# Positive Almost-Sure Termination: Complexity and Proof Rules

RUPAK MAJUMDAR, Max Planck Institute for Software Systems (MPI-SWS), Germany
V. R. SATHIYANARAYANA, Max Planck Institute for Software Systems (MPI-SWS), Germany

We study the recursion-theoretic complexity of Positive Almost-Sure Termination (PAST) in an imperative programming language with rational variables, bounded nondeterministic choice, and discrete probabilistic choice. A program terminates positive almost-surely if, for every scheduler, the program terminates almost-surely and the expected runtime to termination is finite. We show that PAST for our language is complete for the (lightface) co-analytic sets ($\Pi^1_1$-complete). This is in contrast to the related notions of Almost-Sure Termination (AST) and Bounded Termination (BAST), both of which are arithmetical ($\Pi^0_2$- and $\Sigma^0_2$-complete respectively).

Our upper bound implies an effective procedure to reduce reasoning about probabilistic termination to non-probabilistic fair termination in a model with bounded nondeterminism, and to simple program termination in models with unbounded nondeterminism. Our lower bound shows the opposite: for every program with unbounded nondeterministic choice, there is an effectively computable probabilistic program with bounded choice such that the original program is terminating if, and only if, the transformed program is PAST.

We show that every program has an effectively computable normal form, in which each probabilistic choice either continues or terminates execution immediately, each with probability 1/2. For normal form programs, we provide a sound and complete proof rule for PAST. Our proof rule uses transfinite ordinals. We show that reasoning about PAST requires transfinite ordinals up to $\omega_1^{\mathrm{CK}}$; thus, existing techniques for probabilistic termination based on ranking supermartingales that map program states to reals do not suffice to reason about PAST.

CCS Concepts: • **Theory of computation** → **Program reasoning**; • **Mathematics of computing** → *Probabilistic algorithms*; *Stochastic processes*.

Additional Key Words and Phrases: probabilistic programs, demonic non-determinism, positive almost-sure termination, computational complexity, program reasoning

**ACM Reference Format:**
Rupak Majumdar and V. R. Sathiyanarayana. 2024. Positive Almost-Sure Termination: Complexity and Proof Rules. *Proc. ACM Program. Lang.* 8, POPL, Article 37 (January 2024), 29 pages. https://doi.org/10.1145/3632879

## 1 INTRODUCTION

A *probabilistic* program augments an imperative program with primitives for randomization. Probabilistic programs allow direct implementation of randomized computation and probabilistic modeling and have found applications in machine learning, bio-informatics, epidemiology, and information retrieval amongst others; see Katoen et al. [2015] for a comprehensive presentation of their applicability.

Authors' addresses: Rupak Majumdar, Max Planck Institute for Software Systems (MPI-SWS), Paul-Ehrlich-Straße, Building G26, Kaiserslautern, 67663, Germany, rupak@mpi-sws.org; V. R. Sathiyanarayana, Max Planck Institute for Software Systems (MPI-SWS), Paul-Ehrlich-Straße, Building G26, Kaiserslautern, 67663, Germany, sramesh@mpi-sws.org.


ACM 2475-1421/2024/1-ART37
https://doi.org/10.1145/3632879

```
x := 1
while (x ≠ 0):
   x := x + 1 ⊕½ x := x - 1
```

(a) This program is AST but not PAST.

```
1 x, y, z := 0, 0, 1
2 while (x + y = 0):
3    y := 0 [] y := 1
4    x := 0 ⊕½ x := 1
5    z := z * 4
6 while (x = 0 ∧ z > 0):
7    z := z - 1
```

(b) This program is PAST but not BAST.

Prg. 1. Programs showcasing the relationships between AST, PAST, and BAST.

We study programs written in a classical imperative language with constructs for bounded (binary) nondeterministic choice $P_1 \,[]\, P_2$ and discrete probabilistic choice $P_1 \oplus_p P_2$. The first program can nondeterministically reduce to either $P_1$ or $P_2$; the second reduces to $P_1$ with probability $p$ and to $P_2$ with probability $1 - p$.

A fundamental and classical question about programs is *termination*: does the execution of a program stop after a finite number of steps? In the presence of nondeterministic choice, a program can have many executions, depending on how the nondeterminism is resolved. Typically, nondeterminism is modelled as being resolved demonically by an uncaring *scheduler*, and the termination question is modified to ask: does the program stop after a finite number of steps no matter how the scheduler resolves nondeterminism?

If, in addition, a program has probabilistic choice, the notion of termination has to be modified to exclude some ostensibly infinite executions with a total measure of zero. For example, if a program repeatedly tosses a fair coin until it lands heads, it will halt with probability one, as the probability of observing an infinite sequence of tails is zero.

Consequently, several qualitative notions of termination have been defined and studied. A program is *almost sure terminating*, written AST, if for every scheduler, the probability of termination is one. A program is *positive* almost sure terminating, written PAST, if for every scheduler, the expected run time to termination is finite. Finally, a program is *bounded* almost sure terminating, written BAST, if there exists a global bound on expected run times to termination independent of the scheduler.

Clearly, every BAST program is also PAST, and every PAST program is also AST. In the absence of nondeterminism, PAST and BAST coincide. However, these notions are different in general, as illustrated in Programs 1a and 1b.

Program 1a is the famous symmetric random walker, which terminates almost surely (i.e., is AST) but cannot expect to do so in a finite amount of time [Pólya 1921]. Meanwhile, Program 1b is PAST, but the longer the scheduler keeps the execution inside the first loop (from Lines 2 to 5), the greater its expected runtime. Thus, it is not BAST. However, replacing Line 5 by `z = z + 1` induces an upper bound of 4 over all possible expected runtimes, making it BAST.

All these notions have been studied extensively, both with and without (demonic) nondeterminism [Bournez and Garnier 2005; Fu and Chatterjee 2019; McIver and Morgan 2005; McIver et al. 2018; Pnueli 1983]. One main focus of these works has been the development of proof rules to prove that a given program terminates under one of these notions. Most of this work has focused on AST and BAST; relatively little is known for PAST.

In this paper, we characterize the recursion-theoretic complexity of PAST and provide a semantically sound and complete proof rule. Our first result is that membership in PAST is complete for the (lightface) co-analytic sets, that is, $\Pi^1_1$-complete. This is in contrast to AST and BAST, both of which lie in the arithmetic hierarchy ($\Pi^0_2$-complete and $\Sigma^0_2$-complete, respectively [Kaminski et al. 2019]). Hardness already holds with binary nondeterministic choice and probabilistic choice of the form

$$\texttt{skip} \oplus_{1/2} \texttt{exit} \qquad \text{(Knievel form)}$$

which continues execution or halts with probability 1/2 each. A consequence of our result is that every probabilistic program has an effectively constructible *normal form*, which we call *Knievel form* (after Evel Knievel, who made many such choices in his life). Our second main result is a sound and complete proof rule for Knievel form PAST programs. We prove that proof systems for PAST require transfinite ordinals up to the first non-computable ordinal $\omega_1^{\mathsf{CK}}$, also known as the Church-Kleene ordinal. This is in contrast to AST and BAST, neither of which require transfinite ordinals. In fact, most proof systems for AST and BAST use *ranking supermartingales* that map program states to the reals with the proviso that each program transition decreases the expected value of the mapping by a minimum amount [Chakarov and Sankaranarayanan 2013; Fioriti and Hermanns 2015; Fu and Chatterjee 2019]. Our result shows that such an attempt will not work for PAST. To illustrate this claim, we describe in Section 2 a stochastic variant of the *Hydra* game [Kirby and Paris 1982] that shows an intuitive example of a PAST program that requires transfinite ordinals up to $\varepsilon_0$ to demonstrate termination. Recall that the complexity of valid statements in the standard model of arithmetic is $\Delta^1_1$ [Rogers Jr. 1987]; thus, relative completeness results for PAST must use more powerful proof systems.

Our PAST proof rule for Knievel form programs uses two ingredients. The first is a *ranking function* from program states to ordinals up to $\omega_1^{\mathsf{CK}}$ with the property that only terminal states are ranked zero. The second is a state-dependent certificate, based on ranking supermartingales, for a bound on the expected time to reach a state with a lower rank independent of the scheduler.

We show that for every program—not necessarily in Knievel form—the proof rule is complete: from every PAST program, one can extract a rank and a certificate. Moreover, by analyzing the possible traces of programs in Knievel form, we show that the rule is sound: the existence of such a ranking function and a ranking supermartingale implies that the expected running time is bounded for each scheduler. However, soundness depends on the normal form: the rule is not sound if applied to general programs. Since our first result provides an effective transformation to Knievel form, we nevertheless get a semantically sound and complete proof system by first transforming the program into the normal form and then applying the proof rule.

We also show that ordinals up to $\omega_1^{\mathsf{CK}}$ are necessary by explicitly constructing, for each ordinal $\mathrm{o} < \omega_1^{\mathsf{CK}}$, a PAST program for which suitable ranking functions include o in their range. Our construction encodes a recursive $\omega$-tree $T$ into a probabilistic program $P(T)$ such that $T$ is well-founded *iff* $P(T)$ is PAST—recall that the constructible ordinals are coded by such trees [Kozen 2006].

Our results are related to termination and fair termination problems for non-probabilistic programs with unbounded countable nondeterministic choice [Apt and Plotkin 1986; Chandra 1978; Harel 1986; Harel and Kozen 1984]. The $\Pi^1_1$-completeness and the requirement of ordinals up to $\omega_1^{\mathsf{CK}}$ for deciding termination of programs with countable nondeterministic choice was shown by Chandra [1978] and Apt and Plotkin [1986]. Additionally, Harel [1986] showed a general recursive transformation on trees with bounded nondeterministic choice and fairness that reduces fair termination to termination, thereby providing a semantically complete proof system for fair termination. Since fairness can simulate countable nondeterminism using bounded nondeterminism, these results also show a lower complexity bound and the necessity of transfinite ordinals for fair termination. Our results show that countable nondeterminism and discrete probabilistic choice has the same power.

```
n := 4 # initial regrowth capacity
while (True):
    if (empty(hydra)): exit # Hercules has killed the Hydra
    l := Hercules(hydra) # Hercules's choice
    parent := getParent(hydra, l) # the parent node
    grandParent := getParent(hydra, parent) # the grandparent node
    hydra := removeLeaf(hydra, l)  # disconnect head

    if (not empty(grandparent)): # grow new heads
        evolve := 0 [] evolve := 1 # Hydra's move
        while(evolve):
            skip ⊕_{1/2} exit  # die with some probability
            n := n * 4 # quadruple regrowth capacity
            evolve := 0 [] evolve := 1 # Hydra's move
        subtree := getSubtree(hydra, parent)  # find the place to grow heads
        hydra := growNewHeads(n - 1, grandparent, subtree)  # grow new heads
```

Prg. 2. The Hydra Game. $s_1 \oplus_{1/2} s_2$ is a probabilistic choice between statements $s_1$ and $s_2$: the program transitions to $s_1$ or $s_2$ with probability 1/2 each. $s_1 [] s_2$ is a nondeterministic choice: the program transitions to $s_1$ or $s_2$ nondeterministically.

We summarize our main results below:

(1) Deciding if a probabilistic program with bounded nondeterministic and probabilistic choice is PAST is $\Pi_1^1$-complete.
(2) For any probabilistic program $P$, there is an effectively constructible Knievel form program $P_K$ and non-probabilistic program $P_1$ with bounded nondeterministic choice and non-probabilistic program $P_2$ with unbounded choice such that $P$ is PAST *iff* $P_K$ is PAST *iff* $P_1$ is fairly terminating *iff* $P_2$ is terminating.
(3) For any recursive $\omega$-tree $T$, there is a probabilistic program $P(T)$ such that $T$ is well-founded *iff* $P(T)$ is PAST. Hence, proving PAST requires ordinals up to $\omega_1^{\mathsf{CK}}$.
(4) There is a sound and complete proof rule for Knievel form programs that uses a (deterministic) ranking function with codomain $\omega_1^{\mathsf{CK}}$ and ranking supermartingales. While the rule is complete for every PAST program, it is only sound for programs in Knievel form.

## 2 A HYDRA GAME: PAST REQUIRES TRANSFINITE ORDINALS

We now illustrate our main arguments in a stochastic variant of the *Hydra game*, a two player game between the warrior Hercules and the Lernaean Hydra. Introduced by Kirby and Paris [1982], the deterministic Hydra game terminates but requires transfinite ordinals to prove as much. Our stochastic version is PAST and similarly requires transfinite ordinals to prove its membership.

Just like the original [Kirby and Paris 1982], our stochastic variant is a two player-game between Hercules and the Hydra. The Hydra is a finite rooted tree. A *head* of the Hydra is a leaf together with the edge connecting the leaf to the tree. Naturally, the Hydra can have multiple heads.

Each round of the game begins with Hercules chopping off one of the Hydra's heads. In the traditional game, the Hydra responds by growing two new heads and ending the round. Our variant is a little different. First, our game maintains a number n, initially 4, that measures the Hydra's head growth capabilities. Additionally, our Hydra can (try to) improve its chances by attempting to *evolve* several times. Evolution is risky: with probability 1/2, it causes the Hydra to implode, instantly ending the game in Hercules' favour. However, if successful, it quadruples the Hydra's growth capacity n. After (possibly many) successful evolution(s), the Hydra instantly grows new heads in

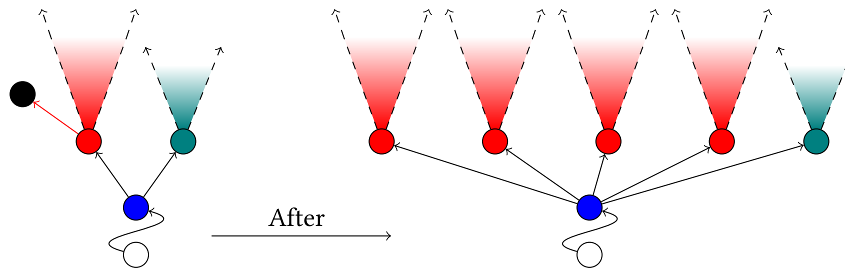


Fig. 1. *A round in the Hydra game.* The curved edges represent potential intermediate nodes. The head Hercules targets is filled in black, its parent is marked red, and its grandparent blue. The remaining subtree from the parent is shaded in red, and is duplicated at the end of the round. Here, the Hydra hasn't evolved, and hence only 3 new heads are grown. Observe that the subtree shaded blue green is entirely unaffected.

the following way: if the grandparent node `grandParent` of the leaf chopped of by Hercules exists, $\mathtt{n}-1$ smaller hydras are grown beneath `grandParent`, with each baby hydra taking the shape of the remaining subtree rooted at the parent of the leaf that was chopped off. Hercules now picks and chops off another head, moving the game onto its next round.

The game is described in greater detail in Program 2. See Fig. 1 for an illustration of a move. Notice that the Hydra cannot evolve or grow new heads if the leaf removed by Hercules had no grandparent.

By fixing Hercules's strategy to any recursive function and considering the nondeterministic choices at Lines 10 and 14 (in Program 2) demonically, the progression of this game becomes the execution of a probabilistic program. The fact that this program has a *finite expected runtime* for every possible nondeterministic scheduler (i.e., is PAST) is observable from two facts: one, the deterministic hydra game only has a finite number of rounds [Kirby and Paris 1982], and two: each round, in expectation, only takes a constant amount of time.

Our goal in this section is to illustrate the apparatus required to prove that the game is PAST. Termination is usually demonstrated through *ranking functions.* In the original, deterministic, Hydra game, there is in fact a ranking function of the form discussed by Francez [1986] and Manna [1974] mapping program states to natural numbers tracking the upper bound on the remaining length of the game. This is because the supremum of the game's length from every state (varied over the strategies employed by Hercules) is always finite, in spite of the ordinals necessary to show this. Unfortunately, because of nondeterministic choices, our variant does not have an upper bound on the expected runtime independent of the scheduler.

Prior work in proving BAST for probabilistic programs [Chatterjee and Fu 2017; Fioriti and Hermanns 2015] uses *ranking supermartingales*, a generalization of ranking functions. A ranking supermartingale maps program states to real values in such a way that in expectation, the function strictly decreases by at least some minimum amount at each execution step. Ranking supermartingales form a sound and complete proof rule for BAST. Unfortunately, we show that, despite a finite expected run time, we cannot find such a function for the stochastic Hydra game. Indeed, we show that a termination argument for the Hydra game must use transfinite ordinals.

We begin by introducing, following Kirby and Paris [1982], a useful mapping $T$ from nodes in the Hydra to ordinals. The range of the mapping is $\varepsilon_0$, the smallest solution to the ordinal equation $x = \omega^x$.

*Definition 2.1 (Ordinal mapping of nodes in the Hydra).* Let $\mathtt{hydra} = (V, E)$ be a finite tree. Define the mapping $T : V \longrightarrow \varepsilon_0$ with following properties:

- For every leaf node $v \in V$, $T(v) = 0$
- For every internal node $v \in V$ with children $v_1, v_2, \ldots v_m$ listed in decreasing order of the ordinals assigned to them by $T$,

$$T(v) = \sum_{i=1}^{m} \omega^{T(v_i)}$$

  In other words, $T(v)$ is the *natural sum* of all $\omega^{T(v')}$ over all children $v'$ of the node $v$.

In each round of the Hydra game, if the Hydra survives, the ordinal assigned to the root of the Hydra by $T$ always reduces [Kirby and Paris 1982]. This is despite the increments to the regeneration capacity enabled by evolution.

In this work, we attempt to generalize ranking arguments to our setting. We want to find ranking functions whose range are the ordinals such that they decrease in expectation in each step, and only terminal states are given rank zero. Since the ordinals are well-founded, this decrease in rank resembles the expected remaining length of execution. One could imagine that perhaps the ordinals are unnecessary and there is a clever encoding into existing ranking arguments, like the ones by Chatterjee and Fu [2017]. Unfortunately, we show that the naturals (or even the reals) cannot serve as an appropriate range for functions that guarantee an expected decrease of 1 in each step.

Consider a starting point of a simple line Hydra of length 2 that, after $n$ nondeterministic evolution steps, grows $4^n - 1$ new heads with probability $1/2^{n+1}$. Note that the Hydra can no longer evolve or grow heads from this state, and the game must hence be played for exactly $4^n + 1$ more steps to terminate. Suppose there is a ranking supermartingale that assigns to the line Hydra a natural (or real) number $m$. This function must necessarily assign to the new Hydra a value of greater than $4^n$.Our requirements on the ranking function now imply that

$$m \geq \frac{1}{2^{n+1}} \times 4^n \implies m \geq 2^{n-1}$$

By engineering a sufficiently large value of $n$, the Hydra can invalidate this inequality. Hence, neither the naturals nor the reals can serve as a sufficient co-domain of the desired ranking function. However, the infinite ordinal $\omega$ is an excellent choice of rank for the line hydra.

Allowing ordinals in the ranking function creates a new challenge. What must be the rank of a state that, with some probability $0 < p < 1$, can reach a state of rank $\omega$? For simplicity, we set the following additional requirement on our ranking functions: if, in one round, the game can reach a state with ordinal rank $x$ with positive probability $p$, then the source state must be ranked above $x$.

With this additional property, we claim that *the smallest appropriate ranking function for the Hydra game agrees with $T$ at all ordinal outputs.* This is because $T$ assigns to the root the smallest ordinal greater than all ordinals reachable in a single step. We formalize this in Lemma 2.2.

Lemma 2.2. *From any Hydra $H$ with root node $r$ with $T(r) \geq \omega$, one can reach, in one step and with non-zero probabilities, an infinite sequence of hydras $H_1, H_2, \ldots$ with roots $r_1, r_2, \ldots$ such that the smallest ordinal larger than $T(r_1), T(r_2), \ldots$ is $T(r)$.*

Hence, the smallest appropriate ranking function for our requirements is $T$, indicating that at the very least, all ordinals under $\varepsilon_0$ are needed to reason about the expected runtime of programs with both nondeterministic and probabilistic operators. In Section 4, we see that ordinals up to the Church-Kleene ordinal $\omega_1^{\mathsf{CK}}$ are needed to reason about general probabilistic programs. We include the proof of Lemma 2.2 in our full version [Majumdar and Sathiyanarayana 2023] for completeness.

In summary, our proof rule for proving PAST has three ingredients: a *normal form* for programs in which every probabilistic choice is of the form `skip` $\oplus_{1/2}$ `exit` (which, fortunately, the Hydra is already in), an ordinal-valued ranking function (like the function $T$ above), and a proof that the rank decreases in an expected finite number of steps despite nondeterminism (a BAST property, for which sound and complete proof rules exist). Putting them together, we can argue that the stochastic Hydra is PAST: the rank decreases in an expected finite number of steps and the rank decreases a finite number of times until termination. As we show later, the proof rule is semantically sound and complete, but the normal form is essential before it can be applied.

## 3 PROBABILISTIC PROGRAMS AND THEIR TERMINATION

We now define the program model and the various notions of termination.

### 3.1 Program Model

The program model we employ is a straightforward nondeterministic extension of the language described by Kaminski et al. [2019]. The syntax mirrors pGCL, an extension of Dijkstra's Guarded Command Language (GCL, [Dijkstra 1976]) that adds binary probabilistic and nondeterministic choice operators.

*Definition 3.1 (Syntax of* pGCL*).* Let Var be a countable set of variable symbols. Programs in pGCL obey the grammar:

$$\mathtt{Prog} ::= \bot \mid v := e \mid \mathtt{Prog};\mathtt{Prog} \mid \mathtt{Prog} \oplus_p \mathtt{Prog} \mid \mathtt{Prog} \,[]\, \mathtt{Prog} \mid \mathtt{while}(b)\{\,\mathtt{Prog};\,\}$$

where $v \in \mathsf{Var}$, $e$, $p$, and $b$ are arithmetical and boolean expressions over Var, $\oplus_p$ is a *probabilistic choice operator*, and $[]$ is a *nondeterministic choice operator.*

$\bot$ here is the empty program. We omit the usual `exit`, `skip`, and `if` structures for brevity, as they can easily be simulated in the mentioned syntax. Note the binary branching at probabilistic and nondeterministic operators.

In order to describe our semantics for pGCL programs, we need to formalize the notion of *the scheduler.* Informally, a scheduler maps execution histories to actions at nondeterministic points in the program. Since the execution of pGCL programs can be uniquely determined from the sequence of decisions made at probabilistic and nondeterministic locations, we present the following more useful non-standard (but equivalent) definition for schedulers:

*Definition 3.2 (Scheduler).* Let $\Sigma_n = \{L_n, R_n\}$ and $\Sigma_p = \{L_p, R_p\}$. A scheduler is simply a *total* mapping from $(\Sigma_n \cup \Sigma_p)^* \to \Sigma_n$. Here, the alphabets $\Sigma_n$ and $\Sigma_p$ represent the *Left* and *Right* directions available at *nondeterministic* and *probabilistic* operators respectively.

The following operational semantics for pGCL programs extends those of Kaminski et al. [2019] with consideration for nondeterministic choice.

*Definition 3.3 (Semantics of* pGCL*).* Declare the following notations:

- $\mathbb{V} \triangleq \{\eta \mid \eta : \mathsf{Var} \to \mathbb{Q}\}$ is the set of all possible *variable valuations.*
- Prog is the collection of all programs derivable in the grammar specified in Definition 3.1.
- $\mathbb{F}$ is the set of all schedulers (defined in Definition 3.2).
- $\mathbb{P} \triangleq \mathtt{Prog} \times \mathbb{V} \times (\mathbb{Q}^+ \cap [0,1]) \times \{L_n, R_n, L_p, R_p\}^*$ is the set of all *execution states.*

Additionally, let $[\![e]\!]_\eta$ and $[\![b]\!]_\eta$ be the evaluations of the arithmetical and boolean expressions $e$ and $b$ under the variable valuation $\eta \in \mathbb{V}$. The operational semantics of pGCL programs under a scheduler $f \in \mathbb{F}$ is defined by the smallest relation $\vdash_f \subseteq \mathbb{P} \times \mathbb{P}$ that complies with the inference rules illustrated in Fig. 2. Furthermore, we define the transitive extensions $\vdash_f^n$ and $\vdash_f^*$ by setting $\vdash_f^1 \triangleq \vdash_f$ and for all $n \in \mathbb{N}$,

$$(\sigma, \sigma') \in \vdash_f^{n+1} \Longleftrightarrow \exists \tau \in \mathbb{P} \cdot (\sigma, \tau) \in \vdash_f^n \wedge (\tau, \sigma') \in \vdash_f \qquad \text{and} \qquad \vdash_f^* \triangleq \bigcup_{i \in \mathbb{N}} \vdash_f^i$$

$$\textsc{assign}\quad (v := e, \eta, a, w) \vdash_f (\bot, \eta[v \mapsto [\![e]\!]_\eta], a, w)$$

$$\textsc{concat1}\ \frac{(P_1, \eta, a, w) \vdash_f (P_1', \eta', a', w')}{(P_1; P_2, \eta, a, w) \vdash_f (P_1'; P_2, \eta', a', w')} \qquad \textsc{concat2}\ \frac{}{(\bot; P_2, \eta, a, w) \vdash_f (P_2, \eta, a, w)}$$

$$\textsc{prob1}\ \frac{[\![p]\!]_\eta \leq 0}{(P_1 \oplus_p P_2, \eta, a, w) \vdash_f (P_2, \eta, a, w \cdot R_p)} \qquad \textsc{prob2}\ \frac{[\![p]\!]_\eta \geq 1}{(P_1 \oplus_p P_2, \eta, a, w) \vdash_f (P_1, \eta, a, w \cdot L_p)}$$

$$\textsc{prob3}\ \frac{0 < [\![p]\!]_\eta < 1}{(P_1 \oplus_p P_2, \eta, a, w) \vdash_f (P_1, \eta, a \times [\![p]\!]_\eta, w \cdot L_p)} \qquad \textsc{prob4}\ \frac{0 < [\![p]\!]_\eta < 1}{(P_1 \oplus_p P_2, \eta, a, w) \vdash_f (P_2, \eta, a \times (1 - [\![p]\!]_\eta), w \cdot R_p)}$$

$$\textsc{nondet1}\ \frac{f(w) = L_n}{(P_1 \mathbin{[]} P_2, \eta, a, w) \vdash_f (P_1, \eta, a, w \cdot L_n)} \qquad \textsc{nondet2}\ \frac{f(w) = R_n}{(P_1 \mathbin{[]} P_2, \eta, a, w) \vdash_f (P_2, \eta, a, w \cdot R_n)}$$

$$\textsc{loop1}\ \frac{[\![b]\!]_\eta = 1}{(\texttt{while}(b)\{P;\}, \eta, a, w) \vdash_f (P; \texttt{while}(b)\{P;\}, \eta, a, w)} \qquad \textsc{loop2}\ \frac{[\![b]\!]_\eta = 0}{(\texttt{while}(b)\{P;\}, \eta, a, w) \vdash_f (\bot, \eta, a, w)}$$

Fig. 2. Semantics of pGCL

In Definition 3.3, $\mathbb{N}$ and $\mathbb{Q}$ are standard denotations for the sets of natural and rational numbers respectively. The operation $\eta[v \mapsto [\![e]\!]_\eta]$ is standard; it refers to the variable valuation that agrees with $\eta$ on all variables in $\mathsf{Var} \setminus \{v\}$ and assigns to $v$ the value $[\![e]\!]_\eta$. Like Dijkstra [1976], we restrict the range of values available to variables to the rationals; this avoids measure-theoretic apparatus that would be required otherwise [Bertsekas and Shreve 1978; Takisaka et al. 2021]. Our semantics extends that of Kaminski et al. [2019] in remembering the decisions made at nondeterministic execution states in addition to the branching at probabilistic states. This additional information facilitates compliance with the scheduler $f$.

To clarify later definitions, we distinguish the notion of the *program state* from the execution state.

*Definition 3.4 (Program states).* A program state is simply a pair of pGCL program $P$ and a variable valuation $\eta$. The set of all program states $\Sigma$ is hence simply $\texttt{Prog} \times \mathbb{V}$.

For a fixed program $P \in \texttt{Prog}$, we denote the *initial program state* $(P, \eta_0)$ by $\sigma_{P,0}$, and the *initial execution state* $(P, \eta_0, 1, \varepsilon)$ by $\sigma^e_{P,0}$. Here, $\varepsilon$ is the empty word in the language $(\Sigma_p \cup \Sigma_n)^*$ and unless otherwise specified, the initial variable valuation $\eta_0$ maps all variables in $\mathsf{Var}$ to 0. Furthermore, every execution state of the form $(\bot, \eta, p, w)$ and program states of the form $(\bot, \eta)$ are said to be *terminal.*

*Definition 3.5 (Execution tree).* Let $P$ be a pGCL program, $f$ be a scheduler, and let $\eta_0$ be some fixed initial variable valuation. Denote the *initial execution state* $(P, \eta_0, 1, \varepsilon)$ by $\sigma^e_{P,0}$. The execution tree of the program $P$ under the scheduler $f$ is the subgraph of $(\mathbb{P}, \vdash_f)$ over the vertices $\{\sigma \in \mathbb{P} \mid \sigma^e_{P,0} \vdash^*_f \sigma\}$

Our semantics ensures that the execution tree is a tree rooted at $\sigma^e_{P,0}$. Incidentally, Definition 3.3 agrees with the semantics defined in Definition 4 of Kaminski et al. [2019] for programs in $\texttt{Prog}$ that do not use the nondeterministic choice operator $[]$ .

### 3.2 Notions of Termination

In this subsection, we formalize the various notions of termination motivated in Section 1. We begin with two necessary projection operations.

*Definition 3.6.* Prob and Hist are total functions from the set of execution states $\mathbb{P}$ that satisfy

$$\mathsf{Prob}((\_,\_,p,\_)) = p \qquad \text{and} \qquad \mathsf{Hist}((\_,\_,\_,w)) = w$$

Here, _ is shorthand for any arbitrary value.

We now turn to termination probabilities. Unlike deterministic programs, the probabilities of termination of pGCL programs depend on the scheduler employed to resolve non-determinism. It is quite possible for a program to fully terminate under one scheduler and run forever under another.

*Definition 3.7 (Termination Probability).* Let $T_e$ be a function that takes in a program state $\sigma = (P, \eta)$ and a scheduler $f$ and returns the set of *terminal execution states* reachable from the corresponding initial execution state $\sigma_e = (P, \eta, 1, \varepsilon)$ under the scheduler $f$. Thus,

$$T_e(\sigma, f) \triangleq \{(\bot, \eta, p, w) \in \mathbb{P} \mid \sigma_e \vdash_f^* (\bot, \eta, p, w)\}$$

*Termination probability* is a function that takes in a program state $\sigma$ and a scheduler $f$ and returns the probability of the termination of the execution initialized at $\sigma$ under $f$ by adding up the probabilities of states in $T_e(\sigma, f)$:

$$\Pr_{\text{term}}(\sigma, f) = \sum_{\sigma' \in T_e(\sigma, f)} \mathsf{Prob}(\sigma')$$

We now define the set AST that we motivated in Section 1.

*Definition 3.8 (Almost-sure termination).* AST (short for *Almost-Surely Terminating*) is the set of all pGCL programs $P$ that yield a termination probability of 1 from their initial states $\sigma_{P,0}$ under every possible scheduler $f \in \mathbb{F}$, i.e.,

$$\mathsf{AST} = \{P \in \mathsf{Prog} \mid \forall\!\!\forall f \in \mathbb{F} \cdot \Pr_{\text{term}}(\sigma_{P,0}, f) = 1\}$$

The symbol $\forall\!\!\forall$ indicates that $f$ is a second-order variable. This is necessary because the set $\mathbb{F}$ is not a countable entity. We will return to this detail in Section 4.

Before we discuss the other notions of termination motivated in Section 1, we present definitions for expected runtime. We extend a useful presentation motivated by Fioriti and Hermanns [2015]: the expected runtime is the sum of the infinite series of the probabilities of surviving beyond $n$ steps.

*Definition 3.9 (Expected runtime).* Let $T_e^{\leq k}$ be a function that takes as input a program state $\sigma = (P, \eta)$ and a scheduler $f$ and returns the set of all terminal states reachable in $\leq k$ steps from the corresponding execution state $\sigma_e = (P, \eta, 1, \varepsilon)$ under the scheduler $f$:

$$T_e^{\leq k}(\sigma, f) = \{(\bot, \eta, p, w) \in \mathbb{P} \mid \exists n \in \mathbb{N} \cdot n \leq k \wedge \sigma_e \vdash_f^n (\bot, \eta, p, w)\}$$

The *expected runtime* from a program state $\sigma$ under the scheduler $f$ is the sum

$$\mathsf{ExpRuntime}(\sigma, f) \triangleq \sum_{k \in \mathbb{N}} \left(1 - \sum_{\sigma' \in T_e^{\leq k}(\sigma, f)} \mathsf{Prob}\,(\sigma')\right)$$

Observe that, as in the case of deterministic programs, the expected runtime can diverge.

We now present two notions: that of *positive almost-sure termination* and *bounded termination*. Positive almost-sure termination, introduced in Bournez and Garnier [2005] and refined in Fioriti and Hermanns [2015], describes programs that yield finite (meaning converging) expected runtimes under all schedulers. This finiteness property is captured by the existence of an upper bound on the series described in Definition 3.9.

*Definition 3.10.* The set PAST contains precisely the pGCL programs that expect to terminate in a finite amount of time under any schedule, i.e.,

$$\mathsf{PAST} \triangleq \left\{P \in \mathtt{Prog} \mid \mathbb{V} f \in \mathbb{F}\ \exists n \in \mathbb{N} \cdot \mathsf{ExpRuntime}(\sigma_{P,0}, f) < n\right\}$$

As with AST, the initial state $\sigma_{P,0}$ maps all variables in Var to 0.

The notion of bounded termination (introduced in Chatterjee and Fu [2017]) is obtained by swapping the positions of the quantifiers in Definition 3.10.

*Definition 3.11.* The set BAST contains precisely the pGCL programs that possess a finite upper bound over the expected runtimes across all schedules, i.e.,

$$\mathsf{BAST} \triangleq \left\{P \in \mathtt{Prog} \mid \exists n \in \mathbb{N}\, \mathbb{V} f \in \mathbb{F} \cdot \mathsf{ExpRuntime}(\sigma_{P,0}, f) \leq n\right\}$$

In the following sections, we study the decision problems AST, PAST, and BAST, which ask: given a pGCL program $P$, is $P \in$ AST (respectively $P \in$ PAST and $P \in$ BAST)? Note that the variants of these problems without nondeterministic choice have already been explored by Kaminski et al. [2019].

### 3.3 Recursion-Theoretic Preliminaries

In order to precisely characterize the complexities of these decision problems, we need to introduce the *arithmetical* and *analytical* hierarchies of undecidability. Informally, these hierarchies describe increasingly undecidable problems by linking each problem to arithmetical formulas in first and second-order logic. We only present relevant definitions here; for a full discussion of the properties of these hierarchies, see Rogers Jr. [1987] and Kozen [2006].

*Definition 3.12 (Arithmetical Hierarchy).* Let $\mathcal{M}_n$ be the set of all total Turing machines characterizing a subset of $\mathbb{N}^n$. For each natural number $n \geq 1$, the family of sets $\Sigma^0_n$ contains the set $L \subseteq \mathbb{N}$ *iff* there exists a machine $M_L \in \mathcal{M}_{n+1}$ such that

$$L = \{x \in \mathbb{N} \mid \exists y_1 \in \mathbb{N}\ \forall y_2 \in \mathbb{N}\ \cdots\ Q_n y_n \in \mathbb{N} \cdot M_L\,(x, y_1, \ldots y_n) = 1\}$$

where $Q_n$ is universal if $n$ is even and existential otherwise. Additionally, define $\Pi^0_n$ as the collection of sets $L \subseteq \mathbb{N}$ such that $(\mathbb{N} \setminus L) \in \Sigma^0_n$.

The collections of sets $\left\{\Sigma^0_n\right\}$ and $\left\{\Pi^0_n\right\}$ form the *Arithmetical Hierarchy* and any set $L \in \Sigma^0_n$ (or $L \in \Pi^0_n$) is said to be *arithmetical*.

*Definition 3.13 (Analytical Hierarchy).* Let $\mathcal{M}^m$ be the set of all total *oracle* Turing machines with access to $m$ oracles, each characterizing a total function of the form $\mathbb{N} \to \mathbb{N}$. For each natural $n \geq 1$, call $\Sigma^1_n$ the collection of sets $L \subseteq \mathbb{N}$ with the property that each $L$ is associated with an $M_L \in \mathcal{M}^n$ and

$$L = \left\{x \in \mathbb{N} \mid \boldsymbol{\exists} f_1 \in \mathbb{N} \to \mathbb{N}\, \mathbb{V} f_2 \in \mathbb{N} \to \mathbb{N} \cdots \mathbf{Q}_n f_n \in \mathbb{N} \to \mathbb{N}\ Q_{n+1} y \in \mathbb{N} \cdot M_L^{f_1, f_2, \ldots f_n}(x, y) = 1\right\}$$

Here, $M_L$ has oracle access to the functions $f_1, \ldots f_n$ and the quantifier $\mathbf{Q}_n$ (and $Q_{n+1}$) is universal (resp. existential) if $n$ is even and existential (resp. universal) otherwise. The doubled symbols $\mathbb{V}$,

$\boldsymbol{\exists}$, and $\mathbf{Q}_n$ are *second-order* quantifiers and the final quantifier $Q_{n+1}$ is first-order. Let $\Pi^1_n$ be the collection of sets $L \subseteq \mathbb{N}$ with $(\mathbb{N} \setminus L) \in \Sigma^1_n$.

The collections $\{\Sigma^1_n\}$ and $\{\Pi^1_n\}$ form the *Analytical Hierarchy*. Any set $L \in \Sigma^1_n$ (or $L \in \Pi^1_n$) is said to be a (lightface) *analytical* set.

The specific classes $\Sigma^1_1$ and $\Pi^1_1$ are referred to as the (lightface) *analytic* and *co-analytic* sets respectively. It can be shown that both the Arithmetical and Analytical hierarchies are strict. Note that Definition 3.13 details a *normal form* for the Analytical hierarchy. In general, there can be arbitrarily many first-order variables after $\mathbf{Q}_n$; sets defined in this way can always be redefined in the normal form [Rogers Jr. 1987]. Notice the implication that the *first levels* of the analytical hierarchy (i.e., $\Sigma^1_1$ and $\Pi^1_1$ sets) contain every arithmetical set.

The strictness of these hierarchies motivates notions of completeness for these complexity classes.

*Definition 3.14 (Completeness).* For any $\Gamma \in \bigcup_{n\in\mathbb{N}} \{\Sigma^1_n, \Pi^1_n, \Sigma^0_n, \Pi^0_n\}$, a set $L \subseteq \mathbb{N}$ is said to be $\Gamma$-*hard* if, for every $L' \in \Gamma$, there exists a recursive procedure that maps $L'$ to $L$ and $\mathbb{N} \setminus L'$ to $\mathbb{N} \setminus L$. Furthermore, $L$ is $\Gamma$-*complete* if it is $\Gamma$-hard and $L \in \Gamma$.

## 4 THE COMPLEXITY OF PROBABILISTIC TERMINATION

Kaminski et al. [2019] showed that the decision problems AST and BAST are arithmetical in a language without nondeterministic choice. Their proof can be extended to the setting with nondeterministic choice:

Proposition 4.1. *The decision problem* AST *is* $\Pi^0_2$*-complete and* BAST *is* $\Sigma^0_2$*-complete.*

We include the proof of Proposition 4.1 in our full version [Majumdar and Sathiyanarayana 2023] for completeness. In contrast, we show that PAST is significantly harder.

Theorem 4.2. *The decision problem* PAST *is* $\Pi^1_1$*-complete.*

*Upper Bound.* Expanding the series defining ExpRuntime in the definition of PAST gives

$$\mathsf{PAST} = \left\{ P \in \mathsf{Prog} \mid \boldsymbol{\forall} f \in \mathbb{F}\ \exists n \in \mathbb{N} \cdot \sum_{k\in\mathbb{N}} \left( 1 - \sum_{\sigma \in T_e^{\leq k}(\sigma_{P,0}, f)} \mathrm{Prob}\,(\sigma) \right) < n \right\}$$

$$\implies \mathsf{PAST} = \left\{ P \in \mathsf{Prog} \mid \boldsymbol{\forall} f \in \mathbb{F}\ \exists n \in \mathbb{N}\ \forall m \in \mathbb{N} \cdot \sum_{k\leq m} \left( 1 - \sum_{\sigma \in T_e^{\leq k}(\sigma_{P,0}, f)} \mathrm{Prob}\,(\sigma) \right) < n \right\} \quad (1)$$

It's quite easy to build a terminating program $M$ with oracle access to $f$ that, on inputs $m$ and $n$, computes the finite sum in the quantifier-free section of Eq. (1). This yields

$$P \in \mathsf{PAST} \iff \boldsymbol{\forall} f \in \mathbb{F}\ \exists n \in \mathbb{N}\ \forall m \in \mathbb{N} \cdot M^f(P, n, m) = 1 \quad (2)$$

Eq. (2) is a characterization of PAST that can be transformed into the normal form for $\Pi^1_1$ (as required by Definition 3.13) using equivalences detailed by Rogers Jr. [1987]. Hence, PAST $\in \Pi^1_1$.

*Lower Bound: Recursion-Theoretic Preliminaries.* To show the $\Pi^1_1$-hardness of PAST, we introduce a canonical $\Pi^1_1$-complete problem. Towards this, we define $\omega$-trees.

*Definition 4.3 ($\omega$-trees, well-founded $\omega$-trees, and recursive $\omega$-trees).* Let $\mathbb{N}^*$ be the set of all finite sequences of natural numbers. Define the prefix relation $\prec_n \subseteq \mathbb{N}^* \times \mathbb{N}^*$ as

$$w_1 \prec_n w_2 \iff |w_1| < |w_2| \land \forall n \leq |w_1| \cdot w_1(n) = w_2(n)$$

```
1  def numGen():
2      x, y, w := 0, 0, 0
3      while (y = 0):
4          x := x + 1
5          y := 0 [] y := 1
6          if (y = 1):
7              break
8          skip ⊕1/2 exit
9          s := 2 * s
10     while (w < s):
11         w := w + 1
12     return x - 1
```

(a) Number generation procedure numGen

```
13 node, s := [], 1 # Globals
14 while (True):
15     x := numGen()
16     node := node.append(x)
17     z := execute(M, node)
18     if (z = 0):
19         n := numGen()
20         while (n--):
21             x := numGen()
22             node := node.append(x)
23         z := execute(M, node)
24         if (z = 1):
25             infLoop()
```

(b) The program $P_M$, calling numGen several times.

Prg. 3. The reduction $P_M$ simulating the recursive $\omega$-tree $M$.

Here, $|w|$ stands for the length of the sequence and $w(n)$ refers to the $n^{th}$ element of $w$.

The pair $(\mathbb{N}^*, \prec_n)$ is the *complete $\omega$-tree*. An *$\omega$-tree* is any subtree of the complete $\omega$-tree rooted at the empty sequence $\varepsilon$. An $\omega$-tree is *well-founded* if there are no infinite branches in the tree.

The characteristic function of an $\omega$-tree takes in sequences $w \in \mathbb{N}^*$ as input and returns 1 when $w$ is a node in the tree and 0 otherwise. An $\omega$-tree is *recursive* if its characteristic function is decidable.

Let $\Omega_{rec}$ be the set of all total Turing machines that characterize *well-founded recursive* $\omega$-trees.

Theorem 4.4. *$\Omega_{rec}$ is $\Pi_1^1$-complete.*

The proof of Theorem 4.4 can be found in various textbooks [Kozen 2006; Rogers Jr. 1987]. We will reduce $\Omega_{rec}$ to PAST.

*Lower Bound: Reduction.* Our reduction leverages nondeterminism in selecting a branch in the complete $\omega$-tree. The remainder of the reduction traverses this branch in the input $\omega$-tree to check its finiteness.

For every Turing machine $M$, we construct the pGCL program $P_M$. The program $P_M$ is detailed in Program 3b. The simulation of $M$ by $P_M$, enabled by the Turing completeness of pGCL [McIver and Morgan 2005], is encapsulated by the function `execute(M, node)`. Here, the finite sequence of natural numbers `node` is supplied to $M$ as input.

$P_M$ invokes a procedure called `numGen` multiple times in its execution. At a high level, `numGen`, specified in Program 3a, makes use of nondeterminism to produce a distribution over natural numbers with the property that every "successful" execution of `numGen` takes, in expectation, a roughly equal amount of time. The first inner loop of `numGen` (from Lines 3 to 9) requires scheduler action at Line 5 to safely exit. Notice that `numGen` terminates execution with probability $1/2$ at every iteration of this loop; hence, the probability of staying inside the loop decreases exponentially the longer the loop is run. The variable `x` tracks the number of iterations of the loop; the output of `numGen` is `x - 1`. The global variable `s` doubles each time the loop is run. Being global, its value persists through multiple executions of `numGen`. The overall design of the reduction ensures that `1/s` tracks the probability value Prob of the current non-terminal execution state.

The second loop (from Lines 10 to 11) of `numGen` induces its principal feature: the stabilization of increments to the expected runtime of the reduction $P_M$ across all executions of `numGen`. It

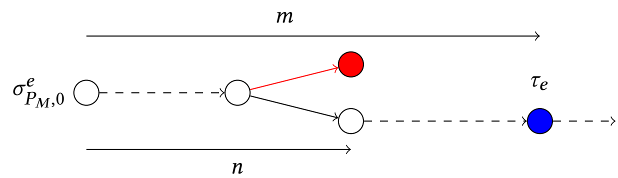


Fig. 3. *An execution from* $\sigma^e_{P_M,0}$. The probabilistic operation at depth $n-1$ yields one terminal and one non-terminal node at depth $n$. Hence, there is at most one non-terminal node at every depth. At depth $m$, the execution reaches $\tau_e$.

isn't difficult to show that the expected runtime increases by at least 1 during each successful (i.e., reaching Line 12 and returning a value) execution of `numGen`.

`numGen` is used by $P_M$ to pick a potential child of the current node in the recursive tree characterized by $M$ (stored by the variable `node`); this is precisely why `numGen` returns `x - 1` at Line 12. Accordingly, the output of `numGen` is appended to the end of `node` at Line 16, and the presence of `node` in the $\omega$-tree is then checked by $M$ at Line 17. If `node` is in the tree, `execute(M, node)` returns 1 at Line 17, and the execution returns to Line 15 and picks another potential child of `node`. Observe that the mandatory singular call to `numGen` in the child-choosing process increases the expected runtime of $P_M$ by at least 1. Consequently, if $M$ were to characterize an infinite branch, $P_M$ could explore this infinite branch and, in the process, make infinitely many calls to `numGen`, pushing its expected runtime to infinity.

In a well-founded tree, $M$ will eventually return 0 at Line 17. Suppose this happens at `node'`. From then on out (i.e., from Line 19), the program checks an edge case: Lines 19 to 22 pick an arbitrary node of the full $\omega$-tree under `node'`, and the `execute` call at Line 23 checks if that node is in the tree validated by $M$. If $M$ characterizes a tree (and not a graph), `execute(M, node)` at Line 23 will always return 0. Note that this check only requires, in expectation, a finite amount of additional time.

We now formally argue for the correctness of the intuitions provided above.

*Case 1: $M$ is not total.* This implies that there is some number $n$ for which $M$ does not halt. Accordingly, take the scheduler $f$ that, on the first execution of `numGen`, exits its inner loop after $n+1$ iterations. The input to $M$ at Line 17 is thus $n$. After reaching that line, $P_M$ runs indefinitely without ever altering its probability value.

Suppose Line 17 is reached at the $m^{th}$ step with probability $p > 0$. For all $m' \geq m$, the probability of termination in $\leq m'$ steps must be bounded above by $1-p$. This is because the probability of non-termination at the $(m')^{th}$ step is $p$. Thus, the expected runtime $\mathsf{ExpRuntime}(\sigma_{P_M,0}, f)$ is

$$\sum_{k\in\mathbb{N}}\left(1-\sum_{\sigma\in T_e^{\leq k}(\sigma_{P,0},f)}\mathsf{Prob}(\sigma)\right)\geq\sum_{m'\in\mathbb{N}^{\geq m}}\left(1-\sum_{\sigma\in T_e^{\leq m'}(\sigma_{P,0},f)}\mathsf{Prob}(\sigma)\right)\geq\sum_{m'\in\mathbb{N}^{\geq m}}p=\infty$$

proving this case.

*Case 2: The scheduler chooses to never leave the first loop of* `numGen`. We label these schedulers as *badly behaved.* Let $f$ be one badly-behaved scheduler. The "bad" behaviour of $f$ can occur after many successful executions of `numGen`. Suppose $P_M$ enters the first loop of `numGen` at Line 3 for the last time in its $m^{th}$ step with probability $p$. This implies an amassed termination probability of $(1-p)$ after $m$ steps.

The design of numGen (specifically, the available options at the probabilistic operation at Line 8) indicates that in every execution tree rooted at the initial state $\sigma^e_{P_M,0}$, there is at most one non-terminal execution state at every depth. Let the execution state at depth $m$ in the tree induced by the scheduler $f$ be $\tau_e$. Let $\tau$ be the program state corresponding to $\tau_e$ and $w_e = \mathsf{Hist}(\tau_e)$. Partitioning the expected runtime series $\mathsf{ExpRuntime}(\sigma_{P_M,0}, f)$ at the $m^{th}$ step gives

$$\mathsf{ExpRuntime}(\sigma_{P_M,0}, f) = \sum_{k\in\mathbb{N}^{<m}} \left(1 - \sum_{\sigma\in T_e^{\leq k}(\sigma_{P_M,0},f)} \mathsf{Prob}(\sigma)\right) + \sum_{k\in\mathbb{N}^{\geq m}} \left(1 - \sum_{\sigma\in T_e^{\leq k}(\sigma_{P_M,0},f)} \mathsf{Prob}(\sigma)\right)$$

The series on the left is finite. The series on the right consists of the probabilities of the non-terminal execution states under $\tau_e$.

Let $f'$ be the scheduler that satisfies $f'(u) = f(w_e u)$ for all histories $u \in (\Sigma_n \cup \Sigma_p)^*$. Let the execution tree from $\tau$ under $f'$ be $T'$ and the subtree of the execution tree from $\sigma_{P_M,0}$ under $f$ rooted at $\tau_e$ be $T$. Then, as far as the program states are concerned, $T$ and $T'$ are identical. This yields a natural mapping $g$ from nodes in $T$ to nodes in $T'$ with the property that $\mathsf{Prob}(\sigma) = p \times \mathsf{Prob}(g(\sigma))$ for every $\sigma \in T$. This means that the second series is just the expected runtime from $\tau$ under $f'$ scaled down by $p$:

$$\sum_{k\in\mathbb{N}^{\geq m}} \left(1 - \sum_{\sigma\in T_e^{\leq k}(\sigma_{P,0},f)} \mathsf{Prob}(\sigma)\right) = p \times \mathsf{ExpRuntime}(\tau, f')$$

Because $f'$ never leaves the inner loop at Line 3, $\mathsf{ExpRuntime}(\tau, f')$ is finite; we omit the details for brevity. Thus, the expected runtime of $P_M$ under badly behaved schedulers is finite.

A well-behaved scheduler is one that is not badly behaved. Well-behaved schedulers always exit numGen with non-zero probability. Each well-behaved $f$ can be identified by the outputs that $f$ induces at executions of numGen, and therefore every well-behaved scheduler corresponds to an infinite branch in the complete $\omega$-tree. From this point on, every machine $M$ is total and every scheduler $f$ is well-behaved.

*Case 3: $M$ fails to characterize a tree.* This means that the subgraph of the complete $\omega$-tree characterized by $M$ is disconnected. This indicates the existence of at least one broken branch, where $M$ returns 1 until depth $m_1$, then returns 0 until depth $m_1 + m_2$, and then returns 1 again at depth $m_1 + m_2 + 1$, for some positive naturals $m_1$ and $m_2$.

Let $f$ be the scheduler corresponding to this broken branch. Under $f$, $P_M$ will merrily execute onward until depth $m_1 + 1$, at which point `execute(M, node)` at Line 17 will return 0. This triggers the instructions under the `if` condition at Line 18, allowing $P_M$ to pick an arbitrary descendant of `node`.

Take the scheduler $f'$ that agrees with $f$ until depth $m_1 + 1$, returns $m_2$ at the numGen call at Line 19, and then picks the node in the broken branch at depth $m_1 + m_2 + 1$ included in the subgraph characterized by $M$ through the loop at Lines 20 to 22. Under $f'$, $M$ will return 1 at the `execute(M, node)` call at Line 23, after which $P_M$ loops infinitely without ever altering its (positive) probability value. It's easy now to see that the expected runtime of $P_M$ under $f'$ is $+\infty$; we leave the details to the diligent reader.

*Case 4: $M$ characterizes a well-founded $\omega$-tree.* This means that every branch in the $\omega$-tree characterized by $M$ is finite. Every well-behaved $f$ thus begets a finite execution tree. Fix a well behaved $f$ and let $m$ be the depth of this finite tree. This means that $T_e^{\leq m}(\sigma_{P_M,0}, f)$ is the set all leaves in the tree. Thus,

$$\sum_{\sigma \in T_e^{\leq m}(\sigma_{P_M,0}, f)} \mathrm{Prob}(\sigma) = 1 \implies \left(1 - \sum_{\sigma \in T_e^{\leq m}(\sigma_{P_M,0}, f)} \mathrm{Prob}(\sigma)\right) = 0$$

Since $m$ is the depth of the tree, for all $m' \geq m$, $T_e^{\leq m'}(\sigma_{P,0}, f) = T_e^{\leq m}(\sigma_{P,0}, f)$. These facts yield

$$\mathrm{ExpRuntime}(\sigma_{P_M,0}, f) = \sum_{k \in \mathbb{N}} \left(1 - \sum_{\sigma \in T_e^{\leq k}(\sigma_{P_M,0}, f)} \mathrm{Prob}(\sigma)\right) = \sum_{k \leq m} \left(1 - \sum_{\sigma \in T_e^{\leq k}(\sigma_{P_M,0}, f)} \mathrm{Prob}(\sigma)\right)$$

This is a finite sum, meaning that the expected runtime is finite.

*Case 5: The $\omega$-tree characterized by $M$ has an infinite branch.* Let $f$ be the scheduler corresponding to this infinite branch. Observe that the execution tree of $P_M$ under $f$ must contain an infinite branch which calls `numGen` infinitely often. Consequently, this branch enters the loop at Line 11 infinitely often.

Isolate one execution of this loop. Suppose the execution enters the loop with probability $p$ in its $m^{th}$ step. Then, the length of the loop is $\mathsf{s} = 1/p$ and the execution exits the loop in its $(m + \mathsf{s})^{th}$ step. Furthermore, the probability of non-termination at each step from $m$ to $(m + \mathsf{s})$ is $p$. This means that

$$\sum_{k=m}^{m+\mathsf{s}} \left(1 - \sum_{\sigma \in T_e^{\leq k}(\sigma_{P,0}, f)} \mathrm{Prob}(\sigma)\right) = p \times \mathsf{s} = p \times (1/p) = 1$$

Hence, the contribution to the expected runtime for $k \in \{m, m+1, \ldots m + \mathsf{s}\}$ is 1. This result holds for all executions of the loop. Every execution of the loop thus corresponds to a constant increase to the expected runtime by 1. Since the loop is executed infinitely often under $f$, the expected runtime under $f$ is $+\infty$.

These five cases show that

$$M \in \Omega_{rec} \iff P_M \in \mathsf{PAST}$$

Hence, PAST is $\Pi_1^1$ hard.

## 5 A PROOF RULE FOR PAST

The reduction proving the $\Pi_1^1$-hardness of PAST (detailed in Section 4) uses the probabilistic choice operator $\oplus$ in a very particular manner. In effect, $\oplus$ is only used to reduce the probability of continued execution. This is realized by supplying $\oplus$ with two options: one immediately terminating program execution and the other continuing it. We use the term *Knievel* to refer to these programs, reflecting the risky choices with terminal consequences made effortlessly by Evel Knievel.

*Definition 5.1 (Knievel form for* pGCL *programs).* A pGCL program $P$ is in *Knievel form* if every instance of the probabilistic choice operator in $P$ is of the form

$$\texttt{skip} \ \oplus_p \ \texttt{exit}$$

for any rational probability value $p$ and some fixed finite step implementation of the statements `skip` and `exit`.

We now propose:

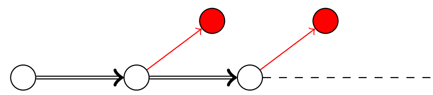

Fig. 4. *An execution tree of a Knievel form program.* The doubled lines represent the potential for multiple intermediate nodes. The mainline artery is depicted as the horizontal branch. Leaving this artery are single terminal nodes.

Proposition 5.2. *There is an effective transformation from any* pGCL *program $P$ into a program $P_K$ in Knievel form such that*

$$P \in \text{PAST} \iff P_K \in \text{PAST}$$

We now provide a brief sketch of the proof for Proposition 5.2. At a high level, the effective Knievel form transformation involves two computable functions. The first is induced by the $\Pi_1^1$-membership of PAST and the $\Pi_1^1$-completeness of $\Omega_{rec}$. By definition of $\Pi_1^1$-hardness, there is a computable function, which we call $f$, that takes pGCL programs $P$ as input and outputs Turing machines $f(P)$ such that $P \in PAST$ *iff* $f(P)$ characterizes a well-founded $\omega$-tree. The second is the program schema we provided in Section 4 (more precisely, in Program 3b) to prove the $\Pi_1^1$-hardness of PAST. More formally, it is a computable function $g$ that takes in Turing machines $M$ as input and produces pGCL programs $g(M)$ such that $M$ characterizes a well-founded $\omega$-tree *iff* $g(M) \in$ PAST. Importantly, $g$ only outputs programs in Knievel form. The effective transformation is the composed function $g \circ f$, which satisfies the following properties: it is computable, its output is a Knievel program, and for any pGCL program $P$, we have $P \in \text{PAST} \iff g(f(P)) \in \text{PAST}$.

Note that we can derive direct constructions for $g \circ f$ from the reductions. We include such a construction in our full version [Majumdar and Sathiyanarayana 2023]; we leave them out here for reasons of space.

Therefore, the Knievel form can be considered to be a kind of *normal form* for PAST programs. Execution trees of these programs have a main arterial branch along which the `skip` option is taken at every probabilistic operation. Branching away from this artery are leaves representing terminal states. See Fig. 4 for an illustration.

In this section, we present a proof rule for proving PAST. We show that this proof rule is sound for programs in Knievel form, and is complete for all PAST programs. Together with the effective transformation to Knievel form, our rule yields a semantically complete proof technique for PAST.

We begin with a few prerequisites.

*Definition 5.3 (Reachable States and Expected Time to Reach).* Let $\sigma = (P, \eta)$ be some program state, and $\sigma_e = (P, \eta, 1, \varepsilon)$ be its initial execution state. The set of states *reachable* from $\sigma$ is

$$\Sigma_r[\sigma] \triangleq \left\{ (P', \eta') \in \Sigma \;\middle|\; \exists f \in \mathbb{F}\; \exists p' \in \mathbb{Q}^+\; \exists w' \in (\Sigma_p \cup \Sigma_n)^* \cdot \sigma_e \vdash_f^* (P', \eta', p', w') \right\}$$

Let $A \subseteq \Sigma_r[\sigma]$ be some subset of states reachable from $\sigma$. Call $A^k_{f,\sigma}$ the subset of execution states belonging to $A$ first reached in $k$ steps under the scheduler $f$. Formally,

$$A^k_{f,\sigma} \triangleq \left\{ (P', \eta', p', w') \in \mathbb{P} \;\middle|\; \begin{array}{l} \exists p' \in \mathbb{Q}^+\; \exists w' \in (\Sigma_n \cup \Sigma_p)^* \cdot (P', \eta') \in A \wedge \sigma_e \vdash_f^k (P', \eta', p', w') \\ \wedge \left( \forall n < k\, \forall \tau \in A^n_{f,\sigma} \cdot \neg(\tau \vdash_f^{k-n} (P', \eta', p', w')) \right) \end{array} \right\}$$

The second line ensures that there are no states belonging to $A$ along the path to the execution states in $A^k_{f,\sigma}$.

Then, the expected time to reach $A$ from $\sigma$ under a scheduler $f$ is given by the series

$$\mathsf{ExpReachRuntime}(\sigma, A, f) \triangleq \sum_{k\in\mathbb{N}} \Pr(\text{not reaching } A \text{ in } k \text{ steps}) \triangleq \sum_{k\in\mathbb{N}} \left(1 - \sum_{i=1}^{k} \sum_{\tau \in A^{i}_{f,\sigma}} \mathsf{Prob}(\tau)\right)$$

In our proof rule, we use the notion of the *Ranking Supermartingale Maps* (RSM-maps). RSM-maps have been proven to be a sound and complete proof technique for BAST by Fu and Chatterjee [2019]. We mildly generalize their notions below.

*Definition 5.4 (RSM-maps).* Let $h : \Sigma \to \mathbb{R}$ be a function from the set of program states to the non-negative reals and $\epsilon > 0$ be an arbitrary real number. The pair $(h, \epsilon)$ is a *Ranking Supermartingale Map* (RSM-map) *iff* $h$ maps terminal states to 0 and satisfies the following properties for every state $\sigma = (P, \eta)$ with $h(\sigma) > 0$:

(1) For deterministic states $\sigma$ with their successors $\sigma' = (P', \eta')$ satisfying the property that

$$\forall\!\!\!\forall f \in \mathbb{F} \cdot (P, \eta, 1, \varepsilon) \vdash_f (P', \eta', 1, \varepsilon)$$

the function $h$ satisfies the following inequality:

$$h(\sigma') + \epsilon \leq h(\sigma)$$

(2) For nondeterministic states $\sigma$ with successors $\sigma_l = (P_l, \eta_l)$ and $\sigma_r = (P_r, \eta_r)$ such that

$$\forall\!\!\!\forall f \in \mathbb{F} \cdot (P, \eta, 1, \varepsilon) \vdash_f (P_l, \eta_l, 1, L_n) \lor (P, \eta, 1, \varepsilon) \vdash_f (P_r, \eta_r, 1, R_n)$$

we have

$$\max(h(\sigma_l), h(\sigma_r)) + \epsilon \leq h(\sigma)$$

(3) For probabilistic states $\sigma$ with the probability value $p$ and successors $\sigma_l = (P_l, \eta_l)$ and $\sigma_r = (P_r, \eta_r)$ such that

$$\forall\!\!\!\forall f \in \mathbb{F} \cdot (P, \eta, 1, \varepsilon) \vdash_f (P_l, \eta_l, p, L_p) \land (P, \eta, 1, \varepsilon) \vdash_f (P_r, \eta_r, 1-p, R_p)$$

we have

$$p \times h(\sigma_l) + (1-p) \times h(\sigma_r) + \epsilon \leq h(\sigma)$$

Note that every program state $\sigma$ is either deterministic, nondeterministic, probabilistic, or terminal.

Unlike Fu and Chatterjee [2019], we do not require RSM-maps to only map terminal states to zero. Our goal is to use them to reason about the expected runtime to reach a collection of states. Towards this, we use the following two lemmas, showing soundness and completeness for BAST, from Fu and Chatterjee [2019]. These are minor modifications of Lemmas 1 (Section 4.1) and 2 (Section 4.2) of Fu and Chatterjee [2019].

Lemma 5.5 (Soundness of RSM-maps). *Let $(h, \epsilon)$ be an RSM-map. Denote by $\Sigma_{tgt}$ the set of states assigned 0 by $h$, i.e.,*

$$\Sigma_{tgt} = \{\sigma \in \Sigma \mid h(\sigma) = 0\}$$

*Then, for all schedulers $f \in \mathbb{F}$ and states $\sigma \in \Sigma$, the expected runtime to reach $\Sigma_{tgt}$ is bounded above:*

$$\forall\!\!\!\forall f \in \mathbb{F}\ \forall \sigma \in \Sigma \cdot \mathsf{ExpReachRuntime}(\sigma, \Sigma_{tgt}, f) \leq \frac{h(\sigma)}{\epsilon}$$

Lemma 5.6 (Completeness of RSM-maps). *Let $\sigma \in \Sigma$ be a program state, $\Sigma_r[\sigma]$ be the set of states reachable from $\sigma$, and $A_{tgt} \subseteq \Sigma_r[\sigma]$ be a target collection of states. Suppose that for all schedulers $f \in \mathbb{F}$, the expected runtime to reach $A_{tgt}$ from $\sigma$ bounded above by some $k \in \mathbb{R}$:*

$$\exists k \in \mathbb{R}\, \forall\!\!\!\forall f \in \mathbb{F} \cdot \mathsf{ExpReachRuntime}(\sigma, A_{tgt}, f) \leq k$$

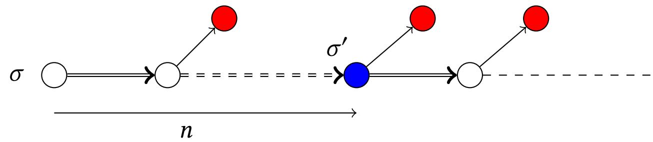


Fig. 5. *Case 2 in the proof of Theorem 5.8.* As earlier, the doubled lines indicate the potential for multiple intermediary nodes. The node $\sigma'$ is marked in blue, and occurs after $n$ steps.

*Then, there must exist an RSM-map $(h_\sigma, 1)$ such that $h_\sigma$ only assigns 0 to states unreachable from $\sigma$ and to states in $A_{tgt}$, i.e.,*

$$h_\sigma(\tau) = 0 \iff \tau \in A_{tgt} \cup (\Sigma \setminus \Sigma_r[\sigma])$$

*Additionally, $h_\sigma(\sigma)$ upper bounds the expected runtime to reach $A_{tgt}$ under any scheduler.*

We now present our proof rule.

*Definition 5.7 (Proof rule for* PAST *programs in Knievel form).* Let $\sigma_0 = (P, \eta_0)$ be an initial program state for the program $P \in \texttt{Prog}$, $\mathbf{o}$ be some ordinal, and $\Sigma_r[\sigma_0]$ be the set of states reachable from $\sigma_{P,0}$. Let $g : \Sigma_r[\sigma_0] \to \mathbf{o}$ and $k : \Sigma_r[\sigma_0] \to ((\Sigma \to \mathbb{R}) \times \mathbb{R})$ be functions that satisfy the following properties

(1) For every $\sigma \in \Sigma_r[\sigma_0]$,

$$g(\sigma) = 0 \iff \sigma = (\bot, \_)$$

In other words, $\sigma$ is terminal *iff* $g(\sigma) = 0$.

(2) For a fixed *non-terminal* state $\sigma \in \Sigma_r[\sigma_0]$, define the set $\mathsf{Lower}_\sigma$ as

$$\mathsf{Lower}_\sigma \triangleq \{\sigma' \in \Sigma_r[\sigma] \mid g(\sigma') < g(\sigma)\}$$

The function $k$ returns an RSM-map $k(\sigma) = (h_\sigma, \epsilon_\sigma)$ that assigns 0 *only* to states *not* reachable from $\sigma$ and to states in $\mathsf{Lower}_\sigma$, i.e.,

$$\tau \in (\mathsf{Lower}_\sigma \cup (\Sigma \setminus \Sigma_r[\sigma])) \iff h_\sigma(\tau) = 0$$

We refer to $g$ and $k$ as the *rank* and *certification* functions respectively.

Notice that $\mathsf{Lower}_\sigma$ is simply the set of states reachable from $\sigma$ that are assigned a lower value by the rank $g$. Applying Lemma 5.5, we see that the RSM-Map $(h_\sigma, \epsilon_\sigma)$ certifies the fact that the expected time to reach $\mathsf{Lower}_\sigma$ from $\sigma$ under every scheduler $f$ is bounded above by a finite value.

## 5.1 Partial Soundness

We now show the soundness of this rule over Knievel form programs.

Theorem 5.8. *Let $P$ be a* pGCL *program in Knievel form, $\sigma_0 = (P, \eta)$ be some initial program state, and $\mathbf{o}$ be some ordinal. Then, if two functions $g : \Sigma_r[\sigma_0] \to \mathbf{o}$ and $k : \Sigma_r[\sigma_0] \to ((\Sigma \to \mathbb{R}) \times \mathbb{R})$ exist that satisfy the properties of the proof rule detailed in Definition 5.7, then $P \in$* PAST.

Proof. We show that from all states $\sigma \in \Sigma_r[\sigma_0]$ and all schedulers $f \in \mathbb{F}$, the expected runtime is finite.

Let $S \subseteq \Sigma_r[\sigma_0]$ be the set of states from which the expected runtime is not finite. Assume the contrary and suppose $S \neq \emptyset$. Order states in $S$ by the values assigned to them by the rank $g$. Since $g$ assigns ordinals under $\mathbf{o}$, the well-ordering principle implies that $S$ has a least element. Denote this least element by $\sigma$.

*Case 1:* $g(\sigma) = 0$. This means $\sigma$ is a terminal state, immediately forming a contradiction.

*Case 2:* $g(\sigma) = x$ for some $0 < x < \mathbf{o}$. Since $x > 0$, the soundness property of the RSM-map $k(\sigma)$ implies that $\mathsf{Lower}_\sigma \neq \emptyset$. Furthermore, the definitions of $S$ and $\sigma$ together imply that $\mathsf{Lower}_\sigma \cap S = \emptyset$. Therefore, for every scheduler $f \in \mathbb{F}$, the expected runtime from every $\sigma' \in \mathsf{Lower}_\sigma$ is finite.

Take an arbitrary scheduler $f$. Denote the RSM-map $k(\sigma)$ by $(h_\sigma, \epsilon_\sigma)$. Combining the properties of $(h_\sigma, \epsilon_\sigma)$ detailed in Definition 5.7 and the results in Lemma 5.5 gives us

$$\mathsf{ExpReachRuntime}(\sigma, \mathsf{Lower}_\sigma, f) \leq \frac{h_\sigma(\sigma)}{\epsilon_\sigma}$$

Take the execution tree corresponding to the scheduler $f$. There are two possibilities.

*Subcase 1: The tree never reaches any $\sigma' \in \mathsf{Lower}_\sigma$ in its main arterial branch.* In this case,

$$\mathsf{ExpReachRuntime}(\sigma, \mathsf{Lower}_\sigma, f) = \mathsf{ExpRuntime}(\sigma, f)$$

This is because the only states from $\mathsf{Lower}_\sigma$ in this tree are the terminal states leaving the arterial branch. The finiteness of the expected runtime follows immediately, forming a contradiction.

*Subcase 2: The tree reaches some $\sigma' \in \mathsf{Lower}_\sigma$ its main arterial branch for the first time after n steps.* See Fig. 5 for an illustration of this case. Call the probability value at the execution state corresponding to $\sigma'$ along the tree $p$.

We now repeat an argument from Section 4. The expected runtime series from $\sigma$ under $f$ can be partitioned at the $n^{th}$ step, i.e., the point at which $\sigma'$ appears in the tree. The actions of the scheduler $f$ in the subtree rooted at $\sigma'$ must correspond to some scheduler $f'$ in an execution initialized at $\sigma'$. Furthermore, the membership of $\sigma' \in \mathsf{Lower}_\sigma$ implies that $\mathsf{ExpRuntime}(\sigma', f')$ is finite. The expected runtime series from $\sigma$ can now be written as

$$\begin{aligned}
\mathsf{ExpRuntime}(\sigma, f) &= \sum_{k \in \mathbb{N}} \Pr(\text{not terminating in } k \text{ steps}) \\
&= \sum_{k \leq n} \Pr(\text{not terminating in } k \text{ steps}) + \sum_{k > n} \Pr(\text{not terminating in } k \text{ steps}) \\
&\leq \frac{h_\sigma(\sigma)}{\epsilon_\sigma} + p \times \mathsf{ExpRuntime}(\sigma', f')
\end{aligned}$$

This series is hence finite, forming a contradiction and completing the proof. □

*Remark 5.9 (Soundness for* AST*).* For every pGCL program $P$, if there exists rank and certification functions $f$ and $g$ that satisfy the properties laid out in Definition 5.7 from the initial state $\sigma_{P,0}$ of $P$, then $P$ is AST. In other words, our proof rule is sound for AST over all pGCL programs, not just those in Knievel form. We leave the proof for this to the reader; it's a simple extension of the proof of Theorem 5.8.

## 5.2 Total Completeness

We now discuss the completeness of the rule detailed in Definition 5.7. Take an arbitrary (i.e., not necessarily Knievel) pGCL program $P$ and its initial state $\sigma_{P,0} = (P, \eta_0)$.

We describe a non-constructive procedure that yields candidates for the rank and certification functions $g$ and $k$. This procedure defines three unbounded sequences: one of partial ranks $\{g_n\}$, one of partial certifications $\{k_n\}$, and one of subsets of reachable states $\{\Sigma_n\}$. Every partial rank $g_n$ maps a subset of values from $\Sigma_r[\sigma_{P,0}]$ to ordinals under the first non-recursive ordinal $\omega_1^{\mathsf{CK}}$. Each partial certificate $k_n$ returns RSM-maps for a subset of $\Sigma_r[\sigma_{P,0}]$, and each $\Sigma_n$ is a subset of $\Sigma_r[\sigma_{P,0}]$. Importantly, the lengths of these sequences can only be measured in ordinals. The domains over which the functions take values grow the further they get from the start.

In parallel to our construction, we will prove the following lemma.

Lemma 5.10. *For every ordinal* $\mathbf{o} < \omega_1^{\mathsf{CK}}$*, let* $g_{\mathbf{o}}$ *be the rank function used in* (Candidate functions) *at the end of the procedure specified in* Section 5.2*. Then, for every state* $\sigma \in \Sigma$*, if* $g_{\mathbf{o}}(\sigma)$ *is defined, then* $g_{\mathbf{o}}(\sigma) \leq \mathbf{o}$.

We begin by first defining $g_0$. Let $\Sigma_0 \subseteq \Sigma_r[\sigma_{P,0}]$ be the set of terminal states reachable from $\sigma_{P,0}$. For all $\sigma \in \Sigma_0$, set $g_0(\sigma)$ to 0. Thus,

$$\Sigma_0 = \left\{(\perp, \eta) \in \Sigma_r[\sigma_{P,0}]\right\} \qquad \text{and} \qquad \forall \sigma \in \Sigma_0 \cdot g_0(\sigma) = 0$$

Since the proof rule only uses RSM-maps for non-terminal states, $k_0$ assigns to each $\sigma \in \Sigma_0$ an arbitrary RSM-map. Observe that Lemma 5.10 trivially holds for the base case of $g_0$.

We now describe a technique to derive the successor rank $g_{\mathbf{o+1}}$ and certificate $k_{\mathbf{o+1}}$ from $g_{\mathbf{o}}$ and $k_{\mathbf{o}}$ for every ordinal $\mathbf{o}$. We begin by requiring $g_{\mathbf{o+1}}$ and $k_{\mathbf{o+1}}$ to agree with $g_{\mathbf{o}}$ and $k_{\mathbf{o}}$ at every state they take values on:

$$\forall \sigma \in \Sigma_{\mathbf{o}} \cdot g_{\mathbf{o+1}}(\sigma) = g_{\mathbf{o}}(\sigma) \wedge k_{\mathbf{o+1}}(\sigma) = k_{\mathbf{o}}(\sigma)$$

We then define the set $\Sigma_{\mathbf{o+1}}$:

$$\Sigma_{\mathbf{o+1}} \triangleq \{\sigma \in \Sigma_r[\sigma_{P,0}] \mid \exists r \in \mathbb{R}\, \mathbb{V} f \in \mathbb{F} \cdot \mathsf{ExpReachRuntime}(\sigma, \Sigma_{\mathbf{o}}, f) \leq r\}$$

Informally, $\Sigma_{\mathbf{o+1}}$ is the set of states from which the expected time to reach $\Sigma_{\mathbf{o}}$ is bounded above by a finite value. For each $\sigma \in \Sigma_{\mathbf{o+1}}$, denote this bound by $r_\sigma$. Observe that, for $A_{tgt} = \Sigma_{\mathbf{o}}$, the bound on the expected time to reach $A_{tgt}$ satisfies the conditions outlined in Lemma 5.6. Hence, there must exist a RSM-map $(h_\sigma, 1)$ with

$$h_\sigma(\tau) = 0 \Longleftrightarrow \tau \in \Sigma_{\mathbf{o}} \cup (\Sigma \setminus \Sigma_r[\sigma])$$

Simply set

$$k_{\mathbf{o+1}}(\sigma) = (h_\sigma, 1)$$

To determine the rank $g_{\mathbf{o+1}}$ of a state $\sigma \in \Sigma_{\mathbf{o+1}}$, we must analyze the subset of $\Sigma_{\mathbf{o}}$ reachable by an execution initialized at $\sigma$. Observe that, by the induction hypothesis of Lemma 5.10,

$$\forall \sigma \in \Sigma_{\mathbf{o}} \cdot g_{\mathbf{o}}(\sigma) \leq \mathbf{o}$$

Hence, the largest measure of any state in $\Sigma_{\mathbf{o}}$ is $\leq \mathbf{o}$. We can hence safely set, for all $\sigma \in \Sigma_{\mathbf{o+1}}$,

$$g_{\mathbf{o+1}}(\sigma) = \mathbf{o} + 1$$

This trivially satisfies the successor induction step in the formal proof of Lemma 5.10.

We now detail the rank $g_{\mathbf{o}}$ and certificate $k_{\mathbf{o}}$ for any *limit ordinal* $\mathbf{o}$. Be begin by defining

$$\Sigma_{\cup} \triangleq \bigcup_{\mathbf{o}' < \mathbf{o}} \Sigma_{\mathbf{o}'}$$

$\Sigma_{\cup}$ is thus the set of states that have been assigned a rank by some $g_{\mathbf{o}'}$. For every $\mathbf{o}' < \mathbf{o}$, set

$$\forall \sigma' \in \Sigma_{\mathbf{o}'} \cdot g_{\mathbf{o}}(\sigma') = g_{\mathbf{o}'}(\sigma') \wedge k_{\mathbf{o}}(\sigma') = k_{\mathbf{o}'}(\sigma')$$

This simply merges the domains of all functions defined for lower ordinals. Now, define $\Sigma_{\mathbf{o}}$ as

$$\Sigma_{\mathbf{o}} \triangleq \{\sigma \in \Sigma_r[\sigma_{P,0}] \mid \exists r \in \mathbb{R}\, \mathbb{V} f \in \mathbb{F} \cdot \mathsf{ExpReachRuntime}(\sigma, \Sigma_{\cup}, f) \leq r\}$$

It's easy to see that $\Sigma_{\mathbf{o}}$ is the set of states from which the runtime for reaching the region of states ranked under $\mathbf{o}$ is bounded. For each $\sigma \in \Sigma_{\mathbf{o}}$, denote this bound by $r_\sigma$. Set $A_{tgt} = \Sigma_{\cup}$ and using $r_\sigma$, apply Lemma 5.6 to derive the RSM-map $(h_\sigma, 1)$ for $A_{tgt}$ and set

$$k_{\mathbf{o}}(\sigma) = (h_\sigma, 1)$$

Similar to the previous case, $g_{\mathbf{o}'}(\sigma') \leq \mathbf{o}'$ for each state $\sigma' \in \Sigma_{\mathbf{o}'}$. Thus, for all $\sigma \in \Sigma_{\mathbf{o}}$, we set

$$g_{\mathbf{o}}(\sigma) = \mathbf{o}$$

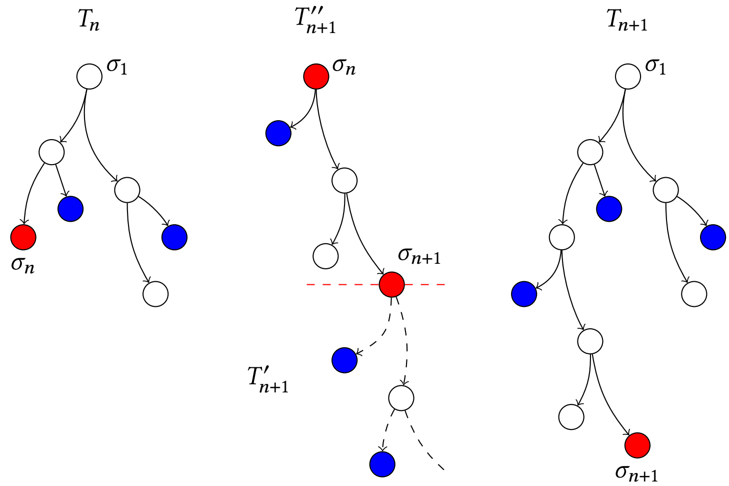


Fig. 6. *The construction of* $T_{n+1}$*.* In each of these trees, the blue nodes represent states belonging to $\Sigma_{good}$. All other nodes belong to $\Sigma_{bad}$. The red nodes are bad leaf nodes selected for extension. The tree $T''_{n+1}$ is produced by removing the subtree rooted at $\sigma_{n+1}$ in $T'_{n+1}$ (also depicted) without diminishing the expected time to reach $\Sigma_{good}$ too much. Observe that $T''_{n+1}$ is merely attached to $\sigma_1$ to produce $T_{n+1}$.

Notice that this completes the proof of Lemma 5.10.

Finally, we define the candidate rank $g : \Sigma_r[\sigma_{P,0}] \longrightarrow \omega_1^{\mathsf{CK}}$ and certificate $k : \Sigma_r[\sigma_{P,0}] \longrightarrow \mathbb{R}$ as

$$g \triangleq \bigcup_{\mathbf{o}<\omega_1^{\mathsf{CK}}} g_{\mathbf{o}} \quad \text{and} \quad k \triangleq \bigcup_{\mathbf{o}<\omega_1^{\mathsf{CK}}} k_{\mathbf{o}} \qquad \text{(Candidate functions)}$$

It's easy to see that, over the domains they're defined, $g$ and $k$ satisfy the requirements detailed in Definition 5.7. We now prove that, for all PAST programs, $g$ and $k$ assign a value to the initial state $\sigma_{P,0}$. In this proof, we mildly abuse our notation and ascribe expected runtimes to execution trees; these are simply the expected runtimes to reach the leaves of the tree from the root of the tree under the scheduler that produces the tree.

Lemma 5.11. *Let $P$ be a* PAST *program and let* $g : \Sigma_r[\sigma_{P,0}] \longrightarrow \omega_1^{\mathsf{CK}}$ *and* $k : \Sigma_r[\sigma_{P,0}] \longrightarrow ((\Sigma \longrightarrow \mathbb{R}) \times \mathbb{R})$ *be the candidate rank and certification functions defined in* (Candidate functions). *Then, $g$ and $k$ are* total.

Proof. We prove this lemma by contradiction. Suppose $g$ and $k$ weren't total. It's easy to see that $g$ and $k$ are always defined over the same collection of states. Define

$$\Sigma_{good} \triangleq \{\sigma \in \Sigma_r[\sigma_{P,0}] \mid \exists \mathbf{o} < \omega_1^{\mathsf{CK}} \cdot g(\sigma) = \mathbf{o}\}$$

In other words, $\Sigma_{good}$ is the collection of states reachable from the initial state $\sigma_{P,0}$ that are assigned a rank by $g$. Define

$$\Sigma_{bad} \triangleq \Sigma_r[\sigma_{P,0}] \setminus \Sigma_{good}$$

Our assumptions indicate that $\Sigma_{bad} \neq \emptyset$. They also indicate that all execution trees rooted at states in $\Sigma_{bad}$ yield finite expected runtimes. We claim that for these states, the expected time to reach $\Sigma_{good}$ is not bounded by a finite value.

Why is this true? Take the set $A_{\sigma_b} \subseteq \Sigma_{good}$ of all good states reachable from some $\sigma_b \in \Sigma_{bad}$. Let $\mathbf{o}_b$ be the smallest ordinal larger than the ranks $g(\sigma_g)$ assigned to every good state $\sigma_g \in A_{\sigma_b}$. It isn't difficult to see that $\mathbf{o}_b$ is recursive, as $g(\sigma_g)$ is recursive for every good state $\sigma_g$ and there are countably many $\sigma_g \in A_{\sigma_b}$. Hence, $g_{\mathbf{o}_b} \subseteq g$ must be defined. If the expected time to reach $A_{\sigma_b}$ was bounded by some $r_{\sigma_b}$, the procedure forces

$$g_{\mathbf{o}_b}(\sigma_b) \leq \mathbf{o}_b$$

This forms a contradiction, justifying the inner claim.

We now construct an infinite sequence of finite execution trees $\{T_n\}$ rooted at some $\sigma_1 \in \Sigma_{bad}$ such that each $T_n$ has at least one bad state from $\Sigma_{bad}$ among its leaves and $T_{n+1}$ extends one of these leaves in $T_n$. Additionally, the expected runtime of each $T_n$ is bounded below by $n$. We then show that there exists a scheduler $f$ that produces the limit $T$ of $\{T_n\}$, and that the expected runtime from $\sigma_1$ under $f$ is $+\infty$.

We begin with $T_1$. Take some state $\sigma_1 \in \Sigma_{bad}$. From our earlier arguments, we know that there must be some execution tree rooted at $\sigma_1$ that yields an expected time to reach $\Sigma_{good}$ at $r'_1$ steps with $r'_1 > 1$. Let $T'_1$ be this tree. The nature of the infinite series defined in Definition 5.3 indicates that there must be a finite subtree of $T'_1$ that still yields a slightly lower expected reachability time $r_1$ with $1 \leq r_1 < r'_1$. Call this finite subtree $T_1$. Observe that there must be at least one bad state $\sigma_2 \in \Sigma_{bad}$ among the leaves of $T_1$; this arises from the strict inequality $r_1 < r'_1$. Furthermore, the expected runtime of $T_1$ is trivially above 1.

We now describe a procedure to build $T_{n+1}$ from $T_n$. Take one bad leaf $\sigma_n \in T_n \cap \Sigma_{bad}$ reached with probability $p_n > 0$. We know that there must be an execution tree rooted at $\sigma_n$ with an expected time to reach $\Sigma_{good}$ of $r'_{n+1} > \frac{1}{p_n}$. Call this tree $T'_{n+1}$, and take the finite subtree $T''_{n+1}$ of $T'_{n+1}$ with an expected time to reach $\Sigma_{good}$ of $r_{n+1}$ with $\frac{1}{p} \leq r_{n+1} < r'_{n+1}$. As before, the strict inequality means that there must be one bad leaf in $T''_{n+1}$. Simply attach $T''_{n+1}$ to the leaf $\sigma_n \in T_n$ to produce $T_{n+1}$. This procedure is illustrated in Fig. 6.

Our construction guarantees that the expected runtime of $T_n$ is at least $n$. The construction of $T_{n+1}$ implies that the expected runtime series of $T_{n+1}$ simply extends that of $T_n$ with the probabilities of non-termination from $T''_{n+1}$. These new probabilities are weighted by $p_n$. Hence,

$$\mathsf{ExpRuntime}(T_{n+1}) = \mathsf{ExpRuntime}(T_n) + p_n \times \mathsf{ExpRuntime}(T''_{n+1}) \geq n + p_n \times \frac{1}{p_n} = n + 1$$

Hence, the expected runtime of $T_{n+1}$ is at least $n+1$, proving the primary property of the construction.

Denote the limit of the sequence $\{T_n\}$ by $T$. Observe that the limit scheduler $f$ of the sequence of schedulers inducing each $T_n$ produces $T$ from $\sigma_1$. Furthermore, the expected runtime of $T$ must be infinite, as its subtrees $T_m \subset T$ ensure that it cannot bounded above by any $m \in \mathbb{N}$. This indicates that the program $P$ is not PAST, forming a contradiction and completing the proof. □

We have thus shown

Theorem 5.12. *For each program $P \in$ PAST, there exist ranking and certification functions $g$ and $k$ that satisfy the requirements of the proof rule detailed in Definition 5.7.*

### 5.3 All the Way to $\omega_1^{\mathsf{CK}}$

We now show, for every recursive ordinal $\mathbf{o} < \omega_1^{\mathsf{CK}}$, a PAST program in Knievel form whose rank has range $\mathbf{o}$. Together with the upper bound in the completeness argument, we conclude that $\omega_1^{\mathsf{CK}}$ is the appropriate range for the rank function $g$.

We begin with inc (see Program 4a), a program for which the smallest rank that can be assigned to its initial state $\sigma_{\mathsf{inc},0}$ is 2. The execution of inc involves a scheduler-directed selection of a power of 2 for the variable x through the loop from Lines 2 to 5. After this selection is made, the program busy waits for x many steps at Lines 6 and 7. The smallest rank that can be ascribed to states at

```
1 x, y := 1, 0
2 while (y = 0):
3     x := 2 * x
4     y := 0 [] y := 1
5     skip ⊕1/2 exit
6 while (x > 0):
7     x := x - 1
```

(a) inc

```
8  node := []
9  while (True):
10     x, y := 0, 1
11     while (y = 0):
12         x := x + 1
13         y := 0 [] y := 1
14         skip ⊕1/2 exit
15     node := node.append(x)
16     M_st := init_M(node)
17     while (not M_st.terminal()):
18         M_st := M_step(M_state)
19         skip ⊕1/2 exit
20     if (M_st.reject()):
21         exit
22     execute(inc)
```

(b) The program $P_M$ using inc

Prg. 4. The full program $P_M$

Line 6 is 1, and since Line 6 can be reached in finitely many steps in expectation, the rank 2 can be assigned to $\sigma_{\text{inc},0}$. Furthermore, because inc $\notin$ BAST, a rank of 1 cannot be ascribed to $\sigma_{\text{inc},0}$.

We now define programs for any recursive ordinal. Lecture 40 of Kozen [2006] describes a mapping between well-founded recursive $\omega$-trees and recursive ordinals. This involves the following finer mapping from the nodes of the $\omega$-trees to recursive ordinals: all leaf nodes are assigned 0 and all internal nodes are assigned the smallest ordinal larger than the values assigned to their immediate children. Finally, the tree is assigned the value of its root. Formally, for every recursive well-founded tree $M \in \Omega_{rec}$, define a function $\text{ord}_M : \mathbb{N}^* \longrightarrow \omega_1^{\text{CK}}$ as

$$\text{ord}_M(w) = \begin{cases} 0 & M(w) = 0 \vee \forall n \in \mathbb{N} \cdot M(\langle w, n \rangle) = 0 \\ \sup_{n \in \mathbb{N}} \text{ord}_M(\langle w, n \rangle) + 1 & \text{otherwise} \end{cases}$$

The first line indicates that $\text{ord}_M$ only maps leaves and nodes not validated by $M$ to 0. Thus, every recursive ordinal $\mathbf{o}$ is associated with some $M \in \Omega_{rec}$ such that $\mathbf{o} = \text{ord}_M(\varepsilon)$.

For every $M \in \Omega_{rec}$, we define a program $P_M$ (see Program 4b) that needs ordinals at least as large as $\text{ord}_M(\varepsilon)$. As in Program 3b, $P_M$ nondeterministically traverses a branch in the tree identified by $M$. Each loop iteration begins with the choice of a candidate child `x` through the inner loop at Line 11. The verification of the candidate child begins at Line 16 and ends at Line 21. The functions `init_M` and `M_step` abstract the initialization and single-step execution of the machine $M$. The structure `M_st` abstracts the current state of the execution of $M$ and provides options for checking whether that state is accepting or rejecting. The insertion of Knievel's risk (continue or terminate) at Line 19 inside the execution of $M$ (Lines 16 to 19) constrains the expected runtime across all children against the running time of $M$. It isn't difficult to show that the expected runtime of each loop iteration from Lines 9 to 20 until the execution of inc at Line 22 is bounded above by a small constant value. *Call this constant value $r_M$.*

The proof for the PAST membership of $P_M$ is similar to the arguments contained in Section 4. We do not repeat them here; instead, we discuss the executions of $P_M$ from program states beginning at the main loop (at Line 9). These program states primarily differ in their values of node, the 'current'

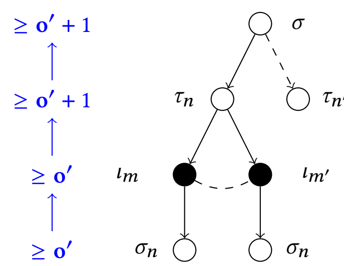


Fig. 7. *The increment mechanism of $P_M$.* The branching in this execution tree is purely nondeterministic; the program can potentially reach any of its children. The minimum rank ascribable to the states is shown in blue. $\sigma$ reaches $\tau_n$ in $r_M$ steps in expectation. From there, for each $m \in \mathbb{N}$, it reaches $\iota_m$, which can each be ascribed rank $\mathbf{o'}$. This causes the minimum possible rank value to increase.

node in the tree recognized by $M$. They are consequently a natural link to the value of $\text{ord}_M(\mathsf{node})$. We show:

Lemma 5.13. *Let $M \in \Omega_{rec}$ be a well-founded recursive tree and $P_M$ be the program corresponding to it in Program 4. Let $S_M$ be the set of program states at Line 9 of Program 4b reachable from the initial state $\sigma_{P_M,0}$. Additionally, let* node : $S_M \to \mathbb{N}^*$ *be a function that maps states in $S_M$ to the value of* node *(i.e., the node) contained in them.*

*Every rank that satisfies the rules detailed in the proof rule (Definition 5.7) must assign to each $\sigma \in S_M$ an ordinal at least as large as $\text{ord}_M(\mathsf{node}(\sigma))$.*

Proof. Observe that from every $\sigma \in S_M$, the execution begins with a scheduler-directed selection of a candidate child x through the loop at Lines 11 - 14. The expected runtime of $P_M$ under a scheduler that never picks a child, or picks a child not in the tree is trivially under $r_M$, the upper bound over the expected runtime of reaching Line 20 from Line 9. The expected runtime under a scheduler that never exits the inc loop at Lines 2 - 5 is also similarly bounded. Hence, we only discuss schedulers picking actual children and actual values at inc.

We prove this lemma by transfinite induction on the value of $\text{ord}_M(\mathsf{node}(\sigma))$.

*Base: $\text{ord}_M(\mathsf{node}(\sigma)) = 0$.* This means $\mathsf{node}(\sigma)$ is a leaf. Therefore, the execution always reaches the terminal state at Line 21, indicating that the expected runtime from $\sigma$ is bounded by $r_M$ under all schedulers. This justifies a rank assignment of 1 to $\sigma$.

*Induction case 1: $\text{ord}_M(\mathsf{node}(\sigma)) = \mathbf{o} + 1$ for some ordinal $\mathbf{o}$.* This implies the existence of a child of $\mathsf{node}(\sigma)$ that was assigned the value $\mathbf{o}$ by $\text{ord}_M$.

Consider the selection of some child $n \in \mathbb{N}$ of $\mathsf{node}(\sigma)$ with $\text{ord}_M(\langle \mathsf{node}(\sigma), n \rangle) = \mathbf{o'}$ and $\mathbf{o'} \leq \mathbf{o}$. Call the program state in $S_M$ corresponding to this new node $\sigma_n$. By the induction hypothesis, the minimum rank that can be ascribed to $\sigma_n$ is $\mathbf{o'}$.

From $\sigma$, the execution tree can select and validate the child $n$ within $r_M$ steps in expectation. After this, the execution enters inc and reaches Line 2 of inc; let $\tau_n$ be the program state at this stage. From $\tau_n$, the execution reaches Line 6 of inc after selecting some $m \in \mathbb{N}$ for the variable x. Call this program state $\iota_m$. From $\iota_m$, the execution reaches $\sigma_n$ in $m$ steps.

We know, from the induction hypothesis, that $\sigma_n$ must be assigned a rank $\geq \mathbf{o'}$. This lower bound on the rank must also apply to $\iota_m$, as all executions from $\iota_m$ deterministically reach $\sigma_n$ in $m$ steps. However, from $\tau_n$, the execution can reach $\iota_m$ for any $m \in \mathbb{N}$. From each $\iota_m$, the expected runtime

```
1 x, y := 0, 0
2 while (y = 0):
3     x := x + 1
4     y := 0 ⊕1/2 y := 1
5     if (y = 1): break
6 y := pow(4, x)
7 while (y > 0): y := y - 1
```

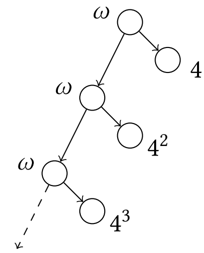


Fig. 8. *The unsoundness example.* (Left) a program that is not PAST; (Right) An execution tree. Each node on the leftmost branch is labelled $\omega$, and each node leaving that branch is labeled by $4^x$. It's easy to see that the expected runtime for this tree is $+\infty$.

for reaching a lower ordinal is bounded below by $m$, an ever increasing quantity. Hence, the rank assigned to $\tau_n$ must at least be $\mathbf{o}' + 1$. Furthermore, because the execution can always expect to reach $\tau_m$ within $r_M$ steps, the state $\sigma$ can be assigned the same rank as $\tau_m$. See Fig. 7 for an illustration.

Now, since there must be some $n \in \mathbb{N}$ such that $\sigma_n$ is ascribed $\mathbf{o}$, the state $\sigma$ must be ascribed a rank of at least $\mathbf{o} + 1$, completing this case.

*Induction case 2: $ord_M(\text{node}(\sigma)) = \mathbf{o}$ for some limit ordinal $\mathbf{o}$.* This is only possible if there are countably many children under $\text{node}(\sigma)$ and for every ordinal $\mathbf{o}' < \mathbf{o}$, there must be some child $n \in \mathbb{N}$ of $\text{node}(\sigma)$ such that $\text{ord}_M(\langle \text{node}(\sigma), n\rangle) > \mathbf{o}'$. Let the program state in $S_M$ corresponding to the node $\langle \text{node}(\sigma), n\rangle$ be $\sigma_n$. Lifting the arguments from the previous case shows that the rank of $\sigma$ must be at least $\mathbf{o}' + 1$ for all $\mathbf{o}' < \mathbf{o}$. This forces the rank of $\sigma$ to be at least $\mathbf{o}$, completing this case, and therefore the proof. □

The initial program state of $P_M$ must thus be assigned a rank of at least $\text{ord}_M(\varepsilon)$, justifying the need for ordinals up to $\omega_1^{\mathsf{CK}}$.

### 5.4 Knievel Form is Necessary

While the rule defined in Definition 5.7 is complete for PAST, it isn't sound for all programs. Take the program in Fig. 8. It is trivial to assign to all program states where the execution remains inside the first loop (at Line 2) the rank $\omega$. We know that the expected runtime bound of 12 (the expected runtime of the first loop) of exiting the loop yields some RSM-map for states inside the loop; simply assign to them this RSM-map. For all states leaving the loop, simply assign to them the value of $y$ and an RSM-map that sets 1 to them and 0 to everything else.

This program is trivially not PAST; however, the rank and certification functions we defined in the previous paragraph satisfy the properties of our proof rule. Thus, our rule must only be applied onto programs in Knievel form to prove their membership in PAST. Nevertheless, our total completeness argument indicates that if one could show that no valid rank and certification functions can exist for a particular pGCL program $P$, then $P \notin$ PAST.

## 6 RELATED WORK

*Termination and Fair Termination.* Termination is a classical problem in computer science, going back to Turing's paper [Turing 1937]. *Ranking functions*, also known as *progress measures*, are a standard technique for proving program termination. Manna [1974] described the use of such functions for demonstrating the termination of deterministic and nondeterministic programs. Their applicability for programs with unbounded nondeterminism has been explored [Chandra 1978; Francez 1986; Harel and Kozen 1984]. The $\Pi_1^1$-completeness of the problem of determining if a program with these features halts is a result by Chandra [1978], and the requirement for ordinals up to $\omega_1^{\mathsf{CK}}$ in these ranking functions was shown by Apt and Plotkin [1986]. Thus, there are recursive procedures transforming positively terminating probabilistic programs with bounded nondeterminism (i.e., PAST programs) to terminating non-probabilistic programs with unbounded nondeterminism, thereby "compiling away" the probabilities in the former.

Harel [1986] showed a general recursive tree transformation that reduced fair termination to termination in the setting of unbounded nondeterminism, thereby providing semantically sound and complete proof rules for fair termination. His reduction also proved the $\Pi_1^1$-completeness for fair termination. We can study fairness in our context, and consider the natural Fair-AST, Fair-PAST, and Fair-BAST sets. These quantify over the set of fair schedulers instead of the set of all schedulers. For a general notion of strong fairness, we can show that Fair-AST, Fair-PAST, and Fair-BAST are all $\Pi_1^1$-hard and are in $\Pi_2^1$—the complexity gap is due to a second, existential second-order quantifier over branches in an infinite tree needed to capture fairness in the probabilistic setting. When we restrict ourselves to the setting of *finitary fairness* [Alur and Henzinger 1998], which replaces the general fairness language with the largest safety language contained within it, we see that Fin-Fair-AST and Fin-Fair-BAST remain $\Pi_2^0$ and $\Sigma_2^0$-complete, and Fin-Fair-PAST remains $\Pi_1^1$-complete. The appropriateness of finitary fairness for probabilistic programs have been argued before [Lengál et al. 2017]. We include proofs for these extensions in our full version [Majumdar and Sathiyanarayana 2023] for completeness.

*Probabilistic Termination.* Termination for probabilistic programs is a well-studied area and trace their provenance to results on infinite-state Markov decision processes. *Ranking supermartingales* are regarded as the probabilistic generalization of ranking functions [Takisaka et al. 2021]. Martingale based techniques have found applications in proving qualitative termination [Avanzini et al. 2020; Bournez and Garnier 2005; Chakarov and Sankaranarayanan 2013; Fioriti and Hermanns 2015; Fu and Chatterjee 2019; Huang et al. 2018]. More recently, they have also been used in proving quantitative termination, where one asks for the probability of termination [Beutner and Ong 2021; Chatterjee et al. 2022, 2017; Takisaka et al. 2021]. Regarding these properties, the use of martingales in the determination of lower and upper bounds on the probability of termination has been shown by Chatterjee et al. [2022, 2017]. Futhermore, Kura et al. [2019] have explored martingale-based approaches toward tail bounds on the expected runtime.

Our work is concerned with the qualitative properties of almost-sure and positive almost-sure termination. Bournez and Garnier [2005] were the first to discuss the use of ranking supermartingales in a sound and complete proof technique for positive almost-sure termination of programs without nondeterminism. The extension of these rules for termination of programs with a global bound on the expected runtime across all schedulers (i.e., BAST programs) have been discussed by Fioriti and Hermanns [2015] and Fu and Chatterjee [2019] with the former only including semi-completeness results and the latter proving completeness. Separately, sound and complete martingale-based proof rules for BAST (called *strong* AST in their paper) have been explored by Avanzini et al. [2020].

Martingales have found applications in the study of almost-sure termination (i.e., AST) as well. A sound proof rule for AST using martingales was described by Chakarov and Sankaranarayanan [2013], and McIver et al. [2018] paired supermartingales with certain intermediary progress functions in a widely applicable sound proof rule for almost-sure termination. Furthermore, algorithms for the synthesis of martingales for interesting subclasses of programs have been explored [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016, 2018].

Proof rules for AST and PAST that operate over the syntax of the programs have been studied [Kaminski et al. 2018; McIver et al. 2018; Olmedo et al. 2016]. The most relevant are the rules that generate bounds on the expected runtime, presented by Kaminski et al. [2018]. Similar rules for recursive programs without loops have been presented by Olmedo et al. [2016]. Additionally, a relatively complete system with the ability to determine AST was introduced by Batz et al. [2021]. Importantly, none of these works include nondeterminism in their program models. Separately, algorithmic analyses of proof rules for AST, PAST, and non-termination have been discussed [Moosbrugger et al. 2021]. Interestingly, we do not know of a "natural" sound and complete proof rule for AST.

Our focus on this paper is purely theoretical. A number of papers have focused on automating the search for termination proofs by fixing a language for expressing ranking supermartingles (e.g., linear or polynomial functions) and then using constraint solving to find appropriate functions [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2016; Colón et al. 2003]. We do not know of many algorithmic heuristics when ranks involve ordinals, even for non-probabilistic programs. Whether our proof rules can be automated in a sound way remains to be seen. One could consider lexicographic ranking functions [Chatterjee et al. 2021; Cook et al. 2013] as a first step, using the standard embedding of a tuple $(a_0, \ldots, a_n)$ to the ordinal sum $a_0\omega^n + \ldots + a_n$.

*Complexity.* Finally, the complexities of AST, PAST, and other related decision problems for probabilistic programs with discrete distributions over their state spaces and without nondeterminism have been discussed in detail by Kaminski et al. [2019]. Their results have been extended by Beutner and Ong [2021] to account for continuous distributions. As far as we know, the complexity analysis for nondeterministic extensions of these problems had not been studied before. For AST and BAST, the extensions are not difficult. Our contribution is to notice the significantly higher complexity of PAST.

## 7 CONCLUSIONS

We have characterized the complexity of PAST for pGCL programs with bounded nondeterministic and probabilistic choice operations. We proved that this problem is $\Pi_1^1$-complete. Using recursion-theoretic insights, we have defined an effectively computable normal form for pGCL, and provided a sound and complete proof rules for PAST for normal form programs. Our proof rule uses ordinals up to $\omega_1^{\mathsf{CK}}$ and this is necessary. A specific implication of our results is that existing techniques based on ranking supermartinagles cannot be complete for PAST.

## ACKNOWLEDGMENTS

We thank the reviewers for their helpful comments. This research was sponsored in part by the Deutsche Forschungsgemeinschaft project 389792660 TRR 248–CPEC (see https://perspicuous-computing.science).

## REFERENCES

Rajeev Alur and Thomas A. Henzinger. 1998. Finitary Fairness. *ACM Trans. Program. Lang. Syst.* 20, 6 (1998), 1171–1194. https://doi.org/10.1145/295656.295659

Krzysztof R. Apt and Gordon D. Plotkin. 1986. Countable nondeterminism and random assignment. *J. ACM* 33, 4 (1986), 724–767. https://doi.org/10.1145/6490.6494

Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. 2020. On probabilistic term rewriting. *Sci. Comput. Program.* 185 (2020). https://doi.org/10.1016/j.scico.2019.102338

Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2021. Relatively complete verification of probabilistic programs: an expressive language for expectation-based reasoning. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–30. https://doi.org/10.1145/3434320

Dmitri P. Bertsekas and Steven E. Shreve. 1978. *Stochastic Optimal Control: The Discrete Time Case.* Academic Press.

Raven Beutner and Luke Ong. 2021. On probabilistic termination of functional programs with continuous distributions. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual*

*Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 1312–1326. https://doi.org/10.1145/3453483.3454111

Olivier Bournez and Florent Garnier. 2005. Proving Positive Almost-Sure Termination. In *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3467)*, Jürgen Giesl (Ed.). Springer, 323–337. https://doi.org/10.1007/978-3-540-32033-3_24

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8044)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

Ashok Chandra. 1978. Computable nondeterministic functions. In *Foundations of Computer Science (FOCS)*. IEEE, 127–131.

Krishnendu Chatterjee and Hongfei Fu. 2017. Termination of Nondeterministic Recursive Probabilistic Programs. arXiv:1701.02944 [cs.PL]

Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2016. Termination Analysis of Probabilistic Programs Through Positivstellensatz's. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9779)*, Swarat Chaudhuri and Azadeh Farzan (Eds.). Springer, 3–22. https://doi.org/10.1007/978-3-319-41528-4_1

Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 40, 2 (2018), 7:1–7:45. https://doi.org/10.1145/3174800

Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Dorde Zikelic. 2022. Sound and Complete Certificates for Quantitative Termination Analysis of Probabilistic Programs. In *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13371)*, Sharon Shoham and Yakir Vizel (Eds.). Springer, 55–78. https://doi.org/10.1007/978-3-031-13185-1_4

Krishnendu Chatterjee, Ehsan Kafshdar Goharshady, Petr Novotný, Jiri Zárevúcky, and Dorde Zikelic. 2021. On Lexicographic Proof Rules for Probabilistic Termination. In *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 13047)*, Marieke Huisman, Corina S. Pasareanu, and Naijun Zhan (Eds.). Springer, 619–639. https://doi.org/10.1007/978-3-030-90870-6_33

Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. 2017. Stochastic invariants for probabilistic termination. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 145–160. https://doi.org/10.1145/3009837.3009873

Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. 2003. Linear Invariant Generation Using Non-linear Constraint Solving. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2725)*, Warren A. Hunt Jr. and Fabio Somenzi (Eds.). Springer, 420–432. https://doi.org/10.1007/978-3-540-45069-6_39

Byron Cook, Abigail See, and Florian Zuleger. 2013. Ramsey vs. Lexicographic Termination Proving. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7795)*, Nir Piterman and Scott A. Smolka (Eds.). Springer, 47–61. https://doi.org/10.1007/978-3-642-36742-7_4

Edsger Wybe Dijkstra. 1976. *A discipline of programming*. Vol. 613924118. prentice-hall Englewood Cliffs.

Luis María Ferrer Fioriti and Holger Hermanns. 2015. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, Sriram K. Rajamani and David Walker (Eds.). ACM, 489–501. https://doi.org/10.1145/2676726.2677001

Nissim Francez. 1986. *Fairness*. Springer. https://doi.org/10.1007/978-1-4612-4886-6

Hongfei Fu and Krishnendu Chatterjee. 2019. Termination of Nondeterministic Probabilistic Programs. In *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11388)*, Constantin Enea and Ruzica Piskac (Eds.). Springer, 468–490. https://doi.org/10.1007/978-3-030-11245-5_22

David Harel. 1986. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. ACM* 33, 1 (1986), 224–248. https://doi.org/10.1145/4904.4993

David Harel and Dexter Kozen. 1984. A programming language for the inductive sets and applications. *Information and Control* 63 (1984), 118–139.

Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. 2018. New Approaches for Almost-Sure Termination of Probabilistic Programs. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11275)*, Sukyoung Ryu (Ed.). Springer, 181–201. https://doi.org/10.1007/978-3-030-02768-1_11

Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2019. On the hardness of analyzing probabilistic programs. *Acta Informatica* 56, 3 (2019), 255–285. https://doi.org/10.1007/s00236-018-0321-1

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 (2018), 30:1–30:68. https://doi.org/10.1145/3208102

Joost-Pieter Katoen, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, and Federico Olmedo. 2015. Understanding Probabilistic Programs. In *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings (Lecture Notes in Computer Science, Vol. 9360)*, Roland Meyer, André Platzer, and Heike Wehrheim (Eds.). Springer, 15–32. https://doi.org/10.1007/978-3-319-23506-6_4

Laurence Kirby and Jeff B. Paris. 1982. Accessible Independence Results for Peano Arithmetic. *Bulletin of The London Mathematical Society* 14 (1982), 285–293.

Dexter Kozen. 2006. *Theory of Computation.* Springer. https://doi.org/10.1007/1-84628-477-5

Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11428)*, Tomás Vojnar and Lijun Zhang (Eds.). Springer, 135–153. https://doi.org/10.1007/978-3-030-17465-1_8

Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer. 2017. Fair Termination for Parameterized Probabilistic Concurrent Systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10205)*, Axel Legay and Tiziana Margaria (Eds.). 499–517. https://doi.org/10.1007/978-3-662-54577-5_29

Rupak Majumdar and V. R. Sathiyanarayana. 2023. Positive Almost-Sure Termination – Complexity and Proof Rules. arXiv:2310.16145 [cs.PL]

Zohar Manna. 1974. *Mathematical Theory of Computation.* McGraw-Hill. https://books.google.de/books?id=D7omAAAAMAAJ

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer. https://doi.org/10.1007/b138392

Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. 2018. A new proof rule for almost-sure termination. *Proc. ACM Program. Lang.* 2, POPL (2018), 33:1–33:28. https://doi.org/10.1145/3158121

Marcel Moosbrugger, Ezio Bartocci, Joost-Pieter Katoen, and Laura Kovács. 2021. Automated Termination Analysis of Polynomial Probabilistic Programs. In *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12648)*, Nobuko Yoshida (Ed.). Springer, 491–518. https://doi.org/10.1007/978-3-030-72019-3_18

Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about Recursive Probabilistic Programs. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 672–681. https://doi.org/10.1145/2933575.2935317

Amir Pnueli. 1983. On the Extremely Fair Treatment of Probabilistic Algorithms. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas (Eds.). ACM, 278–290. https://doi.org/10.1145/800061.808757

George Pólya. 1921. Über eine aufgabe betreffend die irrfahrt im strassennetz. *Math. Ann.* 84 (1921), 149–160.

Hartley Rogers Jr. 1987. *Theory of recursive functions and effective computability (Reprint from 1967).* MIT Press. https://mitpress.mit.edu/9780262680523/theory-of-recursive-functions-and-effective-computability/

Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. 2021. Ranking and Repulsing Supermartingales for Reachability in Randomized Programs. *ACM Trans. Program. Lang. Syst.* 43, 2 (2021), 5:1–5:46. https://doi.org/10.1145/3450967

Alan M. Turing. 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* s2-42, 1 (1937), 230–265. https://doi.org/10.1112/plms/s2-42.1.230

Received 2023-07-11; accepted 2023-11-07