Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms

BENJAMIN LUCIEN KAMINSKI, JOOST-PIETER KATOEN, and CHRISTOPH MATHEJA,
RWTH Aachen University
FEDERICO OLMEDO, Department of Computer Science, University of Chile

This article presents a wp–style calculus for obtaining bounds on the expected runtime of randomized algorithms. Its application includes determining the (possibly infinite) expected termination time of a randomized algorithm and proving positive almost–sure termination—does a program terminate with probability one in finite expected time? We provide several proof rules for bounding the runtime of loops, and prove the soundness of the approach with respect to a simple operational model. We show that our approach is a conservative extension of Nielson's approach for reasoning about the runtime of deterministic programs. We analyze the expected runtime of some example programs including the coupon collector's problem, a one–dimensional random walk and a randomized binary search.

CCS Concepts: • **Theory of computation → Probabilistic computation**; **Invariants**; **Pre- and post-conditions**; **Program verification**; *Logic and verification*; *Operational semantics*; Hoare logic; • **Software and its engineering → Formal software verification**;

Additional Key Words and Phrases: Probabilistic programs, expected runtime, positive almost–sure termination, weakest precondition, program verification.

ACM Reference format:
Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2018. Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms. *J. ACM* 65, 5, Article 30 (August 2018), 68 pages.
https://doi.org/10.1145/3208102

1 INTRODUCTION

The Runtime of Randomized Algorithms. Since the early days of computing, randomization has been an important tool for algorithm design. It is used to obtain an efficient randomized algorithm, possibly at the cost of sacrificing correctness with low probability. The Rabin-Miller primality test and Freivalds' matrix multiplication are prime examples of this principle. Randomization is also used to accelerate existing deterministic algorithms. Randomly selecting the pivot in Hoare's quicksort algorithm lowers the quadratic worst-case runtime to $\mathcal{O}(N\cdot \log N)$, where N is the

This work was supported by the Excellence Initiative of the German federal and state government as well as the Research Training Group 2236 UnRAVeL funded by the German Research Foundation.

Authors' addresses: B. L. Kaminski, J-P. Katoen, and C. Matheja, Lehrstuhl für Informatik 2, RWTH Aachen, Ahornstrasse 55, 52056 Aachen, Germany; emails: {benjamin.kaminski, katoen, matheja}@cs.rwth-aachen.de; F. Olmedo, Department of Computer Science, University of Chile, Av. Beauchef 851, Santiago, Chile; email: folmedo@dcc.uchile.cl.

https://doi.org/10.1145/3208102

size of the input. Furthermore, some problems inherently require randomized solutions, e.g., self-stabilization in anonymous distributed systems.

Randomized algorithms are conveniently described by probabilistic programming languages that (on top of the usual language constructs) offer the possibility of sampling values from a probability distribution. Sampling can be used in assignments as well as in Boolean guards. The interest in probabilistic computations has been rapidly growing recently. This is mainly due to their wide applicability [16]. Randomized algorithms are, e.g., used in security to describe cryptographic constructions and security notions. In machine learning, probabilistic programs are used to describe distribution functions that are analyzed using Bayesian inference.

The runtime of a randomized algorithm is affected by the outcomes of the samplings (aka coin tosses). Technically speaking, the runtime is a random variable, assuming value t_1 with probability p_1, t_2 with probability p_2, and so on. An important measure is the *average* or *expected* runtime $\sum_i p_i\, t_i$.[1] In that, one does not average with respect to a distribution of program inputs but rather with respect to the randomness that is inherent in the algorithm.

Expected Runtimes Are Intricate. Reasoning about the expected runtime of randomized algorithms is subtle and full of nuances. Let us illustrate this by discussing three phenomena of randomized algorithms:

(1) They may have diverging runs but have a finite expected runtime.
(2) They may terminate with probability one but have an infinite expected runtime.
(3) Having a finite expected runtime is not preserved under sequencing.

(1) A single diverging run of an ordinary, i.e., nonprobabilistic, algorithm yields the program to have an infinite runtime. This is not true for randomized algorithms. They may admit arbitrarily long and even infinite runs while still having a finite expected runtime. The program

```
C_geo :   $b := 1;$
            while($b = 1$){
              $b :\approx 1/2 \cdot \langle 0 \rangle + 1/2 \cdot \langle 1 \rangle$ }
```

keeps flipping a fair coin until observing the first heads (represented by 0). The program C_{geo} admits arbitrarily long runs, since the probability of not seeing a heads in the first n trials is nonzero. This holds for every n. It even admits a nonterminating run (occurring with probability zero), namely, the one in which the outcome of all coin flips is tails. The runtime of the program C_{geo}, however, is geometrically distributed and therefore its expected runtime is finite: on average, it terminates after two loop iterations. It is worth mentioning that the decision problem "does a program terminate with probability one in finite expected time (on all inputs)?" is Π_3^0-complete in the arithmetical hierarchy, and thus strictly harder than the universal halting problem for ordinary programs [27].

(2) The program C_{geo} terminates with probability one and its expected time until termination is finite. For ordinary sequential programs, termination always implies finite runtime. For probabilistic programs this is not always true. Consider the program

```
C_rw :   $x := 10;$
           while($x > 0$){
             $x :\approx 1/2 \cdot \langle x-1 \rangle + 1/2 \cdot \langle x+1 \rangle$},
```

which models a one-dimensional random walk of a particle: starting from position 10, in each step the particle moves randomly one step to the left or one step to the right, until reaching position 0.

[1] If the program does not terminate with some probability greater than zero, its expected runtime is infinity.

The particle reaches position 0 with probability one, but doing so requires infinitely many steps on average (cf. [26, Chapter 3.7.3]).

(3) In the classical setting, running two programs with finite runtime in a row yields a program with a finite runtime. For probabilistic programs this closure property breaks down. Consider the pair of programs

```
$C_1$ :   $x := 1;\ b := 1;$
            while($b = 1$){
              $b :\approx 1/2 \cdot \langle 0 \rangle + 1/2 \cdot \langle 1 \rangle;$
              $x := 2x$}.
```

```
$C_2$ : while($x > 0$){
            $x := x - 1$}
```

As the loop in C_1 terminates on average in two iterations, it has a finite expected runtime. From any initial state in which x is nonnegative, C_2 makes x iterations, and thus its expected runtime is finite too. However, the program $C_1; C_2$ has an *infinite* expected runtime—even though it almost surely terminates; i.e. it terminates with probability one. This is intuitively due to the fact that the expected value of x after termination of C_1 is infinite and C_2 needs x steps to terminate.

Determining Expected Runtimes. Bounds on the expected runtime of randomized algorithms are typically obtained using classical probability theory, mostly with arguments relying on random variable expectations or martingales [15, 39]. These runtime bounds are nonetheless obtained partially following an ad hoc reasoning, which, moreover, usually takes for granted nontrivial relationships between the involved random variables. This constitutes a somewhat deficient proof methodology that often yields incomplete proof arguments. This article proposes an alternative using formal reasoning as typically used in deductive verification.

A naive approach toward a rigorous, formal reasoning about expected runtimes equips the program at hand with a runtime counter, say, *rc*. The variable *rc* is initially set to 0 and incremented for each basic operation that consumes time. The expected value of runtime counter *rc* is then obtained using existing verification techniques for randomized algorithms, such as Kozen's probabilistic propositional dynamic logic (PPDL) [32] or the weakest pre-expectation calculus by McIver and Morgan [36]. This approach is, however, *unsound* as the expected value of counter *rc* may not coincide with the expected runtime.

Let us illustrate this by an example: applying the above principle to the certainly diverging program while(*true*){skip} results (assuming skip consumes one time unit) in the program

```
$C_{div}$ : $rc := 0$
          while(true){
              skip; $rc := rc + 1$}.
```

PPDL and the weakest pre-expectation calculus would both yield zero as the expected value of *rc*: since no run of C_{div} terminates, the average value of *rc* upon termination is an empty sum, i.e. zero. However, the program's expected runtime is infinite as every program run executes a statement that consumes time infinitely often. This is not just a corner case: in fact, the expected value of *rc* may assume an arbitrary positive integer value different from the actual expected runtime. Consider the program

```
$C_{halfdiv}$ : $rc := 0;$
              $b :\approx 1/2 \cdot \langle 0 \rangle + 1/2 \cdot \langle 1 \rangle;$
              $\ell := 2k;$                          // consume $2k$ time units
```

```
while($\ell > 0$){
    $\ell := \ell - 1$;  $rc := rc + 1$};
while ($b = 1$){
    skip;  $rc := rc + 1$}
```

that first flips a fair coin to set the variable b to either 0 or 1. Afterward, the first while-loop performs $2k$ loop iterations before the second while-loop either terminates (if $b = 0$) or diverges (if $b = 1$). The variable rc counts the total number of loop iterations. Using the weakest pre-expectation calculus (or PPDL), one can show that for positive k, the expected value of rc for program $C_{halfdiv}$ is upper bounded by $1/2 \cdot 2\lceil k \rceil + 1/2 \cdot 0 = \lceil k \rceil$, although the actual expected runtime of $C_{halfdiv}$ is infinite for any k.

The Approach of This Article. The inability of the weakest pre-expectation calculus and PPDL to soundly reason about expected runtimes, the manifold intricacies in the runtime analyses in the presence of randomization, and the deficiency of various ad hoc runtime analyses in the literature necessitate a more rigorous, systematic, and compelling approach for the runtime analysis of randomized algorithms. This article proposes a technique based on *formal program verification*. This technique derives runtime claims from first principles only. It consists of a wp-style calculus à la Dijkstra [12]. In a similar vein to Dijkstra's predicate transformers, our wp-style calculus uses *runtime transformers*. The core of this calculus is the transformer ert:

$$\mathrm{ert}[C](t)(\sigma)$$

gives the expected runtime of program C when started in initial state σ, under the assumption that t captures the expected runtime of the computation following C. In particular, the expected runtime of program C on initial state σ is given by $\mathrm{ert}[C](\mathbf{0})(\sigma)$, where $\mathbf{0}$ is the constantly zero runtime.

Our calculus is defined over a simple probabilistic imperative language with recursive procedures. For most control structures, it is defined in a straightforward compositional manner. The action of the transformer on guarded loops and recursive procedures is given using fixed-point techniques. To avoid the tedious reasoning about such fixed points and enhance the calculus's usability, we provide *invariant-based proof rules* that establish bounds on the expected runtime of loops and of recursive programs.

At the theoretical level, we validate our wp-style calculus in two ways: First, we show that the calculus corresponds to a simple, intuitive operational model of probabilistic programs based on Markov chains. Second, we show that the calculus is a conservative extension of Nielson's approach for reasoning about the runtime of ordinary, nonprobabilistic programs [41]. On the more practical side, we use our calculus to perform a formal runtime analysis of a one-dimensional random walk, the coupon collector's problem [37], and a Sherwood variant of the binary search algorithm [35].

We stress two important assets of our approach: First, our calculus for expected runtimes is amenable to a large degree of automation. For several cases, loop invariants can be synthesized from previously provided templates and our proof rules allow for mechanical verification of proposed invariants. An implementation of our calculus in the theorem prover Isabelle/HOL has recently been reported [23], certifying all the theoretical results in this article.

A second asset is that our calculus enables determining whether the expected runtime of a randomized algorithm (for all possible inputs) is finite or not. In combination with almost-sure termination, this is a very relevant property. To the best of our knowledge, this is the first formal verification framework that can handle both (universal) almost-sure termination (does a program terminate with probability one on every input?) and (universal) positive almost-sure termination (does a program terminate with probability one in finite expected time on every inputs?).

Main Contributions of this Article. To summarize, this article presents a calculus for reasoning about the—finite or possibly infinite—expected runtime of randomized algorithms

(1) with a set of invariant-based proof rules for obtaining bounds on the expected runtime of loops and recursive procedures,
(2) which corresponds to a simple operational program model using Markov chains,
(3) which conservatively extends a calculus [41] for runtime analysis of ordinary programs, and
(4) which is applied to analyze the runtime of the coupon collector's problem, a one-dimensional random walk, and a randomized binary search.

Organization of the Article. Section 2 defines our probabilistic programming language. Section 3 presents the transformer ert and studies its elementary properties. Section 4 presents proof rules for obtaining upper and lower bounds on the expected runtime of loops. Section 5 shows that the ert-transformer coincides with the expected runtime in a Markov chain that acts as an operational program model. Section 6 proves that the ert-transformer is a conservative extension of Nielson's approach for obtaining upper bounds on deterministic programs. Section 7 extends the ert-calculus with recursion. Section 8 establishes a connection between transformer ert and the weakest precondition semantics of probabilistic programs. Section 9 discusses three case studies in detail. Section 10 summarizes related work, and Section 11 concludes.

The proofs of the results that do not proceed by a tedious induction on the program structure are provided in the article body. All inductive proofs as well as the detailed calculations for the case studies are provided in the appendix. The material presented in this article unifies and extends [28] and [44].

2 A PROBABILISTIC PROGRAMMING LANGUAGE FOR RANDOMIZED ALGORITHMS

In this section, we present the probabilistic programming language used throughout the article, together with its runtime model. For ease of presentation, we treat nonrecursive programs first and extend our calculus with recursion in Section 7. To model randomized algorithms, we employ a standard imperative language à la Dijkstra's Guarded Command Language [12] with two distinguished features: We allow distribution expressions in assignments and guards to be probabilistic. For instance, we allow for probabilistic assignments like

$$y :\approx \texttt{Unif}[1\ldots x],$$

which endows variable y with a uniform distribution in the interval $[1\ldots x]$. Notice that x is another program variable, so the resulting distribution of y depends on the current program state. We also allow for a program like

```
$x := 0;$
while $(p \cdot \langle \mathsf{true} \rangle + (1-p) \cdot \langle \mathsf{false} \rangle)$ {
    $x := x + 1$},
```

which uses a probabilistic loop guard to simulate a geometric distribution with success probability p; i.e., the loop guard evaluates to true with probability p and to false with the remaining probability $1-p$.

Formally, the set of *probabilistic programs* pGCL is given by the grammar

$$\begin{array}{rlll} C & ::= & \texttt{empty} & \text{empty program} \\ & \mid & \texttt{skip} & \text{effectless operation} \\ & \mid & \texttt{halt} & \text{immediate termination} \\ & \mid & x :\approx \mu & \text{probabilistic assignment} \\ & \mid & C; C & \text{sequential composition} \\ & \mid & \texttt{if}\ (\xi)\ \{C\}\ \texttt{else}\ \{C\} & \text{probabilistic conditional choice} \\ & \mid & \texttt{while}\ (\xi)\ \{C\} & \text{probabilistic guarded while-loop} \end{array}$$

Here x represents a *program variable* in Var, μ a *distribution expression* in DExp, and ξ a distribution expression over the truth values, i.e., a *probabilistic guard*, in DExp. We assume distribution expressions in DExp to represent discrete probability distributions with a (possibly *infinite*) support of total probability mass 1. Let $p_1 \cdot \langle a_1 \rangle + \cdots + p_n \cdot \langle a_n \rangle$ denote the distribution expression that assigns probability p_i to a_i. For instance, the distribution expression $1/2 \cdot \langle \mathsf{true} \rangle + 1/2 \cdot \langle \mathsf{false} \rangle$ represents the toss of a fair coin. Deterministic expressions over program variables such as $x - y$ or $x - y > 8$ are special instances of distribution expressions; they are understood as Dirac probability distributions that assign the total probability mass, i.e., one, to a single point.

To describe the effect of the different language constructs, we use some preliminaries. A *program state* σ is a mapping from program variables to values in Val. Let

$$\Sigma \triangleq \{\sigma \mid \sigma : \mathsf{Var} \to \mathsf{Val}\}$$

be the set of all program states. We assume an interpretation function $[\![\,\cdot\,]\!] : \mathsf{DExp} \to (\Sigma \to \mathcal{D}(\mathsf{Val}))$ for distribution expressions, with $\mathcal{D}(\mathsf{Val})$ being the set of discrete probability distributions over Val. For $\mu \in \mathsf{DExp}$, $[\![\mu]\!]$ maps each program state to a probability distribution of values. Let $[\![\mu : v]\!]$ be a shorthand for the function mapping each program state σ to the probability that distribution $[\![\mu]\!](\sigma)$ assigns to value v, i.e.,

$$[\![\mu : v]\!](\sigma) \triangleq \mathsf{Pr}_{[\![\mu]\!](\sigma)}(v),$$

where Pr denotes the probability operator on distributions over values.

The effect of pGCL program constructs and their assumed timing is as follows:

- `empty` has no effect and its execution consumes no time.
- `skip` has also no effect but consumes, in contrast to `empty`, one unit of time.
- `halt` aborts any further program execution and consumes no time.
- $x :\approx \mu$ is a probabilistic assignment that samples a value from $[\![\mu]\!]$ and assigns it to variable x. The sampling and assignment consume (altogether) one unit of time.
- $C_1; C_2$ is the sequential composition of programs C_1 and C_2; i.e. first C_1 is executed, then C_2. The composition itself consumes no additional time.
- $\texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\}$ is a probabilistic conditional branching: with probability $[\![\xi : \mathsf{true}]\!]$ program C_1 is executed, whereas with probability $[\![\xi : \mathsf{false}]\!] = 1 - [\![\xi : \mathsf{true}]\!]$ program C_2 is executed. Evaluating (or more rigorously, sampling a value from) the probabilistic guard requires an additional unit of time.
- $\texttt{while}\ (\xi)\ \{C\}$ is a probabilistic while loop: with probability $[\![\xi : \mathsf{true}]\!]$ the loop body C is executed followed by a recursive execution of the loop, whereas with probability $[\![\xi : \mathsf{false}]\!]$ the loop terminates immediately. As for conditionals, each evaluation of the guard consumes one unit of time.

Alternative Runtime Models. We stress that the above runtime model is a design decision for the sake of concreteness. All our development can be easily adapted to capture alternative models.

These include, for instance, the model where only the number of assignments in a program run or the model where only the number of loop iterations are of relevance. We can also capture more fine-grained models, where for instance the runtime of an assignment depends on some notion of *size* of the distribution expression being sampled.

Example 2.1 (Race between tortoise and hare). To illustrate the use of our programming language, consider the following program adopted from [9]:

```
$h :\approx 0; t :\approx 30;$
while $(h \leq t)$ {
        if $\left(\nicefrac{1}{2} \cdot \langle \mathsf{true} \rangle + \nicefrac{1}{2} \cdot \langle \mathsf{false} \rangle\right)${
                $h :\approx h +$ Unif[0...10] }
        else {empty};
        $t :\approx t + 1$ }.
```

It models a race between a tortoise and a hare; the variables h and t represent their respective positions. The tortoise starts with a lead of 30 and in each round (i.e., in each loop iteration) advances one step forward. The hare with probability $\nicefrac{1}{2}$ advances a random number of steps between 0 and 10 (governed by a uniform distribution) and with the remaining probability remains still. The race ends when the hare passes the tortoise.

Regarding the runtime, the program requires two units of time for the initial assignments. In every loop iteration, the program consumes either three or four units of time: it always takes one unit of time to evaluate the loop guard, evaluate the probabilistic conditional, and update variable t, respectively. If the probabilistic conditional is evaluated to true, an additional unit of time is consumed to update the value of variable h. △

We conclude this section by fixing some notational conventions. To keep our program notation consistent with standard usage, we write $x := \mu$ instead of $x :\approx \mu$ whenever μ represents a Dirac distribution given by a deterministic expression over program variables. For instance, in the program in Example 2.1 above, we shall write $t := t + 1$ instead of $t :\approx t + 1$. Likewise, when ξ is a probabilistic guard given as a deterministic Boolean expression over program variables, we use $[\![\xi]\!]$ to denote $[\![\xi : \mathsf{true}]\!]$ and $[\![\neg\xi]\!]$ to denote $[\![\xi : \mathsf{false}]\!]$. For instance, we write $[\![b = 0]\!]$ instead of $[\![b = 0 : \mathsf{true}]\!]$.

3 A CALCULUS OF EXPECTED RUNTIMES

Our goal is to associate to any program C a function that maps each state σ to the average or expected runtime of executing C on initial state σ. To model these functions, we use the function space of *runtimes:*

$$\mathbb{T} \triangleq \left\{ t \,\middle|\, t : \Sigma \to \mathbb{R}_{\geq 0}^{\infty} \right\} .$$

Here, $\mathbb{R}_{\geq 0}^{\infty}$ represents the set of nonnegative real numbers extended with ∞. Let $\mathbf{k}$ denote the constant runtime $\lambda\sigma\text{.}\, k$ for $k \in \mathbb{R}_{\geq 0}^{\infty}$.

We express the expected runtime of programs in a continuation-passing style by means of the transformer

$$\mathsf{ert}[\,\cdot\,] : \mathsf{pGCL} \to (\mathbb{T} \to \mathbb{T}).$$

Concretely, $\mathsf{ert}\,[C]\,(t)\,(\sigma)$ gives the expected runtime of executing program C on initial state σ assuming that t captures the runtime of the computation that follows C. The function t is usually referred to as the *continuation* and we can think of it as being evaluated in the final states that are

Table 1. Rules for Defining the Expected Runtime Transformer ert

C	$\mathbf{ert}\,[C]\,(t)$
`empty`	t
`skip`	$\mathbf{1} + t$
`halt`	$\mathbf{0}$
$x :\approx\ \mu$	$\mathbf{1} + \lambda\sigma\bullet\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\bullet\, t[x/v](\sigma))$
$C_1; C_2$	$\mathsf{ert}[C_1](\mathsf{ert}[C_2](t))$
`if` (ξ) $\{C_1\}$ `else` $\{C_2\}$	$\mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C_1](t) + [\![\xi : \mathsf{false}]\!] \cdot \mathsf{ert}[C_2](t)$
`while` (ξ) $\{C'\}$	$\mathsf{lfp}\, X \cdot \mathbf{1} + [\![\xi : \mathsf{false}]\!]\bullet\, t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C'](X)$

1 is the constant runtime $\lambda\sigma\bullet\, 1$; $\mathsf{E}_\mu(h) \triangleq \sum_v \mathsf{Pr}_\mu(v) \cdot h(v)$ represents the expected value of (random variable) h w.r.t. distribution μ; $t[x/v] \triangleq \lambda\sigma\bullet\, t(\sigma[x/v])$, where $\sigma[x/v]$ is the state obtained by updating in σ the value of x to v; finally $\mathsf{lfp}\, X\bullet\, F(X)$ represents the least fixed point of transformer $F : \mathbb{T} \to \mathbb{T}$ with respect to the pointwise ordering on $\mathbb{T}$.

reached upon termination of C. Thus, ert $[C]$ $(\mathbf{0})$ (σ) gives the plain expected runtime of executing program C on initial state σ.

The transformer ert is defined by induction on the structure of C following the rules in Table 1. The rules correspond to the runtime model introduced in Section 2. That is, $\mathsf{ert}[C](\mathbf{0})$ captures the expected number of assignments, guard evaluations, and `skip` statements. Most rules in Table 1 are self-explanatory. ert[`empty`] behaves as the identity since `empty` does not modify the program state and its execution consumes no time. On the other hand, ert[`skip`] adds one unit of time to any continuation since this is the time required by the execution of `skip`. ert[`halt`] yields always the constant runtime $\mathbf{0}$ since `halt` aborts any subsequent program execution (making their runtime irrelevant) and consumes no time. The definition of ert on random assignments is more involved: $\mathsf{ert}[x :\approx\ \mu](t)(\sigma) = 1 + \sum_v \mathsf{Pr}_{[\![\mu]\!](\sigma)}(v) \cdot t(\sigma[x/v])$ is obtained by adding one unit of time (due to the distribution sampling and assignment of its outcome) to the sum of the runtime of each possible subsequent execution, weighted according to their probabilities.

As for the composite statements, $\mathsf{ert}[C_1; C_2]$ applies $\mathsf{ert}[C_1]$ to the expected runtime obtained from the application of $\mathsf{ert}[C_2]$ to the continuation t. ert[`if` (ξ) $\{C_1\}$ `else` $\{C_2\}$] adds one unit of time (for the guard evaluation) to the weighted sum of the runtime of the two branches.

The ert of a loop is given as the least fixed point (with respect to the pointwise order in $\mathbb{T}$) of a runtime transformer $F : \mathbb{T} \to \mathbb{T}$, defined in terms of the runtime of the loop body. To guarantee the existence of such a fixed point, we use a standard argument (see, e.g., [50, Ch. 5]): we endow $\mathbb{T}$ with the structure of an ω-complete partial order (ω-cpo for short) and we prove that F is continuous. Since the transformer F is used repeatedly in the rest of our development, we use the following notation:

Definition 3.1 (Characteristic functional of a loop). Given loop `while` (ξ) $\{C\}$ and runtime $t \in \mathbb{T}$, let

$$F_t^{\langle \xi, C \rangle} : \mathbb{T} \to \mathbb{T}, \quad X \mapsto \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C](X)$$

be the *characteristic functional* of the loop with respect to (continuation) t. △

When the loop is understood from the context, we simply write F_t for $F_t^{\langle \xi, C \rangle}$. Using this definition, the ert of a loop can be recast as

$$\mathsf{ert}[\texttt{while}\ (\xi)\ \{C\}](t) \ = \ \mathsf{lfp}\, F_t^{\langle \xi, C \rangle}.$$

The two steps that we take to guarantee the existence of $\mathsf{lfp}\, F_t^{\langle \xi, C\rangle}$ are as follows: First, we endow $\mathbb{T}$ with the structure of an ω-cpo: runtimes are ordered pointwise, i.e., $t_1 \preceq t_2$ if and only if $t_1(\sigma) \leq t_2(\sigma)$ for all $\sigma \in \Sigma$; the supremum of ω-chains is also defined pointwise, i.e., for $t_1 \preceq t_2 \preceq \cdots$, $\sup_n t_n \triangleq \lambda\sigma\text{.}\ \sup_n t_n(\sigma)$; finally, the bottom element of the ω-cpo is the constant runtime 0. Second, we prove that the characteristic functional $F_t^{\langle \xi, C\rangle}$ is continuous. The Kleene Fixed Point Theorem (Theorem A.3) ensures the existence of $\mathsf{lfp}\, F_t^{\langle \xi, C\rangle}$. It follows that the action of ert on loops is well defined. The continuity of $F_t^{\langle \xi, C\rangle}$ follows immediately from the continuity of $\mathsf{ert}[C]$, established below.

Lemma 3.2 (Continuity of ert). *For every program $C \in$ pGCL, $\mathsf{ert}[C] : \mathbb{T} \to \mathbb{T}$ is continuous; i.e., for every ω-chain of runtimes $t_0 \preceq t_1 \preceq \cdots$ we have*

$$\mathsf{ert}[C](\sup_n t_n) \;=\; \sup_n \mathsf{ert}[C](t_n).$$

Proof. By induction on the structure of C; see Appendix B.1. □

We now illustrate the use of the ert transformer by analyzing the runtime of a program that simulates a truncated geometric distribution.

Example 3.3 (Truncated geometric distribution). Program C_{trunc} repeatedly flips a fair coin until observing the first heads (represented by true in the probabilistic guards) or completing the second unsuccessful trial:

```
C_trunc :  if (1/2 ·⟨true⟩ + 1/2 ·⟨false⟩) {
                   succ := true }
           else {
                   if (1/2 ·⟨true⟩ + 1/2 ·⟨false⟩) {
                           succ := true }
                   else {
                           succ := false } }.
```

The calculation of the expected runtime $\mathsf{ert}[C_{trunc}](\mathbf{0})$ of the program goes as follows:

$$\begin{aligned}
\mathsf{ert}[C_{trunc}](\mathbf{0}) &= \mathbf{1} + \tfrac{1}{2} \cdot \mathsf{ert}[succ := \mathsf{true}](\mathbf{0}) \\
&\qquad + \tfrac{1}{2} \cdot \mathsf{ert}[\mathtt{if}\ (\ldots)\ \{succ{:=}\mathsf{true}\}\ \mathtt{else}\ \{succ{:=}\mathsf{false}\}](\mathbf{0}) \\
&= \mathbf{1} + \tfrac{1}{2} \cdot \mathsf{ert}[succ := \mathsf{true}](\mathbf{0}) \\
&\qquad + \tfrac{1}{2} \cdot \Big(\mathbf{1} + \tfrac{1}{2} \cdot \mathsf{ert}[succ := \mathsf{true}](\mathbf{0}) + \tfrac{1}{2} \cdot \mathsf{ert}[succ := \mathsf{false}](\mathbf{0})\Big) \\
&= \mathbf{1} + \tfrac{1}{2} \cdot (\mathbf{1} + 0) + \tfrac{1}{2} \cdot \Big(\mathbf{1} + \tfrac{1}{2} \cdot (\mathbf{1} + 0) + \tfrac{1}{2} \cdot (\mathbf{1} + 0)\Big) \;=\; \tfrac{5}{2}.
\end{aligned}$$

Therefore, the execution of C_{trunc} takes, on average, 2.5 units of time. △

Note that the calculation of the expected runtime in the example above is straightforward as the program is loop-free. Computing the runtime of loops requires the calculation of least fixed points, which is generally not feasible in practice. To circumvent this, in the next section we present proof rules based on loop invariants.

The ert transformer possesses several algebraic properties:

Theorem 3.4 (Basic properties of the ert transformer). *For any program $C \in$ pGCL, any constant runtime $\boldsymbol{k}$ with $k \in \mathbb{R}_{\geq 0}$ and any runtimes $t, t' \in \mathbb{T}$, it holds:*

Monotonicity: $t \preceq t'$ *implies* $\mathsf{ert}[C](t) \preceq \mathsf{ert}[C](t')$;

Constant propagation: $\mathsf{ert}[C](\mathbf{k} + t) = \mathbf{k} + \mathsf{ert}[C](t)$, *provided C is* `halt`*–free;*

Preservation of ∞: $\mathsf{ert}[C](\infty) = \infty$, *provided C is* `halt`*–free.*

Proof. Monotonicity follows from continuity (see Lemma 3.2). For the proof of constant propagation, see Appendix B.2. Finally, preservation of infinity follows from monotonicity and constant propagation since together they entail

$$\mathsf{ert}[C](\infty) \succeq \mathsf{ert}[C](\mathbf{k}) = \mathbf{k} + \mathsf{ert}[C](0) \succeq \mathbf{k}$$

for every $k \in \mathbb{R}_{\geq 0}$. This immediately yields $\mathsf{ert}[C](\infty) = \infty$. □

The ert transformer is not linear in general, but it satisfies the weaker properties of subadditivity and subscalability, which are discussed in Section 8.

4 EXPECTED RUNTIME OF LOOPS

As mentioned earlier, reasoning about the runtime of loop-free programs involves mostly syntactic reasoning. The runtime of a loop, however, is given in terms of a least fixed point. It can thus be obtained by fixed-point iteration, but the fixed point need not be reached within a finite number of iterations. To overcome this problem, we study invariant-based proof rules for approximating the runtime of loops.

We present two families of proof rules that differ in the kind of invariants they build upon. In Section 4.1, we present a proof rule that rests on the presence of an invariant approximating the entire runtime of a loop in a global manner, while in Section 4.2 we present two proof rules that each rely on a parameterized invariant that approximates the runtime of a loop in an incremental fashion. In Section 4.3, we discuss how to tighten the runtime bounds yielded by any of these proof rules.

4.1 Proof Rule Based on Global Invariants

Our first proof rule allows for bounding the expected runtime of loops from above and rests on the notion of *upper invariants*:

Definition 4.1 (Upper invariants). Runtime $I \in \mathbb{T}$ is an *upper invariant of loop* `while` (ξ) $\{C\}$ `with respect to t` *if and only if*

$$F_t^{\langle \xi, C \rangle}(I) \preceq I. \qquad \triangle$$

Such an upper invariant readily establishes an upper bound of the loop's runtime:

Theorem 4.2 (Upper Bounds from Upper Invariants). *If $I \in \mathbb{T}$ is an upper invariant of* `while` (ξ) $\{C\}$ *with respect to t, then*

$$\mathsf{ert}[\texttt{while}\ (\xi)\ \{C\}](t) \preceq I.$$

Proof. The theorem follows by an application of Park's Theorem (Theorem A.4), which in our case, given that $F_t^{\langle \xi, C \rangle}$ is continuous (see Lemma 3.2), states that

$$F_t^{\langle \xi, C \rangle}(I) \preceq I \quad \text{implies} \quad \mathsf{lfp}\, F_t^{\langle \xi, C \rangle} \preceq I.$$

The left-hand side of the implication is equivalent to I being an upper invariant, while the right-hand side is equivalent to $\mathsf{ert}[\texttt{while}\ (\xi)\ \{C\}](f) \preceq I$. □

Example 4.3 (Geometric distribution). Consider the loop

$$\begin{aligned} C_{\mathsf{geo}}:\ & \texttt{while}\,(c = 1)\,\{ \\ & \qquad c :\approx {}^1\!/\!{}_2 \cdot \langle 0 \rangle + {}^1\!/\!{}_2 \cdot \langle 1 \rangle\,\} \end{aligned}$$

whose runtime is geometrically distributed. Its characteristic functional with respect to continuation $t = 0$ is

$$F_{\mathbf{0}}(X) \;=\; \mathbf{1} + [\![c \neq 1]\!] \cdot \mathbf{0} + [\![c = 1]\!] \cdot \mathsf{ert}[c :\approx {}^1\!/\!{}_2 \cdot \langle 0 \rangle + {}^1\!/\!{}_2 \cdot \langle 1 \rangle](X).$$

By the calculations below, we verify that $I = \mathbf{1} + [\![c = 1]\!] \cdot \mathbf{4}$ is an upper invariant of the loop (with respect to $\mathbf{0}$):

$$\begin{aligned} F_0(I) \;&=\; \mathbf{1} + [\![c \neq 1]\!] \cdot \mathbf{0} + [\![c = 1]\!] \cdot \mathsf{ert}[c :\approx {}^1\!/\!{}_2 \cdot \langle 0 \rangle + {}^1\!/\!{}_2 \cdot \langle 1 \rangle](I) \\ &=\; \mathbf{1} + [\![c = 1]\!] \cdot \Big(\mathbf{1} + \tfrac{1}{2} \cdot I[c/0] + \tfrac{1}{2} \cdot I[c/1]\Big) \\ &=\; \mathbf{1} + [\![c = 1]\!] \cdot \Big(\mathbf{1} + \tfrac{1}{2} \cdot \underbrace{(\mathbf{1} + [\![0 = 1]\!] \cdot \mathbf{4})}_{=\mathbf{1}} + \tfrac{1}{2} \cdot \underbrace{(\mathbf{1} + [\![1 = 1]\!] \cdot \mathbf{4})}_{=\mathbf{5}}\Big) \\ &=\; \mathbf{1} + [\![c = 1]\!] \cdot \mathbf{4} \;=\; I \;\preceq\; I. \end{aligned}$$

By applying Theorem 4.2, we obtain

$$\mathsf{ert}\left[C_{\mathsf{geo}}\right](\mathbf{0}) \;\preceq\; \mathbf{1} + [\![c = 1]\!] \cdot \mathbf{4}.$$

In words, the expected runtime of C_{geo} is at most $1 + 4 = 5$ from any initial state where $c = 1$ and at most $1 + 0 = 1$ from any other state. △

Notice that if the loop body is itself loop-free, as in the above example, verifying that some $I \in \mathbb{T}$ is an upper invariant is usually fairly easy. Inferring the invariant, in contrast, is one of the most involved parts of the verification effort.

The invariant-based technique to reason about the runtime of loops presented in Theorem 4.2 is complete in the sense that there always exists an upper invariant that establishes the exact runtime of the loop at hand:

Theorem 4.4. *There exists an upper invariant I of* $\texttt{while}\,(\xi)\,\{C\}$ *with respect to t such that* $\mathsf{ert}[\texttt{while}\,(\xi)\,\{C\}](t) = I$.

Proof. It suffices to show that $\mathsf{ert}[\texttt{while}\,(\xi)\,\{C\}](t)$ itself is an upper invariant of the loop. Since $\mathsf{ert}[\texttt{while}\,(\xi)\,\{C\}](t) = \mathsf{lfp}\, F_t^{\langle \xi, C \rangle}$, this amounts to showing that

$$F_t^{\langle \xi, C \rangle}\left(\mathsf{lfp}\, F_t^{\langle \xi, C \rangle}\right) \;\preceq\; \mathsf{lfp}\, F_t^{\langle \xi, C \rangle},$$

which holds by definition of lfp. □

Intuitively, the proof of this theorem shows that $\mathsf{ert}[\texttt{while}\,(\xi)\,\{C\}](t)$ itself is the tightest upper invariant that the loop admits.

4.2 Proof Rules Based on Parameterized Invariants

We now study a second family of proof rules that builds on the notion of ω-invariants for bounding the runtime of loops from both above and below.

Definition 4.5 (ω-Invariants). Let $n \in \mathbb{N}$. The parameterized runtime $I_n \in \mathbb{T}$ is a *lower ω-invariant* of loop $\texttt{while}\,(\xi)\,\{C\}$ with respect to t if and only if

$$I_0 \;\preceq\; F_t^{\langle \xi, C \rangle}(\mathbf{0}) \quad \text{and} \quad I_{n+1} \;\preceq\; F_t^{\langle \xi, C \rangle}(I_n), \qquad \text{for all } n \geq 0.$$

Dually, I_n is an *upper ω-invariant* if and only if

$$F_t^{\langle \xi, C\rangle}(\mathbf{0}) \preceq I_0 \quad \text{and} \quad F_t^{\langle \xi, C\rangle}(I_n) \preceq I_{n+1}, \qquad \text{for all } n \geq 0. \qquad \triangle$$

Intuitively, a lower (upper) ω-invariant I_n represents a lower (upper) bound for the expected runtime of those program runs that finish within $n+1$ iterations, weighted according to their probabilities. Therefore, we can use the asymptotic behavior of I_n to approximate from below (above) the expected runtime of the entire loop.

Theorem 4.6 (Bounds from ω-invariants).

(1) *If I_n is a lower ω-invariant of loop* `while` (ξ) $\{C\}$ *with respect to t and the limit* $\lim_{n\to\infty} I_n$ *exists*[2], *then*

$$\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t) \succeq \lim_{n\to\infty} I_n.$$

(2) *If I_n is an upper ω-invariant of loop* `while` (ξ) $\{C\}$ *with respect to t and the limit* $\lim_{n\to\infty} I_n$ *exists, then*

$$\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t) \preceq \lim_{n\to\infty} I_n.$$

Proof. We prove only the case of lower ω-invariants since the other case follows by a dual argument. Let F_t be the characteristic functional of the loop with respect to t and let F_t^n denote the n-fold composition of F_t with itself, i.e., the function $F_t \circ \ldots \circ F_t$ (n times). By the Kleene Fixed Point Theorem (Theorem A.3 in Appendix A), $\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t) = \sup_n F_t^n(0)$, and since $F_t^0(0) \preceq F_t^1(0) \preceq \cdots$ forms an ω-chain, by the Monotone Sequence Theorem (Theorem A.2 in Appendix A), $\sup_n F_t^n(0) = \lim_{n\to\infty} F_t^n(0)$. Then the result follows from showing that $F_t^{n+1}(0) \succeq I_n$. We prove this by induction on n. The base case (i.e., $n = 1$) holds, since $F_t^1(0) \succeq I_0$ holds by the assumption that I_n is a lower ω-invariant. For the inductive case we reason as follows:

$$F_t^{n+2}(0) = F_t\left(F_t^{n+1}(0)\right) \succeq F_t(I_n) \succeq I_{n+1}.$$

Here the first inequality follows by I.H. and monotonicity of F_t (recall that $\mathsf{ert}[C]$ is monotonic by Theorem 3.4), while the second inequality holds by the assumption that I_n is a lower ω-invariant. □

Example 4.7 (Lower bounds for C_{geo}). Reconsider loop C_{geo} from Example 4.3. We use Theorem 4.6 (1) to show that $\mathbf{1} + [\![c = 1]\!] \cdot \mathbf{4}$ is also a lower bound of its runtime. Let us first show that $I_n = \mathbf{1} + [\![c = 1]\!] \cdot (\mathbf{4} - {}^{3}\!/\!{}_{2^n})$ is a lower ω-invariant of the loop with respect to $\mathbf{0}$:

$$\begin{aligned} F_{\mathbf{0}}(\mathbf{0}) &= \mathbf{1} + [\![c \neq 1]\!] \cdot \mathbf{0} + [\![c = 1]\!] \cdot \mathsf{ert}[c :\approx {}^{1}\!/\!{}_{2}\langle 0\rangle + {}^{1}\!/\!{}_{2}\langle 1\rangle](\mathbf{0}) \\ &= \mathbf{1} + [\![c = 1]\!] \cdot \left(\mathbf{1} + \tfrac{1}{2} \cdot \mathbf{0}[c/0] + \tfrac{1}{2} \cdot \mathbf{0}[c/1]\right) \\ &= \mathbf{1} + [\![c = 1]\!] \cdot \mathbf{1} = \mathbf{1} + [\![c = 1]\!] \cdot \left(\mathbf{4} - {}^{\mathbf{3}}\!/\!{}_{\mathbf{2^0}}\right) = I_0 \succeq I_0 \end{aligned}$$

$$\begin{aligned} F_{\mathbf{0}}(I_n) &= \mathbf{1} + [\![c \neq 1]\!] \cdot \mathbf{0} + [\![c = 1]\!] \cdot \mathsf{ert}[c :\approx {}^{1}\!/\!{}_{2}\langle 0\rangle + {}^{1}\!/\!{}_{2}\langle 1\rangle](I_n) \\ &= \mathbf{1} + [\![c = 1]\!] \cdot \left(\mathbf{1} + \tfrac{1}{2} \cdot I_n[c/0] + \tfrac{1}{2} \cdot I_n[c/1]\right) \\ &= \mathbf{1} + [\![c = 1]\!] \cdot \left(\mathbf{1} + \tfrac{1}{2} \cdot (\mathbf{1} + \mathbf{0}) + \tfrac{1}{2} \cdot \left(\mathbf{1} + \left(\mathbf{4} - \tfrac{\mathbf{3}}{\mathbf{2}^n}\right)\right)\right) \\ &= \mathbf{1} + [\![c = 1]\!] \cdot \left(\mathbf{4} - \tfrac{\mathbf{3}}{\mathbf{2}^{n+1}}\right) = I_{n+1} \succeq I_{n+1}. \end{aligned}$$

[2] The limit $\lim_{n\to\infty} I_n$ is to be understood pointwise on $\mathbb{R}_{\geq 0}^{\infty}$, i.e., $\lim_{n\to\infty} I_n = \lambda\sigma\text{•}\ \lim_{n\to\infty} I_n(\sigma)$ and $\lim_{n\to\infty} I_n(\sigma) = \infty$ is considered a valid value.

Then from Theorem 4.6 (1) we obtain

$$\mathsf{ert}\left[C_{\mathrm{geo}}\right](\mathbf{0}) \;\succeq\; \lim_{n\to\infty}\left(\mathbf{1} + \llbracket c = 1\rrbracket \cdot \left(\mathbf{4} - \tfrac{\mathbf{3}}{\mathbf{2}^n}\right)\right) \;=\; \mathbf{1} + \llbracket c = 1\rrbracket \cdot \mathbf{4}.$$

Combining this result with the upper bound $\mathsf{ert}\left[C_{\mathrm{geo}}\right](\mathbf{0}) \preceq \mathbf{1} + \llbracket c = 1\rrbracket \cdot \mathbf{4}$ we had established in Example 4.3, we conclude that $\mathbf{1} + \llbracket c = 1\rrbracket \cdot \mathbf{4}$ is the *exact* runtime of C_{geo}. Observe, however, that the above calculations show that I_n is both a lower and an upper ω-invariant (exact equalities $F_0(\mathbf{0}) = I_0$ and $F_0(I_n) = I_{n+1}$ hold). Then we can apply Theorem 4.6 (1) and 4.6 (2) simultaneously to derive the exact runtime without having to resort to the result from Example 4.3.

Invariant Synthesis. In order to obtain invariant $I_n = \mathbf{1} + \llbracket c = 1\rrbracket \cdot (\mathbf{4} - {}^{\mathbf{3}}/_{\mathbf{2}^n})$, we used template $I_n = \mathbf{1} + \llbracket c = 1\rrbracket \cdot \boldsymbol{a_n}$ and observed that under this template the definition of lower ω-invariant reduces to $a_0 \leq 1$, $a_{n+1} \leq 2 + \frac{1}{2}a_n$, with solution $a_n = 4 - {}^3/_{2^n}$. △

Example 4.8 (Almost-sure termination at infinite expected runtime). Recall the program from the introduction that had an infinite expected runtime, despite being the concatenation of two programs with finite expected runtime:

```
C:  1: $x := 1; b := 1;$
    2: while $(b = 1)$ {$b :\approx {}^1/_2\langle 0\rangle + {}^1/_2\langle 1\rangle; x := 2x$};
    3: while $(x > 0)$ {$x := x - 1$}
```

Now we apply Theorem 4.6 to formally analyze its runtime. We show that $\mathsf{ert}[C](\mathbf{0}) \succeq \infty$. In the course of doing so, we use C_i to denote the ith line of C. Since

$$\mathsf{ert}[C](\mathbf{0}) \;=\; \mathsf{ert}[C_1](\mathsf{ert}[C_2](\mathsf{ert}[C_3](\mathbf{0}))),$$

we start by analyzing the runtime of the loop C_3. Using the lower ω-invariant

$$J_n \;=\; \mathbf{1} + \llbracket 0 < x < n\rrbracket \cdot 2x + \llbracket x \geq n\rrbracket \cdot (2n - 1),$$

we conclude that $\mathsf{ert}[C_3](\mathbf{0}) \;\succeq\; \mathbf{1} + \llbracket x > 0\rrbracket \cdot 2x \;=\; \lim_{n\to\infty} J_n$. Next we show that

$$\begin{aligned}\mathsf{ert}[C_2](\mathbf{1} + \llbracket x > 0\rrbracket \cdot 2x) \;\succeq\; & \mathbf{1} + \llbracket b \neq 1\rrbracket \cdot \left(\mathbf{1} + \llbracket x > 0\rrbracket \cdot 2x\right) \\ & + \llbracket b = 1\rrbracket \cdot \left(\mathbf{7} + \llbracket x > 0\rrbracket \cdot \infty\right)\end{aligned}$$

by means of the lower ω-invariant

$$I_n \;=\; \mathbf{1} + \llbracket b \neq 1\rrbracket \cdot \left(\mathbf{1} + \llbracket x > 0\rrbracket \cdot 2x\right) + \llbracket b = 1\rrbracket \cdot \left(\mathbf{7} - \tfrac{\mathbf{5}}{\mathbf{2}^n} + n \cdot \llbracket x > 0\rrbracket \cdot 2x\right).$$

Let F be the characteristic functional of loop C_2 with respect to $\mathbf{1} + \llbracket x > 0\rrbracket \cdot 2x$. The calculations to establish that I_n is a lower ω-invariant go as follows:

$$\begin{aligned}F(\mathbf{0}) =\; & \mathbf{1} + \llbracket b \neq 1\rrbracket \cdot \left(\mathbf{1} + \llbracket x > 0\rrbracket \cdot 2x\right) \\ & + \llbracket b = 1\rrbracket \cdot \left(\mathbf{1} + \tfrac{1}{2} \cdot (\mathbf{1} + \mathbf{0}[x, b/2x, 0]) + \tfrac{1}{2} \cdot (\mathbf{1} + \mathbf{0}[x, b/2x, 1])\right) \\ =\; & \mathbf{1} + \llbracket b \neq 1\rrbracket \cdot \left(\mathbf{1} + \llbracket x > 0\rrbracket \cdot 2x\right) + \llbracket b = 1\rrbracket \cdot \left(\mathbf{1} + \tfrac{1}{2} \cdot \mathbf{1} + \tfrac{1}{2} \cdot \mathbf{1}\right) \\ =\; & \mathbf{1} + \llbracket b \neq 1\rrbracket \cdot \left(\mathbf{1} + \llbracket x > 0\rrbracket \cdot 2x\right) + \llbracket b = 1\rrbracket \cdot \mathbf{2} \;=\; I_0 \;\succeq\; I_0\end{aligned}$$

$$\begin{aligned}
F(I_n) \;=\; & \mathbf{1} + \llbracket b \neq 1 \rrbracket \cdot \Big(\mathbf{1} + \llbracket x > 0 \rrbracket \cdot 2x\Big) \\
& + \llbracket b = 1 \rrbracket \cdot \Big(\mathbf{1} + \tfrac{1}{2} \cdot (\mathbf{1} + I_n[x, b/2x, 0]) + \tfrac{1}{2} \cdot (\mathbf{1} + I_n[x, b/2x, 1])\Big) \\
=\; & \mathbf{1} + \llbracket b \neq 1 \rrbracket \cdot \Big(\mathbf{1} + \llbracket x > 0 \rrbracket \cdot 2x\Big) \\
& + \llbracket b = 1 \rrbracket \cdot \Big(\mathbf{1} + \tfrac{1}{2} \cdot (\mathbf{3} + \llbracket 2x > 0 \rrbracket \cdot 4x) + \tfrac{1}{2} \cdot \Big(\mathbf{9} - \tfrac{\mathbf{5}}{\mathbf{2^n}} + n \cdot \llbracket 2x > 0 \rrbracket \cdot 4x\Big)\Big) \\
=\; & \mathbf{1} + \llbracket b \neq 1 \rrbracket \cdot \Big(\mathbf{1} + \llbracket x > 0 \rrbracket \cdot 2x\Big) \\
& + \llbracket b = 1 \rrbracket \cdot \Big(\mathbf{7} - \tfrac{\mathbf{5}}{\mathbf{2^{n+1}}} + (n+1) \cdot \llbracket x > 0 \rrbracket \cdot 2x\Big) \;=\; I_{n+1} \;\succeq\; I_{n+1}.
\end{aligned}$$

Now we can complete the runtime analysis of program C:

$$\begin{aligned}
\mathrm{ert}[C](\mathbf{0}) \;&= \mathrm{ert}[C_1](\mathrm{ert}[C_2](\mathrm{ert}[C_3](\mathbf{0}))) \\
&\succeq \mathrm{ert}[C_1]\Big(\mathbf{1} + \llbracket b \neq 1 \rrbracket \cdot \Big(\mathbf{1} + \llbracket x > 0 \rrbracket \cdot 2x\Big) + \llbracket b = 1 \rrbracket \cdot \Big(\mathbf{7} + \llbracket x > 0 \rrbracket \cdot \infty\Big)\Big) \\
&= \mathrm{ert}[x := 1]\Big(\mathrm{ert}[b := 1]\Big(\mathbf{1} + \llbracket b \neq 1 \rrbracket \cdot \Big(\mathbf{1} + \llbracket x > 0 \rrbracket \cdot 2x\Big) \\
&\qquad\qquad + \llbracket b = 1 \rrbracket \cdot \Big(\mathbf{7} + \llbracket x > 0 \rrbracket \cdot \infty\Big)\Big)\Big) \\
&= \mathrm{ert}[x := 1](\mathbf{8} + \llbracket x > 0 \rrbracket \cdot \infty) \;=\; \mathbf{8} + \infty \;=\; \infty.
\end{aligned}$$

Overall, we obtain that the expected runtime of program C is infinite even though it terminates with probability one. Notice furthermore that both subprograms C_2 and C_3 have finite expected runtimes since

$$\mathrm{ert}[C_2](\mathbf{0}) \;=\; \mathbf{1} + \llbracket b = 1 \rrbracket \cdot \mathbf{4} \qquad \text{and} \qquad \mathrm{ert}[C_3](\mathbf{0}) \;=\; \mathbf{1} + \llbracket x > 0 \rrbracket \cdot 2x.$$

Invariant Synthesis. In order to synthesize the ω-invariant I_n of loop C_2, we propose the template $I_n = \mathbf{1} + \llbracket b \neq 1 \rrbracket \cdot (\mathbf{1} + \llbracket x > 0 \rrbracket \cdot 2x) + \llbracket b = 1 \rrbracket \cdot (a_n + b_n \cdot \llbracket x > 0 \rrbracket \cdot 2x)$ and from the definition of lower ω-invariants we obtain $a_0 \leq 2$, $a_{n+1} \leq 7/2 + 1/2 \cdot a_n$ and $b_0 \leq 0$, $b_{n+1} \leq 1 + b_n$. These recurrences admit solutions $a_n = 7 - 5/2^n$ and $b_n = n$. △

Just like the proof rule based on upper invariants, the proof rules based on ω-invariants are complete too: given loop `while` (ξ) $\{C\}$ and runtime t, it is enough to consider the ω-invariant $I_n = F_t^{n+1}(0)$, where F_t^n is defined as in the proof of Theorem 4.6 to yield the exact runtime ert[`while` (ξ) $\{C\}$](t) from an application of Theorem 4.6. We formally capture this result by means of the following theorem:

THEOREM 4.9. *There exists a sequence I_n that is both a lower and an upper ω-invariant of* `while` (ξ) $\{C\}$ *with respect to t, such that*

$$\mathrm{ert}[\texttt{while}\ (\xi)\ \{C\}](t) \;=\; \lim_{n \to \infty} I_n.$$

Theorem 4.9 together with Theorem 4.4 shows that the set of invariant-based proof rules presented in this section are complete. Next we study how to refine invariants to make the bounds that these proof rules yield more precise.

4.3 Refinement of Bounds

An important property of both upper and lower bounds of the runtime of loops is that they can be easily refined by repeated application of the characteristic functional. This works as follows. If u is an upper bound of ert[`while` (ξ) $\{C\}$](t) and $F_t^{\langle \xi, C \rangle}(u) \preceq u$, then $F_t^{\langle \xi, C \rangle}(u)$ is also an upper bound at least as precise as u. Dually, if l is a lower bound and $F_t^{\langle \xi, C\rangle}(l) \geq l$, then $F_t^{\langle \xi, C\rangle}(l)$ is also a lower bound at least as precise as l. Formally, we have the following theorem:[3]

Theorem 4.10 (Refinement of bounds).

(1) *If* $\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t) \leq u$ *and* $F_t^{\langle \xi, C\rangle}(u) \leq u$, *then* $\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t) \ \leq\ F_t^{\langle \xi, C\rangle}(u) \leq u$.

(2) *If* $l \leq \mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t)$ *and* $l \leq F_t^{\langle \xi, C\rangle}(l)$, *then* $l \ \leq\ F_t^{\langle \xi, C\rangle}(l) \ \leq\ \mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t)$.

Proof. We prove the first case only, as the proof for lower bounds is analogous. If u is an upper bound of $\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t)$, then $\mathsf{lfp}\, F_t^{\langle \xi, C\rangle} \leq u$. By the monotonicity of $F_t^{\langle \xi, C\rangle}$ (recall that ert is monotonic by Theorem 3.4) and from $F_t^{\langle \xi, C\rangle}(u) \leq u$ we obtain

$$\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t) \ =\ \mathsf{lfp}\, F_t^{\langle \xi, C\rangle} \ =\ F_t^{\langle \xi, C\rangle}(\mathsf{lfp}\, F_t^{\langle \xi, C\rangle}) \ \leq\ F_t^{\langle \xi, C\rangle}(u) \ \leq\ u,$$

which means that $F_t^{\langle \xi, C\rangle}(u)$ is also an upper bound, possibly tighter than u. □

Notice that if I is an upper invariant of $\mathtt{while}\ (\xi)\ \{C\}$, then I fulfills all necessary conditions of Theorem 4.10. In practice, Theorem 4.10 provides a means of iteratively improving the precision of bounds yielded by Theorems 4.2 and 4.6. For instance, for upper invariant I we have

$$\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C\}](t) \ \leq\ \cdots\ \leq\ F_t^{\langle \xi, C\rangle}\left(F_t^{\langle \xi, C\rangle}(I)\right) \ \leq\ F_t^{\langle \xi, C\rangle}(I) \ \leq I.$$

If I_n is an upper (lower, respectively) ω-invariant, applying Theorem 4.10 requires checking that $F_t^{\langle \xi, C\rangle}(L) \leq L$ (and $F_t^{\langle \xi, C\rangle}(L) \geq L$, respectively), where $L = \lim_{n\to\infty} I_n$. This proof obligation can be discharged by showing that I_n forms an ω-chain, i.e., that $I_n \leq I_{n+1}$ for all $n \in \mathbb{N}$.

This concludes our presentation of the ert-calculus for pGCL programs. In the next sections, we validate the calculus twofold. In Section 5, we show that the calculus corresponds to an intuitive operational model of pGCL programs based on Markov chains. Section 6 shows in detail that the ert-calculus is a conservative extension of Nielson's approach—basically an adaptation of Hoare triples—for obtaining upper bounds on the runtime of ordinary, i.e., nonprobabilistic, programs.

5 AN OPERATIONAL MODEL FOR EXPECTED RUNTIMES

We prove the soundness of the expected runtime transformer with respect to a simple operational model for our probabilistic programming language. This model is defined in terms of a reward Markov chain. We first briefly recall all essential notions. For a more comprehensive treatment, see [2, Ch. 10].

A *discrete-time Markov chain* (MC, for short) is a tuple $\mathcal{M} = (\mathcal{S}, \mathbf{P}, s_{init}, rew)$, where $\mathcal{S}$ is a countable, nonempty set of *control states*; $\mathbf{P} : \mathcal{S} \times \mathcal{S} \to [0, 1]$ is a *transition probability function* such that for all states $s \in \mathcal{S}$,

$$\sum_{s' \in \mathcal{S}} \mathbf{P}(s, s') = 1;$$

$s_{init} \in \mathcal{S}$ is the *initial state*; and $rew : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is a *reward function*. Intuitively, for each pair of states s, s', the transition probability function specifies the probability $\mathbf{P}(s, s')$ to take a transition from s to s'. We often write $s \xrightarrow{p} s'$ instead of $\mathbf{P}(s, s') = p$. Our operational model captures the runtime of probabilistic programs in terms of rewards. Thus, computing expected rewards of MCs along a path is essential. Formally, a *path* in an MC $\mathcal{M}$ is a finite sequence $\pi = s_1 \ldots s_n$ such

[3] A reader familiar with abstract interpretation will recognize this as applying a widening or narrowing step.

that $\mathbf{P}(s_i, s_{i+1}) > 0$ for each $1 \leq i < n$. The *set of all paths* in $\mathcal{M}$ (starting in state s) is denoted by $\mathsf{Paths}(\mathcal{M})$ ($\mathsf{Paths}(\mathcal{M}, s)$). Moreover, given a set of target states $\mathcal{T}$, we define the set of all paths starting in state s that reach a state in $\mathcal{T}$ as $\mathsf{Paths}(\mathcal{M}, s, \mathcal{T}) = \mathsf{Paths}(\mathcal{M}, s) \cap (\mathcal{S} \setminus \mathcal{T})^{\star}\mathcal{T}$. For a path $\pi = s_1 \ldots s_n$, the *cumulative reward* of π and the *probability* of π are given by

$$rew(\pi) \triangleq \sum_{k=1}^{n} rew(s_k) \quad \text{and} \quad \mathbf{P}(\pi) \triangleq \prod_{k=1}^{n-1} \mathbf{P}(s_k, s_{k+1}).$$

With these notions readily available, we can define the expected reward of an MC $\mathcal{M}$ eventually reaching a set of target states from its initial state.

Definition 5.1 (Expected rewards over MCs). Let $\mathcal{M} = (\mathcal{S}, \mathbf{P}, s_{init}, rew)$ be a Markov chain and $\mathcal{T} \subseteq \mathcal{S}$ a nonempty set of target states. The *expected reward* of $\mathcal{M}$ eventually reaching $\mathcal{T}$ from s_{init} is given by

$$\mathsf{ExpRew}^{\mathcal{M}}(\mathcal{T}) \triangleq \sum_{\pi \in \mathsf{Paths}(\mathcal{M}, s_{init}, \mathcal{T})} \mathbf{P}(\pi) \cdot rew(\pi)$$

if $\mathcal{T}$ is reached almost surely from s_{init}, i.e., if

$$\sum_{\pi \in \mathsf{Paths}(\mathcal{M}, s_{init}, \mathcal{T})} \mathbf{P}(\pi) = 1.$$

Otherwise, we set $\mathsf{ExpRew}^{\mathcal{M}}(\mathcal{T}) \triangleq \infty$.

If $\mathcal{T} = \{s\}$ is a singleton, we often write $\mathsf{ExpRew}^{\mathcal{M}}(s)$ instead of $\mathsf{ExpRew}^{\mathcal{M}}(\{s\})$. We now turn to our operational model of probabilistic programs. For simplicity, we assume a canonical labeling for each program statement $C \in \mathsf{pGCL}$. We collect the labels used in a given program C in Lab_*, including a special symbol $\downarrow$ to denote successful program termination. These labels will—together with the set of program states Σ—form the state space of our operational model. Furthermore, we employ the following auxiliary functions between program statements and labels:

- $\mathsf{init} : \mathsf{pGCL} \to \mathsf{Lab}_*$ provides the label corresponding to the beginning of a program.
- $\mathsf{stmt} : \mathsf{Lab}_* \to (\mathsf{pGCL} \cup \{\downarrow\})$ yields the statement associated with a program label.
- $\mathsf{first}, \mathsf{second} : \mathsf{Lab}_* \to \mathsf{Lab}_*$ give the first and second successor of a program label. If no such successor exists, we define $\mathsf{first}(\ell) = \downarrow$ and $\mathsf{second}(\ell) = \downarrow$, respectively.

Example 5.2. Consider the following program C where each statement is annotated with its canonical program label:

$$\begin{aligned} C:\ & \mathtt{while}\left(\left[{}^{1}\!/\!{}_{2} \cdot \langle \mathsf{true} \rangle + {}^{1}\!/\!{}_{2} \cdot \langle \mathsf{false} \rangle\right]^1\right)\{ \\ & \qquad [succ := \mathsf{true}]^2 \\ & \}; \\ & [succ := \mathsf{false}]^3 \,. \end{aligned}$$

Thus, $\mathsf{Lab}_* = \{\downarrow, 1, 2, 3\}$. The definition of the auxiliary functions is straightforward:

$\mathsf{init}(C) = 1$	$\mathsf{stmt}(1) = \mathtt{while}\,(\ldots)\,\{succ := \mathsf{true}\}$	$\mathsf{first}(1) = 2$	$\mathsf{second}(1) = 3$
	$\mathsf{stmt}(2) = succ := \mathsf{true}$	$\mathsf{first}(2) = 1$	$\mathsf{second}(2) = \downarrow$
	$\mathsf{stmt}(3) = succ := \mathsf{false}$	$\mathsf{first}(3) = \downarrow$	$\mathsf{second}(3) = \downarrow$
	$\mathsf{stmt}(\downarrow) = \downarrow$	$\mathsf{first}(\downarrow) = \downarrow$	$\mathsf{second}(\downarrow) = \downarrow$.

Definition 5.3 (MC of a pGCL *program).* For initial state $\sigma_0 \in \Sigma$ and $t \in \mathbb{T}$, the *operational Markov chain* of $C \in \mathsf{pGCL}$ is given by $\mathcal{M}^{t}_{\sigma_0}[\![C]\!] = (\mathcal{S}, \mathbf{P}, s_{\mathrm{init}}, rew)$, where:

$$\frac{\mathsf{stmt}(\ell) = \downarrow}{\langle \ell, \sigma\rangle \xrightarrow{1} \langle\, \mathit{sink}\,\rangle}\ [\text{terminated}] \qquad \frac{}{\langle\, \mathit{sink}\,\rangle \xrightarrow{1} \langle\, \mathit{sink}\,\rangle}\ [\text{sink}]$$

$$\frac{\mathsf{stmt}(\ell) = \texttt{empty} \qquad \mathsf{first}(\ell) = \ell'}{\langle \ell, \sigma\rangle \xrightarrow{1} \langle \ell', \sigma\rangle}\ [\text{empty}] \qquad \frac{\mathsf{stmt}(\ell) = \texttt{skip} \qquad \mathsf{first}(\ell) = \ell'}{\langle \ell, \sigma\rangle \xrightarrow{1} \langle \ell', \sigma\rangle}\ [\text{skip}]$$

$$\frac{\mathsf{stmt}(\ell) = \texttt{halt}}{\langle \ell, \sigma\rangle \xrightarrow{1} \langle\, \mathit{sink}\,\rangle}\ [\text{halt}] \qquad \frac{\mathsf{stmt}(\ell) = x \mathrel{:\approx} \mu \qquad [\![\mu \colon v]\!](\sigma) = p > 0 \qquad \mathsf{first}(\ell) = \ell'}{\langle \ell, \sigma\rangle \xrightarrow{p} \langle \ell', \sigma[x/v]\rangle}\ [\text{pr–assgn}]$$

$$\frac{\mathsf{stmt}(\ell) = \texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\} \qquad [\![\xi \colon \mathsf{true}]\!](\sigma) = p > 0 \qquad \mathsf{first}(\ell) = \ell'}{\langle \ell, \sigma\rangle \xrightarrow{p} \langle \ell', \sigma\rangle}\ [\text{if–true}]$$

$$\frac{\mathsf{stmt}(\ell) = \texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\} \qquad [\![\xi \colon \mathsf{false}]\!](\sigma) = p > 0 \qquad \mathsf{second}(\ell) = \ell'}{\langle \ell, \sigma\rangle \xrightarrow{p} \langle \ell', \sigma\rangle}\ [\text{if–false}]$$

$$\frac{\mathsf{stmt}(\ell) = \texttt{while}\ (\xi)\ \{C\} \qquad [\![\xi \colon \mathsf{true}]\!](\sigma) = p > 0 \qquad \mathsf{first}(\ell) = \ell'}{\langle \ell, \sigma\rangle \xrightarrow{p} \langle \ell', \sigma\rangle}\ [\text{while–true}]$$

$$\frac{\mathsf{stmt}(\ell) = \texttt{while}\ (\xi)\ \{C\} \qquad [\![\xi \colon \mathsf{false}]\!](\sigma) = p > 0 \qquad \mathsf{second}(\ell) = \ell'}{\ell, \sigma \ \xrightarrow{p}\ \ell', \sigma}\ [\text{while–false}]$$

Fig. 1. Rules for the transition probability function of operational MCs.

Table 2. The Reward Function $\mathit{rew} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ of Operational MCs

s	**stmt(ℓ)**	***rew*(s)**
$\langle \ell, \sigma\rangle$	$\downarrow$	$t(\sigma)$
$\langle \ell, \sigma\rangle$	skip, $x :\approx\ \mu$, if (ξ) $\{C_1\}$ else $\{C_2\}$, or while (ξ) $\{C\}$	1
$\langle \ell, \sigma\rangle$	empty, halt	0
$\langle\, \mathit{sink}\,\rangle$		0

- $\mathcal{S} = \{\langle \ell, \sigma\rangle \mid \ell \in \mathsf{Lab}_*,\ \sigma \in \Sigma\} \cup \{\langle\, \mathit{sink}\,\rangle\}$,
- the transition probability function $\mathbf{P}$ is given by the rules in Figure 1,
- $s_{init} = \langle \mathsf{init}(C), \sigma_0\rangle$, and
- the reward function $\mathit{rew} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is defined according to Table 2. △

Most of the rules in Figure 1 defining the transition probability function of a program's MC are self-explanatory. As an example, consider the following rule for `while` loops:

$$\frac{\mathsf{stmt}(\ell) = \texttt{while}\ (\xi)\ \{C\} \qquad [\![\xi : \mathsf{true}]\!](\sigma) = p > 0 \qquad \mathsf{first}(\ell) = \ell'}{\langle \ell, \sigma\rangle \xrightarrow{p} \langle \ell', \sigma\rangle}\ [\text{while–true}].$$

To apply this rule, we first have to determine the probability p of the loop's guard being true. If $p > 0$, then the MC contains a transition to move with probability p from its current state to the first statement contained in the loop's body, i.e., $\mathsf{first}(\ell)$. Analogously, by the rule [while–false], there is a transition to leave the loop, i.e., move to $\mathsf{second}(\ell)$, with probability $1{-}p$, the likelihood of the guard being false.

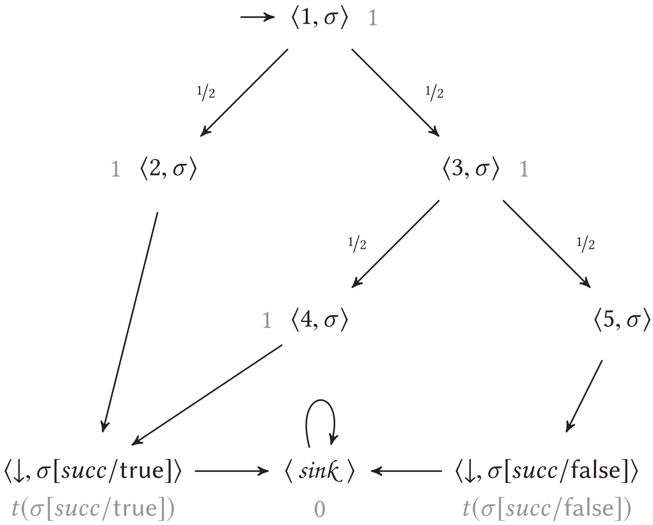

$\langle 1, \sigma \rangle$ 1
1/2
1/2
1 $\langle 2, \sigma \rangle$
$\langle 3, \sigma \rangle$ 1
1/2
1/2
1 $\langle 4, \sigma \rangle$
$\langle 5, \sigma \rangle$ 1
$\langle \downarrow, \sigma[succ/\mathsf{true}] \rangle$
$t(\sigma[succ/\mathsf{true}])$
$\langle\, sink \,\rangle$
0
$\langle \downarrow, \sigma[succ/\mathsf{false}] \rangle$
$t(\sigma[succ/\mathsf{false}])$

Fig. 2. The operational MC $\mathsf{ExpRew}^{\mathcal{M}^t_\sigma[\![C_{trunc}]\!]}(\langle\, sink \,\rangle)$ corresponding to the truncated geometric distribution. For each state, the corresponding reward is provided in gray.

Let us explain the difference between state $\langle\, sink \,\rangle$ and states of the form $\langle \downarrow, \sigma \rangle$. The state $\langle\, sink \,\rangle$ is reached after either successful program termination or premature halting; states of the form $\langle \downarrow, \sigma \rangle$ are reached only upon successful program termination. After halting a program run, the MC immediately evolves to the sink state (see rule [halt] in Figure 1). Note that the continuation $t \in \mathbb{T}$ of a program contributes to the cumulative reward to reach the sink state only upon states of the form $\langle \downarrow, \sigma \rangle$ (see first row of Table 2).

Example 5.4 (MC for truncated geometric distribution). Recall the probabilistic program C_{trunc} from Example 3.3 together with its canonical labeling:

$$
\begin{aligned}
C_{trunc}:\ & \texttt{if}\left([1/2\cdot\langle\mathsf{true}\rangle + 1/2\cdot\langle\mathsf{false}\rangle]^1\right)\{[succ := \mathsf{true}]^2\}\,\texttt{else}\,\{\\
& \qquad \texttt{if}\left([1/2\cdot\langle\mathsf{true}\rangle + 1/2\cdot\langle\mathsf{false}\rangle]^3\right)\{[succ := \mathsf{true}]^4\}\\
& \qquad \texttt{else}\,\{[succ := \mathsf{false}]^5\}\\
& \}.
\end{aligned}
$$

Figure 2 depicts the MC $\mathcal{M}^t_\sigma[\![C_{trunc}]\!]$ for an initial program state $\sigma \in \Sigma$ and an arbitrary continuation $t \in \mathbb{T}$. Here labeled edges denote the value of the transition probability function for the respective states, while the reward of each state is provided in gray next to the state. To improve readability, edge labels are omitted if the probability of a transition equals one.

A brief inspection of Figure 2 reveals that $\mathcal{M}^t_\sigma[\![C_{trunc}]\!]$ contains three finite paths reaching $\langle\, sink \,\rangle$ from initial state $\langle 1, \sigma \rangle$:

$$
\begin{aligned}
\pi_1 &= \langle 1,\sigma\rangle\langle 2,\sigma\rangle\langle\downarrow,\sigma[succ/\mathsf{true}]\rangle\langle\, sink \,\rangle,\\
\pi_2 &= \langle 1,\sigma\rangle\langle 3,\sigma\rangle\langle 4,\sigma\rangle\langle\downarrow,\sigma[succ/\mathsf{true}]\rangle\langle\, sink \,\rangle, \text{ and}\\
\pi_3 &= \langle 1,\sigma\rangle\langle 3,\sigma\rangle\langle 5,\sigma\rangle\langle\downarrow,\sigma[succ/\mathsf{false}]\rangle\langle\, sink \,\rangle.
\end{aligned}
$$

These paths correspond to the results of the two probabilistic guards in C. Hence, the expected reward of $\mathcal{M}^t_\sigma[\![C_{trunc}]\!]$ eventually reaching $\langle\, sink \,\rangle$ is given by

$$\begin{aligned}
\mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![C_{trunc}]\!]}(\langle\, \mathit{sink} \,\rangle) &= \sum_{\pi \in \mathsf{Paths}(\mathcal{M}_\sigma^t[\![C_{trunc}]\!], \langle 1, \sigma\rangle, \langle\, \mathit{sink}\,\rangle)} \mathbf{P}(\pi) \cdot \mathit{rew}(\pi) \\
&= \mathbf{P}(\pi_1) \cdot \mathit{rew}(\pi_1) \;+\; \mathbf{P}(\pi_2) \cdot \mathit{rew}(\pi_2) \;+\; \mathbf{P}(\pi_3) \cdot \mathit{rew}(\pi_3) \\
&= \left(\tfrac{1}{2} \cdot 1 \cdot 1\right) \cdot (1 + 1 + t(\sigma[\mathit{succ}/\mathsf{true}])) \\
&\quad + \left(\tfrac{1}{2} \cdot \tfrac{1}{2} \cdot 1 \cdot 1\right) \cdot (1 + 1 + 1 + t(\sigma[\mathit{succ}/\mathsf{true}])) \\
&\quad + \left(\tfrac{1}{2} \cdot \tfrac{1}{2} \cdot 1 \cdot 1\right) \cdot (1 + 1 + 1 + t(\sigma[\mathit{succ}/\mathsf{false}])) \\
&= \tfrac{5}{2} \;+\; \tfrac{3}{4} \cdot t(\sigma[\mathit{succ}/\mathsf{true}]) \;+\; \tfrac{1}{4} \cdot t(\sigma[\mathit{succ}/\mathsf{false}]),
\end{aligned}$$

as the probability of reaching $\langle\, \mathit{sink} \,\rangle$ (through these three paths) sums up to one.

Observe that for $t = \mathbf{0}$, the expected reward $\mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![C_{trunc}]\!]}(\langle\, \mathit{sink} \,\rangle)$ and the expected runtime $\mathsf{ert}[C](t)(\sigma)$ (cf. Example 3.3) coincide, both yielding $5/2$. △

As the previous example suggests, the expected reward $\mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![C]\!]}(\langle\, \mathit{sink} \,\rangle)$ to reach the sink state in the MC $\mathcal{M}_\sigma^t[\![C]\!]$ of program C coincides with the expected runtime $\mathsf{ert}[C](t)(\sigma)$ as given by the ert-transformer. Formally:

Theorem 5.5 (Soundness of the ert transformer). *Let $C \in$ pGCL. Then for each initial program state $\sigma \in \Sigma$ and continuation $t \in \mathbb{T}$,*

$$\mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![C]\!]}(\langle\, \mathit{sink} \,\rangle) \;=\; \mathsf{ert}[C](t)(\sigma).$$

Proof. By induction on the structure of pGCL programs. See Appendix B.3. □

Hence, the expected runtime transformer ert is sound with respect to our operational program model based on Markov chains.

6 RUNTIME OF DETERMINISTIC PROGRAMS

Nielson [41, 42] has extended Hoare logic [21] so as to obtain a formal verification framework to reason about upper bounds on the runtime of deterministic programs, i.e., programs containing neither probabilistic guards nor probabilistic assignments. Whereas Hoare logic originally focuses on partial correctness, Nielson considered the variant for total correctness (see, e.g., [34]). We show in this section that applying ert to deterministic programs results in the tightest upper bound on the runtime obtained in Nielson's approach.

The language GCL of deterministic programs considered in [42] is given by the grammar

$$C \;::=\; \mathtt{skip} \mid x := E \mid C; C \mid \mathtt{if}\,(B)\,\{C\}\,\mathtt{else}\,\{C\} \mid \mathtt{while}\,(B)\,\{C\}.$$

Here E is a *deterministic* expression and B is a *deterministic* guard; i.e., $[\![E]\!](\sigma)$ and $[\![B]\!](\sigma)$ are Dirac distributions for each $\sigma \in \Sigma$. For simplicity, we slightly abuse notation and write $[\![E]\!](\sigma)$ to denote the unique value $v \in \mathsf{Val}$ such that $[\![E : v]\!](\sigma) = 1$. Analogously, $[\![B]\!](\sigma)$ denotes the unique value $b \in \{\mathsf{true}, \mathsf{false}\}$ such that $[\![B : b]\!](\sigma) = 1$.

Nielson's calculus [41, 42] aims at verifying total program correctness while in addition establishing upper bounds on the program runtime. A correctness property in this extended calculus is of the form

$$\{\, P \,\}\; C \;\{\, E \Downarrow Q \,\},$$

where $C \in$ GCL, E is a deterministic expression over the program variables, and P, Q are assertions expressed in first-order logic. The symbol $\Downarrow$ is just a separator between the postcondition Q and the (bound on the) runtime E. Intuitively, the triple $\{\, P \,\}\; C \;\{\, E \Downarrow Q \,\}$ is valid, written $\models_E \{\, P \,\}\; C \;\{\, E \Downarrow Q \,\}$, if and only if there exists a natural number k such that from each initial state

$$\frac{}{\{\,P\,\}\ \texttt{skip}\ \{\,1 \Downarrow P\,\}}\ [\texttt{skip}] \qquad \frac{}{\{\,Q\big[x/[\![E]\!]\big]\,\}\ x := E\ \{\,1 \Downarrow Q\,\}}\ [\text{assgn}]$$

$$\frac{\{\,P \wedge E_2' = u\,\}\ C_1\ \{\,E_1 \Downarrow Q \wedge E_2 \leq u\,\} \qquad \{\,Q\,\}\ C_2\ \{\,E_2 \Downarrow R\,\}}{\{\,P\,\}\ C_1; C_2\ \{\,E_1{+}E_2' \Downarrow R\,\}}\ [\text{seq}]$$

where u is a fresh logical variable

$$\frac{\{\,P \wedge B\,\}\ C_1\ \{\,E \Downarrow Q\,\} \qquad \{\,P \wedge \neg B\,\}\ C_2\ \{\,E \Downarrow Q\,\}}{\{\,P\,\}\ \texttt{if}\,(B)\,\{C_1\}\ \texttt{else}\ \{C_2\}\ \{\,E \Downarrow Q\,\}}\ [\text{if}]$$

$$\frac{\{\,P(z{+}1) \wedge E' = u\,\}\ C\ \{\,E_1 \Downarrow P(z) \wedge E \leq u\,\}}{\{\,\exists z\text{.}\ P(z)\,\}\ \texttt{while}\,(B)\,\{C\}\ \{\,E \Downarrow P(0)\,\}}\ [\texttt{while}]$$

where $z \in \mathbb{N},\ P(z{+}1) \Rightarrow B \wedge E \geq E_1{+}E',\ P(0) \Rightarrow \neg B \wedge E \geq 1$
and u is a fresh logical variable

$$\frac{\{\,P'\,\}\ C\ \{\,E' \Downarrow Q'\,\}}{\{\,P\,\}\ C\ \{\,E \Downarrow Q\,\}}\ [\text{cons}]$$

where $P \Rightarrow P' \wedge E' \leq k{\cdot}E$ for some $k \in \mathbb{N}$ and $Q' \Rightarrow Q$

Fig. 3. Nielson's [41] inference system for order of magnitude of runtime of deterministic programs.

σ satisfying precondition P, program C terminates after at most $k{\cdot}[\![E]\!](\sigma)$ steps in a state satisfying postcondition Q. Note that E is evaluated in the *initial* state σ.

The inference rules in Figure 3 are taken verbatim from [42] except for minor changes to match our notation. Let us briefly explain the inference rules. Most of the inference rules are self-explanatory extensions of the standard extension of Hoare calculus for total correctness of deterministic programs [34], which is obtained by omitting the gray parts. The runtime of skip and $x := E$ is one time unit. Since guard evaluations are assumed to consume no time in this calculus, any upper bound on the runtime of both branches of a conditional is also an upper bound on the runtime of the conditional itself; cf. rule [if]. The rule of consequence allows one to increase an already proven upper bound on the runtime by an arbitrary constant factor. The runtime of the sequential composition of C_1 and C_2 is, intuitively, the sum of their runtimes E_1 and E_2. However, runtimes are expressions that are evaluated in the initial state. Thus, the runtime of C_2 has to be expressed in the initial state of $C_1; C_2$. Technically, this is achieved by adding a fresh (and hence universally quantified) variable u that is an upper bound on E_2 and at the same time equals a new expression E_2' in the precondition of $C_1; C_2$. The runtime of $C_1; C_2$ is then given by $E_1{+}E_2'$. The same principle is applied to each loop iteration. Here, the runtime of the loop body is given by E_1 and the runtime E' of the remaining z loop iterations is expressed in the initial state by using a fresh variable u. Any upper bound of $E \geq E_1{+}E'$ bounds the runtime of z loop iterations from above.

For deterministic program $C \in \mathsf{GCL}$, let $\vdash_E \{\,P\,\}\ C\ \{\,E \Downarrow Q\,\}$ denote that the correctness property $\{\,P\,\}\ C\ \{\,E \Downarrow Q\,\}$ is provable in Nielson's calculus. Analogously, provability of a total correctness property $\{\,P\,\}\,C\,\{\Downarrow\ Q\,\}$ in the standard Hoare calculus is denoted by $\vdash \{\,P\,\}\,C\,\{\Downarrow\ Q\,\}$. Our first result concerning Nielson's calculus asserts that a correctness proof of C in standard Hoare logic and the ert of C can be combined into a proof in Nielson's proof system.

Theorem 6.1 (Soundness of ert w.r.t. Nielson's calculus). *For all deterministic programs $C \in \mathsf{GCL}$ and assertions P, Q, we have*

$$\vdash \{\,P\,\}\,C\,\{\Downarrow\ Q\,\} \quad \textit{implies} \quad \vdash_E \{\,P\,\}\ C\ \{\,\mathsf{ert}[C]\,(\mathbf{0}) \Downarrow Q\,\}.$$

Proof. By structural induction on C. See Appendix B.4 for a detailed proof. □

Hence, our notion of ert is sound with respect to Nielson's proof system. The next theorem asserts that no tighter bound can be derived in Nielson's calculus. We cannot get a more precise relationship due to different runtime models: while guard evaluations are assumed to consume no time in Nielson's logic, we assume each guard evaluation to consume one unit of time.

Theorem 6.2 (Completeness of ert w.r.t. Nielson's calculus). *For all deterministic programs $C \in \mathsf{GCL}$, assertions P, Q, and deterministic expressions E:*

$$\vdash_E \ \{\, P \,\}\ C\ \{\, E \Downarrow Q \,\} \quad \textit{implies} \quad \mathsf{ert}[C](\mathbf{0})(\sigma) \ \leq \ k\cdot[\![E]\!](\sigma),$$

for some $k \in \mathbb{N}$ and all program states $\sigma \in \Sigma$ satisfying P.

Proof. By induction on C's structure; see Appendix B.5 for a detailed proof. □

Theorem 6.1 together with Theorem 6.2 shows that ert is a conservative extension of Nielson's approach for reasoning about the runtime of deterministic programs. In particular, given a correctness proof of a deterministic program C in Hoare logic, it suffices to compute $\mathsf{ert}[C](\mathbf{0})$ in order to obtain a corresponding proof in Nielson's proof system.

7 RECURSION

We now extend the results of the previous sections to be able to reason about the expected runtime of *recursive* randomized algorithms. For achieving that, we extend the transformer ert to allow for recursive procedures. We also prove that this extension preserves all properties studied earlier (such as continuity, etc.) and present a set of proof rules to effectively reason about the runtime of recursive procedure calls. As for an operational semantics, we show that probabilistic programs with recursive procedures can be interpreted as *push-down* Markov chains and we show that the result from Theorem 5.5 relating the wp-based and the operational approach remains valid for recursive programs.

7.1 A Recursive Probabilistic Language

As a first step, we extend pGCL with recursive programs by incorporating procedure calls. For simplicity, we assume the presence of only a single procedure, say, P. We defer the treatment of multiple (possibly mutually recursive) procedures to Section 7.5. The syntax of our *probabilistic Recursive Guarded Command Language* (pRGCL) thus extends the syntax of pGCL (see Section 2) by an atomic statement for procedure calls:

$$\texttt{call}\, P.$$

We assume that the procedure P manipulates the global program state and we thus dispense with local variables, parameters, and `return` statements for passing information across procedure calls. The *declaration* of P consists of a *procedure body* (much like a loop body), which is written in pRGCL syntax. In particular, the body of a procedure can itself contain procedure calls and thus procedures can recursively invoke themselves. We denote by

$$P \ \triangleright \ C$$

that procedure P has body $C \in \mathsf{pRGCL}$. A pRGCL *program* is a pair $\langle C, \mathcal{D}\rangle$, where $C \in \mathsf{pRGCL}$ is the "main" command and $\mathcal{D} : \{P\} \to \mathsf{pRGCL}$ is the declaration of P.[4]

[4] The declaration of P is a mapping from a singleton and not the mere body of P because this minimizes the changes to accommodate the subsequent treatment of multiple procedures.

Table 3. Inductive Definition for the Runtime Transformer $\mathsf{ert}[\,\cdot\,,\mathcal{D}]$ for Recursive Programs

C	$\mathbf{ert}[C,\mathcal{D}](t)$
`empty`	t
`skip`	$\mathbf{1}+t$
`halt`	$\mathbf{0}$
$x :\approx\ \mu$	$\mathbf{1}+\lambda\sigma\text{.}\ \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\ t[x/v](\sigma))$
$C_1; C_2$	$\mathsf{ert}[C_1,\mathcal{D}]\big(\mathsf{ert}[C_2,\mathcal{D}](t)\big)$
$\texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\}$	$\mathbf{1}+[\![\xi\colon \mathsf{true}]\!]\cdot\mathsf{ert}[C_1,\mathcal{D}](t)+[\![\xi\colon \mathsf{false}]\!]\cdot\mathsf{ert}[C_2,\mathcal{D}](t)$
$\texttt{while}\ (\xi)\ \{C'\}$	$\mathsf{lfp}\,X\text{.}\ \mathbf{1}+[\![\xi\colon \mathsf{false}]\!]\cdot t+[\![\xi\colon \mathsf{true}]\!]\cdot\mathsf{ert}[C',\mathcal{D}](X)$
$\texttt{call}\,P$	$\big(\mathsf{lfp}\,\eta\text{.}\ \underline{\mathbf{1}}\oplus\mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\eta}\big)\ (t)$

Example 7.1 (Faulty recursive factorial). Consider the following recursive procedure for (erroneously) computing the factorial of a natural number stored in x:

```
P_fact ▷ if (x ≤ 0) {y := 1} else {
            if (5/6 · ⟨true⟩ + 1/6 · ⟨false⟩) {
                x := x − 1; call P_fact; x := x + 1
            } else {
                x := x − 2; call P_fact; x := x + 2
            };
            y := y · x
         }.
```

In each recursive call, variable x is decreased by either one or two, with probability $5/6$ and $1/6$, respectively. Therefore, some factors might be missing in the computation of the factorial of x. △

Concerning pRGCL's runtime model, we assume that invoking a procedure (i.e., the procedure call itself) consumes one unit of time. The overall runtime of a procedure call is then one plus the runtime of the procedure's body.

7.2 The ert Transformer for Recursive Randomized Algorithms

Since declarations are part of recursive programs, the ert transformer on pRGCL now has shape

$$\mathsf{ert}[C,\mathcal{D}] : \mathbb{T}\to\mathbb{T}.$$

As a consequence, the rules in Table 1 giving the inductive definition of the transformer must be slightly adapted so that it also propagates declarations. The adaptation for all constructs (except procedure calls) is quite straightforward; see Table 3.

It remains to define the runtime of procedure calls. In the same way as for loops, the action of the transformer on procedure calls is defined using fixed-point techniques. Procedure calls require, however, higher-order fixed points and the use of a subsidiary transformer:

$$\mathsf{ert}[C]^{\sharp}_{\eta} : \mathbb{T}\to\mathbb{T}.$$

This transformer behaves exactly as the ert transformer (see Table 1) for all programs except procedure calls. For those, it reverts to a provided *runtime environment* η in

$$\mathsf{RtEnv} \triangleq \left\{\eta \mid \eta : \mathbb{T} \rightarrow \mathbb{T} \text{is} \text{continuous}\right\},$$

instead of the procedure declaration. We can think of such an environment η as a lookup table, in which the transformer $\mathsf{ert}[C]^{\sharp}_{\eta}$ can look up how to handle the effects of procedure calls. With this in mind, we define

$$\mathsf{ert}[\mathtt{call}\, P]^{\sharp}_{\eta}(t) \triangleq \eta(t).$$

Using the subsidiary transformer $\mathsf{ert}[\,\cdot\,]^{\sharp}_{\eta}$, we can (implicitly) define the action of $\mathsf{ert}[\,\cdot\,,\mathcal{D}]$ on procedure calls using the equation

$$\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}] \;=\; \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}]}.$$

Here, $\underline{\mathbf{1}} \triangleq \lambda t\textbf{.}\, \mathbf{1}$ represents the constantly $\mathbf{1}$ runtime transformer and "$\oplus$" the pointwise sum between runtime transformers; i.e., for $\gamma_1, \gamma_2 : \mathbb{T} \rightarrow \mathbb{T}$, we let $(\gamma_1 \oplus \gamma_2)(t) \triangleq \gamma_1(t) + \gamma_2(t)$. This equation immediately leads to the (fixed point) characterization

$$\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}] \;\triangleq\; \mathsf{lfp}\, \eta\textbf{.}\, \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\eta} \tag{1}$$

as in Table 3. To guarantee the existence of the above fixed point, we follow the same strategy as for loops: we endow the set of runtime environments with the structure of an ω-cpo $(\mathsf{RtEnv}, \sqsubseteq)$ and we prove that the environment transformer $\lambda\eta\textbf{.}\, \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\eta}$ is continuous. Kleene's Fixed Point Theorem ensures that the transformer's fixed point is well defined; for details, see Appendix B.6.

As a next step, we show that all the transformer properties for programs with loops in Section 3 remain valid for recursive programs.

Theorem 7.2 (Basic properties of ert for recursive programs). *For any recursive program* $\langle C, \mathcal{D}\rangle \in \mathsf{pRGCL}$*, any constant runtime* $\mathbf{k}$ *with* $k \in \mathbb{R}_{\geq 0}$*, any runtimes* $t, t' \in \mathbb{T}$*, and any* ω*-chain* $t_0 \preceq t_1 \preceq \cdots$ *of runtimes in* $\mathbb{T}$*:*

Continuity: $\sup_n \mathsf{ert}[C, \mathcal{D}](t_n) = \mathsf{ert}[C, \mathcal{D}](\sup_n t_n)$;

Monotonicity: $t \preceq t' \;\Rightarrow\; \mathsf{ert}[C, \mathcal{D}](t) \;\preceq\; \mathsf{ert}[C, \mathcal{D}](t')$;

Constant propagation: $\mathsf{ert}[C, \mathcal{D}](\mathbf{k} + t) = \mathbf{k} + \mathsf{ert}[C, \mathcal{D}](t)$, *provided* C *is* `halt`*-free*;

Preservation of ∞: $\mathsf{ert}[C, \mathcal{D}](\infty) = \infty$, *provided* C *is* `halt`*-free.*

Proof. For the proof of continuity see Appendix B.7. Monotonicity follows from continuity. Preservation of constants is proved in Appendix B.8. Preservation of infinity follows by the same argument as for nonrecursive programs (see the proof of Theorem 3.4). □

7.3 Proof Rules for Recursion

As for while loops, the runtime of recursive procedures is defined using fixed points. However, reasoning about recursive procedures is typically more involved than reasoning about loops since recursive procedures involve *higher-order* fixed points [19]—the runtime of a loop requires the fixed point of a runtime transformer, while the runtime of a recursive procedure requires the fixed point of an environment transformer. To alleviate this, we propose a set of proof rules for approximating the runtime of (recursive) procedures avoiding the use of any fixed points.

Loosely speaking, the proof rules say that in order to prove that a procedure call satisfies a given runtime specification, it suffices to show that the procedure's body satisfies the specification, assuming that the recursive calls in the body do so too. To formally state the rules, we require the notion of *constructive derivability*. Given logical formulae A and B, we write $A \Vdash B$ to denote that B can be derived assuming A. In particular, we will consider claims of the form

$$\mathsf{ert}[\mathtt{call}\,P](t_1) \bowtie g_1 \;\Vdash\; \mathsf{ert}[C](t_2) \bowtie g_2,$$

where $\bowtie \in \{\preceq, \succeq\}$; t_1, g_1 give the runtime specification of $\mathtt{call}\,P$; and t_2, g_2 give the runtime specification of C. Notice that in such a claim, we omit any procedure declaration $\mathcal{D}$ as the derivation is independent of P's body. That is, whenever we encounter a procedure call $\mathtt{call}\,P$ in C, we may replace it by an upper or lower bound g_1 (depending on $\bowtie$) without taking procedure declarations into account. Formally, the statement $\mathsf{ert}[\mathtt{call}\,P](t_1) \bowtie g_1 \Vdash \mathsf{ert}[C](t_2) \bowtie g_2$ can thus also be understood as

$$\forall \eta \text{ with } \eta(t_1) \bowtie g_1 : \mathsf{ert}[C]^{\sharp}_{\eta}(t_2) \bowtie g_2,$$

where $\eta(t_1)$ plays the role of $\mathsf{ert}[\mathtt{call}\,P](t_1)$.

Theorem 7.3 (Runtime bounds for procedure calls). *Let $\mathcal{D}$ be the declaration of procedure P, $u \in \mathbb{T}$, and let $(l_n)_{n\in\mathbb{N}}$, $(u_n)_{n\in\mathbb{N}}$ be two sequences of runtimes in $\mathbb{T}$.*

(1) If

$$\mathsf{ert}[\mathtt{call}\,P](t) \preceq \mathbf{1} + u \;\Vdash\; \mathsf{ert}[\mathcal{D}(P)](t) \preceq u,$$

then

$$\mathsf{ert}[\mathtt{call}\,P, \mathcal{D}](t) \preceq \mathbf{1} + u.$$

(2) If $\lim_{n\to\infty} u_n$ exists, $u_0 = \mathbf{0}$, and for all $n \geq 0$,

$$\mathsf{ert}[\mathtt{call}\,P](t) \preceq \mathbf{1} + u_n \;\Vdash\; \mathsf{ert}[\mathcal{D}(P)](t) \preceq u_{n+1},$$

then

$$\mathsf{ert}[\mathtt{call}\,P, \mathcal{D}](t) \preceq \mathbf{1} + \lim_{n\to\infty} u_n.$$

(3) If $\lim_{n\to\infty} l_n$ exists, $l_0 = 0$, and for all $n \geq 0$,

$$\mathbf{1} + l_n \preceq \mathsf{ert}[\mathtt{call}\,P](t) \;\Vdash\; l_{n+1} \preceq \mathsf{ert}[\mathcal{D}(P)](t),$$

then

$$\mathbf{1} + \lim_{n\to\infty} l_n \preceq \mathsf{ert}[\mathtt{call}\,P, \mathcal{D}](t).$$

These proof rules can be seen as a direct counterpart of the proof rules for reasoning about the runtime of loops studied in Section 4: rule (1) is the counterpart of the rule based on loop upper invariants (Theorem 4.2), while rules (2) and (3) are the counterpart of the rules based on ω-invariants (Theorem 4.6). The above proof rules are inspired by the traditional proof rule used for reasoning about functional correctness of ordinary recursive programs; for the traditional weakest precondition transformer wp, this rule says that if

$$\mathsf{wp}[\mathtt{call}\,P](Q) \Rightarrow R \;\Vdash\; \mathsf{wp}[\mathcal{D}(P)](Q) \Rightarrow R,$$

then

$$\mathsf{wp}[\mathtt{call}\,P,\ \mathcal{D}](Q) \Rightarrow R,$$

R and Q being the pre- and postcondition of $\mathtt{call}\,P$, respectively [19]. Compared to our proof rule (1) in Theorem 7.3, the partial order "$\preceq$" over runtimes is replaced by the partial order "$\Rightarrow$" over predicates, and the shifting of one unit of time occurs because the runtime of a procedure call is one plus the runtime of its body, whereas the semantics of a procedure call fully agrees with the semantics of its body. A detailed soundness proof for our rules is provided in Appendix B.9.

Example 7.4 (Proving exact expected runtimes). Consider the pRGCL procedure P_{geo} that is given by declaration $\mathcal{D}$:

$$P_{\text{geo}} \rhd \texttt{if}\ (1/2 \cdot \langle\mathsf{true}\rangle + 1/2 \cdot \langle\mathsf{false}\rangle)\ \{\texttt{call}\ P_{\text{geo}}\}\ \texttt{else}\ \{\texttt{skip}\}.$$

We prove that the exact expected runtime of calling procedure P_{geo} is five units of time, i.e., $\mathsf{ert}[\texttt{call}\ P_{\text{geo}}, \mathcal{D}](\mathbf{0}) = \mathbf{5}$, by using simultaneously rules (2) and (3) from Theorem 7.3. To this end, we propose a sequence of runtimes t_n as follows:

$$t_n \;=\; \begin{cases} \mathbf{0}, & \text{if } n = 0 \\ \mathbf{4} - \frac{1}{2^{n-2}}, & \text{if } n > 0. \end{cases}$$

Clearly, $\lim_{n\to\infty} t_n = \mathbf{4}$. To apply Theorem 7.3, it thus suffices to discharge that we have

$$\mathsf{ert}[\texttt{call}\ P_{\text{geo}}](\mathbf{0}) = \mathbf{1} + t_n \;\Vdash\; \mathsf{ert}[\mathcal{D}(P_{\text{geo}})](\mathbf{0}) = t_{n+1}$$

for all natural numbers n. This amounts to the following calculations:

$$\begin{aligned}
&\mathsf{ert}[\mathcal{D}(P_{\text{geo}})](\mathbf{0}) \\
&= \mathbf{1} + 1/2 \cdot \mathsf{ert}[\texttt{call}\ P_{\text{geo}}](\mathbf{0}) + 1/2 \cdot \mathsf{ert}[\texttt{skip}](\mathbf{0}) \\
&= \mathbf{1} + 1/2 \cdot (\mathbf{1} + t_n) + 1/2 \cdot \mathsf{ert}[\texttt{skip}](\mathbf{0}) && (\text{claim} : \mathsf{ert}[\texttt{call}\ P_{\text{geo}}](\mathbf{0}) = \mathbf{1} + t_n) \\
&= \mathbf{1} + 1/2 \cdot (\mathbf{1} + t_n) + 1/2 \cdot (\mathbf{1} + \mathbf{0}) && (\text{Table 3}) \\
&= \mathbf{2} + 1/2 \cdot t_n.
\end{aligned}$$

For $n = 0$, this means that

$$\mathsf{ert}[\mathcal{D}(P_{\text{geo}})](\mathbf{0}) \;=\; \mathbf{2} + 1/2 \cdot t_0 \;=\; \mathbf{2} + \mathbf{0} \;=\; \mathbf{4} - \frac{1}{2^{1-2}} \;=\; t_1.$$

Further, for $n > 0$, we obtain

$$\begin{aligned}
&\mathsf{ert}[\mathcal{D}(P_{\text{geo}})](\mathbf{0}) = \mathbf{2} + 1/2 \cdot t_n \\
&= \mathbf{2} + 1/2 \cdot \left(\mathbf{4} - \frac{1}{2^{n-2}}\right) \\
&= \mathbf{4} - \frac{1}{2^{(n+1)-2}} \\
&= t_{n+1}.
\end{aligned}$$

Hence, Theorem 7.3 yields $\mathsf{ert}[\texttt{call}\ P_{\text{geo}}, \mathcal{D}](\mathbf{0}) \;=\; \mathbf{1} + \lim_{n\to\infty} t_n \;=\; \mathbf{5}$. △

7.4 Finite Approximations of Recursive Programs

Equation (1) characterizes the runtime of recursive procedures in terms of fixed points. We next present an alternative characterization, where the runtime of procedures is given as the limit or asymptotic runtime of their finite approximations. The result will be essential in a subsequent section for extending Theorem 5.5 to recursive programs.

The notion of "finite approximation" to a procedure is materialized by its finite inlinings. Formally, the *n-th inlining* $\texttt{call}_n^{\mathcal{D}}\ P$ of a procedure P with respect to declaration $\mathcal{D}$ is defined inductively, by clauses

$$\begin{aligned}
\texttt{call}_0^{\mathcal{D}}\ P &\triangleq \texttt{halt} \\
\texttt{call}_{n+1}^{\mathcal{D}}\ P &\triangleq \texttt{skip};\ \mathcal{D}(P)[\texttt{call}\ P / \texttt{call}_n^{\mathcal{D}}\ P].
\end{aligned}$$

Here, $\mathcal{D}(P)[\mathtt{call}\,P/\mathtt{call}^{\mathcal{D}}_{n}\,P]$ denotes the syntactic replacement of every occurrence of $\mathtt{call}\,P$ in $\mathcal{D}(P)$ with $\mathtt{call}^{\mathcal{D}}_{n}\,P$ and the initial $\mathtt{skip}$ statement is used to simulate the runtime of the procedure call itself (as both consume one unit of time).[5] As so defined, the family of (call-free) programs $\mathtt{call}^{\mathcal{D}}_{n}\,P$ define a sequence of approximations to $\mathtt{call}\,P$, where $\mathtt{call}^{\mathcal{D}}_{0}\,P$ is the "poorest" approximation, while the larger the n, the more precise the approximation becomes. Observe that, in general, $\mathtt{call}^{\mathcal{D}}_{n+1}\,P$ mimics the exact behavior—and runtime—of $\mathtt{call}\,P$ for all executions that finish after at most n recursive calls. The runtime of $\mathtt{call}\,P$ can then be obtained as the limit of the runtimes of $\mathtt{call}^{\mathcal{D}}_{n}\,P$.

Theorem 7.5 (Recursive procedures as limit of finite approximations). *For any runtime $t \in \mathbb{T}$,*

$$\mathsf{ert}[\mathtt{call}\,P, \mathcal{D}](t) \;=\; \sup_n\, \mathsf{ert}[\mathtt{call}^{\mathcal{D}}_{n}\,P](t).^{6}$$

Proof. Recall that $\mathsf{ert}[\mathtt{call}\,P, \mathcal{D}] = \mathsf{lfp}\,\eta\boldsymbol{.}\,F(\eta)$, where $F(\eta) = \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\eta}$. Since F is continuous (see Appendix B.6), we can apply Kleene's Fixed Point Theorem to express $\mathsf{lfp}\,\eta\boldsymbol{.}\,F(\eta)$ as $\sup_n F^n(\bot_{\mathsf{RtEnv}})$, where F^n denotes the composition of F with itself n times and $\bot_{\mathsf{RtEnv}} = \lambda t\boldsymbol{.}\,\mathbf{0}$. The theorem then follows from

$$\forall n\boldsymbol{.}\, F^n(\bot_{\mathsf{RtEnv}}) \;=\; \mathsf{ert}[\mathtt{call}^{\mathcal{D}}_{n}\,P],$$

which is proven by induction on n. The base case is immediate. For the inductive case we rely on a result that follows from the definition of transformer $\mathsf{ert}[C]^{\sharp}_{\eta}$:

$$\mathsf{ert}[C]^{\sharp}_{\mathsf{ert}[C']} \;=\; \mathsf{ert}[C[\mathtt{call}\,P/C']] \qquad \forall\, C \in \mathsf{pRGCL}, C' \in \mathsf{pGCL}.$$

Using this result, we reason as follows:

$$\begin{aligned}
F^{n+1}(\bot_{\mathsf{RtEnv}}) &= \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{F^n(\bot_{\mathsf{RtEnv}})} && \text{(definition } F^{n+1}\text{)}\\
&= \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_{n}\,P]} && \text{(I.H.)}\\
&= \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)\big[\mathtt{call}\,P/\mathtt{call}^{\mathcal{D}}_{n}\,P\big]] && \text{(auxiliary result)}\\
&= \mathsf{ert}[\mathtt{skip};\ \mathcal{D}(P)\big[\mathtt{call}\,P/\mathtt{call}^{\mathcal{D}}_{n}\,P\big]] && \text{(Table 1)}\\
&= \mathsf{ert}[\mathtt{call}^{\mathcal{D}}_{n+1}\,P] && \text{(definition } \mathtt{call}^{\mathcal{D}}_{n+1}\,P\text{).} \quad \square
\end{aligned}$$

7.5 Mutual Recursion

For the sake of simplicity, we have so far assumed the presence of a single procedure. Notwithstanding, our ert-calculus can be readily extended to handle multiple procedures. Say we want to handle m (possibly mutually recursive) procedures $P_1, \ldots, P_m$ with declaration $\mathcal{D} : \{P_1, \ldots, P_m\} \rightarrow$ pRGCL. A runtime environment is now a tuple $\eta = (\eta_1, \ldots, \eta_m)$, where η_i is meant to provide the behavior of procedure P_i in $\mathsf{ert}[\cdot]^{\sharp}_{\eta}$, i.e., $\mathsf{ert}[\mathtt{call}\,P_i]^{\sharp}_{\eta} = \eta_i$. The action of ert on procedure calls is defined simultaneously as

$$\Big(\mathsf{ert}[\mathtt{call}\,P_1, \mathcal{D}],\ \ldots,\ \mathsf{ert}[\mathtt{call}\,P_m, \mathcal{D}]\Big) \;\triangleq\; \mathsf{lfp}\,\eta\boldsymbol{.}\,\Big(\underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P_1)]^{\sharp}_{\eta},\ \ldots,\ \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P_m)]^{\sharp}_{\eta}\Big).$$

For determining the least fixed point above, environments are compared componentwise, i.e., $(\eta_1, \ldots, \eta_m) \sqsubseteq (\nu_1, \ldots, \nu_m)$ if and only if $\eta_i \sqsubseteq \nu_i$ for all $i = 1 \ldots m$, where "$\sqsubseteq$" is the partial order over RtEnv defined in Appendix B.6. Technically, this corresponds to taking as underlying

[5] The formal definition of the syntactic replacement proceeds by a routine induction on the structure of $\mathcal{D}(P)$.

[6] Strictly speaking, $\mathtt{call}^{\mathcal{D}}_{n}\,P$ is a pGCL-program. We therefore prefer to write $\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_{n}\,P](t)$ rather than $\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_{n}\,P,\ \mathcal{D}](t)$.

ω-cpo in the fixed point above the product ω-cpo

$$(\mathsf{RtEnv}, \sqsubseteq) \times \overset{m \text{ times}}{\cdots} \times (\mathsf{RtEnv}, \sqsubseteq).$$

The proof rules from Theorem 7.3 are also easily adapted to reason about multiple procedures. For instance, rule (1) now says that if for all $i = 1 \ldots m$:

$$\mathsf{ert}[\mathtt{call}\, P_1](t_1) \preceq \mathbf{1} + u_1, \ldots, \mathsf{ert}[\mathtt{call}\, P_m](t_m) \preceq \mathbf{1} + u_m \;\Vdash\; \mathsf{ert}[\mathcal{D}(P_i)](t_i) \preceq u_i,$$

then also for all $i = 1 \ldots m$,

$$\mathsf{ert}[\mathtt{call}\, P_i, \mathcal{D}](t_i) \preceq \mathbf{1} + u_i.$$

The rule reasons about all the procedures simultaneously. Roughly speaking, the rule premise requires deriving the runtime specification for the body of each procedure P_i, assuming the corresponding specification for all procedure calls in it. The rule conclusion establishes the runtime specification of the set of procedures altogether. The other two rules from Theorem 7.3 admit a similar adaptation.

Theorem 7.5 characterizing (the expected runtime of) recursive procedures as the limit of their nth inlinings naturally extends to multiple procedures. We only need to adapt the definition of the n-inlining $\mathtt{call}_n^{\mathcal{D}}\, P_i$ of procedure P_i so as to inline the calls of all procedures:

$$\mathtt{call}_{n+1}^{\mathcal{D}}\, P_i \triangleq \mathtt{skip};\, \mathcal{D}(P_i)[\mathtt{call}\, P_1/\mathtt{call}_n^{\mathcal{D}}\, P_1, \ldots, \mathtt{call}\, P_m/\mathtt{call}_n^{\mathcal{D}}\, P_m].$$

7.6 Operational Model

To incorporate recursion into our operational (Markov chain) model of programs, we need to keep track of the procedures that have been invoked during the current program execution but have not yet terminated. This knowledge is necessary upon the termination of a procedure to determine whether this termination represents the "entire" program termination or the execution is to be continued with the remainder statements of the caller.

This kind of book-keeping of procedure calls is done by extending MCs to *pushdown Markov chains* (PMC) [24] whose verification has been studied by Brázdil, Esparza, and Kucera [6, 7, 33]. Formally, a PMC is a tuple $\mathcal{P} = (\mathcal{S}, \Gamma, \gamma_0, \mathbf{P}, s_{init}, rew)$, where $\mathcal{S}$ is a countable, nonempty set of *control states*, Γ is a finite *stack alphabet*, $\gamma_0 \in \Gamma$ is a special *bottom-of-stack* symbol,

$$\mathbf{P} : \mathcal{S} \times \Gamma \times \mathcal{S} \to [0,1] \times (\Gamma \setminus \{\gamma_0\})^*$$

is a *pushdown transition probability function*, $s_{init} \in \mathcal{S}$ is the *initial control state*, and $rew : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is a *reward function*. We assume that the topmost symbol of a stack $\alpha = \gamma \cdot \alpha' \in \Gamma \cdot \Gamma^\star$ corresponds to the leftmost symbol γ in α.

In contrast to standard Markov chains, each transition of a PMC additionally depends on the topmost stack symbol, which is popped from the stack upon each transition. Moreover, a PMC may push zero or more stack symbols—with the exception of bottom-of-stack symbol γ_0—onto the stack. For convenience, we restrict ourselves to PMCs pushing at most one new symbol onto the stack at once. Thus, we write

- $s \xrightarrow{p, \text{push}(\gamma)} s'$ instead of $\mathbf{P}(s, \alpha, s') = (p, \gamma \cdot \alpha)$,
- $s \xrightarrow{p, \text{pop}(\gamma)} s'$ instead of $\mathbf{P}(s, \gamma, s') = (p, \varepsilon)$,
- $s \xrightarrow{p} s'$ instead of $\mathbf{P}(s, \gamma, s') = (p, \gamma)$, and
- $s \xrightarrow{p, \text{empty}} s'$ instead of $\mathbf{P}(s, \gamma_0, s') = (p, \gamma_0)$

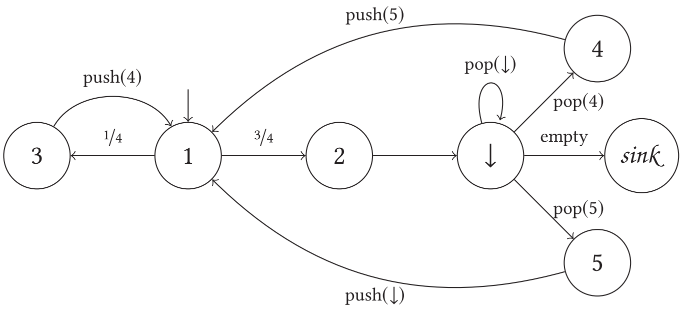


Fig. 4. Illustration of PMC from Example 7.6.

to denote that PMC $\mathcal{P}$ moves with probability p from state s to s' and pushes $\gamma \in \Gamma$ on the stack, pops γ from the stack, ignores the stack, and has an empty stack, respectively. Again, transition probabilities $p = 1$ are usually omitted from figures.

A state-stack pair $(s, \alpha) \in \mathcal{S} \times \Gamma^\star$ is called a *configuration.* Then, all notions defined over states of MCs introduced in Section 5, such as paths, cumulative reward, and expected reward, can be lifted to corresponding notions defined over configurations of PMCs. In particular, a *path* in a PMC is a finite sequence $\pi = (s_1, \alpha_1) \dots (s_n, \alpha_n)$ such that for each $1 \leq i < n$ with $\alpha_i = \gamma \cdot \alpha'$, we have $\mathbf{P}(s_i, \gamma, s_{i+1}) = (p, \beta)$ with $p > 0$ and $\alpha_{i+1} = \beta \cdot \alpha'$. The set of all paths in a PMC $\mathcal{P}$ starting in configuration (s, α) (reaching a goal state in $\mathcal{T} \subseteq \mathcal{S}$) is given by $\mathsf{Paths}(\mathcal{P}, s, \alpha)$ ($\mathsf{Paths}(\mathcal{P}, s, \alpha, \mathcal{T})$). Then the expected reward of a PMC $\mathcal{P}$ eventually reaching a nonempty set $\mathcal{T}$ of target configurations from its initial configuration is defined as

$$\mathsf{ExpRew}^{\mathcal{P}}(\mathcal{T}) \triangleq \sum_{\pi \in \mathsf{Paths}(\mathcal{M}, s_{\mathrm{init}}, \gamma_0, \mathcal{T})} \mathbf{P}(\pi) \cdot rew(\pi)$$

if $\mathcal{T}$ is reached almost surely from initial configuration $(s_{\mathrm{init}}, \gamma_0)$. Otherwise, let $\mathsf{ExpRew}^{\mathcal{P}}(\mathcal{T}) \triangleq \infty$.

Example 7.6. Consider the PMC $(\mathcal{S}, \Gamma, \gamma_0, \mathbf{P}, s_{\mathrm{init}}, rew)$ with states $\mathcal{S} = \{1, 2, 3, 4, 5, \downarrow, \mathit{sink}\}$, stack alphabet $\Gamma = \{\gamma_0, 4, 5, \downarrow\}$, and initial state $s_{\mathrm{init}} = 1$. The pushdown transition probability function $\mathbf{P}$ is depicted in Figure 4. Moreover, the reward *rew* is 1 for states 1,2,3,4,5 and 0 for states ↓ and *sink*, respectively.

Starting in configuration $(1, \gamma_0)$, this PMC first randomly chooses whether to move to state 2 (with probability 3/4) or state 3 (with 1/4). Say the left transition in Figure 4 is chosen; we move to 3. After that 4 is pushed on the stack and we move back to state 1. Thus, the current configuration is $(1, 4 \cdot \gamma_0)$. Now, assume the right transition is chosen; we move to 2 and subsequently to ↓. Since the topmost stack symbol is 4, we move to state 4; the current configuration is $(4, \gamma_0)$. Then 5 is pushed onto the stack and we are in state 1 again. If state ↓ is eventually reached with an empty stack, we end up in the sink state *sink*. △

Coming back to probabilistic programs, we now show how the expected runtime of pRGCL programs is related to the expected reward of PMCs. As in Section 5, we assume a canonical labeling of programs and employ auxiliary functions stmt, first, and second to denote the program statement, the first successor, and the second successor of a label, respectively. If no such successor exists, these functions yield ↓. Furthermore, in addition to the initial label of a pRGCL program, we need the initial label of each procedure. To that end, let

$$\mathsf{init} : \mathsf{pRGCL} \cup \{P_1, \dots, P_m\} \rightarrow \mathsf{Lab}_*,$$

where $P_1, \dots, P_m$ are the available procedures and Lab_* denotes the set of labels used in a program.

Example 7.7. Consider the pRGCL program $\langle C, \mathcal{D} \rangle$, where $\mathcal{D}(P) = C$ and C is the labeled recursive probabilistic program

$$C: \ \mathtt{if}\ \left([{}^{3}\!/\!_{4} \cdot \langle \mathsf{true} \rangle + {}^{1}\!/\!_{4} \cdot \langle \mathsf{false} \rangle]^{1}\right) \left\{[\mathtt{skip}]^{2}\right\}\ \mathtt{else}\ \left\{[\mathtt{call}\, P]^{3} ; [\mathtt{call}\, P]^{4} ; [\mathtt{call}\, P]^{5}\right\}.$$

Moreover, the auxiliary functions from above are given by

$$\mathsf{init}(C) = \mathsf{init}(P) = 1, \quad \mathsf{first}(1) = 2, \quad \mathsf{second}(1) = 3, \quad \mathsf{first}(3) = 4, \quad \mathsf{first}(4) = 5.$$

In every other case, first and second are mapped to ↓. △

Using these notions, the Markov chain interpretation of pGCL programs in Section 5 naturally extends to a *pushdown* Markov chain interpretation of pRGCL programs. Intuitively, each program label may be pushed onto the stack. Whenever a procedure terminates, such a label is popped from the stack and the execution is continued at the popped label. Thus, apart from a new rule for procedure calls, the rules determining the transition probability function are exactly the same is in Figure 1, where the stack is ignored entirely. The only exception is the termination rule [terminated]: Our PMC first tries to pop a label—the return address—from the stack and continues execution at the popped label. If no such label is on the stack, the program has successfully terminated. Consequently, we obtain the following two new rules:

$$\frac{\mathsf{stmt}(\ell) = \downarrow}{\langle \ell, \sigma \rangle \xrightarrow{\text{pop}(\ell')} \langle \ell', \sigma \rangle}\ [\text{return}] \qquad \frac{\mathsf{stmt}(\ell) = \downarrow}{\langle \ell, \sigma \rangle \xrightarrow{\text{empty}} \langle\, \mathit{sink}\, \rangle}\ [\text{terminated}].$$

In addition to that, a new rule for procedure calls is needed. This rule first pushes the return address of the current procedure, i.e., a procedure's direct successor, onto the stack and then executes the first statement of the called procedure:

$$\frac{\mathsf{stmt}(\ell) = \mathtt{call}\, P_i \quad \mathsf{first}(\ell) = \ell'}{\langle \ell, \sigma \rangle \xrightarrow{\text{push}(\ell')} \langle \mathsf{init}(P_i), \sigma \rangle}\ [\text{call}].$$

Since we assume procedure calls to consume one unit of time, the reward of states containing a program label corresponding to a procedure call is set to 1. The rewards of other states are the same as for nonrecursive programs. Formally:

Definition 7.8 (PMC of recursive programs). Given a pRGCL program $\langle C, \mathcal{D} \rangle$, an initial state $\sigma_0 \in \Sigma$, and a continuation $t \in \mathbb{T}$, the *PMC* of C is given by $\mathcal{P}^{t}_{\sigma}[\![C, \mathcal{D}]\!] = (\mathcal{S}, \Gamma, \gamma_0, \mathbf{P}, s_{\text{init}}, \mathit{rew})$, where

- $\mathcal{S} = \{\langle \ell, \sigma \rangle \mid \ell \in \mathsf{Lab}_*,\ \sigma \in \Sigma\} \cup \{\langle\, \mathit{sink}\, \rangle\}$;
- the transition probability function $\mathbf{P}$ is given by the rules in Figure 1 (without [terminated]) and the three aforementioned rules [return], [terminated], and [call];
- $s_{\text{init}} = \langle \mathsf{init}(C), \sigma_0 \rangle$; and
- the reward function $\mathit{rew} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is defined according to Table 4. △

Example 7.9. The PMC $\mathcal{P}$ considered in Example 7.6 and depicted in Figure 4 is the operational PMC of the pRGCL program $\langle C, \mathcal{D} \rangle$, where $\mathcal{D}$ is given by $P \triangleright C$ and

$$C: \ \mathtt{if}\ \left([{}^{3}\!/\!_{4} \cdot \langle \mathsf{true} \rangle + {}^{1}\!/\!_{4} \cdot \langle \mathsf{false} \rangle]^{1}\right) \left\{[\mathtt{skip}]^{2}\right\}\ \mathtt{else}\ \left\{[\mathtt{call}\, P]^{3} ; [\mathtt{call}\, P]^{4} ; [\mathtt{call}\, P]^{5}\right\}. \quad \triangle$$

As for pGCL programs and operational Markov chains (cf. Theorem 5.5), we obtain a correspondence between pRGCL programs and operational pushdown Markov chains.

Theorem 7.10 (Correspondence theorem). *Let $\langle C, \mathcal{D} \rangle$ be a* pRGCL *program and $t \in \mathbb{T}$. Then for each $\sigma \in \Sigma$, we have*

$$\mathsf{ExpRew}^{\mathcal{P}^{t}_{\sigma}[\![C, \mathcal{D}]\!]}(\langle\, \mathit{sink}\, \rangle) \ = \ \mathsf{ert}[C, \mathcal{D}](t)(\sigma).$$

Table 4. Definition of the Reward Function $rew : \mathcal{S} \to \mathbb{R}_{\geq 0}$ of Operational PMCs for Recursive pRGCL Programs

s	$\mathbf{stmt}(\ell)$	$\boldsymbol{rew}(s)$
$\langle \ell, \sigma \rangle$	$\downarrow$	$t(\sigma)$
$\langle \ell, \sigma \rangle$	$\mathtt{skip}$, $x :\approx\ \mu$, $\mathtt{if}\ (\xi)\ \{C_1\}\ \mathtt{else}\ \{C_2\}$, $\mathtt{while}\ (\xi)\ \{C\}$, or $\mathtt{call}\ \ P$	1
$\langle \ell, \sigma \rangle$	$\mathtt{empty}$, $\mathtt{halt}$	0
$\langle\ \mathit{sink}\ \rangle$		0

Proof. By induction on the structure of pRGCL programs. See Appendix B.10. □

Example 7.11. We exploit the correspondence between pRGCL programs and operational PMCs to analyze the expected runtime of the PMC from Example 7.9. More precisely, we show that C terminates, on average, after at most 10 units of time. This follows, in turn, from showing that

$$\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](\mathbf{0}) \preceq \mathbf{11}$$

as our runtime model assumes that the runtime of a procedure call (P in this case) is one unit of time more than the runtime of its body (C in this case). To prove the above runtime specification, we apply the first proof rule from Theorem 7.3, taking $u = \mathbf{10}$. We have to derive $\mathsf{ert}[C](0) \preceq \mathbf{10}$ assuming for the recursive calls that $\mathsf{ert}[\mathtt{call}\, P](0) \preceq \mathbf{11}$.

$$\begin{aligned}
\mathsf{ert}[C](\mathbf{0}) &= \mathbf{1} + \tfrac{3}{4} \cdot 1 + \tfrac{1}{4} \cdot \mathsf{ert}[\mathtt{call}\, P; \mathtt{call}\, P; \mathtt{call}\, P](\mathbf{0}) \\
&= \mathbf{1} + \tfrac{3}{4} \cdot 1 + \tfrac{1}{4} \cdot \mathsf{ert}[\mathtt{call}\, P](\mathsf{ert}[\mathtt{call}\, P](\mathsf{ert}[\mathtt{call}\, P](\mathbf{0}))) \\
&\preceq \mathbf{1} + \tfrac{3}{4} \cdot 1 + \tfrac{1}{4} \cdot \mathsf{ert}[\mathtt{call}\, P](\mathsf{ert}[\mathtt{call}\, P](\mathbf{11})) && \text{(assumption)} \\
&= \mathbf{1} + \tfrac{3}{4} \cdot 1 + \tfrac{1}{4} \cdot (\mathbf{11} + \mathsf{ert}[\mathtt{call}\, P; \mathtt{call}\, P](\mathbf{0})) && \text{(Thm. 7.2, const. prop.)} \\
&\preceq \mathbf{1} + \tfrac{3}{4} \cdot 1 + \tfrac{1}{4} \cdot (\mathbf{11} + \mathbf{11} + \mathbf{11}) && \text{(repeat previous two steps twice)} \\
&= \mathbf{10}. && \text{(algebra)}
\end{aligned}$$

By Theorem 7.10, we can then conclude that the expected runtime of C, or equivalently, the expected reward of the PMC in Figure 4 of reaching $\langle\ \mathit{sink}\ \rangle$, is bounded by 10. △

In the previous example, we exploited the correspondence between ert and operational PMCs to manually compute an upper bound on the expected reward of a PMC. For PMCs with finitely many control states, however, the converse direction is usually more desirable. Expected rewards of finite-state PMCs can be computed by solving linear recurrences [6]. The following result is a direct consequence of their approach.

Corollary 7.12. *Let $\langle C, \mathcal{D} \rangle$ be a* pRGCL *program whose PMC has finitely many control states. Then the expected runtime* $\mathsf{ert}[C, \mathcal{D}](\mathbf{0})$ *is computable in polynomial space.*

8 RELATION TO EXPECTATION TRANSFORMERS

Expectation transformers are the probabilistic counterpart of Dijkstra's predicate transformers approach to program semantics. We now establish a connection between expectation transformers and our runtime transformers and exploit this connection to derive further algebraic properties of our runtime transformer.

8.1 Expectation Transformers Semantics of Programs

In his seminal work, Dijkstra [12] introduced the predicate transformer wp (standing for **w**eakest **p**recondition) to reason about the semantics of a simple imperative language. Intuitively, for a program C and predicate—or postcondition—Q,

$$\mathsf{wp}[C](Q)$$

gives the weakest predicate—or precondition—that must hold in the initial state of C so that the execution of C terminates in a final state satisfying Q. Kozen [32] extended the transformer to probabilistic computations in terms of a PPDL, and later on McIver and Morgan [36] showed how to handle programs that combine both probabilistic and nondeterministic behavior.

The wp-semantics over probabilistic programs generalizes Dijkstra's original wp-semantics over ordinary programs twofold: First, instead of being predicates over program states, pre- and postconditions are now (nonnegative) real-valued functions over program states. Second, instead of merely evaluating a (Boolean-valued) postcondition in the final state(s) of a program, we now *measure* the expected value of a (real-valued) postcondition with respect to the distribution of final states.[7] Formally, for a probabilistic program C and postcondition $f : \Sigma \rightarrow \mathbb{R}_{\geq 0}$, we let

$$\mathsf{wp}[C](f) \;\triangleq\; \lambda\sigma\bullet\, \mathsf{E}_{[\![C]\!](\sigma)}(f),$$

where $[\![C]\!](\sigma)$ denotes the distribution of final states from executing C in initial state σ and $\mathsf{E}_{[\![C]\!](\sigma)}(f)$ denotes the expected value of f with respect to the distribution of final states $[\![C]\!](\sigma)$. Consider, for instance, the program from Example 3.3 that simulates a truncated geometric distribution:

```
$C_{trunc}$ :  if $\left({}^{1}\!/_{2} \cdot \langle \mathsf{true} \rangle + {}^{1}\!/_{2} \cdot \langle \mathsf{false} \rangle\right)$ {$succ := \mathsf{true}$} else {
                   if $\left({}^{1}\!/_{2} \cdot \langle \mathsf{true} \rangle + {}^{1}\!/_{2} \cdot \langle \mathsf{false} \rangle\right)$ {$succ := \mathsf{true}$}
                   else {$succ := \mathsf{false}$}
               }.
```

For this program we have

$$\mathsf{wp}[C_{trunc}](f) \;=\; \lambda\sigma \cdot\ \tfrac{3}{4} \cdot f(\sigma[succ/\mathsf{true}]) + \tfrac{1}{4} \cdot f(\sigma[succ/\mathsf{false}]).$$

Observe that, in particular, if $[A]$ denotes the indicator function of a predicate A over program states, $\mathsf{wp}[C]([A])(\sigma)$ gives the probability of (terminating and) establishing A after executing C from state σ. For instance, we can determine the probability that, from an initial state σ, C_{trunc} terminates in a final state where $succ$=true by

$$\mathsf{wp}[C_{trunc}]([succ{=}\mathsf{true}])(\sigma) \;=\; \tfrac{3}{4} \cdot 1 + \tfrac{1}{4} \cdot 0 \;=\; \tfrac{3}{4}.$$

Formally, the transformer wp operates on unbounded, so-called *expectations* [36] in

$$\mathbb{E} \triangleq \left\{ f \,\middle|\, f : \Sigma \rightarrow \mathbb{R}^{\infty}_{\geq 0} \right\},$$

and thus has type

$$\mathsf{wp}[\,\cdot\,] : \mathsf{pGCL} \rightarrow (\mathbb{E} \rightarrow \mathbb{E}).$$

We include infinity in the range of expectations because even if $f(\sigma) < \infty$ for all $\sigma \in \Sigma$, the expected value $\mathsf{wp}[C](f)(\sigma)$ can be infinite for some initial state σ, and we need a homogeneous treatment of the domain (i.e., postexpectations) and range (i.e., preexpectations) of wp. Observe,

[7] Strictly speaking, we consider *sub*distributions of final states, where the missing mass captures the probability of nontermination.

Table 5. Rules for Defining the Weakest Pre-Expectation Transformer wp

C	$\mathbf{wp}[C](f)$
`empty`	f
`skip`	f
`halt`	$\mathbf{0}$
$x :\approx\ \mu$	$\lambda\sigma\bullet\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\bullet\, f[x/v](\sigma))$
$C_1; C_2$	$\mathsf{wp}[C_1](\mathsf{wp}[C_2](f))$
`if` (ξ) $\{C_1\}$ `else` $\{C_2\}$	$[\![\xi : \mathsf{true}]\!] \cdot \mathsf{wp}[C_1](f) + [\![\xi : \mathsf{false}]\!] \cdot \mathsf{wp}[C_2](f)$
`while` (ξ) $\{C'\}$	$\mathsf{lfp}\, X\bullet\, [\![\xi : \mathsf{false}]\!] \cdot f + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{wp}[C'](X)$

$\mathsf{E}_\eta(h) \triangleq \sum_v \mathsf{Pr}_\eta(v) \cdot h(v)$ represents the expected value of (random variable) h with respect to distribution η; $f[x/v] \triangleq \lambda\sigma\bullet f(\sigma[x/v])$, where $\sigma[x/v]$ is the state obtained by updating in σ the value of x to v; finally, $\mathsf{lfp}\, X\bullet F(X)$ represents the least fixed point of transformer $F : \mathbb{E} \to \mathbb{E}$ with respect to the pointwise ordering on $\mathbb{E}$.

moreover, that the set of expectations $\mathbb{E}$ coincides with the set of runtimes $\mathbb{T}$, but we prefer to distinguish the two sets because they are to represent different objects.

In a similar vein to the (runtime) transformer $\mathsf{ert}[C]$, (expectation) transformer $\mathsf{wp}[C]$ can be defined by induction on the structure of C, following the rules in Table 5. Most of the rules are self-explanatory and akin to those in Table 1 for the $\mathsf{ert}[C]$ transformer.

8.2 Relation between Expectation and Runtime Transformers

We will now establish the relation between expectation and runtime transformers.

To gain some intuition about this relationship, recall the inductive definitions of wp (cf. Table 5) and ert (cf. Table 1). For every atomic program statement C, we observe that $\mathsf{ert}[C](t)$, where $t \in \mathbb{T}$ is some postruntime, can be decomposed into the time consumed by executing C, i.e., $\mathsf{ert}[C](0)$, and the expected value of the postruntime t, i.e., $\mathsf{wp}[C](t)$. Thus, the expected runtime of program C itself is independent of postruntime t, but t might depend on the expected values of program variables manipulated by C. For example, for a probabilistic assignment, we have

$$\mathsf{ert}[x :\approx \mu](t) \;=\; \underbrace{\mathbf{1}}_{=\mathsf{ert}[x:\approx\mu](0)} \;+\; \underbrace{\lambda\sigma\bullet\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\bullet\, t[x/v](\sigma))}_{=\,\mathsf{wp}[x:\approx\mu](t)}\,.$$

This decomposition property for expectation and runtime transformers also holds for composed programs including loops. The precise relationship between expectation and runtime transformers is given by the following theorem.

Theorem 8.1 (Connection between ert and wp). *For every* pGCL *program C and runtime $t \in \mathbb{T}$,*

$$\mathsf{ert}[C](t) \;=\; \mathsf{ert}[C](\mathbf{0}) + \mathsf{wp}[C](t).$$

More generally, for every two runtimes $t, t' \in \mathbb{T}$,

$$\mathsf{ert}[C](t + t') \;=\; \mathsf{ert}[C](t) + \mathsf{wp}[C](t').$$

Proof. By induction on the program structure. See Appendix B.11 for details. □

Combining Theorem 8.1 with the linearity of transformer wp (see, e.g., [36]), we can easily establish the subadditivity and subscaling of ert briefly mentioned in Section 3.

COROLLARY 8.2. *For all* pGCL *program C, runtimes $t, t' \in \mathbb{T}$, and constant $r \in \mathbb{R}_{\geq 0}$,*

$$\begin{aligned} \textit{Subadditivity:} \quad & \mathsf{ert}[C](t + t') \preceq \mathsf{ert}[C](t) + \mathsf{ert}[C](t'); \\ \textit{Subscaling:} \quad & \mathsf{ert}[C](r \cdot t) \succeq \min\{1, r\} \cdot \mathsf{ert}[C](t); \\ & \mathsf{ert}[C](r \cdot t) \preceq \max\{1, r\} \cdot \mathsf{ert}[C](t). \end{aligned}$$

PROOF. As for the subadditivity, we have

$$\begin{aligned} \mathsf{ert}[C](t + t') &= \mathsf{ert}[C](t) + \mathsf{wp}[C](t') && \text{(Theorem 8.1)} \\ &= \mathsf{ert}[C](t) + \mathsf{ert}[C](t') - \mathsf{ert}[C](\mathbf{0}) && \text{(Theorem 8.1)} \\ &\preceq \mathsf{ert}[C](t) + \mathsf{ert}[C](t') && (\mathsf{ert}[C](\mathbf{0}) \succeq \mathbf{0}). \end{aligned}$$

The subscaling follows from the reasoning below (we establish only one inequality; the proof argument for the other one is analogous):

$$\begin{aligned} \mathsf{ert}[C](r \cdot t) &= 1 \cdot \mathsf{ert}[C](\mathbf{0}) + \mathsf{wp}[C](r \cdot t) && \text{(Theorem 8.1)} \\ &= \underbrace{1}_{\geq \min\{1,\, r\}} \cdot \mathsf{ert}[C](\mathbf{0}) + \underbrace{r}_{\geq \min\{1,\, r\}} \cdot \mathsf{wp}[C](t) && (\mathsf{wp}[C] \text{ linear}) \\ &\succeq \min\{1, r\} \cdot (\mathsf{ert}[C](\mathbf{0}) + \mathsf{wp}[C](t)) && \text{(algebra)} \\ &= \min\{1, r\} \cdot \mathsf{ert}[C](t) && \text{(Theorem 8.1)} \quad \square \end{aligned}$$

For the sake of clarity, we have presented the results considering only pGCL programs and left out recursion. All the results are extensible to recursive programs in pRGCL as shown in [44] and its full version [43].

9 CASE STUDIES

In this section, we use the ert-calculus to analyze the runtime of three well-known randomized algorithms: we use global and incremental invariants for nonrecursive probabilistic programs to analyze the *coupon collector's problem* and a fair *one-dimensional (symmetric) random walk*, respectively. Furthermore, our proof rules for reasoning about recursive probabilistic programs are employed to reason about a *randomized binary search* algorithm.

9.1 The Coupon Collector's Problem

To illustrate the use of our proof rule for loops based on global invariants, we apply the ert-calculus to the coupon collector's problem. This problem arises from the following scenario:[8] suppose each box of cereal contains one of N different types of coupons, and once a consumer has collected a coupon of each type, he or she can trade them for a prize. The problem is to determine the average number of cereal boxes a coupon collector has to buy in order to collect at least one coupon of each type. It is assumed that each coupon type occurs with the same probability in the cereal boxes.

We can model the coupon collector's problem by a probabilistic program C_{cp} as follows:

```
C_cp :   cp := [0, ..., 0]; i := 1; x := N;
C_out :  while (x > 0) {
```

[8] The problem formulation presented here is taken from [37].

```
$C_{in}$ :      while ($cp[i] \neq 0$) {
                    $i :\approx$ Unif[1 . . . $N$]
                };
                $cp[i] := 1; x := x - 1$
            }.
```

Here, we use an array *cp* as a shortcut for N distinct program variables. The array *cp* (of size N) is initialized with 0s, and whenever we obtain the first coupon of type i, we set $cp[i]$ to 1. The outer loop C_{out} is iterated N times, and in each iteration of the outer loop we collect a new—unseen—coupon type. The collection of the new coupon type is performed by the inner loop C_{in}.

In order to analyze the runtime of program C_{cp}, we need to find a suitable invariant for the outer loop C_{out}. To this end, we propose the following global upper invariant I:

$$I \triangleq 1 + \sum_{\ell=0}^{\infty} [x > \ell] \cdot \left(\mathbf{4} + 2 \cdot \sum_{k=0}^{\infty} \left(\frac{\#col + \ell}{N} \right)^k \right) - 2 \cdot [cp[i] = 0] \cdot [x > 0] \cdot \sum_{k=0}^{\infty} \left(\frac{\#col}{N} \right)^k ,$$

where $\#col \triangleq \sum_{i=1}^{N} [cp[i] \neq 0]$ denotes the number of coupons that have already been collected. This invariant was essentially obtained by performing a few fixed-point iterations to approximate the least fixed point that determines $\mathrm{ert}[C_{out}]$ $(\mathbf{0})$ and then generalizing the result.

What Is the Intuition Underlying This Invariant? At least one time unit is consumed by the outer loop, because the loop guard is evaluated at least once. Variable ℓ represents the number of iterations of the outer loop. Since x is decremented in every iteration, C_{out} performs x iterations; all summands in I are thus zero if $x \leq \ell$. How long, then, is the expected runtime of every loop iteration? At least four units of time are required to evaluate the guard of the outer loop and the guard of the inner loop for the first time, and to perform the two assignments at the end of the loop body of C_{out}, respectively. Every iteration of the inner loop requires two additional units of time to evaluate the loop guard and perform an assignment. The expected number of iterations performed by the inner loop is expressed by the second sum, where k intuitively represents the number of times we inter the body of the inner loop. However, in the very first iteration of the outer loop, we have $[cp[i] = 0]$. In other words, no iteration of the inner loop is performed; i.e., the inner sum represents runtime that is never actually consumed by the program. We thus subtract this illegally added runtime. Furthermore, notice that the probability of performing the k-t iteration of the inner loop depends on the number of already collected coupons, i.e., the number $\#col$ of initially collected coupons plus ℓ, due to the fact that one coupon is collected in every iteration of the outer loop.

A detailed verification that I is indeed an upper global invariant is found in Appendix C.2 (this invariant verification further requires exhibiting a suitable invariant for the inner loop C_{in}).

The Expected Runtime of C_{cp}. We are now in a position to compute an upper bound for $\mathrm{ert}[C_{cp}](\mathbf{0})$ using the previously proposed invariant I:

$$\begin{aligned} \mathrm{ert}[C_{cp}](\mathbf{0}) &= \mathrm{ert}[cp := [0, \ldots, 0]; i := 1; x := N; C_{out}](0) \\ &= \mathbf{3} + (\mathrm{ert}[C_{out}](\mathbf{0}))\, [x/N, i/1, cp[1]/0, \ldots, cp[N]/0] \\ &\leq \mathbf{3} + I[x/N, i/1, cp[1]/0, \ldots, cp[N]/0]. && (\mathrm{ert}[C_{out}](0) \leq I) \end{aligned}$$

Inserting our invariant I in the computation of $\mathrm{ert}[C_{cp}](\mathbf{0})$ then yields

$$\begin{aligned}
\mathrm{ert}[C_{cp}](0) &\preceq \mathbf{4} + [N > 0] \cdot \left(4N + 2 \sum_{\ell=1}^{N-1} \sum_{k=0}^{\infty} \left(\frac{\ell}{N}\right)^{k}\right) && \left(\left(\textstyle\sum_{\ell=0}^{\infty} [x > \ell]\right)[x/N] = \textstyle\sum_{\ell=0}^{N-1} 1\right) \\
&= \mathbf{4} + [N > 0] \cdot \left(4N + 2 \sum_{\ell=1}^{N-1} \frac{N}{\ell}\right) && \left(\begin{array}{l}\text{geom. series and} \\ \text{sum reordering}\end{array}\right) \\
&= \mathbf{4} + [N > 0] \cdot 2N \cdot (\mathbf{2} + \mathcal{H}_{N-1}),
\end{aligned}$$

where $\mathcal{H}_{N-1} \triangleq 0 + 1/1 + 1/2 + 1/3 + \cdots + 1/N{-}1$ denotes the $(N-1)$-th harmonic number. Since the harmonic numbers approach asymptotically to the natural logarithm, we conclude that the coupon collector algorithm C_{cp} runs in expected time $\mathcal{O}(N \cdot \log(N))$.

9.2 One-Dimensional Random Walk

The probabilistic program

```
C_rw :  $x := 10;$
        while $(x > 0)$ {
C :         $x :\approx 1/2 \cdot \langle x-1 \rangle + 1/2 \cdot \langle x+1 \rangle$
        }
```

models a one-dimensional walk of a particle that starts at position $x = 10$ and moves with equal probability to the left or to the right in each turn. The random walk terminates when the particle reaches position $x = 0$. It can be shown that the program terminates with probability one [25] but requires, on average, an infinite amount of time to do so. We apply the ert-calculus to formally derive this.

It is easy to see that $\mathrm{ert}[C_{rw}](0) \preceq \infty$, so we concentrate on proving that ∞ is a lower bound on the expected runtime of C_{rw}. To that end, we derive a lower ω-invariant I_n of loop `while` $(x > 0)$ $\{C\}$ with respect to continuation $\mathbf{0}$.

Invariant Synthesis. To obtain a suitable invariant I_n, we first propose the following template:

$$I_n \;=\; 1 + \sum_{k=0}^{n} [x > k] \cdot a_{n,k}.$$

We always need one unit of time, because the loop guard is evaluated at least once. Furthermore, parameter n corresponds to the number of loop iterations of loop `while` $(x > 0)$ $\{C\}$. Intuitively, we then consider (a lower bound of) the expected time $a_{n,k}$ such that, after n loop iterations entering the loop body, the position of the particle is k. Since the random walk does not terminate within n loop iterations, we know that it suffices to consider these runtimes for $0 \leq k \leq n$.

As a next step, we show that for all nonnegative constants $a_{n,k}$, I_n is a lower ω-invariant of the loop with respect to continuation 0 whenever these $a_{n,k}$ satisfy the recurrence relation

$$\begin{aligned}
a_{0,0} &= 1, \\
a_{n+1,0} &= 2 + \tfrac{1}{2} \cdot (a_{n,0} + a_{n,1}), \\
a_{n+1,k} &= \tfrac{1}{2} \cdot (a_{n,k-1} + a_{n,k+1}) \text{ for all } 1 \leq k \leq n+1, \\
a_{n,k} &= 0 \text{ for all } k > n,
\end{aligned}$$

for $n \geq 0$. Intuitively, if the target position k after 0 further loop iterations is zero, then the expected runtime after one iteration is one due to the guard evaluation that terminates the loop. Similarly, if $k = 0$ but we have to perform $n + 1$ loop iterations, we require at least two units of time to evaluate the guard and execute the probabilistic assignment in the loop body. The remaining expected runtime is then the weighted sum of $a_{n,k-1}$ (i.e., the position is increased in the last step and it suffices to reach position $k-1$) and $a_{n,k+1}$ (i.e., the position is decreased in the last iteration and it suffices to reach position $k+1$). For the case $1 \leq k < n$, we analogously express the expected time to reach position k in terms of the expected time to reach distance $k-1$ and $k+1$, respectively. Since we are only interested in a lower bound, it suffices to omit the two additional units of time. Furthermore, if $k > n$, then we abort, because the random walk cannot terminate within n loop iterations.

Now, one suitable solution for $a_{n,k}$ is

$$a_{n,k} = \frac{1}{2^n} \left[-\binom{n}{\lfloor \frac{n-k}{2} \rfloor} + 2 \sum_{i=0}^{n-k} 2^i \binom{n-i}{\lfloor \frac{n-i-k}{2} \rfloor} \right].$$

This solution was obtained with the help of Mathematica. A formal proof showing that I_n (with the above coefficients $a_{n,k}$) is indeed a lower ω-invarant with respect to continuation 0 is found in Appendix C.1.

The Expected Runtime of C_{rw}. Now Theorem 4.6 and the fact that $\lim_{n\to\infty} a_{n,0} = \infty$ yield

$$\mathsf{ert}[\mathtt{while}\ (x > 0)\ \{C\}](0) \geq \lim_{n\to\infty} I_n \geq \lim_{n\to\infty} 1 + [x > 0] \cdot a_{n,0} = 1 + [x > 0] \cdot \infty.$$

The calculations for the aforementioned steps can be found in Appendix C.1. Altogether we have

$$\begin{aligned} \mathsf{ert}[C_{rw}](\mathbf{0}) &= \mathsf{ert}[x := 10](\mathsf{ert}[\mathtt{while}\ (x > 0)\ \{C\}](\mathbf{0})) \\ &\geq \mathsf{ert}[x := 10](\mathbf{1} + [x > 0] \cdot \infty) \\ &= \mathbf{1} + (\mathbf{1} + [x > 0] \cdot \infty)[x/10] \\ &= \mathbf{1} + (\mathbf{1} + 1 \cdot \infty) \ = \ \infty. \end{aligned}$$

Thus, $\mathsf{ert}[C_{rw}](\mathbf{0}) \geq \infty$. As the reverse inequality trivially holds, the expected runtime of the one-dimensional random walk is infinite, i.e., $\mathsf{ert}[C_{rw}](\mathbf{0}) = \infty$.

9.3 Randomized Binary Search

As our third case study, we show the applicability of our approach to randomized algorithms by analyzing a probabilistic, so-called Sherwood [35], variant of the classical recursive binary search algorithm. The main difference with the classical version is that in each recursive call the pivot element is picked uniformly at random from the remaining array, aligning, this way, the worst, best, and average case of the algorithm's runtime.

The Sherwood binary search algorithm searches for the value *val* in array $a[\mathit{left}..\ \mathit{right}]$. It is encoded by procedure B with declaration $\mathcal{D}$ presented in Figure 5. We use the random assignment $\mathit{mid} :\approx \mathtt{Unif}(\mathit{left} \ldots \mathit{right})$ to model the random selection of the pivot element. For simplicity, we assume that the random assignment is performed in constant time 1 if $\mathit{left} \leq \mathit{right}$ and that it diverges (thus has runtime ∞) if $\mathit{left} > \mathit{right}$.

As the runtime heavily depends on the input data, we restrict our input and perform a runtime analysis for those inputs where *val* does *not* occur in the array, which constitutes the worst case for the classical binary search algorithm. Under this assumption, we can distinguish two cases: *val* is either smaller than every element in the array or larger than all of them.

For the first case, the expected runtime is bounded from above by $1{+}u$, where

$$u \ = \ [\mathit{left} > \mathit{right}] \cdot \infty + 3 + [\mathit{left} < \mathit{right}] \cdot \left(5 \cdot H_{\mathit{right}-\mathit{left}+1} - {}^{1}\!/_{5}\right),$$

and H_k is the kth harmonic number. Ignoring the actual values of the constants, the intuition for the above expression is this: If $\mathit{left} > \mathit{right}$, the uniform sampling is assumed to diverge and thus we

```
$u = [left > right] \cdot \infty + 3 + [left < right]$
    $\cdot \left(5 + \sum_{i=left}^{right} \left( \frac{[\min(i+1,\ right) < right]}{right - left + 1} \cdot \left(5 \cdot H_{right - \min(i+1,\ right) < right + 1} - 5/2\right)\right)\right)$
B ▷ 1:  mid := Unif(left, right);
        $2 + [left < right] \cdot \big(2 + \big[a[mid] < val\big] \cdot (3$
            $+ [\min(mid+1,\ right) < right] \cdot \big(5 \cdot H_{right - \min(mid+1,\ right)+1} - 5/2\big)$
            $+ \big[a[mid] > val\big] \cdot (\cdots)\big)$
    2:  if (left < right){
            $3 + \big[a[mid] < val\big] \cdot u[left/\min(mid+1,\ right)] + \big[a[mid] > val\big] \cdot (\cdots)$
    3:      if (a[mid] < val){
                $2 + u[left/\min(mid+1,\ right)]$
    4:          left := min(mid + 1, right);
                $1 + u$
    5:          call B
                $0$
    6:      } else {
                $2 + \big[a[mid] > val\big] \cdot (\cdots)$
    7:          if (a[mid] > val){
                    $2 + u[right/\max(mid-1,\ left)]$
    8:              right := max(mid − 1, left);
                    $1 + u$
    9:              call B
                    $0$
   10:          } else { $1$ skip $0$ } $0$
   11:      } $0$
   12:  } else { $1$ skip $0$ } $0$
```

Fig. 5. Declaration (boldface black) of the randomized binary search procedure B together with the runtime analysis (lightface gray) for the case that every value occurring in $a[left..\, right]$ is smaller than val. We write j **C** h for $\mathrm{ert}[C, \mathcal{D}](h) \preceq j$. H_k stands for the kth harmonic number.

get infinite runtime. If $left = right$, $\mathcal{O}(1)$ steps are performed, but the procedure is not recursively called. Finally, if $left < right$, the test for $a[mid] < val$ will by assumption (val is smaller than every value in the array a) fail and the test $a[mid] > val$ will succeed. The binary search procedure is then called recursively on the randomly selected new interval and this takes on average $\mathcal{O}(H_{right-left+1})$ units of time.

For formally showing

$$\mathrm{ert}[\mathtt{call}\, B, \mathcal{D}](\mathbf{0}) \preceq \mathbf{1} + u$$

by application of Theorem 7.3 (1), we have to establish

$$\mathrm{ert}[\mathtt{call}\, B, \mathcal{D}](\mathbf{0}) \preceq \mathbf{1} + u \quad \Vdash \quad \mathrm{ert}[\mathcal{D}(B), \mathcal{D}](\mathbf{0}) \preceq u.$$

The details of this derivation are provided in Figure 5. In the annotations of Line 9, we use the assumption $\mathrm{ert}[\mathtt{call}\, B, \mathcal{D}](\mathbf{0}) \preceq \mathbf{1} + u$. All other annotations are straightforward applications of the rules in Table 1 while keeping in mind that val is smaller than every element in the array. The topmost annotation containing the sum is equal to u by algebraic reasoning. All in all, we have proven $\mathrm{ert}[\mathcal{D}(B), \mathcal{D}](\mathbf{0}) \preceq u$ assuming $\mathrm{ert}[\mathtt{call}\, B, \mathcal{D}](\mathbf{0}) \preceq \mathbf{1} + u$. By Theorem 7.3 (1), we now know that

$\mathrm{ert}[\mathtt{call}\ B, \mathcal{D}](\mathbf{0}) \leq \mathbf{1} + u$ for all initial states in which *val* is smaller than every element in the array.

Similarly, we can show that when *val* is greater than every element in the array, the expected runtime is upper bounded by $1 + u$, with

$$u \ = [\mathit{left} > \mathit{right}] \cdot \infty + 3 + [\mathit{left} < \mathit{right}] \cdot \left(6 \cdot H_{\mathit{right}-\mathit{left}+1} - 3\right).$$

The verification for this case is analogous and therefore omitted here.

Combining the two cases, we conclude that when the sought-after value does not occur in the array, the algorithm terminates in expected time in $O(\log n)$, where $n = \mathit{right} - \mathit{left} + 1$ is the size of the array, since $H_k \in \Theta(\log k)$.

10 RELATED WORK

Resource Analysis of Deterministic Programs. Several works apply wp-style or Floyd-Hoare-style reasoning to study quantitative aspects of classical algorithms. Nielson [41, 42] provides a Hoare logic for determining upper bounds on the runtime of deterministic programs. Our approach applied to such programs yields the tightest upper bound on the runtime that can be derived using Nielson's approach. Arthan et al. [1] provide a general framework for sound and complete Hoare-style logics and show that an instance of their theory can be used to obtain upper bounds on the runtime of while-programs. Hickey and Cohen [20] automate the average-case analysis of deterministic programs by generating a system of recurrence equations derived from a program whose efficiency is to be analyzed. They build on Kozen's seminal work [31] on the semantics of probabilistic programs. Berghammer and Müller-Olm [5] show how Hoare-style reasoning can be extended to obtain bounds on the closeness of results obtained using approximate algorithms to the optimal solution. Deriving space and time consumption of deterministic programs has also been considered by Hehner [17]. Alternative approaches for analyzing resource consumption in deterministic programs include, among others, type-checking [22], abstract interpretation [47], and worst-case execution time analysis [49].

Runtime Analysis of Probabilistic Programs. Classical techniques to analyze the runtime of randomized algorithms include probabilistic recurrence relations [29] and martingale theory. Formal reasoning about probabilistic programs goes back to Kozen [31] and has been developed further by Hehner [18] and McIver and Morgan [36]. A general abstract interpretation framework for the analysis of probabilistic programs has been given by Cousot and Monerau [11]. They capture our approach as an abstraction of a low-level trace semantics, but their work does not provide any proof rules. The work by Celiku and McIver [8] is perhaps the closest to our article. They provide a wp-calculus for obtaining performance properties of probabilistic programs, including upper bounds on expected runtimes. Their focus is on refinement. They neither provide a soundness result of their approach nor consider lower bounds. We believe that our transformer is simpler to work with in practice too. Monniaux [38] exploits abstract interpretation to automatically prove the probabilistic termination of programs using exponential bounds on the tail of the distribution. His analysis can be used to prove the soundness of experimental statistical methods to determine the average runtime of probabilistic programs. Brázdil et al. [7] study the runtime of probabilistic programs with unbounded recursion by considering probabilistic pushdown automata (pPDAs). They show (using Martingale theory) that for every pPDA, the probability of performing a long run decreases exponentially (polynomially) in the length of the run, if and only if the pPDA has a finite (infinite) expected runtime. As opposed to our program verification technique, [7] considers reasoning at the operational level. Fioriti and Hermanns [14] proposed a typing scheme for deciding almost-sure termination. They showed, among others, that if a program is well typed, then it almost surely terminates. This result does not cover positive almost-sure termination; that is, their approach cannot be used to obtain that the runtime is infinite.

Automated Expected Runtime Analysis. Chatterjee et al. [10] recently presented a linear-time algorithm to derive (logarithmic, linear, or almost-linear) bounds on expected runtimes of randomized algorithms whose runtime is described by a set of recurrence relations. The key idea is to overapproximate terms in a recurrence relation through integral and Taylor expansion enabling one to obtain bounds by comparing the leading terms of pseudo-polynomials. This technique derives bounds for randomized univariate and separable bivariate recurrence relations and is applicable to classical algorithms such as quick-select, Sherwood randomized search, and the coupon collector.

The ert-calculus developed in this article provides a solid basis for the automated analysis of expected runtimes. This is witnessed by some recent follow-up works. Ngo et al. [40] combined the principles of the ert-calculus with existing automated amortized resource analysis techniques. This results in an automated approach to derive upper bounds (as symbolic polynomials) on the expected runtime of probabilistic programs. The correctness is shown by proving that the technique provides upper bounds on the expected runtime of a program as defined by our ert-calculus. In essence, their technique assumes potential functions to be a linear combination of base functions and derives using inference rules akin to those in this article of a system of in-equations that is solved by an LP solver. Experimental results show that this works for an interesting class of sample programs.

Batz et al. [4] showed how the ert-calculus can be used to obtain exact expected sampling times of Bayesian networks in a fully automated fashion. The key idea here is that loops in the probabilistic programs describing such networks are statistically independent, enabling obtaining closed-form symbolic expressions for their expected runtime. An experimental evaluation on Bayesian network benchmarks demonstrates that ill-conditioned networks—resulting in very large simulation times—can be automatically inferred within less than a second.

11 CONCLUSION

We have presented a wp-style calculus for reasoning about the expected runtime and positive almost-sure termination of randomized algorithms. Our main contribution consists of several sound and complete proof rules for obtaining upper as well as lower bounds on the expected runtime of loops. We applied these rules to analyze the expected runtime of a variety of example algorithms including the well-known coupon collector problem. While finding invariants is, in general, a challenging task, we were able to find correct invariants by considering a few loop unrollings most of the time. Hence, we believe that our proof rules are natural and widely applicable, and provide a viable alternative to existing techniques to determine the expected runtime. The approach is a conservative extension of Nielson's approach for reasoning about the runtime of deterministic programs and our calculus is sound with respect to a simple operational model defined in terms of (push-down) Markov chains. Toward automation of our approach, an important step is to develop techniques for automated loop-invariant synthesis; initial approaches for probabilistic programs can be found in, e.g., [3, 13, 30].

APPENDIXES

A RECAP OF AUXILIARY RESULTS

For the sake of self-containment, we recall here some well-known theorems that we use to establish our main results.

Theorem A.1 (Lebesgue's Monotone Convergence Theorem). *Let $(f_n)_{n\in\mathbb{N}}$ be a sequence of functions of type $A \to \mathbb{R}_{\geq 0}^{\infty}$ such that $f_n(a) \leq f_{n+1}(a)$ for all $a \in A$. Then, for every probability distribution ν over A, we have*

$$\mathsf{E}_{\nu}(\sup_n f_n) \ = \ \sup_n \mathsf{E}_{\nu}(f_n),$$

where $\mathsf{E}_{\nu}(f)$ denotes the expected value of f with respect to ν and $\sup_n f_n$ is taken pointwise.

Proof. See, e.g., [46, Ch. 21]. □

Theorem A.2 (Monotone Sequence Theorem). *If $(a_n)_{n\in\mathbb{N}}$ is a monotonically increasing sequence in $\mathbb{R}_{\geq 0}^{\infty}$, then*

$$\sup_n a_n \ = \ \lim_{n\to\infty} a_n.$$

Proof. See, e.g., [45, Ch. 2]. □

Theorem A.3 (Kleene's Fixed-Point Theorem). *Let $(A, \preceq)$ be an ω-cpo with bottom element $\perp$ and let $F : A \to A$ be continuous.*[9] *Then F admits a least fixed-point $\mathsf{lfp}\,(F)$, which can moreover be obtained as*

$$\mathsf{lfp}\,(F) \ = \ \sup_n F^n(\perp).$$

Here, F^n denotes the composition of F with itself n times, i.e., $F^0 = id$ and $F^{n+1} = F \circ F^n$.

Proof. See, e.g., [48, Ch. 1]. □

Theorem A.4 (Park's Theorem). *Let $(A, \preceq)$ be an ω-cpo with bottom element and let $F : A \to A$ be continuous. Then, for every $a \in A$,*

$$F(a) \preceq a \ \Rightarrow \ \mathsf{lfp}\,(F) \preceq a.$$

Proof. See [48, Ch. 1]. □

Lemma A.5 (Diagonalization of Doubly Indexed Chains). *Let $a_{n,m}$ be elements of an ω-cpo $(A, \preceq)$ such that $a_{n,m} \preceq a_{n',m'}$ whenever $n \leq n'$ and $m \leq m'$. Then,*

$$\sup_n (\sup_m a_{n,m}) \ = \ \sup_m (\sup_n a_{n,m}) \ = \ \sup_i a_{i,i}.$$

Proof. See [50, Ch. 8]. □

Lemma A.6 (Relational Fixed-Point Fusion). *Let $(\mathcal{D}_1, \preceq_1)$, $(\mathcal{D}_2, \preceq_2)$ and $(\mathcal{D}, \preceq)$ be ω-cpos with bottom elements $\perp_1$, $\perp_2$, and $\perp$, respectively. Moreover, let*

$$F_1 : \mathcal{D}_1 \to \mathcal{D}_1 \qquad F_2 : \mathcal{D}_2 \to \mathcal{D}_2 \qquad f_1 : \mathcal{D}_1 \to \mathcal{D} \qquad f_2 : \mathcal{D}_2 \to \mathcal{D}$$

be continuous and $h_1, h_2 : \mathcal{D} \to \mathcal{D}$. If

1. *$\forall d_1\bullet\, f_1(F_1(d_1)) \preceq h_1(f_1(d_1))$ and $\forall d_2\bullet\, f_2(F_2(d_2)) \preceq h_2(f_2(d_2))$,*
2. *$f_1(\perp_1) \preceq f_2(\mathsf{lfp}\, F_2)$ and $f_2(\perp_2) \preceq f_1(\mathsf{lfp}\, F_1)$, and*
3. *$h_1(f_2(\mathsf{lfp}\, F_2)) \preceq f_2(\mathsf{lfp}\, F_2)$ and $h_2(f_1(\mathsf{lfp}\, F_1)) \preceq f_1(\mathsf{lfp}\, F_1)$,*

then

$$f_1(\mathsf{lfp}\, F_1) \ = \ f_2(\mathsf{lfp}\, F_2).$$

Proof. See [43, App. A.6]. □

[9] A function $F : A \to A$ is said to be *continuous* if and only if it preserves suprema of ω-chains; i.e., for every ω-chain $a_0 \preceq a_1 \preceq \cdots$ in A we have $\sup_n F(a_n) = F(\sup_n a_n)$.

B OMITTED PROOFS

B.1 Proof of Lemma 3.2 (Continuity of ert)

We prove that for every program C, the transformer $\mathsf{ert}[C]$ is continuous; i.e., for every ω-chain of runtimes $t_0 \preceq t_1 \preceq \cdots$,

$$\mathsf{ert}[C](\sup_n t_n) \;=\; \sup_n \mathsf{ert}[C](t_n),$$

by induction on the structure of C. We consider only the cases of assignments, conditional choices, and while-loops as the proof argument for the remaining language constructs is straightforward.

Assignment. The proof relies on Lebesgue's Monotone Convergence Theorem (LMCT) recalled in Theorem A.1. We have:

$$\begin{aligned}
\mathsf{ert}[x :\approx \mu](\sup_n t_n) &= \mathbf{1} + \lambda\sigma\text{.}\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\, (\sup_n t_n)[x/v](\sigma)) && \text{(Table 1)}\\
&= \mathbf{1} + \lambda\sigma\text{.}\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\sup_n \lambda v\text{.}\, t_n[x/v](\sigma)) &&\\
&= \mathbf{1} + \lambda\sigma\text{.}\, \sup_n \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\, t_n[x/v](\sigma)) && \text{(LMCT)}\\
&= \sup_n \mathbf{1} + \lambda\sigma\text{.}\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\, t_n[x/v](\sigma)) && (\mathbf{1} \text{ is constant})\\
&= \sup_n \mathsf{ert}[x :\approx \mu](t_n). && \text{(Table 1)}
\end{aligned}$$

Conditional Choice. The proof relies on a Monotone Sequence Theorem (MST) recalled in Theorem A.2. We have:

$$\begin{aligned}
&\mathsf{ert}[\texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\}](\sup_n t_n)\\
&\quad= \mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C_1](\sup_n t_n) + [\![\xi : \mathsf{false}]\!] \cdot \mathsf{ert}[C_2](\sup_n t_n) && \text{(Table 1)}\\
&\quad= \mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \sup_n \mathsf{ert}[C_1](t_n) + [\![\xi : \mathsf{false}]\!] \cdot \sup_n \mathsf{ert}[C_2](t_n) && \text{(I.H. on } C_1,\, C_2)\\
&\quad= \mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \lim_{n\to\infty} \mathsf{ert}[C_1](t_n) + [\![\xi : \mathsf{false}]\!] \cdot \lim_{n\to\infty} \mathsf{ert}[C_2](t_n) && \text{(MST)}\\
&\quad= \lim_{n\to\infty} \mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C_1](t_n) + [\![\xi : \mathsf{false}]\!] \cdot \mathsf{ert}[C_2](t_n) &&\\
&\quad= \sup_n \mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C_1](t_n) + [\![\xi : \mathsf{false}]\!] \cdot \mathsf{ert}[C_2](t_n) && \text{(MST)}\\
&\quad= \sup_n \mathsf{ert}[\texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\}](t_n). && \text{(Table 1)}
\end{aligned}$$

While-Loop. Let $F_t(X) = \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C'](X)$ be the characteristic functional of loop $\texttt{while}\ (\xi)\ \{C'\}$. The proof relies on three facts about F_t and the lfp operator:

(1) $F_{\sup_n t_n} = \sup_n F_{t_n}$, which follows from a straightforward reasoning.
(2) $\sup_n F_{t_n}$ is continuous (in $\mathbb{T} \to \mathbb{T}$), which follows from the fact that $\langle F_{t_n} \rangle$ forms an ω-chain of continuous transformers (since by I.H. $\mathsf{ert}[C']$ is continuous) and continuous functions are closed under supremums.
(3) $\mathsf{lfp}\ : [\mathbb{T} \to \mathbb{T}] \to \mathbb{T}$ is itself continuous when restricted to the set of continuous transformers in $\mathbb{T} \to \mathbb{T}$, denoted $[\mathbb{T} \to \mathbb{T}]$ [48, Proposition 12].

We then have

$$\begin{aligned}
\mathsf{ert}[\texttt{while}\ (\xi)\ \{C'\}](\sup_n t_n) &= \mathsf{lfp}\ (F_{\sup_n t_n}) && \text{(Table 1)}\\
&= \mathsf{lfp}\ \left(\sup_n F_{t_n}\right) && \text{(Fact(1))}\\
&= \sup_n \mathsf{lfp}\ \left(F_{t_n}\right) && \text{(Facts(2)and(3))}\\
&= \sup_n \mathsf{ert}[\texttt{while}\ (\xi)\ \{C'\}](t_n). && \text{(Table 1)} \quad \square
\end{aligned}$$

B.2 Proof of Theorem 3.4 (Constant Propagation)

For any halt-free program $C \in \text{pGCL}$, any constant $k \in \mathbb{R}_{\geq 0}$, and any runtime $t \in \mathbb{T}$, we prove

$$\mathsf{ert}[C](\mathbf{k} + t) \;=\; \mathbf{k} + \mathsf{ert}[C](t)$$

by induction on the structure of C. We consider only the cases of assignments and while-loops as the proof argument for the remaining language constructs is straightforward.

Assignment. For $x :\approx \mu$, the proof relies on the linearity of the expected value operator; i.e., $\mathsf{E}_\nu(\mathbf{k} + t) = k + \mathsf{E}_\nu(t)$ provided distribution ν has total mass 1. We have

$$\begin{aligned}
\mathsf{ert}[x :\approx \mu](\mathbf{k} + t) &= \mathbf{1} + \lambda\sigma\text{.}\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\, (\mathbf{k} + t)[x/v](\sigma)) && \text{(Table 1)}\\
&= \mathbf{1} + \lambda\sigma\text{.}\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\, k + t[x/v](\sigma)) && (\mathbf{k}[x/v](\sigma) = k)\\
&= \mathbf{1} + \mathbf{k} + \lambda\sigma\text{.}\, \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\, t[x/v](\sigma)) && \text{(linearity of } \mathsf{E}\text{)}\\
&= \mathbf{k} + \mathsf{ert}[x :\approx \mu](t). && \text{(Table 1)}
\end{aligned}$$

While-Loop. Let $F_t(X) = \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C'](X)$ be the characteristic functional of loop while (ξ) $\{C'\}$. We have to show

$$\mathsf{lfp}\, F_{\mathbf{k}+t} \;=\; \mathbf{k} + \mathsf{lfp}\, F_t,$$

which is equivalent to the pair of inequalities

$$\mathsf{lfp}\, F_{\mathbf{k}+t} \;\preceq\; \mathbf{k} + \mathsf{lfp}\, F_t \qquad \text{and} \qquad \mathsf{lfp}\, F_t \;\preceq\; \mathsf{lfp}\, F_{\mathbf{k}+t} - \mathbf{k}.$$

These inequalities follow, in turn, from equalities

$$F_{\mathbf{k}+t}(\mathbf{k} + \mathsf{lfp}\, F_t) \;=\; \mathbf{k} + \mathsf{lfp}\, F_t \qquad \text{and} \qquad F_t(\mathsf{lfp}\, F_{\mathbf{k}+t} - \mathbf{k}) \;=\; \mathsf{lfp}\, F_{\mathbf{k}+t} - \mathbf{k}.$$

(This is because lfp gives the *least* fixed point of a transformer and then $F(x) = x$ implies $\mathsf{lfp}\, F \preceq x$.) Let us now discharge each of the above equalities:

$$\begin{aligned}
F_{\mathbf{k}+t}(\mathbf{k} + \mathsf{lfp}\, F_t) &= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot (\mathbf{k} + t) + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C'](\mathbf{k} + \mathsf{lfp}\, F_t) && (\text{def. } F_{\mathbf{k}+t})\\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot (\mathbf{k} + t) + [\![\xi : \mathsf{true}]\!] \cdot (\mathbf{k} + \mathsf{ert}[C'](\mathsf{lfp}\, F_t)) && (\text{I.H. on } C')\\
&= \mathbf{1} + \mathbf{k} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C'](\mathsf{lfp}\, F_t)\\
&= \mathbf{k} + F_t(\mathsf{lfp}\, F_t) && (\text{def. } F_t)\\
&= \mathbf{k} + \mathsf{lfp}\, F_t. && (\text{def. } \mathsf{lfp}\,)
\end{aligned}$$

$$\begin{aligned}
F_t(\mathsf{lfp}\, F_{\mathbf{k}+t} - \mathbf{k}) &= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C'](\mathsf{lfp}\, F_{\mathbf{k}+t} - \mathbf{k}) && (\text{def. } F_t)\\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot (\mathsf{ert}[C'](\mathsf{lfp}\, F_{\mathbf{k}+t} - \mathbf{k}) + 2\mathbf{k} - 2\mathbf{k})\\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \Big(\mathsf{ert}[C'](\mathsf{lfp}\, F_{\mathbf{k}+t} - \mathbf{k} + 2\mathbf{k}) - 2\mathbf{k}\Big)\\
& && (\text{I.H. on } C')\\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \Big(\mathsf{ert}[C'](\mathsf{lfp}\, F_{\mathbf{k}+t} + \mathbf{k}) - 2\mathbf{k}\Big)\\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \Big(\mathsf{ert}[C'](\mathsf{lfp}\, F_{\mathbf{k}+t}) + \mathbf{k} - 2\mathbf{k}\Big) && (\text{I.H. on } C')\\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \Big(\mathsf{ert}[C'](\mathsf{lfp}\, F_{\mathbf{k}+t}) - \mathbf{k}\Big)\\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot (\mathbf{k} + t) + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C'](\mathsf{lfp}\, F_{\mathbf{k}+t}) - \mathbf{k}\\
&= F_{\mathbf{k}+t}(\mathsf{lfp}\, F_{\mathbf{k}+t}) - \mathbf{k} && (\text{def. } F_{\mathbf{k}+t})\\
&= \mathsf{lfp}\, F_{\mathbf{k}+t} - \mathbf{k}. && (\text{def. } \mathsf{lfp}\,) \quad \square
\end{aligned}$$

B.3 Proof of Theorem 5.5 (Soundness of the ert–Transformer)

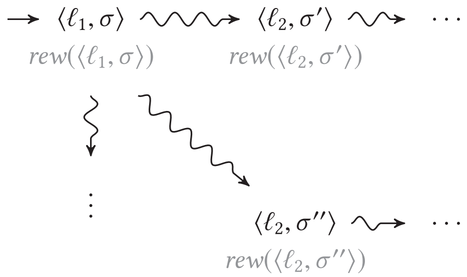
Before we prove the soundness of ert with respect to the simple operational model of our probabilistic programming language introduced in Section 5, some preparation is needed to deal with sequential composition and `while`-loops. In particular, we use the following decomposition lemma.

Lemma B.1. *Let $C_1, C_2 \in \mathsf{pGCL}$, $t \in \mathbb{T}$, and $\sigma \in \Sigma$. Then*

$$\mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![C_1; C_2]\!]}(\langle\, \mathit{sink}\, \rangle) \;=\; \mathsf{ExpRew}^{\mathcal{M}_\sigma^u[\![C_1]\!]}(\langle\, \mathit{sink}\, \rangle)\,,$$

where $u \triangleq \mathsf{ExpRew}^{\lambda\rho\bullet\, \mathcal{M}_\rho^t[\![C_2]\!]}(\langle\, \mathit{sink}\, \rangle)$.

Proof. The MC $\mathcal{M}_\sigma^t[\![C_1; C_2]\!]$ is of the following form


$\langle \ell_1, \sigma\rangle$
$\langle \ell_2, \sigma'\rangle$
$\cdots$
$rew(\langle \ell_1, \sigma\rangle)$
$rew(\langle \ell_2, \sigma'\rangle)$
$\vdots$
$\langle \ell_2, \sigma''\rangle$
$\cdots$
$rew(\langle \ell_2, \sigma''\rangle)$

Here, $\mathsf{stmt}(\ell_1) = C_1$ and $\mathsf{stmt}(\ell_2) = C_2$. Hence, every path starting in $\langle \ell_1, \sigma\rangle$ either eventually reaches $\langle \ell_2, \sigma'\rangle$, for some $\sigma' \in \Sigma$, or diverges, i.e., never reaches $\langle\, \mathit{sink}\, \rangle$. Since $\langle \ell_2, \sigma'\rangle$ is the initial state of the MC $\mathcal{M}_{\sigma'}^t[\![C_2]\!]$, we can transform $\mathcal{M}_\sigma^t[\![C_1; C_2]\!]$ into an MC $\mathcal{M}_\sigma^u[\![C_1]\!]$ with the same expected reward by setting

$$u \triangleq \mathsf{ExpRew}^{\lambda\rho\bullet\, \mathcal{M}_\rho^t[\![C_2]\!]}(\langle\, \mathit{sink}\, \rangle). \qquad \square$$

Furthermore, we have to consider bounded `while`-loops that are obtained by successively unrolling a `while`-loop up to a finite number of executions of the loop body.

Definition B.2 (Bounded `while`*-loops).* Let $\xi \in \mathsf{DExp}$ and $C \in \mathsf{pGCL}$. Then the *bounded* `while` *loops* of `while` are given by

$$\texttt{while}^{<0}\,(\xi)\,\{C\} \;\triangleq \texttt{halt}, \text{and}$$
$$\texttt{while}^{<k+1}\,(\xi)\,\{C\} \;\triangleq \texttt{if}\,(\xi)\,\{C; \texttt{while}^{<k}\,(\xi)\,\{C\}\}\,\texttt{else}\,\{\texttt{empty}\}.$$

As for ordinary programs, the runtime of a while loop can be expressed in terms of the runtime of bounded while loops.

Lemma B.3. *Let $\xi \in \mathsf{DExp}$, $C \in \mathsf{pGCL}$, and $t \in \mathbb{T}$. Then,*

$$\sup_{k \in \mathbb{N}} \mathsf{ert}[\texttt{while}^{<k}\,(\xi)\,\{C\}](t) \;=\; \mathsf{ert}[\texttt{while}\,(\xi)\,\{C\}](t).$$

Proof. Let $F_t(X)$ be the characteristic functional corresponding to `while` $(\xi)\,\{C\}$ as defined in Definition 3.1. Assume, for the moment, that for each $k \in \mathbb{N}$, we have $\mathsf{ert}[\texttt{while}^{<k}\,(\xi)\,\{C\}](t) = F_t^k(\mathbf{0})$. Then, using Kleene's Fixed Point Theorem, we can establish that

$$\sup_{k \in \mathbb{N}} \mathsf{ert}\Big[\texttt{while}^{<k}\,(\xi)\,\{C\}\Big](t) \;=\; \sup_{k \in \mathbb{N}} F_t^k(0) \;=\; \mathsf{lfp}\, X. F_t(X) \;=\; \mathsf{ert}[\texttt{while}\,(\xi)\,\{C\}](t).$$

Hence, it suffices to show that $\mathsf{ert}[\texttt{while}^{<k}\,(\xi)\,\{C\}](t) = F_t^k(\mathbf{0})$ for each $k \in \mathbb{N}$. This can be established by a straightforward induction on k. $\square$

Intuitively, this means that the expected runtime of a loop and the runtime of its finite approximations by bounded loops coincide in the limit. This also holds for the expected reward of the MC of a loop and the expected reward of its finite approximations.

Lemma B.4. *Let $\xi \in \mathsf{DExp}$, $C \in \mathsf{pGCL}$, $t \in \mathbb{T}$, and $\sigma \in \Sigma$. Then*

$$\sup_{k \in \mathbb{N}} \mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![\mathtt{while}^{<k}\ (\xi)\ \{C\}]\!]}(\langle\, sink \,\rangle) \;=\; \mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]}(\langle\, sink \,\rangle)\,.$$

Proof. First, observe that every path in the operational MC $\mathcal{M}_\sigma^t [\![\mathtt{while}^{<k}\ (\xi)\ \{C\}]\!]$ either terminates or halts after k loop iterations. Since the operational MC $\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]$ does not prematurely stop after k loop iterations, it includes the above paths. Thus, we obtain the inequality

$$\sup_{k \in \mathbb{N}} \mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![\mathtt{while}^{<k}\ (\xi)\ \{C\}]\!]}(\langle\, sink \,\rangle) \;\leq\; \mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]}(\langle\, sink \,\rangle).$$

To prove the converse inequality, observe that every infinite path in the MC $\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]$ yields either reward 0 or reward ∞. We distinguish two cases:

(1) All paths in the operational MC $\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]$ that collect positive reward are finite. Then, for each of these paths, there exists some natural number $k \geq 0$ such that the operational MC $\mathcal{M}_\sigma^t [\![\mathtt{while}^{<k}\ (\xi)\ \{C\}]\!]$ includes this path. By taking the supremum of these numbers k, we include every path of $\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]$ with positive reward. Hence,

$$\sup_{k \in \mathbb{N}} \mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![\mathtt{while}^{<k}\ (\xi)\ \{C\}]\!]}(\langle\, sink \,\rangle) \;\geq\; \mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]}(\langle\, sink \,\rangle).$$

(2) There exists an infinite path in the operational MC $\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]$ that yields positive reward. By the above observation, this path yields reward ∞. Then, for all natural numbers k, the operational MC $\mathcal{M}_\sigma^t [\![\mathtt{while}^{<k}\ (\xi)\ \{C\}]\!]$ contains a prefix of this path that yields positive reward proportional to k. By taking the supremum of these numbers k, we end up with an infinite reward. Hence,

$$\sup_{k \in \mathbb{N}} \mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![\mathtt{while}^{<k}\ (\xi)\ \{C\}]\!]}(\langle\, sink \,\rangle) \;=\; \infty \;=\; \mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![\mathtt{while}\ (\xi)\ \{C\}]\!]}(\langle\, sink \,\rangle)\,. \qquad \square$$

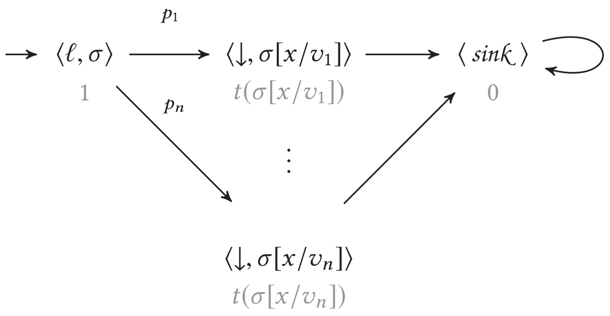
We are now in a position to show the soundness of ert with respect to the operational semantics.

Theorem 5.5 (Soundness of the ert Transformer). *Let $C \in \mathsf{pGCL}$, and $t \in \mathbb{T}$. Then, for each $\sigma \in \Sigma$, we have*

$$\mathsf{ExpRew}^{\mathcal{M}_\sigma^t [\![C]\!]}(\langle\, sink \,\rangle) \;=\; \mathsf{ert}[C](t)(\sigma).$$

Proof. We prove the claim by induction on the structure of pGCL programs $C \in \mathsf{pGCL}$. The base cases $C = \mathtt{empty}$, $C = \mathtt{skip}$, and $C = \mathtt{halt}$ are straightforward.

The Probabilistic Assignment $C = x :\approx \mu$. For some $n \in \mathbb{N}$, the MC $\mathcal{M}_\sigma^t [\![x :\approx \mu]\!]$ is of the following form—where $\mathsf{stmt}(\ell) = x :\approx \mu$:


$\langle \ell, \sigma \rangle$
1
p_1
p_n
$\langle \downarrow, \sigma[x/v_1] \rangle$
$t(\sigma[x/v_1])$
$\langle\, sink \,\rangle$
0
$\langle \downarrow, \sigma[x/v_n] \rangle$
$t(\sigma[x/v_n])$

Hence, all of its prefix-free paths reaching $\langle\, \mathit{sink}\, \rangle$ are of the form

$$\pi_i = \langle \ell, \sigma \rangle \langle \downarrow, \sigma[x/v_i] \rangle \langle\, \mathit{sink}\, \rangle \ldots,$$

with $\Pr\{\pi_i\} = p_i$ for each $v_i \in \mathsf{Val}$ with $\llbracket \mu \rrbracket(\sigma)(v_i) = p_i > 0$ and

$$\sum_{k=1}^{n} p_k = 1.$$

Moreover, $\mathit{rew}(\pi_i) = \mathbf{1} + t(\sigma[x/v_i])$. Thus, we have

$$\begin{aligned}
& \mathsf{ExpRew}^{\mathcal{M}_\sigma^t \llbracket x :\approx \mu \rrbracket}(\langle\, \mathit{sink}\, \rangle) \\
&= \sum_{1 \leq i \leq n} \Pr\{\pi_i\} \cdot \mathit{rew}(\pi_i) \\
&= \sum_{1 \leq i \leq n} \Pr\{\pi_i\} \cdot (1 + t(\sigma[x/v_i])) \\
&= 1 + \sum_{1 \leq i \leq n} \Pr\{\pi_i\} \cdot t(\sigma[x/v_i]) \\
&= 1 + \sum_{1 \leq i \leq n} \llbracket \mu \rrbracket(\sigma)(v_i) \cdot t(\sigma[x/v_i]) \\
&= 1 + \mathbb{E}_{\llbracket \mu \rrbracket(\sigma)}\, (\lambda v_i\text{.}\, t(\sigma[x/v_i])) \\
&= \mathsf{ert}[x :\approx \mu](t)(\sigma).
\end{aligned}$$

Induction Hypothesis. For all (substatements) $C' \in \mathsf{pGCL}$ of C, $t \in \mathbb{T}$ and $\sigma \in \Sigma$,

$$\mathsf{ExpRew}^{\mathcal{M}_\sigma^t \llbracket C' \rrbracket}(\langle\, \mathit{sink}\, \rangle) \;=\; \mathsf{ert}[C'](t)(\sigma).$$

For the induction step, we consider sequential composition and `while`-loops. The case of conditional statements is straightforward.

Sequential Composition. $C = C_1; C_2$.

$$\begin{aligned}
& \mathsf{ExpRew}^{\mathcal{M}_\sigma^t \llbracket C_1;\, C_2 \rrbracket}(\langle\, \mathit{sink}\, \rangle) \\
&= \mathsf{ExpRew}^{\mathcal{M}_\sigma^{\mathsf{ExpRew}^{\lambda\rho\cdot\mathcal{M}_\rho^t \llbracket C_2 \rrbracket}(\langle\, \mathit{sink}\, \rangle)} \llbracket C_1 \rrbracket}(\langle\, \mathit{sink}\, \rangle) && \text{(Lemma B.1)} \\
&= \mathsf{ExpRew}^{\mathcal{M}_\sigma^{\lambda\rho\bullet\, \mathsf{ert}[C_2](t)(\rho)} \llbracket C_1 \rrbracket \langle\, \mathit{sink}\, \rangle}(\langle\, \mathit{sink}\, \rangle) && \text{(I.H. on } C_2\text{)} \\
&= \mathsf{ert}[C_1](\mathsf{ert}[C_2](t))(\sigma) && \text{(I.H. on } C_1\text{)} \\
&= \mathsf{ert}[C_1; C_2](t)(\sigma).
\end{aligned}$$

While-Loop. Let $C = \texttt{while}$. For any natural number $k \geq 1$ and $\sigma \in \Sigma$, we have

$$\begin{aligned}
& \mathsf{ert}[\texttt{while}^{<k}\ (\xi)\ \{C'\}](t)(\sigma) \\
&= \mathsf{ert}[\texttt{if}\ (\xi)\ \{C'; \texttt{while}^{<k-1}\ (\xi)\ \{C'\}\}\ \texttt{else}\ \{\texttt{empty}\}](t)(\sigma) \\
&= 1 + \llbracket \xi : \mathsf{true} \rrbracket(\sigma) \cdot \mathsf{ert}[C'; \texttt{while}^{<k-1}\ (\xi)\ \{C'\}](t)(\sigma) \\
&\qquad + \llbracket \xi : \mathsf{false} \rrbracket(\sigma) \cdot \mathsf{ert}[\texttt{empty}](t)(\sigma) \\
&= 1 + \llbracket \xi : \mathsf{true} \rrbracket(\sigma) \cdot \mathsf{ExpRew}^{\mathcal{M}_\sigma^t \llbracket C'; \texttt{while}^{<k-1}\ (\xi)\ \{C'\} \rrbracket}(\langle\, \mathit{sink}\, \rangle) && \text{(I.H.)} \\
&\qquad + \llbracket \xi : \mathsf{false} \rrbracket(\sigma) \cdot \mathsf{ExpRew}^{\mathcal{M}_\sigma^t \llbracket \texttt{empty} \rrbracket}(\langle\, \mathit{sink}\, \rangle) \\
&= \mathsf{ExpRew}^{\mathcal{M}_\sigma^t \llbracket \texttt{while}^{<k}\ (\xi)\ \{C'\} \rrbracket}(\langle\, \mathit{sink}\, \rangle).
\end{aligned}$$

Moreover, for $k = 0$, we have

$$\begin{aligned}
&\mathsf{ert}[\mathtt{while}^{<0}\ (\xi)\ \{C'\}](t)(\sigma) \\
&= \mathsf{ert}[\mathtt{halt}](t)(\sigma) && \text{Definition B.2} \\
&= \mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![\mathtt{halt}]\!]}(\langle\, \mathit{sink}\,\rangle) && \text{(already shown in base case)} \\
&= \mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![\mathtt{while}^{<0}\ (\xi)\ \{C'\}]\!]}(\langle\, \mathit{sink}\,\rangle). && \text{(Definition B.2)}
\end{aligned}$$

Putting both cases together, we can establish that

$$\begin{aligned}
&\mathsf{ert}[\mathtt{while}\ (\xi)\ \{C'\}](t)(\sigma) \\
&= \sup_{k\in\mathbb{N}} \mathsf{ert}[\mathtt{while}^{<k}\ (\xi)\ \{C'\}](t)(\sigma) && \text{(Lemma B.3)} \\
&= \sup_{k\in\mathbb{N}} \mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![\mathtt{while}^{<k}\ (\xi)\ \{C'\}]\!]}(\langle\, \mathit{sink}\,\rangle) \\
&= \mathsf{ExpRew}^{\mathcal{M}_\sigma^t[\![\mathtt{while}\ (\xi)\ \{C'\}]\!]}(\langle\, \mathit{sink}\,\rangle). && \text{(Lemma B.4)} \quad \square
\end{aligned}$$

B.4 Proof of Theorem 6.1 (Soundness of ert w.r.t. Nielson's Proof System)

The proof relies on three auxiliary results that are presented first. The first lemma shows a standard relationship between while–loops and their unrollings.

Lemma B.5. *For all* pGCL *programs C, $t \in \mathbb{T}$ and* deterministic *guards B, we have*

$$\mathsf{ert}[\mathtt{while}\ (B)\ \{C\}](t) \;=\; \mathsf{ert}[\mathtt{if}\ (B)\ \{C; \mathtt{while}\ (B)\ \{C\}\}\ \mathtt{else}\ \{\mathtt{empty}\}](t).$$

Proof. Let $F_t(X)$ be the characteristic functional corresponding to while (B) $\{C\}$ as introduced in Definition 3.1. Then,

$$\begin{aligned}
&\mathsf{ert}[\mathtt{while}\ (B)\ \{C\}](t) \\
&= \mathsf{lfp}\, F_t \\
&= F_t(\mathsf{lfp}\, F_t) && \text{(Def. lfp)} \\
&= 1 + [\![B : \mathsf{true}]\!] \cdot \mathsf{ert}[C](\mathsf{lfp}\, F_t) + [\![B : \mathsf{false}]\!] \cdot t && \text{(Definition 3.1)} \\
&= 1 + [\![B : \mathsf{true}]\!] \cdot \mathsf{ert}[C](\mathsf{ert}[\mathtt{while}\ (B)\ \{C\}](t)) + [\![B : \mathsf{false}]\!] \cdot t \\
&= 1 + [\![B : \mathsf{true}]\!] \cdot \mathsf{ert}[C](\mathsf{ert}[\mathtt{while}\ (B)\ \{C\}](t)) + [\![B : \mathsf{false}]\!] \cdot \mathsf{ert}[\mathtt{empty}](t) \\
&= 1 + [\![B : \mathsf{true}]\!] \cdot \mathsf{ert}[C; \mathtt{while}\ (B)\ \{C\}](t) + [\![B : \mathsf{false}]\!] \cdot \mathsf{ert}[\mathtt{empty}](t) \\
&= \mathsf{ert}[\mathtt{if}\ (B)\ \{C; \mathtt{while}\ (B)\ \{C\}\}\ \mathtt{else}\ \{\mathtt{empty}\}](t). && \square
\end{aligned}$$

Moreover, we observe that the runtime of two sequentially composed *terminating* deterministic programs C_1, C_2 can be decomposed into the sum of the individual runtimes of C_1 and C_2. To that end, we need the following. The MC $\mathcal{M}_\sigma^{\mathbf{0}}[\![C]\!]$ of a deterministic program C and a program state $\sigma \in \Sigma$ (cf. Definition 5.3) reduces to a labeled transition system. In particular, if a state $\langle \downarrow, \sigma\rangle$ indicating successful termination is reachable from the initial state of $\mathcal{M}_\sigma^{\mathbf{0}}[\![C]\!]$, it is unique. Hence, we capture the effect of a deterministic program by a partial function $\mathbb{C}[\![\,\cdot\,]\!](\,\cdot\,) : \mathsf{GCL} \times \Sigma \rightharpoonup \Sigma$ mapping each deterministic program $C \in \mathsf{GCL}$ and each program state $\sigma \in \Sigma$ to a program state $\sigma' \in \Sigma$ if and only if there exists a state $\langle \downarrow, \sigma'\rangle$ that is reachable in the MC $\mathcal{M}_\sigma^{\mathbf{0}}[\![C]\!]$ from its initial state $\langle \mathsf{init}(C), \sigma\rangle$. Otherwise, $\mathbb{C}[\![C]\!](\sigma)$ is undefined.

Lemma B.6. *Let $C_1 \in \mathsf{GCL}$ terminate on program state $\sigma \in \Sigma$ and $C_2 \in \mathsf{GCL}$ terminate on $\mathbb{C}[\![C_1]\!](\sigma)$. Then,*

$$\mathsf{ert}[C_1; C_2](\mathbf{0})(\sigma) = \mathsf{ert}[C_1](\mathbf{0})(\sigma) + \mathsf{ert}[C_2](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)).$$

Proof. Immediate by inspection of the MC $\mathcal{M}_\sigma^{\mathbf{0}}[\![C_1; C_2]\!]$ and Theorem 5.5. □

Our third lemma extends the previous decomposition result to `while`-loops.

Lemma B.7. *Let B be a deterministic guard and $C \in \mathsf{GCL}$. For all program states $\sigma \in \Sigma$ with $[\![B]\!](\sigma) = \mathsf{true}$ such that* `while` (B) $\{C\}$ *terminates on σ, we have*

$$\mathsf{ert}[\mathtt{while}\ (B)\ \{C\}](\mathbf{0})(\sigma) \;=\; 1 + \mathsf{ert}[C](\mathbf{0})(\sigma) + \mathsf{ert}[\mathtt{while}\ (B)\ \{C\}](\mathbf{0})(\mathbb{C}[\![C]\!](\sigma)).$$

Proof.

$$\begin{aligned}
&\mathsf{ert}[\mathtt{while}\ (B)\ \{C\}](\mathbf{0})(\sigma) \\
&= \mathsf{ert}[\mathtt{if}\ (B)\ \{C; \mathtt{while}\ (B)\ \{C\}\}\ \mathtt{else}\ \{\mathtt{empty}\}](\mathbf{0})(\sigma) && \text{(Lemma B.5)} \\
&= 1 + [\![B]\!](\sigma)\cdot \mathsf{ert}[C; \mathtt{while}\ (B)\ \{C\}](\mathbf{0})(\sigma) + [\![\neg B]\!](\sigma)\cdot 0 \\
&= 1 + \mathsf{ert}[C; \mathtt{while}\ (B)\ \{C\}](\mathbf{0})(\sigma) && ([\![\mathrm{B}]\!](\sigma) = \mathsf{true}) \\
&= 1 + \mathsf{ert}[C](\mathbf{0})(\sigma) + \mathsf{ert}[\mathtt{while}](\mathbf{0})(\mathbb{C}[\![C]\!](\sigma). && \text{(Lemma B.6)}
\end{aligned}$$
□

We are now in a position to prove the soundness of ert with respect to Nielson's proof system.

Theorem 6.1 (Soundness of ert for deterministic programs). *For all deterministic programs $C \in \mathsf{GCL}$ and assertions P, Q, we have*

$$\vdash \{\,P\,\}\, C\, \{\Downarrow\ Q\,\} \quad \text{implies} \quad \vdash_E \{\,P\,\}\ C\ \{\,\mathsf{ert}[C](\mathbf{0}) \Downarrow Q\,\}.$$

Proof. By induction on the structure of GCL-program C. The base cases $C = \mathtt{skip}$ and $C = x := E$ are immediate, because

$$\mathsf{ert}[\mathtt{skip}](\mathbf{0}) = \mathsf{ert}[x := E](\mathbf{0}) = 1$$

and $\{\,P\,\}$ `skip` $\{\,1 \Downarrow P\,\}$ as well as $\{\,P\,\}\ x := E\ \{\,1 \Downarrow P\,\}$ are axioms.

Induction Hypothesis. Assume that for each substatement C' of C and each pair of assertions P, Q, we have

$$\vdash \{\,P\,\}\, C'\, \{\Downarrow\ Q\,\} \text{ implies } \vdash_E \{\,P\,\}\ C'\ \{\,\mathsf{ert}[C'](\mathbf{0}) \Downarrow Q\,\}.$$

For the induction step, we have to consider sequential composition, conditionals, and loops.

Sequential Composition $C' = C_1; C_2$. Assume that

$$\vdash \{\,P\,\}\, C_1; C_2\, \{\Downarrow\ Q\,\}.$$

Then, there exists an assertion R such that

$$\vdash \{\,P\,\}\, C_1\, \{\Downarrow\ R\,\} \text{ and } \vdash \{\,R\,\}\, C_2\, \{\Downarrow\ Q\,\}.$$

By induction hypothesis, we know that

$$\vdash_E \{\,P\,\}\ C_1\ \{\,\mathsf{ert}[C_1](\mathbf{0}) \Downarrow R\,\} \text{ and } \vdash_E \{\,R\,\}\ C_2\ \{\,\mathsf{ert}[C_2](\mathbf{0}) \Downarrow Q\,\}.$$

Now, let

$$E_2' \triangleq \mathsf{ert}[C_1; C_2](\mathbf{0}) - \mathsf{ert}[C_1](\mathbf{0})$$

and consider the triple

$$\{\,P \wedge E_2' = u\,\}\ C_1\ \{\,\mathsf{ert}[C_1](\mathbf{0}) \Downarrow R \wedge \mathsf{ert}[C_2](\mathbf{0}) \leq u\,\},$$

where u is a fresh logical variable. Since u does not occur in P, each state $\sigma \in \Sigma$ satisfying P also satisfies $P \wedge E_2' = u$; the latter is denoted by $\sigma \models P \wedge E_2' = u$. Then,

$$\begin{aligned}
&\sigma \models P \wedge E_2' = u \text{ and } \mathbb{C}[\![C_1]\!](\sigma) \models R \text{ and } \mathsf{ert}[C_2](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) \leq u \\
\Leftrightarrow\ &\sigma \models P \wedge E_2' = u \text{ and } \mathbb{C}[\![C_1]\!](\sigma) \models R \text{ and} \\
&\mathsf{ert}[C_2](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) \leq \mathsf{ert}[C_1; C_2](\mathbf{0})(\sigma) - \mathsf{ert}[C_1](\mathbf{0})(\sigma) && \text{(Definition of } E_2'\text{)} \\
\Leftrightarrow\ &\sigma \models P \wedge E_2' = u \text{ and } \mathbb{C}[\![C_1]\!](\sigma) \models R \text{ and} \\
&\mathsf{ert}[C_1](\mathbf{0})(\sigma) + \mathsf{ert}[C_2](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) \leq \mathsf{ert}[C_1; C_2](\mathbf{0})(\sigma) && \text{(Lemma B.6)} \\
\Rightarrow\ &\models_E \{\, P \wedge E_2' = u \,\}\ C_1\ \{\, \mathsf{ert}[C_1](\mathbf{0}) \Downarrow R \wedge \mathsf{ert}[C_2](\mathbf{0}) \leq u \,\}.
\end{aligned}$$

Since both

$$\{\, P \wedge E_2' = u \,\}\ C_1\ \{\, \mathsf{ert}[C_1](\mathbf{0}) \Downarrow R \wedge \mathsf{ert}[C_2](\mathbf{0}) \leq u \,\}$$

and

$$\{\, R \,\}\ C_2\ \{\, \mathsf{ert}[C_2](\mathbf{0}) \Downarrow Q \,\}$$

are valid triples, we may apply the rule of sequential composition to conclude

$$\begin{aligned}
&\vdash_E \{\, P \,\}\ C_1; C_2\ \{\, \mathsf{ert}[C_1](\mathbf{0}) + E_2' \Downarrow Q \,\} \\
\Leftrightarrow\ &\vdash_E \{\, P \,\}\ C_1; C_2\ \{\, \mathsf{ert}[C_1; C_2](\mathbf{0}) \Downarrow Q \,\}. && \text{(Definition of } E_2'\text{)}
\end{aligned}$$

Conditionals. $C' = \texttt{if}\ (B)\ \{C_1\}\ \texttt{else}\ \{C_2\}$. Assume that

$$\vdash \{\, P \,\}\, \texttt{if}\ (B)\ \{C_1\}\ \texttt{else}\ \{C_2\}\, \{\, \Downarrow\ Q \,\}$$

is a provable triple in Hoare logic. Then, also,

$$\vdash \{\, P \wedge B \,\}\, C_1\, \{\, \Downarrow\ Q \,\} \text{ and } \vdash \{\, P \wedge \neg B \,\}\, C_2\, \{\, \Downarrow\ Q \,\}.$$

By induction hypothesis, it follows that

$$\vdash_E \{\, P \wedge B \,\}\ C_1\ \{\, \mathsf{ert}[C_1](\mathbf{0}) \Downarrow Q \,\} \text{ and } \vdash_E \{\, P \wedge \neg B \,\}\ C_2\ \{\, \mathsf{ert}[C_2](\mathbf{0}) \Downarrow Q \,\}$$

are provable triples in Nielson's logic. Now, let

$$\begin{aligned}
E &\triangleq \mathsf{ert}[\texttt{if}\ (B)\ \{C_1\}\ \texttt{else}\ \{C_2\}](\mathbf{0}) \\
&= 1 + [\![B]\!] \cdot \mathsf{ert}[C_1](\mathbf{0}) + [\![\neg B]\!] \cdot \mathsf{ert}[C_2](\mathbf{0}).
\end{aligned}$$

Then $E \geq \mathsf{ert}[C_1](\mathbf{0})$ and $E \geq \mathsf{ert}[C_2](\mathbf{0})$ hold for all states satisfying the precondition P. Hence, we can apply the rule of consequence to conclude

$$\vdash_E \{\, P \wedge B \,\}\ C_1\ \{\, E \Downarrow Q \,\} \text{ and } \vdash_E \{\, P \wedge \neg B \,\}\ C_2\ \{\, E \Downarrow Q \,\}.$$

Now, applying the rule for conditionals yields

$$\vdash_E \{\, P \,\}\ \texttt{if}\ (B)\ \{C_1\}\ \texttt{else}\ \{C_2\}\ \{\, E \Downarrow Q \,\}.$$

Loops. $C' = \texttt{while}\ (B)\ \{C_1\}$. Assume that

$$\vdash \{\, P \,\}\, \texttt{while}\ (B)\ \{C_1\}\, \{\, \Downarrow\ Q \,\}$$

is a provable triple. Then, there exists an assertion $R(z)$ such that $P \Rightarrow \exists z_\bullet R(z)$, $R(0) \Rightarrow Q$ and

$$\vdash \{\, \exists z_\bullet R(z) \,\}\, \texttt{while}\ (B)\ \{C_1\}\, \{\, \Downarrow\ R(0) \,\}.$$

By the `while`-rule of Hoare logic for total correctness, we have

$$\vdash \{\, R(z{+}1) \,\}\, C_1\, \{\, \Downarrow\ R(z) \,\}. \qquad (*)$$

The induction hypothesis yields

$$\vdash_E \{\, R(z{+}1) \,\}\ C_1\ \{\, E_1 \Downarrow R(z) \,\},$$

where $E_1 = \mathsf{ert}[C_1](\mathbf{0})$. Now, let

$$E' \triangleq \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0}) - \mathsf{ert}[C_1](\mathbf{0})$$

and

$$E \triangleq \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0}).$$

Our goal is to apply the while-rule in order to show

$$\vdash_E \{\, \exists z\boldsymbol{.}\, R(z) \,\}\ \mathtt{while}\ (B)\ \{C_1\}\ \{\, E \Downarrow R(0) \,\}.$$

We first check the side conditions of this rule for our choice of E'; i.e., we show $R(0) \Rightarrow \neg B \wedge E \geq 1$ as well as $R(z{+}1) \Rightarrow B \wedge E \geq E_1 + E'$.

If $R(0)$ is valid, then $\neg B$ is valid due to $(*)$ and

$$[\![E]\!](\sigma) = \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0})(\sigma) = 1 \geq 1.$$

Furthermore, if $R(z{+}1)$ is valid for some $z \in \mathbb{N}$, then B is valid by $(*)$ and for each program state $\sigma \in \Sigma$, we have

$$\begin{aligned}
[\![E]\!](\sigma) &= \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0})(\sigma) \\
&= \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0})(\sigma) - \mathsf{ert}[C_1](\mathbf{0})(\sigma) + \mathsf{ert}[C_1](\mathbf{0})(\sigma) \\
&= [\![E']\!](\sigma) + [\![E_1]\!](\sigma).
\end{aligned}$$

Hence, the side conditions of the while-rule hold. In order to apply the rule, we also have to show validity of the triple

$$\{\, R(z{+}1) + E' = u \,\}\ C_1\ \{\, \mathsf{ert}[C_1](\mathbf{0}) \Downarrow R(z) \wedge E \leq u \,\},$$

where u is a fresh logical variable. Since u does not occur in P, we know that for each state $\sigma \in \Sigma$ with $\sigma \models P$, we also have $\sigma \models R(z{+}1) \wedge E' = u$. Then

$$\begin{aligned}
&\sigma \models R(z{+}1) \wedge E' = u \text{ and } \mathbb{C}[\![C_1]\!](\sigma) \models R(z) \text{ and } E(\mathbb{C}[\![C_1]\!](\sigma)) \leq u \\
\Leftrightarrow\ &\sigma \models R(z{+}1) \wedge E' = u \text{ and } \mathbb{C}[\![C_1]\!](\sigma) \models R(z) \text{ and } E(\mathbb{C}[\![C_1]\!](\sigma)) \leq E'(\sigma) && \text{(Def. of } u\text{)} \\
\Leftrightarrow\ &\sigma \models R(z{+}1) \wedge E' = u \text{ and } \mathbb{C}[\![C_1]\!](\sigma) \models R(z) \text{ and} \\
&E(\mathbb{C}[\![C_1]\!](\sigma)) \leq \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0})(\sigma) - \mathsf{ert}[C_1](\mathbf{0})(\sigma) && \text{(Definition of } E'\text{)} \\
\Leftrightarrow\ &\sigma \models R(z{+}1) \wedge E' = u \text{ and } \mathbb{C}[\![C_1]\!](\sigma) \models R(z) \text{ and} \\
&E(\mathbb{C}[\![C_1]\!](\sigma)) + \mathsf{ert}[C_1](\mathbf{0})(\sigma) \leq \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0})(\sigma) \\
\Rightarrow\ &\models_E \{\, R(z{+}1) \wedge E' = u \,\}\ C_1\ \{\, \mathsf{ert}[C_1](\mathbf{0}) \Downarrow R(z) \wedge E \leq u \,\}. && \text{(Lemma B.7)}
\end{aligned}$$

Thus, we may apply the while-rule to conclude

$$\vdash_E \{\, \exists z\boldsymbol{.}\, R(z) \,\}\ \mathtt{while}\ (B)\ \{C_1\}\ \{\, \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0}) \Downarrow R(0) \,\}.$$

By assumption, the implications $P \Rightarrow \exists z\boldsymbol{.}\, R(z)$ as well as $R(0) \Rightarrow Q$ are valid, i.e.,

$$\vdash_E \{\, P \,\}\ \mathtt{while}\ (B)\ \{C_1\}\ \{\, \mathsf{ert}[\mathtt{while}\ (B)\ \{C_1\}](\mathbf{0}) \Downarrow Q \,\}.$$ □

B.5 Proof of Theorem 6.2 (Completeness of ert w.r.t. Nielson's Proof System)

We show the following claim by induction on the structure of program statements: For all assertions P, Q, deterministic program $C \in \mathsf{GCL}$, and arithmetic expressions E,

$$\vdash_E \{\, P \,\}\ C\ \{\, E \Downarrow Q \,\}$$

implies that there exists a natural number $k \in \mathbb{N}$ such that for all program states $\sigma \in \Sigma$, we have

$$\mathsf{ert}[C](\mathbf{0})(\sigma) \leq k \cdot [\![E]\!](\sigma).$$

The Effectless Program $C = \texttt{skip}$. Assume

$$\vdash_E \{\, P \,\} \ \texttt{skip} \ \{\, E \Downarrow Q \,\}.$$

Then there exists an assertion R such that

$$\vdash_E \{\, R \,\} \ \texttt{skip} \ \{\, 1 \Downarrow R \,\}$$

and $P \Rightarrow R \wedge 1 \leq k \cdot E$ and $R \Rightarrow Q$ are valid for some $k \in \mathbb{N}$. Hence, there exists a $k \in \mathbb{N}$ such that $\mathsf{ert}[\texttt{skip}](\mathbf{0})(\sigma) = 1 \leq k \cdot [\![E]\!](\sigma)$ for each $\sigma \in \Sigma$.

The Assignment $C = x := E$. Analogous to skip.

The Sequential Composition $C = C_1; C_2$. Assume

$$\vdash_E \{\, P \,\} \ C_1; C_2 \ \{\, E \Downarrow Q \,\}.$$

Then there exists an assertion R and arithmetic expressions E_1, E_2, E_2' such that

$$\vdash_E \{\, P \wedge E_2' = u \,\} \ C_1 \ \{\, E_1 \Downarrow R \wedge E_2 \leq u \,\} \quad \text{and} \quad \vdash_E \{\, R \,\} \ C_2 \ \{\, E_2 \Downarrow Q \,\}$$

for some fresh logical variable u. By the induction hypothesis, there exist natural numbers $k_1, k_2 \in \mathbb{N}$ such that for all program states $\sigma \in \Sigma$, we have

$$\mathsf{ert}[C_1](\mathbf{0})(\sigma) \leq k_1 \cdot [\![E_1]\!](\sigma) \text{ and } \mathsf{ert}[C_2](\mathbf{0})(\sigma) \leq k_2 \cdot [\![E_2]\!](\sigma).$$

By setting $k = \max\{k_1, k_2\}$, we obtain

$$\mathsf{ert}[C_1](\mathbf{0})(\sigma) \leq k \cdot [\![E_1]\!](\sigma) \text{ and } \mathsf{ert}[C_2](\mathbf{0})(\sigma) \leq k \cdot [\![E_2]\!](\sigma). \tag{$*$}$$

In particular, this means that

$$\mathsf{ert}[C_2](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) \leq k \cdot [\![E_2]\!](\mathbb{C}[\![C_1]\!](\sigma)) \leq k \cdot [\![E_2']\!](\sigma). \tag{$\dagger$}$$

Hence,

$$\begin{aligned}
&\mathsf{ert}[C_1; C_2](\mathbf{0})(\sigma) \\
&\quad = \ \mathsf{ert}[C_1](\mathbf{0})(\sigma) + \mathsf{ert}[C_2](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) && \text{(Lemma B.6)} \\
&\quad \leq \ k \cdot [\![E_1]\!](\sigma) + \mathsf{ert}[C_2](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) && \text{(by $*$)} \\
&\quad \leq \ k \cdot [\![E_1]\!](\sigma) + k \cdot [\![E_2']\!](\sigma) && \text{(by $\dagger$)} \\
&\quad = \ k \cdot ([\![E_1]\!](\sigma) + [\![E_2']\!](\sigma)).
\end{aligned}$$

Conditionals $C' = \texttt{if}\ (B)\ \{C_1\}\ \texttt{else}\ \{C_2\}$. Assume

$$\vdash_E \{\, P \,\} \ \texttt{if}\ (B)\ \{C_1\}\ \texttt{else}\ \{C_2\} \ \{\, E \Downarrow Q \,\}.$$

Then

$$\vdash_E \{\, P \wedge B \,\} \ C_1 \ \{\, E \Downarrow Q \,\} \text{and } \vdash_E \{\, P \wedge \neg B \,\} \ C_2 \ \{\, E \Downarrow Q \,\}.$$

By induction hypothesis, there exist natural numbers $k_1, k_2 \in \mathbb{N}$ such that for each $\sigma \in \Sigma$ and $i \in \{1, 2\}$,

$$\mathsf{ert}[C_i](\mathbf{0})(\sigma) \leq k_i \cdot [\![E]\!](\sigma).$$

By setting $k = \max\{k_1, k_2\}$, we obtain

$$\mathsf{ert}[C_i](\mathbf{0})(\sigma) \leq k \cdot [\![E]\!](\sigma). \tag{$**$}$$

Thus,

$$\begin{aligned}
&\mathsf{ert}[\texttt{if } (B)\ \{C_1\} \texttt{ else } \{C_2\}](0)(\sigma) \\
&\quad = 1 + [\![B]\!](\sigma) \cdot \mathsf{ert}[C_1](\mathbf{0})(\sigma) + [\![\neg B]\!](\sigma) \cdot \mathsf{ert}[C_2](\mathbf{0})(\sigma) && \text{(Table 1)} \\
&\quad \leq k + [\![B]\!](\sigma) \cdot k \cdot [\![E]\!](\sigma) + [\![\neg B]\!](\sigma) \cdot k \cdot [\![E]\!](\sigma) && \text{(by $*\,*$)} \\
&\quad \leq (3 \cdot k) \cdot [\![E]\!](\sigma).
\end{aligned}$$

Loops. $C' = \texttt{while } (B)\ \{C_1\}$. Assume

$$\vdash_E \{\, P \,\}\ \texttt{while } (B)\ \{C_1\}\ \{\, E \Downarrow Q \,\}.$$

Then there exists an assertion $R(z)$ such that $P \Rightarrow \exists z\boldsymbol{.}\, R(z)$ and $R(0) \Rightarrow Q$ are valid. Furthermore, there exists $z \in \mathbb{N}$ such that

$$\vdash_E \{\, R(z{+}1) \wedge E' = u \,\}\ C_1\ \{\, E_1 \Downarrow R(z) \wedge E \leq u \,\} \qquad (\ddagger)$$

for some fresh logical variable u. Additionally, for each $z \in \mathbb{N}$, the side conditions

$$R(z{+}1) \Rightarrow B \wedge E \geq E_1 + E' \text{ as well as } R(0) \Rightarrow \neg B \wedge E \geq 1 \qquad (\dagger\dagger)$$

are valid. Our proof obligation is to show for some $k \in \mathbb{N}$ and all program states $\sigma \in \Sigma$ satisfying P that

$$\mathsf{ert}[\texttt{while } (B)\ \{C_1\}](\mathbf{0})(\sigma) \leq k \cdot [\![E]\!](\sigma). \qquad (\spadesuit)$$

By induction hypothesis, there exists a $k' \in \mathbb{N}$ such that for each $\sigma \in \Sigma$, we have

$$1 \leq \mathsf{ert}[C_1](\mathbf{0})(\sigma) \leq k' \cdot [\![E_1]\!](\sigma). \qquad (\clubsuit)$$

We show by complete induction over $\mathrm{z} \in \mathbb{N}$ that for all $\sigma \in \Sigma$ with $\sigma \models \mathrm{R(z)}$ and

$$\vdash_E \{\, R(z) \,\}\ \texttt{while } (B)\ \{C_1\}\ \{\, E \Downarrow R(0) \,\},$$

we have $\mathsf{ert}[\texttt{while } (B)\ \{C_1\}](\mathbf{0})(\sigma) \leq (k'{+}1) \cdot [\![E]\!](\sigma)$.

For the base case $z = 0$, the side condition $R(0) \Rightarrow \neg B \wedge E \geq 1$ yields

$$\begin{aligned}
&(k'{+}1) \cdot [\![E]\!](\sigma) \\
&\quad \geq 1 && \text{(by $\dagger\dagger$)} \\
&\quad = 1 + [\![B]\!](\sigma) \cdot \mathsf{ert}[C_1](\mathbf{0})(\sigma) + [\![\neg B]\!](\sigma) \cdot 0 \\
&\quad = \mathsf{ert}[\texttt{while } (B)\ \{C_1\}](\mathbf{0})(\sigma). && \text{(Table 1)}
\end{aligned}$$

Now, for the induction step, assume $\sigma \models R(z{+}1)$. Then the side condition $R(z{+}1) \Rightarrow B \wedge E \geq E_1 + E'$ yields

$$\begin{aligned}
&(k'{+}1) \cdot [\![E]\!](\sigma) \\
&\quad \geq (k'{+}1) \cdot ([\![E_1]\!](\sigma) + [\![E']\!](\sigma)) && \text{(by $\dagger\dagger$)} \\
&\quad \geq (k'{+}1) \cdot [\![E_1]\!](\sigma) + (k'{+}1) \cdot [\![E]\!](\mathbb{C}[\![C_1]\!](\sigma)) && \text{(Postcondition of $\ddagger$)} \\
&\quad \geq (k'{+}1) \cdot [\![E_1]\!](\sigma) + \mathsf{ert}[\texttt{while } (B)\ \{C_1\}](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) && \text{(I.H., $\mathbb{C}[\![C_1]\!](\sigma) \models R(z)$)} \\
&\quad \geq [\![E_1]\!](\sigma) + \mathsf{ert}[C_1](\mathbf{0})(\sigma) + \mathsf{ert}[\texttt{while } (B)\ \{C_1\}](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) && \text{(by $\clubsuit$)} \\
&\quad \geq 1 + \mathsf{ert}[C_1](\mathbf{0})(\sigma) + \mathsf{ert}[\texttt{while } (B)\ \{C_1\}](\mathbf{0})(\mathbb{C}[\![C_1]\!](\sigma)) \\
&\quad = \mathsf{ert}[\texttt{while } (B)\ \{C_1\}](\mathbf{0})(\sigma), && \text{(Lemma B.7)}
\end{aligned}$$

which completes the inner induction. Moreover, $(\spadesuit)$ follows immediately by setting $k = k'{+}1$. □

B.6 Well–Definedness of ert on Procedure Calls

We prove that the least fixed point

$$\mathsf{lfp}\, \eta \bullet\, \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\eta} \tag{2}$$

defining $\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}]$ is well defined. To this end we take the following four steps:

(1) We endow the set RtEnv of runtime environments with the structure of an ω–cpo (Proposition B.8).
(2) We show that transformer $\mathsf{ert}[C]^{\sharp}_{\eta}(t)$ is continuous with respect to t for every $C \in \mathsf{pRGCL}$ (Lemma B.9).
(3) We show that the transformer $\mathsf{ert}[C]^{\sharp}_{\eta}(t)$ is also continuous with respect to η (Lemma B.10).
(4) We conclude that $\lambda\eta \bullet\, \underline{\mathbf{1}} \oplus \mathsf{ert}[C]^{\sharp}_{\eta}$ is continuous and by the Kleene Fixed-Point Theorem (Theorem A.3) the least fixed point in Equation (2) is well defined.

Let "$\sqsubseteq$" denote the pointwise order between runtime environments; i.e., for $\eta_1, \eta_2 \in \mathsf{RtEnv}$, $\eta_1 \sqsubseteq \eta_2$ if and only if $\eta_1(t) \preceq \eta_2(t)$ for every $t \in \mathbb{T}$.

Proposition B.8. *The pair* $(\mathsf{RtEnv}, \sqsubseteq)$ *is an ω-cpo with bottom element* $\bot_{\mathsf{RtEnv}} \triangleq \lambda t : \mathbb{T} \bullet\, \mathbf{0}$, *where the supremum of an ω-chain $\eta_0 \sqsubseteq \eta_1 \sqsubseteq \cdots$ is given pointwise, i.e.,* $(\sup_n \eta_n)(t) \triangleq \sup_n \eta_n(t)$.

Lemma B.9 (Continuity of $\mathsf{ert}[C]^{\sharp}_{\eta}(t)$ w.r.t. t). *Let $t_0 \preceq t_1 \preceq \cdots$ be an ω-chain in $\mathbb{T}$. Then for every $C \in \mathsf{pRGCL}$ and $\eta \in \mathsf{RtEnv}$,*

$$\mathsf{ert}[C]^{\sharp}_{\eta}(\sup_n t_n) \;=\; \sup_n \mathsf{ert}[C]^{\sharp}_{\eta}(t_n).$$

Proof. By induction on the structure of C. Except for procedure calls, all program constructs use the same proof argument as for the continuity of plain transformer $\mathsf{ert}[\,\cdot\,]$ (see the proof of Lemma 3.2). For procedure calls, the statement follows immediately from the continuity of η since

$$\mathsf{ert}[\mathtt{call}\, P]^{\sharp}_{\eta}(\sup_n t_n) \;=\; \eta(\sup_n t_n) \;=\; \sup_n\, \eta(t_n) \;=\; \sup_n \mathsf{ert}[\mathtt{call}\, P]^{\sharp}_{\eta}(t_n). \qquad \square$$

Lemma B.10 (Continuity of $\mathsf{ert}[C]^{\sharp}_{\eta}(t)$ w.r.t. η). *Let $\eta_0 \sqsubseteq \eta_1 \sqsubseteq \cdots$ be an ω-chain in* RtEnv. *Then, for every $C \in \mathsf{pRGCL}$,*

$$\mathsf{ert}[C]^{\sharp}_{\sup_n \eta_n} \;=\; \sup_n \mathsf{ert}[C]^{\sharp}_{\eta_n}.$$

Proof. By induction on the structure of C. For the four basic instructions $\mathtt{empty}$, $\mathtt{skip}$, $\mathtt{halt}$, and $x :\approx \mu$ the proof is straightforward since the action of the transformer is independent of the runtime environment (i.e., constant functions are always continuous). We omit the case of while-loops since they can be readily simulated by recursive procedures. For the remaining program constructs we reason as follows:

Sequential Composition. We use the fact that $\mathsf{ert}[C_1]^{\sharp}_{\sup_m \eta_m}(f)$ is continuous in f (by Lemma B.9) and Lemma A.5 to "merge" a pair of nested supremums into a single supremum:

$$\begin{aligned}
\mathsf{ert}[C_1;C_2]^{\sharp}_{\sup_n \eta_n}(t) \;&=\; \mathsf{ert}[C_1]^{\sharp}_{\sup_m \eta_m}\Big(\mathsf{ert}[C_2]^{\sharp}_{\sup_n \eta_n}(t)\Big) && \text{(Table 3)}\\
&=\; \mathsf{ert}[C_1]^{\sharp}_{\sup_m \eta_m}\Big(\sup_n \mathsf{ert}[C_2]^{\sharp}_{\eta_n}(t)\Big) && \text{(I.H. on } C_2\text{)}\\
&=\; \sup_n\; \mathsf{ert}[C_1]^{\sharp}_{\sup_m \eta_m}\Big(\mathsf{ert}[C_2]^{\sharp}_{\eta_n}(t)\Big) && \text{(Lemma B.9)}\\
&=\; \sup_n\; \sup_m\; \mathsf{ert}[C_1]^{\sharp}_{\eta_m}\Big(\mathsf{ert}[C_2]^{\sharp}_{\eta_n}(t)\Big) && \text{(I.H. on } C_1\text{)}\\
&=\; \sup_i\; \mathsf{ert}[C_1]^{\sharp}_{\eta_i}\Big(\mathsf{ert}[C_2]^{\sharp}_{\eta_i}(t)\Big) && \text{(Lemma A.5)}\\
&=\; \sup_i\; \mathsf{ert}[C_1;C_2]^{\sharp}_{\eta_i}(t). && \text{(Table 3)}
\end{aligned}$$

Conditional Choice. The idea here is to substitute $\sup_n \mathsf{ert}[C_1]^\sharp_{\eta_n}$ with $\lim_{n\to\infty} \mathsf{ert}[C_1]^\sharp_{\eta_n}$ using the Monotone Sequence Theorem. This is possible because by I.H., $\mathsf{ert}[C_1]^\sharp_{\eta_n}$ is (continuous and thus) monotonic in η_n and $\eta_0 \sqsubseteq \eta_1 \sqsubseteq \cdots$; therefore, $(\mathsf{ert}[C_1]^\sharp_{\eta_n}(t))_{n\in\mathbb{N}}$ defines a monotonic sequence:

$$\begin{aligned}
&\mathsf{ert}[\texttt{if }(\xi)\ \{C_1\}\texttt{ else }\{C_2\}]^\sharp_{\sup_n \eta_n}(t) \\
&= \llbracket \xi : \mathsf{true} \rrbracket \cdot \mathsf{ert}[C_1]^\sharp_{\sup_n \eta_n}(t) + \llbracket \xi : \mathsf{false} \rrbracket \cdot \mathsf{ert}[C_2]^\sharp_{\sup_n \eta_n}(t) && \text{(Table 3)} \\
&= \llbracket \xi : \mathsf{true} \rrbracket \cdot \sup_n \mathsf{ert}[C_1]^\sharp_{\eta_n}(t) + \llbracket \xi : \mathsf{false} \rrbracket \cdot \sup_n \mathsf{ert}[C_2]^\sharp_{\eta_n}(t) && \text{(I.H. on } C_1,\ C_2\text{)} \\
&= \llbracket \xi : \mathsf{true} \rrbracket \cdot \lim_{n\to\infty} \mathsf{ert}[C_1]^\sharp_{\eta_n}(t) + \llbracket \xi : \mathsf{false} \rrbracket \cdot \lim_{n\to\infty} \mathsf{ert}[C_2]^\sharp_{\eta_n}(t) && \text{(Theorem A.2)} \\
&= \lim_{n\to\infty} \Big(\llbracket \xi : \mathsf{true} \rrbracket \cdot \mathsf{ert}[C_1]^\sharp_{\eta_n}(t) + \llbracket \xi : \mathsf{false} \rrbracket \cdot \mathsf{ert}[C_2]^\sharp_{\eta_n}(t) \Big) && \text{(algebra of limits)} \\
&= \sup_n \Big(\llbracket \xi : \mathsf{true} \rrbracket \cdot \mathsf{ert}[C_1]^\sharp_{\eta_n}(t) + \llbracket \xi : \mathsf{false} \rrbracket \cdot \mathsf{ert}[C_2]^\sharp_{\eta_n}(t) \Big) && \text{(Theorem A.2)} \\
&= \sup_n \mathsf{ert}[\texttt{if }(\xi)\ \{C_1\}\texttt{ else }\{C_2\}]^\sharp_{\eta_n}. && \text{(Table 3)}
\end{aligned}$$

Procedure Call. The reasoning here is straightforward:

$$\begin{aligned}
\mathsf{ert}[\texttt{call }P]^\sharp_{\sup_n \eta_n}(t) &= (\sup_n \eta_n)(t) && \text{(Table 3)} \\
&= \sup_n \eta_n(t) && \text{(def. } \sup_n \eta_n\text{)} \\
&= \sup_n\ \mathsf{ert}[\texttt{call }P]^\sharp_{\eta_n}(t). && \text{(Table 3)}
\end{aligned}$$

□

B.7 Proof of Theorem 7.2 (Continuity of ert for Recursive Programs)

We prove that for pRGCL program $\langle C, \mathcal{D}\rangle$ and ω-chain of runtimes $t_0 \preceq t_1 \preceq \cdots$,

$$\sup_n \mathsf{ert}[C, \mathcal{D}](f_n) = \mathsf{ert}[C, \mathcal{D}](\sup_n f_n).$$

The proof proceeds by induction on the structure of C. We consider only the case of procedure calls since all other language constructs follow the same reasoning as in the proof of Lemma 3.2 (see Appendix B.1), the only difference being that the transformer now propagates declarations. For $\mathsf{ert}[\texttt{call }P, \mathcal{D}]$, let $F(\eta) = \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^\sharp_\eta$. Since F is continuous (see Appendix B.6), we can apply the Kleene Fixed-Point Theorem to characterize $\mathsf{lfp}\ \eta\bullet F(\eta)$ as $\sup_m F^m(\bot_{\mathsf{RtEnv}})$, and we have

$$\begin{aligned}
\mathsf{ert}[\texttt{call }P, \mathcal{D}](\sup_n t_n) &= \Big(\mathsf{lfp}\ \eta\bullet F(\eta)\Big)(\sup_n t_n) && \text{(Equation (1))} \\
&= \sup_m F^m(\bot_{\mathsf{RtEnv}})(\sup_n t_n) && \text{(Theorem A.3)} \\
&= \sup_m \sup_n F^m(\bot_{\mathsf{RtEnv}})(t_n) && (\mathrm{F^m}(\bot_{\mathsf{RtEnv}})\text{ continuous}) \\
&= \sup_n \sup_m F^m(\bot_{\mathsf{RtEnv}})(t_n) && \text{(Theorem A.5)} \\
&= \sup_n (\mathsf{lfp}\ \eta \cdot F(\eta))(t_n) && \text{(Theorem A.3)} \\
&= \sup_n\ \mathsf{ert}[\texttt{call }P, \mathcal{D}](t_n). && \text{(Equation (1))}
\end{aligned}$$

We are only left to prove that $F^m(\bot_{\mathsf{RtEnv}})$ is continuous for all $m \in \mathbb{N}$. We prove this by induction on m. The base case is immediate since $F^0(\bot_{\mathsf{RtEnv}}) = \bot_{\mathsf{RtEnv}}$ and $\bot_{\mathsf{RtEnv}}$ is continuous. For the inductive case we have $F^{m+1}(\bot_{\mathsf{RtEnv}}) = F(F^m(\bot_{\mathsf{RtEnv}}))$. The continuity of $F^{m+1}(\bot_{\mathsf{RtEnv}})$ follows from the I.H. and the fact that F preserves continuity; i.e., η continuous implies $F(\eta)$ continuous. □

B.8 Proof of Theorem 7.2 (Constant Propagation of ert for Recursive Programs)

We prove that for halt-free pRGCL program $\langle C, \mathcal{D}\rangle$ and constant runtime $\mathbf{k}$ with $k \in \mathbb{R}_{\geq 0}$,

$$\mathsf{ert}[C, \mathcal{D}](\mathbf{k} + t) \ = \ \mathbf{k} + \mathsf{ert}[C, \mathcal{D}](t). \tag{3}$$

The proof relies on two subsidiary results that we present below.

Lemma B.11. *For* halt*-free* pRGCL *program* $\langle C, \mathcal{D}\rangle$ *and constant runtime* $\mathbf{k}$ *with* $k \in \mathbb{R}_{\geq 0}$,

$$\mathsf{ert}\left[C, \mathcal{D}\right]\left(\mathbf{k}\right) \succeq \mathbf{k}.$$

Proof. By induction on the structure of C. Except for the case of procedure calls, all other program constructs pose no difficulty. For a procedure call, we must do a case distinction on whether the procedure terminates almost surely or not. This requires extending the weakest precondition transformer wp to probabilistic recursive programs. A detailed proof containing this case analysis is provided in [43, App. 6]. □

For stating the second auxiliary result, we require the notion of "constant separable" runtime environment. We say that $\eta \in \mathsf{RtEnv}$ is *constant separable into* $\upsilon \in \mathsf{RtEnv}$ if and only if for all $k \in \mathbb{R}_{\geq 0}$ and $t \in \mathbb{T}$, $\eta(\mathbf{k} + t) = \mathbf{k} + \upsilon(t)$.

Lemma B.12. *Let* η *be a runtime environment constant separable*[10] *into* υ. *Then, for all* halt*-free* $C \in$ pRGCL *and constant runtime* $\mathbf{k}$ *with* $k \in \mathbb{R}_{\geq 0}$,

$$\mathsf{ert}[C]^{\sharp}_{\eta}(\mathbf{k} + t) \ = \ \mathbf{k} + \mathsf{ert}[C]^{\sharp}_{\upsilon}(t).$$

Proof. By a routine induction on the structure of C. □

Now we have all prerequisites to establish Equation (3). By letting $F(\eta) = \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\eta}$, we can recast the equation as $(\mathsf{lfp}\, F)(\mathbf{k} + t) = \mathbf{k} + (\mathsf{lfp}\, F)(t)$, or equivalently,

$$\Big(\lambda \eta^{\star}\text{.}\, \lambda t^{\star}\text{.}\, \eta^{\star}(\mathbf{k} + t^{\star})\Big)(\mathsf{lfp}\, F) \ = \ \Big(\lambda \eta^{\star}\text{.}\, \lambda t^{\star}\text{.}\, \mathbf{k} + \eta^{\star}(t^{\star})\Big)(\mathsf{lfp}\, F).$$

To prove this equation, we apply the Lemma A.6 with instantiations

$$\begin{aligned}
F_1 &= F_2 \ = \ F \\
f_1 &= \lambda \eta^{\star}\text{.}\, \lambda t^{\star}\text{.}\, \eta^{\star}(\mathbf{k} + t^{\star}) \\
f_2 &= \lambda \eta^{\star}\text{.}\, \lambda t^{\star}\text{.}\, \mathbf{k} + \eta^{\star}(t^{\star}) \\
h_1 &= \lambda \eta^{\star}\text{.}\, \lambda t^{\star}\text{.}\, \mathbf{1} + \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\lambda t' \cdot \eta^{\star}(t' - \mathbf{k})}(\mathbf{k} + t^{\star}) \\
h_2 &= \lambda \eta^{\star}\text{.}\, \lambda t^{\star}\text{.}\, \mathbf{k} + \mathbf{1} + \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\lambda t' \cdot \eta^{\star}(t') - \mathbf{k}}(t^{\star})
\end{aligned}$$

and underlying ω-cpos $(\mathcal{D}_1, \leq_1) = (\mathcal{D}_2, \leq_2) = (\mathcal{D}, \leq) = (\mathsf{RtEnv}, \sqsubseteq)$ and bottom elements $\bot_1 = \bot_2 = \bot = \bot_{\mathsf{RtEnv}}$. The application of Lemma A.6 requires the continuity of F, which follows from Lemma B.10; the continuity of f_1 and $f2$, which holds because runtime environments are continuous by definition; and finally the monotonicity of h_1 and h_2. This latter fact, together with the fact that h_1 and h_2 are effectively well defined (i.e., have type $\mathsf{RtEnv} \to \mathsf{RtEnv}$), can be proved with an inductive argument (on the structure of $\mathcal{D}(P)$).

We are left to discharge hypotheses 1 through 3 of Lemma A.6. A simple unfolding of the involved functions yields $f_1(F(\eta)) \sqsubseteq h_1(f_1(\eta))$ and $f_2(F(\eta)) \sqsubseteq h_2(f_2(\eta))$ for all $\eta \in \mathsf{RtEnv}$; this establishes hypothesis 1. As for hypothesis 2, $f_1(\bot_{\mathsf{RtEnv}}) \sqsubseteq f_2(\mathsf{lfp}\, F)$ holds because $f_1(\bot_{\mathsf{RtEnv}}) = \bot_{\mathsf{RtEnv}}$

[10] For the definition of *constant separable* runtime environment, see the paragraph above Lemma B.12.

and $f_2(\bot_{\mathsf{RtEnv}}) \sqsubseteq f_1(\mathsf{lfp}\, F)$ reduces to $\mathbf{k} \preceq \mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](\mathbf{k} + t)$, which holds in view of the monotonicity of ert and the auxiliary Lemma B.11. Finally, to discharge hypothesis 3, we let $\eta(t') = \mathbf{k} + \mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](t' {-} \mathbf{k})$ and reason as follows:

$$
\begin{aligned}
& h_1(f_2(\mathsf{lfp}\, F))(t) \;\preceq\; f_2(\mathsf{lfp}\, F)(t) \\
\iff\; & \mathbf{1} + \mathsf{ert}[\mathcal{D}(P)]^{\#}_{\eta}(\mathbf{k} + t) \;\preceq\; \mathbf{k} + \mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](t) && \text{(definition } h_1, f_2,\ F\text{)} \\
\iff\; & \mathbf{1} + \mathbf{k} + \mathsf{ert}[\mathcal{D}(P)]^{\#}_{\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}]}(t) && (\eta \text{ constant separable into)} \\
& \quad \preceq\; \mathbf{k} + \mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](t) && \text{(ert[call } P, \mathcal{D}]\text{; Lemma B.12)} \\
\iff\; & \mathbf{k} + F(\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}])(t) \;\preceq\; \mathbf{k} + \mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](t) && \text{(definition F)} \\
\iff\; & \mathbf{k} + F(\mathsf{lfp}\, F)(t) \;\preceq\; \mathbf{k} + (\mathsf{lfp}\, F)(t) && \text{(Equation (1))} \\
\iff\; & \mathbf{k} + (\mathsf{lfp}\, F)(t) \;\preceq\; \mathbf{k} + (\mathsf{lfp}\, F)(t) && \text{(definition } \mathit{lfp}\text{)} \\
\Longleftarrow\; & \mathsf{true} && (\text{“} \preceq \text{” } \textit{is a partial order})
\end{aligned}
$$

To prove the other part of hypothesis 3, i.e., $h_2(f_1(\mathsf{lfp}\, F))(t) \preceq f_1(\mathsf{lfp}\, F)(t)$, we follow a similar reasoning. □

B.9 Proof of Theorem 7.3 (Proof Rules for Expected Runtimes of Recursive Programs)

We show that proof rules (1) and (2) from Theorem 7.3 reproduced below are sound with respect to the ert-calculus in pRGCL. Proof rule (3) follows the same argument as (2):

$$
\frac{\mathsf{ert}[\mathtt{call}\, P](t) \preceq \mathbf{1} + u \;\Vdash\; \mathsf{ert}[\mathcal{D}(P)](t) \preceq u}{\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](t) \preceq \mathbf{1} + u}\ (1)
$$

$$
\frac{u_0 = \mathbf{0} \qquad \mathsf{ert}[\mathtt{call}\, P](t) \preceq \mathbf{1} + \mathsf{ert}[\mathcal{D}(P)](t) \preceq u_{n+1}}{\mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](t) \preceq \mathbf{1} + \lim_{n \to \infty} u_n}\,.\ (2)
$$

To prove the rules sound, we make use of the following result:

Proposition B.13. *The derivability assertion*

$$\mathsf{ert}[\mathtt{call}\, P](t_1) \preceq u_1 \;\Vdash\; \mathsf{ert}[C](t_2) \preceq u_2$$

implies that for every runtime environment η,

$$\eta(t_1) \preceq u_1 \Rightarrow \mathsf{ert}[C]^{\#}_{\eta}(t_2) \preceq u_2 .$$

Soundness of Rule (1). Let runtime environment $\eta^\star$ map t to u and all other runtimes to (the constant runtime) ∞. Then,

$$
\begin{aligned}
& \mathsf{ert}[\mathtt{call}\, P, \mathcal{D}](t) \preceq \mathbf{1} + u \\
\iff\; & \Big(\mathsf{lfp}\, \eta \centerdot \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\#}_{\eta}\Big)(t) \;\preceq\; \mathbf{1} + u && \text{(Equation (1))} \\
\iff\; & \mathsf{lfp}\, \eta \cdot \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\#}_{\eta} \;\sqsubseteq\; \underline{\mathbf{1}} \oplus \eta^\star && (\text{definition } \eta^\star, \sqsubseteq) \\
\Longleftarrow\; & \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\#}_{\underline{\mathbf{1}} \oplus \eta^\star} \;\sqsubseteq\; \underline{\mathbf{1}} \oplus \eta^\star && \text{(Theorem A.4, B.10)} \\
\iff\; & \mathbf{1} + \mathsf{ert}[\mathcal{D}(P)]^{\#}_{\underline{\mathbf{1}} \oplus \eta^\star}(t) \;\preceq\; \mathbf{1} + u && (\text{definition } \eta^\star, \sqsubseteq) \\
\iff\; & \mathsf{ert}[\mathcal{D}(P)]^{\#}_{\underline{\mathbf{1}} \oplus \eta^\star}(t) \;\preceq\; u && \text{(algebra)} \\
\Longleftarrow\; & (\underline{\mathbf{1}} \oplus \eta^\star)(t) \preceq \mathbf{1} + u && \text{(Proposition B.13, rule premise)} \\
\iff\; & \mathsf{true}. && (\text{definition } \eta^\star)
\end{aligned}
$$

Soundness of Rule (2). Let $F(\eta) = \underline{\mathbf{1}} \oplus \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{\eta}$. Since F is continuous (see Appendix B.6), we can apply the Kleene Fixed-Point Theorem (Theorem A.3) to characterize $\mathsf{lfp}\,\eta\bullet F(\eta)$ as $\sup_n F^n(\bot_{\mathsf{RtEnv}})$. This yields

$$\mathsf{ert}[\mathtt{call}\,P,\mathcal{D}](t) \;=\; \mathsf{lfp}\,\eta\bullet F(\eta) \;=\; \sup_n F^n(\bot_{\mathsf{RtEnv}})(t).$$

Moreover, the sequence $(F^n(\bot_{\mathsf{RtEnv}})(t))_{n\in\mathbb{N}}$ is monotonic. Then by the Monotone Sequence Theorem (Theorem A.2),

$$\sup_n F^n(\bot_{\mathsf{RtEnv}})(t) \;=\; \lim_{n\to\infty} F^n(\bot_{\mathsf{RtEnv}})(t).$$

Combining the above two equations, we obtain

$$\begin{aligned}
&\mathsf{ert}[\mathtt{call}\,P,\mathcal{D}](t) \preceq \mathbf{1} + \lim_{n\to\infty} u_n \\
\iff\ &\lim_{n\to\infty} F^n(\bot_{\mathsf{RtEnv}})(t) \preceq \mathbf{1} + \lim_{n\to\infty} u_n \\
\impliedby\ &\forall n \cdot F^n(\bot_{\mathsf{RtEnv}})(t) \preceq \mathbf{1} + u_n.
\end{aligned}$$

We prove the last statement by induction on n. The base case $F^0(\bot_{\mathsf{RtEnv}})(t) \preceq 1 + u_0$ is immediate since $F^0(\bot_{\mathsf{RtEnv}})(t) = \bot_{\mathsf{RtEnv}}(t) = 0$. For the inductive case we have

$$\begin{aligned}
&F^{n+1}(\bot_{\mathsf{RtEnv}})(t) \preceq \mathbf{1} + u_{n+1} \\
\iff\ &\mathbf{1} + \mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{F^n(\bot_{\mathsf{RtEnv}})}(t) \preceq \mathbf{1} + u_{n+1} && \text{(definition } F^{n+1}(\bot_{\mathsf{RtEnv}})\text{)} \\
\iff\ &\mathsf{ert}[\mathcal{D}(P)]^{\sharp}_{F^n(\bot_{\mathsf{RtEnv}})}(t) \preceq u_{n+1} && \text{(algebra)} \\
\impliedby\ &F^n(\bot_{\mathsf{RtEnv}})(t) \preceq \mathbf{1} + u_n && \text{(Proposition B.13, rule premise)} \\
\iff\ &\mathsf{true}. && \text{(I.H.)} \quad \square
\end{aligned}$$

B.10 Proof of Theorem 7.10 (Soundness of ert w.r.t. Pushdown Markov Chains)

Let us fix procedures $P_1,\ldots,P_k$ and a declaration $\mathcal{D}$. As in the proof of Theorem 5.5, we prove the soundness of ert with respect to operational PMCs by induction on the structure of pRGCL-programs. Similar to bounded `while`-loops (cf. Definition B.2), this proof relies on expressing the runtime of procedure calls in terms of inlined call-free programs. Thus, recall from Section 7.4 the definition of bounded procedure calls $\mathtt{call}^{\mathcal{D}}_n\,P$ (for multiple procedures):

Definition B.14 (Bounded procedure calls). The *bounded procedure calls* of $\mathtt{call}\,P_i$, $1 \le i \le k$, are given by

$$\begin{aligned}
\mathtt{call}^{\mathcal{D}}_0\,P_i &\triangleq \mathtt{halt} \\
\mathtt{call}^{\mathcal{D}}_{n+1}\,P_i &\triangleq \mathtt{skip};\,\mathcal{D}(P_i)\left[\mathtt{call}\,P_1/\mathtt{call}^{\mathcal{D}}_n\,P_1,\ldots,\mathtt{call}\,P_k/\mathtt{call}^{\mathcal{D}}_n\,P_k\right].
\end{aligned}$$

The main motivation to consider bounded procedure calls is to use them as runtime environments. This usage is formally expressed by the following auxiliary result.

Lemma B.15. *For each $C \in$ pRGCL, $1 \le i \le k$, and $t \in \mathbb{T}$, we have*

$$\mathsf{ert}[C,\mathcal{D}](t) \;=\; \sup_{n\in\mathbb{N}} \mathsf{ert}\,[C,\mathcal{D}]^{\sharp}_{(\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_n\,P_1],\ldots,\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_n\,P_k])}(t).$$

Proof. By induction on the structure of pRGCL programs. In all cases except for procedure calls, the claims follows immediately from the definition of $\mathsf{ert}[.](.)$ and $\mathsf{ert}[.]^{\sharp}_{\eta}(.)$ and the induction hypothesis for compound statements. For a procedure call $\mathtt{call}\, P_i$, we have

$$\begin{aligned}
&\mathsf{ert}[\mathtt{call}\, P_i, \mathcal{D}](t) \\
&= \sup_{n\in\mathbb{N}} \mathsf{ert}\left[\mathtt{call}^{\mathcal{D}}_n\, P_i\right](t) && \text{(Theorem 7.5)} \\
&= \sup_{n\in\mathbb{N}} \mathsf{ert}\left[\mathtt{call}\, P_i, \mathcal{D}\right]^{\sharp}_{(\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_n\, P_1],\ldots,\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_n\, P_k])}(t) && (\mathsf{ert}[C]^{\sharp}_{\mathsf{ert}[C']} = \mathsf{ert}[C[\mathtt{call}\, P/C']]).
\end{aligned}$$

□

Now, consider a PMC ${}^{n}\langle\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]\rangle$ that behaves exactly the same as the operational PMC $\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]$ but counts the number of symbols currently on the stack. Moreover, if this number is exactly n and $\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]$ would perform another push onto the stack, ${}^{n}\langle\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]\rangle$ immediately moves to $\langle\, \mathit{sink}\,\rangle$—without entering a state of the form $\langle\downarrow,\sigma'\rangle$ indicating successful termination first and without collecting reward for the procedure call. Intuitively, this means that the modified PMC *halts* its execution after encountering $n+1$ procedure calls without returns. It is evident that:

Lemma B.16. $\mathsf{ExpRew}^{\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]}(\langle\, \mathit{sink}\,\rangle) \;=\; \sup_{n\in\mathbb{N}} \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]\rangle}(\langle\, \mathit{sink}\,\rangle)\,.$

Proof sketch. ${}^{n}\langle\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]\rangle$ exhibits a partial behavior of $\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]$ in the sense that every path of ${}^{n}\langle\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]\rangle$ eventually reaching $\langle\, \mathit{sink}\,\rangle$ is—up to renaming—also a path of $\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]$. Hence,

$$\mathsf{ExpRew}^{\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]}(\langle\, \mathit{sink}\,\rangle) \;\leq\; \sup_{n\in\mathbb{N}} \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]\rangle}(\langle\, \mathit{sink}\,\rangle).$$

Conversely, every finite path π of $\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]$ eventually reaching $\langle\, \mathit{sink}\,\rangle$ can be implemented with finite stack size. Therefore, there exists an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ the path π of $\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]$ is also a path of ${}^{n}\langle\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]\rangle$. Thus,

$$\mathsf{ExpRew}^{\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]}(\langle\, \mathit{sink}\,\rangle) \;\geq\; \sup_{n\in\mathbb{N}} \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^f_\sigma[\![C,\mathcal{D}]\!]\rangle}(\langle\, \mathit{sink}\,\rangle).$$

□

Assume for the moment the following result:

Lemma B.17. *For all* $n \in \mathbb{N}$ *it holds that*

$$\lambda\sigma\text{.}\, \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^t_\sigma[\![C,\mathcal{D}]\!]\rangle}(\langle\, \mathit{sink}\,\rangle) \;=\; \mathsf{ert}\left[C,\mathcal{D}\right]^{\sharp}_{(\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_n\, P_1],\ldots,\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_n\, P_k])}(t).$$

With these auxiliary results readily available, we are in a position to show the transformer ert for pRGCL-programs to be sound with respect to operational PMCs.

Theorem 7.10 (Correspondence Theorem). *Let* $\langle C,\mathcal{D}\rangle$ *be a* pRGCL *program,* $t \in \mathbb{T}$. *Then for each* $\sigma \in \Sigma$, *we have*

$$\mathsf{ExpRew}^{\mathcal{P}^t_\sigma[\![C,\mathcal{D}]\!]}(\langle\, \mathit{sink}\,\rangle) \;=\; \mathsf{ert}[C,\mathcal{D}](t)(\sigma).$$

Proof.

$$\begin{aligned}
&\mathsf{ExpRew}^{\mathcal{P}^t_\sigma[\![C,\mathcal{D}]\!]}(\langle\, \mathit{sink}\,\rangle) \\
&= \sup_{n\in\mathbb{N}} \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^t_\sigma[\![C,\mathcal{D}]\!]\rangle}(\langle\, \mathit{sink}\,\rangle) && \text{(Lemma B.16)} \\
&= \sup_{n\in\mathbb{N}} \mathsf{ert}\left[C,\mathcal{D}\right]^{\sharp}_{(\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_n\, P_1],\ldots,\mathsf{ert}[\mathtt{call}^{\mathcal{D}}_n\, P_k])}(t) && \text{(Lemma B.17)} \\
&= \mathsf{ert}[C,\mathcal{D}](t). && \text{(Lemma B.15.)}
\end{aligned}$$

□

It remains to prove Lemma B.17.

Proof of Lemma B.17. By induction on n.

The Base Case $n = 0$. By Definition B.14, we have $\texttt{call}^{\mathcal{D}}_{0}\, P_i = \texttt{halt}$ for each procedure P_i. We proceed by induction on the structure of pRGCL programs to show that

$$\lambda\sigma\text{.}\ \mathsf{ExpRew}^{{}^{0}\langle\mathcal{P}^{t}_{\sigma}\llbracket C,\mathcal{D}\rrbracket\rangle}(\langle\, \mathit{sink}\, \rangle) \;=\; \mathsf{ert}\,[C,\mathcal{D}]^{\sharp}_{(\texttt{halt},\ldots,\texttt{halt})}\,(t).$$

We first consider the base case of procedure calls $C = \texttt{call}\, P_i$. Since $n = 0$, no single push onto the stack may be performed without the PMC ${}^{n}\langle\mathcal{P}^{t}_{\sigma}\llbracket C,\mathcal{D}\rrbracket\rangle$ immediately moving to $\langle\, \mathit{sink}\, \rangle$. Thus, every path of the operational PMC ${}^{n}\langle\mathcal{P}^{t}_{\sigma}\llbracket \texttt{call}\, P_i,\mathcal{D}\rrbracket\rangle$ is of the form

$$(\langle \mathsf{init}(\texttt{call}\, P_i),\sigma\rangle,\gamma_0) \xrightarrow{1} (\langle\, \mathit{sink}\, \rangle,\gamma_0) \xrightarrow{1} \ldots \xrightarrow{1} (\langle\, \mathit{sink}\, \rangle,\gamma_0),$$

collecting zero reward. Then

$$\begin{aligned}
&\mathsf{ExpRew}^{{}^{0}\langle\mathcal{P}^{t}_{\sigma}\llbracket \texttt{call}\, P_i,\mathcal{D}\rrbracket\rangle}(\langle\, \mathit{sink}\, \rangle)\\
&= 0\\
&= \mathsf{ert}\,[\texttt{halt},\mathcal{D}]^{\sharp}_{(\texttt{halt},\ldots,\texttt{halt})}\,(t)(\sigma)\\
&= \mathsf{ert}\left[\texttt{call}^{\mathcal{D}}_{0}\, P_i,\mathcal{D}\right]^{\sharp}_{(\texttt{halt},\ldots,\texttt{halt})}\,(t)(\sigma).
\end{aligned}$$

All other base cases are analogous to the soundness proof for pGCL-programs presented in the proof of Theorem 5.5, because $\mathsf{ert}\,[C,\mathcal{D}]^{\sharp}_{(\texttt{halt},\ldots,\texttt{halt})}\,(t) = \mathsf{ert}[C](t)$ holds for each call-free program C. The same holds for the compound statements—sequential composition, conditionals, and loops—by using the inductive hypothesis on C.

Inductive Hypothesis on n. For an arbitrary but fixed $n \in \mathbb{N}$ we assume that for all pRGCL–programs C, we have

$$\lambda\sigma\text{.}\ \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^{t}_{\sigma}\llbracket C,\mathcal{D}\rrbracket\rangle}(\langle\, \mathit{sink}\, \rangle) \;=\; \mathsf{ert}\,[C,\mathcal{D}]^{\sharp}_{(\texttt{call}^{\mathcal{D}}_{n}\, P_1,\ldots,\texttt{call}^{\mathcal{D}}_{n}\, P_k)}\,(t).$$

Inductive Step $n \mapsto n{+}1$. As in the base case $n = 0$, the proof proceeds by structural induction on C, where each case except for procedure calls is analogous to the soundness proof for pGCL-programs (cf. Theorem 5.5). We thus concentrate on the treatment of procedure calls, a base case of our structural induction. By definition of the transition relation Δ of ${}^{n}\langle\mathcal{P}^{t}_{\sigma}\llbracket C,\mathcal{D}\rrbracket\rangle$, we observe that

$$\mathsf{ExpRew}^{{}^{n+1}\langle\mathcal{P}^{t}_{\sigma}\llbracket \texttt{call}\, P_i,\mathcal{D}\rrbracket\rangle}(\langle\, \mathit{sink}\, \rangle) \;=\; 1 \;+\; \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^{t}_{\sigma}\llbracket \mathcal{D}(P_i),\mathcal{D}\rrbracket\rangle}(\langle\, \mathit{sink}\, \rangle).$$

In other words, the expected reward of a procedure call equals the reward of the call itself plus the reward of executing the procedure's body. We then obtain the desired result as follows:

$$\begin{aligned}
&\lambda\sigma\text{.}\ \mathsf{ExpRew}^{{}^{n+1}\langle\mathcal{P}^{t}_{\sigma}\llbracket \texttt{call}\, P_i,\mathcal{D}\rrbracket\rangle}(\langle\, \mathit{sink}\, \rangle)\\
&= \lambda\sigma\text{.}\ 1 + \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^{t}_{\sigma}\llbracket \mathcal{D}(P_i),\mathcal{D}\rrbracket\rangle}(\langle\, \mathit{sink}\, \rangle) && \text{(previous observation)}\\
&= \mathbf{1} \;+\; \lambda\sigma\text{.}\ \mathsf{ExpRew}^{{}^{n}\langle\mathcal{P}^{t}_{\sigma}\llbracket \mathcal{D}(P_i),\mathcal{D}\rrbracket\rangle}(\langle\, \mathit{sink}\, \rangle)\\
&= \mathbf{1} \;+\; \mathsf{ert}\,[\mathcal{D}(P_i),\mathcal{D}]^{\sharp}_{(\texttt{call}^{\mathcal{D}}_{n}\, P_1,\ldots,\texttt{call}^{\mathcal{D}}_{n}\, P_k)}\,(t) && \text{(I.H. on } n\text{)}\\
&= \mathsf{ert}\,[\texttt{skip};\, \mathcal{D}(P_i),\mathcal{D}]^{\sharp}_{(\texttt{call}^{\mathcal{D}}_{n}\, P_1,\ldots,\texttt{call}^{\mathcal{D}}_{n}\, P_k)}\,(t)\\
&= \mathsf{ert}\,[\texttt{call}\, P_i,\mathcal{D}]^{\sharp}_{(\texttt{call}^{\mathcal{D}}_{n+1}\, P_1,\ldots,\texttt{call}^{\mathcal{D}}_{n+1}\, P_k)}\,(t).
\end{aligned}$$

This completes the proof of Lemma B.17 as well as Theorem 7.10. □

B.11 Proof of Theorem 8.1 (Connection between ert and wp)

We prove that for every program $C \in$ pGCL,

$$\mathsf{ert}[C](t + t') \;=\; \mathsf{ert}[C](t) + \mathsf{wp}[C](t')$$

by induction on the structure of C. We consider only the cases of compound statements and assignments as the proof argument for the remaining basic instructions is straightforward.

Assignment.

$$\begin{aligned}
&\mathsf{ert}[x :\approx \mu](t + t') \\
&\quad= \mathbf{1} + \lambda\sigma\text{.}\ \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\ (t + t')[x/v](\sigma)) && \text{(Table 1)}\\
&\quad= \mathbf{1} + \lambda\sigma\text{.}\ \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\ t[x/v](\sigma)) + \lambda\sigma\text{.}\ \mathsf{E}_{[\![\mu]\!](\sigma)}(\lambda v\text{.}\ t'[x/v](\sigma)) && (\mathsf{E}_\eta(\cdot)\text{linear})\\
&\quad= \mathsf{ert}[x :\approx \mu](t) + \mathsf{wp}[x :\approx \mu](t') && \text{(Tables 1, 5)}
\end{aligned}$$

Conditionals.

$$\begin{aligned}
&\mathsf{ert}[\texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\}](t + t') \\
&\quad= \mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C_1](t + t') + [\![\xi : \mathsf{false}]\!] \cdot \mathsf{ert}[C_2](t + t') && \text{(Table 1)}\\
&\quad= \mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \big(\mathsf{ert}[C_1](t) + \mathsf{wp}[C_1](t')\big)\\
&\qquad + [\![\xi : \mathsf{false}]\!] \cdot \big(\mathsf{ert}[C_2](t) + \mathsf{wp}[C_2](t')\big) && \text{(I.H. on } C_1,\ C_2)\\
&\quad= \mathbf{1} + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C_1](t) + [\![\xi : \mathsf{false}]\!] \cdot \mathsf{ert}[C_2](t)\\
&\qquad + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{wp}[C_1](t') + [\![\xi : \mathsf{false}]\!] \cdot \mathsf{wp}[C_2](t')\\
&\quad= \mathsf{ert}[\texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\}](t) + \mathsf{wp}[\texttt{if}\ (\xi)\ \{C_1\}\ \texttt{else}\ \{C_2\}](t') && \text{(Tables 1, 5)}
\end{aligned}$$

Sequential Composition.

$$\begin{aligned}
&\mathsf{ert}[C_1; C_2](t + t')\\
&\quad= \mathsf{ert}[C_1](\mathsf{ert}[C_2](t + t')) && \text{(Table 1)}\\
&\quad= \mathsf{ert}[C_1](\mathsf{ert}[C_2](t) + \mathsf{wp}[C_2](t')) && \text{(I.H. on } C_2)\\
&\quad= \mathsf{ert}[C_1](\mathsf{ert}[C_2](t)) + \mathsf{wp}[C_1](\mathsf{wp}[C_2](t')) && \text{(I.H. on } C_1)\\
&\quad= \mathsf{ert}[C_1; C_2](t) + \mathsf{wp}[C_1; C_2](t') && \text{(Tables 1, 5)}
\end{aligned}$$

Loops. Let $F_t(X) \triangleq \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}[C'](X)$ and $G_t(X) \triangleq [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{wp}[C'](X)$. We have to show that

$$\mathsf{lfp}\, X\text{.}\ F_{t+t'}(X) \;=\; \mathsf{lfp}\, X\text{.}\ F_t(X) \;+\; \mathsf{lfp}\, X\text{.}\ G_{t'}(X),$$

which, following the same argument for the soundness of the second rule in Theorem 7.3 (see Appendix B.9), becomes

$$\lim_{n\to\infty} F^n_{t+t'}(\mathbf{0}) \;=\; \lim_{n\to\infty} F^n_t(\mathbf{0}) \;+\; G^n_{t'}(\mathbf{0}),$$

where $F^n_{t+t'}$ denotes the composition of $F_{t+t'}$ with itself n times (and likewise for $G^n_{t'}$). To conclude, we prove by induction on n that

$$\forall n\text{.}\ F^n_{t+t'}(\mathbf{0}) = F^n_t(\mathbf{0}) \;+\; G^n_{t'}(\mathbf{0}).$$

The base case is immediate since it reduces to $\mathbf{0} = \mathbf{0} + \mathbf{0}$. For the inductive case, we derive

$$\begin{aligned}
F^{n+1}_{t+t'}(\mathbf{0}) &= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot (t + t') + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}\left[C'\right]\left(F^n_{t+t'}(\mathbf{0})\right) && (\text{def. } F^{n+1}_{t+t'}) \\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot (t + t') + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}\left[C'\right]\left(F^n_{t}(\mathbf{0}) + G^n_{t'}(\mathbf{0})\right) && (\text{I.H. on n}) \\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot (t + t') \\
&\qquad + [\![\xi : \mathsf{true}]\!] \cdot \left(\mathsf{ert}\left[C'\right](F^n_t(\mathbf{0})) + \mathsf{wp}\left[C'\right]\left(G^n_{t'}(\mathbf{0})\right)\right) && (\text{I.H. on } C') \\
&= \mathbf{1} + [\![\xi : \mathsf{false}]\!] \cdot t + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{ert}\left[C'\right](F^n_t(\mathbf{0})) \\
&\qquad + [\![\xi : \mathsf{false}]\!] \cdot t' + [\![\xi : \mathsf{true}]\!] \cdot \mathsf{wp}\left[C'\right]\left(G^n_{t'}(\mathbf{0})\right) \\
&= F^{n+1}_t(\mathbf{0}) \;+\; G^{n+1}_{t'}(\mathbf{0}). && (\text{def. } F^{n+1}_t, G^{n+1}_{t'}) \quad \square
\end{aligned}$$

C OMITTED CALCULATIONS

C.1 Invariant Verification for the Random Walk

First, we verify that

$$I_n \;=\; 1 + \sum_{k=0}^{n} [\![x > k]\!] \cdot a_{n,k}$$

is a lower ω-invariant of the loop with respect to $\mathbf{0}$ if for all $n \geq 0$,

$$\begin{aligned}
a_{0,0} &= 1 && (4) \\
a_{n+1,0} &= 2 + \tfrac{1}{2} \cdot (a_{n,0} + a_{n,1}) && (5) \\
a_{n+1,k} &= \tfrac{1}{2} \cdot (a_{n,k-1} + a_{n,k+1}), \text{for all } 1 \leq k \leq n{+}1 && (6) \\
a_{n,k} &= 0, \text{for all } k > n. && (7)
\end{aligned}$$

Let F be the characteristic functional of the loop with respect to $\mathbf{0}$. Then, for the first condition $F(0) \succeq I_0$, we have

$$\begin{aligned}
F(0) &= 1 + [\![x \leq 0]\!] \cdot 0 + [\![x > 0]\!] \cdot \mathsf{ert}[C](0) \\
&= 1 + [\![x > 0]\!] \cdot \left(1 + \tfrac{1}{2} \cdot 0[x/x-1] + \tfrac{1}{2} \cdot 0[x/x+1]\right) \\
&= 1 + [\![x > 0]\!] \cdot 1 \\
&= 1 + [\![x > 0]\!] \cdot a_{0,0} = I_0. && (\text{Equation (4)})
\end{aligned}$$

For the second condition $F(I_n) \succeq I_{n+1}$, consider

$$\begin{aligned}
F(I_n) &= 1 + [\![x \leq 0]\!] \cdot 0 + [\![x > 0]\!] \cdot \mathsf{ert}[C](I_n) \\
&= 1 + [\![x > 0]\!] \cdot \left(1 + \tfrac{1}{2} \cdot I_n[x/x-1] + \tfrac{1}{2} \cdot I_n[x/x+1]\right) \\
&= 1 + [\![x > 0]\!] \cdot \left(\mathbf{2} + \tfrac{1}{2} \cdot \sum_{k=0}^{n} [\![x{-}1 > k]\!] \cdot a_{n,k} + \tfrac{1}{2} \cdot \sum_{k=0}^{n} [\![x{+}1 > k]\!] \cdot a_{n,k}\right) \\
&= 1 + [\![x > 0]\!] \cdot \left(\mathbf{2} + \tfrac{1}{2} \cdot \sum_{k=1}^{n+1} [\![x > k]\!] \cdot a_{n,k-1} + \tfrac{1}{2} \cdot \sum_{k=-1}^{n-1} [\![x > k]\!] \cdot a_{n,k+1}\right)
\end{aligned}$$

$$
\begin{aligned}
&= 1 + [\![x > 0]\!] \cdot \Bigg(\mathbf{2} + \tfrac{1}{2} \cdot [\![x > -1]\!] \cdot a_{n,0} + \tfrac{1}{2} \cdot [\![x > 0]\!] \cdot a_{n,1} \\
&\qquad\qquad + \tfrac{1}{2} \cdot \sum_{k=1}^{n-1} [\![x > k]\!] \cdot (a_{n,k-1} + a_{n,k+1}) \\
&\qquad\qquad + \tfrac{1}{2} \cdot [\![x > n]\!] \cdot a_{n,n-1} + \tfrac{1}{2} \cdot [\![x > n+1]\!] \cdot a_{n,n} \Bigg)
\end{aligned}
$$

We distribute $[\![x > 0]\!]$ and use the fact that $[\![x > 0]\!] \cdot [\![x > a]\!] = [\![x > \max\{0, a\}]\!]$ to obtain

$$
\begin{aligned}
&= 1 + [\![x > 0]\!] \cdot \Big(\mathbf{2} + \tfrac{1}{2} \cdot (a_{n,0} + a_{n,1}) \Big) + \tfrac{1}{2} \cdot \sum_{k=1}^{n-1} [\![x > k]\!] \cdot (a_{n,k-1} + a_{n,k+1}) \\
&\qquad + \tfrac{1}{2} \cdot [\![x > n]\!] \cdot a_{n,n-1} + \tfrac{1}{2} \cdot [\![x > n+1]\!] \cdot a_{n,n} \\
&= 1 + [\![x > 0]\!] \cdot a_{n+1,0} + \tfrac{1}{2} \cdot \sum_{k=1}^{n-1} [\![x > k]\!] \cdot (a_{n,k-1} + a_{n,k+1}) && \text{(Equation (5))} \\
&\qquad + \tfrac{1}{2} \cdot [\![x > n]\!] \cdot a_{n,n-1} + \tfrac{1}{2} \cdot [\![x > n+1]\!] \cdot a_{n,n} \\
&= 1 + [\![x > 0]\!] \cdot a_{n+1,0} + \tfrac{1}{2} \cdot \sum_{k=1}^{n-1} [\![x > k]\!] \cdot (a_{n,k-1} + a_{n,k+1}) && \text{(Equation (7))} \\
&\qquad + \tfrac{1}{2} \cdot [\![x > n]\!] \cdot (a_{n,n-1} + a_{n,n+1}) + \tfrac{1}{2} \cdot [\![x > n+1]\!] \cdot (a_{n,n} + a_{n,n+2}) \\
&= 1 + [\![x > 0]\!] \cdot a_{n+1,0} + \tfrac{1}{2} \cdot \sum_{k=1}^{n+1} [\![x > k]\!] \cdot (a_{n,k-1} + a_{n,k+1}) \\
&= 1 + [\![x > 0]\!] \cdot a_{n+1,0} + \sum_{k=1}^{n+1} [\![x > k]\!] \cdot a_{n+1,k} && \text{(Equation (6))} \\
&= 1 + \sum_{k=0}^{n+1} [\![x > k]\!] \cdot a_{n+1,k} \; = \; I_{n+1}.
\end{aligned}
$$

Now we show that

$$
a_{n,k} = \frac{1}{2^n} \left[-\binom{n}{\lfloor \frac{n-k}{2} \rfloor} + 2 \sum_{i=0}^{n-k} 2^i \binom{n-i}{\lfloor \frac{n-i-k}{2} \rfloor} \right]
$$

satisfies the recursion in Equations (4) to (7). Here, we assume that $\binom{n}{m}$ is 0 whenever $m < 0$. Equations (4) and (7) are immediate. For Equation (5), i.e., $a_{n+1,0} = 2 + 1/2 \cdot (a_{n,0} + a_{n,1})$, we make use of the identity

$$
\binom{k}{\lfloor \frac{k+1}{2} \rfloor} = \binom{k}{\lfloor \frac{k}{2} \rfloor}, \tag{$\bigstar$}
$$

which is shown by a simple case analysis on k being even or odd. Then

$$
\begin{aligned}
a_{n+1,0} &= \frac{1}{2^{n+1}} \left[-\binom{n+1}{\lfloor \frac{n+1}{2} \rfloor} + 2 \sum_{i=0}^{n+1} 2^i \binom{n+1-i}{\lfloor \frac{n+1-i}{2} \rfloor} \right] && (\text{Def. } a_{n+1,0}) \\
&= \frac{1}{2^{n+1}} \Bigg[-\binom{n}{\lfloor \frac{n+1}{2} \rfloor} - \binom{n}{\lfloor \frac{n+1}{2} \rfloor - 1} && \left(\left(\binom{\mathrm{k}+1}{\ell+1}\right) = \binom{\mathrm{k}}{\ell} + \binom{\mathrm{k}}{\ell+1}\right)
\end{aligned}
$$

$$
\begin{aligned}
&\qquad\qquad + 2\sum_{i=0}^{n+1} 2^i \left(\binom{n-i}{\lfloor \frac{n+1-i}{2} \rfloor} + \binom{n-i}{\lfloor \frac{n+1-i}{2} \rfloor - 1} \right) \Bigg] \\
&= \frac{1}{2^{n+1}} \Bigg[- \binom{n}{\lfloor \frac{n+1}{2} \rfloor} - \binom{n}{\lfloor \frac{n+1}{2} \rfloor - 1} \\
&\qquad\qquad + 2^{n+2} + 2\sum_{i=0}^{n} 2^i \left(\binom{n-i}{\lfloor \frac{n+1-i}{2} \rfloor} + \binom{n-i}{\lfloor \frac{n+1-i}{2} \rfloor - 1} \right) \Bigg] \\
&= 2 + \frac{1}{2} \cdot \Bigg(\frac{1}{2^n} \Bigg[-\binom{n}{\lfloor \frac{n-1}{2} \rfloor} + 2\sum_{i=0}^{n-1} 2^i \binom{n-i}{\lfloor \frac{n-i-1}{2} \rfloor} \Bigg] \\
&\qquad\qquad + \frac{1}{2^n} \Bigg[-\binom{n}{\lfloor \frac{n+1}{2} \rfloor} + 2\sum_{i=0}^{n} 2^i \binom{n-i}{\lfloor \frac{n+1-i}{2} \rfloor} \Bigg] \Bigg) && \left(\text{by} \binom{0}{-1} = 0 \text{ and } \left\lfloor \tfrac{n+1}{2} \right\rfloor - 1 = \left\lfloor \tfrac{n-1}{2} \right\rfloor \right) \\
&= 2 + \frac{1}{2} \Bigg(a_{n,1} + \frac{1}{2^n} \Bigg[-\binom{n}{\lfloor \frac{n+1}{2} \rfloor} + 2\sum_{i=0}^{n} 2^i \binom{n-i}{\lfloor \frac{n+1-i}{2} \rfloor} \Bigg] \Bigg) && (\text{Def. } a_{n,1}) \\
&= 2 + \frac{1}{2} \Bigg(a_{n,1} + \frac{1}{2^n} \Bigg[-\binom{n}{\lfloor \frac{n}{2} \rfloor} + 2\sum_{i=0}^{n} 2^i \binom{n-i}{\lfloor \frac{n-i}{2} \rfloor} \Bigg] \Bigg) && (\text{using } \bigstar) \\
&= 2 + \frac{1}{2} \left(a_{n,1} + a_{n,0} \right). && (\text{Def. } a_{n,0})
\end{aligned}
$$

It remains to show Equation (6), i.e., $a_{n+1,k} = \frac{1}{2} \cdot (a_{n,k-1} + a_{n,k+1})$, for all $1 \leq k \leq n+1$:

$$
\begin{aligned}
a_{n+1,k} &= \frac{1}{2^{n+1}} \Bigg[-\binom{n+1}{\lfloor \frac{n+1-k}{2} \rfloor} + 2\sum_{i=0}^{n+1-k} 2^i \binom{n+1-i}{\lfloor \frac{n+1-i-k}{2} \rfloor} \Bigg] && (\text{Def. } a_{n+1,k}) \\
&= \frac{1}{2^{n+1}} \Bigg[-\binom{n+1}{\lfloor \frac{n+1-k}{2} \rfloor} + 2^{n+2-k} + 2\sum_{i=0}^{n-k} 2^i \binom{n+1-i}{\lfloor \frac{n+1-i-k}{2} \rfloor} \Bigg] && \left(\binom{m}{0} = 1 \right) \\
&= \frac{1}{2^{n+1}} \Bigg[-\binom{n}{\lfloor \frac{n-(k-1)}{2} \rfloor} - \binom{n}{\lfloor \frac{n-(k+1)}{2} \rfloor} + 2^{n+2-k} && \left(\binom{k+1}{\ell+1} = \binom{k}{\ell} + \binom{k}{\ell+1} \right) \\
&\qquad + 2\sum_{i=0}^{n-k} 2^i \binom{n-i}{\lfloor \frac{n-i-(k-1)}{2} \rfloor} + 2\sum_{i=0}^{n-k} 2^i \binom{n-i}{\lfloor \frac{n-i-(k+1)}{2} \rfloor} \Bigg] \\
&= \frac{1}{2^{n+1}} \Bigg[-\binom{n}{\lfloor \frac{n-(k-1)}{2} \rfloor} - \binom{n}{\lfloor \frac{n-(k+1)}{2} \rfloor} + 2^{n+2-k} && \left(\binom{m}{-1} = 0 \right) \\
&\qquad + 2\sum_{i=0}^{n-k} 2^i \binom{n-i}{\lfloor \frac{n-i-(k-1)}{2} \rfloor} + 2\sum_{i=0}^{n-(k+1)} 2^i \binom{n-i}{\lfloor \frac{n-i-(k+1)}{2} \rfloor} \Bigg] \\
&= \frac{1}{2} \Bigg(a_{n,k+1} + \frac{1}{2^n} \Bigg[-\binom{n}{\lfloor \frac{n-(k-1)}{2} \rfloor} + 2\sum_{i=0}^{n-(k-1)} 2^i \binom{n-i}{\lfloor \frac{n-i-(k-1)}{2} \rfloor} \Bigg] \Bigg) && (\text{Def. } a_{n,k+1}) \\
&= \frac{1}{2} \left(a_{n,k+1} + a_{n,k-1} \right). && (\text{Def. } a_{n,k-1})
\end{aligned}
$$

Finally, we prove that $\lim_{n\to\infty} a_{n,0} = \infty$. The crux of the proof is showing that for all $n \geq 2$, $a_{n,0} \geq 1 + \mathcal{H}_{\lfloor n/2 \rfloor}$, where $\mathcal{H}_m$ denotes the mth Harmonic number, i.e.,

$$\mathcal{H}_m \;=\; \sum_{k=1}^{m} \tfrac{1}{k}.$$

The result then follows since $\lim_{m\to\infty} \mathcal{H}_m = \infty$. Calculations go as follows:

$$\begin{aligned}
a_{n,0} \;&=\; \frac{1}{2^n}\left[-\binom{n}{\lfloor\frac{n}{2}\rfloor} + 2\sum_{i=0}^{n} 2^i \binom{n-i}{\lfloor\frac{n-i}{2}\rfloor}\right] \\
&=\; \frac{1}{2^n}\left[-\binom{n}{\lfloor\frac{n}{2}\rfloor} + 2\sum_{k=0}^{n} 2^{n-k} \binom{k}{\lfloor\frac{k}{2}\rfloor}\right] && (\text{Take} k = n - i) \\
&\geq\; \frac{1}{2^n}\left[-2^n + 2\sum_{k=0}^{n} 2^{n-k} \binom{k}{\lfloor\frac{k}{2}\rfloor}\right] && \left(\binom{\mathrm{n}}{\lfloor \mathrm{n}/2\rfloor} \leq 2^{\mathrm{n}}\right) \\
&=\; -1 + 2\sum_{k=0}^{n} 2^{-k} \binom{k}{\lfloor\frac{k}{2}\rfloor} \\
&\geq\; -1 + 2\sum_{j=0}^{\lfloor n/2\rfloor} 2^{-2j} \binom{2j}{j} && (\text{Keep only even } k's) \\
&\geq\; 1 + 2\sum_{j=1}^{\lfloor n/2\rfloor} 2^{-2j} \binom{2j}{j} && (\text{Extract } j = 0 \text{ out of the sum}) \\
&\geq\; 1 + 2\sum_{j=1}^{\lfloor n/2\rfloor} 2^{-2j} \frac{2^{2j-1}}{\sqrt{j}} && \left(\text{Stirling approximation :} \tbinom{2j}{j} \geq \tfrac{2^{2j-1}}{\sqrt{j}}\right) \\
&=\; 1 + \sum_{j=1}^{\lfloor n/2\rfloor} \frac{1}{\sqrt{j}} \\
&\geq\; 1 + \sum_{j=1}^{\lfloor n/2\rfloor} \frac{1}{j} = 1 + \mathcal{H}_{\lfloor n/2\rfloor}. && (\sqrt{j} \leq j)
\end{aligned}$$

C.2 Invariant Verification for the Coupon Collector Algorithm

Recall our proposed invariant

$$\begin{aligned}
I \triangleq \mathbf{1} \;+\; &\sum_{\ell=0}^{\infty} [x > \ell] \cdot \left(\mathbf{4} + 2 \cdot \sum_{k=0}^{\infty} \left(\frac{\#col + \ell}{N}\right)^{k}\right) \\
&- 2 \cdot [cp[i] = 0] \cdot [x > 0] \cdot \sum_{k=0}^{\infty} \left(\frac{\#col}{N}\right)^{k}.
\end{aligned}$$

To prove this invariant correct, we have to show that $F(I) \leq I$, where $F(X)$ denotes the characteristic functional of the coupon collector's outer loop with respect to runtime 0. As such, $F(X)$ is given by

$$\begin{aligned}
F(X) \;&= \mathbf{1} + [x \leq 0] \cdot 0 + [x > 0] \cdot \mathrm{ert}[C_{in};\, cp[i] := 1;\, x := x - 1](X) \\
&= \mathbf{1} + [x > 0] \cdot (\mathbf{2} + \mathrm{ert}[C_{in}](X\,[x/x - 1, cp[i]/1])),
\end{aligned}$$

where C_{in} corresponds to the inner loop of the coupon collector algorithm. Thus, in order to verify that $F(I) \leq I$, we need an upper invariant for the inner loop first. Note that this invariant cannot be chosen with respect to continuation 0, but with respect to the invariant I. We will, however, derive an invariant of the inner loop with respect to an arbitrary continuation.

Invariant for the Inner Loop. Given an arbitrary runtime $f \in \mathbb{T}$, we propose that

$$
\begin{aligned}
J_f &\triangleq \mathbf{1} + [cp[i] = 0] \cdot f + [cp[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot \underbrace{\left(2 + \frac{1}{N} \cdot \sum_{j=1}^{N} [cp[j] = 0] \cdot f[i/j]\right)}_{= G_f} \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + [cp[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot G_f
\end{aligned}
$$

is an invariant of the inner loop C_{in} with respect to f. Toward a correctness proof of our proposed invariant, we have to verify that $H_f(J_f) \leq J_f$, where $H_f(Y)$ is the characteristic functional of the inner loop with respect to runtime f. Thus,

$$
\begin{aligned}
& H_f(J_f) \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + [cp[i] \neq 0] \cdot \mathsf{ert}[i :\approx \mathtt{Unif}[1 \ldots N]]\big(J_f\big) \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + [cp[i] \neq 0] \cdot \left(1 + \frac{1}{N} \cdot \sum_{k=1}^{N} J_f[i/k]\right) \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + [cp[i] \neq 0] && \text{(Definition } J_f\text{)} \\
&\quad \cdot \left(1 + \frac{1}{N} \cdot \sum_{k=1}^{N} \left(1 + [cp[k] = 0] \cdot f[i/k] + [cp[k] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot G_f\right)\right) \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + \mathbf{2} \cdot [cp[i] \neq 0] \\
&\quad + \frac{[cp[i] \neq 0]}{N} \cdot \sum_{k=1}^{N} [cp[k] = 0] \cdot f[i/k] + \frac{[cp[i] \neq 0]}{N} \cdot \sum_{k=1}^{N} [cp[k] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot G_f \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + \mathbf{2} \cdot [cp[i] \neq 0] && \text{(Definition of } \#col\text{)} \\
&\quad + \frac{[cp[i] \neq 0]}{N} \cdot \sum_{k=1}^{N} [cp[k] = 0] \cdot f[i/k] + \frac{[cp[i] \neq 0]}{N} \cdot \#col \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot G_f \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + [cp[i] \neq 0] \cdot G_f + \frac{[cp[i] \neq 0]}{N} \cdot \#col \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot G_f \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + [cp[i] \neq 0] \cdot G_f + [cp[i] \neq 0] \cdot \sum_{\ell=1}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot G_f \\
&= \mathbf{1} + [cp[i] = 0] \cdot f + [cp[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot G_f \; = \; J_f .
\end{aligned}
$$

Hence, J_f is an upper global invariant of the inner loop C_{in}. We are now in a position to verify that $F(I) \leq I$; i.e., I is an upper global invariant of the outer loop C_{out}.

Invariant Verification for the Outer Loop. In order to keep calculations readable, let

$$
K_i^j \triangleq \sum_{\ell=i}^{\infty} [x > \ell + j] \cdot \left(\mathbf{4} + 2 \cdot \sum_{k=0}^{\infty} \left(\frac{\#col + j + \ell}{N}\right)^{k}\right).
$$

Then our proposed invariant I can be rewritten as

$$I \;=\; 1 + K_0^0 - 2 \cdot [cp[i] = 0] \cdot [x > 0] \cdot \sum_{k=0}^{\infty} \left(\frac{\#col}{N} \right)^k .$$

The proof $F(I) \leq I$ relies on two properties:

Lemma C.1.

$$N - \#col \;=\; \sum_{i=1}^{N} (1 - [cp[i] \neq 0]) \;=\; \sum_{i=1}^{N} [cp[i] = 0].$$

Proof. Immediate by definition of *#col*. □

Lemma C.2.

$$[cp[i] = 0] \cdot I\,[x/x-1, cp[i]/1] \;=\; [cp[i] = 0] \cdot (1 + K_0^1).$$

Proof. Immediate by observing that

$$I_n \,=\, 1 + K_0^0 - 2 \cdot [cp[i] = 0] \cdot [x > 0] \cdot \sum_{k=0} \left(\frac{\#col}{N} \right)^k$$

and applying the respective substitutions. □

We are now in a position to verify that $F(I) \leq I$; i.e., our proposed invariant I is indeed an upper global invariant of the outer loop.

$$\begin{aligned}
& F(I) \\
&= 1 + [x > 0] \cdot (\mathbf{2} + \mathrm{ert}[C_{in}](I\,[x/x-1, cp[i]/1])) && \text{(Definition of } F\text{)} \\
&\leq 1 + [x > 0] \cdot \left(\mathbf{2} + J_{I[x/x-1, cp[i]/1]}\right) && (\mathrm{ert}[\mathrm{C}_{in}]\,(f) \leq J_f) \\
&= 1 + [x > 0] \cdot \Big(\mathbf{3} + [cp[i] = 0] \cdot I\,[x/x-1, cp[i]/1] && \text{(Definition of } \mathrm{J_f}\text{)} \\
&\qquad + [cp[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot G_{I[x/x-1, cp[i]/1]}\Big) \\
&= 1 + [x > 0] \cdot \Bigg(\mathbf{3} + [cp[i] = 0] \cdot I\,[x/x-1, cp[i]/1] && \text{(Definition of } G_f\text{)} \\
&\qquad + [cp[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot \left(\mathbf{2} + \sum_{j=1}^{N} \frac{[cp[j] = 0]}{N} \cdot I[x/x-1, cp[i]/1, i/j]\right)\Bigg) \\
&= 1 + [x > 0] \cdot \Big(\mathbf{3} + [cp[i] = 0] \cdot (1 + K_0^1) && \text{(Lemma C.2)} \\
&\qquad + [cp[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot \left(\mathbf{2} + \sum_{j=1}^{N} \frac{[cp[j] = 0]}{N} \cdot (1 + K_0^1)\right) \\
&= 1 + [x > 0] \cdot \Big(\mathbf{3} + [cp[i] = 0] \cdot (1 + K_1^0) && (K_0^1 = K_1^0) \\
&\qquad + [cp[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot \left(\mathbf{2} + \sum_{j=1}^{N} \frac{[cp[j] = 0]}{N} \cdot (1 + K_1^0)\right) \\
&= 1 + [x > 0] \cdot \Big(\mathbf{3} + [cp[i] = 0] \cdot (1 + K_1^0) && \text{(Lemma C.1)} \\
&\qquad + [cp[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#col}{N}\right)^{\ell} \cdot \left(\mathbf{2} + \left(1 - \frac{\#col}{N}\right) \cdot (1 + K_1^0)\right)
\end{aligned}$$

$$
\begin{aligned}
&= 1 + [x > 0] \cdot \Big(\mathbf{3} + [\mathit{cp}[i] = 0] \cdot (1 + K_1^0) \\
&\qquad + [\mathit{cp}[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell} \cdot \Big(\mathbf{3} + K_1^0 - \frac{\#\mathit{col}}{N} \cdot (1 + K_1^0)\Big) \\
&= 1 + [x > 0] \cdot \Big(\mathbf{4} + K_1^0 \\
&\qquad + [\mathit{cp}[i] \neq 0] \cdot \sum_{\ell=1}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell} \cdot \left(\mathbf{3} + K_1^0 - \frac{\#\mathit{col}}{N} \cdot (1 + K_1^0)\right) \\
&\qquad + [\mathit{cp}[i] \neq 0] \cdot \left(\mathbf{2} - \frac{\#\mathit{col}}{N}(1 + K_1^0)\right) \\
&= 1 + [x > 0] \cdot \Big(\mathbf{4} + K_1^0 \\
&\qquad + [\mathit{cp}[i] \neq 0] \cdot \sum_{\ell=1}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell} \cdot \Big(\left(1 - \frac{\#\mathit{col}}{N}\right) \cdot (1 + K_1^0) \\
&\qquad + [\mathit{cp}[i] \neq 0] \cdot \left(\mathbf{2} - \frac{\#\mathit{col}}{N}(1 + K_1^0)\right) + \mathbf{2} \cdot [\mathit{cp}[i] \neq 0] \cdot \sum_{\ell=1}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell} \\
&= 1 + [x > 0] \cdot \Big(\mathbf{4} + K_1^0 + [\mathit{cp}[i] \neq 0] \cdot \Big(1 + K_1^0\Big) \cdot \frac{\#\mathit{col}}{N} \\
&\qquad + [\mathit{cp}[i] \neq 0] \cdot \left(\mathbf{2} - \frac{\#\mathit{col}}{N}(1 + K_1^0)\right) + \mathbf{2} \cdot [\mathit{cp}[i] \neq 0] \cdot \sum_{\ell=1}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell} \\
&= 1 + [x > 0] \cdot \left(\mathbf{4} + K_1^0 + 2 \cdot [\mathit{cp}[i] \neq 0] + 2 \cdot [\mathit{cp}[i] \neq 0] \cdot \sum_{\ell=1}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell}\right) \\
&= 1 + [x > 0] \cdot \left(\mathbf{4} + K_1^0 + 2 \cdot [\mathit{cp}[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell}\right) \\
&= 1 + K_1^0 + [x > 0] \cdot \left(\mathbf{4} + 2 \cdot [\mathit{cp}[i] \neq 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell}\right) \\
&= 1 + K_1^0 + [x > 0] \cdot \left(\mathbf{4} + 2 \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell}\right) \\
&\qquad - 2 \cdot [x > 0] \cdot [\mathit{cp}[i] = 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell} \\
&= 1 + K_0^0 - 2 \cdot [x > 0] \cdot [\mathit{cp}[i] = 0] \cdot \sum_{\ell=0}^{\infty} \left(\frac{\#\mathit{col}}{N}\right)^{\ell} && \text{(Definition of } K_0^0\text{)} \\
&= I. && \text{(Definition of } I\text{)}
\end{aligned}
$$

Hence, by Theorem 4.2, we know that $\mathrm{ert}[C_{out}](0) \leq I$.

ACKNOWLEDGMENTS

We thank Gilles Barthe for bringing the coupon collector problem as a particularly intricate case study for formal verification of expected runtimes to our attention, Thomas Noll for bringing Nielson's Hoare logic to our attention, Johannes Hölzl for pointing out a flaw in the analysis of the random walk as in [28], and several contributors from the MathOverflow community for their valuable insights in finding a closed form of the invariant for the random walk problem.

REFERENCES

[1] Rob Arthan, Ursula Martin, Erik A. Mathiesen, and Paulo Oliva. 2009. A general framework for sound and complete Floyd-Hoare logics. *ACM Trans. Comput. Log.* 11, 1 (2009), 7:1–7:31.

[2] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press.

[3] Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. 2016. Synthesizing probabilistic invariants via doob's decomposition. In *Computer-Aided Verification (CAV'16) (LNCS)*, Vol. 9779. Springer, 43–61.

[4] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2018. How long, O Bayesian network, will I sample thee? In *European Symposium on Programming (ESOP'18) (LNCS)*, Vol. 10801. Springer.

[5] Rudolf Berghammer and Markus Müller-Olm. 2004. Formal development and verification of approximation algorithms using auxiliary variables. In *Logic Based Program Synthesis and Transformation (LOPSTR'04) (LNCS)*, Vol. 3018. Springer, 59–74.

[6] Tomás Brázdil, Javier Esparza, Stefan Kiefer, and Antonín Kucera. 2013. Analyzing probabilistic pushdown automata. *Formal Methods Syst. Design* 43, 2 (2013), 124–163.

[7] Tomás Brázdil, Stefan Kiefer, Antonín Kucera, and Ivana Hutarová Vareková. 2015. Runtime analysis of probabilistic programs with unbounded recursion. *J. Comput. Syst. Sci.* 81, 1 (2015), 288–310.

[8] Orieta Celiku and Annabelle McIver. 2005. Compositional specification and analysis of cost-based properties in probabilistic programs. In *Formal Methods (FM'05) (LNCS)*, Vol. 3582. Springer, 107–122.

[9] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic program analysis with martingales. In *Computer Aided Verification (CAV'13) (LNCS)*, Vol. 8044. Springer, 511–526.

[10] Krishnendu Chatterjee, Hongfei Fu, and Aniket Murhekar. 2017. Automated recurrence analysis for almost-linear expected-runtime bounds. In *Computer-Aided Verification (CAV'17) (LNCS)*, Vol. 10426. Springer, 118–139.

[11] Patrick Cousot and Michael Monerau. 2012. Probabilistic abstract interpretation. In *European Symposium on Programming (ESOP'12) (LNCS)*, Vol. 7211. Springer, 169–193.

[12] Edgser W. Dijkstra. 1976. *A Discipline of Programming*. Prentice Hall.

[13] Yijun Feng, Lijun Zhang, David N. Jansen, Naijun Zhan, and Bican Xia. 2017. Finding polynomial loop invariants for probabilistic programs. In *Automated Technology for Verification and Analysis (ATVA'17) (LNCS)*, Vol. 10482. Springer, 400–416.

[14] Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic termination: Soundness, completeness, and compositionality. In *Principles of Programming Languages (POPL'15)*. ACM, 489–501.

[15] Gudmund S. Frandsen. 1998. Randomised Algorithms. Lecture Notes, University of Aarhus, Denmark.

[16] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic programming. In *Future of Software Engineering (FOSE'14)*. ACM, 167–181.

[17] Eric C. R. Hehner. 1998. Formalization of time and space. *Formal Aspects Comput.* 10, 3 (1998), 290–306.

[18] Eric C. R. Hehner. 2011. A probability perspective. *Formal Aspects Comput.* 23, 4 (2011), 391–419.

[19] Wim H. Hesselink. 1993. Proof rules for recursive procedures. *Formal Aspects Comput.* 5, 6 (1993), 554–570.

[20] Timothy Hickey and Jacques Cohen. 1988. Automating program analysis. *J. ACM* 35, 1 (1988), 185–220.

[21] Charles A. R. Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (1969), 576–580.

[22] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. 2012. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.* 34, 3 (2012), 14:1–14:62.

[23] Johannes Hölzl. 2016. Formalising semantics for expected running time of probabilistic programs. In *Interactive Theorem Proving (ITP'16) (LNCS)*, Vol. 9807. Springer, 475–482.

[24] Juraj Hromkovic and Georg Schnitger. 2010. On probabilistic pushdown automata. *Inf. Comput.* 208, 8 (2010), 982–995.

[25] Joe Hurd. 2002. A formal approach to probabilistic termination. In *Theorem Proving in Higher Order Logics (TPHOL'02)*. LNCS, Vol. 2410. Springer, 230–245.

[26] Oliver C. Ibe. 2013. *Elements of Random Walk and Diffusion Processes*. John Wiley & Sons.

[27] Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2015. On the hardness of almost-sure termination. In *Mathematical Foundations of Computer Science (MFCS'15), Part I (LNCS)*, Vol. 9234. Springer, 307–318.

[28] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest precondition reasoning for expected run-times of probabilistic programs. In *European Symposium on Programming (ESOP'16) (LNCS)*, Vol. 9632. Springer, 364–389.

[29] Richard M. Karp. 1994. Probabilistic recurrence relations. *J. ACM* 41, 6 (1994), 1136–1150.

[30] Joost-Pieter Katoen, Annabelle McIver, Larissa Meinicke, and Carroll C. Morgan. 2010. Linear-invariant generation for probabilistic programs: - Automated support for proof-based methods. In *Static Analysis Symposium (SAS'10) (LNCS)*, Vol. 6337. Springer, 390–406.

[31] Dexter Kozen. 1981. Semantics of probabilistic programs. *J. Comput. Syst. Sci.* 22, 3 (1981), 328–350.

[32] Dexter Kozen. 1985. A probabilistic PDL. *J. Comput. Syst. Sci.* 30, 2 (1985), 162–178.

[33] Antonín Kucera, Javier Esparza, and Richard Mayr. 2006. Model checking probabilistic pushdown automata. *Logical Methods Comput. Sci.* 2, 1 (2006), 12–21.

[34] Zohar Manna and Amir Pnueli. 1974. Axiomatic approach to total correctness of programs. *Acta Inf.* 3 (1974), 243–263.

[35] Jeffrey J. McConnell. 2008. *Analysis of Algorithms – An Active Learning Approach.* Jones and Bartlett Publishers.

[36] Annabelle McIver and Carroll Morgan. 2004. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer.

[37] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press.

[38] David Monniaux. 2001. An abstract analysis of the probabilistic termination of programs. In *Symposium on Static Analysis (SAS'01) (LNCS)*, Vol. 2126. Springer, 111–126.

[39] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms.* Cambridge University Press.

[40] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded expectations: Resource analysis for probabilistic programs. In *Programming Language Design and Implementation (PLDI'18).* ACM.

[41] Hanne Riis Nielson. 1987. A Hoare-like proof system for analysing the computation time of programs. *Sci. Comput. Program.* 9, 2 (1987), 107–136.

[42] Hanne Riis Nielson and Flemming Nielson. 2007. *Semantics with Applications: An Appetizer.* Springer.

[43] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about recursive probabilistic programs. *CoRR* abs/1603.02922 (2016). Retrieved from http://arxiv.org/abs/1603.02922.

[44] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about recursive probabilistic programs. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16).* IEEE Computer Society, 672–681.

[45] Saminathan Ponnusamy. 2011. *Foundations of Mathematical Analysis.* Springer Science & Business Media.

[46] Eric Schechter. 1996. *Handbook of Analysis and Its Foundations.* Elsevier Science.

[47] Alejandro Serrano, Pedro López-García, and Manuel V. Hermenegildo. 2014. Resource usage analysis of logic programs via abstract interpretation using sized types. *TPLP* 14, 4–5 (2014), 739–754.

[48] Wolfgang Wechler. 1992. *Universal Algebra for Computer Scientists.* EATCS Monographs on Theoretical Computer Science, Vol. 25. Springer.

[49] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenström. 2008. The worst-case execution-time problem - Overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.* 7, 3 (2008), 36:1–36:53.

[50] Glynn Winskel. 1993. *The Formal Semantics of Programming Languages: An Introduction.* MIT Press.

Received May 2017; revised February 2018; accepted April 2018