Reasoning about Recursive Probabilistic Programs

Federico Olmedo
Joost-Pieter Katoen
RWTH Aachen University, Germany

Benjamin Kaminski
Christoph Matheja

LICS 2016
July 8th — New York City
Randomization Leads to Intricate Behaviours
Randomization Leads to Intricate Behaviours

\[ P \triangleright \{ \text{skip} \} \left[ \frac{1}{2} \right] \{ \text{call } P \} \]
$P \triangleright \{\text{skip}\} \left[\frac{1}{2}\right] \{\text{call } P\}$

Probability of Termination: 1

It terminates with probability 1, even though it admits arbitrarily long executions!
Randomization Leads to Intricate Behaviours

\[ P \triangleright \{ \text{skip} \} \left[ \frac{1}{2} \right] \{ \text{call } P \} \]

Probability of Termination: \( 1 \)

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

It terminates with probability 1, even though it admits arbitrarily long executions!
Randomization Leads to Intricate Behaviours

\[ P \triangleright \{\text{skip}\} \left[\frac{1}{2}\right] \{\text{call } P\} \]

Probability of Termination: 1

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

Probability of Termination: 1

It terminates with probability 1, even though it admits arbitrarily long executions!
Randomization Leads to Intricate Behaviours

\[ P \triangleright \{ \text{skip} \} \left[ \frac{1}{2} \right] \{ \text{call } P \} \]

Probability of Termination: 1

It terminates with probability 1, even though it admits arbitrarily long executions!

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

Probability of Termination: 1
Runtime:
Randomization Leads to Intricate Behaviours

\[ P \triangleright \{ \text{skip} \} \left[ \frac{1}{2} \right] \{ \text{call } P \} \]

Probability of Termination: 1

It terminates with probability 1, even though it admits arbitrarily long executions!

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

Probability of Termination: 1
Runtime: 1 sec.

Reasoning about Recursive Probabilistic Programs — Olmedo, Kaminski, Katoen & Matheja
Randomization Leads to Intricate Behaviours

\[ P \triangleright \{\text{skip}\} \left[\frac{1}{2}\right] \{\text{call } P\} \]

Probability of Termination: 1

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

Probability of Termination: 1
Runtime: 1 min.

It terminates with probability 1, even though it admits arbitrarily long executions!
Randomization Leads to Intricate Behaviours

\[ P \xrightarrow{\text{skip}} [\frac{1}{2}] \{ \text{call } P \} \]

Probability of Termination: 1

It terminates with probability 1, even though it admits arbitrarily long executions!

\[ P \xrightarrow{\text{skip}} [\frac{1}{2}] \{ \text{call } P; \text{ call } P \} \]

Probability of Termination: 1
Runtime: 1 hour
Randomization Leads to Intricate Behaviours

\[ P \xrightarrow{\{\text{skip}\}} \frac{1}{2} \xrightarrow{\text{call } P} \]

Probability of Termination: 1

\[ P \xrightarrow{\{\text{skip}\}} \frac{1}{2} \xrightarrow{\text{call } P; \text{ call } P} \]

Probability of Termination: 1
Runtime: \( \infty \)

It terminates with probability 1, even though it admits arbitrarily long executions!

It terminates with probability 1, but reaching termination takes (in average) infinite time!
Randomization Leads to Intricate Behaviours

\[ P \triangleleft \{\text{skip}\} \ [1/2] \ \{\text{call } P\} \]

Probability of Termination: 1

It terminates with probability 1, even though it admits arbitrarily long executions!

\[ P \triangleleft \{\text{skip}\} \ [1/2] \ \{\text{call } P; \text{ call } P\} \]

Probability of Termination: 1
Runtime: \(\infty\)

It terminates with probability 1, but reaching termination takes (in average) infinite time!

\[ P \triangleleft \{\text{skip}\} \ [1/2] \ \{\text{call } P; \text{ call } P; \text{ call } P\} \]
Randomization Leads to Intricate Behaviours

\[ P \xrightarrow{\text{skip}} \frac{1}{2} \{ \text{call } P \} \]
Probability of Termination: 1

\[ P \xrightarrow{\text{skip}} \frac{1}{2} \{ \text{call } P; \text{ call } P \} \]
Probability of Termination: 1
Runtime: \( \infty \)

\[ P \xrightarrow{\text{skip}} \frac{1}{2} \{ \text{call } P; \text{ call } P; \text{ call } P \} \]
Probability of Termination: \( \frac{\sqrt{5}-1}{2} \)
“For many applications, a randomized algorithm is the simplest algorithm available, or the fastest, or both.” [Motwani & Raghavan]
Randomized (Recursive) Algorithms are Natural and Widespread

“For many applications, a randomized algorithm is the simplest algorithm available, or the fastest, or both.” [Motwani & Raghavan]

QuickSort:

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

Deterministic version: \(O(n^2)\) comparisons
Randomized (Recursive) Algorithms are Natural and Widespread

“For many applications, a randomized algorithm is the simplest algorithm available, or the fastest, or both.” [Motwani & Raghavan]

Randomized Quicksort:

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

Randomized version: \( O(n \log(n)) \) comparisons
Randomized (Recursive) Algorithms are Natural and Widespread

“For many applications, a randomized algorithm is the simplest algorithm available, or the fastest, or both.” [Motwani & Raghavan]

Randomized Quicksort:

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

Randomized version: \( O(n \log(n)) \) comparisons

Sample Randomized Recursive Algorithms:

- Quicksort
- Median finding
- Binary search
- Simple path of length \( k \)
- Euclidean matching
- ….
Current Analysis Approaches are Not Satisfactory

Current Analysis Approaches:

- Mathematical ad-hoc reasoning (on involved random variables)
- Probabilistic recurrence relations
- Dedicated techniques for D&C algorithms
Current Analysis Approaches:

- Mathematical ad-hoc reasoning (on involved random variables)
- Probabilistic recurrence relations
- Dedicated techniques for D&C algorithms

- Cover only a fragment of the proof argument
- Non-trivial claims are taken for granted
Current Analysis Approaches are Not Satisfactory

Current Analysis Approaches:
- Mathematical ad-hoc reasoning (on involved random variables)
- Probabilistic recurrence relations
- Dedicated techniques for D&C algorithms

Our Approach:
- Formal verification
  - using only first principles
  - directly from the program code

- Cover only a fragment of the proof argument
- Non-trivial claims are taken for granted
Our Contribution

DEDUCTIVE VERIFICATION OF RANDOMIZED RECURSIVE ALGORITHMS

- Two calculi à la weakest pre-condition:
  - For reasoning about program outcomes, e.g. \( \Pr [x = x^{opt}] \geq 0.9 \)
  - For reasoning about program expected runtimes, e.g. \( \text{ert} \leq x + y \)

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

- Application: probabilistic binary search
For Program Outcomes
[Kozen ’81]

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

\[ wp[c](1_Q): \text{ probability that } c \text{ establishes post-condition } Q. \]
Calculi — Basics

For Program Outcomes
[Kozen ’81]

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

wp[c](\mathbb{1}_Q) : \textbf{probability} that \( c \) establishes post-condition \( Q \).

For Program Expected Runtimes
[ESOP ’16]

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

\[ \text{ert}[c](0) : \textbf{expected runtime} of \( c \). \]

\[ \text{runtime of the computation following } c, \text{ plus the computation following } c \]
Calculi — Basics

For Program Outcomes
[Kozen ‘81]

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

quantitative pre-condition

quantitative post-condition

\[ \text{wp}[c](\mathbb{1}_Q): \text{ probability that } c \text{ establishes post-condition } Q. \]

\[ \text{wp}\left[\{c_1\} [p] \{c_2\}\right](\mathbb{1}_Q) = p \cdot \text{wp}[c_1](\mathbb{1}_Q) + (1-p) \cdot \text{wp}[c_2](\mathbb{1}_Q) \]

For Program Expected Runtimes
[ESOP ‘16]

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

runtime of the computation following c

runtime of c, plus the computation following c

\[ \text{ert}[c](0): \text{ expected runtime of } c. \]

\[ \text{ert}\left[\{c_1\} [p] \{c_2\}\right](t) = 1 + p \cdot \text{ert}[c_1](t) + (1-p) \cdot \text{ert}[c_2](t) \]
For procedure calls, we intuitively have

\[ \text{wp}[\text{call } P](1_Q) = \text{wp}[\text{body}(P)](1_Q) \]

\[ \text{ert}[\text{call } P](t) = 1 + \text{ert}[\text{body}(P)](t) \]

but formal definitions require (higher order) fixed points.
For procedure calls, we intuitively have

\[ \text{wp[\text{call } P]}(1_Q) = \text{wp[body(P)]}(1_Q) \]

\[ \text{ert[call } P](t) = 1 + \text{ert[body(P)]}(t) \]

but formal definitions require (higher order) fixed points.

**Proof Rules for Procedure Calls**

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

- **For upper bounds**
  \[
  \begin{align*}
  \text{wp[call } P](1_Q) &\leq u \\
  \text{wp[body(P)]}(1_Q) &\leq u \\
  \text{wp[call } P](1_Q) &\leq u
  \end{align*}
  \]

- **For lower bounds**
  \[
  \begin{align*}
  l_0 &= 0 \\
  l_n &\leq \text{wp[call } P](1_Q) \\
  l_{n+1} &\leq \text{wp[body(P)]}(1_Q) \\
  \sup_n l_n &\leq \text{wp[call } P](1_Q)
  \end{align*}
  \]

- Dual rule for upper bounds is also sound
For procedure calls, we intuitively have

\[
wp[\text{call } P](1_Q) = wp[body(P)](1_Q)
\]

but formal definitions require (higher order) fixed points.

**Proof Rules for Procedure Calls**

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

**For upper bounds**

\[
\frac{wp[\text{call } P](1_Q) \leq u \quad \models \quad wp[body(P)](1_Q) \leq u}{wp[\text{call } P](1_Q) \leq u}
\]

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

**For lower bounds**

\[
l_0 = 0
\]

\[
l_n \leq wp[\text{call } P](1_Q) \quad \models \quad l_{n+1} \leq wp[body(P)](1_Q)
\]

\[
\sup_n l_n \leq wp[\text{call } P](1_Q)
\]

- Dual rule for upper bounds is also sound
Operational Semantics

Sample Program

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

Associated Pushdown Markov Chain
Operational Semantics

Sample Program

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

Associated Pushdown Markov Chain
Sample Program

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

Associated Pushdown Markov Chain
Operational Semantics

Sample Program

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

Associated Pushdown Markov Chain
Operational Semantics

Sample Program

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

Associated Pushdown Markov Chain
Sample Program

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

Associated Pushdown Markov Chain
Sample Program

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

Associated Pushdown Markov Chain
Sample Program

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

Associated Pushdown Markov Chain
Operational Semantics

Sample Program

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

Associated Pushdown Markov Chain

[Diagram of a pushdown Markov chain with states 1, 2, 3, 4, and transitions involving push, pop, and conditional probabilities.]
Sample Program

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

Associated Pushdown Markov Chain
Sample Program

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

Associated Pushdown Markov Chain
Sample Program

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

Associated Pushdown Markov Chain

\[ \text{wp[call } P](\mathbb{1}_{\text{true}}) = \text{Pr}(\diamond \text{Term}) \]
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)

---

Formal Verification of Correctness & Expected Runtime

**Correctness for case** \(val \in a[left...right]\)

\[1 \cdot 1_Gr^\perp \leq \text{wp[call PBS]}(1_{a[pivot]} = val)\]

left \(\leq\) right \(\land\) sorted\((a[left...right])\) \(\land\) \(val \in a[left...right]\)

**Runtime for case** \(val \notin a[left...right]\)

\[\text{ert[call PBS]}(0) \leq 4 + 1_{-G^\perp} \cdot \infty + 1_{Gr^\perp} \cdot \left(6H_n - 2.5\right)\]

left \(\leq\) right \(\land\) sorted\((a[left...right])\) \(\land\) \(val \notin a[left...right]\)

\[\sum_{i=1}^{n} \frac{1}{i} \text{ with } n = right - left + 1\]

PBS \(\triangleq\)

\[\begin{align*}
pivot &:= \text{rand}[left...right]; \\
\text{if} (left < right) &\text{ if } (a[pivot] < val) \\
&\text{ left := min}\{pivot + 1, right\}; \\
&\text{ call PBS} \\
\text{if } (a[pivot] > val) &\text{ right := max}\{pivot - 1, left\}; \\
&\text{ call PBS}
\end{align*}\]
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

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

Extension to mutual recursion
Summary

What we have done:

Deductive approach for the formal verification of randomized recursive algorithms
- 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. randomized Quicksort)
Summary

What we have done:

**Deductive approach for the formal verification of randomized recursive algorithms**

- 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. randomized Quicksort)

Thanks!
BACKUP SLIDES
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

Program Runtime

Average (or Expected) Runtime:

\[
3 \cdot \frac{1}{2} + 5 \cdot \frac{1}{4} + \cdots + (2n+1) \cdot \frac{1}{2^n} + \cdots = 5
\]
We assume only one procedure $P$.

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

Language Syntax

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

- We assume only one procedure \( P \)
- No argument passing or return expression in \( P \) (it manipulates the \textit{global} program state).

Example: Factorial

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

Language Syntax

\[
C \ := \ \text{skip} \hspace{1cm} \text{nop} \\
| \quad \text{abort} \hspace{1cm} \text{abortion} \\
| \quad x := E \hspace{1cm} \text{assignment} \\
| \quad \text{if}(G)\text{then}\{C\}\text{else}\{C\} \hspace{1cm} \text{conditional} \\
| \quad \{C\}[p]\{C\} \hspace{1cm} \text{probabilistic choice} \\
| \quad \text{call} \ P \hspace{1cm} \text{procedure call} \\
| \quad C;\ C \hspace{1cm} \text{sequence}
\]

- We assume only one procedure \( P \)
- No argument passing or return expression in \( P \) (it manipulates the global program state).

**Example:** Faulty factorial

\[
P \triangleright \text{if } (x \leq 0) \text{then } \{y := 1\} \text{else} \\
\{ x := x-1; \text{call } P; \\
x := x+1; \{y := y \cdot x\}[1/2]{\text{skip}} \}
\]
The Probabilistic Predicate Transformer — Inductive Definition

\[
\begin{align*}
wp[\text{skip}](f) &= f \\
wp[\text{abort}](f) &= 0 \\
wp[x := E](f) &= f[x/E] \\
wp[\text{if } (G) \text{ then } \{c_1\} \text{ else } \{c_2\}](f) &= [G] \cdot wp[c_1](f) + [\neg G] \cdot wp[c_2](f) \\
wp[\{c_1\} \ [p] \ \{c_2\}](f) &= p \cdot wp[c_1](f) + (1-p) \cdot wp[c_2](f) \\
wp[c_1; c_2](f) &= (wp[c_1] \circ wp[c_2])(f) \\
wp[\text{call } P] &= \sup_n wp[\text{call}_n P]
\end{align*}
\]

\(n\)-inlining of \(P\)

\(\text{call}_0 P = \text{abort}\)  \\
\(\text{call}_{n+1} P = \text{body}(P)[\text{call } P/\text{call}_n P]\)
The Expected Runtime Transformer — Inductive Definition

\[ \text{ert}[\text{skip}](t) = 1 + t \]

\[ \text{ert}[\text{abort}](t) = 0 \]

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

\[ \text{ert}[\text{if}(G)\;\text{then}\;\{c_1\}\;\text{else}\;\{c_2\}](t) = 1 + [G] \cdot \text{ert}[c_1](t) + [-G] \cdot \text{ert}[c_2](t) \]

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

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

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

“\text{ert}[\text{call } P](t) = 1 + \text{ert}[\text{body}(P)](t)”
Example 3. Reconsider the procedure $P_{rec3}$ with declaration

$$\mathcal{D}(P_{rec3}): \{\text{skip}\} \{1/2\} \{\text{call } P_{rec3}; \text{call } P_{rec3}; \text{call } P_{rec3}\}$$

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 $wp[\text{call } P, \mathcal{D}](1) \leq \varphi$. To prove this, we apply rule [wp-rec]. We must then establish the derivability claim

$$wp[\text{call } P](1) \leq \varphi \iff wp[\mathcal{D}(P_{rec3})](1) \leq \varphi$$

The derivation goes as follows:

$$wp[\mathcal{D}(P_{rec3})](1) = \{\text{def. of wp}\} \left(\frac{1}{2} \cdot wp[\text{skip}](1) + \frac{1}{2} \cdot wp[\text{call } P_{rec3}; \text{call } P_{rec3}; \text{call } P_{rec3}](1)\right)$$

$$= \{\text{def. of wp}\} \left(\frac{1}{2} + \frac{1}{2} \cdot wp[\text{call } P_{rec3}; \text{call } P_{rec3}](wp[\text{call } P_{rec3}](1))\right) \leq \{\text{assumption, monot. of wp}\} \left(\frac{1}{2} + \frac{1}{2} \cdot wp[\text{call } P_{rec3}; \text{call } P_{rec3}](\varphi)\right)$$

$$\leq \{\text{assumption, monot. of wp}\} \left(\frac{1}{2} + \frac{1}{2} \cdot wp[\text{call } P_{rec3}](\varphi)\right) = \{\text{scalab. of wp}\} \left(\frac{1}{2} + \frac{1}{2} \varphi \cdot wp[\text{call } P_{rec3}](\varphi)\right)$$

$$\leq \{\text{scalab. of wp}\} \left(\frac{1}{2} + \frac{1}{2} \varphi^2 \cdot wp[\text{call } P_{rec3}](1)\right) = \{\text{assumption, monot. of wp}\} \left(\frac{1}{2} + \frac{1}{2} \varphi^3\right) = \{\text{algebra}\} \varphi \triangleq$$
Proof rule for upper bounds

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

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

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

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

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

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

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

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

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

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

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

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

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

Proof rule for lower bounds

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

Dual rule for upper bounds is also sound
Rules from the \( \text{wp} \)—calculus can be easily adapted for the \( \text{ert} \)—calculus

**Proof rule for upper bounds**

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

**Proof rule for upper bounds**

\[
\begin{align*}
l_0 &= 0 \\
l_n + 1 &\leq \text{ert}[\text{call } P](t) \quad \vdash \quad l_{n+1} \leq \text{ert}[\text{body}(P)](t) \\
\sup_n l_n + 1 &\leq \text{ert}[\text{call } P](t)
\end{align*}
\]
Proof Rule for Mutually Recursive Procedures

\[
\begin{align*}
wp[\text{call } P_1](f_1) & \leq g_1, \ldots, wp[\text{call } P_m](f_m) \leq g_m \quad \models \quad wp[\text{body}(P_1)](f_1) \leq g_1 \\
\vdots \\
wp[\text{call } P_1](f_1) & \leq g_1, \ldots, wp[\text{call } P_m](f_m) \leq g_m \quad \models \quad wp[\text{body}(P_m)](f_m) \leq g_m \\
\hline
wp[\text{call } P_i](f_i) & \leq g_i \quad \text{for all } i = 1\ldots m
\end{align*}
\]
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 $ER(\diamond Term)$ upon reaching a terminal state in the Markov chain:

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

**Example:**

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

\[
\begin{align*}
\text{stmt}(\ell) = \text{skip} & \quad \text{succ}_1(\ell) = \ell' & \quad \text{[skip]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_1, \gamma} \langle \ell', s \rangle \\
\text{stmt}(\ell) = \text{if } (G) \{c_1\} \text{ else } \{c_2\} & \quad s \models G \quad \text{succ}_1(\ell) = \ell' & \quad \text{[if1]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_1, \gamma} \langle \ell', s \rangle \\
\text{stmt}(\ell) = \{c_1\} \{p\} \{c_2\} & \quad \text{succ}_1(\ell) = \ell' & \quad \text{[prob1]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_1, \gamma_2} \langle \ell', s \rangle \\
\text{stmt}(\ell) = \text{call } P & \quad \text{succ}_1(\ell) = \ell' & \quad \text{[call]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_1, \gamma_2 \cdot \ell'} \langle \text{init}(D(P)), s \rangle \\
\text{stmt}(\ell) = x := E & \quad \text{succ}_1(\ell) = \ell' & \quad \text{[assign]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_1, \gamma} \langle \ell', s[x \mapsto s(E)] \rangle \\
\text{stmt}(\ell) = \text{abort} & \quad \text{[abort]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_1, \gamma} \langle \ell, s \rangle \\
\text{stmt}(\ell) = \text{if } (G) \{c_1\} \text{ else } \{c_2\} & \quad s \not\models G \quad \text{succ}_2(\ell) = \ell' & \quad \text{[if2]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_1, \gamma} \langle \ell', s \rangle \\
\text{stmt}(\ell) = \{c_1\} \{p\} \{c_2\} & \quad \text{succ}_2(\ell) = \ell' & \quad \text{[prob2]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_1, \gamma_2} \langle \ell', s \rangle \\
\text{stmt}(\ell) = \text{return} & \quad \text{[return]} \\
\langle \ell, s \rangle & \xrightarrow{\ell', 1, \varepsilon} \langle \ell', s \rangle \\
\text{stmt}(\ell) = \text{terminate} & \quad \text{[terminate]} \\
\langle \ell, s \rangle & \xrightarrow{\gamma_0, 1, \gamma_0} \langle \text{Term}, s \rangle
\end{align*}
\]

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 succ_1 and the succ_2 functions.