proposed early on in the development of Incorrectness Logic, but were jettisoned in favor of the IL triple semantics given by O’Hearn [2019]. Though they were subsequently presented briefly in Möller et al. [2021, §5] and mentioned again in Le et al. [2022, §3.2], the potential of these triples has gone largely unexplored. In Appendix C, we show that they are a special case of MHL.

In addition to our own observations that IL is not a good fit for characterizing manifest errors and reasoning about probabilistic incorrectness, Ascari et al. [2022] noted that designing abstract domains compatible with Incorrectness Logic is infeasible. This further motivates our work, as it is well known that abstract domains are compatible with Hoare Logic-style reasoning.

Other approaches to incorrectness reasoning have focused on abstract interpretation [Bruni et al. 2021; Ball et al. 2005] and symbolic execution [Blackshear et al. 2013]; refer to O’Hearn [2019, §7] for a more complete description of this work.
**Probabilistic Program Analysis.** Probabilistic instantiations of MHL have close similarities to existing probabilistic Hoare Logics [Barthe et al. 2018; den Hartog 2002; Rand and Zdancewic 2015]. Some aspects of outcomes could be seen as a generalization of the aforementioned probabilistic assertion logics; Barthe et al. [2018] have a similar Split rule to our own. Our work is compositional (via the Lifting rule) which allows us to easily embed separation logic inside a probabilistic proof system. This is somewhat similar to Polaris which combines probabilistic reasoning with concurrent separation logic and also uses a monadic probabilistic semantics [Tassarotti and Harper 2019].

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-pre. 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 as 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 a number of 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 Incorrectness Logic. 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. Monoidal Hoare Logic achieves just that, with the added benefit of unifying the theories of correctness and incorrectness in a single program logic. To show that MHL can capture incorrectness, we prove a falsification theorem that states that every triple can be disproven within the logic—this means that if there is a bug invalidating a correctness specification, it can be found. MHL also offers a cleaner characterization of manifest errors, suggesting it may be semantically closer to the way that programmers naturally reason about bugs.

In this paper, we introduced MHL 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 program 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 MHL) cannot drop disjuncts; however, as we saw in §2 and §4, MHL can drop outcomes, which we believe is sufficient to make the algorithm scale to large codebases (although this remains to be demonstrated). Furthermore, since MHL 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 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.

REFERENCES


A TOTALITY OF LANGUAGE SEMANTICS

As mentioned in §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 (⋄). 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*.

Regardless of the execution model, proving the fixed point existence requires us to prove that the semantic map [−]† is continuous with respect to some partial order. We remark that a preorder can be generically defined in terms of the monoid operation m1 ⊑ m2 iff there exists m such that m1 ○ m = m2. 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, ⊑ is equivalent to ⊆.

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., ∀σ. [e](σ) ∈ B.

A.1 Nondeterministic Languages

Since the monoid operation for nondeterministic languages is set union (a total function), we can allow unrestricted access to C1 + C2 and C★. 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)(σ) = f†([C](σ)) ∪ unit(σ) has a least fixed point when specialized to the powerset monad.

Proof. We first note that in the lemma statement f : Σ → 2Σ and [C] : Σ → 2Σ. We also define the point-wise partial order f1 ⊑ f2 iff ∀x. f1(x) ⊆ f2(x). Clearly, the function λx.Ø is the bottom of this order. This also means that for any non-empty chain f1 ⊑ f2 ⊑ ... it must be that ∪i f1 = λx. ∪i f1(x).

We now show that F is Scott continuous:

\[
F\left(\bigcup_i f_i\right) = \lambda \sigma. \left(\bigcup_i f_i \right)^\dagger ([C](\sigma)) \cup \{\sigma\}
\]
\[
= \lambda \sigma. \left(\bigcup_{r \in [C](\sigma)} \left(\bigcup_i f_i(r) \right) \right) \cup \{\sigma\}
\]
\[
= \lambda \sigma. \bigcup_{r \in [C](\sigma)} (\bigcup_i f_i(r) \cup \{\sigma\})
\]
\[
= \lambda \sigma. \bigcup_i f_i([C](\sigma)) \cup \{\sigma\}
\]
\[
= \lambda \sigma. \bigcup_i F(f_i)(\sigma)
\]
\[
= \bigcup_i F(f_i)
\]

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

A.2 Probabilistic Languages

In probabilistic languages, we can ensure totality using simple syntactic checks. That is, we syntactically limit programs to not use C1 + C2 and C*, but rather the guarded versions as shown in Example 3.1. 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:

$$|\text{bind}_D(\mu, f)| = \left| \sum_{\sigma \in \text{supp}(\mu)} \mu(\sigma) \cdot f(\sigma) \right| = \sum_{\sigma \in \text{supp}(\sigma)} \mu(\sigma) \cdot |f(\sigma)| \leq \sum_{\sigma \in \text{supp}(\sigma)} \mu(\sigma) = |\mu|$$

Since $f : \Sigma \to \mathcal{D}\Sigma$, then for any $\sigma \in \Sigma$, $|f(\sigma)| \leq 1$. Therefore, we have shown that $|\text{bind}(\mu, f)| \leq |\mu|$ (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 $[[\text{if } e \text{ then } C_1 \text{ else } C_2]](\sigma) = [\text{assume } e \uplus C_1] + [\text{assume } \neg e \uplus C_2](\sigma) = [\text{assume } e \uplus C_1](\sigma) + [\text{assume } \neg e \uplus C_2](\sigma)$. Now, we do case analysis on the value of $[[e]]_{\text{Exp}}(\sigma)$. If $[[e]]_{\text{Exp}}(\sigma) = \text{true}$, then $[[\text{assume } e]](\sigma) = \delta_\sigma$ and $[\text{assume } \neg e](\sigma) = \emptyset$. Therefore, we know that $[[\text{assume } e \uplus C_1]](\sigma) = [C_1](\sigma)$ and $[[\text{assume } \neg e \uplus C_2]](\sigma) = \emptyset$. By the induction hypothesis $[C_1](\sigma)$ is defined and so $[C_1](\sigma) + \emptyset$ must also be defined. The case where $[[e]]_{\text{Exp}}(\sigma) = \text{false}$ is symmetrical.

- **While.** We begin by proposing an alternate semantics for (guarded) while loops: $[[\text{while } e \text{ do } C]](\sigma) = \text{lfp}(F)(\sigma)$ where $F(f)(\sigma) = \delta^\uplus([\text{assume } e \uplus C]](\sigma)) \circ [\text{assume } \neg e](\sigma)$.

In this semantics, we push the assume $\neg e$ in to the fixed point computation which allows $\circ$ to be defined. In the nondeterminism case where $\circ$ 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:

$$(\mu_1 \sqcup \mu_2) + \mu = (\lambda \sigma. \max(\mu_1(\sigma), \mu_2(\sigma))) + \mu = \lambda \sigma. \max((\mu_1 + \mu)(\sigma), (\mu_2 + \mu)(\sigma)) = (\mu_1 + \mu) \sqcup (\mu_2 + \mu)$$

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 $\forall x. f_1(x) \subseteq f_2(x)$.

$$F\left(\bigsqcup_i f_i\right) = \lambda \sigma. \left(\bigsqcup_i f_i\right)^\uplus([\text{assume } e \uplus C](\sigma)) + [\text{assume } \neg e](\sigma)$$

$$= \lambda \sigma. \sum_{\tau \in [\text{assume } e \uplus C](\sigma)} (\bigsqcup_i f_i(\tau)) + [\text{assume } \neg e](\sigma)$$

$$= \lambda \sigma. \bigsqcup_i (f_i^\uplus([\text{assume } e \uplus C](\sigma)) + [\text{assume } \neg e](\sigma))$$

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

$$= \lambda \sigma. \bigsqcup_i F(f_i)(\sigma)$$

$$= \bigsqcup_i F(f_i)$$

Therefore, by the Kleene Fixed Point Theorem, $\text{lfp}(F) = \bigsqcup_{n \in \mathbb{N}} F^n(\lambda x. \emptyset)$.
B UNDER-APPROXIMATION

In Definition 5, we defined under-approximate outcome assertions $m \vdash \varphi$ to be syntactic sugar for $m \not\vDash \varphi \oplus \top$. 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 $m \not\vDash \varphi \oplus \psi$. Then there exists $m_1$ and $m_2$ such that $m \not\vDash m_1 \odot m_2$ and $m_1 \not\vDash \varphi$ and $m_2 \not\vDash \psi$. Clearly, also $m_2 \not\vDash \top$, so $m \not\vDash \varphi \oplus \top$. The second implication is symmetric. ■

**Lemma B.2 (Dropping Outcomes (Under-Approximate)).** In any BI frame, if $m \vdash \varphi \oplus \psi$, then $m \not\vDash \varphi$.

**Proof.** Since $m \vdash \varphi \oplus \psi$, then $m \not\vDash \varphi \oplus \psi \oplus \top$. This means that $m_1 \not\vDash \varphi$, $m_2 \not\vDash \psi$ and $m_3 \not\vDash \top$ such that $m_1 \odot m_2 \odot m_3 \not\vDash m$. Clearly, $m_2 \not\vDash \top$ as well. Recombining these, we get $m \not\vDash \varphi \oplus \top \oplus \top$ which is equivalent to $m \not\vDash \varphi \oplus \top$, or just $m \not\vDash \varphi$. ■

B.1 Alternative Formulation using Intuitionistic BI

In §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:

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

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 intuitionistic BI interpretations must respect the persistence property: if $m \vDash P$ and $m' \vDash m$, then $m' \vDash P$ (this is also referred to as monotonicity in Kripke semantics). We will now show that the under-approximate satisfaction relation $\vdash$ is valid as an intuitionistic satisfaction relation for atomic propositions.

**Lemma B.3 (Under-Approximate Satisfaction is Persistent).** For any $m, m' \in ME_S$ and atomic assertion $P$, if $m \vdash P$ and $m' \geq m$, then $m' \vdash P$.

**Proof.** If $m = \varnothing$, then $m'$ is also $\varnothing$ and so clearly $m' \vdash P$. Now suppose that $m \neq \varnothing$. Since $m \vdash P$, then $m \vDash P \oplus \top$. Since $m' \geq m$ and $m \neq \varnothing$, there is some $m''$ such that $m \odot m'' = m'$. Clearly, $m'' \vDash \top$, so $m' \vDash (P \oplus \top) \oplus \top$. This means that $m' \vDash P \oplus \top$, or in other words $m' \vdash P$. ■

If we combine the under-approximate satisfaction relation with the basic assertions for nondeterministic and probabilistic evaluation models (Definitions 8 and 10), 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, $S \vdash P$ iff $\exists \sigma \in S. \sigma \vDash_S P$.

**Proof.**

($\Rightarrow$) Suppose that $S \vdash P$, 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 $\forall \sigma \in S_1. \sigma \vDash_S P$. Since we know $S_1$ is nonempty, then there exists $\sigma \in S_1. \sigma \vDash S$ and since $S_1 \subseteq S$, then $\sigma \in S$ as well, so $\exists \sigma \in S. \sigma \vDash_S P$.

($\Leftarrow$) Suppose that $\exists \sigma \in S. \sigma \vDash_S P$. 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 $S \vdash P$. ■
LEMMA B.5. In the distribution interpretation of BL, \( \mu \models (P[A] = p) \) iff \( \mathbb{P}_\mu [A] \geq p \), where:
\[
\mathbb{P}_\mu [A] = \sum \{ \mu(\sigma) \mid \sigma \in \text{supp}(\mu), \sigma \vdash_\Sigma A \}
\]

Proof.

(\( \Rightarrow \)) Assume that \( \mu \models (P[A] = p) \), so \( \mu \not\models (P[A] = p) \oplus \top \). Therefore \( \mu_1 \not\models (P[A] = p) \) and \( \mu_2 \not\models \top \) such that \( \mu_1 + \mu_2 = \mu \). This tells us that \( \mathbb{P}_{\mu_1} [A] = p \). When we add \( \mu_2 \) to \( \mu_1 \) to get \( \mu \), 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 \( \mu_1 \) of \( \mu \) such that \( \mu_1 = p \) and \( \forall \sigma \in \text{supp}(\mu_1), \sigma \vdash_\Sigma A \). Let the other part of the distribution be \( \mu_2 \) (so \( \mu = \mu_1 + \mu_2 \)). Now, by construction, \( \mu_1 \not\models (P[A] = p) \) and \( \mu_2 \not\models \top \), so \( \mu \not\models (P[A] = p) \oplus \top \), or equivalently \( \mu \not\models (P[A] = p) \).

\( \square \)

C EQUIVALENCE OF TRIPLES

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

\[
\llbracket P \rrbracket \subseteq \llbracket Q \rrbracket \iff \forall \sigma \in \Sigma, \sigma \vdash_\Sigma P \Rightarrow \exists \tau \in \llbracket C \rrbracket (\sigma). \tau \vdash_\Sigma Q
\]

Now, we will work with nondeterministic instances of MHL where the evaluation model is defined in Definition 7 and the logic of atomic assertions is defined from Definition 8. We let BL disjunctions \(\varphi \lor \psi\) be syntactic sugar for \(\neg(\neg \varphi \land \neg \psi)\) (this encoding is typical in classical logics). We now prove our first result, that Hoare Triples are subsumed by MHL. As we mentioned in §4.2, since Hoare Triples are partial correctness specification, we have to use the postcondition \(Q \lor \top\) to express that \(Q\) is reachable if the program terminates\(^8\).

THEOREM 4.2 (SUBSUMPTION OF HOARE TRIPLES). \( \models \{ P \} C \{ Q \} \) iff \( \models_{pc} \langle P \rangle C \langle Q \rangle \)

Proof.

(\( \Rightarrow \)) Suppose that \( \models \{ P \} C \{ Q \} \) or in other words, for any \( \sigma \vdash_\Sigma P \) and \( \tau \in \llbracket C \rrbracket (\sigma), \tau \vdash_\Sigma Q \). Now suppose that \( S \not\models P \), or in other words, \( S \not\in \emptyset \) and \( \forall \sigma \in S, \sigma \vdash_\Sigma P \). Since \( \models \{ P \} C \{ Q \} \), then for any \( \tau \in \llbracket C \rrbracket (\sigma), \tau \vdash_\Sigma Q \). Now, we know that \( \llbracket C \rrbracket (S) = \text{bind}(S, \llbracket C \rrbracket) = \bigcup_{\sigma \in S} \llbracket C \rrbracket (\sigma) \). This means that for every \( \tau \in \llbracket C \rrbracket (S), \tau \vdash_\Sigma Q \). So, if \( \llbracket C \rrbracket (S) \neq \emptyset \), then \( \llbracket C \rrbracket (S) \not\models \top \). Therefore \( \llbracket C \rrbracket (S) \not\models \top \) and \( \models \{ P \} C \langle Q \rangle \).

(\( \Leftarrow \)) Suppose that \( \models \{ P \} C \langle Q \lor \top \rangle \) and so if \( S \not\models P \), then \( \llbracket C \rrbracket (S) \not\models Q \lor \top \). Now suppose that \( \sigma \vdash_\Sigma P \). Then trivially \( \{ \sigma \} \not\models P \), so we can use our assumption to conclude that \( \llbracket C \rrbracket (\{ \sigma \}) \not\models Q \lor \top \). This implies that \( \forall \tau \in \llbracket C \rrbracket (\sigma), \tau \vdash_\Sigma Q \) (in the case where \( \llbracket C \rrbracket (\{ \sigma \}) \not\models \top \), then \( \llbracket C \rrbracket (\sigma) = \emptyset \), so it holds vacuously).

\( \square \)

\(^8\)Equivalent ways of expressing this include \(\neg Q \Rightarrow \top \) or \(\neg \top \Rightarrow Q \). Alternatively, if we modified the semantics of atomic assertions (Definition 8) to be \( S \not\models P \) iff \( \forall \sigma \in S, \sigma \vdash_\Sigma P \) (without requiring that \( S \not\in \emptyset \)), then we would have a more direct correspondence: \( \models \{ P \} C \{ Q \} \) iff \( \models \{ P \} C \langle Q \rangle \), but then \( P \lor Q \) would behave more like \( P \lor Q \) and not guarantee reachability.
Now, we will prove that MHL triples subsume Backwards Under-Approximate Triples as well. This time, we use the under-approximate variant of MHL 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).** \( \not\vDash \langle P \rangle \ C \ \{Q\} \) iff \( \vDash \langle P \rangle \ C \ \{Q\} \)

**Proof.**

\((\Rightarrow)\) Suppose that \( \not\vDash \langle P \rangle \ C \ \{Q\} \) or in other words, for any \( \sigma \vDash_\Sigma P \), there exists a \( \tau \in \llbracket C \rrbracket (\sigma) \) such that \( \tau \vDash_\Sigma Q \). Now suppose that \( S \not\vDash P \), 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 \( \not\vDash \langle P \rangle \ C \ \{Q\} \), then \( \exists \tau \in \llbracket C \rrbracket (\sigma) \) such that \( \tau \vDash_\Sigma Q \). Since \( \sigma \in S \), then \( S = \{\sigma\} \cup S \) and therefore by linearity, \( \llbracket C \rrbracket(S) = \llbracket C \rrbracket(\{\sigma\} \cup S) = \llbracket C \rrbracket(\{\sigma\}) \cup \llbracket C \rrbracket(S) = \llbracket C \rrbracket(\sigma) \cup \llbracket C \rrbracket(S) \) and since \( \tau \in \llbracket C \rrbracket(\sigma) \) then \( \tau \in \llbracket C \rrbracket(S) \) and so \( \exists \tau \in \llbracket C \rrbracket(S) \). \( \tau \vDash_\Sigma Q \). By Lemma B.4, we can therefore conclude that \( \llbracket C \rrbracket(S) \vDash_\Sigma Q \oplus \top \).

\((\Leftarrow)\) Suppose that \( \vDash \langle P \rangle \ C \ \{Q \oplus \top\} \) and so if \( S \not\vDash P \), then \( \llbracket C \rrbracket(S) \vDash_\Sigma Q \oplus \top \). Now suppose that \( \sigma \vDash_\Sigma P \). Then trivially \( \{\sigma\} \vDash P \), so we can use our assumption to conclude that \( \llbracket C \rrbracket(\{\sigma\}) \vDash_\Sigma Q \oplus \top \). Now, by Lemma B.4, there is some \( \tau \in \llbracket C \rrbracket(\sigma) \) such that \( \tau \vDash_\Sigma Q \). 

\(\Box\)

The combination of Theorem C.1 and Corollary 5.5 suggest that Backwards Under-Approximate triples can disprove any Hoare Triple as well (if the precondition \( \varphi \) 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 § 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 \( \not\vDash \langle P \rangle \ C \ \{Q\} \), then there is some other Incorrectness triple \( \not\vDash \langle P' \rangle \ C \ \{Q'\} \) that disproves it.

\( \not\vDash \langle P \rangle \ C \ \{Q\} \) iff \( \exists \varphi', \varphi'. \ P' \Rightarrow P \) and \( \varphi' \not\vDash Q \) and \( \vDash \langle P' \rangle \ C \ \{Q'\} \)

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 MHL triple (with semantic assertions) can be disproven by another MHL triple.

While this result shows the strength of the MHL 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 MHL instance, the syntactic proofs rely on some additional properties of the particular evaluation model. We lay out the requirements for a falsifiable instance of MHL 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 MHL triple. A semantic assertion is simply a set of satisfying models. We will use the uppercase greek metavariables to denote semantic assertions \( \Phi, \Psi \in 2^{\Sigma} \). The semantic interpretation of a syntactic assertion is the set of models that satisfies it (\( \{\varphi\} \triangleq \{m \mid m \in \Sigma, m \vDash \varphi\} \)). 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 \( \neg \Phi = 2^{\Sigma} \setminus \Phi \) and we say that an assertion is satisfiable \( \text{sat}(\Phi) \) iff \( \Phi \neq \emptyset \). This gives us the following
expected properties:

\[ \langle \varphi \rangle \Rightarrow \langle \psi \rangle \quad \text{iff} \quad \varphi \Rightarrow \psi \quad \quad m \in \neg \Phi \quad \text{iff} \quad m \not\in \Phi \quad \quad \text{sat}(\langle \varphi \rangle) \quad \text{iff} \quad \exists m \in M_\Sigma. \ m \models \varphi \]

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

\[ \models_S \langle \Phi \rangle C \langle \Psi \rangle \quad \text{iff} \quad \forall m \in M_\Sigma. \ m \in \Phi \quad \Longrightarrow \quad [C]^\dagger(m) \in \Psi \]

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

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

\[ \models_S \langle \Phi \rangle C \langle \Psi \rangle \quad \text{iff} \quad \vdash \langle \varphi \rangle C \langle \psi \rangle \]

**Proof.**

\((\Longrightarrow)\) Suppose that \( m \models \varphi \), then \( m \in \langle \varphi \rangle = \Phi \), so using \( \models_S \langle \Phi \rangle C \langle \Psi \rangle \), we know that \([C]^\dagger(m) \in \Psi = \langle \psi \rangle\). This means that \([C]^\dagger(m) \models \psi \), so \( \models \langle \varphi \rangle C \langle \psi \rangle \).

\((\Longleftarrow)\) Suppose that \( m \in \Phi = \langle \varphi \rangle \), then it must be that \( m \not\models \varphi \), so using \( \vdash \langle \varphi \rangle C \langle \psi \rangle \), we can conclude that \([C]^\dagger(m) \not\models \psi \). This means that \([C]^\dagger(m) \in \langle \psi \rangle = \Psi \), so therefore \( \models_S \langle \Phi \rangle C \langle \Psi \rangle \).

\(\square\)

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

**Theorem 5.1 (Semantic Falsification).** For any MHL instance and any program \( C \) and semantic assertions \( \Phi \), \( \Psi \):

\[ \models_S \langle \Phi \rangle C \langle \Psi \rangle \quad \text{iff} \quad \exists \Phi' \text{ such that } \Phi' \Rightarrow \Phi, \text{ sat}(\Phi'), \text{ and } \models_S \langle \Phi' \rangle C \langle \neg \Psi \rangle \]

**Proof.**

\((\Longrightarrow)\) Assume that \( \models_S \langle \Phi \rangle C \langle \Psi \rangle \), so that means that there is some \( m \in M_\Sigma \) such that \( m \in \Phi \) and \([C]^\dagger(m) \not\in \Psi \). By definition, this also means that \([C]^\dagger(m) \in \neg \Psi \). Now, let \( \Phi' = \{m\} \), so clearly \( \Phi' \Rightarrow \Phi \) (since \( \Phi' \subseteq \Phi \)) and sat(\( \Phi' \)). To see that \( \models_S \langle \Phi' \rangle C \langle \neg \Psi \rangle \), 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 \([C]^\dagger(m) \in \neg \Psi \)).

\((\Longleftarrow)\) Assume that there is some \( \Phi' \) such that \( \Phi' \Rightarrow \Phi, \text{ sat}(\Phi'), \text{ and } \models_S \langle \Phi' \rangle C \langle \neg \Psi \rangle \). Then, there must be some \( m \in \Phi' \) and so \( m \in \Phi \) as well. Since \( \models_S \langle \Phi' \rangle C \langle \neg \Psi \rangle \), then \([C]^\dagger(m) \in \neg \Psi \) and so \([C]^\dagger(m) \not\in \Psi \). We therefore know that \( m \in \Phi \) and \([C]^\dagger(m) \not\in \Psi \), so \( \models_S \langle \Phi \rangle C \langle \Psi \rangle \).

\(\square\)

**Theorem 5.2 (Principle of Denial).** For any MHL instance and any program \( C \) and syntactic assertions \( \varphi \), \( \varphi' \), and \( \psi \):

\[ \text{If } \varphi' \Rightarrow \varphi, \text{ sat}(\varphi'), \text{ and } \vdash \langle \varphi' \rangle C \langle \neg \psi \rangle \text{ then } \models \langle \varphi \rangle C \langle \psi \rangle \]

**Proof.** Let \( \Phi' = \langle \varphi' \rangle \), \( \Phi = \langle \varphi \rangle \), and \( \Psi = \langle \psi \rangle \). From our assumptions, we can conclude that \( \Phi' \Rightarrow \Phi \) and sat(\( \Phi' \)) and by Lemma D.1 we can conclude that \( \models_S \langle \Phi' \rangle C \langle \neg \Psi \rangle \). Therefore by Theorem 5.1, this implies that \( \models_S \langle \Phi \rangle C \langle \Psi \rangle \). Using Lemma D.1 again, we conclude that \( \models \langle \varphi \rangle C \langle \psi \rangle \).

\(\square\)
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 \( \not\models (\varphi) C \langle \psi \rangle \), then we know that there exists some \( m \) such that \( m \not\models \varphi \) and \( [C]^\dagger(m) \not\models \psi \) and this implies that \( [C]^\dagger(m) \not\models \neg \psi \) since we are working with classical interpretations of BI. Now, we have a single program execution starting at \( \varphi \) and ending at \( \neg \psi \), and we would like to extrapolate a valid MHL triple from this (possibly with a precondition stronger than \( \varphi \) 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 \( (C_1 \text{ or } C_2) \). This is possible, but only if the postcondition contains no implications. Logical negation is an implication (\( \neg \psi \) is shorthand for \( \psi \Rightarrow \bot \)), therefore we need a different postcondition \( \psi' \) that implies \( \neg \psi \), but is syntactically valid. The precise form of \( \psi' \) will depend on the BI instance.

In addition, the program \( C \) must terminate after finitely many steps, otherwise the preconditions 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 MHL 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 MHL). An instance of MHL is falsifiable if it has the following properties:

1. The PCM operation has the properties:
   a. If \( m_1 \otimes m_2 = \emptyset \), then \( m_1 = m_2 = \emptyset \).
   b. If \( m_1 \otimes m_2 = m_1 \otimes n_1 \), then there exist \( s_1, s_2, t_1, t_2 \) such that \( s_1 \otimes s_2 = n_1, t_1 \otimes t_2 = m_2, s_1 \otimes t_1 = m_1 \), and \( s_2 \otimes t_2 = m_2 \).

2. Atomic assertions \( P \) are splittable, that is if \( m_1 \otimes m_2 \models P \), then there exist \( \varphi_1 \) and \( \varphi_2 \) such that \( m_1 \models \varphi_1 \) and \( m_2 \models \varphi_2 \) and \( \varphi_1 \otimes \varphi_2 \models P \).

3. Sequences of outcomes are falsifiable, \( m \not\models Q_1 \oplus \cdots \oplus Q_n \), iff \( \exists \psi \) containing no implications such that \( m \models \psi \) and \( \psi \Rightarrow \bot \).

4. Atomic commands have trace extrapolation, if \( [C]^\dagger(m) \not\models \psi \), then there exists \( \varphi \) such that \( m \not\models \varphi \) and \( \not\models \langle \varphi \rangle C \langle \psi \rangle \) (where \( \varphi \) and \( \psi \) have no implications).

Lemma D.3 (Splitting). For any BI assertion \( \varphi \) that contains no implications and where the BI frame comes from a falsifiable MHL instance, if \( m_1 \otimes m_2 \not\models \varphi \), then there exist \( \varphi_1 \) and \( \varphi_2 \) such that \( m_1 \not\models \varphi_1 \) and \( m_2 \not\models \varphi_2 \) and \( \varphi_1 \otimes \varphi_2 \not\models \varphi \).

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

- \( \varphi = \top \). Clearly \( m_1 \not\models \top \) and \( m_2 \not\models \top \) and \( \top \otimes \top \Rightarrow \top \).
- \( \varphi = \bot \). Vacuous since \( m_1 \otimes m_2 \not\models \bot \) is impossible.
- \( \varphi = \top^\circ \). If \( m_1 \otimes m_2 \not\models \top^\circ \), then it must be the case that \( m_1 = m_2 = \emptyset \) (by property (1a) of Definition D.2). So, \( m_1 \not\models \top^\circ \) and \( m_2 \not\models \top^\circ \) and \( \top^\circ \otimes \top^\circ \Rightarrow \top^\circ \).
- \( \varphi = \psi' \land \psi \). We know that \( m_1 \otimes m_2 \not\models \psi' \land \psi \), so \( m_1 \otimes m_2 \not\models \psi' \) and \( m_1 \otimes m_2 \not\models \psi \). By the induction hypotheses, There are \( \varphi_1, \varphi_2, \psi_1, \) and \( \psi_2 \) such that \( m_1 \not\models \varphi_1 \) and \( m_2 \not\models \varphi_2 \) and \( m_1 \not\models \psi_1 \) and \( m_2 \not\models \psi_2 \) and \( \varphi_1 \otimes \varphi_2 \not\models \psi' \) and \( \psi_1 \otimes \psi_2 \not\models \psi \). Therefore, \( m_1 \not\models \varphi \land \psi_1 \) and \( m_2 \not\models \varphi \land \psi_2 \). Now, suppose \( m' \not\models (\varphi \land \psi_1) \oplus (\varphi \land \psi_2) \). Then \( m'_1 \not\models \varphi_1, m'_1 \not\models \psi_1, m'_2 \not\models \varphi_2, \) and \( m'_2 \not\models \psi_2 \) such that
\[ m'_1 \land m'_2 = m'. \] So, \( m' \models \varphi_1 \land \varphi_2 \) and \( m' \models \psi_1 \land \psi_2 \) and by the implications from the induction hypotheses, \( m' \models \psi' \) and \( m' \models \psi \), so \( m' \models \psi' \land \psi \).

- \( \varphi = \psi' \lor \psi \). We know that \( m_1 \land m_2 \models \psi' \lor \psi \), so \( n_1 \not\models \psi' \) and \( n_2 \not\models \psi \) such that \( n_1 \land n_2 = m_1 \land m_2 \).

By property (1b) of Definition D.2, there must be \( s_1, s_2, t_1 \) and \( t_2 \) such that \( s_1 \land s_2 = n_1 \), \( t_1 \lor t_2 = n_2 \), \( s_1 \land t_1 = m_1 \), and \( s_2 \lor t_2 = m_2 \). So, \( s_1 \land s_2 \not\models \psi' \) and \( t_1 \lor t_2 \not\models \psi \), and by the induction hypothesis, \( s_1 \models \varphi_1, s_2 \models \varphi_2, t_1 \models \psi_1 \) and \( t_2 \models \psi_2 \) such that \( \varphi_1 \land \varphi_2 \Rightarrow \psi' \land \psi_1 \) \( \psi_2 \Rightarrow \psi \). Recombining terms, we get that \( m_1 \models \varphi_1 \land \psi_1 \) and \( m_2 \models \varphi_2 \land \psi_2 \) and it is easy to see that \( \varphi_1 \land \varphi_2 \land \psi_1 \land \psi_2 \Rightarrow \psi' \land \psi \).

- \( \varphi \models \psi' \Rightarrow \psi \). Vacuous since we assumed \( \varphi \) has no implications.

- \( \varphi = P \). By property (2) from Definition D.2.

\[ \square \]

**Lemma D.4 (Trace Extrapolation).** For any falsifiable MHL instance, if there exists \( m \) such that \([C] \uparrow \downarrow m \models \psi \) (where \( \psi \) contains no implications), then there exists \( \varphi \) (also with no implications) such that \( m \models \varphi \) and \( \models \langle \varphi \rangle C \langle \psi \rangle \).

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

- \( C = 0 \). Assume that \([0] \uparrow \downarrow (m) \models \psi \). Since \([0] \uparrow \downarrow (m) = \varnothing \), then this assumption gives us \( \varnothing \models \psi \). We can then take \( \varphi = \top \) and derive \( \models \langle \varphi \rangle 0 \langle \psi \rangle \): for any \( m' \models \top \) we have \([0] \uparrow \downarrow (m') \models \psi \) since we know \( \varnothing \models \psi \) and \([0] \uparrow \downarrow (m') = \varnothing \), for any \( m' \). We also clearly have \( m \models \top \).

- \( C = 1 \). Assume \([1] \uparrow \downarrow (m) \models \psi \). Since \([1] \uparrow \downarrow (m) = m \), then this assumption gives us \( m \models \psi \). We can take \( \varphi = \psi \) and immediately derive \( m \models \varphi \). We can then also derive \( \models \langle \varphi \rangle 1 \langle \psi \rangle \): for any \( m' \models \varphi \) we have \([1] \uparrow \downarrow (m') = m' \models \psi \) since we know \( \varnothing \models \psi \).

- \( C = C_1 + C_2 \). Assume \([C_1 + C_2] \uparrow \downarrow (m) \models \psi \). We know that \([C_1 + C_2] \uparrow \downarrow (m) = [C_2] \uparrow \downarrow (m) \lor [C_1] \uparrow \downarrow (m)\), so \([C_1] \uparrow \downarrow (m) \models \psi_1 \) and \([C_2] \uparrow \downarrow (m) \models \psi_2 \) and \( \psi_1 \lor \psi_2 \rightarrow \psi \) By induction, there exist \( \varphi_i \) such that \( \models \langle \varphi_i \rangle C_i \langle \psi_i \rangle \) for \( i \in \{1, 2\} \) and \( m \models \varphi_i \). Now, we pick the precondition \( \varphi = \varphi_1 \land \varphi_2 \) (so \( m \models \psi \)). It remains to argue that \( \models \langle \varphi \rangle C_1 + C_2 \langle \psi \rangle \): for any \( m' \models \varphi_1 \land \varphi_2 \), we know that \( m' \models \varphi_1 \) and using the fact that \( \models \langle \varphi_1 \rangle C_1 \langle \psi_1 \rangle \), we conclude that \([C_1] \uparrow \downarrow (m') \models \psi_1 \) (for \( i = 1, 2 \)). Hence, \([C_1 + C_2] \uparrow \downarrow (m') \models \psi_1 \land \psi_2 \) and since \( \psi_1 \land \psi_2 \rightarrow \psi \) we can conclude \([C_1 + C_2] \uparrow \downarrow (m') \models \psi \).

- \( C = C_1 \sqcup C_2 \). Assume \([C_1 \sqcup C_2] \uparrow \downarrow (m) \models \psi \). We know that \([C_1 \sqcup C_2] \uparrow \downarrow (m) = [C_2] \uparrow \downarrow ([C_1] \uparrow \downarrow (m)) \), so \([C_2] \uparrow \downarrow ([C_1] \uparrow \downarrow (m)) \models \psi \). By the induction hypothesis, we conclude that there exists \( \delta \) such that \( \models \langle \delta \rangle C_2 \langle \psi \rangle \) and \([C_1] \uparrow \downarrow (m) \models \delta \). By induction again, we get that \( \models \langle \varphi \rangle C_1 \langle \delta \rangle \) such that \( m \models \varphi \). Now, to show that \( \models \langle \varphi \rangle C_1 \sqcup C_2 \langle \psi \rangle \), suppose that \( m \models \varphi \), then we know that \([C_1] \uparrow \downarrow (m') \models \delta \) (from \( \models \langle \varphi \rangle C_1 \langle \delta \rangle \)), and we know that \([C_2] \uparrow \downarrow ([C_1] \uparrow \downarrow (m')) \models \psi \) (from \( \models \langle \varphi \rangle C_2 \langle \psi \rangle \)), so therefore \( \models \langle \varphi \rangle C_1 \sqcup C_2 \langle \psi \rangle \).

- \( C = C^* \). We will first show that for any \( n \), there is a \( \varphi \) such that \( \models \langle \varphi \rangle C^n \langle \psi \rangle \) and \( m \models \varphi \).

The proof is by induction on \( n \). The case where \( n = 0 \) follows from the 1 case above. Now, by the induction hypothesis we know that there is some \( \delta \) such that \( \models \langle \delta \rangle C^n \langle \psi \rangle \) and \([C] \uparrow \downarrow (m) \models \delta \). By the previous induction hypothesis, we know that \( \models \langle \varphi \rangle C \langle \delta \rangle \), such that \( m \models \varphi \). So, combining these results, we get that \( \models \langle \varphi \rangle C^{n+1} \langle \psi \rangle \).

Now, since we assumed that the program terminates after finitely many steps, there must be some \( n \) such that \([C^*] \uparrow \downarrow (m) = [1] \uparrow \downarrow (m) \lor [C_1] \uparrow \downarrow (m) \lor \cdots \lor [C^n] \uparrow \downarrow (m) \). By repeatedly using Lemma D.3, we can split \( \psi \) into \( \psi_0, \ldots, \psi_n \) such that \([C^*] \uparrow \downarrow (m) \models \psi_k \) (for each \( k \in \{0, \ldots, n\} \)) and \( \psi_0 \lor \cdots \lor \psi_n \rightarrow \psi \). By the inductive proof above, for each \( k \), there is a \( \varphi_k \) such that \( \models \langle \varphi_k \rangle C^k \langle \psi \rangle \) and \( m \models \varphi_k \). Now, let \( \varphi = \varphi_0 \land \cdots \land \varphi_n \), so clearly \( m \models \varphi \). We can conclude that
\[ \equiv \langle \varphi \rangle \ C^* \langle \psi_0 \oplus \cdots \oplus \psi_n \rangle \] by an argument analogous to the \( C = C_1 + C_2 \) case. Finally, since \( \psi_0 \oplus \cdots \oplus \psi_n \Rightarrow \psi \), we can weaken the postcondition to obtain \( \equiv \langle \varphi \rangle \ C^* \langle \psi \rangle \).

\[ \bullet \ C = c. \text{ By property (4) of Definition D.2.} \]

\[\textbf{Theorem D.5 (Falsification).} \quad \forall \text{ any falsifiable MHL instance,} \]

\[ \equiv \langle \varphi \rangle \ C \ominus \mathcal{Q}_{i=1}^n Q_i \iff \exists \varphi' \Rightarrow \varphi \quad \text{and} \quad \exists \psi \Rightarrow \neg \ominus_{i=1}^n Q_i \quad \text{such that} \quad \equiv \langle \varphi' \rangle \ C \langle \psi \rangle \]

Where \( \psi \) has no implications and \( \text{sat}(\varphi') \).

\[\textbf{Proof.} \]

(\(\Rightarrow\)) Since \( \equiv \langle \varphi \rangle \ C \ominus \mathcal{Q}_{i=1}^n Q_i \), then there is an \( m \) such that \( m \equiv \varphi \) and \( \mathcal{C}[m] = \ominus_{i=1}^n Q_i \). By property (3) of Definition D.2, we know that there exists a \( \psi \) with no implications such that \( \psi \Rightarrow \neg \ominus_{i=1}^n Q_i \) and \( \mathcal{C}[m] = \psi \). We can now use Lemma D.4 to conclude that there is a \( \varphi' \) such that \( \equiv \langle \varphi' \rangle \ C \langle \psi \rangle \) and \( m \equiv \varphi' \). Now, let \( \varphi' = \varphi \land \varphi' \) (\( \varphi' \) is satisfiable since \( m \equiv \varphi \) and \( m \equiv \varphi' \)). Clearly also \( \varphi' \Rightarrow \varphi \). It just remains to show that \( \equiv \langle \varphi' \rangle \ C \langle \psi \rangle \) for any \( m' \equiv \varphi' \), then \( m' \equiv \varphi \) and \( \mathcal{C}[m'] = \psi \) (since \( \equiv \langle \varphi \rangle \ C \langle \psi \rangle \)).

(\(\Leftarrow\)) Assume that \( \varphi' \Rightarrow \varphi \) and \( \psi' \Rightarrow \neg \ominus_{i=1}^n Q_i \). By weakening, we can also conclude that \( \equiv \langle \varphi' \rangle \ C \ominus \mathcal{Q}_{i=1}^n Q_i \). By Theorem 5.2, we can conclude that \( \equiv \langle \varphi \rangle \ C \ominus_{i=1}^n Q_i \).

\[\textbf{Remark 1.} \quad \text{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 \( \mathcal{C}[m] = \psi \) implies that \( m \equiv \wp(C, \psi) \) and that \( \equiv \langle \wp(C, \psi) \rangle \ C \langle \psi \rangle \) is a valid triple. Unfortunately, weakest preconditions have complex interactions with choice mechanisms such as \( C_1 + C_2 \) and \( x \leftrightarrow \eta \) (refer to Kaminski [2019] for a more in-depth discussion of \( \wp \) and choice). The question of whether a weakest precondition exists for MHL 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.

\[\textbf{Lemma D.6 (Trace Extrapolation for Pure Commands).} \quad \forall \text{ any MHL instance where trace extrapolation holds for pure commands} \ c \text{ and basic assertions} \ P, Q, \text{ that is:} \]

\[ \mathcal{C}[m] = \psi \quad \Rightarrow \quad \exists \psi \text{ with no implications.} \quad m \equiv \varphi \quad \text{and} \quad \equiv \langle \varphi \rangle \ c \langle \psi \rangle \]

\[\textbf{Proof.} \quad \text{By induction on the structure of} \ \psi: \]

- \( \psi = T. \) Let \( \varphi = T. \) Clearly \( m \equiv T \) and \( \equiv \langle T \rangle \ c \langle T \rangle \): suppose \( m' \equiv T \), then clearly \( \mathcal{C}[m'] = T. \)
- \( \psi = \bot. \) Vacuous since \( \mathcal{C}[m] = \bot \) is impossible.
- \( \psi = \psi_1 \land \psi_2. \) Assume \( \mathcal{C}[m] = \psi_1 \land \psi_2. \) By induction, we know that there is a \( \varphi \) such that \( m \equiv \varphi \) and \( \equiv \langle \varphi \rangle \ c \langle \psi_i \rangle \) for \( i \in \{1, 2\} \). Now, let \( \varphi = \varphi_1 \land \varphi_2 \), so clearly \( m \equiv \varphi. \) We now show that \( \equiv \langle \varphi \rangle \ c \langle \psi_1 \lor \psi_2 \rangle \): suppose \( m' \equiv \varphi \), then \( m' \equiv \psi_i \) for \( i \in \{1, 2\} \). Since \( \equiv \langle \varphi \rangle \ c \langle \psi_i \rangle \), then \( \mathcal{C}[m'] = \psi_i \), therefore \( \mathcal{C}[m'] = \psi_1 \land \psi_2. \)
This section contains proofs for the falsification results in §5.1. The goal is to show that nondeterministic falsification.

D.3 Nondeterministic Falsification

This section contains proofs for the falsification results in §5.1. The goal is to show that nondeterministic instances of MHL are falsifiable by showing that Definition D.2 holds for instances of MHL 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 5.3 (Falsifying Assertions). For any $S \in 2^\Sigma$ and atomic assertions $Q_1, \ldots, Q_n$,

$$ S \not\vdash Q_1 \oplus \cdots \oplus Q_n \quad \text{iff} \quad \exists i. S \not\vdash \overline{Q}_i \quad \text{or} \quad S \vdash (\overline{Q}_1 \land \cdots \land \overline{Q}_n) \oplus \top \quad \text{or} \quad S \vdash \top^\oplus $$

Proof. Suppose that $S \not\vdash Q_1 \oplus \cdots \oplus Q_n$, so for all $S_1, \ldots, S_n$, if $S = \bigcup_{i=1}^n S_i$, there exists some $i$ such that $S_i \not\vdash Q_i$. Now, for each $i$, let $T_i = \{ \sigma \mid \sigma \in S, \sigma \not\vdash Q_i \}$, so by construction, $T_i \subseteq S$ and therefore $\bigcup_{i=1}^n T_i \subseteq S$. If $S \not\vdash \bigcup_{i=1}^n T_i$, then $S \supseteq \bigcup_{i=1}^n T_i$, so there must be some $\tau \in S$ such that for all $i$, $\tau \not\in T_i$ and so for all $i$, $\tau \not\vdash Q_i$, or in other words, $\tau \not\vdash \overline{Q}_i$. So, $S \not\vdash (\overline{Q}_1 \land \cdots \land \overline{Q}_n) \oplus \top$. 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 $T_i \not\vdash Q_i$. By construction, $\sigma \not\vdash Q_i$ for every $\sigma \in T_i$, so it must be that $T_i = \emptyset$. This means that $\sigma \not\vdash Q_i$ for every $\sigma \in S$, so $S \not\vdash \overline{Q}_i$. Or, if $S = \emptyset$, then $S \vdash \top^\oplus$.

Lemma D.7 (Nondeterministic Trace Extrapolation). If $[c]^\top(S) \not\vdash \psi$ and $\psi$ has no implications, then there is some $\varphi$ 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 $[\text{assume } e]^\top(S) \vdash \psi$. Let $S_1 = [\text{assume } e]^\top(S) = \{ \sigma \mid \sigma \in S, \sigma \vdash e \}$ and $S_2 = [\text{assume } \neg e]^\top(S) = \{ \sigma \mid \sigma \in S, \sigma \not\vdash e \}$. Clearly $S_1 \cup S_2 = S$, since the two assume statements partition $S$ into two parts. We now define $\varphi$ as follows:

$$ \varphi = \varphi_1 \oplus \varphi_2 \quad \text{where} \quad \varphi_1 = \begin{cases} \psi \land \bot & \text{if } S_1 \neq \emptyset \\ \psi \land \top^\oplus & \text{if } S_1 = \emptyset \end{cases} \quad \varphi_2 = \begin{cases} \overline{\psi} & \text{if } S_2 \neq \emptyset \\ \top^\oplus & \text{if } S_2 = \emptyset \end{cases} $$
We now show that $S_1 \not\models \phi_1$: we already know that $S_1 \not\models \psi$ 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 $S_1 = \emptyset \not\models T^\oplus$. A similar argument shows that $S_2 \not\models \phi_2$. Given this, $S \not\models \varphi$.

It remains to show that $\not\models \langle \varphi \rangle$ assume $e \langle \psi \rangle$. Suppose $T \not\models \varphi$, so $T_1 \not\models \varphi_1$ and $T_2 \not\models \varphi_2$ such that $T_1 \cup T_2 = T$. Since all the states satisfying $e$ from $T$ are in $T_1$, then $\lnot \exists \langle e \rangle(T) = T_1$. Since $T_1 \not\models \varphi_1$, then $T_1 \not\models \psi$.

• $c$ 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 $[c]^{\top}(S) \not\models Q$. This means that $[c]^{\top}(S) \neq \emptyset$ and $\forall \sigma [c]^{\top}(S) \models \varphi$. For pure commands, there are well known weakest precondition predicate transformations that satisfy $\tau \models Q$ iff $\sigma \models \operatorname{wp}(c, Q)$ such that $\{\tau\} = [c](\sigma)$. This includes the rules for variable assignment $(\operatorname{wp}(x := v, Q) = Q[v/a])$ as well as the backwards reasoning rules for Separation Logic given by Reynolds [2002]. So, it must be the case that $S \not\models \operatorname{wp}(c, Q)$: 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 $\operatorname{wp}(c, Q)$. Finally, we conclude that $\not\models \langle \operatorname{wp}(c, Q) \rangle c \langle Q \rangle$: suppose that $T \not\models \operatorname{wp}(c, Q)$, then $T \neq \emptyset$ and $\forall \sigma \models T. \sigma \models \operatorname{wp}(c, Q)$. By the properties of weakest preconditions, we know that $\tau \models Q$ iff $\{\tau\} = [c](\sigma)$, so everything in $[c]^{\top}(T)$ must satisfy $Q$. Additionally, $c$ is pure and cannot change the size of $T$, so $[c]^{\top}(T) \neq \emptyset$. This means that $[c]^{\top}(T) \not\models Q$.

□

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

**Proof.**

1. Properties of the PCM $\langle 2^X, \cup, \emptyset \rangle$:
   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 $S_1 \cup S_2 \not\models P$, then there are three options. If $S_1 = \emptyset$, then $S_1 \not\models T^\oplus$ and $S_2 \not\models P$ and clearly $T^\oplus \oplus P \Rightarrow P$. The case where $S_2 = \emptyset$ is symmetrical. Finally, if both $S_1$ and $S_2$ are nonempty, then $S_1 \not\models P$ and $S_2 \not\models P$ and $P \Rightarrow P$.
3. Assertion falsification: Follows from Lemma 5.3.
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 5.3) instead of existentially quantifying the postcondition.

**Theorem 5.4 (Nondeterministic Falsification).** For any MHL instance based on the nondeterministic evaluation model (Definition 7) and outcome logic (Definition 8), $\not\models \langle \varphi \rangle C \langle \bigoplus_{i=1}^n Q_i \rangle$ iff:

$$\exists \varphi' \Rightarrow \exists \text{ sat}(\varphi') \text{ and } \exists i. \not\models \langle \varphi' \rangle C \langle Q_i \rangle \text{ or } \not\models \langle \varphi' \rangle C \langle \bigwedge_{i=1}^n Q_i \rangle \text{ or } \not\models \langle \varphi' \rangle C \langle T^\oplus \rangle$$

**Proof.** Follows directly from Lemmas 5.3 and D.8 and theorem D.5. □

Now, we show that MHL can disprove any Hoare Triple, which means that it fully subsumes the use case of Incorrectness Logic.
Corollary 5.5 (Hoare Logic Falsification).
\[
\forall \{P\} C \{Q\} \iff \exists \varphi \Rightarrow P. \text{ sat}(\varphi) \text{ and } \vdash \langle \varphi \rangle C \langle \bar{Q} \rangle
\]

Proof.

(⇒) Assume \( \forall \{P\} C \{Q\} \). From Theorem 4.2, we get that \( \forall \langle P \rangle C \langle Q \cup T^0 \rangle \). This means that there is some \( S \) such that \( S \models P \) and \( [C] \langle S \rangle \models Q \cup T^0 \), which implies that \( [C] \langle S \rangle \not\models Q \) and \( [C] \langle S \rangle \models T^0 \), which implies that \( [C] \langle S \rangle \models \bar{Q} \oplus T \). Now, we can use Lemma D.4 to conclude that there is an assertion \( \vartheta \) such that \( S \models \vartheta \) and \( \vdash \langle \vartheta \rangle C \langle \bar{Q} \oplus T \rangle \). Now let \( \varphi = \vartheta \land P \), so clearly \( S \models \varphi \) and since \( \varphi \Rightarrow \vartheta \), then \( \vdash \langle \varphi \rangle C \langle \bar{Q} \oplus T \rangle \), or equivalently, \( \vdash \langle \varphi \rangle C \langle \bar{Q} \rangle \).

(⇐) Since \( \text{sat}(\varphi) \), there is some \( S \models \varphi \) and since \( \varphi \Rightarrow P \), then also \( S \models P \). From \( \vdash \langle \varphi \rangle C \langle \bar{Q} \rangle \), we know that there is a \( \tau \in [C] \langle S \rangle \) such that \( S \models \tau \). There must also be some \( \sigma \in S \) such that \( \tau \in [C] \langle \sigma \rangle \), and since \( \sigma \in S \) and \( S \models P \), then \( \sigma \models \tau \). So, we have now shown that \( S \models \tau \) and \( \tau \models Q \), therefore \( \forall \{P\} C \{Q\} \).

\( \square \)

D.4 Probabilistic Falsification

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

Lemma D.9 (Falsifying Probabilistic Assertions).
\[
\mu \not\vDash \bigoplus_{i=1}^{n} (P[A_i] = p_i) \iff \exists \psi (\text{with no implications}). \mu \models \psi \text{ and } \psi \Rightarrow \neg \bigoplus_{i=1}^{n} (P[A_i] = p_i)
\]

Proof.

(⇒) We begin by defining \( \psi \) as follows:
\[
\psi \triangleq \bigoplus_{x \in \{0,1\}^n} (P[B_x] = P_\mu[B_x]) \text{ where } B_x \triangleq \bigwedge_{i=1}^{n} \text{xor}(A_i, x_i)
\]
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 \( \sigma \models B_x \). This is because for each \( A_i \), either \( \sigma \models A_i \) or \( \sigma \models \neg A_i \), and so a unique \( B_x \) corresponds to these choices. That means that \( \mu \) can be partitioned by its support into sub-distributions \( \mu_x \) such that \( \forall \sigma \in \text{supp}(\mu_x). \sigma \models B_x \) and \( \mu = \sum_{x \in \{0,1\}^n} \mu_x \). Additionally, \( |\mu_x| = P_\mu[B_x] \) since \( \text{supp}(\mu_x) \) contains exactly those states that satisfy \( B_x \). Therefore, for each \( x, \mu_x \models (P[B_x] = P_\mu[B_x]) \) and so \( \mu \models \psi \).

Now we must show that \( \psi \Rightarrow \bigoplus_{i=1}^{n} (P[A_i] = p_i) \). Suppose that \( \eta \models \psi \). This means that \( \forall x \in \{0,1\}^n \) there is an \( \eta_x \) such that \( \eta_x \models (P[B_x] = P_\mu[B_x]) \) 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 \( \eta \models \bigoplus_{i=1}^{n} (P[A_i] = p_i) \). In order for this to be true, then for each \( i \), we would need the following:
\[
\sum_{x \in \{0,1\}^n, x_i = 0} \alpha_{x,i} \cdot \eta_x \not\models P[A_i] = p_i
\]
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 $\eta_x$ 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:
\[
\sum_{x \in \{0,1\}^n, x_i = 0} \alpha_{x,i} \cdot \mu_x = \mathbb{P}[A_i] = p_i
\]
And this implies that $\mu \models \bigoplus_{i=1}^n (\mathbb{P}[A_i] = p_i)$, which is a contradiction, therefore it must be the case that $\eta \models \neg \bigoplus_{i=1}^n (\mathbb{P}[A_i] = p_i)$.

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

\[\Box\]

The following lemma is needed for trace extrapolation.

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

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

- $\varphi = T$. Let $\psi = T$. Clearly $\mu \models T$ iff $\alpha \cdot \mu \models T$ since both are always true.
- $\varphi = \bot$. Let $\psi = T$. Clearly $\mu \models \bot$ iff $\alpha \cdot \mu \models \bot$ since both are always false.
- $\varphi = T^\circ$. Let $\psi = T^\circ$. Clearly $\mu \models T^\circ$ iff $\alpha \cdot \mu \models T^\circ$ since both are true iff $\mu = \emptyset$.
- $\varphi = \varphi_1 \land \varphi_2$. By the induction hypothesis, there exist $\psi_1$ and $\psi_2$ such that for any $\mu$, $\mu \models \varphi_1$ iff $\alpha \cdot \mu \models \psi_1$ for $i \in \{1,2\}$. Now, let $\psi = \psi_1 \land \psi_2$, so clearly $\mu \models \varphi_1$ iff $\alpha \cdot \mu \models \psi_1 \land \psi_2$.
- $\varphi = \varphi_1 \lor \varphi_2$. By the induction hypothesis, there exist $\psi_1$ and $\psi_2$ such that for any $\mu$, $\mu \models \varphi_1$ iff $\alpha \cdot \mu \models \psi_1$ for $i \in \{1,2\}$. Now, let $\psi = \psi_1 \lor \psi_2$. It must be that $\mu \models \varphi_1 \lor \varphi_2$ iff $\alpha \cdot \mu \models \psi_1$ and $\alpha \cdot \mu \models \psi_2$.
- $\varphi = \varphi_1 \Rightarrow \varphi_2$. By the induction hypothesis, there exist $\psi_1$ and $\psi_2$ such that for any $\mu$, $\mu \models \varphi_1$ iff $\alpha \cdot \mu \models \psi_1$ for $i \in \{1,2\}$. Now, let $\psi = \varphi_1 \Rightarrow \varphi_2$, so clearly $\mu \models \varphi_1$ iff $\alpha \cdot \mu \models \psi_1 \Rightarrow \psi_2$.
- $\varphi \equiv (\mathbb{P}[A] = p)$. Let $\psi = (\mathbb{P}[A] = \alpha \cdot p)$. It is easy to see that $\mu \equiv (\mathbb{P}[A] = p)$ iff $\alpha \cdot \mu \equiv (\mathbb{P}[A] = \alpha \cdot p)$ since $|\mu| = p$ iff $\alpha \cdot |\mu| = \alpha \cdot p$.

\[\Box\]

**Lemma D.11 (Probabilistic Trace Extrapolation).** If $\llbracket c \rrbracket^{\circ}(\mu) \models \psi$ and $\psi$ has no implications, then there is some $\varphi$ such that $\mu \models \varphi$ and $\llbracket \langle \varphi \rangle \rrbracket \in \langle \psi \rangle$.

**Proof.** By cases on the structure of $c$.

- $c = (x \leftarrow \eta)$. First, we know that:
\[
\llbracket x \leftarrow \eta \rrbracket^{\circ}(\mu) = \sum_{\sigma \in \text{supp}(\mu)} \sum_{\eta \in \text{supp}(\eta)} \eta(v) \cdot \llbracket x := v \rrbracket^{\circ}(\sigma) = \sum_{\sigma \in \text{supp}(\mu)} \llbracket x := v \rrbracket^{\circ}(\eta(v) \cdot \mu)
\]
So, we can apply Lemma D.3 many times to get a $\psi_0$ for each $v$ such that $\llbracket x := v \rrbracket^{\circ}(\eta(v) \cdot \mu) \models \psi_0$ and $\bigoplus_{\eta \in \text{supp}(\eta)} \llbracket \psi_0 \rrbracket \Rightarrow \psi$. Now, since $x := v$ is pure, we can use the next case of this proof to conclude that there is a $\varphi_0$ such that $\eta(v) \cdot \mu \models \varphi_0$ and $\llbracket \langle \varphi_0 \rangle \rrbracket x := v \llbracket \psi_0 \rrbracket$. Using Lemma D.10 (with $\alpha = \frac{1}{\eta(v)}$) we can get a $\varphi'_0$ such that $\mu' \models \varphi'_0$ iff $\eta(v) \cdot \mu' \models \psi_0$ (and therefore $\mu \models \varphi'_0$). Now, let $\varphi = \bigwedge_{\eta \in \text{supp}(\eta)} \varphi'_0$, so clearly $\mu \models \varphi$. We can also show that $\llbracket \langle \varphi \rangle \rrbracket x \leftarrow \eta \llbracket \psi \rrbracket$.

Suppose that $\mu' \models \varphi$. Then, $\mu' \models \varphi'_0$ for each $v$. This also means that $\eta(v) \cdot \mu' \models \psi_0$. Now, using $\llbracket \langle \varphi_0 \rangle \rrbracket x := v \llbracket \psi_0 \rrbracket$, we know that $\llbracket x := v \rrbracket^{\circ}(\eta(v) \cdot \mu') \models \psi_0$. Combining these, we get $\llbracket x \leftarrow \eta \rrbracket^{\circ}(\mu') \models \bigoplus_{\eta \in \text{supp}(\eta)} \psi_0$. This implies that $\llbracket x \leftarrow \eta \rrbracket^{\circ}(\mu') \models \psi$.

Noam Zilberstein, Derek Dreyer, and Alexandra Silva

Vol. 1, No. 1, Article . Publication date: October 2022.
\textbf{Lemma D.12.} The probabilistic instance of \textit{MHL} is falsifiable

\textbf{Proof.}

\begin{enumerate}
\item Properties of the PCM $\langle D\Sigma, +, \emptyset \rangle$:
\begin{enumerate}
\item If $\mu_1 + \mu_2 = \emptyset$, then it must be the case that $\mu_1 = \mu_2 = \emptyset$ since the monoid operation $+$ can only add probability mass, not remove it.
\item Suppose that $\mu_1 + \mu_2 = \eta_1 + \eta_2$. We now define the following:

\begin{align*}
\alpha_1 &= \lambda x. \min(\mu_1(x), \eta_1(x)) & \alpha_2 &= \lambda x. \eta_1(x) - \alpha_1(x) \\
\beta_1 &= \lambda x. \min(\mu_2(x), \eta_2(x)) & \beta_1 &= \lambda x. \eta_2(x) - \beta_2(x)
\end{align*}

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{align*}
\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{align*}

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

$\mu_1(x)$
\end{enumerate}
\end{enumerate}
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 $\mu_1 \lor \mu_2 \equiv \mathbb{P}[A] = p$, so $|\mu_1| + |\mu_2| = p$ and all the states in both supports satisfy $A$. That means that $\mu_1 \equiv (\mathbb{P}[A] = |\mu_1|)$ and $\mu_2 \equiv (\mathbb{P}[A] = |\mu_2|)$ 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 MHL 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 5.6 (Disjoint Falsification).** First, let $A_0 = \bigwedge_{i=1}^n \neg A_i$. If all the events are disjoint (for all $i \neq j$, $A_i \land A_j$ iff false), then:

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

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

**Proof.** In general, if all the $B_j$s are disjoint, then $\mu \equiv \bigoplus_{j=1}^m (\mathbb{P}[B_j] = r_j)$ 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 $\mu$. 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 $\exists \vec{q}. \mu \not\vDash \bigoplus_{i=0}^n (\mathbb{P}[A_i] = q_i)$. 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 $q_i \neq p_i$.

(⇒) Let each $q_i = \mathbb{P}_\mu[A_i]$, with the addition of $A_0$, the $A_i$s form a tautology, so they account for all the states in $\mu$ 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 $q_0 \neq 0$.

(⇐) 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 5.7 (Principle of Denial for Lower Bounds).**

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

**Proof.** We first show that $\mathbb{P}[\neg A] \geq q \Rightarrow \neg \mathbb{P}[A] \geq q$. Suppose that $\mu \not\vDash \mathbb{P}[\neg A] \geq q$, 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 $\not\vDash \langle \varphi' \rangle C \langle \neg (\mathbb{P}[A] \geq p) \rangle$. 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 §6.4.
E.1 Semantics of the Assertion Logic

Recall the syntax for separation logic.

\[ p \in \text{SL} ::= \textbf{emp} \mid \exists x.p \mid p \land q \mid p \lor q \mid p \Rightarrow q \mid p \ast q \mid p \rightarrow q \mid e \mid e_1 \leftarrow e_2 \mid e \leftrightarrow e \mid e \not\leftrightarrow e \]

First we define the disjoint union of two heaps \( \mathcal{H} \to \mathcal{H} \to \mathcal{H} \) as follows:

\[ h_1 \uplus h_2 \triangleq \lambda \ell. \begin{cases} h_1(\ell) & \text{if } \ell \in \text{dom}(h_1) \\ h_2(\ell) & \text{if } \ell \notin \text{dom}(h_1) \cap \text{dom}(h_2) \end{cases} \]

The satisfaction relation \( \models \subseteq (S \times \mathcal{H}) \times \text{SL} \) is defined as follows.

\[
\begin{align*}
(s, h) &\not\models \textbf{emp} \quad \text{iff} \quad \text{dom}(h) = \emptyset \\
(s, h) &\not\models \exists x.p \quad \text{iff} \quad (s, h) \not\models p[v/x] \text{ for some } v \\
(s, h) &\models p \land q \quad \text{iff} \quad (s, h) \models p \text{ and } (s, h) \models q \\
(s, h) &\models p \lor q \quad \text{iff} \quad (s, h) \models p \text{ or } (s, h) \models q \\
(s, h) &\models p \Rightarrow q \quad \text{iff} \quad \text{if } (s, h) \not\models p \text{ then } (s, h) \models q \\
(s, h) &\models p \ast q \quad \text{iff} \quad \exists h_1, h_2 \text{ such that } h = h_1 \uplus h_2 \text{ and } (s, h_1) \not\models p \text{ and } (s, h_2) \models q \\
(s, h) &\models p \rightarrow q \quad \text{iff} \quad \forall h_1, h_2 \text{ such that } h = h_1 \uplus h_2 \text{ if } (s, h_1) \not\models p \text{ then } (s, h_2) \not\models q \\
(s, h) &\models e \quad \text{iff} \quad \llbracket e \rrbracket(s) \models \text{true} \\
(s, h) &\models e_1 \leftarrow e_2 \quad \text{iff} \quad \llbracket e_1 \rrbracket(s) = \ell \text{ and } \text{dom}(h) = \{\ell\} \text{ and } h(\ell) = \llbracket e_2 \rrbracket(s) \\
(s, h) &\models e \leftrightarrow \neg e \quad \text{iff} \quad \llbracket e \rrbracket(s) = \ell \text{ and } \text{dom}(h) = \{\ell\} \text{ and } h(\ell) = \bot \\
\end{align*}
\]

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 \( x \mapsto v \) as syntactic sugar for \( x \mapsto v \ast \text{true} \) which is satisfied by any stack–heap pair \((s, h)\) where \( h([x](s)) = [v](s) \). The difference between \( x \mapsto v \) and \( x \mapsto v \) is very similar to the difference between over- and under-approximate versions of outcomes that we saw in §4, where we defined under-approximation \( m \downarrow P \) to be \( m \models P \oplus \top \).

E.2 Logical Operations on Errors

Let \( \models_A \subseteq A \times \text{Prop}_A \) and \( \models_B \subseteq A \times \text{Prop}_B \) be two logical satisfaction relations in which the assertion syntaxes \((\text{Prop}_A \text{ and Prop}_B)\) contain the usual logical constructs \( \text{true} \), \( \text{false} \), \( \land \), \( \lor \), and \( \rightarrow \). In addition, let \( \text{Prop} = \text{Prop}_A \times \text{Prop}_B \) and \( \models \subseteq (B + A) \times \text{Prop} \) is the satisfaction relation from Definition 6.1. We now add the following logical operations:

\[
\begin{align*}
\text{true} &\triangleq (\text{true}, \text{true}) \\
\text{false} &\triangleq (\text{false}, \text{false}) \\

\neg (p, q) &\triangleq (\neg p, \neg q)
\end{align*}
\]

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 \( m, p, p', q, q' \).

- **True:** \( m \models \text{true} \)
- **False:** \( m \not\models \text{false} \)
- **Conjunction:** \( m \not\models (p, q) \land (p', q') \) iff \( m \not\models (p, q) \) and \( m \not\models (p', q') \)
- **Disjunction:** \( m \not\models (p, q) \lor (p', q') \) iff \( m \not\models (p, q) \) or \( m \not\models (p', q') \)
- **Negation:** \( m \not\models \neg (p, q) \) iff \( m \not\models (p, q) \)
- **Sugar Syntax:** \( (\text{ok} : p) \lor (\text{er} : q) \) iff \( (p, q) \)

**Proof.** We prove each case assuming that \( m = 1_L(b) \). The cases where \( m = 1_R(a) \) are symmetric.
Lemma 6.6 (Manifest Error Characterization).

Proof. First, recall that by definition, \( \text{HeapLang} \) as to allow some location \( \text{HeapLang} \) Recall the syntax of the atomic

\[ \text{E.3 Semantics of Programs} \]

Recall the syntax of the atomic \( \text{HeapLang} \) commands.

\[
\begin{align*}
\text{ assume } e \mid x := e \mid x := \text{alloc}() \mid \text{free}(e) \mid x \leftarrow [e] \mid [e_1] \leftarrow e_2 \mid \text{error}()
\end{align*}
\]

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

\[
\begin{align*}
\semantics{\text{assume } e}(s, h) &= \begin{cases} 
\text{unit}_M((s, h)) & \text{if } [e](s) \neq 0 \\
\varnothing & \text{otherwise}
\end{cases} \\
\semantics{x := e}(s, h) &= \text{unit}_M((s[x \mapsto [e](s)], h)) \\
\semantics{x := \text{alloc}()}(s, h) &= \text{unit}_M((s[x \mapsto \ell], h[\ell \mapsto \text{null}])) \quad \text{where } \ell = \max(\text{dom}(h)) + 1 \\
\semantics{\text{free}(e)}(s, h) &= \begin{cases} 
\text{unit}_M((s, h[\ell \mapsto \bot]) & \text{if } [e](\sigma) = \ell \text{ and } \ell \in \text{dom}(h) \text{ and } h(\ell) \neq \bot \\
\text{unit}_M((s, h)) & \text{otherwise}
\end{cases} \\
\semantics{[e_1] \leftarrow e_2}(s, h) &= \begin{cases} 
\text{unit}_M((s[\ell \mapsto [e_2](\sigma)], h) & \text{if } [e_1](\sigma) = \ell \text{ and } \ell \in \text{dom}(h) \text{ and } h(\ell) \neq \bot \\
\text{unit}_M((s, h)) & \text{otherwise}
\end{cases} \\
\semantics{x \leftarrow [e]}(s, h) &= \begin{cases} 
\text{unit}_M((s[x \mapsto h(\ell), h], h) & \text{if } [e](s) = \ell \text{ and } \ell \in \text{dom}(h) \text{ and } h(\ell) \neq \bot \\
\text{unit}_M((s, h)) & \text{otherwise}
\end{cases} \\
\semantics{\text{error}}(s, h) &= \text{unit}_M((s, h)))
\end{align*}
\]

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

\[
\semantics{x := \text{alloc}()}(s, h) = \{ i_R((s[x \mapsto \ell], h[\ell \mapsto v], v)) \mid \ell \in \mathbb{N}^+, v \in \text{Val}, \ell \notin \text{dom}(h) \lor h(\ell) = \bot \}
\]

E.4 Manifest Errors

Lemma 6.6 (Manifest Error Characterization).

\[ \forall \sigma. \exists \sigma \in \semantics{C}(\sigma) \text{ such that } \tau \vdash (er : q \ast true) \]

\[ \vdash \semantics{[p]} C \semantics{er : q} \text{ is a manifest error} \quad \text{iff} \quad \vdash (ok : true) C \semantics{er : q \ast true} \]

Proof. First, recall that by definition, \( \vdash \semantics{[p]} C \semantics{er : q} \) is a manifest error \( \forall \sigma. \exists \sigma \in \semantics{C}(\sigma) \text{ such that } \tau \vdash (er : q \ast true) \).

\( (\Rightarrow) \) Suppose that \( S \vdash (ok : true) \). This means that there must be a \( \sigma \) such that \( i_R(\sigma) \in S \). By the definition of manifest errors, we know that \( \exists \sigma \in \semantics{C}(\sigma) \text{ such that } \tau \vdash (er : q \ast true) \).

Now, since \( \semantics{C}(\sigma) = \semantics{C}^\downarrow(i_R(\sigma)) \) and \( i_R(\sigma) \subseteq S \), then \( \tau \in \semantics{C}^\downarrow(S) \) and so \( \semantics{C}^\downarrow(S) \vdash (er : q \ast true) \). Therefore, \( \vdash (ok : true) C \semantics{er : q \ast true} \).

\( (\Leftarrow) \) Let \( \sigma \) be any program state. From \( \vdash (ok : true) C \semantics{er : q \ast true} \), we know that \( \semantics{C}^\downarrow(i_R(\sigma)) \vdash (er : q \ast true) \). So, by Lemma B.4 there must be some \( \tau \in \semantics{C}(\sigma) \) such that \( \tau \vdash (er : q \ast true) \) (since \( \semantics{C}^\downarrow(i_R(\sigma)) = \semantics{C}(\sigma) \)).

\[ \Box \]
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:

\[
\frac{\left( \bigoplus_{i=1}^{n} (e_i : p_i) \right) C \left( \bigoplus_{j=1}^{m} (e_j' : q_j) \right)}{\left( \bigoplus_{i=1}^{n} (e_i : p_i + r) \right) C \left( \bigoplus_{j=1}^{m} (e_j' : q_j + r) \right)} \quad \text{fv}(r) \cap \text{mod}(C) = \emptyset}
\]

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 $\text{fv}(r) \cap \text{mod}(C) = \emptyset$.

F SOUNDNESS PROOFS

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

**Proof.** By induction on the derivation $\vdash (P) C (Q)$.

- **Zero.** Suppose that $m \not\models \varphi$. We know that $[0]^\uparrow(m) = \emptyset$ and $\emptyset \not\models \top$, therefore $\not\vdash \langle \varphi \rangle \langle \top \rangle$.
- **One.** Suppose that $m \not\models \varphi$. The know that $[1]^\uparrow(m) = m$ and we assumed that $m \not\models \varphi$, so $\not\vdash \langle \varphi \rangle 1 \langle \varphi \rangle$.
- **Seq.** Suppose that $m \not\models \varphi$. By induction, we know that $[C_1]^\uparrow(m) \models \psi$. By induction again, we know that $[C_2]^\uparrow([C_1]^\uparrow(m)) \models \vartheta$. In addition:

  \[
  [C_2]^\uparrow([C_1]^\uparrow(m)) = \text{bind}([C_1]^\uparrow(m), [C_2])
  = \text{bind}(\text{bind}(m, [C_1], [C_2]))
  = \text{bind}(m, \lambda \sigma. \text{bind}([C_1](\sigma), [C_2]))
  = \text{bind}(m, [C_1; C_2])
  = [C_1; C_2]^\uparrow(m)
  \]

So, $[C_1; C_2]^\uparrow(m) \models \vartheta$ and therefore $\not\vdash \langle \varphi \rangle C_1; C_2 \langle \vartheta \rangle$.

- **For.** Since for $N \in C$ is syntactic sugar for $C^n$ (or, equivalently, $C; \ldots; C$), this rule can be derived by induction on $N$ using the SEQ use.
- **Split.** Suppose $m \not\models \varphi_1 + \varphi_2$, then there exists $m_1$ and $m_2$ such that $m_1 \cdot m_2 = m$ and $m_1 \not\models \varphi_1$ and $m_2 \not\models \varphi_2$. By induction, we know that $[C]^\uparrow(m_1) \models \psi_1$ and $[C]^\uparrow(m_2) \models \psi_2$. By linearity, we know that $[C]^\uparrow(m_1 \cdot m_2) = [C]^\uparrow(m_1) \odot [C]^\uparrow(m_2) = [C]^\uparrow(m)$. Note that this does not necessarily mean that $[C]^\uparrow(m)$ is defined, but if we limit $C$ to be syntactically valid (as described in Appendix A), then it must be defined and so $[C]^\uparrow(m) \models \psi_1 + \psi_2$.
- **Consequence.** Suppose that $m \not\models \varphi \Rightarrow$. By the assumption that $\varphi \Rightarrow \varphi$, this means that $m \not\models \varphi$. By induction, we know that $[C]^\uparrow(m) \models \psi$ and so $[C]^\uparrow(m) \models \psi$ (since $\psi \Rightarrow \psi$) and therefore $\not\vdash \langle \varphi \Rightarrow \rangle C (\psi \Rightarrow)$.
- **Empty.** Suppose that $m \not\models \top$, then $m = \emptyset$. We also know that $\text{bind}(\emptyset, f) = \emptyset$ for any $f$, so $[C]^\uparrow(\emptyset) = \emptyset$, therefore $[C]^\uparrow(m) \not\models \top$.

\[\square\]
Lemma F.2 (Soundness of nondeterministic rules in Figure 4). If \( \vdash (P) \ C \ (Q) \) then \( \vDash (P) \ C \ (Q) \).

Proof. By induction on the derivation \( \vdash (P) \ C \ (Q) \).

- **PLUS.** Suppose that \( m \Vdash \varphi \). By induction, we know that \( [C_1] \uparrow (m) \vDash \psi_1 \) and \( [C_2] \uparrow (m) \vDash \psi_2 \). By the definition of \([\cdot] \) we also know that \( [C_1] \uparrow (m) \cup [C_2] \uparrow (m) = [C_1 + C_2] \uparrow (m) \) and therefore \( [C_1 + C_2] \uparrow (m) \vDash \psi_1 \cup \psi_2 \).

- **INDUCTION.** Suppose that \( m \not\vDash \varphi \). We know by induction that \( [1 + C ; C^*] \uparrow (m) \not\vDash \psi \). Let \( F = \lambda f. \lambda \sigma. f \uparrow ([C] \, (\sigma)) \cup \text{unit}(\sigma) \) and note that:

\[
[C^*] \uparrow (m)
\]

So, \( [C^*] \uparrow (m) \not\vDash \psi \).

\[\square\]

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

Proof. By induction on the derivation \( \vdash (P) \ C \ (Q) \).

- **ASSUME.** Suppose that \( m \not\vDash P_1 \oplus P_2 \). Since \( P_1 \vDash e \) and \( P_2 \vDash \neg e \), we know by the definition of expression entailment that \( \lceil \text{assume } e \rceil \uparrow (m) \not\vDash P \).

- **ASSIGN.** Suppose that \( m \not\vDash P[e/x] \). By the required properties of substitution, we know that \( \lceil x := e \rceil \uparrow (m) \not\vDash P \).

- **If.** Suppose that \( m \not\vDash P_1 \oplus P_2 \). Now observe that:

\[
\lceil \text{if } e \text{ then } C_1 \text{ else } C_2 \rceil \uparrow (m) = \lceil (\text{assume } e ; C_1) + (\text{assume } \neg e ; C_2) \rceil \uparrow (m) \\
= \lceil \text{assume } e ; C_1 \rceil \uparrow (m) \odot \lceil \text{assume } \neg e ; C_2 \rceil \uparrow (m) \\
= [C_1] \uparrow (\lceil \text{assume } e \rceil \uparrow (m)) \odot [C_2] \uparrow (\lceil \text{assume } \neg e \rceil \uparrow (m))
\]

Now, let \( m_1 = \lceil \text{assume } e \rceil \uparrow (m) \) and \( m_2 = \lceil \text{assume } \neg e \rceil \uparrow (m) \). 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).

\[
[C_1] \uparrow (m_1) \odot [C_2] \uparrow (m_2)
\]

By the induction hypotheses, we also know that \( [C_1] \uparrow (m_1) \vDash Q_1 \) and \( [C_2] \uparrow (m_2) \vDash Q_2 \), therefore \( [C_1] \uparrow (m_1) \odot [C_2] \uparrow (m_2) \vDash Q_1 \oplus Q_2 \). Note that this composition with \( \odot \) 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.

\[\square\]

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

\[
\frac{\Gamma \text{pure } (p) \ C \ (q) \quad \text{pure seq}(C)}{(p) \ C \ (q)} \quad \text{NONDETERMINISTIC LIFT}
\]
Proof. By induction on the derivation ⊢ ⟨p⟩ C ⟨q⟩. Suppose that S ⊨ p, so that means that S ≠ ∅ and ∀σ ∈ S. σ ⊨ p. We know by induction that for any σ ⊨ p, there is some τ such that [C]⁺(σ) = {τ} and τ ⊨ q. We also know that [C]⁺(S) = ∪σ∈S [C]⁺(σ) and since each for each σ, there is a τ such that [C]⁺(σ) = {τ}, then ∀τ ∈ [C]⁺(S), τ ⊨ q and so [C]⁺(S) ⊨ q.

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

\[ \begin{array}{c}
\text{pure}_M(C) \\
\hline
\vdash_{\text{pure}} \langle \text{er : p} \rangle C \langle \text{er : p} \rangle
\end{array} \]

Proof. Suppose that m = (er : p), and so there must be some σ such that m = i_L(σ) and i_L(σ) ⊨ (er : p). Now, we have:

\[ [C]⁺(\text{unit}_M(i_L(σ))) = \text{bind}_M(\text{unit}_M(i_L(σ)), \lambda x. \begin{cases} [C]⁺(y) & \text{if } x = i_R(y) \\ \text{unit}_M(x) & \text{if } x = i_L(y) \end{cases} \]

And since we already know that i_L(σ) ⊨ (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 ⊢ ⟨P⟩ C ⟨Q⟩.

- Lifting. Suppose that μ ⊨ (P[A] = p), so for every σ ∈ supp(μ), σ ⊨ A and |μ| = p. We know by induction that for any σ there is some τσ such that [C]⁺(σ) = δτσ and τσ ⊨ B. So, μ' = [C]⁺(μ) = ∑σ∈supp(μ) μ(σ) · [C]⁺(σ) = ∑σ∈supp(μ) μ(σ) · δτσ. Therefore |μ'| = |μ| and ∀τ ∈ supp(μ'), τ ⊨ B, so μ' ⊨ (P[B] = p).

- Sample. First, observe that:

\[ [x \leftarrow \eta]⁺(μ) = \text{bind}(μ, λσ. \text{bind}(η, λv. [x := v](σ))) = \sum_{σ \in \text{supp}(μ)} \sum_{v \in \text{supp}(η)} μ(σ) · [x := v]⁺(σ) = \sum_{v \in \text{supp}(η)} [x := v]⁺(μ) \]

Now, by the same argument that we used in the lifting cases, since μ ⊨ (P[A] = p) and (A) x := v (B_v), then [x := v]⁺(μ) ⊨ (P[B_v] = p). Therefore, we can also weight the distribution to obtain η(v) · [x := v]⁺(μ) ⊨ (P[B_v] = η(v) · p). Now, the sum over v ∈ supp(η) corresponds exactly to an outcome conjunction, so we have:

\[ \sum_{v \in \text{supp}(η)} η(v) · [x := v]⁺(μ) = \bigoplus_{v \in \text{supp}(η)} (P[B_v] = η(v) · p) \]

Lemma F.7 (Correctness of Expression Entailment). If m ⊨ P ⊕ Q and P ⊨ e and Q ⊨ ¬e, then [assume e]⁺(m) ⊨ P for both the nondeterministic and probabilistic interpretations of expression entailment.

Proof.
• NONDETERMINISM. First note that \(\{\text{assume } e\}^{\updownarrow}(S) = \{\sigma \mid \sigma \in S, [e]_{\text{ex}}(\sigma) = \text{true}\}\). 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 \(S_1 \not\vdash P\) \(\text{and}\) \(S_2 \not\vdash Q\). Depending on which atomic assertions we are using, \(P\) is either some assertion \(p\) or \((\text{ok} : p)\), 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 \([\text{assume } e]\{S_1\} = S_1\). By a similar argument, \([\text{assume } e]\{S_2\} = \emptyset\). Therefore \([\text{assume } e]\{S\} = S_1 \cup \emptyset = S_1\) and we already know that \(S_1 \not\vdash P\).

• 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 \(\mu_2 \not\vdash (\mathbb{P}[B] = q)\) 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 \([\text{assume } e]\{\mu_1\} = \mu_1\). The opposite is true for \(\mu_2\), so \([\text{assume } e]\{\mu_2\} = \emptyset\). Therefore \([\text{assume } e]\{\mu\} = \mu_1 + \emptyset = \mu_1\) and we already know that \(\mu_1 \not\vdash (\mathbb{P}[A] = p)\).

\[\square\]

G ADDITIONAL RULES FOR CONDITIONALS AND LOOPS

As mentioned in §4.3, fully generic looping rules for MHL 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:

\[
\forall i. P_i \vdash e \quad \forall i. Q_i \vdash \neg e \quad \forall i. \langle P_i \rangle C \langle P_{i+1} \oplus Q_{i+1} \rangle \Rightarrow \langle P_0 \oplus Q_0 \rangle \text{ while } e \text{ do } C \langle \bigvee_{i=0}^{n} Q_i \oplus \top \rangle \quad \text{Bounded Unrolling}
\]

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 \(\langle P_i\rangle\) are passed forward into the next iteration whereas the false components that cause the loop to exit \(\langle Q_i\rangle\) 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:

\[
\forall i > 0. (p_i \Rightarrow e) \land e_i = \text{ok} \quad \forall i \in \mathbb{N}. \langle e_{i+1} : p_{i+1} \rangle C \langle e_i : p_i \rangle \quad \langle p_0 \Rightarrow \neg e \rangle \lor (e_0 = \text{er}) \Rightarrow \langle e_n : p_n \rangle \text{ while } e \text{ do } C \langle e_0 : p_0 \rangle \quad \text{While}
\]

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:

\[
\langle \text{ok} : p \land e \rangle C_1 \langle Q \rangle \quad \langle \text{ok} : p \land \neg e \rangle C_2 \langle Q \rangle \quad \langle \text{ok} : p \rangle \text{ if } e \text{ then } C_1 \text{ else } C_2 \langle Q \rangle \quad \text{If (Single Outcome)}
\]