Semantic Completeness of Higher-Order Probabilistic Separation Logics

Puming Liu
New York University Shanghai
Shanghai, China
puming.liu@nyu.edu

Abstract

Various models of higher-order probabilistic separation logics have been developed in recent years, raising natural questions about their expressiveness and theoretical foundations. This work establishes semantic completeness theorems for three Iris-based program logics—Eris$_t$, Tachis, and Approxis—restricted to pure programs. For Eris$_t$, a total-correctness separation logic based on error credits, completeness means that every program that terminates with probability 1 admits a proof of termination in the logic. Tachis, which reasons about expected running time via time credits, admits an analogous result. For Approxis, a relational logic for probabilistic refinement, we establish a relational completeness theorem: whenever an approximate coupling exists between two programs, a relational weakest precondition can be derived. All results are mechanized in the Rocq proof assistant, and can be seen as converses of the respective adequacy theorems. These are the first completeness proofs for Iris-based program logics.

Keywords

separation logic, probabilistic programs, completeness, Iris, Rocq, weakest precondition, program verification

1 Introduction

There has long been interest in formal systems for reasoning about probabilistic programs. Probabilistic programs arise across many domains—randomized algorithms, cryptographic protocols, and machine learning systems—and their correctness properties are inherently distributional, expressed as guarantees about the likelihood of outcomes or the distribution of costs. Verifying such properties rigorously is a central challenge in programming language theory: probabilistic arguments are notoriously error-prone, and the gap between intuition and formal proof is often wide. A long line of work has studied proof systems, type theories, and semantic models tailored to probabilistic computation, with the goal of providing sound and usable tools for program verification.

Separation logic, introduced by Reynolds and O'Hearn [15, 16], has had a transformative influence on program verification. Its central innovation—the separating conjunction $P * Q$, which asserts that P and Q hold over disjoint portions of state—enables local, modular proofs that scale to large programs and complex data structures. Separation logic underpins industrial-strength verification tools such as Facebook's Infer and underlies the Iris framework [11], which has been used to verify a wide range of language features, including concurrency, higher-order functions, and fine-grained memory models.

Recently, a number of works have introduced separation logic to probabilistic program verification [2, 3, 5, 17]. In particular, several of these logics have been based on the Iris separation logic framework. In this work, we show that three of these—Eris, Tachis, and Approxis—are semantically complete for pure programs (i.e., programs without mutable state): if the probability distribution generated by a pure program satisfies an appropriate property, then we can derive a proof of a corresponding specification in the program logic by directly constructing a proof via the semantic definition of the weakest precondition predicate. These results have all been mechanized in Rocq, and are converses of these logics' *soundness* or *adequacy* theorems.

Completeness of proof systems is a question of fundamental theoretical interest. In classical logic, Gödel's completeness theorem established that every semantically valid first-order formula is provable, drawing a sharp boundary between syntax and semantics. For program logics, the analogous question—whether every semantically valid specification is derivable—is considerably more subtle. Cook [4] gave a foundational answer for Hoare logic, showing relative completeness: any valid triple is provable, given an expressive enough assertion language. Completeness results for richer settings, including higher-order programs and resource-aware logics, remain an active area of investigation, as the interplay between logical framework, semantic model, and language features introduces new obstacles that must be overcome. Our work presents the *first* completeness proofs for Iris-based program logics, filling a theoretical gap in recent work on Iris.

This extended abstract covers the completeness results for Eris$_t$ [1] and Tachis [7] in dedicated sections (§3 and 4); the result for Approxis is briefly discussed in §5. Eris$_t$ is a separation logic that uses a special resource, called error credits, to track and bound the probability of error. It provides a modular way to reason about probabilistic termination of higher-order programs. For Eris$_t$, completeness implies that for every program whose termination probability is 1, we can derive a proof of termination in the program logic. Tachis replaces error credits with *time credits* to reason about expected running time, and admits an analogous completeness theorem. Approxis [6] reasons about refinement of two probabilistic programs and admits a relational completeness result.

2 Preliminaries

In this section, we present some basic definitions in probability theory, the semantics of our language, and a brief background on separation logic.

2.1 Probability Theory

We model program execution using *subdistributions*, which allow the total probability mass to be strictly less than 1. The missing mass accounts for non-terminating executions: a program that diverges with probability p is represented by a subdistribution whose weights sum to at most $1 - p$.

Definition 2.1 (Distribution). *A discrete subdistribution (henceforth simply* distribution*) on a countable set A is a function $\mu : A \to [0,1]$ such that $\sum_{a \in A} \mu(a) \leq 1$. The collection of distributions on A is denoted by $\mathcal{D}(A)$.*

Definition 2.2 (Probability Mass). *Given a distribution $\mu \in \mathcal{D}(A)$ and a predicate $\phi \subseteq A$, we write $\Pr_\mu[\phi]$ to denote the probability that ϕ holds, i.e., $\sum_{a \in \phi} \mu(a)$.*

Lemma 2.3 (Discrete Distribution Monad). *We can equip $\mathcal{D}$ with a monadic structure, with operations*

$$\mathrm{ret} : A \to \mathcal{D}(A) \qquad \mathrm{bind} : (A \to \mathcal{D}(B)) \to \mathcal{D}(A) \to \mathcal{D}(B)$$

$$\mathrm{ret}(a)(a') \triangleq \begin{cases} 1 & \text{if } a = a' \\ 0 & \text{otherwise} \end{cases} \qquad \mathrm{bind}(f, \mu)(b) \triangleq \sum_{a \in A} \mu(a) \cdot f(a)(b)$$

We use the notation $\mu \gg\!= f$ for $\mathrm{bind}(f, \mu)$.

2.2 The RandML$_p$ Language and Operational Semantics

Eris, Approxis, and Tachis all work with RandML [6], a higher-order ML-like probabilistic language with mutable state and presampling tapes. In this work, we consider RandML$_p$, a subset of RandML that removes mutable state and presampling tapes, retaining its core higher-order lambda calculus with random sampling. The syntax is defined by the grammar below.

Definition 2.4 (The Grammar of RandML$_p$).

$$v, w \in \mathit{Val} ::= z \in \mathbb{Z} \mid b \in \mathbb{B} \mid () \mid \mathsf{rec\ f\ x} = e \mid (v, w) \mid \mathsf{inl}\, v \mid \mathsf{inr}\, v$$

$$e \in \mathit{Expr} ::= v \mid \mathsf{x} \mid \mathsf{rec\ f\ x} = e \mid e_1\ e_2 \mid e_1 + e_2 \mid e_1 - e_2 \mid \ldots \mid \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid (e_1, e_2) \mid \mathsf{fst}\, e \mid \ldots \mid \mathsf{rand}\ e$$

$$K \in \mathit{Ectx} ::= - \mid e\,K \mid K\,v \mid \mathsf{rand}\,K \mid \mathsf{inr}\,K \mid \mathsf{inl}\,K \mid \ldots$$

The language feature of particular interest is rand e: when e evaluates to a natural number N, it samples uniformly at random from $\{0, 1, \ldots, N\}$ and returns the result. All other constructs are standard: rec f x = e defines a (possibly recursive) function, arithmetic operators work as expected, and if /fst/snd/inl/inr provide the usual control flow and product/sum type eliminators.

Operational Semantics. To define full program execution, we define $\mathrm{step}(e) \in \mathcal{D}(\mathit{Expr})$, the distribution induced by the single step reduction of program $e \in \mathit{Expr}$. The semantics of step is standard: all non-probabilistic constructs reduce deterministically as usual, *e.g.*, $\mathrm{step}(\mathsf{if\ true\ then}\ e_1\ \mathsf{else}\ e_2) = \mathrm{ret}(e_1)$, and the probabilistic choice operator rand N reduces uniformly at random:

$$\mathrm{step}(\mathsf{rand}\ N)(n) = \begin{cases} \frac{1}{N+1} & \text{for } n \in \{0, 1, \ldots, N\}, \\ 0 & \text{otherwise.} \end{cases}$$

With the single step reduction step defined, we now define a stratified execution probability $\mathrm{exec}_n : \mathit{Expr} \to \mathcal{D}(\mathit{Val})$ by induction on n:

$$\mathrm{exec}_n(e) \triangleq \begin{cases} \mathbf{0} & \text{if } e \notin \mathit{Val} \text{ and } n = 0, \\ \mathrm{ret}(e) & \text{if } e \in \mathit{Val}, \\ \mathrm{step}(e) \gg\!= \mathrm{exec}_{(n-1)} & \text{otherwise.} \end{cases}$$

where $\mathbf{0}$ denotes the everywhere-zero distribution. Intuitively, $\mathrm{exec}_n(e)$ is the distribution over terminal values that e produces when given at most n reduction steps; executions that require more than n steps contribute to the "missing mass" (the total probability may be strictly less than 1). The probability that a full execution, starting from program e, reaches a value v is taken as the limit of its stratified approximations:

$$\mathrm{exec}(e)(v) \triangleq \lim_{n \to \infty} \mathrm{exec}_n(e)(v)$$

2.3 Separation Logic

The program logics that we mentioned are all built on top of the Iris separation logic framework [11, 12]. We briefly recall the key concepts; see [11] for a full account. In Iris, logical judgments take the form $P \vdash Q$, where P and Q are separation logic assertions. Separation logic extends classical Hoare logic with a *separating conjunction* $P * Q$, which asserts that the resources described by P and Q are owned *disjointly*. The *magic wand* $P \mathrel{-\!\!*} Q$ is its adjoint: it asserts that, given a disjoint extension satisfying P, the combined resources satisfy Q. The canonical resource in separation logic is the *points-to assertion* $\ell \mapsto v$, which asserts exclusive ownership of heap location ℓ holding value v. The exclusiveness of the ownership is captured by this rule:

$$\ell \mapsto v * \ell \mapsto v' \vdash \mathsf{False}$$

In modern separation logics like Iris, resources are not limited to heap cells: the framework supports logical tokens and custom *ghost state* used purely for proof purposes—including the error credits and time credits central to this work. A key property of separating conjunction is that, in general, $P \not\vdash P * P$: owning a resource does not entitle you to two copies of it. The resources that can be duplicated freely are called *persistent* propositions. For example, an equality is a persistent proposition, while a points-to predicate is not. Duplicable resources are important because we can always create a copy of them to give away to other threads. The *persistently modality* $\square P$ marks P as a duplicable resource. One way to state this precisely is the rule $\square P \vdash \square P * P$.

Specifications are written as Hoare triples $\{P\}\ e\ \{v.\, Q\}$: if the precondition P holds, then every result v that e returns satisfies the postcondition Q. The frame rule

$$\frac{\vdash \{P\}\ e\ \{Q\}}{\vdash \{P * R\}\ e\ \{Q * R\}}$$

is the cornerstone of modular reasoning: it allows one to extend any proof with additional disjoint resources R that are untouched by e.

3 Probabilistic Termination in Eris$_t$

Eris [1] is a higher-order separation logic for proving error probability bounds for probabilistic programs. A key feature of Eris is *error credits*, a separation logic resource that tracks the error probability bound of the program. The ownership of ε error credits is a

first-class proposition in the separation logic, written $\lightning(\varepsilon)$ (read: "up to ε"). Error credits satisfy the following rules:

$$\lightning(\varepsilon_1) * \lightning(\varepsilon_2) \dashv\vdash \lightning(\varepsilon_1 + \varepsilon_2) \qquad \lightning(\varepsilon_1) * (\varepsilon_2 < \varepsilon_1) \vdash \lightning(\varepsilon_2)$$
$$\lightning(1) \vdash \text{False} \qquad \vdash \lightning(0)$$

Throughout this section, we use the notation $[P]\ e\ [Q]$ (square brackets) for Eris$_t$'s *total* Hoare triple, which is syntactically distinct from Eris's *partial* Hoare triple $\{P\}\ e\ \{Q\}$ (curly brackets). Eris is a *partial-correctness logic*: its adequacy theorem counts diverging executions (those that never reach a value) as vacuously safe, so a program that *always* diverges trivially satisfies any Hoare triple with any postcondition. More generally, Eris's bound only applies to the probability of *terminating* with a result satisfying ϕ; divergence probability is not charged against the error budget. Its variant Eris$_t$, on the other hand, is a *total-correctness logic*. A specification $\vdash [P * \lightning(\varepsilon)]\ e\ [\phi]$ in Eris$_t$ intuitively means: "if P holds then the probability that e *both terminates and* produces a value satisfying ϕ is at least $1-\varepsilon$". This is formally expressed with the adequacy theorem shown below.

Theorem 3.1 (Total Adequacy). *If* $\vdash [\lightning(\varepsilon)]\ e\ [\phi]$ *then* $\Pr_{\text{exec}(e)}[\phi] \geq 1-\varepsilon$.

The program logic can interact with error credits via an *expectation-preserving* composition of error credits over a randomized step.

$$\frac{\sum_{i=0}^{N} \frac{\mathcal{E}_2(i)}{N+1} = \varepsilon_1}{\vdash [\lightning(\varepsilon_1)]\ \mathsf{rand}\ N\ [n\ .\ \lightning(\mathcal{E}_2(n))]}\ \textsc{tht-rand-exp}$$

With this rule, we can prove probabilistic termination of programs via a technique called *error amplification*. As a running example, consider coinToss below (with flip $\triangleq$ rand 1), which flips a fair coin until it lands heads; since each flip succeeds independently with probability $1/2$, the program terminates with probability 1. To establish this in Eris$_t$, it suffices to prove that $[\lightning(\varepsilon)]$ coinToss () [True] for any $\varepsilon > 0$.

$$\mathsf{coinToss}\ () \triangleq \mathsf{if}\ \mathsf{flip}\ \mathsf{then}\ ()\ \mathsf{else}\ \mathsf{coinToss}()$$

The central observation is that error credits can be amplified: if we have a budget ε, we can apply THT-RAND-EXP at the coin flip. The heads branch (flip $= 1$) terminates at cost 0, and the tails branch (flip $= 0$) recurses at cost 2ε, yielding an expected cost of $(1/2)\cdot 0 + (1/2)\cdot 2\varepsilon = \varepsilon$, which matches the original budget. Consequently, provability with budget ε follows from provability with the doubled budget 2ε:

$$[\lightning(2\varepsilon)]\ \text{coinToss}\ ()\ [\text{True}] \vdash [\lightning(\varepsilon)]\ \text{coinToss}\ ()\ [\text{True}]$$

Another way to read this judgement is: if we start with ε error credits, we can amplify this budget by a factor of 2 after a step of coinToss. This closes the argument because any positive ε eventually reaches or exceeds 1 under repeated doubling, at which point $\lightning(1) \vdash \text{False}$ trivially implies the postcondition. Figure 1 shows the resulting tree of error budgets along the execution paths of coinToss.

The error amplification argument establishes $[\lightning(\varepsilon)]$ coinToss () [True] for every $\varepsilon > 0$, but the precondition still demands a positive error budget. Eris$_t$ provides an *out-of-thin-air* rule [13] to close this gap:

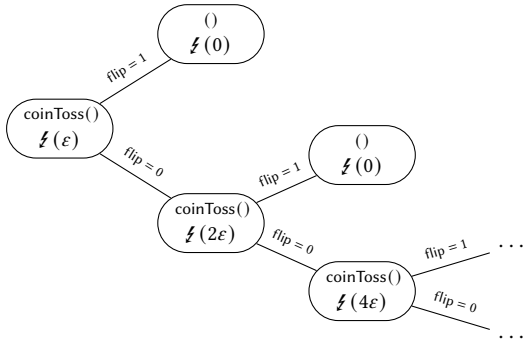

Figure 1: Error amplification for the proof of coinToss

if a specification holds for all budgets strictly greater than ε, it holds for ε itself.

$$\frac{\forall \varepsilon'.\ \varepsilon' > \varepsilon \vdash \left[\lightning(\varepsilon')\right]\ e\ [\phi]}{[\lightning(\varepsilon)]\ e\ [\phi]}\ \textsc{ht-thin-air}$$

Since $\lightning(0)$ is freely derivable, applying HT-THIN-AIR at $\varepsilon = 0$ lifts the family of proofs indexed by $\varepsilon' > 0$ to a proof with a trivial precondition, yielding [True] coinToss () [True].

This example shows that Eris$_t$'s proof rules are expressive enough to verify termination of coinToss. A natural question is whether they are expressive enough in general: can every almost-surely terminating program be proved to terminate in Eris$_t$? The main result of this work answers this affirmatively.

Theorem 3.2 (Completeness of Eris$_t$). *If* $\Pr_{\text{exec}(e)}[\phi] \geq 1-\varepsilon$ *then* $\vdash [\lightning(\varepsilon)]\ e\ [\phi]$

To explain the proof of this theorem, we first need to explain the semantic model of Hoare triples in Eris$_t$. Following standard practice in Iris [11], the Hoare triple in Eris$_t$ is defined via a *weakest precondition* predicate.

$$[P]\ e\ [Q] \triangleq \Box(P \mathrel{-\!\!*} \mathsf{twp}\ e\ \{Q\})$$

A simplified model of Eris$_t$'s weakest precondition, used in our proof, is shown below:

$$\begin{aligned}\mathsf{twp}\ e_1\ \{\Phi\} \triangleq\ & (e_1 \in \mathit{Val} \wedge \Phi(e_1)) \\ & \vee (e_1 \notin \mathit{Val} \wedge \forall \varepsilon_1.\ \lightning_\bullet(\varepsilon_1) \mathrel{-\!\!*} \\ & \quad \mathsf{GLM}(e_1, \varepsilon_1, (\lambda\ e_2, \varepsilon_2\ .\ (\lightning_\bullet(\varepsilon_2)\ * \mathsf{twp}\ e_2\ \{\Phi\}))))\end{aligned}$$

The definition proceeds by cases. If the expression e_1 is already a value, the precondition holds simply if that value satisfies the postcondition ϕ.

If e_1 is not a value, we must reason about the next step of execution. The second disjunct quantifies over an authoritative error budget $\lightning_\bullet(\varepsilon_1)$: here $\lightning_\bullet(\varepsilon)$ is the authoritative element of the resource algebra [11, 12] Auth$(\mathbb{R}_0^+, +)$ (with validity restricted to $[0,1)$) of which $\lightning(\varepsilon)$ is a fragmental view, satisfying $\lightning(\varepsilon) * \lightning_\bullet(\varepsilon') \vdash \varepsilon \leq \varepsilon'$. The *graded lifting modality* GLM then distributes this budget across successor expressions in an expectation-preserving way (as in THT-RAND-EXP), with each successor e_2 receiving a reallocated budget $\lightning_\bullet(\varepsilon_2)$ and a recursive occurrence of twp. The full definition of GLM is rather complicated — we only need the following corollary to prove our result.

$$\frac{\mathrm{red}(e_1) \qquad \exists r.\, \forall e_2.\, \mathcal{E}_2(e_2) \le r \qquad \sum_{e_2 \in Expr} \mathrm{step}(e_1)(e_2) \cdot \mathcal{E}_2(e_2) \le \varepsilon \qquad \forall e_2.\, \mathrm{step}(e_1)(e_2) > 0 \twoheadrightarrow \mathcal{E}_2(e_2) \ge 1 \vee Z(e_2, (\mathcal{E}_2(e_2)))}{\mathrm{GLM}(e_1, \varepsilon, Z)}\ \textsc{step-exp-step}$$

To approach the completeness theorem, we start from executions up to a finite number of steps.

Lemma 3.3. *If* $\Pr_{\mathrm{exec}_n(e)}[\phi] \ge 1 - \varepsilon$ *then* $\vdash [\lightning(\varepsilon)]\ e\ [\phi]$

We prove this lemma via induction on n for all e. When $n = 0$, either $e = v$ for some $v \in \phi$, or $\varepsilon \ge 1$. In both cases, the Hoare triple can be easily derived.

In the inductive step, the nontrivial case happens when e is a reducible expression (and hence not a value). In that case, given $\Pr_{\mathrm{exec}_{n+1}(e)}[\phi] \ge 1 - \varepsilon$ and $\lightning(\varepsilon) * \lightning_\bullet(\varepsilon_1)$, we can apply the rule step-exp-step, with the error random variable $\mathcal{E}_2(e') \triangleq \varepsilon_0 + 1 - \Pr_{\mathrm{exec}_n(e')}[\phi]$, where $\varepsilon_0 = \varepsilon_1 - \varepsilon \ge 0$. The third premise of the rule can be proved by observing that $\Pr_{\mathrm{exec}_{n+1}(e)}[\phi] = \sum_{e' \in Expr} \mathrm{step}(e)(e') \cdot \Pr_{\mathrm{exec}_n(e')}[\phi]$.

Finally, we can finish the fourth premise: for every e' such that $\mathrm{step}(e)(e') > 0$, we first update the resources $\lightning(\varepsilon) * \lightning_\bullet(\varepsilon_1)$ to $\lightning\left(1 - \Pr_{\mathrm{exec}_n(e')}[\phi]\right) * \lightning_\bullet\left(\varepsilon_0 + 1 - \Pr_{\mathrm{exec}_n(e')}[\phi]\right)$. The second part of the conjunction is just $\lightning_\bullet(\mathcal{E}_2(e'))$, which fulfills the authoritative error requirement. The recursive occurrence of the weakest precondition can then be solved by applying the induction hypothesis.

Now we can prove Theorem 3.2: we first apply ht-thin-air, giving us $\lightning(\varepsilon')$ for some $\varepsilon' > \varepsilon$. From this inequality, it follows that $\Pr_{\mathrm{exec}(e)}[\phi] > 1 - \varepsilon'$. By monotonicity of exec, there exists n such that $\Pr_{\mathrm{exec}_n(e)}[\phi] \ge 1 - \varepsilon'$. It follows by Lemma 3.3 that $\vdash [\lightning(\varepsilon')]\ e\ [\phi]$.

4 Expected Running Time in Tachis

Tachis [7] is a higher-order separation logic for reasoning about the *expected cost* (e.g., running time) of probabilistic programs.

A *cost model* is a function $cost\colon Expr \to \mathbb{R}_{\ge 0}$ that assigns a nonnegative cost to one step of evaluating an expression, invariant under evaluation contexts. For example, $cost_{\mathsf{all}} \triangleq \lambda\_.\, 1$ charges one unit per reduction step, recovering the standard notion of running time. The *stratified expected cost* is defined by induction on n, mirroring the construction of exec_n in §2:

$$\mathrm{EC}_n(e) \triangleq \begin{cases} 0 & \text{if } n = 0 \text{ or } e \in Val, \\ cost(e) + \sum_{e'} \mathrm{step}(e)(e') \cdot \mathrm{EC}_{n-1}(e') & \text{otherwise.} \end{cases}$$

The full *expected cost* is $\mathrm{EC}(e) \triangleq \sup_n \mathrm{EC}_n(e)$, taking the value ∞ if the sequence is unbounded.

Tachis introduces *time credits* as a separation logic resource: ownership of $x \in \mathbb{R}_{\ge 0}$ credits is written $\$(x)$. Time credits can be split and merged freely ($\$(x_1) * \$(x_2) \dashv\vdash \$(x_1 + x_2)$), and each reduction step consumes $\$(cost(e))$ from the precondition. For a randomized step, the credits are distributed across outcomes in an expectation-preserving way, analogous to tht-rand-exp:

$$\frac{cost(\mathsf{rand}\ N) + \sum_{n=0}^{N} \frac{X_2(n)}{N+1} \le x_1}{\vdash \{\$(x_1)\}\ \mathsf{rand}\ N\ \{n\ .\ \$(X_2(n))\}}\ \textsc{ht-rand-exp}$$

The premise requires that the cost of the step plus the *expected* remaining credits does not exceed the initial budget x_1.

The adequacy theorem confirms that a derivable specification yields a valid upper bound on the expected cost:

Theorem 4.1 (Tachis Adequacy). *If* $\vdash \{\$(x)\}\ e\ \{\mathsf{True}\}$, *then* $\mathrm{EC}(e) \le x$.

Similar to Eris, the result we present is the converse of the adequacy:

Theorem 4.2 (Completeness of Tachis). *If* $\mathrm{EC}(e) < x$, *then* $\vdash \{\$(x)\}\ e\ \{\mathsf{True}\}$.

The proof follows the same strategy as the Eris_t completeness theorem (Theorem 3.2): the Tachis weakest precondition is again a least fixpoint, and the argument proceeds by induction on n using the finite-horizon approximation $\mathrm{EC}_n(e)$.

5 Conclusion and Related Work

We have proved the semantic completeness of Eris_t and Tachis, both restricted to pure programs. We have also proved a completeness result for Approxis [6]. There, we show that if an *approximate coupling* exists between two programs, then a relational weakest precondition can be derived. All of the aforementioned results are mechanized in the Clutch repository with the Rocq proof assistant [18].

A key limitation of these results is the restriction to pure programs. The root cause is not probabilistic: it stems from the deterministic heap allocation in the operational semantics of the language. In a recent work [9], we address this by extending the completeness results to programs with mutable state but without allocation, and we show that completeness can be established using only the proof rules of the logic, without unfolding the semantic model of the weakest precondition. Independently of that paper, we have also used the technique to derive completeness results for Coneris [13], the concurrent version of Eris.

Related Work. For classical Hoare logic, relative completeness was established by Cook [4], who showed that if the assertion language is expressive enough to define the strongest postcondition, then any semantically valid triple is provable. Our setting differs substantially: Iris is a higher-order logic with no separation between assertions and specifications, and the programs we reason about can encode arbitrary recursion.

For sequential separation logic, Ishtiaq and O'Hearn [10] and Yang and O'Hearn [19] established completeness by showing that the weakest precondition is expressible in the logic. Haslbeck and Nipkow [8] proved relative completeness for a logic with time credits, in a language without dynamic allocation—the same restriction we face here.

For probabilistic programs, Majumdar and Sathiyanarayana [14] gave sound and relatively complete proof rules for almost-sure termination in a first-order imperative language. Our results are

complementary: we target higher-order probabilistic programs inside a separation logic framework.

Our recent work [9] takes a broader view, providing a general methodology for completeness proofs of Iris-based logics. It subsumes and strengthens the results presented here for Eris, and additionally covers the default Iris program logic (both partial and total), λ_{Rust}, and a relational logic for refinement.

References

[1] Alejandro Aguirre, Philipp G. Haselwarter, Markus de Medeiros, Kwing Hei Li, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2024. Error Credits: Resourceful Reasoning about Error Bounds for Higher-Order Probabilistic Programs. *Proc. ACM Program. Lang.* 8, ICFP, Article 246 (Aug. 2024), 33 pages. doi:10.1145/3674635

[2] Gilles Barthe, Justin Hsu, and Kevin Liao. 2019. A probabilistic separation logic. *Proc. ACM Program. Lang.* 4, POPL, Article 55 (Dec. 2019), 30 pages. doi:10.1145/3371123

[3] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Thomas Noll. 2019. Quantitative separation logic: a logic for reasoning about probabilistic pointer programs. *Proc. ACM Program. Lang.* 3, POPL, Article 34 (Jan. 2019), 29 pages. doi:10.1145/3290347

[4] Stephen A. Cook. 1978. Soundness and Completeness of an Axiom System for Program Verification. *SIAM J. Comput.* 7, 1 (1978), 70–90. doi:10.1137/0207005

[5] Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2024. Asynchronous Probabilistic Couplings in Higher-Order Separation Logic. *Proc. ACM Program. Lang.* 8, POPL, Article 26 (Jan. 2024), 32 pages. doi:10.1145/3632868

[6] Philipp G. Haselwarter, Kwing Hei Li, Alejandro Aguirre, Simon Oddershede Gregersen, Joseph Tassarotti, and Lars Birkedal. 2025. Approximate Relational Reasoning for Higher-Order Probabilistic Programs. *Proc. ACM Program. Lang.* 9, POPL, Article 41 (Jan. 2025), 31 pages. doi:10.1145/3704877

[7] Philipp G. Haselwarter, Kwing Hei Li, Markus de Medeiros, Simon Oddershede Gregersen, Alejandro Aguirre, Joseph Tassarotti, and Lars Birkedal. 2024. Tachis: Higher-Order Separation Logic with Credits for Expected Costs. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 313 (Oct. 2024), 30 pages. doi:10.1145/3689753

[8] Maximilian P. L. Haslbeck and Tobias Nipkow. 2018. Hoare Logics for Time Bounds — A Study in Meta Theory. In *Proceedings of TACAS 2018 (LNCS, Vol. 10805)*. 155–171. doi:10.1007/978-3-319-89960-2_9

[9] Johannes Hostert, Zichen Zhang, Puming Liu, Simon Oddershede Gregersen, Ralf Jung, and Joseph Tassarotti. 2026. Completeness of Iris-Based Program Logics. (2026). Preprint.

[10] Samin S. Ishtiaq and Peter W. O'Hearn. 2001. BI as an Assertion Language for Mutable Data Structures. In *Proceedings of POPL 2001*. 14–26. doi:10.1145/360204.375719

[11] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.* 28 (2018), e20. doi:10.1017/S0956796818000151

[12] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. 637–650. doi:10.1145/2676726.2676980

[13] Kwing Hei Li, Alejandro Aguirre, Simon Oddershede Gregersen, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2025. Modular Reasoning about Error Bounds for Concurrent Probabilistic Programs. *Proc. ACM Program. Lang.* 9, ICFP, Article 245 (Aug. 2025), 30 pages. doi:10.1145/3747514

[14] Rupak Majumdar and V.R. Sathiyanarayana. 2025. Sound and Complete Proof Rules for Probabilistic Termination. *Proc. ACM Program. Lang.* 9, POPL, Article 63 (Jan. 2025), 32 pages. doi:10.1145/3704899

[15] Peter O'Hearn. 2019. Separation logic. *Commun. ACM* 62, 2 (Jan. 2019), 86–95. doi:10.1145/3211968

[16] John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS '02)*. IEEE Computer Society, USA, 55–74.

[17] Joseph Tassarotti and Robert Harper. 2019. A separation logic for concurrent randomized programs. *Proc. ACM Program. Lang.* 3, POPL, Article 64 (Jan. 2019), 30 pages. doi:10.1145/3290377

[18] The Rocq Development Team. 2024. *The Rocq Prover*. doi:10.5281/zenodo.11551307

[19] Hongseok Yang and Peter W. O'Hearn. 2002. A Semantic Basis for Local Reasoning. In *Proceedings of FoSSaCS 2002*. 402–416.