Quantitative Separation Logic

Weakest Preexpectation Reasoning for Probabilistic Pointer Programs

Christoph Matheja

November 25th, ETH Zürich
Why Probabilistic Pointer Programs?

Probabilistic programs: “Ordinary” program + sampling from discrete probability distributions

\[
\begin{align*}
x &:= 1; y := 1 \\
\text{while}(x = 1)\{ \\
&\quad \{ y := 2 \cdot y \} [1/2] \{ x := 0 \} \\
\}\end{align*}
\]
Why Probabilistic Pointer Programs?

Probabilistic programs: “Ordinary” program + sampling from discrete probability distributions

```
x := 1; y := 1
while (x = 1) {
    { y := 2 · y }^{\frac{1}{2}} { x := 0 }
}
```

1. does not always terminate
2. terminates almost-surely
3. expected runtime: constant

Back in 2016:
Analysis of expected runtimes [ESOP'16, LICS'16, J.ACM'18]

But:
The first chapter 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]

[Motwani & Raghavan'95]

Goal: A separation logic for formal verification of probabilistic pointer programs
Why Probabilistic Pointer Programs?

Probabilistic programs: “Ordinary” program + sampling from discrete probability distributions

```
x := 1; y := 1
while (x = 1)
    { y := 2 \cdot y } [1/2] { x := 0 }
```

1. does not always terminate
2. terminates almost-surely
3. expected runtime: constant

Back in 2016: Analysis of expected runtimes

[ESOP’16, LICS’16, J.ACM’18]
Why Probabilistic Pointer Programs?

Probabilistic programs: “Ordinary” program + sampling from discrete probability distributions

\[
x := 1; \ y := 1
\]

\[
\text{while}(x = 1)\{ \\
\quad \{ y := 2 \cdot y \}^{[1/2]} \{ x := 0 \}
\}
\]

1. does not always terminate
2. terminates almost-surely
3. expected runtime: constant

Back in 2016: Analysis of expected runtimes [ESOP’16, LICS’16, J.ACM’18]
Why Probabilistic Pointer Programs?

Probabilistic programs: “Ordinary” program + sampling from discrete probability distributions

\[
x := 1; y := 1 \\
\text{while} (x = 1) \{ \\
\quad \{ y := 2 \cdot y \}^{1/2} \{ x := 0 \} \\
\}
\]

1. does not always terminate
2. terminates almost-surely
3. expected runtime: constant

Back in 2016: Analysis of expected runtimes [ESOP'16, LICS'16, J.ACM'18]

But: The first chapter 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]

[Motwani & Raghavan’95]
Why Probabilistic Pointer Programs?

Probabilistic programs: “Ordinary” program + sampling from discrete probability distributions

\[
x := 1; y := 1
\]

while \((x = 1)\) {
  \[
  \{ y := 2 \cdot y \} \left[ \frac{1}{2} \right] \{ x := 0 \}
  \]
}

1. does not always terminate
2. terminates almost-surely
3. expected runtime: constant

Back in 2016: Analysis of expected runtimes [ESOP’16, LICS’16, J.ACM’18]

But: The first chapter 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]

Goal: A separation logic for formal verification of probabilistic pointer programs

[Motwani & Raghavan’95]
Applications: Randomized Algorithms

randomLeaf(root) {
    nextL := ⟨root⟩; // dereference root
    nextR := ⟨root + 1⟩;
    if (nextL = 0 and nextR = 0) {
        return root
    } else {
        next := nextL [0.5] {next := nextR};
        return randomLeaf(root)
    }
}

Randomized Meldable Priority Queues

Anna Gambin and Adam Malinowski
Instytut Informatyki, Uniwersytet Warszawski,
Banańska 2, Warszawa 02-097, Poland,
{aniag, amal}@impani.edu.pl

Abstract. We present a practical meldable priority queue implementation. All priority queue operations are very simple and their logarithmic time bound holds with high probability, which makes this data structure more suitable for real-time applications than those with only amortized performance guarantees. Our solution is also space-efficient, since it does not require storing any auxiliary information within the queue nodes.

1 Introduction

In this paper we present a randomized approach to the problem of efficient meldable priority queue implementation. The operations supported by this data structure are the following [10]:

MAKEQUEUE returns an empty priority queue.  
FINDMIN(Q) returns the minimum item from priority queue Q.

[Gambin & Malinowski, 1998]

Theorem 1: The expected length of a random walk in a binary tree with n nodes is at most log(n + 1).
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...
null pointer dereferences, memory leaks, **unbounded nondeterminism**, ...
Challenges

Probabilities = Problematic for Mathematicians

“In no other branch of mathematics is it so easy to make mistakes as in probability theory” [Tijms’04, Understanding Probability]
Probabilities = Problematic for Mathematicians

“In no other branch of mathematics is it so easy to make mistakes as in probability theory” [Tijms’04, Understanding Probability]

Example:

\[ C_1: \quad \begin{array}{l} x := 1; y := 1 \\
\text{while}(x = 1)\{ \\
\quad \{y := 2 \cdot y \}^{1/2} \{x := 0\} \\
\} \end{array} \]

Expected runtime
finite (constant)
Challenges

Probabilities = Problematic for Mathematicians

“In no other branch of mathematics is it so easy to make mistakes as in probability theory” [Tijms’04, Understanding Probability]

Example:

\[
C_1: \quad x := 1; y := 1 \\
\text{while}(x = 1)\{ \\
\quad \{ y := 2 \cdot y \}^{1/2} \{ x := 0 \} \\
\}
\]

\[
C_2: \quad \text{while}(y > 0)\{ y := y - 1 \}
\]

Expected runtime

\[
C_1 \quad \text{finite (constant)} \\
C_2 \quad \text{finite (linear in } y) \\
\]
Challenges

Probabilities $\Rightarrow$ Problematic for Mathematicians

“In no other branch of mathematics is it so easy to make mistakes as in probability theory” [Tijms'04, Understanding Probability]

Example:

$C_1$: \[x := 1; y := 1\]

\[\text{while}\ (x = 1)\{\]

\[\{ y := 2 \cdot y \}^{1/2} \{ x := 0 \}\]

\[
\}

$C_2$: \[\text{while}\ (y > 0)\{ y := y - 1 \}\]

Expected runtime

$C_1$: finite (constant)

$C_2$: finite (linear in $y$)

$C_1; C_2$: $\infty$
Challenges

Probabilities = Problematic for Mathematicians

“In no other branch of mathematics is it so easy to make mistakes as in probability theory” \[\text{Tijms'04, Understanding Probability}\]

Example:

\[C_1: \quad x := 1; y := 1 \quad \text{while} \ (x = 1) \{ \quad \{ y := 2 \cdot y \}^{1/2} \{ x := 0 \} \} \]

\[C_2: \quad \text{while} \ (y > 0) \{ y := y - 1 \} \]

Expected runtime

\[C_1 \quad \text{finite (constant)} \]
\[C_2 \quad \text{finite (linear in } y) \]
\[C_1; C_2 \quad \infty \]

Proving that the expected runtime is finite is \textbf{strictly harder} than the halting problem! \[\text{Kaminski'15, Acta'18}\]
Quantitative Separation Logic — Overview

1. Introduction

2. Assertion Language

3. Verification System

4. Theorems

5. Case Studies

6. Epilogue
Expectations: “Instead of truth, we measure a quantity”

\[ \text{States} = \{ (s, h) \mid s : \text{Vars} \to \mathbb{Z}, \quad h : \underbrace{\text{dom}(h)}_{\subset \mathbb{N}\setminus\{0\} \text{ finite}} \to \mathbb{Z} \} \]

Expectations: \[ f : \text{States} \to \mathbb{R}^{\infty}_{\geq 0} \]
Expectations: “Instead of truth, we measure a quantity”

\[
States = \{(s, h) \mid s: \text{Vars} \to \mathbb{Z}, \quad h: \text{dom}(h) \to \mathbb{Z} \}
\]

\[
\subseteq \mathbb{N}\setminus\{0\} \text{ finite}
\]

Expectations: \[ f: States \to \mathbb{R}_{\geq 0} \]

Examples: \[ x^2 = \lambda(s, h). s(x)^2 \]
Expectations: “Instead of truth, we measure a quantity”

\[
States = \{(s, h) \mid s: \text{Vars} \rightarrow \mathbb{Z}, \quad h: \text{dom}(h) \rightarrow \mathbb{Z} \}
\]

\[
\subseteq \mathbb{N}\setminus\{0\} \quad \text{finite}
\]

Expectations: \( f: States \rightarrow \mathbb{R}_{\geq 0} \)

Examples: \( x^2 = \lambda(s, h). s(x)^2 \quad \text{size} = \lambda(s, h). |\text{dom}(h)| \)
Expectations: “Instead of truth, we measure a quantity”

\[ States = \{(s, h) \mid s: \text{Vars} \rightarrow \mathbb{Z}, \quad h: \text{dom}(h) \rightarrow \mathbb{Z}\} \]

Expectations: \( f: States \rightarrow \mathbb{R}_{\geq 0} \)

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

\[ [\text{emp}] = \lambda(s, h). \begin{cases} 1 & \text{if } \text{dom}(h) = \emptyset \\ 0 & \text{otherwise} \end{cases} \]
Expectations: “Instead of truth, we measure a quantity”

\[
States = \{(s, h) \mid s: Vars \rightarrow \mathbb{Z}, \ h: \ \text{dom}(h) \rightarrow \mathbb{Z}\}
\]

\[
\subseteq \mathbb{N}\setminus\{0\} \text{ finite}
\]

Expectations: \( f: States \rightarrow \mathbb{R}_{\geq 0} \)

Examples:

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

\[
\begin{align*}
\text{[emp]} &= \lambda(s, h). \begin{cases} 
1 & \text{if } \text{dom}(h) = \emptyset \\
0 & \text{otherwise}
\end{cases} \\
\text{[x} \mapsto y] &= \lambda(s, h). \begin{cases} 
1 & \text{if } s, h \models x \mapsto y \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]
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:

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

Classical separating conjunction:

\[(s, h) \models F \ast G \iff \exists h_1, h_2: h = h_1 \ast h_2 \text{ and } (s, h_1) \models F \text{ and } (s, h_2) \models G\]
Separating Conjunction

Classical separating conjunction:

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

Classical separating conjunction:

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

Quantitative separating conjunction:

\[f \star g = \lambda(s, h). \max \{ f(s, h_1) \cdot g(s, h_2) \mid h = h_1 \star h_2 \}\]
Adjointness of $\star$ and $\rightarrow$:

\[
(F \star G) \Rightarrow J \iff F \Rightarrow (G \star J)
\]

\[
f \star g \leq j \iff f \leq g \star j
\]
Adjointness of $\star$ and $\rightarrow\star$:

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

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

Quantitative magic wand:

$$ [F] \rightarrow\star g = \lambda(s, h). \inf \{ g(s, h \star h') \mid h' \perp h \text{ and } (s, h') \models F \} $$
Magic Wand & Properties

Adjointness of $\star$ and $\Rightarrow$:

\[(F \star G) \Rightarrow J \iff F \Rightarrow (G \star J)\]

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

Quantitative magic wand:

\[[F] \rightarrow \star g = \lambda(s, h). \inf \{ g(s, h \star h') \mid h' \perp h \text{ and } (s, h') \models F \}\]

\[f \rightarrow \star g = \lambda(s, h). \inf \left\{ \frac{g(s, h \star h')}{f(s, h')} \mid h' \perp h \text{ and } f(s, h') > 0 \text{ and } \neg f(s, h') = \infty = g(s, h \star h') \right\}\]
Magic Wand & Properties

Adjointness of $\star$ and $\rightarrow$:

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

$$f \star g \leq j \iff f \leq g \star j$$

Quantitative magic wand:

$$[F] \star g = \lambda(s, h). \inf \{ g(s, h \star h') \mid h' \perp h \text{ and } (s, h') \models F \}$$

$$f \star g = \lambda(s, h). \inf \left\{ \frac{g(s, h \star h')}{f(s, h')} \mid h' \perp h \text{ and } f(s, h') > 0 \text{ and } \not f(s, h') = \infty = g(s, h \star h') \right\}$$

+ virtually all properties of [Ishtiaq & O’Hearn’01, Reynolds’02]
Quantitative Separation Logic — Overview

1. Introduction

2. Assertion Language

3. Verification System

4. Theorems

5. Case Studies

6. Epilogue
The Weakest Preexpectation Transformer [Kozen’83, McLver’99, McLver & Morgan’05]

Given: Probabilistic program $C$ and initial state $(s, h)$

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

**Given:** Probabilistic program $C$ and initial state $(s, h)$

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

**Question:** Given a postexpectation $f$, what is the expected value of $f$ after successful termination of $C$?
Given: Probabilistic program $C$ and initial state $(s, h)$

Semantics: a (sub-)distribution $\llbracket C \rrbracket(s, h)$ over final states

Question: Given a postexpectation $f$, 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 program** $C$ and initial state $(s, h)$

Semantics: a (sub-)**distribution** $\llbracket C \rrbracket(s, h)$ over final states

**Question:** Given a **postexpectation** $f$, 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 program \( C \) and initial state \((s, h)\)

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

**Question:** Given a postexpectation \( f \), what is the expected value of \( f \) after successful termination of \( C \)?

Call this the **weakest preexpectation** of \( C \) w.r.t. \( f \):

\[
wp [[C]](f) = \lambda(s, h). \int [[C][s, h]] f
\]
## Examples of Quantitative Specifications

<table>
<thead>
<tr>
<th>Postexpectation $f$</th>
<th>Weakest Preexpectation $\wp \llbracket c \rrbracket (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda(s, h). 1$</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 $wp \llbracket c \rrbracket (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda(s, h). 1$</td>
<td>Probability of memory-safe termination</td>
</tr>
<tr>
<td>$\lambda(s, h). 0$</td>
<td>$\lambda(s, h). 0$ (divergence, memory fault, ...)</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>( \lambda(s, h). 0 )</td>
<td>( \lambda(s, h). 0 ) (divergence, memory fault, \ldots)</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 $\text{wp} [c] (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda(s, h). 1$</td>
<td>Probability of memory-safe termination</td>
</tr>
<tr>
<td>$\lambda(s, h). 0$</td>
<td>$\lambda(s, h). 0$ (divergence, memory fault, ...)</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 $wp \llbracket c \rrbracket (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda(s, h). 1$</td>
<td>Probability of memory-safe termination</td>
</tr>
<tr>
<td>$\lambda(s, h). 0$</td>
<td>$\lambda(s, h). 0$ (divergence, memory fault, ...)</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{ls}(x, y)] \cdot \text{size}$</td>
<td>Expected length of list segment from $x$ to $y$</td>
</tr>
</tbody>
</table>
The Weakest Preexpectation Calculus for QSL

\[
\begin{align*}
C & \quad \text{wp} \mathbb{[}C\mathbb{]}(f) \\
\{ C_1 \} [p] \{ C_2 \} & \quad p \cdot \text{wp} \mathbb{[}C_1\mathbb{]}(f) + (1 - p) \cdot \text{wp} \mathbb{[}C_2\mathbb{]}(f)
\end{align*}
\]
The Weakest Preexpectation Calculus for QSL

\[
\begin{align*}
\{ C_1 \} [p] \{ C_2 \} &= p \cdot \text{wp} \, [C_1] (f) + (1 - p) \cdot \text{wp} \, [C_2] (f) \\
x := \text{new}(E) &= \inf_{v \in \mathbb{N}} [E \mapsto v] \ast ([E \mapsto v] \ast f [x \mapsto v]) \\
x := \langle E \rangle &= \sup_{v \in \mathbb{Z}} [E \mapsto v] \ast ([E \mapsto v] \ast f [x \mapsto v]) \\
\langle E \rangle := E' &= [E \mapsto -] \ast ([E \mapsto E'] \ast f) \\
\text{free}(E) &= [E \mapsto -] \ast f
\end{align*}
\]
The Weakest Preexpectation Calculus for QSL

<table>
<thead>
<tr>
<th>$C$</th>
<th>$\text{wp} ; [C] ; (f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>$f$</td>
</tr>
<tr>
<td>$x := E$</td>
<td>$f ; [x/E]$</td>
</tr>
<tr>
<td>$C_1 ; ; ; C_2$</td>
<td>$\text{wp} ; [C_1] ; (\text{wp} ; [C_2] ; (f))$</td>
</tr>
<tr>
<td>$\text{if} ; (B) { ; C_1 } ; \text{else} { ; C_2 }$</td>
<td>$[B] \cdot \text{wp} ; [C_1] ; (f) + [\neg B] \cdot \text{wp} ; [C_2] ; (f)$</td>
</tr>
<tr>
<td>$\text{while} ; (B) { ; C' }$</td>
<td>$\text{lfp} ; X. ; [\neg B] \cdot f + [B] \cdot \text{wp} ; [C'] ; (X)$</td>
</tr>
<tr>
<td>${ ; C_1 } ; [p] ; { ; C_2 }$</td>
<td>$p \cdot \text{wp} ; [C_1] ; (f) + (1 - p) \cdot \text{wp} ; [C_2] ; (f)$</td>
</tr>
<tr>
<td>$x := \text{new} ; (E)$</td>
<td>$\text{inf} ; \nu \in \mathbb{N} ; [\nu \mapsto E] \rightarrow f ; [x/\nu]$</td>
</tr>
<tr>
<td>$x := \langle E \rangle$</td>
<td>$\text{sup} ; \nu \in \mathbb{Z} ; [E \mapsto \nu] \star ([E \mapsto \nu] \rightarrow f ; [x/\nu])$</td>
</tr>
<tr>
<td>$\langle E \rangle := E'$</td>
<td>$[E \mapsto -] \star ([E \mapsto E'] \rightarrow f)$</td>
</tr>
</tbody>
</table>
| free$(E)$                    | $[E \mapsto -] \star f$     | (demonic scheduler)
### The Weakest Preexpectation Calculus for QSL

<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>$f [x/E]$</td>
</tr>
<tr>
<td>$C_1 ; C_2$</td>
<td>$\text{wp} \left[ C_1 \right] (\text{wp} \left[ C_2 \right] (f))$</td>
</tr>
<tr>
<td>if $(B) { C_1 }$ 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>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>$\inf_{v \in \mathbb{N}} [v \mapsto E] \rightarrow f [x/v]$</td>
</tr>
<tr>
<td>$x := \langle E \rangle$</td>
<td>$\sup_{v \in \mathbb{Z}} [E \mapsto v] \star ([E \mapsto v] \rightarrow f [x/v])$</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>

(demonic scheduler)

+ probabilistic assignment

+ procedures
Quantitative Separation Logic — Overview

1. Introduction

2. Assertion Language

3. Verification System

4. Theorems

5. Case Studies

6. Epilogue
Theorem I: Conservativity

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 $f$, $g$ corresponding expectations
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 \(f(s, h) = 1\)
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) \(\{ F \} C \{ G \}\) valid iff \(f \leq \text{wp}[C](g)\)
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 $(s, h)$,

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

of operational Markov Decision Process
Theorem III: The Quantitative Frame Rule

The classical frame rule:

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

$$
\frac{\{ F \} C \{ G \}}{\{ F \ast R \} C \{ G \ast R \}}.
$$
Theorem III: The Quantitative Frame Rule

The classical frame rule:

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

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

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

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

The quantitative frame rule:

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

1. Introduction

2. Assertion Language

3. Verification System

4. Theorems

5. Case Studies

6. Epilogue
Case Study I: Array Shuffle

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

shuffle(array, n)

\[
\begin{align*}
i &:= 0; \\
\text{while}(0 \leq i < n)\{ \\
&\quad j := \text{uniform}(i, n - 1); \\
&\quad \text{swap}(\text{array}, i, j); \\
&\quad i := i + 1
\}
\end{align*}
\]

// \([\text{array } \mapsto \alpha_0, \ldots, \alpha_{n-1}]\)
Case Study II: Randomized Meldable Heaps

// [root ≠ 0] · [tree(root)] · log(1 + 1/2 · size)
randomLeaf (root) {
    nextL := ⟨root⟩;
    nextR := ⟨root + 1⟩;
    if (nextL = 0 and nextR = 0) {
        return root
    } else {
        {next := nextL} [0.5] {next := nextR};
        return randomLeaf (root)
    }
} // [tree(root)] · (1 * [path(root, result)]) · 1/2 · size

Randomized Meldable Priority Queues
Anna Gambin and Adam Malinowski
Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, Warszawa 00-972, Poland;
{faniag,amal}@mimuw.edu.pl

Abstract. We present a practical meldable priority queue implementation. All priority queue operations are very simple and their logarithmic time bound holds with high probability, which makes this data structure more suitable for real-time applications than those with only amortized performance guarantees. Our solution is also space efficient, since it does not require any auxiliary information within the queue nodes.

1 Introduction
In this paper we present a randomized approach to the problem of efficient meldable priority queue implementation. The operations supported by the data structure are the following [11]:

- MakeQueue returns an empty priority queue.
- FindMin(Q) returns the minimum item from priority queue Q.
- DecreaseKey(Q; e; e0) replaces item e by e0 in priority queue Q provided e0 ≤ e and the location of e in Q is known.
- DecreaseKey(Q; e) returns the priority queue formed by combining disjoint priority queues Q and Qe.

[11] [Gambin, Malinowski, 1998]
Quantitative Separation Logic — Overview

1. Introduction

2. Assertion Language

3. Verification System

4. Theorems

5. Case Studies

6. Epilogue
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: *unbounded nondeterminism* (memory allocation)

\(^1\)courtesy Max Haslbeck. https://github.com/maxhaslbeck/QuantSepCon
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 is *applicable* to reason about actual *randomized algorithms*
Future Work and Further Reading

Future:
• underapproximation vs. overapproximation
• frame rule for upper bounds
• automation (quantitative entailments?, PSI?)
• QSL + time credits
• concurrency, continuous distributions, . . .
Future Work and Further Reading

Future:
• underapproximation vs. overapproximation
• frame rule for upper bounds
• automation (quantitative entailments?, PSI?)
• QSL + time credits
• concurrency, continuous distributions, ...
Future Work and Further Reading

Future:
• underapproximation vs. overapproximation
• frame rule for upper bounds
• automation (quantitative entailments?, PSI?)
• QSL + time credits
• concurrency, continuous distributions, ...

Further details:
• POPL 2019
• full version on arXiv
• part II in my PhD thesis (to appear)

A big thanks to my co-authors!
Hannah Arndt, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Thomas Noll
Future Work and Further Reading

Future:
• underapproximation vs. overapproximation
• frame rule for upper bounds
• automation (quantitative entailments?, PSI?)
• QSL + time credits
• concurrency, continuous distributions, . . .

Further details:
• POPL 2019
• full version on arXiv
• part II in my PhD thesis (to appear)

A big thanks to my co-authors!
Hannah Arndt, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Thomas Noll

Thank you for listening!
Backup slides
// [tree(x)] · (1 − p)^{\text{size}/2}

delete(x) {
    if (x ≠ 0) {  // fails with probability p
        { skip } [p] {
            ℓ := ⟨x⟩; r := ⟨x + 1⟩;
            delete(ℓ);
            delete(r);  // apply frame rule twice
            free(x); free(x + 1)
        }
    }
}

// [emp]

**Challenge:** This proof yields a lower bound. Determine the exact probability.
Weakest Preexpectation of Memory Allocation

<table>
<thead>
<tr>
<th>$s$:</th>
<th>$h$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$:</td>
<td>$\alpha$:</td>
</tr>
<tr>
<td>$y$:</td>
<td>$\beta$:</td>
</tr>
<tr>
<td>$\ldots$:</td>
<td>$\gamma$:</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>
<tr>
<td></td>
<td>$\ldots$:</td>
</tr>
</tbody>
</table>
Weakest Preexpectation of Memory Allocation

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

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

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

\[
\langle s\{x \mapsto \text{new}(e)}\rangle
\]

\[
v = 1
\]

\[
v = 2
\]

countable nondeterminism!
Weakest Preexpectation of Memory Allocation

```
x := new(e)
```

![Diagram showing memory allocation and preexpectation]

```
s:  x: y: ...  u  w
h:  α: β: γ: ...  a  b  c

s_1:  x: y: ...  v  w
h_1:  α: β: γ: ...  a  b  c

* v: s(e)
```
Weakest Preexpectation of Memory Allocation

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

\[
\langle x := \text{new}(e), s, h \rangle \quad \langle s[x/1], h \star \{1 \mapsto s(e)\} \rangle
\]

\[
v = 1
\]

\[
\langle s \downarrow: h \leftrightarrow s(e) \rangle
\]
Weakest Preexpectation of Memory Allocation

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

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\hline
s: & x & y & \ldots & \alpha & \beta & \gamma & \ldots \\
\hline
u & w & & & a & b & c & \ldots \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\hline
h: & x & y & \ldots & \alpha & \beta & \gamma & \ldots \\
\hline
\vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\hline
s_1: & x & y & \ldots & \alpha & \beta & \gamma & \ldots \\
\hline
v & w & & & a & b & c & \ldots \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\hline
h_1: & x & y & \ldots & \alpha & \beta & \gamma & \ldots \\
\hline
\vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
\hline
\end{array}
\]

\[ v : s(e) \]

\[ \langle x := \text{new}(e), s, h \rangle \quad v = 2 \quad \langle s[x/2], h \star \{2 \mapsto s(e)\} \rangle \]

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

\[ v = 1 \]

\[ \langle x := \text{new}(e), s, h \rangle \quad v = 2 \quad \langle s[x/2], h \star \{2 \mapsto s(e)\} \rangle \]

\[ \langle s[x/1], h \star \{1 \mapsto s(e)\} \rangle \]
Weakest Preexpectation of Memory Allocation

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

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

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

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

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

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

\[
\begin{align*}
\langle x := \text{new}(e), s, h \rangle & \quad \text{\textbf{\textit{\vspace{1em}}}} \langle s[x/1], h \star \{1 \mapsto s(e)\} \rangle \\
\langle x := \text{new}(e), s, h \rangle & \quad \text{\textbf{\textit{\vspace{1em}}}} \langle s[x/2], h \star \{2 \mapsto s(e)\} \rangle \\
\text{\textbf{\textit{\vspace{1em}}}} & \quad \text{\textbf{\textit{\vspace{1em}}}} \text{\textbf{countable nondeterminism!}}
\end{align*}
\]
Weakest Preexpectation of Memory Allocation

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

\[ \langle x := \text{new}(e), s, h \rangle \quad v = 1 \quad \langle s[x/1], h \star \{1 \mapsto s(e)\} \rangle \]

\[ \langle x := \text{new}(e), s, h \rangle \quad v = 2 \quad \langle s[x/2], h \star \{2 \mapsto s(e)\} \rangle \]

: countable nondeterminism!
Weakest Preexpectation of Memory Allocation

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

\[ \inf_{v \in \mathbb{N}} [v \mapsto e] \to f[x/v] \]

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

\[ \langle x := \text{new}(e), s, h \rangle \quad \begin{cases} v = 1 & \langle s[x/1], h \ast \{1 \mapsto s(e)\} \rangle \\ v = 2 & \langle s[x/2], h \ast \{2 \mapsto s(e)\} \rangle \end{cases} \]

: countable nondeterminism!