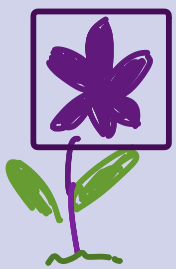
#Slides

Higher-Order Separation Logic For Probabilistic Programs



5th November, 2024

Hei Li[1]
PhD student of LogSem
Supervised by Lars Birkedal

Joint work with:
Philipp Haselwarter[1]
Alejandro Aguirre[1]
Simon Gregersen[2]
Markus de Medeiros[2]
Joe Tassarotti[2]
Lars Birkedal[1]

[1] × NYU[2]

Probabilistic Programs

Useful

- Cryptography
- Randomized data structures/algos

Probabilistic Programs

Useful

- Cryptography
- Randomized data structures/algos

} How can we reason about and verify properties of these probabilistic programs?

Probabilistic Programs

Useful
- Cryptography
- Randomized data structures/algos

How can we reason about and verify properties of these probabilistic programs?

Many approaches, including
1. Denotational Semantics
2. Martingale Analysis
3. Program Logics

Probabilistic Programs

Useful
- Cryptography
- Randomized data structures/algos

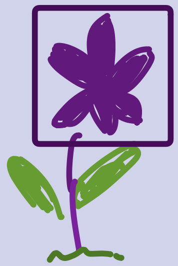
How can we reason about and verify properties of these probabilistic programs?

Many approaches, including
1. Denotational Semantics
2. Martingale Analysis
3. Program Logics

Limitations:
- Missing complex language features
  - higher-order
  - local state
- Modularity/compositional reasoning
- (Not mechanized/foundational)

Our Approach

We construct probabilistic program logics on top of

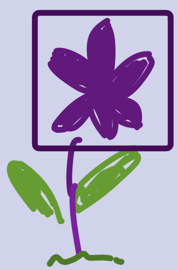


Iris, a

- Step-indexed
- Higher-order
- Machine-checked
- Separation Logic Framework

Our Approach

We construct probabilistic program logics on top of

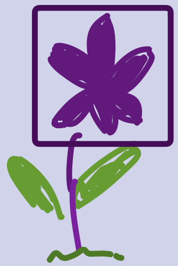


Iris, a
- Step-indexed
- Higher-order
- Machine-checked
- Separation Logic Framework

We develop program logics that are
- Higher order
- Can reason about higher-order stores
- Modular
- All formalized!

Our Approach

We construct probabilistic program logics on top of



Iris, a
- Step-indexed
- Higher-order
- Machine-checked
- Separation Logic Framework

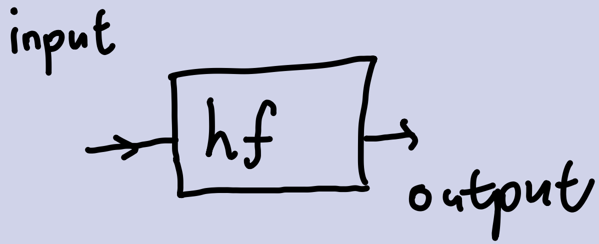
We develop program logics that are
- Higher order
- Can reason about higher-order stores
- Modular
- All formalized!

Examples:
- Clutch: asynchronous couplings (POPL 2024)
- Caliper: termination preserving refinements (ICFP 2024)
- Eris: error bounds (ICFP 2024)
- Tachis: Expected cost (OOPSLA 2024)
- Approxis: approximate refinement (cond. accepted @ POPL 2025)

Something I contributed to!

Case Study

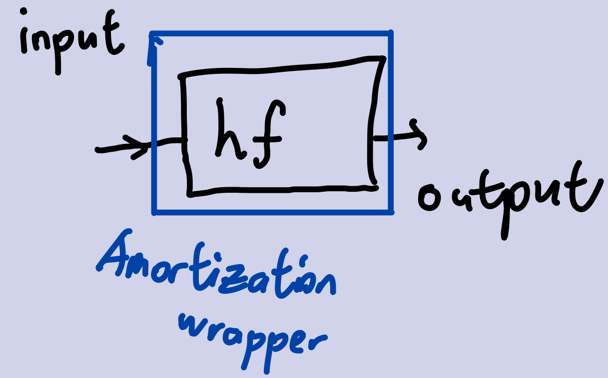
Stage 1. An idealized hash function with error proportional to queries made



Case Study

Stage 1. An idealized hash function with error proportional to queries made

Stage 2: An amortized hash with Constant error

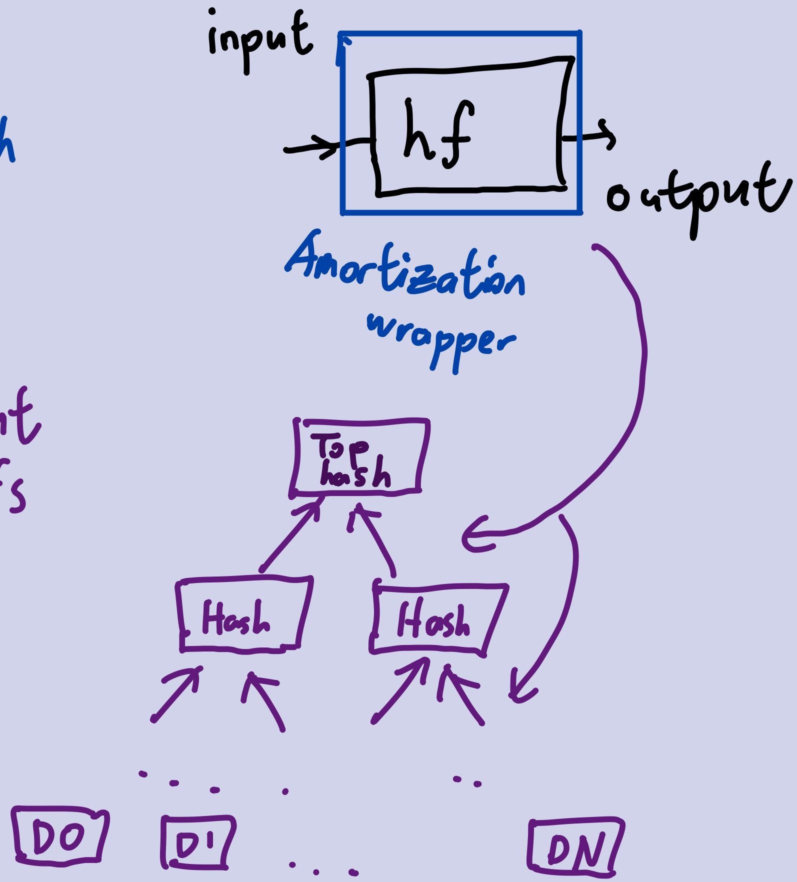


Case Study

Stage 1. An idealized hash function with error proportional to queries made

Stage 2: An amortized hash with <u>constant</u> error

Stage 3: A merkle tree with constant error bounds for validating proofs



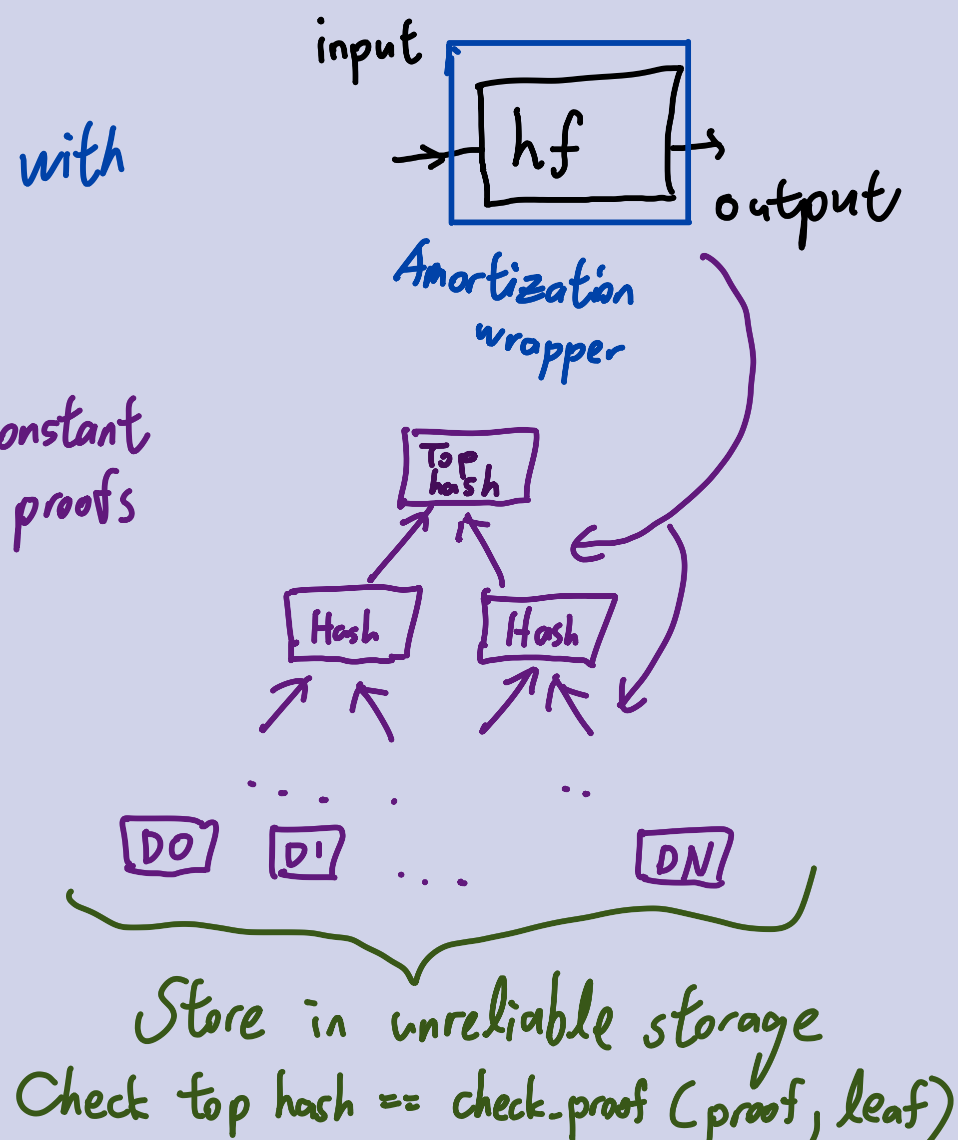
Case Study

Stage 1. An idealized hash function with error proportional to queries made

Stage 2: An amortized hash with Constant error

Stage 3: A merkle tree with constant error bounds for validating proofs

Stage 4: Unreliable storage



Conclusion

By formalizing our program logics with Iris,
we can now verify many large, complex, and EXCITING examples,
that are beyond the scope of previous techniques!

Conclusion

By formalizing our program logics with Iris,
we can now verify many large, complex, and EXCITING examples,
that are beyond the scope of previous techniques!

Many more convincing examples:

- Security of ElGamal Public Key Encryption
- Amortized error bounds for hash map with resizing
- Expected cost of randomized meldable heap
- IND$-CPA security of symmetric encryption
- Contextual equivalences of B^+ tree sampling algorithms
- Many more ... =)