Automated Reasoning and Randomization in Separation Logic

Ph.D. Defense

Christoph Matheja

January 9th, RWTH Aachen University
A Common Error in Programming (since 1967)

\[ x := \langle E \rangle \text{ // store value at address } E \text{ in } x \]
\[ y := \text{object.field} \]
A Common Error in Programming (since 1967)

\[ x := \langle E \rangle \quad //\text{store value at address } E \text{ in } x \]

\[ y := \text{object.field} \]

What happens if \( E \) is not a valid address?

What happens if \( \text{object equals null} \)?
x := \langle E \rangle // store value at address E in x
y := object.field

What happens if $E$ is not a valid address?
What happens if object equals null?
A Common Error in Programming (since 1967)

\[ x := \langle E \rangle \quad \text{// store value at address } E \text{ in } x \]
\[ y := \text{object.field} \]

What happens if \( E \) is not a valid address?
What happens if \text{object} \text{ equals} \text{ null}?

“[The null reference] has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

Common \text{V}ulnerabilities and \text{E}xposures:
\( > 31000 \) records on wrong pointer usage
Tree cur = root; // input
Tree sen = new Tree();
Tree prev = sen;
while(cur != sen){
    Tree next = cur.left;
    cur.left = cur.right;
    cur.right = prev;
    prev = cur;
    cur = next;
    if (cur == null){
        cur = prev;
        prev = null;
    }
}
Tree cur = root; // input
Tree sen = new Tree();
Tree prev = sen;
while (cur != sen) {
    Tree next = cur.left;
    cur.left = cur.right;
    cur.right = prev;
    prev = cur;
    cur = next;
    if (cur == null) {
        cur = prev;
        prev = null;
    }
}
Tree cur = root; // input
Tree sen = new Tree();
Tree prev = sen;
while(cur != sen){
    Tree next = cur.left;
    cur.left = cur.right;
    cur.right = prev;
    prev = cur;
    cur = next;
    if (cur == null){
        cur = prev;
        prev = null;
    }
}
Tree cur = root; // input
Tree sen = new Tree();
Tree prev = sen;
while (cur != sen) {
    Tree next = cur.left;
    cur.left = cur.right;
    cur.right = prev;
    prev = cur;
    cur = next;
    if (cur == null) {
        cur = prev;
        prev = null;
    }
}
Programming with pointers often enables efficient implementations, but it's easy to make mistakes.
What will be the next billion dollar mistake in programming?

“In no other branch of mathematics it is so easy to make mistakes as in probability theory.”

H. Tijms
What will be the next billion dollar mistake in programming?

“In no other branch of mathematics it is so easy to make mistakes as in probability theory.”

Probabilistic Programs: Ordinary program + Sampling

```
geometric() {
    coin := flip();
    if (coin = heads) {
        return 0
    } else {
        return 1 + geometric()
    }
}
```
What will be the next billion dollar mistake in programming?

“In no other branch of mathematics it is so easy to make mistakes as in probability theory.”

Probabilistic Programs: Ordinary program + Sampling

```javascript
geometric() {
  coin := flip();
  if (coin = heads) {
    return 0
  } else {
    return 1 + geometric()
  }
}
```

![Bar graph showing the probability distribution of the returned value for the geometric() function.](image)
What will be the next billion dollar mistake in programming?

“In no other branch of mathematics it is so easy to make mistakes as in probability theory.”

Probabilistic Programs: Ordinary program + Sampling

geometric() {
    coin := flip();
    if (coin = heads) {
        return 0
    } else {
        return 1 + geometric()
    }
}

Phenomena:
- Programs may fail with some probability
- Having a finite expected runtime is not compositional [ESOP'16, J.ACM'18]
- Reasoning about termination is harder than for ordinary programs [Kaminski & Katoen'15, Acta Inf.'19]
Practical Relevance: Alternative to Statistical Models
Verifying Quantitative Reliability for Programs
That Execute on Unreliable Hardware

Michael Carbin  Sasa Misailovic  Martin C. Rinard
MIT CSAIL
{mcarbin, misailo, rinard}@csail.mit.edu

Abstract
Emerging high-performance architectures are anticipated to contain unreliable components that may exhibit soft errors, which silently corrupt the results of computations. Full detection and masking of soft errors is challenging, expensive, and, for some applications, unnecessary. For example, approximate computing applications (such as multimedia processing, machine learning, and big data analytics) can often naturally tolerate soft errors.

1. Introduction
System reliability is a major challenge in the design of emerging architectures. Energy efficiency and circuit scaling are becoming major goals when designing new devices. However, aggressively pursuing these design goals can often increase the frequency of soft errors in small [67] and large systems [10] alike. Researchers have developed numerous techniques for detecting and masking soft errors in both hardware [22] and software [20, 52, 57, 64]. These tech-
Practical Relevance: Randomized Algorithms

“Randomized skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.” [Pugh ‘89]
“Randomized skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.” [Pugh'89]

“The expected running time [of randomized splay trees] is smaller [than deterministic variants].” [Albers & Karpinski’02]
“Randomized skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.” [Pugh’89]

“The expected running time [of randomized splay trees] is smaller [than deterministic variants].” [Albers & Karpinski’02]

“We present a randomized approach to the problem of efficient meldable priority queue implementation.” [Gambin & Malinowski’88]
“Randomized skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.” [Pugh’89]

“The expected running time [of randomized splay trees] is smaller [than deterministic variants].” [Albers & Karpinski’02]

“We present a randomized approach to the problem of efficient meldable priority queue implementation.” [Gambin & Malinowski’88]

Observation: These are probabilistic pointer programs
“Randomized skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.” [Pugh’89]

“The expected running time [of randomized splay trees] is smaller [than deterministic variants].” [Albers & Karpinski’02]

“We present a randomized approach to the problem of efficient meldable priority queue implementation.” [Gambin & Malinowski’88]

Observation: These are probabilistic pointer programs

1st chapter of [Motwani & Raghavan’95] on applications is titled data structures
“Randomized skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.” [Pugh’89]

“The expected running time [of randomized splay trees] is smaller [than deterministic variants].” [Albers & Karpinski’02]

“We present a randomized approach to the problem of efficient meldable priority queue implementation.” [Gambin & Malinowski’88]

Observation: These are probabilistic pointer programs

1st chapter of [Motwani & Raghavan’95] on applications is titled data structures

This talk: How to reason about such programs?
Outline

0. Introduction

1. Primer on Separation Logic

2. Mechanized Reasoning about Probabilistic Pointer Programs

3. Automated Reasoning with/about Separation Logic

4. Epilogue
1. Primer on Separation Logic

Separation Logic = Hoare Logic + Pointers + Local Reasoning
The Hoare triple $\langle \text{Pre} \rangle C \langle \text{Post} \rangle$ is valid iff

1. whenever we start program $C$ in some state satisfying precondition $\text{Pre}$,

2. $C$ terminates in some state satisfying postcondition $\text{Post}$.

The weakest precondition $\text{wp}_J C K (\text{Post})$ of $C$ and $\text{Post}$ captures the largest set of states such that $\langle \text{wp}_J C K (\text{Post}) \rangle C \langle \text{Post} \rangle$ is valid.
The Hoare triple \( \langle \text{Pre} \rangle C \langle \text{Post} \rangle \) is valid iff

1. whenever we start program \( C \) in some state satisfying precondition \( \text{Pre} \), then
2. \( C \) terminates in some state satisfying postcondition \( \text{Post} \).
The Hoare triple $\langle Pre \rangle C \langle Post \rangle$ is valid iff

1. whenever we start program $C$ in some state satisfying precondition $Pre$, then
2. $C$ terminates in some state satisfying postcondition $Post$.

The weakest precondition $wp[\lceil C \rceil](Post)$ of $C$ and $Post$ captures the largest set of states such that $\langle wp[\lceil C \rceil](Post) \rangle C \langle Post \rangle$ is valid.
Applying Hoare Logic to Pointer Programs: A Naïve Example

Syntactic Definition of \( \text{wp} \):

\[
\begin{align*}
\text{wp} \left[ x := E \right] (Post) &= Post \left[ x/E \right] \\
\text{wp} \left[ C_1; C_2 \right] (Post) &= \text{wp} \left[ C_1 \right] (\text{wp} \left[ C_2 \right] (Post))
\end{align*}
\]
Applying Hoare Logic to Pointer Programs: A Naïve Example

Syntactic Definition of $wp$:

$$
\langle x \rangle := 3; \\
\langle y \rangle := 17; \\
z := \langle x \rangle \\
// z = 3
$$

$$
wp[x := E] (Post) = Post[x/E] \\
wp[C_1 ; C_2] (Post) = wp[C_1] (wp[C_2] (Post)) \\
$$
Applying Hoare Logic to Pointer Programs: A Naïve Example

Syntactic Definition of \( \text{wp} \):

\[
\begin{align*}
\langle x \rangle & := 3; \\
\langle y \rangle & := 17; \\
\text{\textit{}} & \langle x \rangle = 3 \\
// & \langle x \rangle = 3 \\
z & := \langle x \rangle \\
// & z = 3
\end{align*}
\]

\[
\begin{align*}
\text{wp} \left[ x := E \right] (\text{Post}) & = \text{Post} \left[ x/E \right] \\
\text{wp} \left[ C_1 ; C_2 \right] (\text{Post}) & = \text{wp} \left[ C_1 \right] (\text{wp} \left[ C_2 \right] (\text{Post}))
\end{align*}
\]
Applying Hoare Logic to Pointer Programs: A Naïve Example

Syntactic Definition of $wp$:

$$\\langle x \rangle := 3 ;$$
$$// \ \langle x \rangle = 3$$
$$\langle y \rangle := 17 ;$$
$$// \ \langle x \rangle = 3$$
$$z := \langle x \rangle$$
$$// \ z = 3$$

$$wp[x := E](Post) = Post[x/E]$$

$$wp[C_1 ; C_2](Post) = wp[C_1](wp[C_2](Post))$$

...
Applying Hoare Logic to Pointer Programs: A Naïve Example

// 3 = 3
⟨x⟩ := 3;
// ⟨x⟩ = 3
⟨y⟩ := 17;
// ⟨x⟩ = 3
z := ⟨x⟩
// z = 3

Syntactic Definition of \( wp \):

\[
wp \left[ x := E \right] (Post) = Post \left[ x/E \right]
\]

\[
wp \left[ C_1 ; C_2 \right] (Post) = wp \left[ C_1 \right] (wp \left[ C_2 \right] (Post))
\]

⇒ Separation Logic
Applying Hoare Logic to Pointer Programs: A Naïve Example

// 3 = 3
⟨x⟩ := 3;

// ⟨x⟩ = 3
⟨y⟩ := 17;

// ⟨x⟩ = 3
z := ⟨x⟩

// z = 3

Syntactic Definition of wp:

\[
wp [x := E] (Post) = Post [x/E]
\]

\[
wp [C_1 ; C_2] (Post) = wp [C_1] (wp [C_2] (Post))
\]

⇒ Separation Logic

So the program always terminates with \( z = 3 \)?
Applying Hoare Logic to Pointer Programs: A Naïve Example

// 3 = 3
⟨x⟩ := 3;
// ⟨x⟩ = 3
⟨y⟩ := 17;
// ⟨x⟩ = 3
z := ⟨x⟩
// z = 3

So the program always terminates with z = 3?

No! If x = y, then z = 17.

Syntactic Definition of wp:

\[
\begin{align*}
wp[x := E](Post) &= Post[x/E] \\
wp[C_1; C_2](Post) &= wp[C_1](wp[C_2](Post))
\end{align*}
\]
Applying Hoare Logic to Pointer Programs: A Naïve Example

// 3 = 3
⟨ x ⟩ := 3;
// ⟨ x ⟩ = 3
⟨ y ⟩ := 17;
// ⟨ x ⟩ = 3
z := ⟨ x ⟩
// z = 3

So the program always terminates with z = 3?

No! If x = y, then z = 17.

No! If x = 0 or y = 0, then the program crashes.

Syntactic Definition of wp:

\[
\begin{align*}
wp \left[ x := E \right] (Post) &= Post \left[ x/E \right] \\
wp \left[ C_1 ; C_2 \right] (Post) &= wp \left[ C_1 \right] (wp \left[ C_2 \right] (Post))
\end{align*}
\]
Applying Hoare Logic to Pointer Programs: A Naïve Example

// 3 = 3
⟨x⟩ := 3;
// ⟨x⟩ = 3
⟨y⟩ := 17;
// ⟨x⟩ = 3
z := ⟨x⟩
// z = 3

So the program always terminates with z = 3?

No! If x = y, then z = 17.

No! If x = 0 or y = 0, then the program crashes.

⇒ Separation Logic
States consist of **values assigned to variables** and **random access memory**

- modeled by stack \( s : \text{Vars} \rightarrow \mathbb{Z} \)
- modeled by heap \( h : \mathbb{N}_{>0} \rightarrow \text{fin}\mathbb{Z} \)
States consist of **values assigned to variables** and **random access memory**

- Modeled by **stack** $s : \text{Vars} \rightarrow \mathbb{Z}$
- Modeled by **heap** $h : \mathbb{N}_{>0} \rightarrow \text{fin} \mathbb{Z}$

\[
\begin{align*}
\text{x, y} \\
\downarrow \\
9 & \ 10 & 17 & 18 & 0 \\
\text{42} & \text{23} \\
\end{align*}
\]

\[
\begin{align*}
x := \langle y \rangle \\
\end{align*}
\]
States consist of **values assigned to variables** and **random access memory**

- **Stack** modeled by \( s: \text{Vars} \rightarrow \mathbb{Z} \)
- **Heap** modeled by \( h: \mathbb{N}_{>0} \rightarrow \text{fin}\mathbb{Z} \)

**Predicates** \( P: \text{States} \rightarrow \{\text{true}, \text{false}\} \) capture sets of states

- **empty-heap:** \( s, h \models \text{emp} \) iff \( \text{dom}(h) = \emptyset \)
- **single memory cell in which \( x \) points-to 17:** \( s, h \models x \leftrightarrow 17 \) iff \( h = \{s(x) \mapsto 17\} \)
- **heap contains a pointer from \( x \) to 17:** \( s, h \models x \mapsto 17 \) iff \( h(s(x)) = 17 \)
Separation Logic Predicates

Separating Conjunction:

\[ s, h \models P \star Q \iff \exists h_1, h_2 : h = h_1 \uplus h_2 \text{ and } s, h_1 \models P \text{ and } s, h_2 \models Q \]
Separation Logic Predicates

Separating Conjunction:
\[ s, h \models P \star Q \iff \exists h_1, h_2 : h = h_1 \uplus h_2 \text{ and } s, h_1 \models P \text{ and } s, h_2 \models Q \]

Example:
\[
\begin{array}{c}
\begin{array}{c}
4 \\
\bullet \\
\end{array} & \uplus & \\
\begin{array}{c}
7 \\
\bullet \\
\end{array}
\end{array}
\models 4 \leftrightarrow 7 \star (4 \leftrightarrow 1 \rightarrow 4 \leftarrow 1)
\]
Separation Logic Predicates [Ishtiaq & O’Hearn’01, Reynolds’02]

Separating Conjunction:
\[ s, h \models P \star Q \iff \exists h_1, h_2 : h = h_1 \uplus h_2 \text{ and } s, h_1 \models P \text{ and } s, h_2 \models Q \]

Example:

\[ 4 \leftarrow 7 \]
\[ 1 \leftarrow 7 \]

\[ \models 4 \leftrightarrow 7 \star (4 \leftrightarrow 1 \rightarrow 4 \leftarrow 1) \]

iff

\[ 4 \leftrightarrow 1 \rightarrow 4 \leftarrow 1 \]
Separation Logic Predicates  

[Ishtiaq & O’Hearn’01, Reynolds’02]

**Separating Conjunction:**

\[ s, h \models P \star Q \iff \exists h_1, h_2 : h = h_1 \uplus h_2 \text{ and } s, h_1 \models P \text{ and } s, h_2 \models Q \]

**Separating Implication:**

\[ s, h \models P \multimap Q \iff \forall h' : (h \not= h' \text{ and } s, h' \models P) \text{ implies } s, h \uplus h' \models Q \]

**Example:**

\[ \begin{array}{c}
4 & \quad \uplus \quad & 7 \\
\text{iff} & & \text{iff}
\end{array} \]

\[ \models 4 \multimap 7 \star (4 \multimap 1 \multimap 4 \multimap 1) \]

\[ \models 4 \multimap 1 \multimap 4 \multimap 1 \]
Separation Logic Predicates

[Ishtiaq & O’Hearn’01, Reynolds’02]

Separating Conjunction:
\[ s, h \models P \star Q \iff \exists h_1, h_2 : h = h_1 \uplus h_2 \text{ and } s, h_1 \models P \text{ and } s, h_2 \models Q \]

Separating Implication:
\[ s, h \models P \rightarrow Q \iff \forall h' : (h \neq h' \text{ and } s, h' \models P) \text{ implies } s, h \uplus h' \models Q \]

Example:

\[
\begin{align*}
4 \leftrightarrow 7 \uplus (4 \leftrightarrow 1 \rightarrow 4 \leftrightarrow 1) \\
\text{iff} \\
1 \leftrightarrow 7 \uplus (4 \leftrightarrow 1 \rightarrow 4 \leftrightarrow 1) &\models 4 \leftrightarrow 1 \rightarrow 4 \leftrightarrow 1 \\
\text{iff} \\
1 \leftrightarrow 4 \uplus 7 &\models 4 \leftrightarrow 1 \\
\end{align*}
\]
Alias prevention:

\[ x \mapsto u \star y \mapsto v \text{ implies } x \neq y \]
Separation Logic Proof Rules

Alias prevention:
\[ x \mapsto u \ast y \mapsto v \implies x \neq y \]

Rule for mutation:
\[ \text{wp} \{ \langle E \rangle := E' \} (Q) = E \mapsto - \ast (E \mapsto E' \mapsto Q) \]
where \[ x \mapsto - = \exists y : x \mapsto y \]
Separation Logic Proof Rules  

**Alias prevention:**  
\[ x \rightarrow u \star y \rightarrow v \] implies \( x \neq y \)

**Rule for mutation:**  
\[ \text{wp}[\langle E \rangle := E'] (Q) = E \rightarrow \neg \star (E \rightarrow E' \rightarrow Q) \]  
where \( x \rightarrow \neg = \exists y : x \leftrightarrow y \)

---

**Naïve Example Revisited:**  
// false for \( x = y, x = 0, \) or \( y = 0 \)  
// \( x \rightarrow \neg \star y \rightarrow \neg \star \text{true} \)  
// \( x \rightarrow \neg \star (x \rightarrow 3 \rightarrow \star) \)  
// \( (y \rightarrow \neg \star (y \rightarrow 17 \rightarrow \star x \leftrightarrow 3)) \)  
\( \langle x \rangle := 3; \)  
// \( y \rightarrow \neg \star (y \rightarrow 17 \rightarrow \star x \leftrightarrow 3) \)  
\( \langle y \rangle := 17; \)  
// \( x \leftrightarrow 3 \)  
// \( \exists v : x \leftrightarrow v \land v = 3 \)  
// \( \exists v : x \leftrightarrow v \star (x \leftrightarrow v \star v = 3) \)  
\( z := \langle x \rangle \)  
// \( z = 3 \)
Local Reasoning about Dynamic Data Structures

Specifying data structures:

\[ u \mapsto v, w = u \mapsto v \ast u + 1 \mapsto w \]
Local Reasoning about Dynamic Data Structures

Specifying data structures:

\[ u \mapsto v, w = u \mapsto v \star u + 1 \mapsto w \]

\[ \text{tree}(u) = ( u = 0 \lor \text{emp} ) \]
\[ \lor ( \exists v, w: u \mapsto v, w \star \text{tree}(v) \star \text{tree}(w) ) \]
Specifying data structures:

\[ u \mapsto v, w = u \mapsto v * u + 1 \mapsto w \]

\[ \text{tree}(u) = (u = 0 \lor \text{emp}) \lor (\exists v, w: u \mapsto v, w * \text{tree}(v) * \text{tree}(w)) \]

// \text{tree}(x)

\text{delete}(x) \{ \\
\text{if } (x \neq 0) \{ \\
\text{left} := \langle x \rangle; \text{right} := \langle x + 1 \rangle; \\
\text{delete}(\text{left}); \\
\text{delete}(\text{right}); \\
\text{free}(x); \text{free}(x + 1) \\
\} \} // \text{emp}
Specifying data structures:

\[ u \mapsto v, w = u \mapsto v \star u + 1 \mapsto w \]

\[
\text{tree}(u) = ( u = 0 \lor \text{emp}) \\
\lor ( \exists v, w: u \mapsto v, w \star \text{tree}(v) \star \text{tree}(w) )
\]

// tree(x)
delete(x) {
    if (x \neq 0) {
        left := \langle x \rangle; right := \langle x + 1 \rangle;
        delete(left);
        // ...
        // x \mapsto -, -, * tree(right)
        // x \mapsto -, -, * wlp[delete(right)](emp)
        // wlp[delete(right)](x \mapsto -, -, * emp)
        delete(right);
        // x \mapsto -, -, * emp
        free(x); free(x + 1)
        // emp
    }
}
} // emp
2. Mechanized Reasoning about Probabilistic Pointer Programs

“Instead of truth, we measure a quantity”
delete (x) {
    fail := flipCoin () ;
    if ( x \neq 0 and fail = heads ) {
        left := \langle x \rangle ; right := \langle x + 1 \rangle ;
        delete (left) ;
        delete (right) ;
        free(x) ; free(x + 1)
    }
}

What is the probability to terminate successfully with an empty heap?
What is the expected amount of garbage?
What is the probability that at least half the tree is deleted?
From Qualitative to Quantitative Reasoning

```
delete(x) {
    fail := flipCoin();
    if (x ≠ 0 and fail = heads) {
        left := ⟨x⟩; right := ⟨x + 1⟩;
        delete(left);
        delete(right);
        free(x); free(x + 1)
    }
}
```

What is the probability to terminate successfully with an empty heap?

What is the expected amount of garbage?

What is the probability that at least half the tree is deleted?
From Qualitative to Quantitative Reasoning

\[
d\{x\} \{ \\
\text{fail} := \text{flipCoin}() ; \\
\text{if (} x \neq 0 \text{ and } \text{fail} = \text{heads} \text{)} \{ \\
\text{left} := \langle x \rangle ; \text{right} := \langle x + 1 \rangle ; \\
\text{delete(left)} ; \\
\text{delete(right)} ; \\
\text{free}(x) ; \text{free}(x + 1) \\
\text{\}} \\
\text{\}}
\]

What is the probability to terminate successfully with an empty heap?

What is the expected amount of garbage?
From Qualitative to Quantitative Reasoning

```plaintext
define delete(x) {
    fail := flipCoin();
    if (x ≠ 0 and fail = heads) {
        left := ⟨x⟩; right := ⟨x + 1⟩;
        delete(left);
        delete(right);
        free(x); free(x + 1)
    }
}
```

What is the **probability** to terminate successfully with an empty heap?

What is the **expected** amount of garbage?

What is the **probability** that at least half the tree is deleted?
Quantitative Separation Logic (QSL) — Overview

2.1. Assertion Language

2.2. Verification System

2.3. Theorems

2.4. Case Studies

2.5. Final remarks
Previously: $F : \text{States} \rightarrow \{\text{true}, \text{false}\}$
Assertion Language: Expectations

Previously: \( F: \text{States} \rightarrow \{\text{true, false}\} \)

Expectations: \( f: \text{States} \rightarrow \mathbb{R}_{\geq 0}^{\infty} \)
Assertion Language: Expectations

Previously: \[ F : \text{States} \rightarrow \{ \text{true, false} \} \]

Expectations: \[ f : \text{States} \rightarrow \mathbb{R}_{\geq 0} \]  
Think: random variable
Previously: \[ F : \text{States} \rightarrow \{\text{true, false}\} \]

Expectations: \[ f : \text{States} \rightarrow \mathbb{R}_{\geq 0} \quad \text{Think: random variable} \]

Examples:

\[ \frac{1}{2} = \lambda(s, h). \frac{1}{2} \]
Assertion Language: Expectations

Previously: \( F : \text{States} \rightarrow \{ \text{true}, \text{false} \} \)

Expectations: \( f : \text{States} \rightarrow \mathbb{R}_{\geq 0} \)  \( \text{Think: random variable} \)

Examples:

\[
\frac{1}{2} = \lambda(s, h) \cdot \frac{1}{2} \quad \text{and} \quad x^2 = \lambda(s, h) \cdot s(x)^2
\]
Assertion Language: Expectations

Previously: \( F : \text{States} \rightarrow \{\text{true, false}\} \)

Expectations: \( f : \text{States} \rightarrow \mathbb{R}_\geq 0 \)  
Think: random variable

Examples:

\[
\frac{1}{2} = \lambda(s, h). \frac{1}{2} \quad \quad x^2 = \lambda(s, h). s(x)^2 \quad \quad \text{size} = \lambda(s, h). |\text{dom}(h)|
\]
Assertion Language: Expectations

Previously: \( F : \text{States} \rightarrow \{\text{true, false}\} \)

Expectations: \( f : \text{States} \rightarrow \mathbb{R}_0^\infty \) \hspace{1cm} \text{Think: random variable}

Examples:

\[ \frac{1}{2} = \lambda(s, h). \frac{1}{2} \quad \quad x^2 = \lambda(s, h). s(x)^2 \quad \quad \text{size} = \lambda(s, h). |\text{dom}(h)| \]

\[ [F] = \lambda(s, h). \begin{cases} 1, & \text{if } s, h \models F \\ 0, & \text{otherwise} \end{cases} \]
Quantitative Conjunction

Classical conjunction:

\[ F \land G \]
Quantitative Conjunction

Classical conjunction:

\[ F \land G \]

Quantitative conjunction:

\[ f \cdot g = \lambda(s, h). \ f(s, h) \cdot g(s, h) \]
Quantitative Conjunction

Classical conjunction:

\[ F \land G \]

Quantitative conjunction:

\[ f \cdot g = \lambda(s, h). f(s, h) \cdot g(s, h) \]

Notice:

\[ [F \land G] = [F] \cdot [G] \]
Quantitative Separating Conjunction

Classical separating conjunction:

\[ s, h \models F \star G \quad \text{iff} \quad \exists h_1, h_2 : h = h_1 \uplus h_2 \quad \text{and} \quad s, h_1 \models F \quad \text{and} \quad s, h_2 \models G \]
Quantitative Separating Conjunction

Classical separating conjunction:

\[ s, h \models F \star G \quad \text{iff} \quad \exists h_1, h_2 : h = h_1 \uplus h_2 \quad \text{and} \quad s, h_1 \models F \quad \text{and} \quad s, h_2 \models G \]
Quantitative Separating Conjunction

Classical separating conjunction:

\[ s, h \models F \star G \quad \text{iff} \quad \exists h_1, h_2 : h = h_1 \uplus h_2 \quad \text{and} \quad s, h_1 \models F \quad \text{and} \quad s, h_2 \models G \]

Quantitative separating conjunction:

\[ f \star g = \lambda(s, h). \max \{ f(s, h_1) \cdot g(s, h_2) \mid h = h_1 \uplus h_2 \} \]
Quantitative Separating Implication

Classical separating implication:

\[ s, h \models F \rightarrow G \iff \forall h' \text{ with } h \neq h' \text{ and } s, h' \models F : s, h \uplus h' \models G \]
Quantitative Separating Implication

**Classical separating implication:**

\[ s, h \models F \rightarrow G \iff \forall h' \text{ with } h \neq h' \text{ and } s, h' \models F : s, h \cup h' \models G \]

**Quantitative separating implication:**

\[ f \rightarrow g = \lambda(s, h). \inf \left\{ \frac{g(s, h \# h')}{f(s, h')} \mid h \neq h' \text{ and } f(s, h') > 0 \text{ and } \text{not } f(s, h') = \infty = g(s, h \cup h') \right\} \]
Quantitative Separating Implication

Classical separating implication:
\[ s, h \models F \rightarrow G \iff \forall h' \text{ with } h \not= h' \text{ and } s, h' \models F : s, h \uplus h' \models G \]

Quantitative separating implication:
\[ f \rightarrow g = \lambda(s, h). \inf \left\{ \frac{g(s, h \uplus h')}{f(s, h')} \mid h \not= h' \text{ and } f(s, h') > 0 \text{ and not } f(s, h') = \infty = g(s, h \uplus h') \right\} \]

Notice:
\[ [F] \rightarrow g = \lambda(s, h). \inf \left\{ g(s, h \uplus h') \mid h \not= h' \text{ and } s, h' \models F \right\} \]
Properties of $\star$ and $\rightarrow$ 

**Classical adjointness:**

$((F \star G) \Rightarrow J) \iff (F \Rightarrow (G \rightarrow J))$
Properties of $\star$ and $\rightarrow$:

**Classical adjointness:**

\[
((F \star G) \implies J) \iff (F \implies (G \rightarrow J))
\]

**Intuition:**

\[
a - b \leq c \iff a \leq b + c
\]
Properties of $\star$ and $\rightarrow$ $\star$

**Classical adjointness:**

\[
(F \star G) \implies J \iff (F \implies (G \rightarrow J))
\]

**Intuition:**

\[a - b \leq c \iff a \leq b + c\]

**Quantitative adjointness:**

\[f \star g \preceq j \iff f \preceq g \rightarrow j\]
Properties of $\star$ and $\rightarrow^\star$

**Classical adjointness:**

\[
(F \star G) \implies J \iff F \implies (G \rightarrow^\star J)
\]

**Intuition:**

\[
a - b \leq c \iff a \leq b + c
\]

**Quantitative adjointness:**

\[
f \star g \preceq j \iff f \preceq g \rightarrow^\star j
\]

+ virtually all properties of [Ishtiaq & O’Hearn’01, Reynolds’02]
Properties of $\star$ and $\rightarrow$ ⋆

Classical adjointness:

$((F \star G) \Rightarrow J) \iff (F \Rightarrow (G \rightarrow J))$

Intuition:

$a - b \leq c \iff a \leq b + c$

Quantitative adjointness:

$f \star g \preceq j \iff f \preceq g \rightarrow j$

+ virtually all properties of [Ishtiaq & O’Hearn’01, Reynolds’02]

Why virtually? $\text{false} \rightarrow \star F \equiv \text{true} \ vs. \ [\text{false}] \rightarrow \star [F] = \infty \neq [\text{true}]$
Quantitative Separation Logic (QSL) — Overview

2.1. Assertion Language

2.2. Verification System

2.3. Theorems

2.4. Case Studies

2.5. Final remarks
The Weakest Preexpectation Transformer [Kozen’83, McIver’99, McIver & Morgan’05]

Given:

- Probabilistic pointer program $C$
- Postexpectation function $f$

Semantics:

A (sub-)distribution $I_{C:K}(s, h)$ over final states given initial state $(s, h)$

Question:

What is the expected value of $f$ after successful termination of $C$?
The Weakest Preexpectation Transformer [Kozen’83, McIver’99, McIver & Morgan’05]

**Given:** Probabilistic pointer program $C$ and postexpectation $f$

Weakest preexpectation: Mapping from initial state $(s, h)$ to expected value of $f$ evaluated in final states reached after successful termination of $C$ on $(s, h)$:

$\text{wp}\ J\ C\ K( f ) = \lambda (s, h). \int J\ C\ K( s, h ) f \in \{ f \mid f : \text{States} \rightarrow \mathbb{R} \geq 0 \}$
The Weakest Preexpectation Transformer [Kozen’83, McIver’99, McIver & Morgan’05]

**Given:** Probabilistic **pointer** program $C$ and **postexpectation** $f$

**Semantics:** a (sub-)**distribution** $\mathbb{J}[C](s, h)$ over final states given initial state $(s, h)$
The Weakest Preexpectation Transformer [Kozen’83, McIver’99, McIver & Morgan’05]

**Given:** Probabilistic pointer program $C$ and postexpectation $f$

**Semantics:** a (sub-)distribution $\mathbb{J}_C(s, h)$ over final states given initial state $(s, h)$

**Question:** What is the expected value of $f$ after successful termination of $C$?
The Weakest Preexpectation Transformer [Kozen’83, Mclver’99, Mclver & Morgan’05]

**Given:** Probabilistic pointer program $C$ and postexpectation $f$

**Semantics:** a (sub-)distribution $\mathbb{P}[C](s, h)$ over final states given initial state $(s, h)$

**Question:** What is the expected value of $f$ after successful termination of $C$?

![Diagram](image_url)
The Weakest Preexpectation Transformer [Kozen’83, McIver’99, McIver & Morgan’05]

**Given:** Probabilistic pointer program $C$ and postexpectation $f$

**Semantics:** a (sub-)distribution $\mathbb{[C]}(s, h)$ over final states given initial state $(s, h)$

**Question:** What is the expected value of $f$ after successful termination of $C$?
The Weakest Preexpectation Transformer [Kozen’83, McIver’99, McIver & Morgan’05]

**Given:** Probabilistic pointer program \( C \) and postexpectation \( f \)

**Semantics:** a (sub-)distribution \([C](s, h)\) over final states given initial state \((s, h)\)

**Question:** What is the expected value of \( f \) after successful termination of \( C \)?

### Weakest preexpectation:
Mapping from initial state \((s, h)\) to expected value of \( f \) evaluated in final states reached after successful termination of \( C \) on \((s, h)\):

\[
\text{wp} \ [C] (f) = \lambda(s, h). \int_{[C](s, h)} f \quad \in \{ f \mid f : \text{States} \to \mathbb{R}_{\geq 0} \}
\]
### Examples of Quantitative Specifications

<table>
<thead>
<tr>
<th>Postexpectation $f$</th>
<th>Weakest Preexpectation $\wp \left<a href="f"> c \right</a>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda(s, \hat{h}). 1$</td>
<td>Probability of memory-safe termination</td>
</tr>
</tbody>
</table>
# Examples of Quantitative Specifications

<table>
<thead>
<tr>
<th>Postexpectation $f$</th>
<th>Weakest Preexpectation $\wp [c] (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda(s, h). 1$</td>
<td>Probability of memory-safe termination</td>
</tr>
<tr>
<td>$</td>
<td>x</td>
</tr>
</tbody>
</table>
### Examples of Quantitative Specifications

<table>
<thead>
<tr>
<th>Postexpectation $f$</th>
<th>Weakest Preexpectation $\wp \sqsubseteq (c) (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda(s, h). 1$</td>
<td>Probability of memory-safe termination</td>
</tr>
<tr>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td>$\text{[emp]}$</td>
<td>Probability of terminating with an empty heap</td>
</tr>
</tbody>
</table>
## Examples of Quantitative Specifications

<table>
<thead>
<tr>
<th>Postexpectation $f$</th>
<th>Weakest Preexpectation $\text{wp} \ [c] (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda(s, h). 1$</td>
<td>Probability of memory-safe termination</td>
</tr>
<tr>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td>$\text{emp}$</td>
<td>Probability of terminating with an empty heap</td>
</tr>
<tr>
<td>$\text{tree}(x) \cdot \text{size}$</td>
<td>Expected size of a tree with root $x$</td>
</tr>
</tbody>
</table>
Weakest Preexpectation of Memory Allocation

<table>
<thead>
<tr>
<th>s:</th>
<th>h:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:</td>
<td>α:</td>
</tr>
<tr>
<td>y:</td>
<td>β:</td>
</tr>
<tr>
<td>...</td>
<td>γ:</td>
</tr>
<tr>
<td>u</td>
<td>a</td>
</tr>
<tr>
<td>w</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c</td>
</tr>
</tbody>
</table>

... unbounded nondeterminism!
Weakest Preexpectation of Memory Allocation

\[ x := \text{new}(E) \]

\[
\begin{array}{c}
\text{\(s:\)} \\
x: \quad y: \\
u \\
\text{\(h:\)} \\
\text{\(\alpha:\)} \\
\beta: \\
a \\
\gamma:\ \\
b \\
\ldots \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(s:\) } x/1 \\
\text{\(h:\) } \rightarrow s(E) \\
\end{array}
\]

\[
\begin{array}{c}
\text{\(s:\) } x/2 \\
\text{\(h:\) } \rightarrow s(E) \cup \{2 \mapsto s(E)\} \\
\end{array}
\]
Weakest Preexpectation of Memory Allocation

\[
x := \text{new}(E)
\]
Weakest Preexpectation of Memory Allocation

\[ \langle x := \text{new} \left( E \right), s, h \rangle \]

\[ \langle s[x/1], h \uplus \{1 \mapsto s(E)\} \rangle \]

\[ v = 1 \]
Weakest Preexpectation of Memory Allocation

\[ x := \text{new}(E) \]

\[ s : \]
\[
\begin{array}{cccc}
  x & y & \ldots & \\
  u & w & \alpha & \beta & \gamma & \ldots \\
\end{array}
\]

\[ h : \]
\[
\begin{array}{cccc}
  a & b & c & \ldots \\
\end{array}
\]

\[ s_1 : \]
\[
\begin{array}{cccc}
  x & y & \ldots & \\
  v & w & \alpha & \beta & \gamma & \ldots \\
\end{array}
\]

\[ h_1 : \]
\[
\begin{array}{cccc}
  a & b & c & \ldots \\
\end{array}
\]

\[ \cup \]
\[
\begin{array}{cccc}
  v & \subseteq & s(E) \\
\end{array}
\]

\[ v = 1 \]
\[
\langle s[x/1], h \cup \{1 \mapsto s(E)\} \rangle
\]

\[ v = 2 \]
\[
\langle x := \text{new}(E), s, h \rangle \xrightarrow{v = 2} \langle s[x/2], h \cup \{2 \mapsto s(E)\} \rangle
\]
Weakest Preexpectation of Memory Allocation

\[ x := \text{new}(E) \]

<table>
<thead>
<tr>
<th>s:</th>
<th>h:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:</td>
<td>y:</td>
</tr>
<tr>
<td>u:</td>
<td>w:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s':</th>
<th>h':</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:</td>
<td>y:</td>
</tr>
<tr>
<td>\alpha:</td>
<td>\beta:</td>
</tr>
<tr>
<td>a:</td>
<td>b:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\cup</th>
<th>v:</th>
</tr>
</thead>
<tbody>
<tr>
<td>v:</td>
<td></td>
</tr>
<tr>
<td>\cup</td>
<td></td>
</tr>
<tr>
<td>s(E)</td>
<td></td>
</tr>
</tbody>
</table>

\[
\langle s[x/1], h \cup \{1 \mapsto s(E)\} \rangle
\]

\[
\langle x := \text{new}(E), s, h \rangle \quad \overset{v = 2}{\longrightarrow} \quad \langle s[x/2], h \cup \{2 \mapsto s(E)\} \rangle
\]

\[ : \quad \text{unbounded nondeterminism!} \]
Weakest Preexpectation of Memory Allocation

\[ x := \text{new}(E) \]

\[ s : x : y : \ldots \]
\[ u : w \]

\[ h : \]
\[ \alpha : \beta : \gamma : \ldots \]
\[ a : b : c \]

\[ s_s : \]
\[ h_s : \]
\[ \cup \]
\[ v : \]
\[ v \in N \]
\[ v \mapsto \text{E} \]

\[ \text{wp} [x := \text{new}(E)] \]

\[ f \]

\[ v = 1 \]
\[ \langle s[x/1], h \cup \{1 \mapsto s(E)\} \rangle \]

\[ v = 2 \]
\[ \langle s[x/2], h \cup \{2 \mapsto s(E)\} \rangle \]

\[ \vdots \]
unbounded nondeterminism!
Weakest Preexpectation of Memory Allocation

\[ x := \text{new}(E) \]

\[ [v \mapsto E] \xrightarrow{f[x/v]} \]

wp \[ x := \text{new}(E) \]

\[ v = 1 \]

\[ \langle s[x/1], h \cup \{1 \mapsto s(E)\} \rangle \]

\[ v = 2 \]

\[ \langle s[x/2], h \cup \{2 \mapsto s(E)\} \rangle \]

\[ \vdots \]

unbounded nondeterminism!
Weakest Preexpectation of Memory Allocation

\[ x := \text{new}(E) \]

\[ \inf_{v \in \mathbb{N}} [v \mapsto E] \rightarrow f[x/v] \]

\[ \text{wp} [x := \text{new}(E)] \]

\[ \langle s[x/1], h \cup \{1 \mapsto s(E)\} \rangle \]

\[ \langle x := \text{new}(E), s, h \rangle \rightarrow_{v = 2} \langle s[x/2], h \cup \{2 \mapsto s(E)\} \rangle \]

\[ \vdots \quad \text{unbounded nondeterminism!} \]
The Weakest Preexpectation Calculus for Quantitative Separation Logic

\[
\begin{align*}
C & \quad \text{wp} \left[ C \right] (f) \\
\end{align*}
\]

\[
x := \text{new} (E) \\
\inf_{v \in \mathbb{N}} \left[ v \mapsto E \right] \star f \left[ x / v \right]
\]
The Weakest Preexpectation Calculus for Quantitative Separation Logic

\[
\begin{align*}
C & \quad \text{wp} \left[ C \right] (f) \\
\{ C_1 \} [p] \{ C_2 \} & \quad p \cdot \text{wp} \left[ C_1 \right] (f) + (1 - p) \cdot \text{wp} \left[ C_2 \right] (f) \\
x := \text{new} (E) & \quad \text{inf} \left[ v \mapsto E \right] \overset{*}{\rightarrow} f [x/v] \\
\end{align*}
\]

\[x := \text{new} (E)\]
The Weakest Preexpectation Calculus for Quantitative Separation Logic

\[ C \quad \text{wp} \left[ C \right] (f) \]

\[
\{ C_1 \} [p] \{ C_2 \} \\
x := \text{new}(E) \\
x := \langle E \rangle \\
\langle E \rangle := E' \\
\text{free}(E)
\]

\[
p \cdot \text{wp} \left[ C_1 \right] (f) + (1 - p) \cdot \text{wp} \left[ C_2 \right] (f)
\]

\[
\inf_{v \in \mathbb{N}} [v \mapsto E] \star f [x/v] \\
\sup_{v \in \mathbb{Z}} [E \mapsto v] \star ([E \mapsto v] \star f [x/v]) \\
[E \mapsto -] \star ([E \mapsto E'] \star f) \\
[E \mapsto -] \star f
\]
## The Weakest Preexpectation Calculus for Quantitative Separation Logic

<table>
<thead>
<tr>
<th>$C$</th>
<th>$\text{wp} \left[ C \right] (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>$f$</td>
</tr>
<tr>
<td>$x := E$</td>
<td>$f \left[ x/E \right]$</td>
</tr>
<tr>
<td>$C_1; C_2$</td>
<td>$\text{wp} \left[ C_1 \right] \left( \text{wp} \left[ C_2 \right] (f) \right)$</td>
</tr>
<tr>
<td>if $(B) { C_1 } \text{ else } { C_2 }$</td>
<td>$[B] \cdot \text{wp} \left[ C_1 \right] (f) + [\neg B] \cdot \text{wp} \left[ C_2 \right] (f)$</td>
</tr>
<tr>
<td>while $(B) { C' }$</td>
<td>$\text{lfp } X. \ [\neg B] \cdot f + [B] \cdot \text{wp} \left[ C' \right] (X)$</td>
</tr>
<tr>
<td>${ C_1 } [p] { C_2 }$</td>
<td>$p \cdot \text{wp} \left[ C_1 \right] (f) + (1 - p) \cdot \text{wp} \left[ C_2 \right] (f)$</td>
</tr>
<tr>
<td>$x := \text{new}(E)$</td>
<td>$\inf_{v \in \mathbb{N}} [v \mapsto E] \rightarrow f \left[ x/v \right]$</td>
</tr>
<tr>
<td>$x := \langle E \rangle$</td>
<td>$\sup_{v \in \mathbb{Z}} [E \mapsto v] \star ([E \mapsto v] \rightarrow f \left[ x/v \right])$</td>
</tr>
<tr>
<td>$\langle E \rangle := E'$</td>
<td>$[E \mapsto -] \star ([E \mapsto E'] \rightarrow f)$</td>
</tr>
<tr>
<td>free($E$)</td>
<td>$[E \mapsto -] \star f$</td>
</tr>
</tbody>
</table>

+ probabilistic assignment
+ recursive procedures
Quantitative Separation Logic (QSL) — Overview

2.1. Assertion Language

2.2. Verification System

2.3. Theorems

2.4. Case Studies

2.5. Final remarks
QSL is a conservative extension of both SL and weakest preexpectations.
**Theorem I: Conservativity**

QSL is a **conservative extension** of both SL and weakest preexpectations.

$F, G$ SL formulas \hspace{2cm} $[F], [G]$ corresponding expectations
Theorem I: Conservativity

QSL is a conservative extension of both SL and weakest preexpectations.

For all states \((s, h)\) and all non-probabilistic pointer programs:

1) \(s, h \models F \iff \llbracket F \rrbracket(s, h) = 1\)
Theorem I: Conservativity

QSL is a conservative extension of both SL and weakest preexpectations.

\[ F, G \text{ SL formulas} \quad [F], [G] \text{ corresponding expectations} \]

For all states \((s, h)\) and all non-probabilistic pointer programs:

1) \(s, h \models F \iff [F](s, h) = 1\)

2) \(\langle F \rangle C \langle G \rangle \text{ valid} \iff [F] \preceq \wp [C] ([G])\)
Theorem II: Soundness

For all programs $C$, expectations $f$ and states $(s, h)$,
Theorem II: Soundness

For all programs $C$, expectations $f$ and states $(s, h)$,
Theorem II: Soundness

For all programs $C$, expectations $f$ and states $(s, h)$,
Theorem II: Soundness

For all programs $C$, expectations $f$ and states $(s, h)$,

$$wp[C](f)(s, h) = \text{ExpRew}[f](\langle C, s, h \rangle \models \Diamond \lozenge)$$

of operational Markov Decision Process.
The classical frame rule:

\[
\text{For all SL formulas } F, \; G, \; R \text{ with } \text{Mod}(C) \cap \text{Vars}(R) = \emptyset, \quad \langle F \rangle C \langle G \rangle = \langle F \star R \rangle C \langle G \star R \rangle.
\]
The classical frame rule:

For all SL formulas $F$, $G$, $R$ with $\text{Mod}(C) \cap \text{Vars}(R) = \emptyset$,

$$\langle F \rangle_C \langle G \rangle_C \quad \text{and} \quad \langle F \star R \rangle_C \langle G \star R \rangle_C.$$ 

For $F = \text{wp}[C](G)$, this is equivalent to

$$\text{wp}[C](G) \star R \Rightarrow \text{wp}[C](G \star R).$$
Theorem III: The Quantitative Frame Rule

The classical frame rule:

For all SL formulas $F, G, R$ with $\text{Mod}(C) \cap \text{Vars}(R) = \emptyset$, $\langle F \rangle_C \langle G \rangle = \emptyset$,

$$\langle F \star R \rangle_C \langle G \star R \rangle.$$ 

For $F = \text{wp}[[C]](G)$, this is equivalent to $\text{wp}[[C]](G \star R) \Rightarrow \text{wp}[[C]](G \star R)$.

The quantitative frame rule:

For all expectations $g, r$ with $\text{Mod}(C) \cap \text{Vars}(r) = \emptyset$, $\text{wp}[[C]](g \star r) \leq \text{wp}[[C]](g \star r)$.
Quantitative Separation Logic (QSL) — Overview

2.1. Assertion Language

2.2. Verification System

2.3. Theorems

2.4. Case Studies

2.5. Final remarks
Case Study I: Faulty Garbage Collector

```plaintext
// [tree(x)] \cdot 0.5^{\text{size}/2}

delete(x) {
    fail := flipCoin() ;
    if (x \neq 0 and fail = \text{heads}) {
        left := \langle x \rangle ; right := \langle x + 1 \rangle ;
        delete(left) ;
        delete(right) ; // apply frame rule twice
        free(x); free(x + 1)
    }
}

// [emp]
```
Case Study II: Array Randomization

// $1/n! \cdot \max\{[\text{array} \mapsto \alpha_{\pi(0)}, \ldots, \alpha_{\pi(n-1)}] \mid \pi \text{ permutation}\}$

randomize(array, n) {
    \( i := 0; \)
    \[ \text{while} (0 \leq i < n) \{ \]
        \( j := \text{uniform}(i, n-1); \)
        \( \text{swap}(\text{array}, i, j); \)
        \( i := i + 1 \)
    \}\n
// $[\text{array} \mapsto \alpha_0, \ldots, \alpha_{n-1}]$

“many randomized algorithms randomize by permuting the given input”
Case Study III: Randomized Meldable Heaps

// [root \neq 0] \cdot \text{[tree(root)]} \cdot \log(1 + \frac{1}{2} \cdot \text{size})
randomLeaf(root){
    nextL := \langle root \rangle;
    nextR := \langle root + 1 \rangle;
    if (nextL = 0 and nextR = 0) {
        return root
    } else {
        \{ next := nextL \} [0.5] \{ next := nextR \};
    return randomLeaf(root)
}

// \text{[tree(root)]} \cdot (1 \ast \text{[path(root, result)]}) \cdot \frac{1}{2} \cdot \text{size}
Quantitative Separation Logic (QSL) — Overview

2.1. Assertion Language

2.2. Verification System

2.3. Theorems

2.4. Case Studies

2.5. Final remarks
Mechanizing QSL in

The assertion language of QSL has been certified in Isabelle/HOL\(^1\)
- expectations
- quantitative separating connectives
- algebraic properties
- embedding of SL into QSL
- ...
- \(\approx 2000\) LOC

No errors were found; proofs could almost be taken one-to-one

Next step: Mechanization of QSL’s weakest preexpectations
- Main challenge: \textit{unbounded nondeterminism} (memory allocation)

\(^1\)courtesy Max Haslbeck. https://github.com/maxhaslbeck/QuantSepCon
Wrap-up

1. QSL combines discrete probabilities with pointers

2. QSL mixes probabilistic choice and unbounded nondeterminism

3. QSL preserves virtually all properties of both:
   - classical separation logic à la [Ishtiaq & O’Hearn’01, Reynolds’02]
   - weakest preexpectations à la [Kozen’83, McIver & Morgan’05]

4. Elementary properties certified in Isabelle/HOL

5. QSL has been applied to verify actual randomized algorithms

Related work:
- Verification of randomized algorithms: [LICS’16], [Chatterjee et al.’17], [Eberl et al.’18], [Tassarotti & Harper’18], [Tassarotti & Harper’19]
- Quantitative aspects of separation logic: [Bozga et al.’10], [Atkey’11], [Chin et al.’11]

Further reading: Part II, [POPL’19]
3. Automated Reasoning with/about Separation Logic

“Use separation logic predicates for abstraction”
\begin{align*}
x & := \textit{head}; \\
\text{while}(x.next \neq 0) \{ \\
\quad x & := x.next \\
\}
\end{align*}

\[\text{sll}(x, y) = x \mapsto y \]
\[\lor \exists z : x \mapsto z \ast \text{sll}(z, y) \]
\[\lor \exists z : \text{sll}(x, z) \ast \text{sll}(z, y) \]

\begin{align*}
G \{ \text{sll}(\textit{head}, 0) \} \\
F \{ \text{visited}(x) \} \\
G \{ \{ \text{terminated} \} \rightarrow \{ \text{emp} \} \}
Attestor's State Space Generation Loop
Attestor’s State Space Generation Loop

\[ \iff \exists y : y \mapsto 0 \land x = y \]
\[ \exists y : \text{sll}(y, 0) \land x = y \]
\[ \iff \exists y, z : y \mapsto z \# \text{sll}(z, 0) \land x = y \]
Attestor’s State Space Generation Loop

\[ \iff \exists y : y \mapsto 0 \land x = y \]

\[ \exists y : \text{sll}(y, 0) \land x = y \]

\[ \iff \exists y, z : y \mapsto z \ast \text{sll}(z, 0) \land x = y \]

\[ \exists y, z : y \mapsto z \ast \text{sll}(z, 0) \land x = y \]

\[ x := x\text{.next} \]

\[ \exists y, z : y \mapsto z \ast \text{sll}(z, 0) \land x = z \]
Attestor’s State Space Generation Loop

\[ \exists y: y \mapsto 0 \land x = y \]
\[ \exists y: \text{sll}(y, 0) \land x = y \]
\[ \exists y, z: y \mapsto z \star \text{sll}(z, 0) \land x = y \]

\[ \exists y, z: y \mapsto z \star \text{sll}(z, 0) \land x = z \]

unique for confluent graphical SIDs

\[ x := x \text{.next} \]

\[ \exists y, z: y \mapsto z \star \text{sll}(z, 0) \land x = z \]

decidable using heap automata

add predicate to state space unless it entails an existing one

decidable in \(\text{NP} \) for considered fragment
Attestor’s State Space Generation Loop

\[ \Longleftarrow \exists y: y \mapsto 0 \land x = y \]
\[ \exists y: \text{sll}(y, 0) \land x = y \]
\[ \Longleftarrow \exists y, z: y \mapsto z \star \text{sll}(z, 0) \land x = y \]

∀y, z: \text{sll}(y, z) \star \text{sll}(z, 0) \land x = z \implies \{ \text{reach}(x, 0) \}

unique for confluent graphical SIDs

decidable using heap automata
Attestor’s State Space Generation Loop

\[
\exists y: y \mapsto 0 \land x = y \\
\exists y: \text{sl}(y, 0) \land x = y \\
\exists y, z: y \mapsto z \ast \text{sl}(z, 0) \land x = y
\]

add predicate to state space unless it entails an existing one

decidable in NP for considered fragment

\[
\exists y, z: \text{sl}(y, z) \ast \text{sl}(z, 0) \land x = z \models \{ \text{reach}(x, 0) \}
\]
decidable using heap automata

unique for confluent graphical SIDs
Attestor’s Output
Attestor’s Output
Attestor’s Output

State Space \models \text{Spec}\? \quad \Rightarrow \quad \text{satisfied} \quad \land \quad \text{violated} \quad + \quad \text{realizable counterexample} \quad \lor \quad \text{unknown}
Wrap-up

1. **Attestor** enables automated verification of (non-probabilistic) pointer programs

2. Support for intricate **temporal specifications**

3. **Heap automata** enable reasoning about **robustness** of certain separation logic predicates

4. A simple **decision procedure** for **entailments** between certain “graphical” symbolic heaps

5. Additional support for analyzing lengths, balancedness, etc. [SEFM’18]

**Related work:**
- Fragments with user-supplied SIDs: [Brotherston et al.’11], [Iosif et al.’13], [Iosif et al.’14], [Brotherston et al.’14], [Brotherston et al.’16], [Iosif & Serban’18]
- Verification tools: [Berdine et al.’05] (Smallfoot), [Calcagno et al.’11] (Infer), [Chin et al.’11] (HIP/Sleek), [Holik et al.’13] (Forester), [Heinen et al.’15] (Juggrnaut), …

**Further reading:** Part III, [ESOP’17], [CAV’18], https://github.com/moves-rwth/attestor
4. Epilogue
Conclusion: What’s in the thesis?

1. Primer on Separation Logic
   - Hoare logic and weakest preconditions
   - Reasoning about recursion
   - $x \mapsto y$, $\ast$, $\rightarrow$, and local reasoning

2. Reasoning about Prob. Pointer Programs
   - Quantitative Separation Logic
   - Sound extension that preserves virtually all proof rules
   - Case studies including 3 randomized algorithms

3. Automated Reasoning
   - Graphical symbolic heaps
   - Decision procedures for robustness and entailment
   - Attestor
Future Work

- Semi-automated reasoning about probabilistic pointer programs
- Expected runtimes as a resource for separation logic
- Underapproximation in verification (see [O'Hearn'2020])
- Frame rule for upper bounds
- Learning confluent data structure specifications
- Test case generation based on Attestor's counterexamples
- Extend support for data theories in Attestor
Future Work

- Semi-automated reasoning about probabilistic pointer programs
- Expected runtimes as a resource for separation logic
- Underapproximation in verification (see [O'Hearn'2020])
- Frame rule for upper bounds
- Learning confluent data structure specifications
- Test case generation based on Attestor's counterexamples
- Extend support for data theories in Attestor

Thank you for listening!