Reasoning about Recursive Probabilistic Programs

Federico Olmedo
Joost-Pieter Katoen

Benjamin Kaminski
Christoph Matheja

RWTH Aachen University, Germany

Pr[nobody disturbs] ≥ \(\frac{1}{2}\)

LICS 2016
July 8th — New York City
What is a Probabilistic Program?

Probabilistic program that simulates a geometric distribution

\[ C_{\text{geo}}: \quad n := 0; \]
\[ \text{repeat} \]
\[ \quad n := n + 1; \]
\[ \quad c := \text{coin_flip}(0.5) \]
\[ \text{until } (c = \text{heads}); \]
\[ \text{return } n \]

Program Output Distribution

<table>
<thead>
<tr>
<th>Output</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
</tr>
<tr>
<td>2</td>
<td>1/4</td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
</tr>
<tr>
<td>4</td>
<td>1/16</td>
</tr>
<tr>
<td>5</td>
<td>1/32</td>
</tr>
</tbody>
</table>

Program Runtime

<table>
<thead>
<tr>
<th>Run–Time</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1/2</td>
</tr>
<tr>
<td>5</td>
<td>1/4</td>
</tr>
<tr>
<td>7</td>
<td>1/8</td>
</tr>
<tr>
<td>9</td>
<td>1/16</td>
</tr>
<tr>
<td>11</td>
<td>1/32</td>
</tr>
</tbody>
</table>

Average (or Expected) Runtime:

\[ 3 \cdot \frac{1}{2} + 5 \cdot \frac{1}{4} + \cdots + (2n+1) \cdot \frac{1}{2^n} + \cdots = 5 \]
Why do we care about Recursive Probabilistic Programs?

- They allow representing **Randomized Divide & Conquer (D&C) Algorithms**

- Randomized (D&C) algorithms are “fast”

“For many applications, a randomized algorithm is the simplest algorithm available, or the fastest, or both.”
Why do we care about Recursive Probabilistic Programs?

- They allow representing **Randomized Divide & Conquer (D&C) Algorithms**
- Randomized (D&C) algorithms are “fast”

**Quicksort:**

\[
QS(A) \triangleq \\
\text{if } (|A| \leq 1) \text{ then } \text{return}(A); \\
i := \lfloor |A|/2 \rfloor; \\
A_\text{<} := \{a' \in A \mid a' < A[i]\}; \\
A_\text{>} := \{a' \in A \mid a' > A[i]\}; \\
\text{return} (QS(A_\text{<}) ++ A[i] ++ QS(A_\text{>}) )
\]

**Worst case complexity:**

\(O(n^2)\) comparisons
Why do we care about Recursive Probabilistic Programs?

- They allow representing **Randomized Divide & Conquer (D&C) Algorithms**
- Randomized (D&C) algorithms are “fast”

**QuickSort:**

\[
\text{QS}(A) \triangleq \begin{cases} 
\text{if } (|A| \leq 1) \text{ then return } (A); \\
i := \lceil |A|/2 \rceil; \\
A_{<} := \{a' \in A \mid a' < A[i]\}; \\
A_{>} := \{a' \in A \mid a' > A[i]\}; \\
\text{return } (\text{QS}(A_{<}) ++ A[i] ++ \text{QS}(A_{>}))
\end{cases}
\]

**Worst case complexity:**

\(O(n^2)\) comparisons

**Randomized QuickSort:**

\[
\text{rQS}(A) \triangleq \begin{cases} 
\text{if } (|A| \leq 1) \text{ then return } (A); \\
i := \text{rand}[1 \ldots |A|]; \\
A_{<} := \{a' \in A \mid a' < A[i]\}; \\
A_{>} := \{a' \in A \mid a' > A[i]\}; \\
\text{return } (\text{QS}(A_{<}) ++ A[i] ++ \text{QS}(A_{>}))
\end{cases}
\]

**Worst case complexity:**

\(O(n \log(n))\) expected comparisons

“For many applications, a randomized algorithm is the simplest algorithm available, or the fastest, or both.”

They allow representing **Randomized Divide & Conquer (D&C) Algorithms**

Randomized (D&C) algorithms are “fast”
Las Vegas Algorithms

- Quicksort
- Median finding
- Binary search

- **Runtime:** random variable
- **Error:** 0

Monte Carlo Algorithms

- Simple path of length $k$
- Low rank matrix approximation
- Euclidean matching

- **Runtime:** constant (always fast)
- **Error:** > 0

The analysis of randomized D&Q algorithms requires reasoning about both their correctness and performance.
Our Contribution

FORMAL VERIFICATION OF RECURSIVE PROBABILISTIC PROGRAMS

- Two calculi à la weakest pre-condition:
  - For reasoning about program outcomes:
    - Provability of post-conditions: \( \Pr[x = 1] = \frac{1}{2} \)
    - Expected values of numeric expressions: \( \mathbb{E}[x] = 2 \)
  - For reasoning about program expected runtimes:
    - Average time until program termination: \( [x > 0] \cdot y + [x \leq 0] \cdot 2 \)

- Soundness of the calculi w.r.t. an operational semantics

- Application: probabilistic binary search
Our Contribution

Two calculi à la weakest pre-condition:

- For reasoning about **program outcomes**:
  - Provability of post-conditions: $\Pr[x = 1] = \frac{1}{2}$
  - Expected values of numeric expressions: $E[x] = 2$

- For reasoning about
  - Average time until program termination: $[x > 0] \cdot y + [x \leq 0] \cdot 2$

- Soundness of the calculi w.r.t. an operational semantics

- Application: probabilistic binary search
We assume only one procedure $P$

No argument passing or return expression in $P$ (it manipulates the *global* program state).
Our Programming Model

We assume only one procedure $P$

No argument passing or return expression in $P$ (it manipulates the global program state).

Example: Factorial

$$P \triangleright \text{if } (x \leq 0) \text{ then } \{y := 1\} \text{ else }$$
$$\{x := x-1; \text{ call } P; \}$$
$$x := x+1; y := y \cdot x\}$$
Our Programming Model

Language Syntax

\[ C \,:= \, \text{skip} \quad | \quad \text{nop} \]
\[ | \quad \text{abort} \quad | \quad \text{abortion} \]
\[ | \quad x := E \quad | \quad \text{assignment} \]
\[ | \quad \text{if} (G) \text{then} \{C\} \text{else} \{C\} \quad | \quad \text{conditional} \]
\[ | \quad \{C\} [p] \{C\} \quad | \quad \text{probabilistic choice} \]
\[ | \quad \text{call } P \quad | \quad \text{procedure call} \]
\[ | \quad C; \ C \quad | \quad \text{sequence} \]

We assume only one procedure \( P \)

No argument passing or return expression in \( P \) (it manipulates the \textit{global} program state).

Example: Faulty factorial

\[
P \triangleright \, \text{if } (x \leq 0) \text{then } \{y := 1\} \text{ else } \{x := x-1; \text{call } P; \{x := x+1; \{y := y \cdot x\}[1/2]\{\text{skip}\}\}\}
\]

\[ x = 6 \quad \xrightarrow{\frac{1}{4}} \quad y = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \]
Probabilistic Predicate Transformer — Basics

\[
wp[c]: (\mathcal{S} \rightarrow \{0,1\}) \rightarrow (\mathcal{S} \rightarrow \{0,1\})
\]
$wp[c] : (\mathbb{S} \to [0, 1]) \to (\mathbb{S} \to [0, 1])$

$wp[c](f) = \lambda s \cdot E_{[c]}(s)(f)$
\[
wp[c]: (\mathbb{S} \rightarrow [0, 1]) \rightarrow (\mathbb{S} \rightarrow [0, 1]) \\
wp[c](f) = \lambda s \cdot E\llbracket c \rrbracket(s)(f)
\]
**Probabilistic Predicate Transformer — Basics**

\[ wp[c]: (\mathbb{S} \rightarrow [0, 1]) \rightarrow (\mathbb{S} \rightarrow [0, 1]) \]

\[ wp[c](f) = \lambda s \cdot E_{[c]}(s)(f) \]

- \( f \): utility or reward function over final program states
- \( wp[c](f) \): average earned reward for each initial program state

**c:**

\[ \{ x := 0 \} \frac{1}{2} \{ x := 1 \}; \]
\[ \{ y := 0 \} \frac{1}{2} \{ y := 1 \} \]

\[ wp[c](f) = \lambda s \cdot \frac{1}{4} \cdot f(s[x, y/0, 0]) + \frac{1}{4} \cdot f(s[x, y/0, 1]) \]
\[ + \frac{1}{4} \cdot f(s[x, y/1, 0]) + \frac{1}{4} \cdot f(s[x, y/1, 1]) \]
Probabilistic Predicate Transformer — Basics

\[ wp[c]: (\mathbb{S} \to [0, 1]) \to (\mathbb{S} \to [0, 1]) \]

\[ wp[c](f) = \lambda s \cdot E_{[c]}(s)(f) \]

- \( f \): utility or reward function over final program states
- \( wp[c](f) \): average earned reward for each initial program state
- \( wp[c]([Q]) \): probability of establishing post-condition \( Q \).

\[ wp[c](f) = \lambda s \cdot \frac{1}{4} \cdot f(s[x, y/0, 0]) + \frac{1}{4} \cdot f(s[x, y/0, 1]) \]
\[ + \frac{1}{4} \cdot f(s[x, y/1, 0]) + \frac{1}{4} \cdot f(s[x, y/1, 1]) \]

\[ wp[c]([x = y]) = \lambda s \cdot \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 0 + \frac{1}{4} \cdot 0 + \frac{1}{4} \cdot 1 = \lambda s \cdot \frac{1}{2} \]
\[
\begin{align*}
wp[x := E](f) &= f[x/E] \\
wp[c_1; c_2](f) &= (wp[c_1] \circ wp[c_2])(f) \\
&\vdots \\
wp[c_1 \{p\} c_2](f) &= p \cdot wp[c_1](f) + (1-p) \cdot wp[c_2](f)
\end{align*}
\]

Like ordinary \(wp[\cdot]\)
probabilistic predicate transformers — inductive definition

\[
\begin{align*}
wp[x := E](f) &= f[x/E] \\
wp[c_1; c_2](f) &= (wp[c_1] \circ wp[c_2])(f) \\
\vdots
\end{align*}
\]

\[
wp\{c_1\} [p] \{c_2\}(f) = p \cdot wp[c_1](f) + (1-p) \cdot wp[c_2](f)
\]

\[
wp[\text{call } P](f) = \sup_n wp[\text{call}_n P](f)
\]

like ordinary \(wp[\cdot]\)

\[n\text{-inlining of } P\]

\[
\begin{align*}
call_0 P &= \text{abort} \\
call_{n+1} P &= \text{body}(P)[\text{call } P/call_n P]
\end{align*}
\]
Proof rule for upper bounds

“Prove the desired specification for the procedure’s body assuming it already holds for the recursive calls in it.”

\[
\begin{align*}
\text{wp[call } P\text{(}f\text{)](}f\text{)} &\leq u \\
\vdash \text{wp[body}(P)\text{)](}f\text{)} &\leq u \\
\end{align*}
\]

\[
\frac{
\text{wp[call } P\text{(}f\text{)](}f\text{)} \leq u}
{\text{wp[call } P\text{(}f\text{)](}f\text{)} \leq u}
\]

\[
\langle u \rangle
\]

\[
\begin{array}{ll}
\text{body}(P) & \text{call } P \\
\vdots & \\
\langle f \rangle & \\
\end{array}
\]
Proof rule for upper bounds

"Prove the desired specification for the procedure’s body assuming it already holds for the recursive calls in it."

\[
\begin{align*}
\text{wp[call } P](f) & \leq u \quad \vdash \quad \text{wp[body}(P)\text{](f)} \leq u \\
\text{wp[call } P\text{](f)} & \leq u
\end{align*}
\]
Proof rule for upper bounds

“Prove the desired specification for the procedure’s body assuming it already holds for the recursive calls in it.”

\[
\text{wp[call } P\text{](}f\text{)} \leq u \quad \vdash \quad \text{wp[body}(P)\text{)](}f\text{)} \leq u
\]

\[
\text{wp[call } P\text{](}f\text{)} \leq u
\]

\[
\begin{align*}
\text{wp[body}(P)\text{)](}f\text{)} & \leq u \\
\vdash & \\
\text{wp[call } P\text{](}f\text{)} & \leq u
\end{align*}
\]

\[
\begin{align*}
\text{body}(P) & \left\{ \begin{array}{l}
\langle u \rangle \\
\vdots \\
\langle f' \rangle \\
\vdots \\
\langle f \rangle \\
f' \leq f
\end{array} \right. \\
\text{call } P
\end{align*}
\]
Proof rule for upper bounds

“Prove the desired specification for the procedure’s body assuming it already holds for the recursive calls in it.”

\[
\begin{align*}
    \text{wp}[\text{call } P](f) & \leq u \\
    \implies \text{wp}[\text{body}(P)](f) & \leq u \\
    \text{wp}[\text{call } P](f) & \leq u
\end{align*}
\]
Proof rule for upper bounds

"Prove the desired specification for the procedure's body assuming it already holds for the recursive calls in it."

\[
\begin{align*}
\mathit{wp}[\text{call } P](f) \leq u & \quad \vdash \quad \mathit{wp}[\text{body}(P)](f) \leq u \\
\hline
\mathit{wp}[\text{call } P](f) \leq u
\end{align*}
\]
Proof rule for upper bounds

"Prove the desired specification for the procedure’s body assuming it already holds for the recursive calls in it."

\[
\begin{align*}
\text{wp[call } P(f) \leq u \quad \vdash \quad \text{wp[body}(P)](f) \leq u \\
\text{wp[call } P](f) \leq u
\end{align*}
\]

Proof rule for lower bounds

\[
\begin{align*}
l_0 &= 0 \\
l_n \leq \text{wp[call } P](f) &\vdash l_{n+1} \leq \text{wp[body}(P)](f) \\
\sup_n l_n \leq \text{wp[call } P](f)
\end{align*}
\]

- Dual rule for upper bounds is also sound
Our Contribution

Two calculi à la weakest pre-condition:

- For reasoning about
  - Provability of post-conditions: \( \text{Pr}[x = 1] = \frac{1}{2} \)
  - Expected values of numeric expressions: \( \text{E}[x] = 2 \)

- For reasoning about program expected runtimes:
  - Average time until program termination: \([x > 0] \cdot y + [x \leq 0] \cdot 2\)

- Soundness of the calculi w.r.t. an operational semantics

- Application: probabilistic binary search
The Expected Runtime Transformer — Basics

\[ [c] \bar{\Xi} : S \rightarrow \mathbb{R}_{\geq 0} \]

\[ [c] \bar{\Xi}(s) = \text{number of skips, assignments, guard evaluations, coin flips and procedure calls in the execution of } c \text{ from state } s \]

To define \([c] \bar{\Xi}\) we use a continuation passing style through transformer

\[ \text{ert } [c] : (S \rightarrow \mathbb{R}_{\geq 0}) \rightarrow (S \rightarrow \mathbb{R}_{\geq 0}) \]

\[ t = \text{runtime of the computation following program } c \]

\[ \text{ert } [c](t) = \text{runtime of } c, \text{ plus the computation following } c \]

In particular,

\[ \text{ert } [c](0) = [c] \bar{\Xi} \]
\[\text{ert} [x := E](t) = \]
\[\text{ert} [{c_1} [p] {c_2}](t) = \]
\[\text{ert} [c_1; c_2](t) = \]
\[\text{ert} [\text{call } P](t) = \]
The Expected Runtime Transformer — Inductive Definition

\[
er[t[x := E]](t) = 1 + t[x/E]
\]

\[
er[{c_1} \{p\} \{c_2\}](t) =
\]

\[
er[c_1; c_2](t) =
\]

\[
er[\text{call } P](t) =
\]
The Expected Runtime Transformer — Inductive Definition

\[
\begin{align*}
\text{ert } [x := E](t) &= 1 + t[x/E] \\
\text{ert } [\{c_1\} [p] \{c_2\}](t) &= 1 + p \cdot \text{ert } [c_1](t) + (1 - p) \cdot \text{ert } [c_2](t) \\
\text{ert } [c_1; c_2](t) &= \\
\text{ert } [\text{call } P](t) &= 
\end{align*}
\]
\begin{align*}
\text{ert}[x := E](t) &= 1 + t[x/E] \\
\text{ert}[[c_1\ [p\ {c_2}]](t) &= 1 + p \cdot \text{ert}[c_1](t) + (1-p) \cdot \text{ert}[c_2](t) \\
\text{ert}[c_1; c_2](t) &= \text{ert}[c_1](\text{ert}[c_2](t)) \\
\text{ert}[\text{call } P](t) &=
\end{align*}
The Expected Runtime Transformer — Inductive Definition

\[\text{ert} [x := E](t) = 1 + t[x/E]\]

\[\text{ert} [{c_1}; \{c_2\}](t) = 1 + p \cdot \text{ert}[c_1](t) + (1-p) \cdot \text{ert}[c_2](t)\]

\[\text{ert} [c_1; c_2](t) = \text{ert}[c_1](\text{ert}[c_2](t))\]

\[\text{ert} [\text{call } P](t) = \text{lfp}\left(\lambda \eta \cdot 1 \oplus \text{ert}[\text{body}(P)]_{\eta}\right)(t)\]

“\text{ert} [\text{call } P](t) = 1 + \text{ert}[\text{body}(P)](t)\)”
Rules from the wp—calculus can be easily adapted for the ert—calculus

\[
\text{ert[call } P(t) \leq u \quad \vdash \quad \text{ert[body}(P)](t) \leq u
\]

\[
\text{ert[call } P(t) \leq u
\]
Rules from the \( \text{wp—calculus} \) can be easily adapted for the \( \text{ert—calculus} \)

\[
\text{ert}[\text{call } P](t) \leq u + 1 \quad \models \quad \text{ert}[\text{body}(P)](t) \leq u
\]

\[
\text{ert}[\text{call } P](t) \leq u + 1
\]
Our Contribution

FORMAL VERIFICATION OF RECURSIVE PROBABILISTIC

- Two calculi
  - For reasoning about
    - Provability of post-conditions: $\Pr[x = 1] = \frac{1}{2}$
    - Expected values of numeric expressions: $E[x] = 2$
  - For reasoning about
    - Average time until program termination: $[x > 0] \cdot y + [x \leq 0] \cdot 2$

- Soundness of the calculi w.r.t. an operational semantics

- Application: probabilistic binary search
To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $\mathcal{M}_{s_0}^f [c]$

We prove that the weakest pre-condition $wp[c](f)(s_0)$ coincides with the expected reward $ER(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$$wp[c](f)(s_0) = ER(\Diamond \text{Term})$$
Operational Semantics

To each program \( c \), initial state \( s_0 \) and post-condition \( f \) we associate a reward pushdown Markov chain \( M_{s_0}^f [c] \).

We prove that the weakest pre-condition \( wp[c](f)(s_0) \) coincides with the expected reward \( ER(\diamond Term) \) upon reaching a terminal state in the Markov chain.

\[
wp[c](f)(s_0) = ER(\diamond Term)
\]

Example:

\[
P \triangleright \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}
\]
To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $\mathcal{M}_{s_0}^f[c]$

We prove that the weakest pre-condition $\wp[c](f)(s_0)$ coincides with the expected reward $\text{ER}(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$$\wp[c](f)(s_0) = \text{ER}(\Diamond \text{Term})$$

Example:

$$P \triangleright \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}$$
To each program $c$, initial state $s_0$ and post-condition $f$ we associate a **reward pushdown Markov chain** $M^f_{s_0} [c]$.

We prove that the weakest pre-condition $wp[c](f)(s_0)$ coincides with the **expected reward** $ER(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain.

$$wp[c](f)(s_0) = ER(\Diamond \text{Term})$$

**Example:**

$$P \triangleright \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}$$
To each program \( c \), initial state \( s_0 \) and post-condition \( f \) we associate a reward pushdown Markov chain \( M_{s_0}^f[c] \).

We prove that the weakest pre-condition \( \text{wp}[c](f)(s_0) \) coincides with the expected reward \( \text{ER}(\diamond \text{Term}) \) upon reaching a terminal state in the Markov chain:

\[
\text{wp}[c](f)(s_0) = \text{ER}(\diamond \text{Term})
\]

Example:

\[
P \triangleright \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}
\]
Operational Semantics

- To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $M^f_{s_0}[c]$

- We prove that the weakest pre-condition $\wp[c](f)(s_0)$ coincides with the expected reward $ER(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$$\wp[c](f)(s_0) = ER(\Diamond \text{Term})$$

Example:

$P \triangleright \{\text{skip}^1\} \ [1/2]^2 \ \{\text{call } P^3; \text{ call } P^4\}$
Operational Semantics

To each program \( c \), initial state \( s_0 \) and post-condition \( f \) we associate a reward pushdown Markov chain \( M_{s_0}^f[c] \).

We prove that the weakest pre-condition \( wp[c](f)(s_0) \) coincides with the expected reward \( ER(\diamond \text{Term}) \) upon reaching a terminal state in the Markov chain.

\[
wp[c](f)(s_0) = ER(\diamond \text{Term})
\]

Example:

\[
P \Rightarrow \{\text{skip}^1\} \ [1/2]^2 \ \{\text{call } P^3; \ \text{call } P^4\}
\]
To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $\mathcal{M}_{s_0}[c]$

We prove that the weakest pre-condition $wp[c](f)(s_0)$ coincides with the expected reward $\text{ER}(\diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$$wp[c](f)(s_0) = \text{ER}(\diamond \text{Term})$$

Example:

$$P \triangleright \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}$$
To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $\mathcal{M}_s [c]$.

We prove that the weakest pre-condition $wp[c](f)(s_0)$ coincides with the expected reward $ER(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$\text{Example:}$

$$\exists P \triangleright \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}$$

\begin{center}
\begin{tikzpicture}
\node (1) at (0,0) [circle,draw] {1};
\node (2) at (-2,-2) [circle,draw] {2};
\node (3) at (0,-2) [circle,draw] {3};
\node (term) at (2,0) [circle,draw] {Term};

\draw [->] (1) edge node [above] {$\downarrow$} (term);
\draw [->] (term) edge node [below] {empty} (term);
\draw [->] (2) edge node [left] {$\frac{1}{2}$} (1);
\draw [->] (3) edge node [left] {$\frac{1}{2}$} (2);
\draw [->] (2) edge node [left] {push(4)} (3);
\end{tikzpicture}
\end{center}
Operational Semantics

To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $M_{s_0}^f[c]$

We prove that the weakest pre-condition $wp[c](f)(s_0)$ coincides with the expected reward $ER(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$$wp[c](f)(s_0) = ER(\Diamond \text{Term})$$

Example:

$$P \triangleright \{\text{skip}^1\} \ [1/2]^2 \ \{\text{call } P^3; \text{ call } P^4\}$$

![Diagram](image)
Operational Semantics

- To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $\mathcal{M}_{s_0}^f[c]$.

- We prove that the weakest pre-condition $\wp[c](f)(s_0)$ coincides with the expected reward $ER(\diamond Term)$ upon reaching a terminal state in the Markov chain.

\[
\wp[c](f)(s_0) = ER(\diamond Term)
\]

Example:

\[
P \triangleright \{\text{skip}^1\} \left[\frac{1}{2}\right]^2 \{\text{call } P^3; \text{ call } P^4\}
\]
To each program \( c \), initial state \( s_0 \) and post-condition \( f \) we associate a reward pushdown Markov chain \( M_{s_0}^f[c] \).

We prove that the weakest pre-condition \( wp[c](f)(s_0) \) coincides with the expected reward \( ER(\Diamond \text{Term}) \) upon reaching a terminal state in the Markov chain:

\[
wp[c](f)(s_0) = ER(\Diamond \text{Term})
\]

Example:

\[
\begin{array}{c}
P \triangleright \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}
\end{array}
\]
To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $M^f_{s_0}[c]$

We prove that the weakest pre-condition $wp[c](f)(s_0)$ coincides with the expected reward $ER(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$$wp[c](f)(s_0) = ER(\Diamond \text{Term})$$

Example:

$$P \triangleright \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}$$
Operational Semantics

To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $M^f_{s_0}[c]$

We prove that the weakest pre-condition $wp[c](f)(s_0)$ coincides with the expected reward $ER(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$$wp[c](f)(s_0) = ER(\Diamond \text{Term})$$

Example:

$P \Rightarrow \{\text{skip}^1\} [1/2]^2 \{\text{call } P^3; \text{ call } P^4\}$
Operational Semantics

To each program $c$, initial state $s_0$ and post-condition $f$ we associate a **reward pushdown Markov chain** $\mathcal{M}_{s_0}^f[c]$.

We prove that the weakest pre-condition $\text{wp}[c](f)(s_0)$ coincides with the **expected reward** $\text{ER}(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain.

$$\text{wp}[c](f)(s_0) = \text{ER}(\Diamond \text{Term})$$

**Example:**

\[ P \triangleright \{\text{skip}^1\} \left[\frac{1}{2}\right]^2 \{\text{call } P^3; \text{ call } P^4\} \]
Operational Semantics

To each program \( c \), initial state \( s_0 \) and post-condition \( f \) we associate a reward pushdown Markov chain \( \mathcal{M}_{s_0}^f[c] \).

We prove that the weakest pre-condition \( \wp[c](f)(s_0) \) coincides with the expected reward \( \text{ER}(\Diamond \text{Term}) \) upon reaching a terminal state in the Markov chain.

\[
\wp[c](f)(s_0) = \text{ER}(\Diamond \text{Term})
\]

Example:

\[
P \triangleright \{\text{skip}^1\} \cdot [1/2]^2 \cdot \{\text{call } P^3; \text{ call } P^4\}
\]
Operational Semantics

To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $\mathcal{M}_{s_0}^f[c]$

We prove that the weakest pre-condition $\wp[c](f)(s_0)$ coincides with the expected reward $ER(\Diamond \text{Term})$ upon reaching a terminal state in the Markov chain

$$\wp[c](f)(s_0) = ER(\Diamond \text{Term})$$

Example:

$$P \triangleright \{\text{skip}^1\} \cdot [1/2]^2 \cdot \{\text{call } P^3; \text{ call } P^4\}$$
To each program \( c \), initial state \( s_0 \) and post-condition \( f \) we associate a **reward pushdown Markov chain** \( \mathcal{M}_{s_0}^f[c] \)

We prove that the weakest pre-condition \( \wp[c](f)(s_0) \) coincides with the **expected reward** \( \text{ER}(\triangleright \text{Term}) \) upon reaching a terminal state in the Markov chain

\[
\text{wp}[c](f)(s_0) = \text{ER}(\triangleright \text{Term})
\]

**Example:**

\[
P \triangleright \{\text{skip}^1\} \left[\frac{1}{2}\right]^2 \{\text{call } P^3; \text{ call } P^4\}
\]

\[
\text{ER}(\triangleright \text{Term}) = \sum \Pr(\pi) \cdot f(s')
\]

\[
\pi : (\ell_0, s_0) \sim (\text{Term}, s')
\]
To each program $c$, initial state $s_0$ and post-condition $f$ we associate a reward pushdown Markov chain $\mathcal{M}^{f}_{s_0}[c]$. We prove that the weakest pre-condition $wp[c](f)(s_0)$ coincides with the expected reward $\text{ER}(\diamond \text{Term})$ upon reaching a terminal state in the Markov chain.

$$wp[c](f)(s_0) = \text{ER}(\diamond \text{Term})$$

**Example:**

$$P \triangleright \{\text{skip}^1\} \rightsquigarrow \{[\frac{1}{2}]^2 \ \{\text{call } P^3; \text{ call } P^4\}\}$$

$$\text{ER}(\diamond \text{Term}) = \sum_{\pi : \langle \ell_0, s_0 \rangle \rightsquigarrow \langle \text{Term}, s' \rangle} \Pr(\pi) \cdot f(s')$$

$$f = 1$$

$$= \frac{1}{2} + \frac{1}{2} \cdot \left(\frac{1}{2}\right)^2 + \ldots$$
Our Contribution

Two calculi

- For reasoning about
  - Provability of post-conditions: \( \Pr [x = 1] = \frac{1}{2} \)
  - Expected values of numeric expressions: \( E[x] = 2 \)

- For reasoning about
  - Average time until program termination: \( [x > 0] \cdot y + [x \leq 0] \cdot 2 \)

Soundness of the calculi w.r.t. an operational semantics

Application: probabilistic binary search
Case Study: Probabilistic Binary Search

Input: sorted array \(a[left...right]\), value \(val\) to search in the array

Output: index of the array containing \(val\) (if any)

\[
\text{BinSearch} \triangleq \\
pivot := \text{unif}[left...right]; \\
\text{if (} left < right \text{)} \\
\quad \text{if (} a[pivot] < val \text{)} \\
\quad \quad left := \min\{pivot + 1, right\}; \\
\quad \quad \text{call BinSearch} \\
\quad \text{if (} a[pivot] > val \text{)} \\
\quad \quad right := \max\{pivot - 1, left\}; \\
\quad \quad \text{call BinSearch} \\
\]

// if the array has at least two elements
// search \(val\) in \(a[pivot+1...right]\)
// search \(val\) in \(a[left...pivot-1]\)
// \(val = a[pivot]\)
// if the array has one element, \(left=right=pivot\)
Case Study: Probabilistic Binary Search

**Correctness:** if the array contains the searched value, upon the program termination its index will be \( \text{pivot} \) (with probability one)

\[
1 \cdot [G^{\top} ] \leq \text{wp[call BinSearch]}([a[\text{pivot}] = \text{val}])
\]

\[
G^{\top} \triangleq \text{left} \leq \text{right} \land \text{sorted}(a[\text{left}...\text{right}]) \land \text{val} \in a[\text{left}...\text{right}]
\]

**Expected Runtime:** if the array does not contain the searched value, the expected runtime of the program is in \( \Theta(\log n) \), where \( n = \text{right} - \text{left} + 1 \) is the size of the array

\[
\text{ert[call BinSearch]}(0) \leq 4 + [\neg G^{\top} ] \cdot \infty + [G^{\top} ] \cdot (6 H_n - 2.5)
\]

\[
G^{\top} \triangleq \text{left} \leq \text{right} \land \text{sorted}(a[\text{left}...\text{right}]) \land \text{val} \not\in a[\text{left}...\text{right}]
\]

\[
H_n \triangleq \sum_{i=1}^{n} \frac{1}{i} \quad (n\text{-th harmonic number})
\]
Algebraic properties of both transformers $\text{wp}[]{\cdot}$ and $\text{ert}[]{\cdot}$, e.g.

\[
\text{wp}[c](a \cdot f + b \cdot g) = a \cdot \text{wp}[c](f) + b \cdot \text{wp}[c](g)
\]

\[
\text{ert}[c](k + t) = k + \text{ert}[c](t)^*
\]

\[
\text{ert}[c](t) = \text{ert}[c](0) + \text{wp}[c](t)^*
\]

Relation between finite expected runtime and program termination

\[
\text{ert}[c](0)(s) < \infty \implies \text{wp}[c](1)(s) = 1^*
\]

Liberal variant of transformer $\text{wp}[]{\cdot}$

Fixed point characterisation of $w(l)p[\text{call } P]$
Summary

What we have done:

- Deductive technique for verifying recursive probabilistic programs
  - Two calculi for reasoning about the outcome and runtime of programs
  - Set of proof rules for reasoning about recursive programs
  - Soundness w.r.t an operational semantics
  - Application: probabilistic binary search

What we would like to do:

- Automate the verification process
- More challenging case studies (e.g. randomised quicksort)
Summary

What we have done:

- Deductive technique for verifying recursive probabilistic programs
  - Two calculi for reasoning about the outcome and runtime of programs
  - Set of proof rules for reasoning about recursive programs
  - Soundness w.r.t an operational semantics
  - Application: probabilistic binary search

What we would like to do:

- Automate the verification process
- More challenging case studies (e.g. randomised quicksort)
BACKUP SLIDES
\[ wp[	ext{skip}](f) = f \]
\[ wp[	ext{abort}](f) = 0 \]
\[ wp[x := E](f) = f[x/E] \]
\[ wp[\text{if } (G) \text{ then } \{ c_1 \} \text{ else } \{ c_2 \}](f) = [G] \cdot wp[c_1](f) + [\neg G] \cdot wp[c_2](f) \]
\[ wp[\{ c_1 \} [p] \{ c_2 \}](f) = p \cdot wp[c_1](f) + (1 - p) \cdot wp[c_2](f) \]
\[ wp[c_1; c_2](f) = (wp[c_1] \circ wp[c_2])(f) \]
\[ wp[	ext{call } P] = \sup_n wp[\text{call}_n P] \]
Example 3. Reconsider the procedure $P_{rec_3}$ with declaration

$$D(P_{rec_3}) : \{\text{skip}\} [1/2] \{\text{call } P_{rec_3}; \text{call } P_{rec_3}; \text{call } P_{rec_3}\}$$

presented in the introduction. We prove that it terminates with probability at most $\varphi = \frac{\sqrt{5} - 1}{2}$ from any initial state. Formally, this is captured by $\wp[\text{call } P, D](1) \leq \varphi$. To prove this, we apply rule [wp-rec]. We must then establish the derivability claim

$$\wp[\text{call } P](1) \leq \varphi \quad \vdash \quad \wp[D(P_{rec_3})](1) \leq \varphi.$$

The derivation goes as follows:

$$\begin{align*}
\wp[D(P_{rec_3})](1) \\
= \{\text{def. of } \wp\} \\
\frac{1}{2} \cdot \wp[\text{skip}](1) + \frac{1}{2} \cdot \wp[\text{call } P_{rec_3}; \text{call } P_{rec_3}; \text{call } P_{rec_3}](1) \\
= \{\text{def. of } \wp\} \\
\frac{1}{2} + \frac{1}{2} \cdot \wp[\text{call } P_{rec_3}; \text{call } P_{rec_3}](\wp[\text{call } P_{rec_3}](1)) \\
\leq \{\text{assumption, monot. of } \wp\} \\
\frac{1}{2} + \frac{1}{2} \cdot \wp[\text{call } P_{rec_3}; \text{call } P_{rec_3}](\varphi) \\
= \{\text{def. of } \wp, \text{scalab. of } \wp \text{ twice}\} \\
\frac{1}{2} + \frac{1}{2} \varphi \cdot \wp[\text{call } P_{rec_3}](\wp[\text{call } P_{rec_3}](1)) \\
\leq \{\text{assumption, monot. of } \wp\} \\
\frac{1}{2} + \frac{1}{2} \varphi \cdot \wp[\text{call } P_{rec_3}](\varphi) \\
= \{\text{scalab. of } \wp\} \\
\frac{1}{2} + \frac{1}{2} \varphi^2 \cdot \wp[\text{call } P_{rec_3}](1) \\
\leq \{\text{assumption, monot. of } \wp\} \\
\frac{1}{2} + \frac{1}{2} \varphi^3 \\
= \{\text{algebra}\} \\
\varphi \\
\triangle
\end{align*}$$
The Expected Runtime Transformer — Inductive Definition

\[
\begin{align*}
\text{ert}[\text{skip}](t) & = 1 + t \\
\text{ert}[\text{abort}](t) & = 0 \\
\text{ert}[x := E](t) & = 1 + t[x/E] \\
\text{ert}[\text{if}(G)\text{ then } \{c_1\} \text{ else } \{c_2\}](t) & = 1 + [G] \cdot \text{ert}[c_1](t) + [\neg G] \cdot \text{ert}[c_2](t) \\
\text{ert}[\{c_1\} [p] \{c_2\}](t) & = 1 + p \cdot \text{ert}[c_1](t) + (1 - p) \cdot \text{ert}[c_2](t) \\
\text{ert}[c_1; c_2](t) & = (\text{ert}[c_1] \circ \text{ert}[c_2])(t) \\
\text{ert}[\text{call } P](t) & = \text{lfp}\left(\lambda \eta \cdot 1 \oplus \text{ert}[\text{body}(P)]^{\eta}\right)(t)
\end{align*}
\]
Rules from the wp—calculus can be easily adapted for the ert—calculus

Proof rule for upper bounds

\[
\begin{align*}
\mathsf{ert} \left[ \text{call } P \right](t) &\leq u + 1 \quad \vdash \quad \mathsf{ert} \left[ \text{body}(P) \right](t) \leq u \\
\mathsf{ert} \left[ \text{call } P \right](t) &\leq u + 1
\end{align*}
\]

Proof rule for upper bounds

\[
\begin{align*}
l_0 &= 0 \\
l_n + 1 \leq \mathsf{ert} \left[ \text{call } P \right](t) &\quad \vdash \quad l_{n+1} \leq \mathsf{ert} \left[ \text{body}(P) \right](t) \\
\sup_n (l_n + 1) &\leq \mathsf{ert} \left[ \text{call } P \right](t)
\end{align*}
\]
### Operational Semantics

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt(ℓ) = skip</td>
<td>[ skip ]</td>
</tr>
<tr>
<td>succ₁(ℓ) = ℓ’</td>
<td></td>
</tr>
<tr>
<td>( \langle \ell, s \rangle \xrightarrow{\gamma, 1, \gamma} \langle \ell’, s \rangle )</td>
<td></td>
</tr>
<tr>
<td>stmt(ℓ) = if (G) { c₁ } else { c₂ }</td>
<td>[ \text{if1} ]</td>
</tr>
<tr>
<td>succ₁(ℓ) = ℓ’</td>
<td></td>
</tr>
<tr>
<td>( \langle \ell, s \rangle \xrightarrow{\gamma, 1, \gamma} \langle \ell’, s \rangle )</td>
<td></td>
</tr>
<tr>
<td>stmt(ℓ) = call P</td>
<td>[ \text{call} ]</td>
</tr>
<tr>
<td>succ₁(ℓ) = ℓ’</td>
<td></td>
</tr>
<tr>
<td>( \langle \ell, s \rangle \xrightarrow{\gamma, 1, \gamma} \langle \text{init}(\mathcal{D}(P)), s \rangle )</td>
<td></td>
</tr>
<tr>
<td>stmt(ℓ) = x := E</td>
<td>[ \text{assign} ]</td>
</tr>
<tr>
<td>succ₁(ℓ) = ℓ’</td>
<td></td>
</tr>
<tr>
<td>( \langle \ell, s \rangle \xrightarrow{\gamma, 1, \gamma} \langle \ell’, s[x \mapsto s(E)] \rangle )</td>
<td></td>
</tr>
<tr>
<td>stmt(ℓ) = abort</td>
<td>[ \text{abort} ]</td>
</tr>
<tr>
<td>( \langle \ell, s \rangle \xrightarrow{\gamma, 1, \gamma} \langle \ell’, s \rangle )</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.** Rules for defining an operational semantics for pRGCL programs. For sequential composition there is no dedicated rule as the control flow is encoded via the succ₁ and the succ₂ functions.
Proof Rule for Mutually Recursive Procedures

\[
\begin{align*}
\wp[\text{call } P_1](f_1) & \leq g_1, \ldots, \wp[\text{call } P_m](f_m) \leq g_m \quad \vdash \quad \wp[\text{body}(P_1)](f_1) \leq g_1 \\
\vdots
\end{align*}
\]

\[
\begin{align*}
\wp[\text{call } P_1](f_1) & \leq g_1, \ldots, \wp[\text{call } P_m](f_m) \leq g_m \quad \vdash \quad \wp[\text{body}(P_m)](f_m) \leq g_m \\
\hline
\wp[\text{call } P_i](f_i) & \leq g_i \quad \text{for all } i = 1 \ldots m
\end{align*}
\]
Case Study: Probabilistic Binary Search

Input: sorted array \(a[left...right]\), value \(val\) to search in the array

Output: index of the array containing \(val\) (if any)

- **Correctness:** if the array contains \(val\), upon the program termination its index will be \(pivot\) (with probability one)

\[
1 \cdot [G^{\neg \square}] \leq \wp[\text{call BinSearch}](\lbrack a[pivot] = val \rbrack)
\]

\[
G^{\neg \square} \triangleq left \leq right \land \text{sorted}(a[left...right]) \land val \in a[left...right]
\]

- **Expected Runtime:** if the array does not contain the searched value, the expected runtime of the program is \(6H_n + 1.5\), where \(H_n = \sum_{i=1}^{n} \frac{1}{i}\) is the \(n\)-th harmonic number and \(n = right - left + 1\) the size of the array

\[
\text{ert}[\text{call BinSearch}](0) \leq 4 + [\neg G^{\neg \square}] \cdot \infty + [G^{\neg \square}] \cdot (6H_n - 2.5)
\]

\[
G^{\neg \square} \triangleq left \leq right \land \text{sorted}(a[left...right]) \land val \notin a[left...right]
\]