Research Statement: Formal Foundations for Programs with Outcomes

Noam Zilberstein, Cornell University

The reliance on complex software in security and privacy sensitive domains has made techniques to ensure the correctness of that software critical. Formal methods provide tools for expressing and proving correctness properties, many of which center on *program logics*. Countless success stories have shown how formal methods eliminate bugs in production software [10, 30, 38, 29, 43, 21], including my prior work as a Staff Software Engineer at Facebook [Haskell '20, CPP '22].

But the emergence of new paradigms means that there are now many disjoint logical foundations for program analysis. For example, Meta engineers found that the Infer static analyzer excelled at bug detection over correctness verification [21], requiring a new theoretical foundation tailored for reasoning about incorrectness [40]. Orthogonally, programs behave unpredictably due to *effects* such as nondeterminism, randomization, nontermination, memory access, and concurrency, which also make testing difficult. As a result, many new logics have been developed for specialized reasoning about those effects.

The core idea behind my research is that a wide variety of analysis techniques can be captured by directly reasoning about the program's *outcomes*: the behaviors arising from effects such as nondeterminism, randomness, or concurrency. I have developed a new foundational approach called *Outcome Logic*, which describes programs in terms of their full spectrum of possible behaviors and surpasses the expressiveness of traditional approaches where only a single behavior can be described.

> **Key Insight:** *Outcome Logic* provides a unifying perspective for reasoning about a wide variety of programs and properties about those programs.

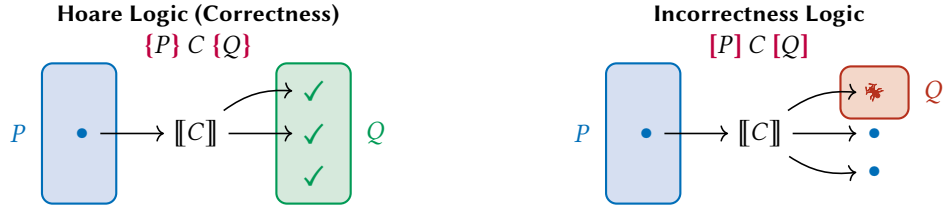
Two of the practical benefits of Outcome Logic are described below:

1. **Unifying Correctness and Incorrectness**: Outcome Logic bridges reasoning about the *absence* of errors (correctness) and the *presence* of bugs (incorrectness). Whereas prior techniques were specialized to one modality, Outcome Logic enables new forms of static analysis by incorporating elements from both styles.
2. **Logics for Randomized Concurrency**: By composing multiple types of effects, I have developed an expressive foundation for the analysis of randomized distributed algorithms, which goes beyond prior approaches in its ability to reason about probabilistic behavior. Applications include verification of security, privacy, and blockchain algorithms.

Stemming from those theoretical advancements, my research has formed the basis for over one million dollars of funding from Amazon, ARIA, and the NSF, and earned me the 2024 ACM SIGPLAN John Vlissides Award [SPLASH '24]. My research has positioned me as an expert on program logics, further evidenced by the fact that I have been asked to provide expert reviews for over a dozen papers at top PL conferences and journals, and I have organized an annual static analysis workshop at POPL for the last three years. My foundational formal methods expertise paired with my previous applied formal methods experience in industry uniquely position me to develop novel, but also practical, techniques. In the remainder of this statement, I will overview my prior work on Outcome Logic and then discuss my ongoing research vision.

INITIAL RESULTS: OUTCOME LOGIC

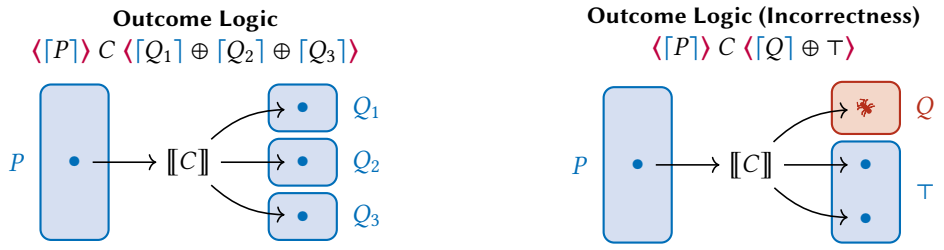
Correctness and Incorrectness. Traditional correctness analysis, based on Hoare Logic triples $\{P\}\ C\ \{Q\}$, describes the behavior of a program C in terms of a precondition P and a postcondition Q: if the program C is run in a state satisfying P, then it will end in a state satisfying Q. Thus, Q *over-approximates* the reachable states, as seen in the diagram below, making Hoare Logic suitable for correctness. As long as Q only describes *safe* behaviors, then the program will never go wrong. However, as shown by O'Hearn, Hoare Logic cannot identify true positive bugs, As an alternative, Incorrectness Logic (IL) triples $[P]\ C\ [Q]$ mean that any state satisfying Q is *reachable* from a state satisfying P [40]. So, in IL, Q *under-approximates* the reachable states, meaning that IL can describe bugs that sometimes—*but not always*—arise.



More concretely, consider the C program $x \coloneqq \texttt{malloc}() \fatsemi *x \coloneqq 1$, which allocates a pointer, but does not perform a null-check before dereferencing it. In static analysis, **malloc** is often treated as nondeterministic: it either returns a valid pointer

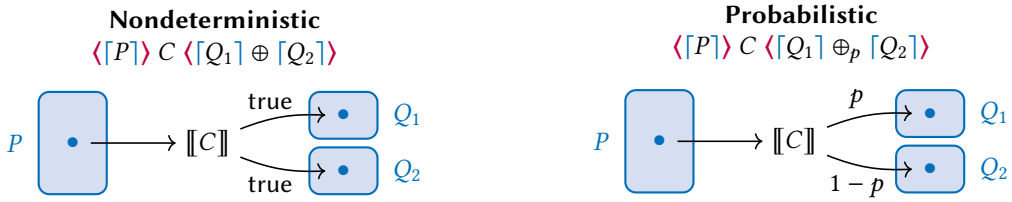
or null. The program above has two outcomes, one of which results in the program crashing. Because Hoare Logic is over-approximate, any specification for the program must encompass all of the outcomes, meaning that the crash cannot be singled out as a reachable outcome. Incorrectness Logic can guarantee that the crash is a true positive, but IL is incompatible with techniques such as abstract interpretation [5, 6] and struggles to describe the cause of a bug [32, OOPSLA '24a, 7].

I first developed Outcome Logic as a means to *unify correctness and incorrectness reasoning* by guaranteeing true positives using a theory that is closer to Hoare Logic [OOPSLA '23]. Similar to Hoare Logic, Outcome Logic uses $\lceil P \rceil$ to indicate that P over-approximates the reachable states, but OL also provides a new logical connective $\varphi \oplus \psi$—the *outcome conjunction*—to state that multiple outcomes are non-vacuously reachable. In the context of incorrectness, we can specify a bug using the postcondition $\lceil Q \rceil \oplus \top$ where Q describes the error state and $\top$ vacuously covers the remaining outcomes.



Following the initial success of Outcome Logic, Amazon funded further development of automated reasoning for correctness and incorrectness through an Amazon Research Award[1]. Together with an undergraduate mentee, I developed joint techniques for verification and bug-finding. In doing so, we showed that our approach accurately models the bug finding analyses built at Meta [OOPSLA '24a], which was also corroborated by Raad et al. [42], some of the original developers of Incorrectness Logic. Further tools developed at Meta [7] and Bloomberg [42] have subsequently moved towards the OL style of incorrectness, as IL is unable to capture properties such as nontermination.

Generalizing to Additional Effects. Remarkably, the Outcome Logic metatheory generalizes cleanly to programs with many different kinds of branching by assigning *weights* to each outcome. In the case of nondeterministic programs, those weights are Booleans, indicating whether or not each end state is possible. In a probabilistic program, weights are real numbers indicating the likelihood of each outcome.



More instantiations represent outcomes as multisets [27, 31] and indexed valuations [45] (two semantic domains for mixing nondeterminism and randomization), path conditions in symbolic execution, and more. The result is a single sound and relatively complete logic for reasoning about a wide variety of programs [TOPLAS '25].

I have also investigated how outcomes interact with other kinds of effects including exceptions [OOPSLA '23] and mutable state [OOPSLA '24a]. Together with a Masters student, I added abilities to reason about nontermination in Outcome Logic, and we additionally showed that Outcome Logic subsumes multiple taxonomies of nondeterminism logics [TPSA '25]. Finally, with collaborators at Cornell and NYU, I combined two kinds of outcomes—nondeterministic and probabilistic—to produce Demonic Outcome Logic [POPL '25]. Based on that body of work [SPLASH '24], I was awarded the 2024 ACM SIGPLAN John Vlissides award for Outcome Logic's *"strong potential for practical impact."*[2]

Probabilistic Concurrency. For decades, randomization has played a central role in distributed computing, offering elegant and practical solutions to problems that are difficult or impossible to solve deterministically. Applications include distributed cryptography and blockchain [1, 23, 36, 2], database systems [18, 25], and beyond. But despite their prevalence, randomized distributed algorithms are notoriously difficult to reason about, as unexpected behavior arises from the interaction between random sampling and concurrent execution. As Lehmann and Rabin remarked when developing a proof of the randomized *Dining Philosophers*:

[1] https://www.amazon.science/research-awards/recipients/alexandra-silva
[2] https://www.sigplan.org/Awards/Other/#2024_Noam_Zilberstein__Cornell_University

> "*Proofs of correctness for probabilistic distributed systems are extremely slippery; in fact the proof presented here (hopefully correct) is only the last one in a sequence of incorrect proofs.*" [33]

My research in formal methods for probabilistic concurrency offers a solution, guaranteeing that we do not further extend the "*sequence of incorrect proofs.*" While formal verification is broadly applicable, the cost is often a barrier, and therefore it is most effectively deployed on critical components of a system where bugs are catastrophic, and testing is difficult or impossible. In my prior industry role at Facebook, I participated in such a verification effort that uncovered bugs and an optimization in the underlying data structure of a microkernel [CPP '22], demonstrating that formal methods lead to real world impact when applied appropriately. Those bugs involved non-blocking concurrency and manifested only under a rare thread interleaving, making them unlikely to be caught through testing alone. Randomization introduces another layer of complexity; it is difficult to ensure that tests exercise all the random behaviors, and tests cannot ensure that the results are distributed correctly over many runs, or that the program avoids deadlock in the limit. Those complications, along with the fact that randomization is critical to security, privacy, and blockchain applications, make randomized distributed systems a prime target for formal verification. As further evidence of the importance of this work, the development is supported by a $900,000 NSF Medium grant (awards #2504142 and #2504143).

Whereas many approaches to probabilistic concurrency verification start with concurrency and add randomness [44, 35], we have taken the opposite approach of adding concurrency to Demonic Outcome Logic [POPL '25], which has rich capabilities for probabilistic reasoning combined with the ability to reason about the nondeterminism of the scheduler. However, concurrency adds an extraordinary amount of technical complexity. We needed to start from first principles to develop a new semantic model for probabilistic concurrency, as existing ones could not adequately capture the interaction between scheduling, random sampling, and while loops [CONCUR '25]. Building on that semantic model, we developed Probabilistic Concurrent Outcome Logic (pcOL), which is the first program logic for reasoning about randomized concurrent programs in terms of the full distributions of outcomes that they produce [POPL '26].

By incorporating elements from Concurrent Separation Logic (CSL) [39, 14] and Probabilistic Separation Logics (PSLs) [12, 34, 11] into Outcome Logic, pcOL introduces expressive new compositional reasoning techniques. In a concurrent program, nondeterminism occurs at every step of computation, as the scheduler nondeterministically chooses how to interleave the threads of computation. Reasoning about all of the possible interleavings is intractable, so compositional techniques are needed to tame the space of outcomes. Consider the following two OL specifications, stating that after flipping a coin and storing the result in x (resp. y), the value of x (resp. y) is distributed like a fair coin flip.

$$\langle x \mapsto - \rangle\ x \coloneqq \mathtt{flip}\left(\tfrac{1}{2}\right)\ \langle x \sim \mathtt{flip}\left(\tfrac{1}{2}\right) \rangle \qquad\qquad \langle y \mapsto - \rangle\ y \coloneqq \mathtt{flip}\left(\tfrac{1}{2}\right)\ \langle y \sim \mathtt{flip}\left(\tfrac{1}{2}\right) \rangle$$

Now, if we run these threads in parallel, we would ideally not only like to know how x and y are distributed, but also their *joint* distribution. The pcOL Par inference rule below, inspired by CSL, allows us to analyze two threads compositionally as long as their memory footprints are disjoint, which is signified by the *separating conjunction* $\varphi * \psi$.

$$\frac{\langle x \mapsto - \rangle\ x \coloneqq \mathtt{flip}\left(\tfrac{1}{2}\right)\ \langle x \sim \mathtt{flip}\left(\tfrac{1}{2}\right) \rangle \qquad \langle y \mapsto - \rangle\ y \coloneqq \mathtt{flip}\left(\tfrac{1}{2}\right)\ \langle y \sim \mathtt{flip}\left(\tfrac{1}{2}\right) \rangle}{\langle (x \mapsto -) * (y \mapsto -) \rangle\ x \coloneqq \mathtt{flip}\left(\tfrac{1}{2}\right) \parallel y \coloneqq \mathtt{flip}\left(\tfrac{1}{2}\right)\ \langle \left(x \sim \mathtt{flip}\left(\tfrac{1}{2}\right)\right) * \left(y \sim \mathtt{flip}\left(\tfrac{1}{2}\right)\right) \rangle}\ \textsc{Par}$$

In PSL, $\varphi * \psi$ additionally guarantees that the resources of φ and ψ are *probabilistically independent*, so we can conclude that $x = y = 1$ with probability $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$, hence we obtain the entire joint distribution over all the variables. While putting CSL and PSL together may seem like a sensible idea, it required a brand-new logical foundation to ensure soundness, making pcOL much more than the sum of its parts. Incorporating Outcome Logic additionally allows us to do case analysis over randomized shared state. We have demonstrated pcOL on a variety of case studies, including concurrent shuffling algorithms [9], private information retrieval [16], and randomness-conserving protocols [46].

RESEARCH VISION: OUTCOME LOGIC AS A GENERALIZED FOUNDATION

Through my prior research, I have repeatedly shown that outcomes provide a key abstraction for a large variety of verification tasks. As I move into a faculty role, I plan to build a research group with a diverse skillset to simultaneously advance the Outcome Logic theory, and to apply that theory to new and more complex domains.

> **Unifying Research Vision:** Make it easier to bootstrap verification of new types of programs by providing an extensible foundation for formal methods.

For the next several years, I plan to work towards that vision across three axes: mechanizing Outcome Logic in Lean, deepening Outcome Logic's applicability to distributed probabilistic algorithms, and applying Outcome Logic to automated correctness and incorrectness static analysis. The axes are comprised of many projects, suitable for students whose research interests range from theoretical to applied.

Axis I: Lean Mechanization and Real-World Verification

The Lean Proof Assistant [37] is quickly gaining momentum among mathematicians, computer scientists, and companies such as Amazon and Google DeepMind. I plan to mechanize Outcome Logic in Lean in order to formally verify the underlying metatheory, boost its extensibility, and scale it to larger applications. Lean is the ideal choice for Outcome Logic, as a large amount of the underlying probability theory, domain theory, and topology have already been formalized in mathlib [17].

Outcome Logic in cslib. Following the success of mathlib, Amazon and Google DeepMind have backed the Lean community to formalize undergraduate-level computer science in a new Lean development called cslib[3]. Program logics will feature in cslib, as they are an important part of programming language theory and formal methods. Given Outcome Logic's expressiveness [TOPLAS '25], a single Lean mechanization of OL could subsume many otherwise disjoint implementations for nondeterministic programs, probabilistic programs, incorrectness, and others. I will therefore advocate for Outcome Logic to become a core component of cslib, which will reduce the overall implementation load.

The initial OL mechanization will make it easier for researchers to extend verification to new domains. For example, verification of quantum computation requires reasoning about the probabilistic outcomes of quantum phenomena, so I believe that it can build on the core OL theory.

Scaling Outcome Logic Verification. Building on my experience formally verifying the inter-process communication of an operating system at Facebook [CPP '22], I plan to use the Lean mechanization of Outcome Logic to verify more real-world software. Although formal verification is extremely time consuming, at Facebook, we achieved success by focusing our effort on a crucial component: correctness of the entire system hinged on a non-blocking concurrent queue data structure. By applying formal methods to that queue, we found a bug and an optimization in the implementation, which uncovered more bugs in client code.

The key lesson was that formal methods have the largest return on investment when applied in a targeted way to highly intricate components of a system. That is why I plan to focus on distributed randomized algorithms, which are used in critical applications such as cryptography, privacy, and consensus protocols, and which are extremely hard to understand and test. To begin, I will verify increasingly complex case studies, which would be impossible using pen-and-paper. For example, I will verify randomized consensus protocols such as Ben-Or's algorithm [13], which will stretch the capabilities of the logic in that it requires a fair scheduler, and because the probability of avoiding deadlock only converges to 1 after an infinite amount of time. Once the mechanization is ironed out, I plan to move to verification of open source software, following the same formula that I used at Facebook.

Axis II: Verification of Distributed Probabilistic Algorithms

I have a long-term vision to position Probabilistic Concurrent Outcome Logic (pcOL) as an extensible platform for verification of probabilistic programs, similar to how Iris [28] is positioned for higher order concurrent programs. Part of this vision has already been funded by an NSF medium award.

The combination of reasoning about concurrency and also the full distribution of program outcomes makes pcOL a unique foundational theory, but substantial developments are needed before it can be applied to industrial verification tasks. It took Iris 10 years to reach its current state of maturity; building on those lessons, I believe that we can develop pcOL more quickly, but this research direction nonetheless represents years of work. The following milestones will bring pcOL to the point where it can be used to verify real world algorithms.

Abstractions for Fine-Grained Concurrency. State of the art concurrency logics such as Iris [28] and VST [3] include advanced additional features to guarantee that threads interact with shared state in well-behaved ways. Examples include:

- **Resource Algebras and Ghost State**. Users must be able to define custom resource structures to express logical ownership over shared state. For example, a certain memory configuration may indicate that a particular thread owns a lock, and therefore no other threads can modify the resources guarded by that lock.
- **Linearizability and Logical Atomicity**. Linearizability ensures that each operation on a concurrent object appears to occur instantaneously, while logical atomicity further allows complex operations to behave as if they were atomic. For example, a logically atomic read-modify-write on a lock-free counter behaves like a single atomic update, even though it may retry internally.
- **Protocols and Updates**. When threads interact with shared state, there are often rules governing how each thread is allowed to update that state. For example, a counter's value increases monotonically over time.

[3] https://docs.google.com/presentation/d/1aJFM3EaI4LcppHR_2YFQHiBjUfMMhMKxCeM3BfINi48

These features are necessary for reasoning about fine-grained concurrency; however, many questions remain about how they interact with probabilistic computation. It will take a substantial amount of theoretical work to fit the two models together.

New Threat Models. In PCOL, we assume that the scheduler is controlled by a very strong adversary, which is free to pick thread interleavings in any way it chooses. While this provides robust guarantees, it is too strong for many verification tasks, which are only correct subject to weaker adversaries [24, 13, 33, 8, 22]. Two examples are:

- **Fair Schedulers**. A fair scheduler cannot starve any thread for an infinite amount of time. Fairness is a common assumption and often arises, *e.g.,* when one thread busy-waits on a resource. An unfair adversary can choose to only schedule the waiting thread, causing the program to never terminate, but we would not consider that program incorrect.
- **Oblivious Schedulers**. A scheduler is oblivious if it cannot see the outcome of random coin flips. Many security applications require oblivious schedulers; we must assume that the adversary cannot see randomly sampled keys.

Reasoning in weaker threat models introduces significant technical challenges [4]. While pen-and-paper proof techniques exist [24], they are not compositional in that they involve reasoning globally about all the threads and do not generalize into reusable rules. I plan to develop thread-local reasoning techniques, which will be applicable to large classes of programs.

Axis III: Correctness and Incorrectness

In my prior work, I have explored Outcome Logic's theoretical applicability to incorrectness [OOPSLA '23, OOPSLA '24a, OOPSLA '24b, TOPLAS '25], which provides benefits such as easily identifying the causes of bugs and the ability to capture additional kinds of incorrectness such as nontermination bugs. Going forward, I will develop real static analysis tools and investigate novel ways to improve their performance and efficacy, leveraging the OL theory.

Abstract Interpretation for Bug Finding. Incorrectness Logic (IL) is not compatible with abstract interpretation [5, 6], and therefore Meta's Infer static analyzer, which is based on IL, relies on alternative techniques [41, 32]. Practically speaking, the following issues arise:

- **Efficiency**: Abstract interpretation [19] has traditionally been the key to scalable static analysis [20], providing a sound way to approximate program behavior. The efficiency of Infer's bug finding algorithm can thus likely be improved using abstract interpretation and shape analysis.
- **Soundness**: Infer currently handles unknown functions in an unsound way, leading to false positives [32]. Outcome Logic can fix this problem, as its underlying theory is able to capture uncertainty of the program state.
- **Analysis of Loops**: Incorrectness Logic gives no way to approximate the behavior of loops, so Infer simply unrolls loops for a fixed number of iterations [41]. This means that bugs will be missed, whereas in Outcome Logic, standard techniques based on loop invariants and termination checking can be used to properly approximate the loop's behavior.
- **Manifest Errors** are bugs that occur regardless of how a procedure is invoked and are particularly important as they have the highest fix rates [32]. In Incorrectness Logic, manifest errors cannot be identified easily, so a secondary filtering step is used. In Outcome Logic, manifest errors can be identified trivially from the precondition.

Outcome Logic has the ability to address the issues above. I plan to develop an Outcome Logic analysis inside of Infer and evaluate it against existing tools.

Unified Static Analysis Algorithms. The previous project will bring incorrectness analysis closer to correctness analysis. The next step is to develop a single analysis that is able to perform both functions, using heuristics to move between correctness and incorrectness modes. One of the costliest steps in static analysis is to precompute specifications for all intermediary procedures [15]. In the setting where correctness is based on Hoare Logic and bug-finding is based on Incorrectness Logic, multiple summaries are needed for each procedure. Take the **malloc** procedure for instance, we would need three specifications: one for correctness and two for the reachability of each incorrectness outcome.

$$\{\mathsf{true}\}\ x := \mathbf{malloc}()\ \{(x \mapsto -) \vee (x = \mathbf{null})\} \qquad [\mathsf{true}]\ x := \mathbf{malloc}()\ [x \mapsto -] \qquad [\mathsf{true}]\ x := \mathbf{malloc}()\ [x = \mathbf{null}]$$

All three of those could be replaced by a single outcome logic specification, which both guarantees reachability of the two outcomes while also over-approximating the overall program behavior, thereby making it usable for both correctness and incorrectness applications.

$$\langle\lceil\mathsf{true}\rceil\rangle\ x := \mathbf{malloc}()\ \langle\lceil x \mapsto -\rceil \oplus \lceil x = \mathbf{null}\rceil\rangle$$

I conjecture that this reduction in intermediary specifications will again result in a performance improvement, and plan to implement and evaluate such an algorithm against IL-based tools.

PAST WORK

[POPL '26] **Noam Zilberstein**, Alexandra Silva, and Joseph Tassarotti. "Probabilistic Concurrent Reasoning in Outcome Logic: Independence, Conditioning, and Invariants". In: *Proc. ACM Program. Lang.* 10.POPL [Jan. 2026]. To Appear.

[TPSA '25] James Li, **Noam Zilberstein**, and Alexandra Silva. Total Outcome Logic: Termination and Nontermination Proving for Effectful Branching. Presented at the Workshop on Theory and Practice of Static Analysis. Colocated with POPL '25, Denver, CO. Jan. 2025. URL: https://popl25.sigplan.org/details/tpsa-2025-papers/10/Total-Outcome-Logic-Termination-and-Nontermination-Proving-for-Effectful-Branching.

[TOPLAS '25] **Noam Zilberstein**. "Outcome Logic: A Unified Approach to the Metatheory of Program Logics with Branching Effects". In: *ACM Trans. Program. Lang. Syst.* 47.3 [Sept. 2025]. ISSN: 0164-0925. DOI: 10.1145/3743131.

[CONCUR '25] **Noam Zilberstein**, Daniele Gorla, and Alexandra Silva. "Denotational Semantics for Probabilistic and Concurrent Programs". In: *36th International Conference on Concurrency Theory (CONCUR 2025)*. Vol. 348. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum f"ur Informatik, 2025, 39:1–39:24. DOI: 10.4230/LIPIcs.CONCUR.2025.39.

[POPL '25] **Noam Zilberstein**, Dexter Kozen, Alexandra Silva, and Joseph Tassarotti. "A Demonic Outcome Logic for Randomized Nondeterminism". In: *Proc. ACM Program. Lang.* 9.POPL [Jan. 2025]. DOI: 10.1145/3704855.

[OOPSLA '24b] Linpeng Zhang, **Noam Zilberstein**, Benjamin Lucien Kaminski, and Alexandra Silva. "Quantitative Weakest Hyper Pre: Unifying Correctness and Incorrectness Hyperproperties via Predicate Transformers". In: *Proc. ACM Program. Lang.* 8.OOPSLA2 [Oct. 2024]. DOI: 10.1145/3689740.

[SPLASH '24] **Noam Zilberstein**. "Unified Analysis Techniques for Programs with Outcomes". In: *Companion Proceedings of the 2024 ACM SIGPLAN SPLASH Conference*. Pasadena, CA, USA: Association for Computing Machinery, 2024, pp. 4–6. ISBN: 9798400712142. DOI: 10.1145/3689491.3691814.

[OOPSLA '24a] **Noam Zilberstein**, Angelina Saliling, and Alexandra Silva. "Outcome Separation Logic: Local Reasoning for Correctness and Incorrectness with Computational Effects". In: *Proc. ACM Program. Lang.* 8.OOPSLA1 [Apr. 2024]. DOI: 10.1145/3649821.

[OOPSLA '23] **Noam Zilberstein**, Derek Dreyer, and Alexandra Silva. "Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning". In: *Proc. ACM Program. Lang.* 7.OOPSLA1 [Apr. 2023]. DOI: 10.1145/3586045.

[CPP '22] Quentin Carbonneaux, **Noam Zilberstein**, Christoph Klee, Peter W. O'Hearn, and Francesco Zappa Nardelli. "Applying Formal Verification to Microkernel IPC at Meta". In: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2022. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 116–129. ISBN: 9781450391825. DOI: 10.1145/3497775.3503681.

[Haskell '20] **Noam Zilberstein**. "Eliminating Bugs with Dependent Haskell (Experience Report)". In: *Proceedings of the 13th ACM SIGPLAN International Symposium on Haskell*. Haskell 2020. Virtual Event, USA: Association for Computing Machinery, 2020, p. 9. ISBN: 9781450380508. DOI: 10.1145/3406088.3409020.

FURTHER REFERENCES

[1] Orestis Alpos and Christian Cachin. "Do Not Trust in Numbers: Practical Distributed Cryptography with General Trust". In: *Stabilization, Safety, and Security of Distributed Systems*. Cham: Springer Nature Switzerland, 2023, pp. 536–551. ISBN: 978-3-031-44274-2.

[2] Ignacio Amores-Sesar, Christian Cachin, and Philipp Schneider. "An Analysis of Avalanche Consensus". In: *Structural Information and Communication Complexity*. Ed. by Yuval Emek. Cham: Springer Nature Switzerland, 2024, pp. 27–44. ISBN: 978-3-031-60603-8. DOI: 10.1007/978-3-031-60603-8_2.

[3] Andrew W. Appel. "Verified software toolchain". In: *Proceedings of the 20th European Conference on Programming Languages and Systems: Part of the Joint European Conferences on Theory and Practice of Software*. ESOP'11/ETAPS'11. Saarbrücken, Germany: Springer-Verlag, 2011, pp. 1–17. ISBN: 9783642197178.

[4] Krzysztof Apt and Gordon Plotkin. "Countable nondeterminism and random assignment". In: *J. ACM* 33.4 [Aug. 1986], pp. 724–767. ISSN: 0004-5411. DOI: 10.1145/6490.6494.

[5] Flavio Ascari, Roberto Bruni, and Roberta Gori. "Limits and difficulties in the design of under-approximation abstract domains". In: *Foundations of Software Science and Computation Structures*. Cham: Springer International Publishing, 2022, pp. 21–39. ISBN: 978-3-030-99253-8. DOI: 10.1007/978-3-030-99253-8_2.

[6] Flavio Ascari, Roberto Bruni, and Roberta Gori. "Limits and Difficulties in the Design of Under-Approximation Abstract Domains". In: *ACM Trans. Program. Lang. Syst.* 46.3 [Oct. 2024]. ISSN: 0164-0925. DOI: 10.1145/3666014.

[7] Flavio Ascari, Roberto Bruni, Roberta Gori, and Francesco Logozzo. "Revealing Sources of (Memory) Errors via Backward Analysis". In: *Proc. ACM Program. Lang.* 9.OOPSLA1 [Apr. 2025]. DOI: 10.1145/3720486.

[8] Yonatan Aumann and Michael A. Bender. "Efficient low-contention asynchronous consensus with the value-oblivious adversary scheduler". In: *Distributed Computing* 17.3 [Mar. 2005], pp. 191–207. ISSN: 1432-0452. DOI: 10.1007/s00446-004-0113-4.

[9] Axel Bacher, Olivier Bodini, Alexandros Hollender, and Jérémie Lumbroso. MergeShuffle: A Very Fast, Parallel Random Permutation Algorithm. 2015. arXiv: 1508.03167 [cs.DS]. URL: https://arxiv.org/abs/1508.03167.

[10] Thomas Ball and Sriram K. Rajamani. "The SLAM project: debugging system software via static analysis". In: *SIGPLAN Not.* 37.1 [Jan. 2002], pp. 1–3. ISSN: 0362-1340. DOI: 10.1145/565816.503274.

[11] Jialu Bao, Emanuele D'Osualdo, and Azadeh Farzan. "Bluebell: An Alliance of Relational Lifting and Independence for Probabilistic Reasoning". In: *Proc. ACM Program. Lang.* 9.POPL [Jan. 2025]. DOI: 10.1145/3704894.

[12] Gilles Barthe, Justin Hsu, and Kevin Liao. "A Probabilistic Separation Logic". In: *Proc. ACM Program. Lang.* 4.POPL [Jan. 2020]. DOI: 10.1145/3371123.

[13] Michael Ben-Or. "Another advantage of free choice: Completely asynchronous agreement protocols". In: *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*. PODC '83. Montreal, Quebec, Canada: Association for Computing Machinery, 1983, pp. 27–30. ISBN: 0897911105. DOI: 10.1145/800221.806707.

[14] Stephen Brookes. "A Semantics for Concurrent Separation Logic". In: *CONCUR 2004 - Concurrency Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–34. ISBN: 978-3-540-28644-8. DOI: 10.1007/978-3-540-28644-8_2.

[15] Cristiano Calcagno, Dino Distefano, Peter O'Hearn, and Hongseok Yang. "Compositional Shape Analysis by Means of Bi-Abduction". In: *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '09. Savannah, GA, USA: Association for Computing Machinery, 2009, pp. 289–300. DOI: 10.1145/1480881.1480917.

[16] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. "Private information retrieval". In: *J. ACM* 45.6 [Nov. 1998], pp. 965–981. ISSN: 0004-5411. DOI: 10.1145/293347.293350.

[17] The mathlib Community. "The lean mathematical library". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 367–381. ISBN: 9781450370974. DOI: 10.1145/3372885.3373824.

[18] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. "Spanner: Google's globally-distributed database". In: *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*. OSDI'12. Hollywood, CA, USA: USENIX Association, 2012, pp. 251–264. ISBN: 9781931971966. URL: https://dl.acm.org/doi/10.5555/2387880.2387905.

[19] Patrick Cousot and Radhia Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". In: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '77. Los Angeles, California: Association for Computing Machinery, 1977, pp. 238–252. ISBN: 9781450373500. DOI: 10.1145/512950.512973.

[20] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. "Why does Astrée scale up?" In: *Formal Methods Syst. Des.* 35.3 [2009], pp. 229–264. DOI: 10.1007/S10703-009-0089-6.

[21] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. "Scaling Static Analyses at Facebook". In: *Commun. ACM* 62.8 [July 2019], pp. 62–70. ISSN: 0001-0782. DOI: 10.1145/3338112.

[22] Weijie Fan, Hongjin Liang, Xinyu Feng, and Hanru Jiang. "A Program Logic for Concurrent Randomized Programs in the Oblivious Adversary Model". In: *Programming Languages and Systems*. Ed. by Viktor Vafeiadis. Cham: Springer Nature Switzerland, 2025, pp. 322–348. ISBN: 978-3-031-91118-7. DOI: 10.1007/978-3-031-91118-7_13.

[23] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Shanghai, China: Association for Computing Machinery, 2017, pp. 51–68. ISBN: 9781450350853. DOI: 10.1145/3132747.3132757.

[24] Sergiu Hart, Micha Sharir, and Amir Pnueli. "Termination of Probabilistic Concurrent Program". In: *ACM Trans. Program. Lang. Syst.* 5.3 [July 1983], pp. 356–380. ISSN: 0164-0925. DOI: 10.1145/2166.357214.

[25] Stefan Heule, Marc Nunkesser, and Alexander Hall. "HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm". In: *Proceedings of the 16th International Conference on Extending Database Technology*. EDBT '13. Genoa, Italy: Association for Computing Machinery, 2013, pp. 683–692. ISBN: 9781450315975. DOI: 10.1145/2452376.2452456.

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

[27] Bart Jacobs. "From Multisets over Distributions to Distributions over Multisets". In: *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '21. Rome, Italy: Association for Computing Machinery, 2021. ISBN: 9781665448956. DOI: 10.1109/LICS52264.2021.9470678.

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

[29] Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir Frolov, Erik Reeber, and Armaghan Naik. "Replacing Testing with Formal Verification in Intel Core i7 Processor Execution Engine Validation". In: *Computer Aided Verification*. Ed. by Ahmed Bouajjani and Oded Maler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 414–429. ISBN: 978-3-642-02658-4. DOI: 10.1007/978-3-642-02658-4_32.

[30] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. "seL4: formal verification of an OS kernel". In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP '09. Big Sky, Montana, USA: Association for Computing Machinery, 2009, pp. 207–220. ISBN: 9781605587523. DOI: 10.1145/1629575.1629596.

[31] Dexter Kozen and Alexandra Silva. "Multisets and Distributions". In: *Logics and Type Systems in Theory and Practice: Essays Dedicated to Herman Geuvers on The Occasion of His 60th Birthday*. Ed. by Venanzio Capretta, Robbert Krebbers, and Freek Wiedijk. Cham: Springer Nature Switzerland, 2024, pp. 168–187. ISBN: 978-3-031-61716-4. DOI: 10.1007/978-3-031-61716-4_11.

[32] Quang Loc Le, Azalea Raad, Jules Villard, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. "Finding Real Bugs in Big Programs with Incorrectness Logic". In: *Proc. ACM Program. Lang.* 6.OOPSLA1 [Apr. 2022]. DOI: 10.1145/3527325.

[33] Daniel Lehmann and Michael O. Rabin. "On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem". In: *Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '81. Williamsburg, Virginia: Association for Computing Machinery, 1981, pp. 133–138. ISBN: 089791029X. DOI: 10.1145/567532.567547.

[34] John M. Li, Amal Ahmed, and Steven Holtzen. "Lilac: A Modal Separation Logic for Conditional Probability". In: *Proc. ACM Program. Lang.* 7.PLDI [June 2023]. DOI: 10.1145/3591226.

[35] Kwing Hei Li, Alejandro Aguirre, Simon Oddershede Gregersen, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. Modular Reasoning about Error Bounds for Concurrent Probabilistic Programs. 2025. arXiv: 2503.04512 [cs.LO]. URL: https://arxiv.org/abs/2503.04512.

[36] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. "The Honey Badger of BFT Protocols". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 31–42. ISBN: 9781450341394. DOI: 10.1145/2976749.2978399.

[37] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. "The Lean Theorem Prover (System Description)". In: *Automated Deduction - CADE-25*. Ed. by Amy P. Felty and Aart Middeldorp. Cham: Springer International Publishing, 2015, pp. 378–388. ISBN: 978-3-319-21401-6. DOI: 10.1007/978-3-319-21401-6_26.

[38] Chris Newcombe. "Why Amazon Chose TLA+". In: *Abstract State Machines, Alloy, B, TLA, VDM, and Z*. Ed. by Yamine Ait Ameur and Klaus-Dieter Schewe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 25–39. ISBN: 978-3-662-43652-3. DOI: 10.1007/978-3-662-43652-3_3.

[39] Peter W. O'Hearn. "Resources, Concurrency and Local Reasoning". In: *CONCUR 2004 - Concurrency Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 49–67. ISBN: 978-3-540-28644-8. DOI: 10.1016/j.tcs.2006.12.035.

[40] Peter W. O'Hearn. "Incorrectness Logic". In: *Proc. ACM Program. Lang.* 4.POPL [Jan. 2020]. DOI: 10.1145/3371078.

[41] Azalea Raad, Josh Berdine, Hoang-Hai Dang, Derek Dreyer, Peter O'Hearn, and Jules Villard. "Local Reasoning About the Presence of Bugs: Incorrectness Separation Logic". In: *Computer Aided Verification*. Cham: Springer International Publishing, 2020, pp. 225–252. ISBN: 978-3-030-53291-8. DOI: 10.1007/978-3-030-53291-8_14.

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

[43] Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny. "Formal Verification of Avionics Software Products". In: *FM 2009: Formal Methods*. Ed. by Ana Cavalcanti and Dennis R. Dams. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 532–546. ISBN: 978-3-642-05089-3. DOI: 10.1007/978-3-642-05089-3_3.

[44] Joseph Tassarotti and Robert Harper. "A Separation Logic for Concurrent Randomized Programs". In: *Proc. ACM Program. Lang.* 3.POPL [Jan. 2019]. DOI: 10.1145/3290377.

[45] Daniele Varacca. "The powerdomain of indexed valuations". In: *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*. 2002, pp. 299–308. DOI: 10.1109/LICS.2002.1029838.

[46] John von Neumann. "Various techniques used in connection with random digits". In: *Monte Carlo Method*. Ed. by A.S. Householder, G.E. Forsythe, and H.H. Germond. Washington, D.C.: U.S. Government Printing Office: National Bureau of Standards Applied Mathematics Series, 12, 1951, pp. 36–38.