Logical Methods in Computer Science
Volume 16, Issue 4, 2020, pp. 2:1–2:57
https://lmcs.episciences.org/

Submitted Nov. 23, 2019
Published Oct. 02, 2020

ON THE TERMINATION PROBLEM FOR PROBABILISTIC HIGHER-ORDER RECURSIVE PROGRAMS

NAOKI KOBAYASHI [a], UGO DAL LAGO [b], AND CHARLES GRELLOIS [c]

[a] The University of Tokyo, Japan

[b] University of Bologna, Italy

[c] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Abstract. In the last two decades, there has been much progress on model checking of both *probabilistic* systems and *higher-order* programs. In spite of the emergence of higher-order probabilistic programming languages, not much has been done to combine those two approaches. In this paper, we initiate a study on the probabilistic higher-order model checking problem, by giving some first theoretical and experimental results. As a first step towards our goal, we introduce PHORS, a probabilistic extension of higher-order recursion schemes (HORS), as a model of probabilistic higher-order programs. The model of PHORS may alternatively be viewed as a higher-order extension of recursive Markov chains. We then investigate the probabilistic termination problem — or, equivalently, the probabilistic reachability problem. We prove that almost sure termination of order-2 PHORS is undecidable. We also provide a fixpoint characterization of the termination probability of PHORS, and develop a sound (although possibly incomplete) procedure for approximately computing the termination probability. We have implemented the procedure for order-2 PHORS, and confirmed that the procedure works well through preliminary experiments.

1. Introduction

Computer science has interacted with probability theory in many fruitful ways, since the very early days [dLMSS56]. Probability theory enables *state abstraction*, reducing in this way the state space's cardinality. It has also led to a new *model of computation*, used for instance in randomized computation [MR95] or in cryptography [GM84]. The trend of a rise of probability theory's importance in computer science has been followed by the programming language community, up to the point that probabilistic programming is nowadays a very active research area. Probabilistic choice can be modeled in various ways in programming, and fair binary probabilistic choice is for instance perfectly sufficient to obtain universality if the underlying programming language is universal itself [San69, DZ12]. This has been the path followed in probabilistic λ-calculi [Sah78, JP89, DH02, DZ12, ETP14, DSA14, HKSY17].

Key words and phrases: model checking, probabilistic programs, higher-order programs, termination probabilities.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.23638/LMCS-16(4:2)2020

In the present paper, we are interested in the analysis of probabilistic, higher-order recursive programs. Model checking of probabilistic finite state systems has been a very active research field (see [BK08, CHVB18] for a survey). Over the last two decades, there has also been much interest and progress in model checking of probabilistic *recursive* programs [EY09, EY15, BEKK13, BBFK14], which *cannot* be modeled as finite state systems, and thus escape the classic model checking framework and algorithms. None of the proposals in the literature on probabilistic model checking, however, is capable of handling *higher-order* functions, which are a natural feature in functional languages. This is in sharp contrast with what happens for *non-probabilistic* higher-order programs, for which model checking techniques can be fruitfully employed for proving both reachability and safety properties, as shown in the extensive literature on the subject (e.g. [Ong06, HMOS08, Kob13, KO09, KSU11, GM15b, GM15a, TO14, SW11]). There have been some studies on the termination of probabilistic higher-order programs [DLG17], but to our knowledge, they have not provided a procedure for *precisely* computing the termination probability, nor discussed whether it is possible at all: see Section 7 for more details. Summing up, little has been known about the decidability and complexity of model checking of probabilistic higher-order programs, and even less about the existence of practical procedures for approximately solving model checking problems.

One may think that probabilistic *and* higher-order computation is rather an exotic research topic, but it is important for precisely modeling and verifying any higher-order functional programs that interact with a probabilistic environment. As a simple example, consider the following (non-higher-order) OCaml-like program, which uses a primitive `flip` for generating `true` or `false` with probability $\frac{1}{2}$.

```
let rec f() = if flip() then () else f() in f()
```

The program almost surely terminates (i.e., terminates with probability 1), but if we ignore the probabilistic aspects and model `flip()` as a *non-deterministic* (rather than *probabilistic*) primitive, then we would conclude that the program can *may* diverge. The following program makes use of an interesting combination of probabilistic choice and higher-order functions:

```
let boolgen() = flip()
let rec listgen f () =
  if flip() then [] else f()::listgen f ()
in listgen (listgen boolgen) ()
```

The function `listgen` above takes a generator `f` of elements as an argument, and creates a list of elements, each of them obtained by calling f. Thus, the whole program generates a list of lists of Booleans. The length of such a list of lists is randomized, and distributed according to the geometric distribution. We may then wish to ask, for example, (i) whether it almost surely terminates, and (ii) what is the probability that a list of even length is generated. Generating random data structures like the one produced by `listgen` is not an artificial task, being central to, e.g., random test generation [PF17, MRH18].

As a model of probabilistic higher-order programs, we first introduce PHORS, a probabilistic extension of higher-order recursion schemes (HORS) [KNU02, Ong06]. Our model of PHORS is expressive enough to accurately model probabilistic higher-order functions, but the underlying non-probabilistic language (i.e., HORS, obtained by removing probabilistic choice) is *not* Turing-complete; thus, we can hope for the existence of algorithmic solutions to some of the verification problems. As an example, we can decide indeed whether the

termination probability of PHORS is 0, by reduction to a model checking problem for non-probabilistic HORS.

Through the well-known correspondence between HORS and (collapsible) higher-order pushdown automata [KNU02, HMOS08], PHORS can be considered a higher-order extension of probabilistic pushdown systems [BEKK13, BBFK14] and of recursive Markov chains [YE05], the computation models used in previous work on model checking of probabilistic recursive programs. We can also view PHORS as an extension of the λY-calculus [Sta04] with probabilities, just like HORS can be viewed as an alternative presentation of the λY-calculus. The correspondence between HORS and the λY-calculus has been useful for transferring techniques for typed λ-calculi (most notably, game semantics [Ong06], intersection types [Kob09b, KO09] and Krivine machines [SW11]) to HORS; thus, we expect similar benefits in using PHORS (rather than probabilistic higher-order pushdown automata) as models of probabilistic higher-order programs.

As a first step towards understanding the nature of the model checking problem for probabilistic higher-order programs, the present paper studies the problem of computing the termination (or equivalently, reachability) probabilities of PHORS. Note that, as in a non-probabilistic setting, one can easily reduce a safety property verification problem to a may-termination problem (i.e. the problem of checking whether a program may terminate), by encoding safety violation as termination. We can also verify certain liveness properties, by encoding a good event as a termination and checking that the termination probability is 1. As we will see in Section 2, the two questions (i) and (ii) mentioned earlier on the `listgen` program can also be reduced to the problem of computing the termination probability of a PHORS. Note also that computing the termination (or equivalently, reachability) probability has been a key to solving more general model checking problems (such as LTL/CTL model checking) for recursive programs [YE05, BBFK14].

As the first result on the problem of computing termination probabilities, we prove that the almost sure termination problem, i.e., whether a given PHORS terminates with probability 1, is undecidable at order-2 or higher. This contrasts with the case of recursive Markov chains, for which the almost sure termination problem can be decided in PSPACE [EY09]. The proof of undecidability is based on a reduction from the undecidability of Hilbert's tenth problem (i.e. unsolvability of Diophantine equations) [Mat93]. The undecidability result also implies that it is not possible to compute the *exact* termination probability. More precisely, for any rational number $r \in (0, 1]$, the set $\{\mathcal{G} \mid \Pr(\mathcal{G}) \geq r\}$ (where $\Pr(\mathcal{G})$ denotes the termination probability of $\mathcal{G}$) is not recursively enumerable (in other words, the set is Π^0_1-hard in the arithmetical hierarchy). Note, however, that this negative result does not preclude the possibility to compute the termination probability with arbitrary precision; there may exist an algorithm that, given a PHORS $\mathcal{G}$ and $\epsilon > 0$ as inputs, finds r such that the termination probability of $\mathcal{G}$ belongs to $(r, r + \epsilon)$. The existence of such an approximation algorithm remains open.

As a positive result towards approximately computing the termination probability, we show that the termination probability of order-n PHORS can be characterized by fixpoint equations on order-$(n-1)$ functions on real numbers. The fixpoint characterization of the termination probability of recursive Markov chains [EY09] can be viewed as a special case of our result where $n = 1$. The fixpoint characterization immediately provides a semi-algorithm for the lower-bound problem: "Given a PHORS $\mathcal{G}$ and a rational number $r \in [0, 1]$, does $\Pr(\mathcal{G}) > r$ hold?" Recall, however, that $\{\mathcal{G} \mid \Pr(\mathcal{G}) \geq r\}$ is *not* recursively enumerable, so

there is no semi-algorithm for the variation: "Given a PHORS $\mathcal{G}$ and a rational number $r \in [0,1]$, does $\Pr(\mathcal{G}) \geq r$ hold?"

The remaining question is whether an upper-bound on the termination probability can be computed with arbitrary precision. We have not settled this question yet, but propose a procedure for *soundly estimating* an upper-bound of the termination probability of order-2 PHORS by using the fixpoint characterization above, *à la* FEM (finite element method). We have implemented the procedure, and conducted preliminary experiments to confirm that the procedure works fairly well in practice: combined with the lower-bound computation based on the fixpoint characterization, the procedure was able to instantly compute the termination probabilities of (small but) non-trivial examples with precision 10^{-2}. We also briefly discuss how to generalize the procedure to deal with PHORS of arbitrary orders.

The contributions of this article can thus be summarized as follows:

(i) A formalization of probabilistic higher-order recursion schemes (PHORS) and their termination probabilities. This is in Section 2.
(ii) A proof of undecidability of the almost sure termination problem for PHORS (of order 2 or higher), which can be found in Section 3.
(iii) A fixpoint characterization of the termination probability of PHORS, which immediately yields the semi-decidability of the lower-bound problem. This is in Section 4.
(iv) A sound procedure for computing an upper-bound to the termination probability of order-2 PHORS (which is described in Section 5) accompanied by an implementation and preliminary experiments with promising results, reported in Section 6.

We also discuss related work in Section 7, and conclude the article in Section 8. A preliminary summary of this article appeared in Proceedings of LICS 2019 [KDLG19].

2. Probabilistic Higher-Order Recursion Schemes (PHORS) and Termination Probabilities

This section introduces *probabilistic higher-order recursion schemes* (PHORS[1]), an extension of higher-order recursion schemes [KNU02, Ong06] in which programs can at any evaluation step perform a discrete probabilistic choice, then proceeding according to its outcome. Higher-order recursion schemes are usually treated as generators of infinite trees, but as we are only interested in the termination probability, we consider only nullary tree constructors e and Ω, which represent termination and divergence respectively.

We first define types and applicative terms. The set of *types*, ranged over by κ, is given by:

$$\kappa ::= \mathsf{o} \mid \kappa_1 \to \kappa_2.$$

Intuitively, o describes the unit value, and $\kappa_1 \to \kappa_2$ describes functions from κ_1 to κ_2. As usual, the *order* of a type κ is defined by:

$$\mathit{order}(\mathsf{o}) = 0$$
$$\mathit{order}(\kappa_1 \to \kappa_2) = \max(\mathit{order}(\kappa_1) + 1, \mathit{order}(\kappa_2)).$$

We often write $\mathsf{o}^\ell \to \mathsf{o}$ for $\underbrace{\mathsf{o} \to \cdots \to \mathsf{o}}_{\ell} \to \mathsf{o}$, and abbreviate $\kappa_1 \to \cdots \to \kappa_k \to \kappa$ to $\widetilde{\kappa} \to \kappa$.

The set of *applicative terms*, ranged over by t, is given by:

$$t ::= \mathsf{e} \mid \Omega \mid x \mid t_1 t_2,$$

[1]We write PHORS for both singular and plural forms.

where $\mathtt{e}$ and Ω are (the only) constants of type $\mathtt{o}$ and x ranges over a set of variables. Intuitively, $\mathtt{e}$ and Ω denote termination and divergence respectively (the latter can be defined as a derived form, but assuming it as a primitive is convenient for Section 4). We consider the following standard simple type system for applicative terms, where $\mathcal{K}$, called a *type environment*, is a map from a finite set of variables to the set of types.

$$\frac{}{\mathcal{K} \vdash \mathtt{e} : \mathtt{o}} \qquad \frac{}{\mathcal{K} \vdash \Omega : \mathtt{o}} \qquad \frac{\mathcal{K}(x) = \kappa}{\mathcal{K} \vdash x : \kappa} \qquad \frac{\mathcal{K} \vdash t_1 : \kappa_2 \to \kappa \qquad \mathcal{K} \vdash t_2 : \kappa_2}{\mathcal{K} \vdash t_1 t_2 : \kappa}$$

Definition 2.1 (PHORS)**.** *A* probabilistic higher-order recursion scheme *(PHORS) is a triple $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$, where:*

(1) *$\mathcal{N}$ is a map from a finite set of variables (called* non-terminals *and typically denoted F, G, ...) to the set of types.*
(2) *$\mathcal{R}$ is a map from $dom(\mathcal{N})$ to terms of the form $\lambda x_1. \cdots \lambda x_k. t_L \oplus_p t_R$, where $p \in [0,1]$ is a rational number, and t_L, t_R are applicative terms. If $\mathcal{N}(F) = \kappa_1 \to \cdots \to \kappa_k \to \mathtt{o}$, $\mathcal{R}(F)$ must be of the form $\lambda x_1. \cdots \lambda x_k. t_L \oplus_p t_R$, where $\mathcal{N}, x_1 : \kappa_1, \ldots, x_k : \kappa_k \vdash t_L : \mathtt{o}$ and $\mathcal{N}, x_1 : \kappa_1, \ldots, x_k : \kappa_k \vdash t_R : \mathtt{o}$.*
(3) *$S \in dom(\mathcal{N})$, called the* start symbol*, is a distinguished non-terminal that satisfies $\mathcal{N}(S) = \mathtt{o}$.*

The order *of a PHORS $(\mathcal{N}, \mathcal{R}, S)$ is $\max_{F \in dom(\mathcal{N})} order(\mathcal{N}(F))$, i.e., the highest order of the types of its non-terminals. We write $\mathcal{P}_k$ for the set of order-k PHORS.*

When $\mathcal{R}(F) = \lambda x_1. \cdots \lambda x_k. t_L \oplus_p t_R$, we often write $F\, x_1 \cdots x_k = t_L \oplus_p t_R$, and specify $\mathcal{R}$ as a set of such equations. The rule $F\, x_1 \cdots x_k = t_L \oplus_p t_R$ intuitively means that $F\, t_1 \cdots t_k$ is reduced to $[t_1/x_1, \ldots, t_k/x_k]t_L$ and $[t_1/x_1, \ldots, t_k/x_k]t_R$ with probabilities p and $1-p$, respectively. We often write just $F\, x_1 \cdots x_k = t_L$ for $F\, x_1 \cdots x_k = t_L \oplus_1 t_R$.

Definition 2.2 (Operational Semantics and Termination Probability of PHORS)**.** *Given a PHORS $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$, the rewriting relation $\xrightarrow{d,p}_{\mathcal{G}}$ (where $d \in \{L, R\}$ and $p \in [0,1]$) is defined by:*

$$\frac{\mathcal{R}(F) = \lambda x_1. \cdots \lambda x_k. t_L \oplus_p t_R}{F\, t_1 \cdots t_k \xrightarrow{L,p}_{\mathcal{G}} [t_1/x_1, \ldots, t_k/x_k]t_L} \qquad \frac{\mathcal{R}(F) = \lambda x_1. \cdots \lambda x_k. t_L \oplus_p t_R}{F\, t_1 \cdots t_k \xrightarrow{R,1-p}_{\mathcal{G}} [t_1/x_1, \ldots, t_k/x_k]t_R}$$

We write $\xRightarrow{\pi,p}_{\mathcal{G}}$ for the relational composition of $\xrightarrow{d_1,p_1}_{\mathcal{G}}, \ldots, \xrightarrow{d_n,p_n}_{\mathcal{G}}$, when $\pi = d_1 \cdots d_n$ and $p = \prod_{i=1}^{n} p_i$. Note that n may be 0, so that we have $t_1 \xRightarrow{\epsilon,1}_{\mathcal{G}} t_2$ iff $t_1 = t_2$. By definition, for each $\pi \in \{L, R\}^*$, there exists at most one p such that $S \xRightarrow{\pi,p}_{\mathcal{G}} \mathtt{e}$. For an applicative term t, we define $\mathcal{P}(\mathcal{G}, t, \pi)$ by:

$$\mathcal{P}(\mathcal{G}, t, \pi) = \begin{cases} p & \text{if } t \xRightarrow{\pi,p}_{\mathcal{G}} \mathtt{e} \\ 0 & \text{if } t \xRightarrow{\pi,p}_{\mathcal{G}} \mathtt{e} \text{ does not hold for any } p \end{cases}.$$

The partial *and* full termination probabilities*, written $\mathcal{P}(\mathcal{G}, t, n)$ and $\mathcal{P}(\mathcal{G}, t)$, are defined by:*

$$\mathcal{P}(\mathcal{G}, t, n) = \sum_{\pi \in \{L,R\}^{\leq n}} \mathcal{P}(\mathcal{G}, t, \pi) \qquad \text{and} \qquad \mathcal{P}(\mathcal{G}, t) = \sum_{\pi \in \{L,R\}^*} \mathcal{P}(\mathcal{G}, t, \pi).$$

Finally, we set $\mathcal{P}(\mathcal{G}, n) = \mathcal{P}(\mathcal{G}, S, n)$ and $\mathcal{P}(\mathcal{G}) = \mathcal{P}(\mathcal{G}, S)$.

We often omit the subscript $\mathcal{G}$ below and just write $\xrightarrow{d,p}$ and $\xRightarrow{\pi,p}$ for $\xrightarrow{d,p}_{\mathcal{G}}$ and $\xRightarrow{\pi,p}_{\mathcal{G}}$ respectively. The *termination probability of* $\mathcal{G}$ refers to its full termination probability $\mathcal{P}(\mathcal{G})$.

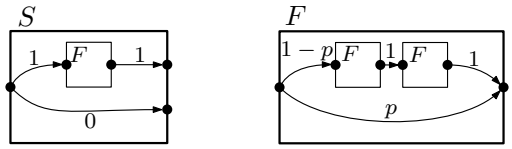

FIGURE 1. A Recursive Markov Chain Modeling $\mathcal{G}_1$.

Example 2.3. Let $\mathcal{G}_1$ be the order-1 PHORS $(\mathcal{N}_1, \mathcal{R}_1, S)$, where:

$$\mathcal{N}_1 = \{S \mapsto \mathsf{o}, F \mapsto \mathsf{o} \to \mathsf{o}\}$$
$$\mathcal{R}_1 = \{S \;=\; F\,\mathsf{e} \oplus_1 \Omega, \quad F\,x \;=\; x \oplus_p F(F\,x)\}.$$

The start symbol S can be reduced, for example, as follows.

$$S \xrightarrow{L,1} F(\mathsf{e}) \xrightarrow{R,1-p} F(F\,\mathsf{e}) \xrightarrow{L,p} F\,\mathsf{e} \xrightarrow{L,p} \mathsf{e}.$$

Thus, we have $S \xrightarrow{LRLL,p^2(1-p)} \mathsf{e}$. As we will see in Section 4, the termination probability $\mathcal{P}(\mathcal{G}_1)$ is the least solution for r of the fixpoint equation: $r = p + (1-p)r^2$. Therefore, $\mathcal{P}(\mathcal{G}_1) = \frac{p}{1-p}$ if $0 \leq p < \frac{1}{2}$ and $\mathcal{P}(\mathcal{G}_1) = 1$ if $\frac{1}{2} \leq p$. The corresponding example of a recursive Markov chain is shown in Figure 1, using the notational conventions from [EY09]. $\mathcal{G}_1$ can be seen as realizing a binary, random walk on the natural numbers, starting from 1. □

As the previous example suggests, there is a mutual translation between recursive Markov chains and order-1 PHORS; see the Appendix A.1 for details.

Example 2.4. Let $\mathcal{G}_2$ be the order-2 PHORS $(\mathcal{N}_2, \mathcal{R}_2, S)$ where:

$$\mathcal{N}_2 = \{S \mapsto \mathsf{o}, H \mapsto \mathsf{o} \to \mathsf{o}, F \mapsto (\mathsf{o} \to \mathsf{o}) \to \mathsf{o},$$
$$D \mapsto (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o}\}$$
$$\mathcal{R}_2 = \{S = \; (F\,H) \oplus_1 \mathsf{e},\; H\,x = x \oplus_{\frac{1}{2}} \Omega,$$
$$F\,g = (g\,\mathsf{e}) \oplus_{\frac{1}{2}} (F(D\,g)),\; D\,g\,x = (g\,(g\,x)) \oplus_1 \Omega\}.$$

The start symbol S can be reduced, for example, as follows.

$$S \xrightarrow{L,1} F\,H \xrightarrow{R,\frac{1}{2}} F(D\,H) \xrightarrow{L,\frac{1}{2}} D\,H\,\mathsf{e} \xrightarrow{L,1} H(H\,\mathsf{e})$$
$$\xrightarrow{L,\frac{1}{2}} H\,\mathsf{e} \xrightarrow{L,\frac{1}{2}} \mathsf{e}.$$

Contrary to $\mathcal{G}_1$, it is quite hard to find an RMC which models the behavior of $\mathcal{G}_2$. In fact, this happens for very good reasons, as we will see in Section 3. □

The following result is obvious from the definition of $\mathcal{P}(\mathcal{G})$.

Theorem 2.5. *For any rational number $r \in [0,1]$, the set $\{\mathcal{G} \mid \mathcal{P}(\mathcal{G}) > r\}$ is recursively enumerable.*

Proof. This follows immediately from the facts that $\mathcal{P}(\mathcal{G}) > r$ if and only if $\mathcal{P}(\mathcal{G}, n) = \sum_{\pi \in \{L,R\}^{\leq n}} \mathcal{P}(\mathcal{G}, S, \pi) > r$ for some n, and that $\mathcal{P}(\mathcal{G}, n)$ is computable. □

In other words, whether $\mathcal{P}(\mathcal{G}) > r$ is semi-decidable, i.e., there exists a procedure that eventually answers "yes" whenever $\mathcal{P}(\mathcal{G}) > r$. As we will see in Section 3, however, for every $r \in (0,1]$, $\{\mathcal{G} \mid \mathcal{P}(\mathcal{G}) \geq r\}$ is *not* recursively enumerable.

Remark 2.6. *Given a PHORS $\mathcal{G}$, replacing each probabilistic operator $\oplus_p$ s.t. $0 < p < 1$ with a binary tree constructor* `br` *and replacing $t_L \oplus_1 t_R$ ($t_L \oplus_0 t_R$, resp.) with t_L (t_R, resp.), we obtain an ordinary HORS $\mathcal{G}^{ND}$. Then $\mathcal{P}(\mathcal{G}) = 0$ if and only if the tree generated by $\mathcal{G}^{ND}$ has no finite path to* `e`*. Thus, by* [KO11] *(see the paragraph below the proof of Theorem 4.5 about the complexity of the reachability problem), whether $\mathcal{P}(\mathcal{G}) = 0$ is decidable, and $(n-1)$-EXPTIME complete. Note, on the other hand, that there is no clear correspondence between the almost sure termination problem $\mathcal{P}(\mathcal{G}) \stackrel{?}{=} 1$ and a model checking problem for $\mathcal{G}^{ND}$. If the tree of $\mathcal{G}^{ND}$ has neither Ω nor an infinite path (which is decidable), then $\mathcal{P}(\mathcal{G})$=1, but the converse does not hold.*

Remark 2.7. *The restriction that a probabilistic choice may occur only at the top-level of each rule is* not *a genuine restriction. Indeed, whenever we wish to write a rule of the form $F\ \widetilde{x} = C[t_L \oplus_p t_R]$, we can normalize it to $F\ \widetilde{x} = C[G\ \widetilde{x}]$, where G is defined by $G\ \widetilde{x} = t_L \oplus_p t_R$. Keeping this in mind, we sometimes allow probabilistic choices to occur inside terms. In fact, a PHORS can be considered as a term (of type* `o`*) of a probabilistic extension of the (call-by-name) λY-calculus* [Sta04]*. We define the set of probabilistic λY terms by:*

$$s ::= \mathtt{e} \mid \Omega \mid x \mid \lambda x.s \mid s_1 s_2 \mid Y(\lambda f.\lambda x.s) \mid s_1 \oplus_p s_2.$$

Here, $\oplus_p$ is a probabilistic choice operator of type $\mathtt{o} \to \mathtt{o} \to \mathtt{o}$, and other terms are simply-typed in the usual way. Then, PHORS and probabilistic λY terms can be converted to each other. We use PHORS in the present paper for the convenience of the fixpoint characterizations discussed in Section 4.

Remark 2.8. *We adopt the call-by-name semantics, and allow probabilistic choices only on terms of type* `o`*. The call-by-value semantics, as well as probabilistic choices at higher-order types can be modeled by applying a standard CPS transformation. Moreover, a PHORS does not have data other than functions, but as in ordinary HORS* [Kob13]*, elements of a finite set (such as Booleans) can be modeled by using Church encoding.*

We provide a few more examples of PHORS below.

Example 2.9. Recall the list generator example in Section 1, whose termination is equivalent to that of the following program, obtained by replacing the output of each function with the unit value `()`.

```
let boolgen() = flip() in
let rec listgen f ()=
  if flip() then () else (f(); listgen f ())
in listgen (listgen boolgen) ()
```

With a kind of CPS transformation, termination of the above program is reduced to that of the following PHORS $\mathcal{G}_3$:

$$S = \mathit{Listgen}\ (\mathit{Listgen}\ \mathit{Boolgen})\ \mathtt{e}$$
$$\mathit{Boolgen}\ k = k$$
$$\mathit{Listgen}\ f\ k = k \oplus_{\frac{1}{2}} f(\mathit{Listgen}\ f\ k)$$

It is not difficult to confirm that $\mathcal{P}(\mathcal{G}_3) = 1$ (using the fixpoint characterization given in Section 4).

Example 2.10. The following is a variation of the list generator example (Example 2.9), which generates ternary trees instead of lists:

```
let boolgen() = flip() in
let rec treegen f =
   if flip() then Leaf
   else Node(f(), treegen f, treegen f, treegen f) in
treegen(boolgen)
```

The following PHORS $\mathcal{G}_4$ captures the termination probability of the aforementioned program:

$$\begin{aligned} S &= \mathit{Treegen}\ \mathit{Boolgen}\ \mathtt{e} \\ \mathit{Boolgen}\ k &= k \\ \mathit{Treegen}\ f\ k &= k \oplus_{\frac{1}{2}} (f(\mathit{Treegen}\, f\, (\mathit{Treegen}\, f\, (\mathit{Treegen}\, f\, k)))) \end{aligned}$$

Interestingly, $\mathcal{G}_4$ is *not* almost surely terminating, since $\mathcal{P}(\mathcal{G}_4) = \frac{\sqrt{5}-1}{2}$.

To increase the chance of termination, let us change the original program as follows:

```
let boolgen() = flip() in
let rec treegen p f =
  if flipp(p) then Leaf
  else Node(f(), treegen (p+1)/2 f, treegen (p+1)/2 f, treegen (p+1)/2 f) in
treegen 1/2 boolgen
```

where `flipp` is the natural generalization of `flip`. Here, `treegen` is parameterized with probability `p`, which is increased upon each recursive call. We assume that `flipp(p)` returns `true` with probability `p` and `false` with $1 - \mathtt{p}$. The corresponding PHORS $\mathcal{G}_5$ is:

$$\begin{aligned} S &= \mathit{Treegen}\ H\ \mathit{Boolgen}\ \mathtt{e} \\ \mathit{Boolgen}\ k &= k \\ H\ x\ y &= x \oplus_{\frac{1}{2}} y \\ G\ p\ x\ y &= x \oplus_{\frac{1}{2}} (p\ x\ y) \\ \mathit{Treegen}\ p\ f\ k &= p\ k\ (f(\mathit{Treegen}\,(G\ p)\, f\, (\mathit{Treegen}\,(G\ p)\, f\, (\mathit{Treegen}\,(G\ p)\, f\, k)))) \end{aligned}$$

The function *Treegen* is parameterized by a probabilistic choice function p, which is initially set to the function H (that chooses the first argument with probability $\frac{1}{2}$). The function G takes a probabilistic choice function p, and returns a probabilistic function $\lambda x.\lambda y.x \oplus_{\frac{1}{2}} (p\ x\ y)$, which chooses the first argument with probability $\frac{\mathtt{p}+1}{2}$ where `p` is the probability that p chooses the first argument. As expected, $\mathcal{G}_5$ is almost surely terminating. □

Example 2.11. Recall the list generator example again. Suppose that we wish to compute the probability that `listgen(boolgen)` generates a list of even length. It can be reduced to the problem of computing the termination probability of the following program:

```
let boolgen() = flip() in
let rec loop() = loop() in
let rec listgenE f () =
  if flip() then () else (f();listgenO f ())
and listgenO f () =
```

```
   if flip() then loop() else (f();listgenE f ()) in
 listgenE boolgen ()
```

Here, we have duplicated `listgen` to `listgenE` and `listgenO`, which are expected to simulate the generation of even and odd lists respectively. Thus, the then-branches of `listgenE` and `listgenO` have been replaced by termination and divergence respectively. As in the previous example, the above program can further be translated to the following PHORS $\mathcal{G}_6$:

$$\begin{aligned} S &= \mathit{ListgenE}\ \mathit{Boolgen}\ \mathtt{e} \\ \mathit{Boolgen}\ k &= k \\ \mathit{ListgenE}\ f\ k &= k \oplus_{\frac{1}{2}} (f(\mathit{ListgenO}\ f\ k)) \\ \mathit{ListgenO}\ f\ k &= \Omega \oplus_{\frac{1}{2}} (f(\mathit{ListgenE}\ f\ k)). \end{aligned}$$

The termination probability of the PHORS is

$$\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{4} + \frac{1}{2} \cdot \left(\frac{1}{4}\right)^2 + \cdots = \frac{1}{2} \cdot \sum_{i=0}^{\infty} \frac{1}{4^i} = \frac{2}{3}.$$

Thus, the probability that the original program generates an even list is also $\frac{2}{3}$.

Let us also consider the problem of computing the probability that `listgen(boolgen)` generates a list containing an even number of `true`'s. It can be reduced to the termination probability of the following PHORS.

$$\begin{aligned} S &= \mathit{ListgenE}\ \mathit{Boolgen}\ \mathtt{e} \\ \mathit{Boolgen}\ k_1\ k_2 &= k_1 \oplus_{\frac{1}{2}} k_2 \\ \mathit{ListgenE}\ f\ k &= k \oplus_{\frac{1}{2}} (f(\mathit{ListgenO}\ f\ k)(\mathit{ListgenE}\ f\ k)) \\ \mathit{ListgenO}\ f\ k &= \Omega \oplus_{\frac{1}{2}} (f(\mathit{ListgenE}\ f\ k)(\mathit{ListgenO}\ f\ k)). \end{aligned}$$

The function *Boolgen* now takes two continuations k_1 and k_2 as arguments, and calls k_1 or k_2 according to whether `true` or `false` is generated in the original program. The function *ListgenE* (*ListgenO*, resp.) is called when the number of `true`'s generated so far is even (odd, resp.). The termination probability of the PHORS above is $\frac{3}{4}$. □

In the following example, a standard program transformation for randomized algorithms is captured as a PHORS. More specifically, a higher-order function is defined, which turns any Las-Vegas algorithm that sometimes declares not to be able to provide the correct answer into one that *always* produces the correct answer. (For more details about the use of the scheme above, please refer to [Hro05]).

Example 2.12. Consider a probabilistic function f, which takes a value of type A, and returns a value of type B with probability p and `Unknown` with probability $1 - p$, where $p > 0$. The following higher-order function `determinize` takes such a function f as an argument, and generates a function from A to B.

```
type 'b pans = Ans of 'b | Unknown
let rec determinize(f:'a->'b pans)(x:'a)=
   match f x with
       Ans(r) -> r
     | Unknown -> determinize f x
```

To confirm that `determinize` f almost surely terminates and returns a value of type B, it suffices to check that the PHORS term $\mathit{Determinize}\ g$ almost surely terminates for $g = \lambda y.\lambda z.y \oplus_p z$, where *Determinize* is defined by:

$$\mathit{Determinize}\ g = g\ \mathtt{e}\ (\mathit{Determinize}\ g).$$

Here, the first argument of g corresponds to the body of the clause `Ans(r)->`$\cdots$, while the second argument corresponds to that of the clause `Unknown->`$\cdots$. Almost sure termination of $\mathit{Determinize}(\lambda y.\lambda z.y \oplus_p z)$ for any $p > 0$ can further be encoded as that of the following PHORS $\mathcal{G}_7$:

$$\begin{aligned}
S &= (\mathit{Determinize}\ \mathit{One}) \oplus_{\frac{1}{2}} (\mathit{ForallP}\ \mathit{Zero}\ \mathit{One})\\
\mathit{One}\ y\ z &= y\\
\mathit{Zero}\ y\ z &= z\\
\mathit{Avg}\ p\ q\ y\ z &= (p\ y\ z) \oplus_{\frac{1}{2}} (q\ y\ z)\\
\mathit{ForallP}\ p\ q &= (\mathit{Determinize}\ (\mathit{Avg}\ p\ q)) \oplus_{\frac{1}{2}}\\
&\qquad ((\mathit{ForallP}\ p\ (\mathit{Avg}\ p\ q)) \oplus_{\frac{1}{2}} (\mathit{ForallP}\ (\mathit{Avg}\ p\ q)\ q))
\end{aligned}$$

It runs $\mathit{Determinize}\ (\lambda y.\lambda z.y \oplus_p z)$ for every p $(0 < p \leq 1)$ of the form $\frac{k}{2^n}$ with non-zero probability. Thus, $\mathcal{P}(\mathcal{G}_7) = 1$ if $\mathit{Determinize}(\lambda y.\lambda z.y \oplus_p z)$ almost surely terminates for every $p > 0$. Conversely, by the continuity of the termination probability of $\mathit{Determinize}\ (\lambda y.\lambda z.y \oplus_p z)$ except at $p = 0$ (which we omit to discuss formally), $\mathcal{P}(\mathcal{G}_7) = 1$ implies that $\mathit{Determinize}(\lambda y.\lambda z.y \oplus_p z)$ almost surely terminates for every $p > 0$. □

Remark 2.13. *Although PHORS do not have probabilities as first-class values, as demonstrated in the examples above, certain operations on probabilities can be realized by encoding a probability p into a probabilistic function $\lambda x.\lambda y.x \oplus_p y$. The function Avg in Example 2.12 realizes the average operation $\frac{p_1+p_2}{2}$. The multiplication p_1p_2 can be represented by $\mathit{Mult}\ p_1\ p_2$, where $\mathit{Mult}\ p_1\ p_2\ x\ y = p_1\ (p_2\ x\ y)\ y$.*

3. Undecidability of Almost Sure Termination of Order-2 PHORS

We prove in this section that the almost sure termination problem, i.e., whether the termination probability $\mathcal{P}(\mathcal{G})$ of a given PHORS $\mathcal{G}$ is 1, is undecidable even for order-2 PHORS. The proof is by reduction from the undecidability of Hilbert's 10th problem [Mat93] (i.e. unsolvability of Diophantine equations). Note that almost sure termination of an order-1 PHORS is decidable, as order-1 PHORS are essentially equi-expressive with probabilistic pushdown systems and recursive Markov chains [EY09, EY15, BEKK13, BBFK14]. In fact, by the fixpoint characterization given in Section 4.3, the termination probability of an order-1 PHORS can be expressed as the least solution of fixpoint equations over reals, which can be solved as discussed in [EY09]. Thus, our undecidability result for order-2 PHORS is optimal.

We start by giving an easy reformulation of the unsolvability of Diophantine equations in terms of polynomials with *non-negative* coefficients, which follows immediately from the original result.

Lemma 3.1. *Given two polynomials $P(x_1, \ldots, x_k)$ and $Q(x_1, \ldots, x_k)$ with non-negative integer coefficients, whether $P(x_1, \ldots, x_k) < Q(x_1, \ldots, x_k)$ for some $x_1, \ldots, x_k \in \mathbf{Nat}$ is undecidable. More precisely, the set of pairs of polynomials: $\{(P(x_1, \ldots, x_k), Q(x_1, \ldots, x_k)) \mid$*

$\exists x_1, \ldots, x_k \in \mathbf{Nat}. P(x_1, \ldots, x_k) < Q(x_1, \ldots, x_k)\}$ *is* Σ_1^0*-complete in the arithmetical hierarchy.*

Proof. Let $D(x_1, \ldots, x_k)$ be a multivariate polynomial with integer coefficients. Then, for all natural numbers $x_1, \ldots, x_k \in \mathbf{Nat}$, $D(x_1, \ldots, x_k) = 0$ if and only if $(D(x_1, \ldots, x_k))^2 - 1 < 0$. Any such polynomial $(D(x_1, \ldots, x_k))^2 - 1$ may be rewritten as $P(x_1, \ldots, x_k) - Q(x_1, \ldots, x_k)$, where $P(x_1, \ldots, x_k)$ and $Q(x_1, \ldots, x_k)$ have only non-negative integer coefficients. Then, $D(x_1, \ldots, x_k) = 0$ if and only if $P(x_1, \ldots, x_k) < Q(x_1, \ldots, x_k)$. Since whether $D(x_1, \ldots, x_k) = 0$ for some $x_1, \ldots, x_k \in \mathbf{Nat}$ is undecidable [Mat93], it is also undecidable whether $P(x_1, \ldots, x_k) < Q(x_1, \ldots, x_k)$ for some $x_1, \ldots, x_k \in \mathbf{Nat}$. Furthermore, since the set of sastisfiable Diophantine equations is Σ_1^0-complete, the set $\{(P(x_1, \ldots, x_k), Q(x_1, \ldots, x_k)) \mid \exists x_1, \ldots, x_k \in \mathbf{Nat}. P(x_1, \ldots, x_k) < Q(x_1, \ldots, x_k)\}$ is Σ_1^0-hard. The set is also obviously recursively enumerable, hence belongs to Σ_1^0. □

Roughly, the idea of our undecidability proof is to show that for every P and Q as above, one can effectively construct an order-2 PHORS that *does not* almost surely terminate if and only if $P(x_1, \ldots, x_k) < Q(x_1, \ldots, x_k)$ *for some* $x_1, \ldots, x_k$. Henceforth, we say t is *non-AST* if t is not almost surely terminating. For ease of understanding, we first construct an order-3 PHORS $\mathcal{G}_3^{P,Q}$ that satisfies the property above in Section 3.1 and then refine the construction to obtain an order-2 PHORS $\mathcal{G}_2^{P,Q}$ with the same property in Section 3.2.

3.1. **Construction of the Order-3 PHORS $\mathcal{G}_3^{P,Q}$.** Let $P(x_1, \ldots, x_k)$ and $Q(x_1, \ldots, x_k)$ be, as above, polynomials with non-negative coefficients. We give the construction of $\mathcal{G}_3^{P,Q}$ in a top-down manner. We let $\mathcal{G}_3^{P,Q}$ enumerate all the tuples of natural numbers $(n_1, \ldots, n_k)$, and for each tuple, spawn a process $Lt\,(P(n_1, \ldots, n_k))\,(Q(n_1, \ldots, n_k))$ with non-zero probability, where $Lt\,m_1\,m_2$ is a process that is non-AST if and only if $m_1 < m_2$. Thus, we define the start symbol S of $\mathcal{G}_3^{P,Q}$ by:

$$\begin{aligned} S &= \mathit{Loop}\ \mathit{Zero}\ \cdots\ \mathit{Zero}. \\ \mathit{Loop}\, x_1\ \cdots\ x_k &= (\mathit{Lt}\,(P\, x_1\ \cdots\ x_k)\,(Q\, x_1\ \cdots\ x_k)) \\ &\qquad \oplus_{\frac{1}{2}}\ (\mathit{Loop}\,(\mathit{Succ}\, x_1)\ \cdots\ x_k)\ \oplus_{\frac{1}{2}}\ \cdots \qquad \oplus_{\frac{1}{2}}\ (\mathit{Loop}\, x_1\ \cdots\ (\mathit{Succ}\, x_k)). \end{aligned}$$

Here, for readability, we have extended the righthand sides of rules to n-ary probabilistic choices:

$$t_1\ \oplus_{p_1}\ t_2\ \oplus_{p_2}\ \cdots\ \oplus_{p_{n-1}}\ t_n.$$

These can be expressed as $t_1\ \oplus_{p_1}\ (F_2\, x_1\ \cdots\ x_k)$, where auxiliary non-terminals are defined by:

$$F_2\, x_1\ \cdots\ x_k = t_2\ \oplus_{p_2}\ (F_3\, x_1\ \cdots\ x_k) \qquad \cdots \qquad F_{n-1}\, x_1\ \cdots\ x_k = t_{n-1}\ \oplus_{p_{n-1}}\ t_n.$$

We can express natural numbers and operations on them by using Church encoding:

$$\begin{aligned} \mathit{Zero}\ s\ z &= z & \mathit{Succ}\ n\ s\ z &= s\ (n\ s\ z) \\ \mathit{Add}\ n\ m\ s\ z &= n\ s\ (m\ s\ z) & \mathit{Mult}\ n\ m\ s\ z &= n\ (m\ s)\ z. \end{aligned}$$

Here, the types of non-terminals above are given by:

$$\begin{aligned} \mathcal{N}(\mathit{Zero}) &= \mathsf{CT} \\ \mathcal{N}(\mathit{Succ}) &= \mathsf{CT} \to \mathsf{CT} \\ \mathcal{N}(\mathit{Add}) = \mathcal{N}(\mathit{Mult}) &= \mathsf{CT} \to \mathsf{CT} \to \mathsf{CT}, \end{aligned}$$

where $\mathsf{CT} = (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o}$ is the usual type of Church numerals. Note that the order of CT is 2, while that of $\mathcal{N}(\mathit{Succ})$, $\mathcal{N}(\mathit{Add})$, and $\mathcal{N}(\mathit{Mult})$ is 3. By using the just introduced operators, we can easily define P and Q as order-3 non-terminals. By abuse of notation, we often use symbols P and Q to denote both polynomials and the representations of them as non-terminals; similarly for natural numbers.

It remains to define an order-3 non-terminal Lt, so that $\mathit{Lt}\, m_1\, m_2$ is non-AST if and only if $m_1 < m_2$. Since $\mathcal{G}_3^{P,Q}$ runs $\mathit{Lt}\,(P\, n_1 \cdots n_k)\,(Q\, n_1 \cdots n_k)$ for each tuple of Church numerals $(n_1, \ldots, n_k)$ with non-zero probability, $\mathcal{G}_3^{P,Q}$ is non-AST if and only if $P(n_1, \ldots, n_k) < Q(n_1, \ldots, n_k)$ for *some* natural numbers $n_1, \ldots, n_k$. The key ingredient used for the construction of Lt is the function *CheckHalf* of type $(\mathsf{o} \to \mathsf{o} \to \mathsf{o}) \to \mathsf{o}$, defined as follows:

$$\mathit{CheckHalf}\ g = F'\ g\ \mathsf{e} \qquad F'\ g\ x = g\ x\ (F'\ g\ (F'\ g\ x)).$$

Here, F' above is a parameterized version of F from Example 2.3: $F'\ \oplus_p$ (where $\oplus_p$ is treated as a function of type $\mathsf{o} \to \mathsf{o} \to \mathsf{o}$, which chooses the first argument with probability p and the second one with $1-p$) corresponds to F. As discussed in Example 2.3, $F\,\mathsf{e}$ is non-AST if and only if $p < \frac{1}{2}$. Thus, $\mathit{CheckHalf}\ g = F'\ g\ \mathsf{e}$ (which is equivalent to $F\,\mathsf{e}$ when $g = \oplus_p$) is non-AST if and only if the probability that g chooses the first argument is smaller than $\frac{1}{2}$. Let *CheckLt* (which will be defined shortly) be a function which takes Church numerals m_1 and m_2, and returns a function of type $\mathsf{o} \to \mathsf{o} \to \mathsf{o}$ that chooses the first argument with probability smaller than $\frac{1}{2}$ if and only if $m_1 < m_2$. Then, Lt can be defined as:

$$\mathit{Lt}\ m_1\ m_2 = \mathit{CheckHalf}(\mathit{CheckLt}\ m_1\ m_2).$$

Finally, *CheckLt* can be defined by:

$$\mathit{CheckLt}\ m_1\ m_2\ x\ y = (\mathit{NatToPr}\ m_1\ x\ y) \oplus_{\frac{1}{2}} (\mathit{NatToPr}\ m_2\ y\ x).$$
$$\mathit{NatToPr}\ m\ x\ y = m\ (H\ x)\ y. \qquad H\ x\ y = x \oplus_{\frac{1}{2}} y.$$

Let us write $[m]$ for the natural number represented by a Church numeral m. For a Church numeral m, $\mathit{NatToPr}\ m\ x\ y$ (which is equivalent to $(H\,x)^{[m]}y$) chooses x with probability $1 - \frac{1}{2^{[m]}}$ and y with probability $\frac{1}{2^{[m]}}$. Thus, the probability that $\mathit{CheckLt}\ m_1\ m_2\ x\ y$ chooses x is

$$\frac{1}{2} \cdot \left(1 - \frac{1}{2^{[m_1]}}\right) + \frac{1}{2} \cdot \frac{1}{2^{[m_2]}} = \frac{1}{2} + \frac{1}{2} \cdot \left(\frac{1}{2^{[m_2]}} - \frac{1}{2^{[m_1]}}\right),$$

which is smaller than $\frac{1}{2}$ if and only if $[m_1] < [m_2]$, as required. This completes the construction of $\mathcal{G}_3^{P,Q}$. See Figure 2 for the whole rules of $\mathcal{G}_3^{P,Q}$. From the discussion above, it should be trivial that $\mathcal{G}_3^{P,Q}$ is non-AST if and only if $P(x_1, \ldots, x_k) < Q(x_1, \ldots, x_k)$ holds for some $x_1, \ldots, x_k \in \mathbf{Nat}$.

3.2. **Decreasing the Order.** We now refine the construction of $\mathcal{G}_3^{P,Q}$ to obtain an order-2 PHORS $\mathcal{G}_2^{P,Q}$ that satisfies the same property. The idea is, instead of passing around a Church numeral m, to pass a probabilistic function equivalent to $\mathit{NatToPr}\, m$, which takes two arguments and chooses the first and second arguments with probabilities $1 - \frac{1}{2^{[m]}}$ and $\frac{1}{2^{[m]}}$, respectively. Note that a Church numeral m has an order-2 type $\mathsf{CT} = (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o}$, whereas $\mathit{NatToPr}\, m$ has an order-1 type $\mathsf{o} \to \mathsf{o} \to \mathsf{o}$. This ultimately allows us to decrease the order of the PHORS.

$$
\begin{aligned}
S &= \mathit{Loop}\ \mathit{Zero}\ \cdots\ \mathit{Zero}.\\
\mathit{Loop}\, x_1\ \cdots\ x_k &= (\mathit{Lt}\,(P\, x_1\ \cdots\ x_k)\,(Q\, x_1\ \cdots\ x_k))\\
&\qquad \oplus_{\frac{1}{2}}\ (\mathit{TestAll}\,(\mathit{Succ}\, x_1)\ \cdots\ x_k)\ \oplus_{\frac{1}{2}}\ \cdots\ \oplus_{\frac{1}{2}}\ (\mathit{TestAll}\, x_1\ \cdots\ (\mathit{Succ}\, x_k)).\\
\mathit{Lt}\ m_1\ m_2 &= \mathit{CheckHalf}\,(\mathit{CheckLt}\ m_1\ m_2).\\
\mathit{CheckHalf}\ y &= F'\ y\ \mathtt{e}.\\
F'\ g\ x &= g\ x\ (F'\ g\,(F'\ g\, x)).\\
\mathit{CheckLt}\ m_1\ m_2\ x\ y &= (\mathit{NatToPr}\ m_1\ x\ y)\ \oplus_{\frac{1}{2}}\ (\mathit{NatToPr}\ m_2\ y\ x).\\
\mathit{NatToPr}\ m\ x\ y &= m\ (H\ x)\ y.\\
H\ x\ y &= x\ \oplus_{\frac{1}{2}}\ y.\\
\mathit{Zero}\, s\, z &= z.\\
\mathit{Succ}\, n\, s\, z &= s\,(n\, s\, z).\\
\mathit{Add}\, n\, m\, s\, z &= n\, s\,(m\, s\, z).\\
\mathit{Mult}\, n\, m\, s\, z &= n\,(m\, s)\, z.\\
P\, x_1\ \cdots\ x_k &= t_P.\\
Q\, x_1\ \cdots\ x_k &= t_Q.
\end{aligned}
$$

FIGURE 2. The rules of $\mathcal{G}_3^{P,Q}$, where t_P and t_Q are terms encoding the polynomials P and Q by way of *Zero*, *Succ*, *Add*, and *Mult*.

Based on the idea above, we replace *Lt* with *LtPr*, which now takes probabilistic functions of type $\mathtt{o} \to \mathtt{o} \to \mathtt{o}$ as arguments:

$$
\begin{aligned}
\mathit{LtPr}\ g_1\ g_2 &= \mathit{CheckHalf}\,(\mathit{CheckLtPr}\ g_1\ g_2).\\
\mathit{CheckLtPr}\ g_1\ g_2\ x\ y &= (g_1\ x\ y)\ \oplus_{\frac{1}{2}}\ (g_2\ y\ x).
\end{aligned}
$$

Here, *CheckLtPr* is an analogous version of *CheckLt*, and *CheckHalf* is as before: *CheckHalf* g is non-AST if and only if the probability that g chooses the first argument is smaller than $\frac{1}{2}$. Then, $\mathit{LtPr}\ (\mathit{NatToPr}\ (P\, n_1\ \cdots\ n_k))\ (\mathit{NatToPr}\ (Q\, n_1\ \cdots\ n_k))$ is non-AST if and only if $P(n_1, \ldots, n_k) < Q(n_1, \ldots, n_k)$.

It remains to modify the top-level loop *Loop*, so that we can enumerate (terms equivalent to) $\mathit{LtPr}\ (\mathit{NatToPr}\ (P\, n_1\ \cdots\ n_k))\ (\mathit{NatToPr}\ (Q\, n_1\ \cdots\ n_k))$ for all $n_1, \ldots, n_k \in \mathbf{Nat}$, without explicitly constructing Church numerals. Instead of using Church encodings, we can encode natural numbers and operations on them (except multiplication) into probabilistic functions as follows.

$$
\begin{aligned}
\mathit{ZeroPr}\ x\ y &= y & \qquad \mathit{SuccPr}\ g\ x\ y &= x\ \oplus_{\frac{1}{2}}\ (g\ x\ y)\\
\mathit{OnePr}\ x\ y &= x\ \oplus_{\frac{1}{2}}\ y & \qquad \mathit{AddPr}\ g_1\ g_2\ x\ y &= g_1\ x\ (g_2\ x\ y).
\end{aligned}
$$

Basically, a natural number m is encoded as a probabilistic function of type $\mathtt{o} \to \mathtt{o} \to \mathtt{o}$, which chooses the first and second arguments with probabilities $1 - \frac{1}{2^m}$ and $\frac{1}{2^m}$ respectively. Notice that $\mathit{AddPr}\ (\mathit{NatToPr}\ m_1)\ (\mathit{NatToPr}\ m_2)$ is equivalent to $\mathit{NatToPr}\ (\mathit{Add}\ m_1\ m_2)$, because the probability that $\mathit{AddPr}\ (\mathit{NatToPr}\ m_1)\ (\mathit{NatToPr}\ m_2)\ x\ y$ chooses y is $\frac{1}{2^{[m_1]}} \cdot \frac{1}{2^{[m_2]}} =$

$\frac{1}{2^{[m_1]+[m_2]}}$. We call this encoding the *probabilistic function encoding*, or *PF encoding* for short.

The multiplication cannot, however, be directly encoded. To compensate for the lack of the multiplication operator, instead of passing around just $n_1, \ldots, n_k$ in the top-level loop, we pass around the PF encodings of the values of $n_1^{i_1} \cdots n_k^{i_k}$ for each $i_1 \leq d_1, \ldots, i_k \leq d_k$, where $d_1, \ldots, d_k$ respectively are the largest degrees of $P(x_1, \ldots, x_k) + Q(x_1, \ldots, x_k)$ in $x_1, \ldots, x_k$. We thus define the start symbol S of $\mathcal{G}_2^{P,Q}$ by:

$$S = \mathit{LoopPr}\ \mathit{OnePr}\ \ \underbrace{\mathit{ZeroPr} \cdots \mathit{ZeroPr}}_{(d_1+1)\cdots(d_k+1)-1 \text{ times}}\ .$$

$$\begin{aligned}\mathit{LoopPr}\ \widetilde{x} = {} & (\mathit{LtPr}\ (P'\ \widetilde{x})\ \ (Q'\ \widetilde{x})) \\ & \oplus_{\frac{1}{2}}\ (\mathit{LoopPr}\ (\mathit{Inc}_{1,(0,\ldots,0)}\ \widetilde{x}) \cdots (\mathit{Inc}_{1,(d_1,\ldots,d_k)}\ \widetilde{x}))\ \oplus_{\frac{1}{2}}\ \cdots \\ & \oplus_{\frac{1}{2}}\ (\mathit{LoopPr}\ (\mathit{Inc}_{k,(0,\ldots,0)}\ \widetilde{x}) \cdots (\mathit{Inc}_{k,(d_1,\ldots,d_k)}\ \widetilde{x})).\end{aligned}$$

Here, $\widetilde{x}$ denotes the sequence of $(d_1+1)\cdots(d_k+1)$ variables $x_{(0,\ldots,0)}, \ldots, x_{(d_1,\ldots,d_k)}$, consisting of $x_{(i_1,\ldots,i_k)}$ for each $i_1 \in \{0, \ldots, d_1\}, \ldots, i_k \in \{0, \ldots, d_k\}$. Each variable $x_{(i_1,\ldots,i_k)}$ holds (the PF encoding of) the value of $n_1^{i_1} \cdots n_k^{i_k}$.

Moreover, the functions P' and Q' are the PF encodings of the polynomials P and Q. Since P and Q can be represented as linear combinations of monomials $x_1^{i_1} \cdots x_k^{i_k}$ for $i_1 \leq d_1, \ldots, i_k \leq d_k$, P' and Q' can be defined using *ZeroPr* and *AddPr*. For example, if $P(x_1, x_2) = x_1^2 + 2x_1x_2$, then P' is defined by: $P'\ \widetilde{x}\ y\ z = \mathit{AddPr}\ x_{(2,0)}\ (\mathit{AddPr}\ x_{(1,1)}\ x_{(1,1)})\ y\ z$.

The function $\mathit{Inc}_{j,(i_1,\ldots,i_k)}\ \widetilde{x}$ represents the PF encoding of $n_1^{i_1} \cdots (n_j+1)^{i_j} \cdots n_k^{i_k}$, assuming that $\widetilde{x}$ represents (the PF encoding of) the values $n_1^0 \cdots n_k^0, \ldots, n_1^{d_1} \cdots n_k^{d_k}$. Note that $\mathit{Inc}_{j,(i_1,\ldots,i_k)}$ can also be defined by using *ZeroPr* and *AddPr*, since $x_1^{i_1} \cdots (x_j + 1)^{i_j} \cdots x_k^{i_k}$ can be expressed as a linear combination of monomials $x_1^0 \cdots x_k^0, \ldots, x_1^{d_1} \cdots x_k^{d_k}$. For example, if $k = 2$, then $\mathit{Inc}_{2,(1,2)}$ can be defined by $\mathit{Inc}_{2,(1,2)}\ \widetilde{x}\ y\ z = \mathit{AddPr}\ x_{(1,2)}\ (\mathit{AddPr}\ x_{(1,1)}\ (\mathit{AddPr}\ x_{(1,1)}\ x_{(1,0)}))\ y\ z$, because $x_1(x_2+1)^2 = x_1x_2^2 + 2x_1x_2 + x_1$. □

This completes the construction of $\mathcal{G}_2^{P,Q}$. See Figure 3 for the list of all rules of $\mathcal{G}_2^{P,Q}$. By the discussion above, we have:

Theorem 3.2. *The almost sure termination of order-2 PHORS is undecidable. More precisely, the set* $\{\mathcal{G} \mid \mathcal{P}(\mathcal{G}) = 1, \mathcal{G}$ *is an order-2 PHORS}* Π_1^0*-hard.*

Proof. By the construction of $\mathcal{G}_2^{P,Q}$ above, $\mathcal{P}(\mathcal{G}_2^{P,Q}) = 1$ if and only if $P(x_1, \ldots, x_k) \geq Q(x_1, \ldots, x_k)$ holds for all $x_1, \ldots, x_k \in \mathbf{Nat}$. By Lemma 3.1, the set of pairs (P, Q) that satisfy the latter is Π_1^0-complete, hence the set $\{\mathcal{G} \mid \mathcal{P}(\mathcal{G}) = 1, \mathcal{G}$ is an order-2 PHORS$\}$ is Π_1^0-hard. □

As a corollary, we also have:

Theorem 3.3. *For any rational number* $r \in (0, 1]$*, the followings are undecidable:*

(1) *whether a given order-2 PHORS* $\mathcal{G}$ *satisfies* $\Pr(\mathcal{G}) \geq r$.
(2) *whether a given order-2 PHORS* $\mathcal{G}$ *satisfies* $\Pr(\mathcal{G}) = r$.

More precisely, the sets $\{\mathcal{G} \in \mathcal{P}_2 \mid \Pr(\mathcal{G}) \geq r\}$ *and* $\{\mathcal{G} \in \mathcal{P}_2 \mid \Pr(\mathcal{G}) = r\}$ *are* Π_1^0*-hard.*

$$
\begin{aligned}
S &= \mathit{LoopPr}\ \mathit{OnePr}\ \underbrace{\mathit{ZeroPr}\ \cdots\ \mathit{ZeroPr}}_{(d_1+1)\cdots(d_k+1)-1}.\\
\mathit{LoopPr}\ \widetilde{x} &= (\mathit{LtPr}\ (P'\ \widetilde{x})\ \ (Q'\ \widetilde{x}))\\
&\quad \oplus_{\frac{1}{2}}\ (\mathit{LoopPr}\,(\mathit{Inc}_{1,(0,\ldots,0)}\,\widetilde{x})\ \cdots\ (\mathit{Inc}_{1,(d_1,\ldots,d_k)}\,\widetilde{x}))\\
&\quad \oplus_{\frac{1}{2}}\ \cdots\ \oplus_{\frac{1}{2}}\ (\mathit{LoopPr}\,(\mathit{Inc}_{k,(0,\ldots,0)}\,\widetilde{x})\ \cdots\ (\mathit{Inc}_{k,(d_1,\ldots,d_k)}\,\widetilde{x})).\\
\mathit{LtPr}\ g_1\ g_2 &= \mathit{CheckHalf}(\mathit{CheckLtPr}\ g_1\ g_2).\\
\mathit{CheckLtPr}\ g_1\ g_2\ x\ y &= (g_1\ x\ y)\ \oplus_{\frac{1}{2}}\ (g_2\ y\ x).\\
\mathit{CheckHalf}\ y &= F'\ y\ \mathsf{e}.\\
F'\ g\ x &= g\ x\ (F'\ g\,(F'\ g\,x)).\\
\mathit{ZeroPr}\ x\ y &= y.\\
\mathit{OnePr}\ x\ y &= x\ \oplus_{\frac{1}{2}}\ y.\\
\mathit{SuccPr}\ g\ x\ y &= x\ \oplus_{\frac{1}{2}}\ (g\ x\ y).\\
\mathit{AddPr}\ g_1\ g_2\ x\ y &= g_1\ x\ (g_2\ x\ y).\\
P'\,\widetilde{x} &= t'_P\\
Q'\,\widetilde{x} &= t'_Q\\
\mathit{Inc}_{j,(i_1,\ldots,i_k)} &= t_{\mathit{Inc}}^{j(i_1,\ldots,i_k)}
\end{aligned}
$$

FIGURE 3. The rules of $\mathcal{G}_2^{P,Q}$, where the terms t'_P, t'_Q and $t_{\mathit{Inc}}^{j,(i_1,\ldots,i_k)}$ are defined based on *ZeroPr*, *OnePr*, and *SuccPr* and *AddPr*.

Proof. Let $\mathcal{G}$ be an order-2 PHORS with the start symbol S. Define $\mathcal{G}'$ as the PHORS obtained by replacing the start symbol with S' and adding the rules $S' = S \oplus_r \Omega$. Then $\Pr(\mathcal{G}') \geq r$ if and only if $\Pr(\mathcal{G}') = r$ if and only if $\Pr(\mathcal{G}) = 1$. Thus, the result follows from Theorem 3.2. □

Remark 3.4. *Let us write $\Psi_{\sim r}$ for the set of order-2 PHORS $\mathcal{G}$ such that $\Pr(\mathcal{G}) \sim r$ where $\sim \in \{<, \leq, =, \geq, >\}$. By Theorem 2.5 and Theorem 3.2, we have:*

(i) *For any rational number $r \in [0,1]$, $\Psi_{>r}$ is recursively enumerable (or, belongs to Σ_1^0).*
(ii) *For any rational number $r \in (0,1]$, $\Psi_{\geq r}$ is Π_1^0-hard (whereas $\Psi_{\geq 0}$ is obviously recursive).*
(iii) *For any rational number $r \in (0,1]$, $\Psi_{=r}$ is Π_1^0-hard (whereas $\Psi_{=0}$ is recursive; recall Remark 2.6).*

It is open whether the following propositions hold or not.

(iv) *$\Psi_{<r}$ is recursively enumerable for every rational number r.*
(v) *$\Psi_{\leq r}$ is recursively enumerable for every rational number r.*
(vi) *There exists an algorithm that takes an order-2 PHORS $\mathcal{G}$ and a rational number $\epsilon > 0$ as inputs, and returns a rational number r such that $|\Pr(\mathcal{G}) - r| < \epsilon$.*

Statements (iv) and (vi) are equivalent. In fact, if (iv) is true, we can construct an algorithm for (vi) as follows. First, test whether $\Pr(\mathcal{G}) = 0$ (which is decidable). If so, output $r = 0$. Otherwise, pick a natural number m such that $\frac{1}{m} < \frac{1}{2}\epsilon$, and divide the interval $(0, 1 + \frac{1}{2}\epsilon)$ to

Table 1. Hardness of the termination problems in terms of the arithmetical hierarchy. For recursive sets (i.e. those in $\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$), more precise computational complexities of the membership problems are given. "PHORS" means order-k PHORS where $k \geq 2$.

Models		$\Psi_{>0}$	$\Psi_{>r}$ $(r \in (0,1))$	$\Psi_{<r}$ $(r \in (0,1])$
RMC		P	PSPACE	PSPACE
PHORS	*Hardness*	$(k-1)$-EXPTIME	$(k-1)$-EXPTIME	Σ_1^0
	Containment	$(k-1)$-EXPTIME	Σ_1^0	Σ_2^0
Turing-complete language		Σ_1^0-complete	Σ_1^0-complete	Σ_2^0-complete

m (overlapping) intervals

$$\left(0, \tfrac{1}{m} + \tfrac{1}{2}\epsilon\right), \left(\tfrac{1}{m}, \tfrac{2}{m} + \tfrac{1}{2}\epsilon\right), \ldots, \left(\tfrac{m-2}{m}, \tfrac{m-1}{m} + \tfrac{1}{2}\epsilon\right), \left(\tfrac{m-1}{m}, 1 + \tfrac{1}{2}\epsilon\right).$$

By using procedures for (i) and (iv), one can enumerate all the order-2 PHORS whose termination probabilities belong to each interval. Thus, $\mathcal{G}$ is eventually enumerated for one of the intervals $(\frac{i}{m}, \frac{i+1}{m} + \frac{1}{2}\epsilon)$; one can then output $\frac{i}{m}$ as r. Conversely, suppose that we have an algorithm for (vi). For each order-2 PHORS $\mathcal{G}$, repeatedly run the algorithm for $\epsilon = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$, and output $\mathcal{G}$ if the output r' for $(\mathcal{G}, \epsilon)$ satisfies $r' + \epsilon < r$. Then, $\mathcal{G}$ is eventually output just if $\Pr(\mathcal{G}) < r$ (note that if $\Pr(\mathcal{G}) < r$, then ϵ eventually becomes smaller than $\frac{1}{2}(r - \Pr(\mathcal{G}))$; at that point, the output r' satisfies $r' + \epsilon < (\Pr(\mathcal{G}) + \epsilon) + \epsilon < r$).

Proposition (v) implies (iv) (and hence also (vi)). If there is a procedure for (v), one can enumerate all the elements of $\Psi_{<r}$ by running the procedure for enumerating $\Psi_{\leq r-\epsilon}$ for $\epsilon = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ □

Remark 3.5. *Table 1 summarizes the hardness of termination problems in terms of the arithmetical hierarchy for recursive Markov chains (RMC), PHORS, and a probabilistic language whose underlying (non-probabilistic) language is Turing-complete. The results for RMC and the Turing-complete language come from* [EY09] *and* [MMKK18]*. As seen in the table, the results on PHORS are not tight, except for the problem $\mathcal{P}(\mathcal{G}) > 0$. Since the expressive power of PHORS is between those of RMC and the Turing complete language, the hardness of each problem is between those of the two models. Theorem 3.3 shows Σ_1^0-hardness of $\mathcal{P}(\mathcal{G}) < r$, but we do not know yet whether the problem is Σ_1^0-complete or Σ_2^0-complete, or lies between the two classes.*

Remark 3.6. *Theorem 3.2 implies that, in contrast to the decidability of LTL model checking of recursive Markov chains* [BEKK13, EY12]*, the corresponding problem for order-2 PHORS (of computing the probability that an infinite transition sequence satisfies a given LTL property) is undecidable and there are even no precise approximation algorithms. Let us extend terms with events:*

$$t ::= \cdots \mid \texttt{event}\ a; t$$

where `event` *$a; t$ raises an event a and evaluates t. Consider the problem of, given an order-2 PHORS $\mathcal{G}$, computing the probability $\mathcal{P}_{a^\omega}(\mathcal{G})$ that a occurs infinitely often. Then there is no algorithm to compute $\mathcal{P}_{a^\omega}(\mathcal{G})$ with arbitrary precision, in the sense of (vi) of Remark 3.4. To see this, notice that by parametric $\mathcal{G}_2^{P,Q}$ with* `e`*, we can define a nonterminal $F : \texttt{o} \to \texttt{o}$ such that $F\ x$ almost surely reduces to x if and only if there exist no $n_1, \ldots, n_k$ such that*

$P(n_1,\ldots,n_k) < Q(n_1,\ldots,n_k)$. Consider the (extended) PHORS $\mathcal{G}^{P,Q,a^\omega}$ whose start symbol S is defined by $S = \texttt{event}\ a; F(S)$. Then

$$\mathcal{P}_{a^\omega}(\mathcal{G}^{P,Q,a^\omega}) = \begin{cases} 0 & \text{if there exists } n_1,\ldots,n_k \text{ such that } P(n_1,\ldots,n_k) < Q(n_1,\ldots,n_k) \\ 1 & \text{otherwise.} \end{cases}$$

Thus, there is no algorithm to approximately compute $\mathcal{P}_{a^\omega}(\mathcal{G})$ even within the precision of $\epsilon = \frac{1}{2}$. □

Remark 3.7. *The PHORS $\mathcal{G}_2^{P,Q}$ obtained above satisfies the so called "safety" restriction* [KNU01, KS15]*. Thus, based on the correspondence between safe grammars and pushdown systems* [KNU01]*, the undecidability result above would also hold for probabilistic second-order pushdown systems (without collapse operations* [HMOS08]*).*

4. Fixpoint Characterization of Termination Probability

Although, as observed in the previous section, there is no general algorithm for exactly computing the termination probability of PHORS, there is still hope that we can *approximately* compute the termination probability. As a possible route towards this goal, this section shows that the termination probability of any PHORS $\mathcal{G}$ can be characterized as the least solution of fixpoint equations on higher-order functions over $[0,1]$. As mentioned in Section 1, the fixpoint characterization immediately yields a procedure for computing lower-bounds of termination probabilities, and also serves as a justification for the method for computing upper-bounds discussed in Section 5. We first introduce higher-order fixpoint equations in Section 4.1. We then characterize the termination probability of an order-n PHORS in terms of fixpoint equations on order-n functions over $[0,1]$ (Section 4.2), and then improve the result by characterizing the same probability in terms of order-$(n-1)$ fixpoint equations for the case $n \geq 1$ (Section 4.3). The latter characterization can be seen as a generalization of the characterization of termination probabilities of recursive Markov chains as polynomial equations [EY09], which served as a key step in the analysis of recursive Markov chains (or probabilistic pushdown systems) [EY09, EY15, BEKK13, BBFK14].

4.1. Higher-order Fixpoint Equations.

We define the syntax and semantics of fixpoint equations that are commonly used in Sections 4.2 and 4.3. We first define the syntax of fixpoint equations.

$$\begin{array}{l} \mathcal{E} \text{ (equations)} ::= \{f_1\,(\widetilde{x}_{1,1}) \cdots (\widetilde{x}_{1,\ell_1}) = e_1, \ldots, f_m\,(\widetilde{x}_{m,1}) \cdots (\widetilde{x}_{m,\ell_m}) = e_m\}; \\ e \text{ (expressions)} ::= r \mid x \mid f \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e_1 e_2 \mid (e_1,\ldots,e_k). \end{array}$$

Here, r ranges over the set of real numbers in $[0,1]$, and $(\widetilde{x})$ represents a tuple of variables $(x_1,\ldots,x_k)$. In the set $\mathcal{E}$ of equations, we require that each function symbol occurs at most once on the lefthand side. The expression $e_1 \cdot e_2$ represents the multiplication of the values of e_1 and e_2, whereas $e_1 e_2$ represents a function application; however, we sometimes omit $\cdot$ when there is no confusion (e.g., we write $0.5x$ for $0.5 \cdot x$). Expressions must be well-typed under the type system given in Figure 4. The *order of a system of fixpoint equations* $\mathcal{E}$ is the largest order of the types of functions in $\mathcal{E}$, where the order of the type $\mathtt{R}$ of reals is 0, and the order of a function type is defined analogously to the order of types for PHORS in Section 2.

$$\tau \text{ (types)} ::= \mathtt{R} \mid \tau_1 \to \tau_2 \mid \tau_1 \times \cdots \times \tau_n.$$

$$\frac{r \in \mathbf{R}}{\Gamma \vdash r : \mathtt{R}} \qquad \frac{\Gamma \vdash e_1 : \mathtt{R} \qquad \Gamma \vdash e_2 : \mathtt{R}}{\Gamma \vdash e_1 + e_2 : \mathtt{R}} \qquad \frac{\Gamma \vdash e_1 : \mathtt{R} \qquad \Gamma \vdash e_2 : \mathtt{R}}{\Gamma \vdash e_1 \cdot e_2 : \mathtt{R}}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \to \tau \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau} \qquad \frac{\Gamma \vdash e_i : \tau_i \text{ for each } i \in \{1, \ldots, k\}}{\Gamma \vdash (e_1, \ldots, e_k) : \tau_1 \times \cdots \times \tau_k}$$

$$\frac{\Gamma, (\widetilde{x}_{i,1}) : \tau_{i,1}, \ldots, (\widetilde{x}_{i,\ell_i}) : \tau_{i,\ell_i i} \vdash e_i \quad \Gamma(f_i) = \tau_{i,1} \to \cdots \to \tau_{i,\ell_i} \to \mathtt{R} \text{ (for each } i \in \{1, \ldots, m\})}{\Gamma \vdash \{f_i\,(\widetilde{x}_{i,1}) \cdots (\widetilde{x}_{i,\ell_i}) = e_i \mid i \in \{1, \ldots, m\}\}}$$

FIGURE 4. Type system for fixpoint equations, where $(x_1, \ldots, x_k) : \tau$ denotes $x_1 : \tau_1, \ldots, x_k : \tau_k$ whenever $\tau = \tau_1 \times \cdots \times \tau_k$.

Example 4.1. The following is a system of order-2 fixpoint equations:

$$\{f_1 = f_2\ f_3\ (0.5, 0.5), f_2\ g\ (x_1, x_2) = g(x_1 + x_2), f_3\, x = 0.3\, x + 0.7 f_3(f_3\, x)\}.$$

It is well-typed under $f_1 : \mathtt{R}, f_2 : (\mathtt{R} \to \mathtt{R}) \to (\mathtt{R} \times \mathtt{R}) \to \mathtt{R}, f_3 : \mathtt{R} \to \mathtt{R}$. □

The semantics of fixpoint equations is defined in an obvious manner. Let $\mathbf{R}_\infty$ be the set consisting of non-negative real numbers and ∞. We extend addition and multiplication by: $x + \infty = \infty + x = \infty$, $0 \cdot \infty = \infty \cdot 0 = 0$, and $x \cdot \infty = \infty \cdot x = \infty$ if $x \neq 0$. Note that $(\mathbf{R}_\infty, \leq, 0)$ forms an ω-cpo, where $\leq$ is the extension of the usual inequality on reals with $x \leq \infty$ for every $x \in \mathbf{R}_\infty$. For each type τ, we interpret τ as the cpo $[\![\tau]\!] = (X_\tau, \sqsubseteq_\tau, \bot_\tau)$, defined by induction on τ:

$$\begin{aligned}
X_{\mathtt{R}} &= \mathbf{R}_\infty \\
\sqsubseteq_{\mathtt{R}} &= \leq \\
\bot_{\mathtt{R}} &= 0 \\
X_{\tau_1 \to \tau_2} &= \{f \in X_{\tau_1} \to X_{\tau_2} \mid \ f \text{ is monotonic and } \omega\text{-continuous}\} \\
\sqsubseteq_{\tau_1 \to \tau_2} &= \{(f_1, f_2) \in X_{\tau_1 \to \tau_2} \times X_{\tau_1 \to \tau_2} \mid \forall x \in X_{\tau_1}. f_1(x) \sqsubseteq_{\tau_2} f_2(x)\} \\
\bot_{\tau_1 \to \tau_2} &= \lambda\!\!\lambda\, x \in X_{\tau_1}. \bot_{\tau_2} \\
X_{\tau_1 \times \cdots \times \tau_k} &= X_{\tau_1} \times \cdots \times X_{\tau_k} \\
\sqsubseteq_{\tau_1 \times \cdots \times \tau_k} &= \{((x_1, \ldots, x_k), (y_1, \ldots, y_k)) \mid x_i \sqsubseteq_{\tau_i} y_i \text{ for each } i \in \{1, \ldots, k\}\} \\
\bot_{\tau_1 \times \cdots \times \tau_k} &= (\bot_{\tau_1}, \ldots, \bot_{\tau_k}).
\end{aligned}$$

By abuse of notation, we often write $[\![\tau]\!]$ also for X_τ. We also often omit the subscript τ and just write $\sqsubseteq$ and $\bot$ for $\sqsubseteq_\tau$ and $\bot_\tau$ respectively. The interpretation of base type $\mathtt{R}$ can actually be restricted to $[0, 1]$, but for technical convenience (to make the existence of a fixpoint trivial) we have defined $X_{\mathtt{R}}$ as $\mathbf{R}_\infty$.

For a type environment Γ, we write $[\![\Gamma]\!]$ for the set of functions that map each $x \in dom(\Gamma)$ to an element of $[\![\Gamma(x)]\!]$. Given $\rho \in [\![\Gamma]\!]$ and e such that $\Gamma \vdash e : \tau$, its semantics $[\![e]\!]_\rho \in [\![\tau]\!]$ is

defined by:

$$\begin{aligned}
[\![r]\!]_\rho &= r \\
[\![x]\!]_\rho &= \rho(x) \\
[\![f]\!]_\rho &= \rho(f) \\
[\![e_1 + e_2]\!]_\rho &= [\![e_1]\!]_\rho + [\![e_2]\!]_\rho \\
[\![e_1 \cdot e_2]\!]_\rho &= [\![e_1]\!]_\rho \cdot [\![e_2]\!]_\rho \\
[\![e_1 e_2]\!]_\rho &= ([\![e_1]\!]_\rho)([\![e_2]\!]_\rho) \\
[\![(e_1, \ldots, e_k)]\!]_\rho &= ([\![e_1]\!]_\rho, \ldots, [\![e_k]\!]_\rho).
\end{aligned}$$

Given $\mathcal{E}$ such that $\Gamma \vdash \mathcal{E}$, we write $\rho_\mathcal{E}$ for the least solution of $\mathcal{E}$, i.e., the least $\rho \in [\![\Gamma]\!]$ such that $[\![f(\widetilde{x}_1) \cdots (\widetilde{x}_\ell)]\!]_{\rho\{\widetilde{x}_1 \mapsto \widetilde{\mathbf{y}}_1, \ldots, \widetilde{x}_\ell \mapsto \widetilde{\mathbf{y}}_\ell\}} = [\![e]\!]_{\rho\{\widetilde{x}_1 \mapsto \widetilde{\mathbf{y}}_1, \ldots, \widetilde{x}_\ell \mapsto \widetilde{\mathbf{y}}_\ell\}}$ for every equation $f(\widetilde{x}_1) \cdots (\widetilde{x}_\ell) = e \in \mathcal{E}$ and $(\widetilde{\mathbf{y}}_1) \in [\![\tau_1]\!], \ldots, (\widetilde{\mathbf{y}}_\ell) \in [\![\tau_\ell]\!]$ with $\Gamma(f) = \tau_1 \to \cdots \to \tau_\ell \to \mathtt{R}$. Note that $\rho_\mathcal{E}$ always exists, and is given by: $\rho_\mathcal{E} = \mathbf{lfp}(\mathcal{F}_\mathcal{E}) = \bigsqcup_{i \in \omega} \mathcal{F}_\mathcal{E}^i(\bot_{[\![\Gamma]\!]})$, where $\mathcal{F}_\mathcal{E} \in [\![\Gamma]\!] \to [\![\Gamma]\!]$ is defined as the map such that

$$\mathcal{F}_\mathcal{E}(\rho)(f) = \lambda\!\!\lambda(\widetilde{\mathbf{y}}_1) \in [\![\tau_1]\!]. \ldots \lambda\!\!\lambda(\widetilde{\mathbf{y}}_\ell) \in [\![\tau_\ell]\!]. [\![e]\!]_{\rho\{\widetilde{x}_1 \mapsto \widetilde{\mathbf{y}}_1, \ldots, \widetilde{x}_\ell \mapsto \widetilde{\mathbf{y}}_\ell\}}$$

for each $f(\widetilde{x}_1) \cdots (\widetilde{x}_\ell) = e \in \mathcal{E}$ with $\Gamma(f) = \tau_1 \to \cdots \to \tau_\ell \to \mathtt{R}$. Note that $\mathcal{F}_\mathcal{E}$ is continuous in the ω-cpo $[\![\Gamma]\!]$.

Example 4.2. Let $\mathcal{E}$ be the system of equations in Example 4.1. Then, $\rho_\mathcal{E}$ is:

$$\left\{ f_1 \mapsto \frac{3}{7}, f_2 \mapsto \lambda\!\!\lambda\, g \in \mathbf{R} \to \mathbf{R}.\, \lambda\!\!\lambda(x_1, x_2) \in \mathbf{R} \times \mathbf{R}. g(x_1 + x_2), f_3 \mapsto \lambda\!\!\lambda\, x \in \mathbf{R}. \frac{3}{7} x \right\}. \qquad \square$$

4.2. **Order-n Fixpoint Characterization.** We now give a translation from an order-n PHORS $\mathcal{G}$ to a system of order-n fixpoint equations $\mathcal{E}$, so that $\mathcal{P}(\mathcal{G}, S) = \rho_\mathcal{E}(S)$. The translation is actually straightforward: we just need to replace $\mathtt{e}$ and Ω with the termination probabilities 1 and 0, and probabilistic choices with summation and multiplication of probabilities. The translation function $(\cdot)^\#$ is defined by:

$$\begin{aligned}
(\mathcal{N}, \mathcal{R}, S)^\# &= (\mathcal{R}^\#, S) \\
\mathcal{R}^\# &= \{F\, \widetilde{x} = p \cdot (t_L)^\# + (1-p) \cdot (t_R)^\# \mid \mathcal{R}(F) = \lambda \widetilde{x}. t_L \oplus_p t_R\} \\
\mathtt{e}^\# &= 1 \qquad \Omega^\# = 0 \qquad x^\# = x \qquad (st)^\# = s^\# t^\#.
\end{aligned}$$

We write $\mathcal{E}_\mathcal{G}$ for $\mathcal{R}^\#$. We define the translation of types and type environments by:

$$\begin{aligned}
\mathtt{o}^\# &= \mathtt{R} \\
(\kappa_1 \to \kappa_2)^\# &= \kappa_1^\# \to \kappa_2^\# \\
(x_1 : \kappa_1, \ldots, x_n : \kappa_n)^\# &= x_1 : \kappa_1^\#, \ldots, x_n : \kappa_n^\#.
\end{aligned}$$

The following lemma states that the output of the translation is well-typed.

Lemma 4.3. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ be an order-n PHORS. Then $\mathcal{N}^\# \vdash \mathcal{E}_\mathcal{G}$ and $\mathcal{N}^\# \vdash S : \mathtt{R}$.*

By the above lemma and the definition of the translation of type environments, it follows that for an order-n PHORS $\mathcal{G}$, the order of $\mathcal{E}_\mathcal{G}$ is also n. The following theorem states the correctness of the translation (see Appendix B.1 for a proof).

Theorem 4.4. *Let $\mathcal{G}$ be an order-n PHORS. Then $\mathcal{P}(\mathcal{G}) = \rho_{\mathcal{E}_\mathcal{G}}(S)$.*

Example 4.5. Recall $\mathcal{G}_1 = (\mathcal{N}_1, \mathcal{R}_1, S)$ from Example 2.3:

$$\mathcal{N}_1 = \{S \mapsto \mathtt{o}, F \mapsto \mathtt{o} \to \mathtt{o}\};$$
$$\mathcal{R}_1 = \{S \;=\; F\,\mathtt{e} \oplus_1 \Omega, \quad F\,x \;=\; x \oplus_p F(F\,x)\}.$$

$\mathcal{N}_1^\# = \{S \mapsto \mathbf{R}, F \mapsto \mathbf{R} \to \mathbf{R}\}$, and $\mathcal{E}_{\mathcal{G}_1}$ consists of: $S = 1 \cdot F(1)$ and $F\,x = p \cdot x + (1-p) \cdot F(F\,x)$. The least solution $\rho_{\mathcal{E}_{\mathcal{G}_1}}$ is

$$S = \begin{cases} \frac{p}{1-p} & \text{if } 0 \le p < \frac{1}{2} \\ 1 & \text{if } \frac{1}{2} \le p \le 1 \end{cases} \qquad F = \lambda\!\!\lambda \mathbf{x}. \begin{cases} \frac{p}{1-p} \cdot \mathbf{x} & \text{if } 0 \le p < \frac{1}{2} \\ \mathbf{x} & \text{if } \frac{1}{2} \le p \le 1. \end{cases}$$ □

Example 4.6. Recall $\mathcal{G}_3$ from Example 2.9:

$$S = \mathit{Listgen}\,(\mathit{Listgen}\,\mathit{Boolgen})\,\mathtt{e} \quad \mathit{Boolgen}\,k = k \quad \mathit{Listgen}\,f\,k = k \oplus_{\frac{1}{2}} (f(\mathit{Listgen}\,f\,k)).$$

The corresponding fixpoint equations are:

$$S = \mathit{Listgen}\,(\mathit{Listgen}\,\mathit{Boolgen})\,1$$
$$\mathit{Boolgen}\,k = k$$
$$\mathit{Listgen}\,f\,k = \frac{1}{2}k + \frac{1}{2}(f(\mathit{Listgen}\,f\,k)).$$

By specializing *Listgen* for the cases $f = \mathit{Listgen}\,\mathit{Boolgen}$ and $f = \mathit{Boolgen}$, we obtain:

$$S = \mathit{ListgenList}\,1$$
$$\mathit{Boolgen}\,k = k$$
$$\mathit{ListgenList}\,k = \frac{1}{2}k + \frac{1}{2}(\mathit{ListgenBool}(\mathit{ListgenList}\,k))$$
$$\mathit{ListgenBool}\,k = \frac{1}{2}k + \frac{1}{2}(\mathit{Boolgen}(\mathit{ListgenBool}\,k)).$$

The least solution is:

$$S = 1 \quad \mathit{Boolgen}\,k = \mathit{ListgenList}\,k = \mathit{ListgenBool}\,k = k.$$ □

4.3. **Order-$(n-1)$ Fixpoint Characterization.** We now characterize the termination probability of order-n PHORS (where $n > 0$) in terms of order-$(n-1)$ equations, so that the fixpoint equations are easier to solve. When $n = 1$, the characterization yields polynomial equations on probabilities; thus the result below may be considered as a generalization of the now classic result on the reachability problem for recursive Markov chains [EY09].

The basic observation (that is also behind the fixpoint characterization for recursive Markov chains [EY09]) is that the termination behavior of an order-1 function of type $\mathtt{o}^\ell \to \mathtt{o}$ can be represented by a tuple of *probabilities* $(p_0, p_1, \ldots, p_\ell)$, where (i) p_0 is the probability that the function terminates *without* using any of its arguments, and (ii) p_i is the probability that the function uses the i-th argument. To see why, consider a term $f\,t_1 \cdots t_\ell$ of type $\mathtt{o}$, where f is an order-1 function of type $\mathtt{o}^\ell \to \mathtt{o}$. In order for $f\,t_1 \cdots t_\ell$ to terminate, the only possibilities are: (i) f terminates without calling any of the arguments, or (ii) f calls t_i for some $i \in \{1, \ldots, \ell\}$, and t_i terminates (notice, in this case, that none of the other t_j's are called: since t_i is of type $\mathtt{o}$, once t_i is called from f, the control cannot go back to f). Thus, the probability that $f\,t_1 \cdots t_\ell$ terminates can be calculated by $p_0 + p_1q_1 + \cdots p_\ell q_\ell$,

where each q_i denotes the probability that t_i terminates. The termination probability is, therefore, independent of the precise internal behavior of f; only $(p_0, p_1, \ldots, p_\ell)$ matters. Thus, information about an order-1 function can be represented as a tuple of real numbers, which is order 0. By generalizing this observation, we can represent information about an order-n function as an order-$(n-1)$ function on (tuples of) real numbers. Since the general translation is quite subtle and requires a further insight, however, let us first confirm the above idea by revisiting Example 2.3.

Example 4.7. Recall $\mathcal{G}_1$ from Example 2.3, consisting of: $S = F\ \mathtt{e}$ and $F\,x = x \oplus_p F(F\,x)$. Here, we have two functions: S of type $\mathtt{o}$ and F of type $\mathtt{o} \to \mathtt{o}$. Based on the observation above, their behaviors can be represented by S_0 and (F_0, F_1) respectively, where S_0 (F_0, resp.) denotes the probability that S (F, resp.) terminates, and F_1 represents the probability that F uses the argument. Those values are obtained as the least solutions for the following system of equations.

$$\begin{aligned} S_0 &= F_0 + F_1 \cdot 1 \\ F_0 &= p \cdot 0 + (1-p)(F_0 + F_1 \cdot F_0) \\ F_1 &= p \cdot 1 + (1-p)(F_1 \cdot F_1 \cdot 1). \end{aligned}$$

To understand the last equation, note that the possibilities that x is used are: (i) F chooses the left branch (with probability p) and then uses x with probability 1, or (ii) F chooses the right branch (with probability $1-p$), the outer call of F uses the argument $F\,x$ (with probability F_1), and the inner call of F uses the argument x. By simplifying the equations, we obtain:

$$\begin{aligned} S_0 &= F_0 + F_1 \\ F_0 &= (1-p)(F_0 + F_1 F_0) \\ F_1 &= p + (1-p)F_1^2. \end{aligned}$$

The least solution is the following:

$$F_0 = 0 \qquad S_0 = F_1 = \begin{cases} \frac{p}{1-p} & \text{if } 0 \le p < \frac{1}{2} \\ 1 & \text{if } \frac{1}{2} \le p \le 1. \end{cases}$$ □

The translation for general orders is more involved. For technical convenience in formalizing the translation, we assume below that the rules of PHORS do not contain $\mathtt{e}$; instead, the start symbol S (which is now a non-terminal of type $\mathtt{o} \to \mathtt{o}$) takes $\mathtt{e}$ from the environment. Thus, the termination probability we consider is $\mathcal{P}(\mathcal{G}, S\,\mathtt{e})$, where $\mathtt{e}$ does not occur in $\mathcal{R}$. This is without any loss of generality, since $\mathtt{e}$ can be passed around as an argument without increasing the order of the underlying PHORS, if it is higher than 0.

To see how we can generalize the idea above to deal with higher-order functions, let us now consider the following example of an order-2 PHORS:

$$\begin{aligned} S\,x &= F\,(H\,x)\,x \\ F\,f\,y &= f(f\,y) \\ H\,x\,y &= x \oplus_{\frac{1}{2}} (y \oplus_{\frac{1}{2}} \Omega). \end{aligned}$$

Suppose we wish to characterize the termination probability of $S\ \mathtt{e}$, i.e., the probability that S uses the first argument. (In this particular case, one can easily compute the termination probability by unfolding all the functions, but we wish to find a compositional translation

which works well in presence of recursion.) We need to compute the probability that $F\ (H\ x)\ x$ reaches (i.e., reduces to) x, which is the probability p_1 that $F\ (H\ x^{(1)})\ x^{(2)}$ reaches $x^{(1)}$, plus the probability p_2 that $F\ (H\ x^{(1)})\ x^{(2)}$ reaches $x^{(2)}$; we have added annotations to distinguish between the two occurrences of x. What information on F is required for computing it? To compute p_2, we need to obtain the probability that F uses the formal argument y. Since it depends on f, we represent it as a function F_1 defined by:

$$F_1\ f_1 = f_1 \cdot f_1.$$

Here, f_1 represents the probability that the original argument f uses its first argument. We can thus represent p_2 as $F_1(H_2)$, where H_2 is $\frac{1}{4}$, the probability that $f = H\,x$ uses the first argument, i.e., the probability that H uses the second argument. Now let us consider how to represent p_1, the probability that $F\,(H\,x^{(1)})\,x^{(2)}$ reaches $x^{(1)}$. We construct another function F_0 from the definition of F for this purpose. A challenge is that the variable x is not visible in (the definition of) F; only the caller of F knows the reachability target x. Thus, we pass to F_0, in addition to f_1 above, another argument f_0, which represents the probability that the argument f reaches the current target (which is x in this case). Therefore, p_1 is represented as $F_0\,(H_1, H_2)$, where

$$F_0\,(f_0, f_1) = f_0 + f_1 \cdot f_0 \qquad H_1 = \frac{1}{2}.$$

In $f_0 + f_1 \cdot f_0$, the occurrence of f_0 on the lefthand side represents the probability that the outer call of f in $f(f\,x)$ reaches the target (without using $(f\,x)$), and $f_1 \cdot f_0$ represents the probability that the outer call of f uses the argument $f\,x$, and then the inner call of f reaches the target. Now, the whole probability that S uses its argument is represented as S_1, where

$$S_1 = F_0(H_1, H_2) + F_1(H_2),$$

with the functions F_0, F_1, H_1 and H_2 being as defined above. Note that the order of the resulting equations is one. In summary, as information about an order-1 argument f of arity k, we pass around a tuple of real numbers $(f_0, f_1, \ldots, f_k)$ where f_i $(i > 0)$ represents the probability that the i-th argument is reached, and f_0 represents the probability that the "current target" (which is chosen by a caller) is reached.

A further twist is required in the case of order-3 or higher. Consider an order-3 function G defined by:

$$G\ h\ z = h\,(H\,z)\,z$$

where $G : ((\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o}$, and H is as defined above. Following the definition of F_1 above, one may be tempted to define G_1 (for computing the reachability probability to z) as $G_1\ h_1 = \cdots$, where h_1 is a function to be used for computing the probability that h uses its order-0 argument. However, h_1 is not sufficient for computing the reachability probability to z; passing the reachability probability to the *current* target (like f_0 above) does not help either, since a caller of G does not know the current target z. We thus need to add an additional argument h_2 for computing the probability to a target that is yet to be set by a caller of h. Thus, the definition of G_1 is:[2]

$$G_1\ (h_1, h_2) = h_2(H_1, H_2) + h_1(1).$$

[2]For the sake of simplicity, the following translation slightly deviates from the general translation defined later.

Here, $h_2(H_1, H_2)$ and $h_1(1)$ respectively represent the probabilities that $G\,(H\,z^{(1)})\,z^{(2)}$ reaches $z^{(1)}$, and $z^{(2)}$. The first argument of h_2 (i.e., H_1) represents the probability that $H\,z$ reaches z, and the second argument of h_2 (i.e., H_2) represents the probability that $H\,z$ reaches its argument (the second argument of H).

We can now formalize the general translation based on the intuitions above. We often write $\kappa_1 \to \cdots \to \kappa_k \Rightarrow \mathtt{o}^\ell \to \mathtt{o}$ for $\kappa_1 \to \cdots \to \kappa_k \to \mathtt{o}^\ell \to \mathtt{o}$ when either $\mathit{order}(\kappa_k) > 0$ or $k = 0$. We define $\mathtt{ar}(\kappa)$ as the number of the last order-0 arguments, i.e., $\mathtt{ar}(\kappa_1 \to \cdots \to \kappa_k \Rightarrow \mathtt{o}^\ell \to \mathtt{o}) = \ell$.

Given a rule $F\,z_1\,\ldots\,z_m = t_L \oplus_p t_R$ of PHORS, we uniquely decompose $z_1, \ldots, z_m$ into two (possibly empty) subsequences $z_1, \ldots, z_\ell$ and $z_{\ell+1}, \ldots, z_m$ so that the order of z_ℓ is greater than 0 if $\ell > 0$ (note, however, that the orders of $z_1, \ldots, z_{\ell-1}$ may be 0), and $z_{\ell+1}, \ldots, z_m$ are order-0 variables (in other words, $z_{\ell+1}, \ldots, z_m$ is the maximal postfix of $z_1, \ldots, z_m$ consisting of only order-0 variables). Since (the last consecutive occurrences of) order-0 arguments will be treated in a special manner, as a notational convenience, when we write $F\,\widetilde{y}\,\widetilde{x} = t_L \oplus_p t_R$ for a rule of PHORS, we implicitly assume that $\widetilde{x}$ is the maximal postfix of the sequence $\widetilde{y}\,\widetilde{x}$ consisting of only order-0 variables. Similarly, when we write $F\,\widetilde{s}\,\widetilde{t}$ for a fully-applied term (of order 0), we implicitly assume that $\widetilde{t}$ is the maximal postfix of the sequence of arguments, consisting of only order-0 terms.

Consider a function definition of the form:

$$F\ y_1\ \cdots\ y_m\ x_1\ \cdots\ x_k = t_L \oplus_p t_R$$

where (following the notational convention above) the sequence $x_1, \ldots, x_k$ is the maximal postfix of $y_1, \ldots, y_m, x_1, \ldots, x_k$ consisting of only order-0 variables. We transform each subterm t of the righthand side $t_L \oplus_p t_R$ by using the translation relation of the form:

$$\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (e_0, e_1, \ldots, e_{\ell+k+1})$$

where $\mathcal{N}$ and $\mathcal{K}$ are type environments for the underlying non-terminals and $y_1, \ldots, y_m$ respectively, and κ is the type of t with $\mathtt{ar}(\kappa) = \ell$. We often omit the subscript $\mathcal{N}$. The output of the translation, $(e_0, e_1, \ldots, e_{\ell+k+1})$, can be interpreted as capturing the following information.

- e_0: the reachability probability (or a function that returns the probability, given appropriate arguments; similarly for the other e_i's below) to the current target (set by a caller of F).
- e_i ($i \in \{1, \ldots, \ell\}$): the reachability probability to t's i-th order-0 argument.
- $e_{\ell+i}$ ($i \in \{1, \ldots, k\}$): the reachability probability to x_i.
- $e_{\ell+k+1}$: the reachability probability to a "fresh" target (that can be set by a caller of t); this is the component that should be passed as h_2 in the discussion above. In a sense, this component represents the reachability probability to a variable x_{k+1} that is "fresh" for t (in that it does not occur in t).

In the translation, each variable y (including non-terminals) of type $\widetilde{\kappa} \Rightarrow \mathtt{o}^m \to \mathtt{o}$ is replaced by $(y_0, y_1, \ldots, y_m, y_{m+1})$, which represents information analogous to $(e_0, e_1, \ldots, e_\ell, e_{\ell+k+1})$: y_0 represents (a function for computing) the reachability probability to the current target, y_i ($i \in \{1, \ldots, m\}$) represents the reachability probability to the i-th order-0 argument (among the last m argument), and y_{m+1} (which corresponds to h_2 in the explanation above) represents the reachability probability to a fresh target (to be set later). In contrast, the variables $x_1, \ldots, x_k$ will be removed by the translation.

$$\frac{}{\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} \Omega : \mathtt{o} \rightsquigarrow (0^{k+2})} \quad \text{(Tr-Omega)}$$

$$\frac{}{\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} x_i : \mathtt{o} \rightsquigarrow (0^i, 1, 0^{k-i+1})} \quad \text{(Tr-GVar)}$$

$$\frac{\mathcal{K}(y) = \widetilde{\kappa} \Rightarrow \mathtt{o}^\ell \to \mathtt{o}}{\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} y : \widetilde{\kappa} \Rightarrow \mathtt{o}^\ell \to \mathtt{o} \rightsquigarrow (y_0, y_1, \ldots, y_\ell, (y_{\ell+1})^{k+1})} \quad \text{(Tr-Var)}$$

$$\frac{\mathcal{N}(F) = \widetilde{\kappa} \Rightarrow \mathtt{o}^\ell \to \mathtt{o}}{\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} F : \widetilde{\kappa} \Rightarrow \mathtt{o}^\ell \to \mathtt{o} \rightsquigarrow (F_0, F_1, \ldots, F_\ell, (F_0)^{k+1})} \quad \text{(Tr-NT)}$$

$$\frac{\begin{array}{c}\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} s : \kappa_1 \to \widetilde{\kappa} \Rightarrow \mathtt{o}^\ell \to \mathtt{o} \rightsquigarrow (s_0, \ldots, s_{\ell+k+1}) \\ \mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} t : \kappa_1 \rightsquigarrow (t_0, \ldots, t_{\ell'+k+1}) \qquad \mathtt{ar}(\kappa_1) = \ell'\end{array}}{\begin{array}{l}\mathcal{K}; \widetilde{x} \vdash_{\mathcal{N}} st : \widetilde{\kappa} \Rightarrow \mathtt{o}^\ell \to \mathtt{o} \rightsquigarrow (s_0(t_0, \ldots, t_{\ell'}, t_{\ell'+k+1}), s_1(t_1, \ldots, t_{\ell'}, t_{\ell'+k+1}), \ldots, s_\ell(t_1, \ldots, t_{\ell'}, t_{\ell'+k+1}), \\ \qquad s_{\ell+1}(t_{\ell'+1}, t_1, \ldots, t_{\ell'}, t_{\ell'+k+1}) \ldots, s_{\ell+k+1}(t_{\ell'+k+1}, t_1, \ldots, t_{\ell'}, t_{\ell'+k+1}))\end{array}} \quad \text{(Tr-App)}$$

$$\frac{\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} s : \mathtt{o}^{\ell+1} \to \mathtt{o} \rightsquigarrow (s_0, \ldots, s_{k+\ell+2}) \qquad \mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} t : \mathtt{o} \rightsquigarrow (t_0, \ldots, t_{k+1})}{\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} st : \mathtt{o}^\ell \to \mathtt{o} \rightsquigarrow (s_0 + s_1 \cdot t_0, s_2, \ldots, s_{\ell+1}, s_{\ell+2} + s_1 \cdot t_1, \ldots, s_{\ell+k+2} + s_1 \cdot t_{k+1})} \quad \text{(Tr-AppG)}$$

$$\frac{\begin{array}{c}y_1 : \kappa_1, \ldots, y_\ell : \kappa_\ell; x_1, \ldots, x_k \vdash_{\mathcal{N}} t_d : \mathtt{o} \rightsquigarrow (t_{d,0}, \ldots, t_{d,k+1}) \text{ for each } d \in \{L, R\} \\ \widetilde{y_i} = (y_{i,0}, \ldots, y_{i,\mathtt{ar}(\kappa_i)+1}) \qquad \widetilde{y_i}' = (y_{i,1}, \ldots, y_{i,\mathtt{ar}(\kappa_i)+1})\end{array}}{\begin{array}{c}\mathcal{N} \vdash (F\, \widetilde{y}\, x_1 \cdots x_k = t_L \oplus_p t_R) \rightsquigarrow \\ \{F_i\, \widetilde{y_1}' \cdots \widetilde{y_\ell}' = p t_{L,i} + (1-p) t_{R,i} \mid i \in \{1, \ldots, k\}\} \cup \{F_0\, \widetilde{y_1} \cdots \widetilde{y_\ell} = p t_{L,0} + (1-p) t_{R,0}\}\end{array}} \quad \text{(Tr-Rule)}$$

$$\frac{\mathcal{E} = \bigcup \{\mathcal{E}_i \mid \mathcal{N} \vdash (F\, \widetilde{y}\, x_1 \cdots x_k = t_L \oplus_p t_R) \rightsquigarrow \mathcal{E}_i, (F\, \widetilde{y}\, x_1 \cdots x_k = t_L \oplus_p t_R) \in \mathcal{R}\}}{(\mathcal{N}, \mathcal{R}, S) \rightsquigarrow (\mathcal{E}, S_1)} \quad \text{(Tr-Gram)}$$

FIGURE 5. Translation rules for the order-$(n-1)$ fixpoint characterization

The translation rules are given in Figure 5. In the rules, to clarify the correspondence between source terms and target expressions, we use metavariables $s, t, \ldots$ (with subscripts) also for target expressions (instead of e). We write e^k for the k repetitions of e.

We now explain the translation rules. In rule Tr-Omega for the constant Ω, all the components are 0 because Ω represents divergence. There is no rule for $\mathtt{e}$; this is due to the assumption that $\mathtt{e}$ never occurs in the rules. Rule Tr-GVar is for order-0 variables, for which only one component is 1 and all the others are 0. The $(i+1)$-th component is 1, because it represents the probability that x_i is reached. In rule Tr-Var for variables, the first $\ell+1$ components are provided by the environment. Since y (that is provided by the environment) does not "know" the local variables $x_1, \ldots, x_k$ (in other words, y cannot be instantiated to a term that contains x_i), the default parameter $y_{\ell+1}$ (for computing the reachability probability to a "fresh" target) is used for all of those components. The rule Tr-NT for non-terminals is almost the same as Tr-Var, except that F_0 is used instead of $F_{\ell+1}$. This is because F does not contain any free variables; the reachability target for F is not set yet, hence F_0 can be used for computing the reachability probability to a fresh target. Rule Tr-App is for applications. Basically, the output of the translation of t is passed to s_i; note however that t_0 is passed only to s_0; since $s_1, \ldots, s_{\ell+k}$ should provide the reachability probability to order-0 arguments of s or local variables, the reachability probability to the current target (that is represented by t_0) is irrelevant for them. For $s_{\ell+1}, \ldots, s_{\ell+k}$, the

reachability targets are $x_1, \ldots, x_k$; thus, information about how t reaches those variables is passed as the first argument of $s_{\ell+1}, \ldots, s_{\ell+k}$. For the last component, $s_{\ell+k+1}$ and $t_{\ell'+k+1}$ are used so that the reachability target can be set later. In rule Tr-AppG, the reachability probability to the current target (expressed by the first component) is computed by $s_0 + s_1 \cdot t_0$, because the current target is reached without using t (as represented by s_0), or t is used (as represented by s_1) and t reaches the current target (as represented by t_0); similarly for the reachability probability to local variables. Tr-Rule is the rule for translating a function definition. From the definition for F, we generate definitions for functions $F_0, \ldots, F_k$. For $i \in \{1, \ldots, k\}$, $t_{d,i}$ is chosen as the body of F_i, since it represents the reachability probability to x_i. Rule Tr-Gram is the translation for the whole PHORS; we just collect the output of the translation for each rule.

For a PHORS $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ (where $\mathcal{N}(S) = \mathsf{o} \to \mathsf{o}$), we write $\mathcal{E}^{\mathtt{ref}}_{\mathcal{G}}$ for $\mathcal{E}$ such that $(\mathcal{N}, \mathcal{R}, S) \rightsquigarrow (\mathcal{E}, S_1)$. Such an $\mathcal{E}$ is actually unique (up to α-equivalence), given $\mathcal{N}$ and $\mathcal{R}$. Note also that by definition of the translation relation, the output of the translation always exists.

Example 4.8. Recall the order-2 PHORS $\mathcal{G}_2$ in Example 2.4:

$$\begin{aligned} S &= F\, H \\ H\, x &= x \oplus_{\frac{1}{2}} \Omega \\ F\, g &= (g\, \mathtt{e}) \oplus_{\frac{1}{2}} (F(D\, g)) \\ D\, g\, x &= g\, (g\, x). \end{aligned}$$

It can be modified to the following rules so that $\mathtt{e}$ does not occur.

$$\begin{aligned} S\, z &= F\, H\, z \\ H\, x &= x \oplus_{\frac{1}{2}} \Omega \\ F\, g\, z &= (g\, z) \oplus_{\frac{1}{2}} (F(D\, g)\, z) \\ D\, g\, x &= g\, (g\, x). \end{aligned}$$

Here, $\mathtt{e}$ can be passed around through the variable z. Consider the body $F\, H\, z$ of S. F and H are translated as follows.

$$\begin{aligned} &\emptyset; z \vdash_{\mathcal{N}} F : (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o} \rightsquigarrow (F_0, F_1, F_0, F_0) \\ &\emptyset; z \vdash_{\mathcal{N}} H : \mathsf{o} \to \mathsf{o} \rightsquigarrow (H_0, H_1, H_0, H_0) \end{aligned}$$

By applying Tr-App, we obtain:

$$\emptyset; z \vdash_{\mathcal{N}} F\, H : \mathsf{o} \to \mathsf{o} \rightsquigarrow (F_0(H_0, H_1, H_0), F_1(H_1, H_0), F_0(H_0, H_1, H_0), F_0(H_0, H_1, H_0)).$$

Using Tr-GVar, z can be translated as follows.

$$\emptyset; z \vdash_{\mathcal{N}} z : \mathsf{o} \rightsquigarrow (0, 1, 0).$$

Thus, by applying Tr-AppG, we obtain:

$$\begin{aligned} &\emptyset; z \vdash_{\mathcal{N}} F\, H\, z : \mathsf{o} \rightsquigarrow \\ &\quad (F_0(H_0, H_1, H_0) + F_1(H_1, H_0) \cdot 0, \\ &\quad\ F_0(H_0, H_1, H_0) + F_1(H_1, H_0) \cdot 1, F_0(H_0, H_1, H_0) + F_1(H_1, H_0) \cdot 0). \end{aligned}$$

By simplifying the output, we obtain:

$$\emptyset; z \vdash_{\mathcal{N}} F\, H\, z : \mathsf{o} \rightsquigarrow (F_0(H_0, H_1, H_0), F_0(H_0, H_1, H_0) + F_1(H_1, H_0), F_0(H_0, H_1, H_0)).$$

Thus, we have the following equations for S_0 and S_1.

$$S_0 = F_0(H_0, H_1, H_0) \qquad S_1 = F_0(H_0, H_1, H_0) + F_1(H_1, H_0).$$

The following equations are obtained for the other non-terminals.

$$\begin{aligned} H_0 &= 0 \qquad\qquad H_1 = \frac{1}{2} \\ F_0\,(g_0, g_1, g_2) &= \frac{1}{2}g_0 + F_0(D_0(g_0, g_1, g_2), D_1(g_1, g_2), D_0(g_2, g_1, g_2)) \\ F_1\,(g_1, g_2) &= \frac{1}{2}(g_1 + g_2) + \frac{1}{2}(F_0(D_0(g_2, g_1, g_2), D_1(g_1, g_2), D_0(g_2, g_1, g_2)) \\ &\quad + F_1(D_1(g_1, g_2), D_0(g_2, g_1, g_2)) \\ D_0(g_0, g_1, g_2) &= g_0 + g_1 g_0 \\ D_1(g_1, g_2) &= g_2 + g_1(g_1 + g_2). \end{aligned}$$

We can observe that the values of the variables g_0 and g_2 are always 0. Thus, by removing redundant arguments, we obtain:

$$\begin{aligned} S_0 &= F_0(\frac{1}{2}) \\ S_1 &= F_0(\frac{1}{2}) + F_1(\frac{1}{2}) \\ F_0(g_1) &= F_0(D_1(g_1)) \\ F_1(g_1) &= \frac{1}{2}g_1 + \frac{1}{2}(F_0(D_1(g_1)) + F_1(D_1(g_1))) \\ D_0(g_1) &= 0 \\ D_1(g_1) &= g_1^2. \end{aligned}$$

By further simplification (noting that the least solution for F_0 is $\lambda g_1.0$), we obtain:

$$S_1 = F_1(\tfrac{1}{2}) \qquad F_1(g_1) = \tfrac{1}{2}g_1 + \tfrac{1}{2}F_1(g_1^2).$$

The least solution of S_1 is $\Sigma_{i\geq 0}\frac{1}{2^{2^i+i+1}} = 0.3205\cdots$. □

Example 4.9. Consider the following order-3 PHORS:

$$S\,x = F(C\,x) \quad F\,g = g\,H \quad C\,x\,f = f\,x \quad H\,x = x \oplus_{\frac{1}{2}} \Omega,$$

where

$$S : \mathsf{o} \to \mathsf{o}, F : ((\mathsf{o} \to \mathsf{o}) \to \mathsf{o}) \to \mathsf{o}, C : \mathsf{o} \to (\mathsf{o} \to \mathsf{o}) \to \mathsf{o}, H : \mathsf{o} \to \mathsf{o}.$$

This is a tricky example, where in the body of S, x is embedded into the closure $C\,x$ and passed to another function F; so, in order to compute how S uses x, we have to take into account how F uses the closure passed as the argument. The PHORS is translated to:

$$\begin{aligned} S_0 &= F_0(C_0(0,0), C_0(0,0)) \\ S_1 &= F_0(C_0(1,0), C_0(0,0)) \\ F_0\,(g_0, g_1) &= g_0(H_0, H_1, H_0) \\ C_0\,(x_0, x_1)\,(f_0, f_1, f_2) &= f_0 + f_1 \cdot x_0 \\ H_0 &= 0 \qquad H_1 = \frac{1}{2}, \end{aligned}$$

where

$$\begin{aligned}&S_0 : \mathtt{R}, S_1 : \mathtt{R},\\ &F_0 : (\mathtt{R} \times \mathtt{R} \times \mathtt{R} \to \mathtt{R}) \times (\mathtt{R} \times \mathtt{R} \times \mathtt{R} \to \mathtt{R}) \to \mathtt{R},\\ &C_0 : (\mathtt{R} \times \mathtt{R}) \to (\mathtt{R} \times \mathtt{R} \times \mathtt{R}) \to \mathtt{R},\\ &H_0 : \mathtt{R}, H_1 : \mathtt{R}.\end{aligned}$$

The order of the equations is 2 (where the largest order is that of the type of F_0). We have:

$$S_1 = F_0(C_0(1,0), C_0(0,0)) = C_0(1,0)(H_0, H_1, H_0) = H_0 + H_1 \cdot 1 = \frac{1}{2}.$$

In fact, the probability that $S\,\mathtt{e}$ reaches $\mathtt{e}$ is $\frac{1}{2}$. □

Example 4.10. Recall PHORS $\mathcal{G}_5$ from Example 2.10:

$$\begin{aligned}S\ x &= \mathit{Treegen}\ H\ \mathit{Boolgen}\ x\\ \mathit{Boolgen}\ k &= k\\ H\ x\ y &= x \oplus_{\frac{1}{2}} y\\ G\ p\ x\ y &= x \oplus_{\frac{1}{2}} (p\ x\ y)\\ \mathit{Treegen}\ p\ f\ k &= p\ k\ (f(\mathit{Treegen}\,(G\ p)\ f\ (\mathit{Treegen}\,(G\ p)\ f\ (\mathit{Treegen}\,(G\ p)\ f\ k)))).\end{aligned}$$

(Here, we have slightly modified the original PHORS so that S is parameterized with $\mathtt{e}$.) As the output of the translation as defined above is too complex, we show below a hand-optimized version of the fixpoint equations.

$$\begin{aligned}S_1 &= \mathit{Treegen}_1\ (H_1, H_2)\\ H_1 = H_2 &= \frac{1}{2}\\ G_1\ (p_1, p_2) &= \frac{1}{2} + \frac{1}{2}p_1\\ G_2\ (p_1, p_2) &= \frac{1}{2}p_2\\ \mathit{Treegen}_1\ (p_1, p_2) &= p_1 + p_2 \cdot (\mathit{Treegen}_1\ (G_1(p_1, p_2), G_2(p_1, p_2)))^3.\end{aligned}$$

Here, $\mathit{Treegen}_1$ is the function that returns the probability that $\mathit{Treegen}\ p\ \mathit{Boolgen}\ x$ reaches x, where the parameters p_1 and p_2 represent the probabilities that p chooses the first and second branches respectively. Let ρ be the least solution of the fixpoint equations above. We can find $\rho(S_1) = 1$ based on the following reasoning (which is also confirmed by the experiment reported in Section 6). Let us define an m-th approximation $\mathit{Treegen}_1^{(m)}$ of $\rho(\mathit{Treegen}_1)$ by

$$\begin{aligned}\mathit{Treegen}_1^{(0)}\ (p_1, p_2) &= 0\\ \mathit{Treegen}_1^{(m+1)}\ (p_1, p_2) &= p_1 + p_2 \cdot (\mathit{Treegen}_1^{(m)}\ (G_1(p_1, p_2), G_2(p_1, p_2)))^3.\end{aligned}$$

Then $\rho(\mathit{Treegen}_1)\ (p_1, p_2) \geq \mathit{Treegen}_1^{(m)}\ (p_1, p_2)$ for every $m \geq 0$. We show $\mathit{Treegen}_1^{(m)}\ (1 - \frac{1}{2^n}, \frac{1}{2^n}) \geq 1 - \frac{1}{2^{n+m-1}}$ for every $n \geq 2$, $m \geq 1$ by induction on n. When $m = 1$, we have:

$$\mathit{Treegen}_1^{(1)}\ \left(1 - \frac{1}{2^n}, \frac{1}{2^n}\right) = \left(1 - \frac{1}{2^n}\right) + \frac{1}{2^n} \cdot 0 = 1 - \frac{1}{2^n} = 1 - \frac{1}{2^{n+m-1}}.$$

About the inductive step, we have

$$\begin{aligned}
\mathit{Treegen}_1^{(m+1)}\left(1-\frac{1}{2^n},\frac{1}{2^n}\right) &= 1-\frac{1}{2^n}+\frac{1}{2^n}\cdot\left(\mathit{Treegen}_1^{(m)}\left(1-\frac{1}{2^{n+1}},\frac{1}{2^{n+1}}\right)\right)^3\\
&\geq 1-\frac{1}{2^n}+\frac{1}{2^n}\cdot\left(1-\frac{1}{2^{n+m}}\right)^3\\
&\geq 1-\frac{1}{2^n}+\frac{1}{2^n}\cdot\left(1-3\frac{1}{2^{n+m}}\right)\\
&\geq 1-\frac{1}{2^{2n+m-2}} \geq 1-\frac{1}{2^{n+(m+1)-1}}
\end{aligned}$$

as required. Thus,

$$\mathit{Treegen}_1^{(m)}\left(\frac{1}{2},\frac{1}{2}\right)=\frac{1}{2}+\frac{1}{2}\left(\mathit{Treegen}_1^{(m-1)}\left(1-\frac{1}{2^2},\frac{1}{2^2}\right)\right)^3 \geq \frac{1}{2}+\frac{1}{2}\left(1-\frac{1}{2^{(2+(m-1)-1)}}\right)^3$$

for every $m \geq 2$. Thus, $\rho(S_1) = \rho(\mathit{Treegen}_1)(\frac{1}{2},\frac{1}{2})$ (which should be no less than $\mathit{Treegen}_1^{(m)}(\frac{1}{2},\frac{1}{2})$ for every m) must be 1. □

Correctness of the Translation. To state the well-formedness of the output of the translation, we define the translation of types as follows.

$$\begin{aligned}
&(\kappa_1 \to \cdots \to \kappa_k \Rightarrow \mathtt{o}^\ell \to \mathtt{o})^\dagger\\
&= (\kappa_1{}^\dagger \to \cdots \to \kappa_k{}^\dagger \to \mathtt{R}) \times (\kappa_1{}^{\dagger'} \to \cdots \to \kappa_k{}^{\dagger'} \to \mathtt{R})^\ell \times (\kappa_1{}^\dagger \to \cdots \to \kappa_k{}^\dagger \to \mathtt{R})\\
&(\kappa_1 \to \cdots \to \kappa_k \Rightarrow \mathtt{o}^\ell \to \mathtt{o})^{\dagger'}\\
&= (\kappa_1{}^{\dagger'} \to \cdots \to \kappa_k{}^{\dagger'} \to \mathtt{R})^\ell \times (\kappa_1{}^\dagger \to \cdots \to \kappa_k{}^\dagger \to \mathtt{R}).
\end{aligned}$$

We also write $(\kappa_1 \to \cdots \to \kappa_k \Rightarrow \mathtt{o}^\ell \to \mathtt{o})^{\dagger+m}$ for

$$(\kappa_1{}^\dagger \to \cdots \to \kappa_k{}^\dagger \to \mathtt{R}) \times (\kappa_1{}^{\dagger'} \to \cdots \to \kappa_k{}^{\dagger'} \to \mathtt{R})^\ell \times (\kappa_1{}^\dagger \to \cdots \to \kappa_k{}^\dagger \to \mathtt{R})^{m+1}.$$

It represents the type of the tuple $(e_0,\ldots,e_{\ell+m+1})$ obtained by translating a term of type $\kappa_1 \to \cdots \to \kappa_k \Rightarrow \mathtt{o}^\ell \to \mathtt{o}$ with order-0 variables $x_1,\ldots,x_m$. The distinction between $\kappa_i{}^\dagger$ and $\kappa_i{}^{\dagger'}$ reflects the fact that in the output $(e_0,e_1,\ldots,e_\ell,e_{\ell+1},\ldots,e_{\ell+m+1})$ of the translation, $e_1,\ldots,e_\ell$ take one less argument (recall Tr-App). The translation of the type environment $\mathcal{N}$ for non-terminals is defined by:

$$(F_1:\kappa_1,\ldots,F_k:\kappa_k)^\dagger = (F_{1,0},\ldots,F_{1,\mathtt{ar}(\kappa_1)}):\kappa_1{}^{\dagger-1},\ldots,(F_{k,0},\ldots,F_{k,\mathtt{ar}(\kappa_k)}):\kappa_k^{\dagger-1}.$$

The following lemma states that the output of the translation is well-typed.

Lemma 4.11 (Well-typedness of the output of transformation)**.** *Let $\mathcal{G}=(\mathcal{N},\mathcal{R},S)$ be a PHORS. If $\mathcal{G} \rightsquigarrow (\mathcal{E},S_1)$, then $\mathcal{N}^\dagger \vdash \mathcal{E}$ and $\mathcal{N}^\dagger(S_1)=\mathtt{R}$.*

As a corollary, it follows that for any order-n PHORS $\mathcal{G}$ (where $n>0$), the order of $\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}$ is $n-1$.

The following result is the main theorem of this section, which states the correctness of the translation.

Theorem 4.12. *Let $\mathcal{G}=(\mathcal{N},\mathcal{R},S)$ be an order-n PHORS, Then, $\mathcal{P}(\mathcal{G},S\,\mathtt{e}) = \rho_{\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}}(S_1)$.*

A proof of the theorem is found in Appendix B.2. Here we only sketch the proof. We first prove the theorem for *recursion-free* PHORS (so that any term is strongly normalizing; see Appendix B.1 for the precise definition), and extend it to general PHORS $\mathcal{G}$ by using *finite approximations* of $\mathcal{G}$, obtained by unfolding each non-terminal a finite number of times. To show the theorem for recursion-free PHORS, we prove that the translation relation is preserved by reductions in a certain sense; this is, however, much more involved than the corresponding proof for Section 4.2: we introduce an alternative operational semantics for PHORS that uses *explicit* substitutions. See Appendix B.2 for details.

5. Computing Upper-Bounds of Termination Probability

Theorems 4.4 and 4.12 immediately provide procedures for computing *lower*-bounds of the termination probability as precisely as we need [3] The termination probability, in other words, is a *recursively enumerable* real number (see, e.g. [Cal02]), but it is still open whether it is a *recursive* one. Indeed, computing good upper-bounds is non-trivial. For example, an upper-bound for the *greatest* solution of $\mathcal{E}^{\mathtt{ref}}_{\mathcal{G}}$ can be easily computed, but it does not provide a good upper-bound for the *least* solution of $\mathcal{E}^{\mathtt{ref}}_{\mathcal{G}}$, unless the solution is unique. Take, as an example, the trivial PHORS consisting of a single equation $S = S$: the greatest solution is 1, while the least is 0.

In this section, we will describe how *upper approximations* to the termination probability can be computed in practice. We focus our attention mainly on order-2 PHORS, which yield equations over first-order functions on real numbers. Order-n case is only briefly discussed in Section 5.3.

5.1. Properties of the Fixpoint Equations Obtained from PHORS.

Before discussing how to compute an upper-bound of the termination probability, we first summarize several important properties of the (order-1) fixpoint equations obtained from an order-2 PHORS (by the translation in Section 4.3), which are exploited in computing upper-bounds.

(1) The fixpoint equations can be written in the form:

$$f_1(x_1, \ldots, x_{\ell_1}) = e_1, \qquad \cdots \qquad f_k(x_1, \ldots, x_{\ell_k}) = e_k, \tag{5.1}$$

where each e_i consists of (i) non-negative constants, (ii) additions, (iii) multiplications, and (iv) function applications. Each variable x_i ranges over $[0, 1]$.

(2) The formal arguments $x_1, \ldots, x_{\ell_i}$ of each function f_i can be partitioned into several groups of variables $(x_1, \ldots, x_{d_{i,1}}), (x_{d_{i,1}+1}, \ldots, x_{d_{i,2}}), \ldots, (x_{d_{g_i-1}+1}, \ldots, x_{\ell_i})$, so that the relevant input values are those such that *the sum* of the values of the variables in each group ranges over $[0, 1]$. This is because each group of variables $(x_{d_{i,j-1}+1}, \ldots, x_{d_{i,h}})$ either corresponds to an order-0 argument (and has thus length 1) or to an order-1 argument of an order-2 function F_i of the original PHORS, where one of the variables represents the probability that F_i terminates without using any arguments, and each of the other variables represents the probability that F_i uses each argument of F_i. Since these events are mutually exclusive, the sum of those values ranges over $[0, 1]$.

[3] Theorem 2.5 also provides a procedure for computing lower-bounds, but the fixpoint characterizations by Theorems 4.4 and 4.12 provide a more efficient procedure.

(3) The functions $f_1, \ldots, f_k$ can also be partitioned into several groups of functions $(f_1, \ldots, f_{j_1}), (f_{j_1+1}, \ldots, f_{j_2}), \ldots, (f_{j_{\ell-1}+1}, \ldots, f_{j_\ell})$, so that the sum $f_{j_{m-1}+1}(\widetilde{x}) + \cdots + f_{j_m}(\widetilde{x})$ of the return values of the functions in each group ranges over $[0, 1]$ (assuming that the arguments $\widetilde{x}$ are in the valid domain, i.e., the sum of $\widetilde{x}$ ranges over $[0, 1]$). This is because an order-2 function F_i is translated to a tuple of order-1 functions $(F_{i,0}, \ldots, F_{i,j})$, and the components of the tuple return the probabilities to reach mutually different targets.[4] We write $\texttt{fgrp}(f)$ for the partition that f belongs to, i.e., $\texttt{fgrp}(f_i) = \{f_{j_{m-1}+1}, \ldots, f_{j_m}\}$ if $j_{m-1} + 1 \leq i \leq j_m$.

(4) Suppose that $(x_1, \ldots, x_{\ell_i})$ ranges over the valid domain of f_i. Then, the value of each subexpression of e_i ranges over $[0, 1]$; this is because each subexpression represents some probability. This invariant is not necessarily preserved by simplifications like $\frac{1}{2}x + \frac{1}{2}y = \frac{1}{2}(x + y)$; the value of $x + y$ may not belong to $[0, 1]$. We apply simplifications only so that the invariant is maintained.

The properties above can be easily verified by inspecting the translations from Section 4.3.

Finally, another important property is pointwise convexity. The least solution f of the fixpoint equations, as well as any finite approximations obtained from $\bot$ by Kleene iterations, are *pointwise convex*, i.e., convex on each variable, i.e., $f(x_1, \ldots, (1-p)x + py, \ldots, x_n) \leq (1-p)f(x_1, \ldots, x, \ldots, x_n) + pf(x_1, \ldots, y, \ldots, x_n)$ whenever $0 \leq p \leq 1$ and $0 \leq x, y$. Note, however, that f is not necessarily convex in the usual sense: $f((1-p)\vec{x} + p\vec{y}) \leq (1-p)f(\vec{x}) + pf(\vec{y})$ may not hold for some $\vec{0} \leq \vec{x}, \vec{y}$ and $0 \leq p \leq 1$. For example, let $f(x_1, x_2)$ be $x_1 \cdot x_2$. Then, $\frac{1}{4} = f\left(\frac{1}{2}, \frac{1}{2}\right) = f\left(\frac{1}{2}(0, 1) + \frac{1}{2}(1, 0)\right) > \frac{1}{2}f(0, 1) + \frac{1}{2}f(1, 0) = 0$. Recall that $\mathcal{F}_{\mathcal{E}_{\mathcal{G}}^{\texttt{ref}}}$ is the functional associated with the fixpoint equations $\mathcal{E}_{\mathcal{G}}^{\texttt{ref}}$; we simply write $\mathcal{F}$ for $\mathcal{F}_{\mathcal{E}_{\mathcal{G}}^{\texttt{ref}}}$ below.

Lemma 5.1. *$\mathcal{F}^m(\bot)$ and $\mathbf{lfp}(\mathcal{F})$ are both pointwise convex. They are also monotonic.*

Proof. The pointwise convexity and monotonicity of $\mathcal{F}^m(\bot)$ follow from the fact that, following our first observation, it is (a tuple of) multi-variate polynomials with non-negative integer coefficients. The pointwise convexity of $\mathbf{lfp}(\mathcal{F})$ follows from the fact that, for every m, when $\vec{x}$ and $\vec{y}$ differ by at most one coordinate,

$$\begin{array}{rcl} (1-p)\mathbf{lfp}(\mathcal{F})(\vec{x}) + p\mathbf{lfp}(\mathcal{F})(\vec{y}) & \geq & (1-p)\mathcal{F}^m(\bot)(\vec{x}) + p\mathcal{F}^m(\bot)(\vec{y}) \\ & \geq & \mathcal{F}^m(\bot)((1-p)\vec{x} + p\vec{y}) \end{array}$$

and we can then take the supremum to conclude. The monotonicity of $\mathbf{lfp}(\mathcal{F})$ also follows from a similar argument. □

5.2. **Computing an Upper-Bound by Discretization.** Given fixpoint equations as in (5.1), we can compute an upper-bound of the least solution of the equations, by overapproximating the values of $f_1, \ldots, f_k$ at a finite number of discrete points, *à la* "Finite Element Method". To clarify the idea, we first describe a method for the simplest case of a single equation $f(x) = e$ on a unary function in Section 5.2.1. We then extend it to deal with a *binary* function in Section 5.2.2, and discuss the general case (where we need to deal with *multiple equations on multi-variate functions*) in Section 5.2.3.

[4] According to the translation in Section 4.3, the first element $F_{i,0}$ takes one more argument than the other elements; for the sake of simplicity, we assume in this section that all the functions in each partition take the same number of arguments, by adding dummy arguments as necessary.

5.2.1. *Computing an Upper-Bound for a Unary Function.* Suppose that we are given a PHORS $\mathcal{G}$ and that $\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}$ consists of a single equation $f(x) = e$, where f is a function f from $[0,1]$ to $[0,1]$, and where e consists of non-negative real constants, the variable x, additions, multiplications, and applications of f. We abstract f to a sequence of real numbers $(r_0, \ldots, r_n) \in [0,1]^{n+1}$, where r_i represents the value of $f(\frac{i}{n})$. Thus, the abstraction function α mapping any function $f : [0,1] \to [0,1]$ to its abstract form $[0,1]^{n+1}$ is defined by

$$\alpha(f) = \left(f\left(\frac{0}{n}\right), f\left(\frac{1}{n}\right), \ldots, f\left(\frac{n}{n}\right) \right).$$

We write γ for any concretization function, mapping any element of $[0,1]^{n+1}$ back to a function in $[0,1] \to [0,1]$. The idea here is that *if γ satisfies certain assumptions*, to be given later in Lemma 5.2, then we can obtain an upper-bound of the least solution of $f = \mathcal{F}(f)$ by solving the following system of inequalities on the real numbers $\vec{r} = (r_0, \ldots, r_n)$:

$$\vec{r} \geq \alpha(\mathcal{F}(\gamma(\vec{r}))). \tag{5.2}$$

Let $\widehat{\mathcal{F}}$ be the functional $\lambda\!\!\lambda \vec{s}.\alpha(\mathcal{F}(\gamma(\vec{s})))$. Notice that solutions to (5.2) are precisely the pre-fixpoints of $\widehat{\mathcal{F}}$, and we will thus call them *abstract pre-fixpoints* of $\mathcal{F}$.

There are at least two degrees of freedom here:

(1) **How could we define the concretization function?** Here we have at least two choices (see Figure 6):
   (a) $\gamma(\vec{r})$ could be the *step function* $\hat{f}$ such that $\hat{f}(0) = r_0$ and $\hat{f}(x) = r_i$ if $x \in (\frac{i-1}{n}, \frac{i}{n}]$.
   (b) $\gamma(\vec{r})$ could be the *piecewise linear function* $\hat{f}$ such that $\hat{f}(x) = r_i + \frac{x - \frac{i}{n}}{\frac{1}{n}}(r_{i+1} - r_i) (= (i + 1 - nx)r_i + (nx - i)r_{i+1})$ if $x \in [\frac{i}{n}, \frac{i+1}{n}]$.

   The discussion above on abstract pre-fixpoints suggests that it is natural to require that (α, γ) satisfies a Galois connection-like property. The first choice indeed turns (α, γ) into a Galois connection between the set of monotonic functions and that of non-decreasing sequences of real numbers. The second choice of (α, γ) is not exactly a Galois connection (because $\alpha(f) \leq \vec{r}$ does not imply $f \leq \gamma(\vec{r})$ if f is not convex), but is quite close: if an abstraction $\vec{r}$ majorizes $\alpha(f)$ for some pointwise convex function f, we immediately have $\gamma(\vec{r}) \geq f$. This way, γ satisfies the assumption of Lemma 5.2 below.

(2) **How could we solve inequalities?** Again, we have at least two choices.
   (c) Use the decidability of *theories of real arithmetic* (e.g., minimize $\sum_i r_i$ so that all the inequalities are satisfied).
   (d) *Abstract* also the values of $\vec{r}$ so that they can take only finitely many discrete values, say, $0, \frac{1}{m}, \ldots, \frac{m-1}{m}, 1$. The inequality (5.2) is then replaced by:

$$\vec{s} \geq \alpha_h(\alpha(\mathcal{F}(\gamma(\vec{s})))),$$

   where every s_i is the "discretized version" of r_i, and the abstraction function α_h, given a tuple of reals as an input, replaces each element $r \in [0,1]$ with $\frac{\lceil rm \rceil}{m}$. Since they are now inequalities over a *finite* domain, we can obtain the least solution by a *finite* number of Kleene iterations, starting from $\vec{s} = \vec{0}$.

The following lemma ensures that the inequality (5.2) is indeed a sufficient condition for $\gamma(\vec{r})$ to be an upper-bound on $\mathbf{lfp}(\mathcal{F})$. Note that both step functions and stepwise linear functions satisfy the assumption of the lemma below.

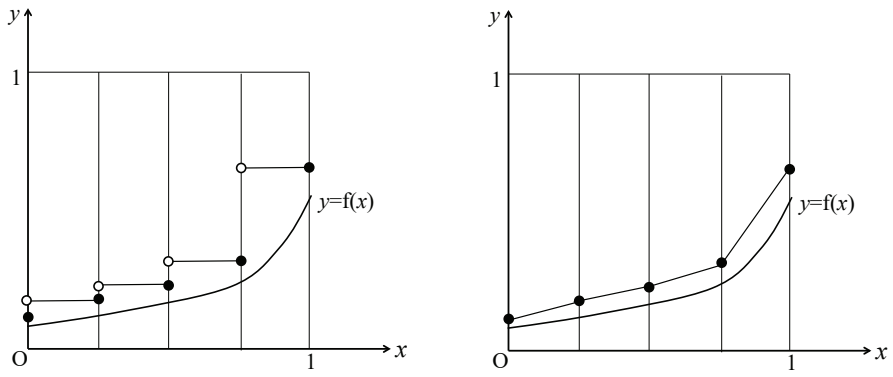

FIGURE 6. Overapproximation by a step-function (left) and a stepwise linear function (right)

Lemma 5.2. *Suppose that the concretization function γ is monotonic, and that $\vec{r} \geq \alpha(f)$ implies $\gamma(\vec{r}) \geq f$ for every pointwise convex f. Then, any abstract pre-fixpoint of $\mathcal{F}$ is an upper bound of* $\mathbf{lfp}(\mathcal{F})$.

Proof. First, we show that $\gamma(\widehat{\mathcal{F}}^m(\bot)) \geq \mathcal{F}^m(\bot)$ holds for every m, by induction on m. The base case $m = 0$ is trivial, since $\mathcal{F}^0(\bot) = \bot$. If $m > 0$, then we have

$$\widehat{\mathcal{F}}^m(\bot) = \alpha(\mathcal{F}(\gamma(\widehat{\mathcal{F}}^{m-1}(\bot)))) \geq \alpha(\mathcal{F}(\mathcal{F}^{m-1}(\bot))) = \alpha(\mathcal{F}^m(\bot)).$$

Since, by hypothesis, $\vec{r} \geq \alpha(\mathcal{F}^m(\bot))$ implies that $\gamma(\vec{r}) \geq \mathcal{F}^m(\bot)$ (since $\mathcal{F}^m(\bot)$ is pointwise convex), we can conclude that $\gamma(\widehat{\mathcal{F}}^m(\bot)) \geq \mathcal{F}^m(\bot)$. Now, suppose $\vec{r}$ is an abstract pre-fixpoint of $\mathcal{F}$. Then $\gamma(\vec{r}) \geq \gamma(\widehat{\mathcal{F}}^m(\vec{r})) \geq \gamma(\widehat{\mathcal{F}}^m(\bot)) \geq \mathcal{F}^m(\bot)$, and as $\mathcal{F}$ is ω-continuous, we have $\gamma(\vec{r}) \geq \sup_{m\in\omega} \mathcal{F}^m(\bot) = \mathbf{lpf}(\mathcal{F})$ as required. □

Below we consider the combination of (b) and (d). Figure 7 shows a pseudo code for computing an upper-bound of $f(c)$ for the least solution f of $f(x) = e$ and $c \in [0, 1]$. In the figure, $\alpha_h(x) = \frac{\lceil mx \rceil}{m}$. The algorithm terminates under the assumptions that (i) e consists of non-negative constants, x, $+$, $\cdot$, and applications of f, and (ii) every subexpression of e evaluates to a value in $[0, 1]$ (if $x \in [0, 1]$ and $f \in [0, 1] \to [0, 1]$), which are satisfied by the fixpoint equations obtained from a PHORS (recall Section 5.1).

Example 5.3. Consider $f(x) = \frac{1}{4}x + \frac{3}{4}f(f(x))$ and let $n = 2$ and $m = 4$. The value $r^{(j)}$ of r after the j-th iteration is given by:

$$r^{(0)} = [0, 0, 0]; \qquad r^{(1)} = [0, 0.25, 0.25]; \qquad r^{(2)} = [0, 0.25, 0.5]; \qquad r^{(3)} = [0, 0.25, 0.5].$$

Thus, the upper-bound obtained for $f(1)$ is 0.5. The exact value of $f(1) = \frac{1}{3}$. A more precise upper-bound is obtained by increasing the values of n and m. For example, if $n = 16$ and $m = 256$, the upper-bound (obtained by running the tool reported in a later section) is $0.3398\cdots$. □

```
main(e, c){
  r := [0,...,0];  r' := [1,...,1] (* dummy *);
  while not(r=r') do {
    r' := r; (* copy the contents of array r to r' *)
    for each i ∈ {0,...,n} do r'[i] := α_h(eval(e, {f ↦ r, x ↦ i/n}))};
  return apply(r, c); }
apply(r, c) { (* apply the function represented by array r to c *)
  i := ⌊nc⌋; (* i/n ≤ c < (i+1)/n *)
  return (i+1−nc)r[i] + (nc−i)r[i+1]; }
eval(e, ρ){
  match e with
     x → return ρ(x) | c → return c
   | f(e') → return apply(ρ(f), eval(e', ρ))
   | e1 + e2 → return eval(e1,ρ)+eval(e2,ρ)
   | e1 · e2 → return eval(e1,ρ)·eval(e2,ρ) }
```

FIGURE 7. Pseudo code for computing an upper-bound of $f(c)$ where $f(x) = e$ (unary function case)

5.2.2. *Computing an upper-bound for a binary function.* We now consider a fixpoint equation of the form $f(x_1, x_2) = e$, where x_1 and x_2 are such that $0 \leq x_1, x_2$, and $x_1 + x_2 \leq 1$. Such an equation is obtained from an order-2 PHORS by using the fixpoint characterization in the previous section. A new difficulty compared with the unary case is that $f(x_1, x_2)$ may take a value outside $[0, 1]$, or may even be undefined for $(x_1, x_2) \in [0, 1] \times [0, 1]$ such that $x_1 + x_2 > 1$. Figure 8 shows how we discretize the domain of f. The grey-colored and red-colored areas show the valid domain of f, for which we wish to approximate $f(x_1, x_2)$ using the values at discrete points. An upper-bound of the value of f at a point (x_1, x_2) in the grey area can be obtained by (pointwise) linear interpolations from (upper-bounds of) the values at the surrounding four points, i.e., $(\frac{i_1}{n}, \frac{i_2}{n}), (\frac{i_1}{n}, \frac{i_2+1}{n}), (\frac{i_1+1}{n}, \frac{i_2}{n}), (\frac{i_1+1}{n}, \frac{i_2+1}{n})$ where $x_1 \in [\frac{i_1}{n}, \frac{i_1+1}{n}]$ and $x_2 \in [\frac{i_2}{n}, \frac{i_2+1}{n}]$, as follows.

$$\begin{aligned}
&\hat{f}(x_1, x_2)\\
&= (i_2 + 1 - nx_2)\hat{f}(x_1, \tfrac{i_2}{n}) + (nx_2 - i_2)\hat{f}(x_1, \tfrac{i_2+1}{n})\\
&= (i_2 + 1 - nx_2)(i_1 + 1 - nx_1)\hat{f}(\tfrac{i_1}{n}, \tfrac{i_2}{n})\\
&\quad +(i_2 + 1 - nx_2)(nx_1 - i_1)\hat{f}(\tfrac{i_1+1}{n}, \tfrac{i_2}{n})\\
&\quad +(nx_2 - i_2)(i_1 + 1 - nx_1)\hat{f}(\tfrac{i_1}{n}, \tfrac{i_2+1}{n})\\
&\quad +(nx_2 - i_2)(nx_1 - i_1)\hat{f}(\tfrac{i_1+1}{n}, \tfrac{i_2+1}{n}).
\end{aligned}$$

Note that $\hat{f}(x, y) \geq f(x, y)$ at the four points imply that $\hat{f}(x_1, x_2) \geq f(x_1, x_2)$, because $f(x, y)$ is convex on each of x and y (recall Section 5.1).

A difficulty is that to estimate the value of f at a point in the red area, we need the value at a red point $\Diamond$, but the value of f at the red point may be greater than 1 or even ∞, being outside f's domain. To this end, we discretize the codomain of f to $\{0, \frac{1}{m}, \ldots, \frac{mh-1}{m}, h, \infty\}$ (instead of $\{0, \frac{1}{m}, \ldots, \frac{m-1}{m}, 1\}$) for some $h \geq 1$. Any value greater than h is approximated to ∞. The value at a point in the red area is then approximated in the same way as for the case of a point in the grey area, except that if $\hat{f}(\frac{i_1+1}{m}, \frac{i_2+1}{m}) = \infty$, then $\hat{f}(x_1, x_2) = 1$.

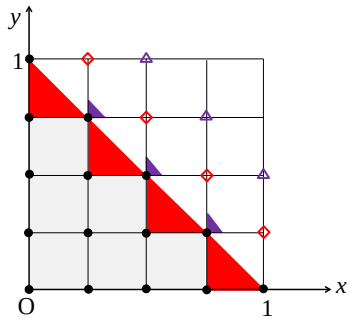

FIGURE 8. Discretization in the case of a binary function $f(x,y)$, whose domain is $\{(x,y) \mid 0 \leq x, 0 \leq y, x+y \leq 1\}$ (i.e., the grey and red areas). Outside the domain, the value of $f(x,y)$ may not belong to $[0,1]$, or may be even undefined. The value at a point in the grey area can be estimated by using those at discrete points $\bullet$. To estimate the value at a point in the red area, we also need the value of $f(x,y)$ at a point marked by $\Diamond$.

A further complication arises when the equation contains function compositions, as in $f(x_1,x_2) = E[f(f(x_1,x_2),x_2)]$ where E denotes some context. In this case, the point $(\hat{f}(x_1,x_2),x_2)$ may even be outside the area surrounded by $\Diamond$ and $\bullet$-points either if (x_1,x_2) is a $\Diamond$-point or if (x_1,x_2) is a $\bullet$-point but $\hat{f}(x_1,x_2)+x_2$ is too large due to an overapproximation. In such a case, $(\hat{f}(x_1,x_2),x_2)$ belongs to the purple area (lower left triangles) in the figure. To this end, we also compute (upper-bounds of) the values at points marked by $\triangle$ and use them to estimate the value at a point in the purple area. If the point $(\hat{f}(x_1,x_2),x_2)$ is even outside the area surrounded by $\bullet$, $\Diamond$, or $\triangle$, then we use ∞ as an upper-bound of $f(f(x_1,x_2),x_2)$ if (x_1,x_2) is a $\Diamond$-, or $\triangle$-point, and 1 if (x_1,x_2) is a $\bullet$-point.

Except the above differences, the overall algorithm is similar to the unary case in Figure 7, and essentially the same soundness argument as Lemma 5.2 applies.

Example 5.4. Consider $f(x_1,x_2) = x_1 + x_2(f(x_1,x_2))^2$, and let $n = m = 2$. The value of $r = \begin{pmatrix} r_{0,2} & r_{1,2} & r_{2,2} \\ r_{0,1} & r_{1,1} & r_{2,1} \\ r_{0,0} & r_{1,0} & r_{2,0} \end{pmatrix}$, where $r_{i,j}$ is an upper-bound of the value of $f(\frac{i}{2},\frac{j}{2})$, changes as follows.

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & 1 & \infty \\ 0 & 1 & \infty \\ 0 & 0.5 & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & \infty & \infty \\ 0 & 1 & \infty \\ 0 & 0.5 & 1 \end{pmatrix}$$

Thus, for example, $f(0,0.5)$ and $f(0.3,0.3)$ are overapproximated respectively by 0 and $0.36\hat{f}(0.5,0.5)+0.24\hat{f}(0.5,0)+0.24\hat{f}(0,0.5)+0.16\hat{f}(0,0) = 0.48$. The exact values for $f(0,0)$ and $f(0.3,0.3)$ are 0 and $\frac{1}{3}$; so the upper-bound for $f(0.3,0.3)$ is sound but imprecise. By choosing $n = 16$ and $m = 256$, we obtain $0.3359\cdots$ as an upper-bound of $f(0.3,0.3)$.

This is an example where the values of f at red points in Figure 8 are ∞. The exact value of $f(x_1,x_2)$ for general x_1 and x_2 is given by:

$$f(x_1,x_2) = \begin{cases} x_1 & \text{if } x_2 = 0 \\ \frac{1-\sqrt{1-4x_1x_2}}{2x_2} & \text{if } x_1 > 0 \end{cases}$$

Thus, $f(0.5, 0.5) = 1$, but $f(x_1, x_2)$ is undefined whenever $x_1 x_2 > 0.25$; in particular $f(0.5, 1)$ and $f(1, 0.5)$ (the values of red points in Figure 8 for the case $n = 2$) are undefined. □

5.2.3. *Computing an Upper-Bound: General Case.* The binary case discussed above can be easily extended to handle the general case, where the goal is to estimate the value of $f_1(c_1, \ldots, c_{\ell_1})$ for the least solution of the fixpoint equations:

$$f_1(x_1, \ldots, x_{\ell_1}) = e_1 \qquad \cdots \qquad f_k(x_1, \ldots, x_{\ell_k}) = e_k.$$

Here, the formal arguments $x_1, \ldots, x_{\ell_i}$ of each function f_i are partitioned into several groups $(x_1, \ldots, x_{d_{i,1}}), (x_{d_{i,1}+1}, \ldots, x_{d_{i,2}}), \ldots, (x_{d_{g_i-1}+1}, \ldots, x_{\ell_i})$, so that the sum of the values of the variables in each group ranges over $[0, 1]$. Following the binary case, we discretize the domain so that each variable ranges over $\{0, \frac{1}{n}, \ldots, \frac{n-1}{n}, 1\}$, where the variables in each group $(x_{d_{i,j-1}+1}, \ldots, x_{d_{i,j}})$ are constrained by $x_{d_{i,j-1}+1} + \cdots + x_{d_{i,j}} \leq \frac{n+2}{n}$. Note that we choose $\frac{n+2}{n}$ instead of 1 as the upper-bound of the sum, to include the points ♢ and △ in Figure 8. We write D_i for the discretized domain of function f_i, and D'_i for the subset of D_i where the variables in each group are constrained by $x_{d_{i,j-1}+1} + \cdots + x_{d_{i,j}} \leq 1$; note that $f_i(x_1, \ldots, x_{\ell_i}) \in [0, 1]$ for $(x_1, \ldots, x_{\ell_i}) \in D'_i$, but $f_i(x_1, \ldots, x_{\ell_i})$ may be greater than 1 or undefined for $(x_1, \ldots, x_{\ell_i}) \in D_i \setminus D'_i$. We also write $\overline{D_i}$ for the set $\{(x_1, \ldots, x_{\ell_i}) \mid (\lceil nx_1 \rceil / n, \ldots, \lceil nx_{\ell_i} \rceil / n) \in D_i\}$ (i.e., the set of points for which the value of f_i can be approximated by using values at points in D_i).

The pseudo code for computing an upper-bound of $f_1(c_1, \ldots, c_{\ell_1})$ is given in Figure 9. On the 9th line ("`if` $\vec{v} \in D'_i$ `then ...`"), we also make use of the constraint that $\Sigma_{f' \in \mathtt{fgrp}(f_i)} f'(\widetilde{v})$ ranges over $[0, 1]$ if $\widetilde{v}$ belongs to the valid domain D'_i (recall the 3rd property in Section 5.1). We assume that the procedure $\mathtt{lb}(f', \widetilde{v})$ returns a sound lower-bound of $f'(\widetilde{v})$, e.g., by using Kleene iteration. See Remark 5.5 to understand the need for this additional twist.

The function α_h takes a real value (or ∞) x, and returns the least element in $\{0, \frac{1}{n}, \ldots, \frac{mh-1}{m}, h, \infty\}$ that is no less than x. The function `apply` in the figure takes the current approximations of values of f_i at the points D_i and the arguments $\vec{v} \in \overline{D_i}$, and returns an approximation of $f_i(\vec{v})$. It is given by $\hat{f}(\vec{v})$, where:

$$\hat{f}_i(x_1, \ldots, x_{\ell_i}) = \textstyle\sum_{b_1, \ldots, b_{\ell_i} \in \{0,1\}} p_1^{b_1} (1 - p_1)^{1 - b_1} \cdots p_{\ell_i}^{b_{\ell_i}} (1 - p_{\ell_i})^{1 - b_{\ell_i}} \hat{f}_i \left(\tfrac{i_1 + b_1}{n}, \ldots, \tfrac{i_{\ell_i} + b_{\ell_i}}{n} \right)$$

Here, $i_j = \lfloor nx_j \rfloor$, $p_j = nx - i_j$, and $\hat{f}_i(\frac{i_1 + b_1}{n}, \ldots, \frac{i_m + b_m}{n})$ is the current approximation of the value of f_i at $(\frac{i_1 + b_1}{n}, \ldots, \frac{i_m + b_m}{n}) \in D_i$. The function $\hat{f}$ above is obtained by applying linear interpolations coordinate-wise.

Remark 5.5. To see the motivation for the 9th line in Figure 9, consider the following fixpoint equations:

$$\begin{aligned} S &= f_1() \\ f_1() &= 0.5 \cdot (f_1() \cdot f_1() + f_2() \cdot f_2()) \\ f_2() &= 0.5 + f_1() \cdot f_2(). \end{aligned}$$

They are obtained from the following order-1 PHORS $\mathcal{G}_{\mathtt{treeeven}}$:

$$\begin{aligned} S\, z &= F\, z\, \Omega \\ F\, x_1\, x_2 &= x_2 \oplus_p F\, (F\, x_1\, x_2)\, (F\, x_2\, x_1), \end{aligned}$$

```
main(e_1, ..., e_k, c⃗){
  ρ := [f_1 ↦ [0⃗], ..., f_k ↦ [0⃗]];
      (* ρ(f_i) is an array indexed by each element of D_i *)
  ρ' := [f_1 ↦ [1⃗], ..., f_k ↦ [1⃗]]; (* dummy *)
  while not(ρ=ρ') do {
    ρ' := ρ; (* copy the contents *)
    for each i ∈ {0, ..., k} do
       for each v⃗ ∈ D_i do
          let r = eval(e_i, ρ{x⃗ ↦ v⃗}, v⃗ ∈? D'_i) in
            if v⃗ ∈ D'_i then ρ'(f_i)[v⃗] := α_h(min(r, 1 − Σ_{f'∈fgrp(f_i)\{f}} lb(f', v⃗)))
            else ρ'(f_i)[v⃗] := α_h(r);
  return apply(ρ(f_1), c⃗); }

eval(e, ρ, b){
 (* b represents whether we are computing the value of f_i in the valid
    domain; in that case, the value of e should range over [0, 1]. *)
  let r =
    match e with
       x → ρ(x) | c → c
     | f_i(e⃗') → let v⃗ = eval(e⃗',ρ,b) in
                  if v⃗ ∉ D_i then ∞ else apply(ρ(f_i), v⃗)
     | e_1 + e_2 → eval(e_1, ρ, b)+eval(e_2, ρ, b)
     | e_1 · e_2 → eval(e_1, ρ, b)·eval(e_2, ρ, b)
  in if b then return min(r,1) else return r }
```

FIGURE 9. Pseudo code for computing an upper-bound for the general case

where $p = 0.5$, and $f_1()$ ($f_2()$, resp.) represents the probability that x_1 (x_2, resp.) is used by F. This PHORS is actually a variation of $\mathcal{G}_6$ from Example 2.11 (with manual optimization), whose termination probability represents the probability that a program that randomly generates binary trees (instead of lists, unlike in the case of Example 2.11) contains an even number of leaves. Since the events that F uses the first and second arguments are mutually exclusive, we have the constraint $f_1() + f_2() \leq 1$. The exact solution for the equations above is $f_1() = 1 - \frac{1}{\sqrt{2}}$ and $f_2() = \frac{1}{\sqrt{2}}$. Since their lower-bounds can be computed with arbitrary precision, thanks to the part $1 - \Sigma_{f' \in \mathtt{fgrp}(f_i)\setminus\{f\}}\ \mathtt{lb}(f', \vec{v})$ of the 9th line of Figure 9, we can also compute upper-bounds with arbitrary precision (as upper-bounds of $f_1()$ and $f_2()$ are respectively provided by $1 - \mathtt{lb}(f_2, ())$ and $1 - \mathtt{lb}(f_1, ())$).

If the then-clause were the same as the else-clause on the 10th line, then we would not get a precise upper-bound for the following reason. When the main loop in Figure 9 stops, upper-bounds $\overline{f}_1()$ and $\overline{f}_2()$ must either have reached the maximal value 1, or satisfy:

$$\overline{f}_1() \geq 0.5 \cdot (\overline{f}_1() \cdot \overline{f}_1() + \overline{f}_2() \cdot \overline{f}_2())$$
$$\overline{f}_2() \geq 0.5 + \overline{f}_1() \cdot \overline{f}_2().$$

These conditions imply that:

$$\overline{f}_1() + \overline{f}_2() \geq 0.5 \cdot (\overline{f}_1() \cdot \overline{f}_1() + \overline{f}_2() \cdot \overline{f}_2()) + 0.5 + \overline{f}_1() \cdot \overline{f}_2(),$$

i.e.,

$$0 \geq (\overline{f}_1() + \overline{f}_2() - 1)^2,$$

which is equivalent to $\overline{f}_1() + \overline{f}_2() = 1$. Thus, unless the co-domain of α_h contains the exact values $1 - \frac{1}{\sqrt{2}}$ and $\frac{1}{\sqrt{2}}$, the main loop would only return the imprecise upper-bound $\overline{f}_1() = \overline{f}_2() = 1$. □

5.3. **Order-n Case.** We now briefly discuss how to extend the method discussed above to obtain a sound (but incomplete) method for overapproximating the termination probability of PHORS of order greater than 2. Recall that by the fixpoint characterization given in Section 4.3, it suffices to overapproximate the least solution of equations of the form $\vec{f} = \mathcal{F}(\vec{f})$ where $\vec{f}$ is a tuple of order-$(n-1)$ functions on reals.

The abstract interpretation framework [Cou97] provides a sound but incomplete methodology: the reason why we decided to slightly divert from it in Section 5.2 is that this allows us to use piecewise linear functions, which are more precise. We first recall a basic principle of abstract interpretation. Let $(C, \sqsubseteq_C, \bot_C)$ and $(A, \sqsubseteq_A, \bot_A)$ be ω-cpos. Suppose that $\alpha : (C, \sqsubseteq_C) \to (A, \sqsubseteq_A)$ and $\gamma : (A, \sqsubseteq_A) \to (C, \sqsubseteq_C)$ are continuous (hence also monotonic) such that $\alpha(\gamma(a)) = a$ for every $a \in A$, and $c \sqsubseteq_C \gamma(\alpha(c))$ for every $c \in C$. Suppose also that $\mathcal{F}$ is a continuous function from $(C, \sqsubseteq_C, \bot_C)$ to $(C, \sqsubseteq_C, \bot_C)$. Let $\widehat{\mathcal{F}} : (A, \sqsubseteq_A, \bot_A) \to (A, \sqsubseteq_A, \bot_A)$ be $\lambda x \in A.\alpha(\mathcal{F}(\gamma(x)))$, which is an "abstract version" of $\mathcal{F}$. Note that $\widehat{\mathcal{F}}$ is also continuous. Then, we have:

Proposition 5.6. $\mathbf{lfp}(\mathcal{F}) \sqsubseteq_C \gamma(\mathbf{lfp}(\widehat{\mathcal{F}}))$.

This result is standard (see, e.g., [Cou97], Proposition 18) but we provide a proof for the convenience of the reader.

Proof of Proposition 5.6. By the monotonicity of $\mathcal{F}$ and $\widehat{\mathcal{F}}$, we have: $\bot_C \sqsubseteq_C \mathcal{F}(\bot_C) \sqsubseteq_C \mathcal{F}^2(\bot_C) \sqsubseteq_C \cdots$ and $\bot_A \sqsubseteq_A \widehat{\mathcal{F}}(\bot_A) \sqsubseteq_A \widehat{\mathcal{F}}^2(\bot_A) \sqsubseteq_A \cdots$; hence both $\bigsqcup_C \{\mathcal{F}^i(\bot_C) \mid i \in \omega\}$ and $\bigsqcup_A \{\widehat{\mathcal{F}}^i(\bot_A) \mid i \in \omega\}$ exist, and by the ω-continuity of $\mathcal{F}$ and $\widehat{\mathcal{F}}$, they are the least fixpoints of $\mathcal{F}$ and $\widehat{\mathcal{F}}$ respectively. Therefore, it suffices to show that $\mathcal{F}^i(\bot_C) \sqsubseteq_C \gamma(\widehat{\mathcal{F}}^i(\bot_C))$. The proof proceeds by induction on i. The base case $i = 0$ is trivial. If $i > 0$, we have:

$$\begin{aligned}
\gamma(\widehat{\mathcal{F}}^i(\bot_C)) &= \gamma(\alpha(\mathcal{F}(\gamma(\widehat{\mathcal{F}}^{i-1}(\bot_C))))) && \text{(by the definition of } \widehat{\mathcal{F}}) \\
&\sqsupseteq_C \mathcal{F}(\gamma(\widehat{\mathcal{F}}^{i-1}(\bot_C))) && \text{(by } \gamma(\alpha(x)) \sqsupseteq_C x) \\
&\sqsupseteq_C \mathcal{F}(\mathcal{F}^{i-1}(\bot_C)) && \text{(by induction hypothesis)} \\
&= \mathcal{F}^i(\bot_C). &&
\end{aligned}$$

□

By the proposition above, to overapproximate the least fixpoint of $\mathcal{F}$, it suffices to find an appropriate abstract domain $(A, \sqsubseteq_A, \bot_A)$ and α, γ that satisfy the conditions above, so that the least fixpoint of $\widehat{\mathcal{F}}$ is easily computable. In the case of overapproximation of the termination probability of order-n PHORS, we need to set up an abstract domain $(A, \sqsubseteq_A, \bot_A)$ for a tuple of order-$(n-1)$ functions on reals. A simple solution (that is probably too naive in practice) is to use the abstract domain consisting of higher-order step functions,

inductively defined by:

$$
\begin{aligned}
A^{\mathtt{R}} &= \{\frac{0}{m}, \frac{1}{m}, \ldots, \frac{m-1}{m}, \frac{m}{m}, \infty\} \\
\sqsubseteq_{A^{\mathtt{R}}} &= \{(\frac{i}{m}, \frac{j}{m}) \mid 0 \leq i \leq j \leq m\} \cup \{(\frac{i}{m}, \infty) \mid 0 \leq i \leq m\} \\
\alpha^{\mathtt{R}}(x) &= \begin{cases} \frac{i}{m} & \text{if } \frac{i-1}{m} \leq x \leq \frac{i}{m} \\ \infty & \text{if } x > 1 \end{cases} \\
\gamma^{\mathtt{R}}(x) &= x \\
A^{\tau_1 \to \tau_2} &= \{f \in A^{\tau_1} \to A^{\tau_2} \mid f \text{ is monotonic}\} \\
\sqsubseteq_{A^{\tau_1 \to \tau_2}} &= \{(f_1, f_2) \in A^{\tau_1 \to \tau_2} \times A^{\tau_1 \to \tau_2} \mid \forall x \in A^{\tau_1 \to \tau_2}. f_1\, x \sqsubseteq_{A^{\tau_2}} f_2\, x\} \\
\alpha^{\tau_1 \to \tau_2}(f) &= \{(y, \alpha^{\tau_2}(f(\gamma^{\tau_1}(y)))) \mid y \in A^{\tau_1}\} \\
\gamma^{\tau_1 \to \tau_2}(f') &= \{(x, \gamma^{\tau_2}(f'(\alpha^{\tau_1}(x)))) \mid x \in C^{\tau_1}\}.
\end{aligned}
$$

Here, the concrete domain C^τ denotes $[\![\tau]\!]$ in Section 4.1. Then, α^τ and β^τ satisfy the required conditions ($\alpha(\gamma(a)) = a$ and $c \sqsubseteq_C \gamma(\alpha(c))$). Since A^τ is finite, we can effectively compute $\mathbf{lfp}(\widehat{\mathcal{F}})$.

We note, however, that the above approach has the following shortcomings. First, although A^τ is finite, its size is too large: k-fold exponential for order-k type τ. As in the case of non-probabilistic HORS model checking [Kob09a, BK13, RNO14], therefore, we need a practical algorithm that avoids eager enumeration of abstract elements. Second, due to the use of step functions, the obtained upper-bound will be too imprecise. To see why step functions suffer from the incompleteness, consider the equations: $s = f(\frac{1}{2})$ and $f(x) = \frac{1}{2}x + f(\frac{1}{2}x)$. The exact least solution is $s = \frac{1}{2}$ and $f(x) = x$. With step functions (where $\frac{1}{n}$ is the size of each interval), however, the abstract value $\hat{f}(\frac{1}{n})$ must be no less than $\frac{1}{2}\frac{1}{n} + f(\frac{1}{2} \cdot \frac{1}{n})$, but $f(\frac{1}{2} \cdot \frac{1}{n})$ is overapproximated as $\hat{f}(\frac{1}{n})$ (because $\frac{1}{2n}$ belongs to the interval $(0, \frac{1}{n}]$). Therefore, $\hat{f}(\frac{1}{n})$ should be no less than $\frac{1}{2n} + \hat{f}(\frac{1}{n})$, which is impossible. Thus, the computation diverges and 1 is obtained as an obvious upper bound.

The step functions only use monotonicity of the least solution of fixpoint equations. As in the use of stepwise multilinear functions in the case of order-1 equations (for order-2 PHORS), exploiting an additional property like convexity would be important for obtaining a more precise method; this is left for future work.

6. Experiments

We have implemented a prototype tool to compute lower/upper bounds of the least solution of order-1 fixpoint equations (that are supposed to have been obtained from order-2 or order-1 PHORS by using the translations in Section 4 modulo some simplifications; we have not yet implemented the translators from PHORS to fixpoint equations, which is easy but tedious). The computation of a lower bound is based on naive Kleene iterations, and that of an upper-bound is based on the method discussed in Section 5.2. The tool uses floating point arithmetic, and ignores rounding errors.

We have tested the tool on several small but tricky examples. The experimental results are summarized in Table 2. The column "equations" lists the names of systems of equations. The column "#iter" shows the number of Kleene iterations used for computing a lower-bound. The columns "#dom" and "#codom" show the numbers of partitions of the interval $[0, 1]$

TABLE 2. Experimental results (times are in seconds).

equations	#iter	#dom	#codom	l.b.	u.b.	step	exact	time
Ex2.3-1	12	16	512	0.333	0.336	1.0	$\frac{1}{3}$	0.010
Ex2.3-0	12	16	512	0.333	0.334	0.334	$\frac{1}{3}$	0.008
Ex2.3-v1	12	16	512	0.312	0.315	0.365	-	0.005
Ex2.3-v2	12	16	512	0.262	0.266	0.321	-	0.022
Ex2.3-v3	12	16	512	0.263	0.266	0.309	-	0.01
Ex2.4	12	16	512	0.320	0.323	0.329	-	0.011
Double	12	16	512	0.649	0.653	1.0	-	0.010
Listgen	15	16	512	0.999	1.0	1.0	1.0	0.009
Treegen	15	64	4096	0.618	0.619	1.0	$\frac{\sqrt{5}-1}{2}$	0.471
Treegenp	12	16	512	1.0	1.0	1.0	1.0	0.011
ListEven	12	32	1024	0.666	0.667	0.667	$\frac{2}{3}$	0.009
ListEven2	12	16	512	0.749	0.75	0.75	$\frac{3}{4}$	0.013
Determinize	12	16	512	0.993	1.0	1.0	1.0	9.64
TreeEven(0.5)	15	64	4096	0.286	0.299	0.300	$1-\frac{1}{\sqrt{2}}$	0.050
TreeEven(0.49)	15	64	4096	0.276	0.280	0.280	$0.2774\cdots$	0.052
TreeEven(0.51)	15	64	4096	0.287	0.290	0.290	$0.2887\cdots$	0.055
Ex5.4(0,0)	12	16	512	0.0	0.0	0.0	0	0.008
Ex5.4(0.3,0.3)	12	16	512	0.333	0.336	0.35	$\frac{1}{3}$	0.007
Ex5.4(0.5,0.5)	10000	16	512	0.999	1.0	1.0	1	0.010
Discont(0,1)	12	16	512	0.0	0.0	0.0	0	0.006
Discont(0.01,0.99)	1000	16	512	0.999	1.0	1.0	1	0.006
Incomp	10000	16	512	0.299	1.0	1.0	0.3	0.003
Incomp	10000	10	100	0.299	0.3	0.3	0.3	0.003
Incomp2	12	16	512	0.249	1.0	1.0	0.25	0.003
Incomp2	12	256	65536	0.249	1.0	1.0	0.25	2.87

for the domain and codomain of a function respectively. The default values for them were set to 12, 16, and 512, respectively in the experiment; they were, however, adjusted for some of the equations. The columns "l.b." and "u.b." are lower/upper bounds computed by the tool. The lower (upper, resp.) bounds shown in the table have been obtained by rounding down (up, resp.) the outputs of the tool to 3-decimal places. The column "step" shows the upper-bounds obtained by using step functions instead of piecewise linear functions; this column has been prepared to confirm the advantage of piecewise linear functions over step functions. The column "exact" shows the exact value of the least solution when we know it. The column "time" shows the total time for computing both lower and upper bounds (excluding the time for "step").

The equations "Ex2.3-1" and "Ex2.3-0" are order-1 and order-0 equations obtained from the PHORS in Example 2.3 (see also Examples 4.5 and 4.7) by using the translations in Sections 4.2 and 4.3 respectively; specifically, "Ex2.3-1" consists of $s = f(1)$ and $f(x) = 0.25x + 0.75f(f(x))$. The equations "Ex2.3-v1", "Ex2.3-v2", and "Ex2.3-v3" are variations of them, where the equation on f is replaced by $f(x) = 0.25x + 0.75f(f(x^2))$, $f(x) = 0.25x + 0.75f(f(f(x^2)))$, and $f(x) = 0.25x + 0.75(f(x))^2$, respectively. "Ex2.4" is the

equations obtained from the order-2 PHORS in Example 2.4 (see also Example 4.8). The equations "Double" are those obtained from the following order-2 PHORS:

$$\begin{aligned} S &= F\,H \\ H\,x\,y &= x \oplus_{\frac{1}{2}} y \\ F\,g &= g\,\mathsf{e}\,(F(D\,g)) \\ D\,g\,x\,y &= g\,(g\,x\,y)\,y, \end{aligned}$$

with manual simplifications. The equations "Listgen", "Treegen", and "Treegenp" are from Example 2.9, corresponding to $\mathcal{G}_3$, $\mathcal{G}_4$ and $\mathcal{G}_5$, respectively. The equations "ListEven" and "ListEven2" are from Example 2.11, and "Determinize" is from Example 2.12. "TreeEven(p)" (for $p \in \{0.5, 0.49, 0.51\}$) is from Remark 5.5. If we disable the trick (the one on line 9 in Figure 7) we discussed in the remark, the tool returns an imprecise upper-bound of 1.0 for $p = 0.5$ (for $p = 0.49$ and $p = 0.51$, however, the tool can compute a precise upper-bound even without the trick). The equations "Ex5.4(x,y)" (for $(x, y) \in \{(0, 0), (0.3, 0.3), (0.5, 0.5)\}$) are from Example 5.4. The equations "Discont(p,$1-p$)" consist of: $s = f(p, 1-p)$ and $f(x_0, x_1) = x_0 + x_1 f(x_0, x_1)$, which is obtained from PHORS:

$$S = F\,G \qquad F\,g = g\,\mathsf{e}\,(F\,g) \qquad G\,x_0\,x_1 = x_0 \oplus_p x_1.$$

Interestingly, f is discontinuous at $(0, 1)$ (in the usual sense of analysis in mathematics; it is still ω-continuous as functions on ω-cpo's): the exact value of f is given by:

$$f(x_0, x_1) = \begin{cases} 0 & \text{if } x_0 = 0 \\ \frac{x_0}{1-x_1} & \text{if } x_0 > 0. \end{cases}$$

The equations "Incomp" and "Incomp2" consist of:

$$s = f(s) \qquad f\,x = x^2 + 0.4x + 0.09,$$

and

$$s = f(s) \qquad f\,x = 0.5x^2 + 2f(0.5x)$$

respectively. They do not correspond to any PHORS — in fact, the value of $f(1)$ for Incomp is 1.49, which does not make sense as a probability. We have included them since they show a source of the possible incompleteness of our method. Indeed, the tool fails to find precise upper bounds. To see why the tool does not work for Incomp1 (with the the default values of #dom and #codom), note that since $s = f(s) = s^2 + 0.4s + 0.09$, $\hat{s} \geq \hat{s}^2 + 0.4\hat{s} + 0.09$ must be satisfied for any valid upper-bound $\hat{s}$. However, $\hat{s} \geq \hat{s}^2 + 0.4\hat{s} + 0.09$ is equivalent to $0 \geq (\hat{s} - 0.3)^2$, which is satisfied only by $\hat{s} = 0.3$. So, the only valid upper-bound for s is actually the exact one 0.3. But then an upper-bound $\hat{f}$ of f must satisfy $\hat{f}(0.3) = 0.3$, which can be found only if the set of discrete values (used for abstracting the domain and codomain) contains 0.3. That is why the tool returns 1 (which is the largest value, assuming that s represents a probability) for the default values of #dom and #codom. When we adjust them to 10 and 100 (so that 0.3 belongs to the sets of abstract values of domains and codomains), the precise upper-bound (i.e., 0.3) is obtained; this is, however, impossible in general, without knowing the exact solution a priori.

The reason for "Incomp2" is more subtle. Notice that the least solution for

$$s = f(s) \qquad f\,x = 0.5x^2 + 2f(0.5x)$$

is $f(x) = x^2$. Let $\frac{1}{n}$ be the size of each interval used for abstracting the domain. Suppose that, at some point, an upper-bound of $f(\frac{1}{n})$ becomes $\frac{c}{n^2}$. Due to the linear interpolation (and

since the value at $x = 0$ converges to 0), the value of f at $0.5r$ (which belongs to the domain $(0, \frac{1}{n})$) is overapproximated by $0.5 \cdot \frac{c}{n^2}$. Thus, at the next iteration, the upper-bound at $\frac{1}{n}$ is further updated to a value greater than $0.5\frac{1}{n^2} + 2 \cdot 0.5 \cdot \frac{c}{n^2} = \frac{c+0.5}{n^2}$. Thus, the computation of an upper-bound for the value at $x = \frac{1}{n}$ never converges. In this case, changing the parameters #dom and #codom does not help. We do not know, however, whether such situations occur in the fixpoint equations that arise from actual order-2 PHORS; it is left for future work to see whether our method (or a minor modification of it) is actually complete (in the sense that upper-bounds can always be computed with arbitrary precision by increasing the parameters #dom and #codom).

To summarize, for all the *valid* inputs (i.e., except "Incomp" and "Incomp2", which are invalid in the sense that they do not correspond to PHORS), our tool (with piecewise linear functions) could properly compute lower/upper bounds. In contrast, from the column "step", we can observe that the replacement of piecewise (multi)linear functions with step functions not only worsens the precision (as in "Ex2.3-v1", "Ex2.3-v2", and "Ex2.3-v3") significantly, but also makes the procedure obviously incomplete[5], as in "Ex2.3-1" and "Double" (recall the discussion on the incompleteness of step functions in Section 5.3).

7. Related Work

As already mentioned in Section 1, this work is intimately related to both probabilistic model checking, and higher-order model checking. Let us give some hints on *how* our work is related to the two aforementioned research areas, without any hope to be exhaustive.

Model checking of probabilistic recursive systems. Model checking of probabilistic systems with *recursion* (but not higher-order functions), such as recursive Markov chains and probabilistic pushdown systems, has been actively studied [EY09, EY15, BBFK14]. Our PHORS are strictly more expressive than those models, as witnessed by the undecidability result from Section 3, and the encoding of recursive Markov chains into order-1 PHORSs given in Appendix A.1. Our fixpoint characterization of the termination probability of PHORS is a generalization of the fixpoint characterization of the termination probability for recursive Markov models [EY09] to arbitrary orders. Various methods have been studied for solving the order-0 fixpoint equations (or, polynomial equations) obtained from recursive Markov chains [EY09, KLE07, EGK10]. Interestingly, also in those methods, computing an upper-bound of the least solution is more involved than computing a lower-bound. It is left for future work to investigate whether some of the ideas in those methods can be used also for solving order-1 fixpoint equations.

Termination of probabilistic infinite-data programs. Methods for computing the termination probabilities of infinite-data programs (with real-valued variables, but without higher-order recursion) have also been actively studied, mainly in the realm of imperative programs (see, as an example, [BG05, EGK12, FH15, CNZ17, MMKK18, ADLY18, CS13]); to the best of our knowledge, none of those methods deal with higher-order programs, at least directly. All these pieces of work present sound but *incomplete* methodologies for checking almost sure termination of programs. Incompleteness is of course inevitable due to the Turing completeness of the underlying language considered. In fact, Kaminsiki and Katoen [KK15] have shown that almost sure termination of probabilistic imperative

[5]As already mentioned, our method with piecewise linear functions may also be incomplete, but that does not show up in the current benchmark set.

programs is Π^0_2-complete. Since their proof relies on Turing completeness of the underlying language, it does not apply to the setting of our model PHORS, which is a probabilistic extension of a Turing-*incomplete* language, namely that of HORS.

Model checking of higher-order programs. Model checking of (non-probabilistic) higher-order programs has been an active topic of research in the last fifteen years, with many positive results [KNU02, Ong06, HMOS08, Kob13, KO09, KSU11, GM15b, GM15a, TO14, SW11, Par18]. Strikingly, not only termination, but also a much larger class of properties (those expressible in the modal μ-calculus) are known to be decidable for ordinary (i.e. non-probabilistic) HORS. This is in stark contrast with our undecidability result from Section 3: already at order-2 and for a very simple property like termination, verification cannot be effectively solved.

Probabilistic functional programs. Probabilistic functional programs have recently attracted the attention of the programming language community, although probabilistic λ-calculi have been known for forty years now [Sah78, JP89]. Most of the work in this field is concerned with operational semantics [DZ12], denotational semantics (see, e.g., [JT98, DH02, ETP14, SYW$^+$16, BFK$^+$18]), or program equivalence (see, e.g., [DSA14, CDL14, SV16]), which sometimes becomes decidable (e.g. [MO05]), but only when higher-order recursion is forbidden. The interest in probabilistic higher-order functional languages stems from their use as a way of writing probabilistic graphical models, as in languages like Church [GMR$^+$08] or Anglican [WMM14]. There are some studies to analyze the termination behavior of probabilistic higher-order programs (with infinite data) by using types. Dal Lago and Grellois [DLG17] generalized sized types [HPS96, BFG$^+$04] to obtain a sound but highly incomplete technique. Breuvart and Dal Lago [BL18] developed systems of intersection types from which the termination probability of higher-order programs can be inferred from (infinitely many) type derivations. This however does *not* lead to any practical verification methodology.

Relevant proof techniques. Our technique (of using the undecidability of Hilbert's 10th problem) for proving the undecidability of almost sure termination of order-2 PHORS has been inspired by Kobayashi's proof of undecidability of the inclusion between order-2 (non-probabilistic) word languages and the Dyck language [Kob19]. Other undecidability results on probabilistic systems include the undecidability of the emptiness of probabilistic automata [GO10]. Their proof is based on the reduction from Post correspondence problem. The technique does not seem applicable to our context.

8. Conclusion

We have introduced PHORS, a probabilistic extension of higher-order recursion schemes, and studied the problem of computing their termination probability. We have shown that almost sure termination is undecidable. As positive results, we have also shown that the termination probability of order-n PHORS can be characterized by order-$(n-1)$ fixpoint equations, which immediately yields a method for computing a precise lower-bound of the termination probability. Based on the fixpoint characterization, we have proposed a sound procedure for computing an upper-bound of the termination probability, which worked well on preliminary experiments.

It is left for future work to settle the question of whether it is possible to compute the termination probability with arbitrary precision, which seems to be a difficult problem.

Another direction of future work is to develop a (sound but incomplete) model checking procedure for PHORS, using the procedure for computing the termination probability as a backend.

Acknowledgments. We would like to thank Kazuyuki Asada and Takeshi Tsukada for discussions on the topic, and anonymous referees for useful comments. This work was supported by JSPS KAKENHI Grant Number JP15H05706, JP20H00577, and JP20H05703, and by ANR PPS Grant Number 19-CE48-0014, and by ERC CoG DIAPASoN Grant Agreement 818616.

References

[ADLY18] Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On probabilistic term rewriting. In John P. Gallagher and Martin Sulzmann, editors, *FLOPS 2018*, volume 10818 of *LNCS*, pages 132–148. Springer, 2018.

[BBFK14] Tomás Brázdil, Václav Brozek, Vojtech Forejt, and Antonín Kucera. Branching-time model-checking of probabilistic pushdown automata. *J. Comput. Syst. Sci.*, 80(1):139–156, 2014.

[BEKK13] Tomás Brázdil, Javier Esparza, Stefan Kiefer, and Antonín Kucera. Analyzing probabilistic pushdown automata. *Formal Methods in System Design*, 43(2):124–163, 2013.

[BFG+04] Gilles Barthe, Maria João Frade, Eduardo Giménez, Luis Pinto, and Tarmo Uustalu. Type-based termination of recursive definitions. *MSCS*, 14(1):97–141, 2004.

[BFK+18] Giorgio Bacci, Robert Furber, Dexter Kozen, Radu Mardare, Prakash Panangaden, and Dana Scott. Boolean-valued semantics for the stochastic λ-calculus. In *LICS 2018*, pages 669–678, 2018.

[BG05] Olivier Bournez and Florent Garnier. Proving positive almost-sure termination. In Jürgen Giesl, editor, *RTA 2005*, volume 3467 of *LNCS*, pages 323–337. Springer, 2005.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[BK13] Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *CSL 2013*, volume 23 of *LIPIcs*, pages 129–148, 2013.

[BL18] Flavien Breuvart and Ugo Dal Lago. On intersection types and probabilistic lambda calculi. In David Sabel and Peter Thiemann, editors, *PPDP 2018*, pages 8:1–8:13. ACM, 2018.

[Cal02] Cristian Calude. *Information and Randomness: An Algorithmic Perspective*. Springer-Verlag, Berlin, Heidelberg, 2nd edition, 2002.

[CDL14] Raphaëlle Crubillé and Ugo Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi. In Zhong Shao, editor, *ESOP 2014*, volume 8410 of *LNCS*, pages 209–228. Springer, 2014.

[CHVB18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[CNZ17] Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. Stochastic invariants for probabilistic termination. In Giuseppe Castagna and Andrew D. Gordon, editors, *POPL 2017*, pages 145–160. ACM, 2017.

[Cou97] Patrick Cousot. Types as abstract interpretations. In Peter Lee, Fritz Henglein, and Neil D. Jones, editors, *POPL 1997*, pages 316–331. ACM Press, 1997.

[CS13] Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, pages 511–526, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[DH02] Vincent Danos and Russell Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.

[DLG17] Ugo Dal Lago and Charles Grellois. Probabilistic termination by monadic affine sized typing. In *ESOP 2017*, pages 393–419, 2017.

[dLMSS56] Karel de Leeuw, Edward F. Moore, Claude E. Shannon, and Norman Shapiro. Computability by probabilistic machines. *Automata Studies*, 34:183–212, 1956.

[DSA14] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *POPL 2014*, pages 297–308, 2014.

[DZ12] Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, 46(3):413–450, 2012.

[EGK10] Javier Esparza, Andreas Gaiser, and Stefan Kiefer. Computing least fixed points of probabilistic systems of polynomials. In *STACS 2010*, volume 5 of *LIPIcs*, pages 359–370. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

[EGK12] Javier Esparza, Andreas Gaiser, and Stefan Kiefer. Proving termination of probabilistic programs using patterns. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV 2012*, pages 123–138, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[ETP14] Thomas Ehrhard, Christine Tasson, and Michele Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *POPL 2014*, pages 309–320, 2014.

[EY09] Kousha Etessami and Mihalis Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1):1:1–1:66, 2009.

[EY12] Kousha Etessami and Mihalis Yannakakis. Model checking of recursive probabilistic systems. *ACM Trans. Comput. Log.*, 13(2):12:1–12:40, 2012.

[EY15] Kousha Etessami and Mihalis Yannakakis. Recursive Markov decision processes and recursive stochastic games. *J. ACM*, 62(2):11:1–11:69, 2015.

[FH15] Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In Sriram K. Rajamani and David Walker, editors, *POPL 2015*, pages 489–501. ACM, 2015.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GM15a] Charles Grellois and Paul-André Melliès. Finitary semantics of linear logic and higher-order model-checking. In Italiano et al. [IPS15], pages 256–268.

[GM15b] Charles Grellois and Paul-André Melliès. Relational semantics of linear logic and higher-order model checking. In *CSL 2015*, volume 41 of *LIPIcs*, pages 260–276. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[GMR+08] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *In UAI*, pages 220–229, 2008.

[GO10] Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP 2010*, volume 6199 of *LNCS*, pages 527–538. Springer, 2010.

[HKSY17] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

[HMOS08] Matthew Hague, Andrzej Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS 2008*, pages 452–461. IEEE Computer Society, 2008.

[HPS96] John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In Hans-Juergen Boehm and Guy L. Steele Jr., editors, *POPL 1996*, pages 410–423. ACM Press, 1996.

[Hro05] Juraj Hromkovic. *Design and Analysis of Randomized Algorithms - Introduction to Design Paradigms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2005.

[IPS15] Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors. *MFCS 2015*, volume 9234 of *LNCS*. Springer, 2015.

[JP89] C. Jones and Gordon D. Plotkin. A probabilistic powerdomain of evaluations. In *LICS 1989*, pages 186–195. IEEE Computer Society, 1989.

[JT98] Achim Jung and Regina Tix. The troublesome probabilistic powerdomain. *ENTCS*, 13:70 – 91, 1998. Comprox III, Third Workshop on Computation and Approximation.

[KDLG19] Naoki Kobayashi, Ugo Dal Lago, and Charles Grellois. On the termination problem for probabilistic higher-order recursive programs. In *Proceedings of LICS 2019*. IEEE, 2019.

[KK15] Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the hardness of almost-sure termination. In Italiano et al. [IPS15], pages 307–318.

[KLE07] Stefan Kiefer, Michael Luttenberger, and Javier Esparza. On the convergence of newton's method for monotone systems of polynomial equations. In *STOC 2007*, pages 217–226. ACM, 2007.

[KNU01] Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA 2001*, volume 2044 of *LNCS*, pages 253–267. Springer, 2001.

[KNU02] Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.

[KO09] Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.

[KO11] Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *LMCS*, 7(4), 2011.

[Kob09a] Naoki Kobayashi. Model-checking higher-order functions. In *PPDP 2009*, pages 25–36. ACM Press, 2009.

[Kob09b] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL 2009*, pages 416–428. ACM Press, 2009.

[Kob13] Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3), 2013.

[Kob19] Naoki Kobayashi. Inclusion between the frontier language of a non-deterministic recursive program scheme and the dyck language is undecidable. *Theor. Comput. Sci.*, 777:409–416, 2019.

[KS15] Gregory M. Kobele and Sylvain Salvati. The IO and OI hierarchies revisited. *Inf. Comput.*, 243:205–221, 2015.

[KSU11] Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and CEGAR for higher-order model checking. In *PLDI 2011*, pages 222–233. ACM Press, 2011.

[Mat93] Yuri V. Matiyasevich. *Hilbert's Tenth Problem*. The MIT Press, 1993.

[MMKK18] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A new proof rule for almost-sure termination. *POPL 2018*, pages 33:1–33:28, 2018.

[MO05] Andrzej S. Murawski and Joël Ouaknine. On probabilistic program equivalence and refinement. In *CONCUR 2005*, pages 156–170, 2005.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. CUP, New York, NY, USA, 1995.

[MRH18] Agustín Mista, Alejandro Russo, and John Hughes. Branching processes for quickcheck generators. *CoRR*, abs/1808.01520, 2018.

[Ong06] C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society Press, 2006.

[Par18] Pawel Parys. Recursion schemes and the WMSO+U logic. In *STACS 2018*, volume 96 of *LIPIcs*, pages 53:1–53:16, 2018.

[PF17] Simon M. Poulding and Robert Feldt. Automated random testing in multiple dispatch languages. In *ICST 2017*, pages 333–344. IEEE Computer Society, 2017.

[RNO14] Steven Ramsay, Robin Neatherway, and C.-H. Luke Ong. An abstraction refinement approach to higher-order model checking. In *POPL 2014*, pages 61–72. ACM, 2014.

[Sah78] N. Saheb-Djahromi. Probabilistic LCF. In Józef Winkowski, editor, *MFCS 1978*, volume 64 of *LNCS*, pages 442–451. Springer, 1978.

[San69] Eugene S. Santos. Probabilistic Turing machines and computability. *Proc. of the AMS*, 22(3):704–710, 1969.

[Sta04] Richard Statman. On the lambda Y calculus. *APAL*, 130(1-3):325–337, 2004.

[SV16] Davide Sangiorgi and Valeria Vignudelli. Environmental bisimulations for probabilistic higher-order languages. In *POPL 2016*, pages 595–607, 2016.

[SW11] Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. In *ICALP 2011*, volume 6756 of *LNCS*, pages 162–173. Springer, 2011.

[SYW+16] Sam Staton, Hongseok Yang, Frank D. Wood, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *LICS 2016*, pages 525–534, 2016.

[TO14] Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via ω-regular games over Böhm trees. In *CSL-LICS 2014*, pages 78:1–78:10. ACM, 2014.

[WMM14] Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In Samuel Kaski and Jukka Corander, editors, *Int. Conf. on Artificial Intelligence and Statistics*, volume 33 of *Proc. of Machine Learning Research*, pages 1024–1032, Reykjavik, Iceland, 22–25 Apr 2014. PMLR.

[YE05] Mihalis Yannakakis and Kousha Etessami. Checking LTL properties of recursive Markov chains. In *QEST 2005*, pages 155–165. IEEE Computer Society, 2005.

Appendix

Appendix A. Relationship between PHORS and Recursive Markov Chains

In this section, we provide mutual translations between order-1 PHORS and recursive Markov chains.

A.1. **Encoding Recursive Markov Chains into Order-1 PHORS.** In this section, we will give a sketch of a proof that any recursive Markov chain (RMC in the following) can be faithfully encoded as an order-1 PHORS. In doing that, we will closely follow the notational conventions and definitions from [EY09], Section 2.

Let us first of all fix an RMC $A = (A_1, \ldots, A_k)$, where each component graph is $A_i = (N_i, B_i, Y_i, En_i, Ex_i, \delta_i)$. We fix a reachability problem, given in the form of a triple (i_I, s_I, q_I) where $i_I \in \{1, \ldots, k\}$, s_I is a vertex of A_{i_I}, and $q_I \in Ex_{i_I}$, where a vertex of each A_i is defined as an element of

$$N_i \cup \bigcup_{b \in B_i} Call_b \cup \bigcup_{b \in B_i} Return_b.$$

Here, $Call_b = \{(b, en) \mid en \in En_{Y_i(b)}\}$ and $Return_b = \{(b, ex) \mid ex \in Ex_{Y_i(b)}\}$ The reachability problem (i_I, s_I, q_I) specifies $\langle \epsilon, s_I \rangle$ as the initial state, where s_I is a vertex of the component graph i_I, and $\langle \epsilon, q_I \rangle$ as the reachability target (cf. Section 2.2 of [EY09]). The PHORS $\mathcal{G}_A = (\mathcal{N}_A, \mathcal{R}_A, S_A)$ is defined as follows:

- Nonterminals are defined as symbols of the form $F_{i,s}$ where $i \in \{1, \ldots, k\}$, and s is a vertex A_i. The type $\mathcal{N}_A(F_{i,s})$ is $\mathsf{o}^{|Ex_i|} \to \mathsf{o}$. There is also a nonterminal S_A of type o, which is taken to be (i_I, s_I, q_I). The start symbol is S_A.
- Rules in $\mathcal{R}_A$ are of four kinds:
  - There is a rule
  $$(i_I, s_I, q_I) = S_A = F_{i_I, s_I}(\underbrace{\Omega, \ldots, \Omega}_{j-1 \text{ times}}, \mathsf{e}, \Omega, \ldots, \Omega)$$
  where $Ex_i = \{s_1, \ldots, s_{|Ex_i|}\}$ and $q_I = s_j$.
  - For every i and for every exit node $s_j \in Ex_i = \{s_1, \ldots, s_{|Ex_i|}\}$, there is a rule
  $$F_{i,s_j}(x_1, \ldots, x_{|Ex_i|}) = x_j$$
  - For each i and for each non-exit node or return port s of A_i, there is a rule
  $$F_{i,s}(x_1, \ldots, x) = \bigoplus_j p_{s,q} F_{i,q}(x_1, \ldots, x)$$
  where $p_{s,q}$ is the probability to go from s to q, as given by the transition function δ_i.
  - For every i and for every call port $s = (b, en)$ of A_i which is in $Call_b$, there is a rule
  $$F_{i,s}(\vec{x}) = F_{Y_i(b),en}(F_{i,(b,ex_1)}(\vec{x}), \ldots, F_{i,(b,ex_v)}(\vec{x}))$$
  and $Ex_{Y_i(b)} = \{ex_1, \ldots, ex_v\}$.

The next step is to put any global state in M_A in correspondence to a term of $\mathcal{G}_A$. This is actually quite easy, once one realizes that:

- $\mathcal{G}_A$ is designed so that every term to which S_A reduces can be seen as a complete ordered tree.
- The rules in $\mathcal{R}_A$ have been designed so as to closely mimick the four inductive clauses by which the transition relation Δ of the Markov chain M_A is defined. In particular, any such pair $\langle \beta, u\rangle$ is such that the length of β corresponds to the height of the corresponding term to which S_A reduces. The only caveat is that the first such inductive clause needs to be restricted, because in PHORS, contrarily to Markov chains, one needs to fix *one* initial state.
- $(\langle \beta, u\rangle, p, \langle \beta', u'\rangle) \in \Delta$ if and only if the term corresponding to $\langle \beta, u\rangle$ rewrites to the term corresponding to $\langle \beta', u'\rangle$ with probability p in one step.

As a consequence, one easily derive that $\mathcal{P}(\mathcal{G}_A)$ is precisely the probability, in M_A, to reach $\langle \epsilon, q_I\rangle$ starting from $\langle \epsilon, s_I\rangle$.

A.2. **Encoding Order-1 PHORS into Recursive Markov Chains.** In this section, we show that any order-1 PHORS can be encoded into a recursive Markov chain that has the same termination probability.

First, we can normalize any order-1 PHORS to the one consisting of the rewriting rules of the form:

$$\begin{aligned} &S = F_1\,\mathsf{e}\cdots\mathsf{e}\\ &F_1\,x_1\cdots x_k = t_{1,L}\ \oplus_{p_1}\ t_{1,R}\\ &\cdots\\ &F_m\,x_1\cdots x_k = t_{m,L}\ \oplus_{p_1}\ t_{m,R}, \end{aligned}$$

where each $t_{i,d}$ ($i \in \{1,\ldots,m\}, d \in \{L,R\}$) is a variable x_j ($j \in \{1,\ldots,k\}$), or is of the form:

$$F_i\,(F_{j_1}\,x_1\cdots x_k)\cdots(F_{j_k}\,x_1\cdots x_k).$$

Note that Ω on the righthand side can be replaced by $F\,x_1\cdots x_k$ where F is defined by

$$F\,x_1,\cdots x_k = F(F\,x_1,\cdots x_k)\cdots(F\,x_1,\cdots x_k).$$

Given the normalized order-1 HORS above, let M be a recursive Markov chain consisting of a single component $A_1 = (N_1, B_1, Y_1, En_1, Ex_1, \delta_1)$ where:

- B_1 is the set of terms of the form $F_i\,(F_{j_1}\,x_1\cdots x_k)\cdots(F_{j_k}\,x_1\cdots x_k)$ on the righthand side.
- $Y_1(b) = 1$ for every $b \in B_1$.
- $En_1 = \{F_1,\ldots,F_m\}$.
- $Ex_1 = \{x_1,\ldots,x_k\}$.
- $N_1 = En_1 \cup Ex_1$.
- δ_1 is the least set of the transitions that satisfies:
  - $(F_i, p, x_j) \in \delta_1$ for each transition rule $F_i\,x_1,\cdots x_k \xrightarrow{d,p} x_j$
    (recall that we write $F\,x_1,\cdots x_k \xrightarrow{L,p} t_L$ and $F\,x_1,\cdots x_k \xrightarrow{R,1-p} t_R$ if there is a rule $F\,x_1,\cdots x_k = t_L\ \oplus_p\ t_R$).
  - $(F_i, p, (t, F_j)) \in \delta_1$
    if $F_i\,x_1,\cdots x_k \xrightarrow{d,p} t$ and t is of the form $F_j\,(F_{j_1}\,x_1\cdots x_k)\cdots(F_{j_k}\,x_1\cdots x_k)$.

- $((t, x_i), p, x_\ell) \in \delta_1$
  if $t = F_j\,(F_{j_1}\,x_1\,\cdots\,x_k)\,\cdots\,(F_{j_k}\,x_1\,\cdots\,x_k)$, and $F_{j_i}\,x_1\,\cdots\,x_k \xrightarrow{d,p} x_\ell$.
- $((t, x_i), p, (t', F_{j'})) \in \delta_1$
  if $t = F_j\,(F_{j_1}\,x_1\,\cdots\,x_k)\,\cdots\,(F_{j_k}\,x_1\,\cdots\,x_k)$ and $F_{j_i}\,x_1\,\cdots\,x_k \xrightarrow{d,p} t'$, where $t' = F_{j'}\,(F_{j'_1}\,x_1\,\cdots\,x_k)\,\cdots\,(F_{j'_k}\,x_1\,\cdots\,x_k)$,

Intuitively, a PHORS term of the form $F_j\,(F_{j_1}\,x_1\,\cdots\,x_k)\,\cdots\,(F_{j_k}\,x_1\,\cdots\,x_k)$ is modeled as a call of F_j, where F_{j_i} is executed when the call exits from the exit port x_i. That is why, in the third and fourth kinds of transition rules above, the next node is determined by the rule for F_{j_i}. From this intuition, it should be trivial that the termination probabilities of the RMC and the original PHORS coincide.

Appendix B. Proofs for Section 4

B.1. **Proofs for Section 4.2.**

B.1.1. *Proof of Lemma 4.3.* We first prove the following lemma:

Lemma B.1. *If $\mathcal{K} \vdash t : \kappa$, then $\mathcal{K}^\# \vdash t^\# : \kappa^\#$.*

Proof. This follows by straightforward induction on the derivation of $\mathcal{K} \vdash t : \kappa$. □

Proof of Lemma 4.3. $\mathcal{N}^\# \vdash S : \mathtt{R}$ follows immediately from $\mathcal{N}(S) = \mathtt{o}$, Lemma B.1, and $\mathtt{o}^\# = \mathtt{R}$. Let $\mathcal{R}$ be $\{F_i\,x_1\,\cdots, x_{\ell_i} = t_i \mid i \in \{1, \ldots, m\}\}$. Then, by the definition of PHORS, we have $\mathcal{N}, x_1 : \kappa_{i,1}, \ldots, x_{\ell_i} : \kappa_{i,\ell_i} \vdash t_i : \mathtt{o}$, with $\mathcal{N}(F) = \kappa_{i,1} \to \cdots \to \kappa_{i,\ell_i} \to \mathtt{o}$. We need to show that

$$\mathcal{N}^\#, x_1 : \kappa^\#_{i,1}, \ldots, x_i : \kappa^\#_{i,\ell_i} \vdash t_i^\# : \mathtt{R}$$

for each i, but this follows immediately from the typing of t_i above and Lemma B.1. □

B.1.2. *Proof of Theorem 4.4.* We call a PHORS $\mathcal{G}$ *recursion-free* if there is no cyclic dependency on its non-terminals. More precisely, given a PHORS $\mathcal{G}$, we define the relation $\succ_\mathcal{G}$ on its non-terminals by: $F_i \succ_\mathcal{G} F_j$ iff F_j occurs on the righthand side of the rule for F_i. A PHORS $\mathcal{G}$ is defined to be recursion-free if the transitive closure of $\succ_\mathcal{G}$ is irreflexive.

Below we write $t^\#_\rho$ for $[\![t^\#]\!]_\rho$.

Lemma B.2. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ be a recursion-free PHORS, and ρ be the least solution of $\mathcal{E}_\mathcal{G}$. If $\mathcal{N} \vdash t : \mathtt{o}$, then $\mathcal{P}(\mathcal{G}, t) = t^\#_\rho$.*

Proof. Since $\mathcal{G}$ is recursion-free, it follows from the strong normalization of the simply-typed λ-calculus that t does not have any infinite reduction sequence. Because the reduction relation is finitely branching, by König's lemma, there are only finitely many reduction sequences from t; thus a longest reduction sequence from t exists, and we write $\sharp(t)$ for its length. The proof proceeds by induction on $\sharp(t)$. If $\sharp(t) = 0$, then t is either $\mathtt{e}$ (in which case, both sides of the equation are 1) or Ω (in which case, both sides of the equation are

0); thus, the result follows immediately. Otherwise, t must be of the form $F\ s_1 \cdots s_k$ where $F\ x_1 \cdots x_k = t_1 \oplus_p t_2$. Then

$$\begin{aligned}\mathcal{P}(\mathcal{G}, t) = p\mathcal{P}(\mathcal{G}, [s_1/x_1, \ldots, s_k/x_k]t_1)\\ +(1-p)\mathcal{P}(\mathcal{G}, [s_1/x_1, \ldots, s_k/x_k]t_2).\end{aligned}$$

Since $\sharp(t) > \sharp([s_1/x_1, \ldots, s_k/x_k]t_i)$, by the induction hypothesis, the righthand side equals:

$$\begin{aligned}&p([s_1/x_1, \ldots, s_k/x_k]t_1)^{\#}_{\rho} + (1-p)([s_1/x_1, \ldots, s_k/x_k]t_2)^{\#}_{\rho}\\ &= p(t_1)^{\#}_{\rho\{\widetilde{x} \mapsto \widetilde{s}^{\#}_{\rho}\}} + (1-p)(t_2)^{\#}_{\rho\{\widetilde{x} \mapsto \widetilde{s}^{\#}_{\rho}\}}\\ &= (F\ s_1 \cdots s_k)^{\#}_{\rho} = t^{\#}_{\rho},\end{aligned}$$

as required. □

For a PHORS $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ with $dom(\mathcal{N}) = \{F_1, \ldots, F_m\}$, we define its *k-th approximation* $\mathcal{G}^{(k)} = (\mathcal{N}^{(k)}, \mathcal{R}^{(k)}, S^{(k)})$ by:

$$\begin{aligned}&\mathcal{N}^{(k)} = \{F_j^{(i)} \mapsto \mathcal{N}(F_j) \mid j \in \{1, \ldots, m\}, 0 \le i \le k\}\\ &\mathcal{R}^{(k)}(F_j^{(i)}) = [F_1^{(i-1)}/F_1, \ldots, F_m^{(i-1)}/F_m]\mathcal{R}(F_j)\\ &\qquad\qquad\qquad\qquad \text{for each } i \in \{1, \ldots, k\}\\ &\mathcal{R}^{(k)}(F_j^{(0)}) = \lambda\widetilde{x}.\Omega \oplus_1 \Omega.\end{aligned}$$

The following properties follow immediately from the construction of $\mathcal{G}^{(k)}$. (Recall that $\mathcal{F}_{\mathcal{E}}$ denotes the function associated with the fixpoint equations $\mathcal{E}$, as defined in Section 4.1.)

Lemma B.3. (1) *$\mathcal{G}^{(k)}$ is recursion-free.*
(2) *$\mathcal{P}(\mathcal{G}) = \bigsqcup_{k\in\omega} \mathcal{P}(\mathcal{G}^{(k)})$.*
(3) *$\mathcal{F}^k_{\mathcal{E}_{\mathcal{G}}}(\bot_{[\![\mathcal{N}]\!]})(F) = \mathcal{F}^k_{\mathcal{E}_{\mathcal{G}^{(k)}}}(\bot_{[\![\mathcal{N}^{(k)}]\!]})(F^{(k)}) = \mathbf{lfp}(\mathcal{F}_{\mathcal{E}_{\mathcal{G}^{(k)}}})(F^{(k)})$ for each non-terminal F of $\mathcal{G}$.*

Proof. (1) This follows immediately from the fact that $F_\ell^{(i)} \succ_{\mathcal{G}^{(k)}} F_{\ell'}^{(j)}$ only if $j = i-1$.

(2) Let P be the set $\{(\pi, p) \mid S \xrightarrow{\pi,p}_{\mathcal{G}} \mathtt{e}\}$ and $P^{(k)}$ be $\{(\pi, p) \mid S^{(k)} \xrightarrow{\pi,p}_{\mathcal{G}^{(k)}} \mathtt{e}\}$. Then $\mathcal{P}(\mathcal{G}) = \sum_{(\pi,p)\in P} p$ and $\mathcal{P}(\mathcal{G}^{(k)}) = \sum_{(\pi,p)\in P^{(k)}} p$. Note that for any reduction $s \xrightarrow{d,p}_{\mathcal{G}^{(k)}} t$ with $t \neq \Omega$, there exists a corresponding reduction $s^! \xrightarrow{d,p}_{\mathcal{G}} t^!$ where $s^!$ and $t^!$ are the terms of $\mathcal{G}$ obtained from s and t respectively, by removing indices, i.e. by replacing each $F^{(i)}$ with F. Thus, $P^{(k)} \subseteq P$ for any k. Conversely, if $S \xrightarrow{\pi,p}_{\mathcal{G}} \mathtt{e}$, then $S^{(|\pi|)} \xrightarrow{\pi,p}_{\mathcal{G}^{(|\pi|)}} \mathtt{e}$, because non-terminals are unfolded at most $|\pi|$ times in $S \xrightarrow{\pi,p}_{\mathcal{G}} \mathtt{e}$. Therefore, $P = \bigcup_k P^{(k)}$, from which the result follows.

(3) We show that $\mathcal{F}^k_{\mathcal{E}_{\mathcal{G}}}(\bot_{[\![\mathcal{N}]\!]})(F) = \mathcal{F}^k_{\mathcal{E}_{\mathcal{G}^{(\ell)}}}(\bot_{[\![\mathcal{N}^{(\ell)}]\!]})(F^{(k)})$ holds for any $\ell \ge k$, by induction on k, from which the first equality follows. The base case $k = 0$ is trivial. For $k > 0$, By the definition of the rule for $F^{(k)}$ and the induction hypothesis, we have:

$$\begin{aligned}&\mathcal{F}^k_{\mathcal{E}_{\mathcal{G}^{(\ell)}}}(\bot_{[\![\mathcal{N}^{(\ell)}]\!]})(F^{(k)})\\ &= [\![([F_1^{(k-1)}/F_1, \ldots, F_m^{(k-1)}/F_m]\mathcal{R}(F))^{\#}]\!]_{\mathcal{F}^{k-1}_{\mathcal{E}_{\mathcal{G}^{(\ell)}}}(\bot_{[\![\mathcal{N}^{(\ell)}]\!]})}\\ &= [\![\mathcal{R}(F)^{\#}]\!]_{\{F_i \mapsto \mathcal{F}^{k-1}_{\mathcal{E}_{\mathcal{G}^{(\ell)}}}(\bot_{[\![\mathcal{N}^{(\ell)}]\!]})(F_i^{(k-1)}) \mid i \in \{1,\ldots,m\}\}}\\ &= [\![\mathcal{R}(F)^{\#}]\!]_{\mathcal{F}^{k-1}_{\mathcal{E}_{\mathcal{G}}}(\bot_{[\![\mathcal{N}]\!]})} \text{ (by the induction hypothesis)}\\ &= \mathcal{F}^k_{\mathcal{E}_{\mathcal{G}}}(\bot_{[\![\mathcal{N}]\!]})(F),\end{aligned}$$

as required. (Here, we have extended $(\cdot)^{\#}$ and $[\![t]\!]_\rho$ for λ-terms in the obvious manner.)

For the second equality, we can show that $\mathcal{F}^k_{\mathcal{E}_{\mathcal{G}^{(\ell)}}}(\bot_{[\![\mathcal{N}^{(\ell)}]\!]})(F^{(k)}) = \mathbf{lfp}(\mathcal{F}_{\mathcal{E}_{\mathcal{G}^{(\ell)}}})(F^{(k)})$ holds for any $\ell \geq k$, by straightforward induction on k. □

Theorem 4.4 follows as a corollary of the above lemmas.

Proof of Theorem 4.4. By Lemmas B.2 and B.3, we have

$$\begin{aligned}\mathcal{P}(\mathcal{G}) &= \bigsqcup_k \mathcal{P}(\mathcal{G}^{(k)}) = \bigsqcup_k \mathbf{lfp}(\mathcal{F}_{\mathcal{E}_{\mathcal{G}^{(k)}}})(S^{(k)}) \\ &= \bigsqcup_k \mathcal{F}^k_{\mathcal{E}_{\mathcal{G}}}(\bot)(S) = \mathbf{lfp}(\mathcal{F}_{\mathcal{E}_{\mathcal{G}}})(S)\end{aligned}$$

as required. □

B.2. **Proofs for Section 4.3.**

B.2.1. *Proof of Lemma 4.11.* We define the translation for a type environment on variables (other than non-terminals; note that the translation is different from the one for $\mathcal{N}$) by:

$$(y_1 : \kappa_1, \ldots, y_k : \kappa_k)^\dagger = (y_{1,0}, \ldots, y_{1,\mathtt{ar}(\kappa_1)+1}) : \kappa_1^\dagger, \ldots, (y_{k,0}, \ldots, y_{k,\mathtt{ar}(\kappa_k)+1}) : \kappa_k^\dagger.$$

Lemma B.4. *If $\mathcal{N} \cup \mathcal{K}, \widetilde{x} : \widetilde{\mathtt{o}} \vdash t : \kappa$ and $\mathcal{K}; \widetilde{x} \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow e$, then $\mathcal{N}^\dagger \cup \mathcal{K}^\dagger \vdash e : \kappa^{\dagger + |\widetilde{x}|}$.*

Proof. This follows by straightforward induction on the derivation of $\mathcal{N} \cup \mathcal{K}, \widetilde{x} : \widetilde{\mathtt{o}} \vdash t : \kappa$. □

We also prepare the following lemma on the syntactic property of the translation result, which is important for Lemma 4.11 and the substitution lemma (Lemma B.10 below) proved later.

Lemma B.5. *Suppose:*

$$\mathcal{K}; \widetilde{z} \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (t_0, \ldots, t_{\mathtt{ar}(\kappa)}, t_{\mathtt{ar}(\kappa)+1}, \ldots, t_{\mathtt{ar}(\kappa)+|\widetilde{z}|+1}).$$

Then, for each $y_i \in dom(\mathcal{K})$, $y_{i,0}$ does not occur in $t_1, \ldots, t_{\mathtt{ar}(\kappa)+|\widetilde{z}|+1}$.

Proof. This follows by straightforward induction on the structure of t. □

Proof of Lemma 4.11. $\mathcal{N}^\dagger(S_1) = \mathtt{R}$ follows immediately from $\mathcal{N}(S) = \mathtt{o} \to \mathtt{o}$ and the definition of $\mathcal{N}^\dagger$. To prove $\mathcal{N}^\dagger \vdash \mathcal{E}$, let $F\, y_1 \cdots y_\ell\, x_1 \cdots x_k = t_L \oplus_p t_R \in \mathcal{R}$, with $\mathcal{N}(F) = \kappa_1 \to \cdots \to \kappa_\ell \Rightarrow \mathtt{o}^k \to \mathtt{o}$. Suppose also

$$y_1 : \kappa_1, \ldots, y_\ell : \kappa_\ell; x_1, \ldots, x_k \vdash_{\mathcal{N}} t_d : \mathtt{o} \rightsquigarrow (t_{d,0}, \ldots, t_{d,k+1})$$

for $d \in \{L, R\}$. We need to prove

$$\mathcal{N}^\dagger, (\widetilde{y_1}) : {\kappa_1}^\dagger, \ldots, (\widetilde{y_\ell}) : {\kappa_\ell}^\dagger \vdash t_{d,0} : \mathtt{R}$$

and

$$\mathcal{N}^\dagger, (\widetilde{y_1}') : {\kappa_1}^{\dagger'}, \ldots, (\widetilde{y_\ell}') : {\kappa_\ell}^{\dagger'} \vdash t_{d,i} : \mathtt{R}$$

for $i \in \{1, \ldots, k\}$, where $\widetilde{y_i}$ and $\widetilde{y_i}'$ are as given in the premises of the rule Tr-Rule. The former follows immediately from Lemma B.4. For the latter, by Lemma B.4, we have

$$\mathcal{N}^\dagger, (\widetilde{y_1}) : {\kappa_1}^\dagger, \ldots, (\widetilde{y_\ell}) : {\kappa_\ell}^\dagger \vdash t_{d,i} : \mathtt{R}.$$

By Lemma B.5, $y_{i,0}$ does not occur in $t_{d,i}$. Thus we can remove the type bindings on them and obtain

$$\mathcal{N}^\dagger, (\widetilde{y_1}') : {\kappa_1}^{\dagger'}, \ldots, (\widetilde{y_\ell}') : {\kappa_\ell}^{\dagger'} \vdash t_{d,i} : \mathtt{R}$$

as required. □

B.2.2. *Proof of Theorem 4.12.* Given two expressions e_1, e_2 and fixpoint equations $\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}$, we write $e_1 \cong_{\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}} e_2$ if $[\![e_1]\!]_{\rho_{\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}}} = [\![e_e]\!]_{\rho_{\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}}}$. We often omit the subscript.

As sketched in Section 4.3, we first prove the theorem for recursion-free PHORS. A key property used for showing it is that the translation relation is preserved by reductions, roughly in the sense that if $t \xrightarrow{L,p} t_L$ and $t \xrightarrow{R,1-p} t_R$, then $t \rightsquigarrow e$ (i.e., t is translated to e) implies that there exist e_L and e_R such that $t_L \rightsquigarrow e_L$, $t_R \rightsquigarrow e_R$ and $e \cong p \cdot e_L + (1-p) \cdot e_R$ (where $+$ and $\cdot$ are pointwise extended to operations on tuples). Thus, the property that e represents the termination probability of t follows from the corresponding properties of e_L and e_R; by induction (note that since we are considering recursion-free PHORS, $\flat(t) > \flat(t_L), \flat(t_R)$, where $\flat(t)$ denotes the length of the longest reduction sequence from t), it follows that if the initial term is translated to e_0, then e_0 represents the termination probability of the initial term.

Unfortunately, however, the translation relation is *not* necessarily preserved by the standard reduction relation $\xrightarrow{d,p}_{\mathcal{G}}$ defined in Section 2. We thus introduce another reduction relation that uses *explicit substitutions* on order-0 variables. To this end, we extend the syntax of terms as follows.

$$t \text{ (extended terms)} ::= \Omega \mid x \mid t_1 t_2 \mid \{t_1/x_1, \ldots, t_k/x_k\}t_0$$

Here, $\{t_1/x_1, \ldots, t_k/x_k\}t_0$ represents an *explicit* substitution; the intended meaning is the same as the ordinary substitution $[t_1/x_1, \ldots, t_k/x_k]t_0$ (which represents the term obtained from t_0 by simultaneously substituting t_i for x_i), but the substitution is delayed until one of the variables in $x_1, \ldots, x_k$ becomes necessary. We often abbreviate $\{t_1/x_1, \ldots, t_k/x_k\}$ as $\{\widetilde{t}/\widetilde{x}\}$. Note that we have omitted e; we consider an open term $S\,x$ as the initial term instead of $S\,\mathsf{e}$. The type judgment relation for terms is extended by adding the following rule:

$$\frac{\mathcal{K} \vdash s_i : \mathsf{o} \text{ (for each } i \in \{1, \ldots, k\}) \qquad \mathcal{K}, x_1 : \mathsf{o}, \ldots, x_k : \mathsf{o} \vdash t : \mathsf{o}}{\mathcal{K} \vdash \{s_1/x_1, \ldots, s_k/x_k\}t : \mathsf{o}}$$

Thus, explicit substitutions are allowed only for order-0 variables.

Reductions with explicit substitutions:

We now define a reduction relation for extended terms. The set of evaluation contexts, ranged over by E, is defined by:

$$E ::= [\,] \mid \{\widetilde{t}/\widetilde{x}\}E.$$

The new reduction relation $t \xrightarrow{d,p}_{\mathsf{es},\mathcal{G}} t'$ (where $d \in \{L, R, \epsilon\}$) is defined as follows.

$$\frac{z \notin \{x_1, \ldots, x_k\}}{E[\{t_1/x_1, \ldots, t_k/x_k\}z] \xrightarrow{\epsilon,1}_{\mathsf{es},\mathcal{G}} E[z]}$$

$$\frac{}{E[\{t_1/x_1, \ldots, t_k/x_k\}x_i] \xrightarrow{\epsilon,1}_{\mathsf{es},\mathcal{G}} E[t_i]}$$

$$\frac{\begin{array}{c}\mathcal{R}(F) = \lambda\widetilde{y}.\lambda\widetilde{z}.u_L \oplus_p u_R \\ \mathcal{N}(F) = \widetilde{\kappa} \Rightarrow \mathtt{o}^{\ell} \to \mathtt{o} \qquad \ell = |\widetilde{z}| = |\widetilde{t}| \qquad |\widetilde{y}| = |\widetilde{s}| \\ \widetilde{z} \text{ do not occur in } E[F\,\widetilde{s}\,\widetilde{t}]\end{array}}{E[F\,\widetilde{s}\,\widetilde{t}] \xrightarrow{L,p}_{\mathtt{es},\mathcal{G}} E[\{\widetilde{t}/\widetilde{z}\}[\widetilde{s}/\widetilde{y}]u_L]}$$

$$\frac{\begin{array}{c}\mathcal{R}(F) = \lambda\widetilde{y}.\lambda\widetilde{z}.u_L \oplus_p u_R \\ \mathcal{N}(F) = \widetilde{\kappa} \Rightarrow \mathtt{o}^{\ell} \to \mathtt{o} \qquad \ell = |\widetilde{z}| = |\widetilde{t}| \qquad |\widetilde{y}| = |\widetilde{s}| \\ \widetilde{z} \text{ do not occur in } E[F\,\widetilde{s}\,\widetilde{t}]\end{array}}{E[F\,\widetilde{s}\,\widetilde{t}] \xrightarrow{R,1-p}_{\mathtt{es},\mathcal{G}} E[\{\widetilde{t}/\widetilde{z}\}[\widetilde{s}/\widetilde{y}]u_R]}$$

We call reductions using the first two rules (i.e., reductions labeled by $\xrightarrow{\epsilon,p}_{\mathtt{es},\mathcal{G}}$) *administrative reductions*. In the last two rules, we assume that α-conversion is implicitly applied so that $\widetilde{z}$ do not clash with variables that are already used in $E[F\,\widetilde{s}\,\widetilde{t}]$. In those rules, recall also our notational convention that when we write $F\,\widetilde{s}\,\widetilde{t}$, the second sequence $\widetilde{t}$ is the maximal sequence of order-0 terms (that condition is made explicit in the above rules, but below we often omit to state it). As before, we often omit the subscript $\mathcal{G}$.

For an extended term t, we write t^* for the term obtained by replacing explicit substitutions with ordinary substitutions. For example, $(\{t/x\}(F\,x))^* = F\,t$. The following lemma states that the new reduction relation is essentially equivalent to the original reduction relation:

Lemma B.6. *Let s be an extended term.*

(1) *If $s \xrightarrow{\epsilon,1}_{es} t$, then $s^* = t^*$.*

(2) *If $s \xrightarrow{d,p}_{es} t$ with $d \in \{L, R\}$, then $s^* \xrightarrow{d,p} t^*$.*

(3) *If $s^* \xrightarrow{d,p} u$, then there exists t such that $s(\xrightarrow{\epsilon,1}_{es})^* \xrightarrow{d,p}_{es} t$ and $t^* = u$.*

Proof. Immediate from the definitions of $\xrightarrow{d,p}$ and $\xrightarrow{d,p}_{\mathtt{es}}$. □

We define $\overset{\pi,p}{\Longrightarrow}_{\mathtt{es},\mathcal{G}}$ in an analogous manner to $\overset{\pi,p}{\Longrightarrow}_{\mathcal{G}}$, where the label ϵ is treated as an empty word. For an extended term t that may contain an order-0 free variable x, we write $P_{\mathtt{es}}(\mathcal{G}, t, x)$ for the set $\{(\pi, p) \mid t \overset{\pi,p}{\Longrightarrow}_{\mathtt{es},\mathcal{G}} x\}$, and write $\mathcal{P}_{\mathtt{es}}(\mathcal{G}, t, x)$ for $\sum_{(\pi,p)\in P_{\mathtt{es}}(\mathcal{G},t,x)} p$, based on the new reduction relation. The following lemma follows immediately from the above definitions and Lemma B.6.

Lemma B.7. *Let t be a term of PHORS $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ such that $\mathcal{N}, x : \mathtt{o} \vdash t : \mathtt{o}$ and t does not contain $\mathtt{e}$. Then $\mathcal{P}(\mathcal{G}, [\mathtt{e}/x]t) = \mathcal{P}_{\mathtt{es}}(\mathcal{G}, t, x)$.*

Proof. By Lemma B.6, $t \overset{\pi,p}{\Longrightarrow}_{\mathtt{es}} x$ if and only if $t^* \overset{\pi,p}{\Longrightarrow} x$, if and only if $[\mathtt{e}/x]t^* \overset{\pi,p}{\Longrightarrow} \mathtt{e}$ (for the second "if and only if", recall the assumption that $\mathtt{e}$ does not occur in $\mathcal{R}$), from which the result follows. □

We extend the translation relation for terms with the following rule.

$$\frac{\begin{array}{c}\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} s_i : \mathtt{o} \rightsquigarrow (s_{i,0}, \ldots, s_{i,k+1}) \text{ (for each } i \in \{1, \ldots, \ell\}) \\ \mathcal{K}; z_1, \ldots, z_\ell, x_1, \ldots, x_k \vdash_{\mathcal{N}} t : \mathtt{o} \rightsquigarrow (t_0, \ldots, t_{k+\ell+1})\end{array}}{\begin{array}{c}\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} \{s_1/z_1, \ldots, s_\ell/z_\ell\}t : \mathtt{o} \rightsquigarrow \\ (t_0 + \Sigma_{i=1}^{\ell} t_i \cdot s_{i,0}, t_{\ell+1} + \Sigma_{i=1}^{\ell} t_i \cdot s_{i,1}, \ldots, t_{k+\ell+1} + \Sigma_{i=1}^{\ell} t_i \cdot s_{i,k+1})\end{array}} \quad (\textsc{Tr-Sub})$$

We shall prove that the translation relation is preserved by the new reduction relation (Lemmas B.11, B.12, and B.13 below).

Lemma B.8 (Weakening)**.** (1) *If* $\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow e$*, then* $\mathcal{K}, y : \kappa_y; x_1, \ldots, x_k \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow e$*.*
(2) *If* $\mathcal{K}; x_1, \ldots, x_k \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (t_0, \ldots, t_\ell)$*, then* $\mathcal{K}; x_1, \ldots, x_k, x_{k+1} \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (t_0, \ldots, t_\ell, t_\ell)$*.*

Proof. This follows by straightforward induction on the structure of t. □

Lemma B.9 (Exchange)**.**
If $\mathcal{K}; x_1, \ldots, x_i, x_{i+1}, \ldots, x_k \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (t_0, \ldots, t_{\mathtt{ar}(\kappa)+i}, t_{\mathtt{ar}(\kappa)+i+1}, \ldots, t_{\mathtt{ar}(\kappa)+k+1})$*, then* $\mathcal{K}; x_1, \ldots, x_{i+1}, x_i, \ldots, x_k \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (t_0, \ldots, t_{\mathtt{ar}(\kappa)+i+1}, t_{\mathtt{ar}(\kappa)+i}, \ldots, t_{\mathtt{ar}(\kappa)+k+1})$*.*

Proof. This follows by straightforward induction on the structure of t. □

As usual, the substitution lemma, stated below, is a critical lemma for proving subject reduction. The statement of our substitution lemma is, however, quite delicate, due to a special treatment of order-0 variables.

Lemma B.10 (Substitution)**.** *Suppose* t *does not contain any explicit substitutions (i.e., any subterms of the form* $\{\widetilde{u}/\widetilde{x}\}s$*). If* $\widetilde{y} : \widetilde{\kappa_y}; \widetilde{z} \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (\widetilde{t}, t_{m+1})$ *and* $\emptyset; x_1, \ldots, x_k \vdash_{\mathcal{N}} s_i : \kappa_{y,i} \rightsquigarrow (\widetilde{s_i}, s_{i,\ell_i+1}, \ldots, s_{i,\ell_i+k+1})$ *with* $\{x_1, \ldots, x_k\} \cap \{\widetilde{z}\} = \emptyset$ *and* $\ell_i = \mathtt{ar}(\kappa_{y,i})$*, then:*

$$\emptyset; \widetilde{z}, x_1, \ldots, x_k \vdash_{\mathcal{N}} [\widetilde{s}/\widetilde{y}]t : \kappa \rightsquigarrow (\theta_0\widetilde{t}, \theta_1 t_0, \ldots, \theta_k t_0, \theta_0 t_{m+1}),$$

where the substitutions $\theta_j (j \in \{0, \ldots, k\})$ *are defined by:*

$$\begin{array}{l}\theta_j = \theta_{1,j} \cdots \theta_{|\widetilde{s}|,j} \text{ for } j \in \{1, \ldots, k\} \\ \theta_{i,0} = [s_{i,0}/y_{i,0}, \ldots, s_{i,\ell_i}/y_{i,\ell_i}, s_{i,\ell_i+k+1}/y_{i,\ell_i+1}] \text{ for } i \in \{1, \ldots, |\widetilde{s}|\} \\ \theta_{i,j} = [s_{i,\ell_i+j}/y_{i,0}, s_{i,1}/y_{i,1}, \ldots, s_{i,\ell_i}/y_{i,\ell_i}, s_{i,\ell_i+k+1}/y_{i,\ell_i+1}] \\ \qquad\qquad \text{for } i \in \{1, \ldots, |\widetilde{s}|\}, j \in \{1, \ldots, k\}.\end{array}$$

Here, the part $\theta_1 t_0, \ldots, \theta_k t_0$ accounts for information about reachability to the newly introduced variables $x_1, \ldots, x_k$.

Proof. Induction on the derivation of $\widetilde{y} : \widetilde{\kappa_y}; \widetilde{z} \vdash_{\mathcal{N}} t : \kappa \rightsquigarrow (\widetilde{t}, t_{m+1})$.

- Case Tr-Omega: In this case, $t = \Omega$ with $\widetilde{t} = \widetilde{0}$ and $t_{m+1} = 0$. Thus, the result follows immediately from the rule Tr-Omega.
- Case Tr-GVar: In this case, $t = z_i$ and $\kappa = \mathtt{o}$, with $\widetilde{t} = 0^i, 1, 0^{|z|-i}$ and $t_{m+1} = 0$. By using Tr-GVar, we obtain

  $$\emptyset; \widetilde{z}, x_1, \ldots, x_k \vdash_{\mathcal{N}} [\widetilde{s}/\widetilde{y}]t(= z_i) : \kappa \rightsquigarrow (0^i, 1, 0^{|\widetilde{z}|+k-i+1}).$$

  Since

  $$(\theta_0\widetilde{t}, \theta_1 t_0, \ldots, \theta_k t_0, \theta_0 t_{m+1}) = (\widetilde{t}, \underbrace{t_0, \ldots, t_0}_{k}, t_{m+1}) = (0^i, 1, 0^{|\widetilde{z}|+k-i+1}),$$

  we have the required result.
- Case Tr-Var: In this case, $t = y_i$, with $\widetilde{t} = y_{i,0}, \ldots, y_{i,\ell_i}, y_{i,\ell_i+1}^{|\widetilde{z}|}$, and $t_{m+1} = y_{i,\ell_i+1}$. By applying Lemmas B.8 and B.9 to $\emptyset; x_1, \ldots, x_k \vdash_{\mathcal{N}} s_i : \kappa_{y,i} \rightsquigarrow (\widetilde{s_i}, s_{i,\ell_i+1}, \ldots, s_{i,\ell_i+k+1})$, we obtain:

  $$\emptyset; \widetilde{z}, x_1, \ldots, x_k \vdash_{\mathcal{N}} s_i : \kappa_{y,i} \rightsquigarrow (\widetilde{s_i}, s_{i,\ell_i+k+1}^{|\widetilde{z}|}, s_{i,\ell_i+1}, \ldots, s_{i,\ell_i+k+1}).$$

Since $\widetilde{t} = y_{i,0}, \ldots, y_{i,\ell_i}, y_{i,\ell_i+1}^{|\widetilde{z}|}$, and $t_{m+1} = y_{i,\ell_i+1}$, we have

$$(\theta_0\widetilde{t}, \theta_1 t_0, \ldots, \theta_k t_0, \theta_0 t_{m+1}) = (\widetilde{s_i}, s_{i,\ell_i+k+1}^{|\widetilde{z}|}, s_{i,\ell_i+1}, \ldots, s_{i,\ell_i+k+1}).$$

Thus we have the required result.

- Case Tr-App: In this case, we have $t = uv$ and

$$\begin{array}{l}\widetilde{y} : \widetilde{\kappa_y}; \widetilde{z} \vdash_{\mathcal{N}} u : \kappa_v \to \kappa \rightsquigarrow (u_0, \widetilde{u}, u_{\ell'+1}, \ldots, u_{\ell'+|\widetilde{z}|+1})\\ \widetilde{y} : \widetilde{\kappa_y}; \widetilde{z} \vdash_{\mathcal{N}} v : \kappa_v \rightsquigarrow (v_0, \widetilde{v}, v_{\ell''+1}, \ldots, v_{\ell''+|\widetilde{z}|+1})\\ \widetilde{t} = u_0(v_0, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1}), \widetilde{u}(\widetilde{v}, v_{\ell''+|\widetilde{z}|+1}),\\ \qquad\qquad u_{\ell'+1}(v_{\ell''+1}, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1}), \ldots, u_{\ell'+|\widetilde{z}|}(v_{\ell''+|\widetilde{z}|}, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1})\\ t_{m+1} = u_{\ell'+|\widetilde{z}|+1}(v_{\ell''+|\widetilde{z}|+1}, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1})\\ \ell' = |\widetilde{u}| = \mathtt{ar}(\kappa) \quad m = \ell' + |\widetilde{z}| \quad \ell'' = \mathtt{ar}(\kappa_v) = |\widetilde{v}|.\end{array}$$

By the induction hypothesis, we have:

$$\begin{array}{l}\emptyset; \widetilde{z}, x_1, \ldots, x_k \vdash_{\mathcal{N}} [\widetilde{s}/\widetilde{y}]u : \kappa_v \to \kappa \rightsquigarrow\\ \qquad (\theta_0(u_0, \widetilde{u}, u_{\ell'+1}, \ldots, u_{\ell'+|\widetilde{z}|}), \theta_1 u_0 \ldots, \theta_k u_0, \theta_0 u_{\ell'+|\widetilde{z}|+1})\\ \emptyset; \widetilde{z}, x_1, \ldots, x_k \vdash_{\mathcal{N}} [\widetilde{s}/\widetilde{y}]v : \kappa_v \rightsquigarrow\\ \qquad (\theta_0(v_0, \widetilde{v}, v_{\ell''+1}, \ldots, v_{\ell''+|\widetilde{z}|}), \theta_1 v_0, \ldots, \theta_k v_0, \theta_0 v_{\ell''+|\widetilde{z}|+1})\end{array}$$

By applying Tr-App, we obtain:

$$\begin{array}{l}\emptyset; \widetilde{z}, x_1, \ldots, x_k \vdash_{\mathcal{N}} ([\widetilde{s}/\widetilde{y}]u)([\widetilde{s}/\widetilde{y}]v) : \kappa \rightsquigarrow\\ \quad ((\theta_0 u_0)(\theta_0 v_0, \theta_0\widetilde{v}, \theta_0 v_{\ell''+|\widetilde{z}|+1}), (\theta_0\widetilde{u})(\theta_0\widetilde{v}, \theta_0 v_{\ell''+|\widetilde{z}|+1}),\\ \quad (\theta_0 u_{\ell'+1})(\theta_0 v_{\ell''+1}, \theta_0\widetilde{v}, \theta_0 v_{\ell''+|\widetilde{z}|+1}), \ldots, (\theta_0 u_{\ell'+|\widetilde{z}|})(\theta_0 v_{\ell''+|\widetilde{z}|}, \theta_0\widetilde{v}, \theta_0 v_{\ell''+|\widetilde{z}|+1}),\\ \quad (\theta_1 u_0)(\theta_1 v_0, \theta_0\widetilde{v}, \theta_0 v_{\ell''+|\widetilde{z}|+1}), \ldots, (\theta_k u_0)(\theta_k v_0, \theta_0\widetilde{v}, \theta_0 v_{\ell''+|\widetilde{z}|+1})\\ \quad (\theta_0 u_{\ell'+|\widetilde{z}|+1})(\theta_0 v_{\ell'+|\widetilde{z}|+1}, \theta_0\widetilde{v}, \theta_0 v_{\ell''+|\widetilde{z}|+1}).\end{array}$$

Since $y_{i,0}$ does not occur in $\widetilde{v}$ and $v_{\ell''+|\widetilde{z}|+1}$ (Lemma B.5), $\theta_0\widetilde{v}$ and $\theta_0 u_{\ell'+|\widetilde{z}|+1}$ are equivalent to $\theta_j\widetilde{v}$ and $\theta_j u_{\ell'+|\widetilde{z}|+1}$ respectively for any $j \in \{1, \ldots, k\}$. Therefore, the whole output of transformation is equivalent to:

$$\begin{array}{l}(\theta_0(u_0(v_0, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1})), \theta_0(\widetilde{u}(\widetilde{v}, v_{\ell''+|\widetilde{z}|})),\\ \quad \theta_0(u_{\ell'+1}(v_{\ell''+1}, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1})), \ldots, \theta_0(u_{\ell'+|\widetilde{z}|}(v_{\ell''+|\widetilde{z}|}, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1})),\\ \quad \theta_1(u_0(v_0, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1})), \ldots, \theta_k(u_0(v_0, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1})),\\ \quad \theta_0(u_{\ell'+|\widetilde{z}|+1}(v_{\ell'+|\widetilde{z}|+1}, \widetilde{v}, v_{\ell''+|\widetilde{z}|+1}))).\end{array}$$

Thus, we have the required result.

- Case Tr-NT: The result follows immediately from Tr-NT.
- Case Tr-AppG: In this case, we have $t = uv$ and:

$$\begin{array}{l}\widetilde{y} : \widetilde{\kappa_y}; \widetilde{z} \vdash_{\mathcal{N}} u : \mathtt{o}^{\ell'+1} \to \mathtt{o} \rightsquigarrow (u_0, \ldots, u_{\ell'+|\widetilde{z}|+2})\\ \widetilde{y} : \widetilde{\kappa_y}; \widetilde{z} \vdash_{\mathcal{N}} v : \mathtt{o} \rightsquigarrow (v_0, \ldots, v_{|\widetilde{z}|+1})\\ (\widetilde{t}, t_{m+1}) = (u_0 + u_1 \cdot v_0, u_2, \ldots, u_{\ell'+1}, u_{\ell'+2} + u_1 \cdot v_1, \ldots, u_{\ell'+|\widetilde{z}|+2} + u_1 \cdot v_{|\widetilde{z}|+1})\\ \kappa = \mathtt{o}^{\ell'} \to \mathtt{o}.\end{array}$$

By the induction hypothesis, we have:

$$\begin{array}{l}\emptyset; \widetilde{z}, x_1, \ldots, x_k \vdash_{\mathcal{N}} [\widetilde{s}/\widetilde{y}]u : \mathtt{o}^{\ell'+1} \to \mathtt{o} \rightsquigarrow (\theta_0 u_0, \ldots, \theta_0 u_{\ell'+|\widetilde{z}|+1}, \theta_1 u_0, \ldots, \theta_k u_0, \theta_0 u_{\ell'+|\widetilde{z}|+k+2})\\ \emptyset; \widetilde{z}, x_1, \ldots, x_k \vdash_{\mathcal{N}} [\widetilde{s}/\widetilde{y}]v : \mathtt{o} \rightsquigarrow (\theta_0 v_0, \ldots, \theta_0 v_{|\widetilde{z}|}, \theta_1 v_0, \ldots, \theta_k v_0, \theta_0 v_{|\widetilde{z}|+1}).\end{array}$$

By applying Tr-AppG, we obtain:

$$\begin{aligned}&\emptyset; \widetilde{z}, x_1, \dots, x_k \vdash_{\mathcal{N}} ([\widetilde{s}/\widetilde{y}]u)([\widetilde{s}/\widetilde{y}]v) : \mathtt{o}^{\ell'} \to \mathtt{o} \rightsquigarrow \\ &\quad (\theta_0 u_0 + \theta_0 u_1 \cdot \theta_0 v_0, \theta_0 u_2, \dots, \theta_0 u_{\ell'+1}, \\ &\quad \theta_0 u_{\ell'+2} + \theta_0 u_1 \cdot \theta_0 v_1, \dots, \theta_0 u_{\ell'+|\widetilde{z}|+1} + \theta_0 u_1 \cdot \theta_0 v_{|\widetilde{z}|}, \\ &\quad \theta_1 u_0 + \theta_0 u_1 \cdot \theta_1 v_0, \dots, \theta_k u_0 + \theta_0 u_1 \cdot \theta_k v_0, \theta_0 u_{\ell'+|\widetilde{z}|+2} + \theta_0 u_1 \cdot \theta_0 v_{|\widetilde{z}|+1}).\end{aligned}$$

Since u_1 does not contain any occurrence of $y_{i,0}$ (Lemma B.5), $\theta_0 u_1 = \theta_j u_1$ for any $j \in \{1, \dots, k\}$. Therefore, the output of the translation is equivalent to:

$$\begin{aligned}&(\theta_0(u_0 + u_1 \cdot v_0), \theta_0 u_2, \dots, \theta_0 u_{\ell'+1}, \\ &\quad \theta_0(u_{\ell'+2} + u_1 \cdot v_1), \dots, \theta_0(u_{\ell'+|\widetilde{z}|+1} + u_1 \cdot v_{|\widetilde{z}|}), \\ &\quad \theta_1(u_0 + u_1 \cdot v_0), \dots, \theta_k(u_0 + u_1 \cdot v_0), \theta_0(u_{\ell'+|\widetilde{z}|+2} + u_1 \cdot v_{|\widetilde{z}|+1})).\end{aligned}$$

Thus, we have the required result. □

We are now ready to prove that the translation relation is preserved by reductions (Lemmas B.11, B.12, and B.13 below).

Lemma B.11 (Subject Reduction)**.** *If $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} F\,\widetilde{s}\,\widetilde{t} : \mathtt{o} \rightsquigarrow (v_0, \dots, v_{|\widetilde{x}|+1})$ and $\mathcal{R}(F) = \lambda\widetilde{y}.\lambda\widetilde{z}.u_L \oplus_p u_R$ (where $\{\widetilde{x}\} \cap \{\widetilde{z}\} = \emptyset$), then there exist $w_{L,i}, w_{R,i}$ $(i \in \{0, \dots, |\widetilde{x}| + 1\})$ such that $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} \{\widetilde{t}/\widetilde{z}\}[\widetilde{s}/\widetilde{y}]u_d \rightsquigarrow (w_{d,0}, \dots, w_{d,|\widetilde{x}|+1})$ and $v_i \cong p w_{L,i} + (1-p) w_{R,i}$ for each $i \in \{0, \dots, |\widetilde{x}| + 1\}$.*

Proof. By the assumptions, we have:

$$\begin{aligned}&\emptyset; \widetilde{x} \vdash_{\mathcal{N}} s_i : \kappa_i \rightsquigarrow (s_{i,0}, \dots, s_{i,\mathtt{ar}(\kappa_i)+|\widetilde{x}|+1}) && \text{for each } i \in \{1, \dots, |\widetilde{s}|\} \\ &\emptyset; \widetilde{x} \vdash_{\mathcal{N}} t_i : \mathtt{o} \rightsquigarrow (t_{i,0}, \dots, t_{i,|\widetilde{x}|+1}) \text{ for each } i \in \{1, \dots, |\widetilde{z}|\} \\ &v'_0 = F_0(s_{1,0}, \dots, s_{1,\mathtt{ar}(\kappa_1)}, s_{1,\mathtt{ar}(\kappa_1)+|\widetilde{x}|+1}) \cdots (s_{|\widetilde{s}|,0}, \dots, s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})}, s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})+|\widetilde{x}|+1}) \\ &v'_j = F_j(s_{1,1}, \dots, s_{1,\mathtt{ar}(\kappa_1)}, s_{1,\mathtt{ar}(\kappa_1)+|\widetilde{x}|+1}) \cdots (s_{|\widetilde{s}|,1}, \dots, s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})}, s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})+|\widetilde{x}|+1}) \\ &\qquad && \text{for each } j \in \{1, \dots, |\widetilde{z}|\} \\ &v'_{|\widetilde{z}|+j} = F_0(s_{1,\mathtt{ar}(\kappa_1)+j}, s_{1,1}, \dots, s_{1,\mathtt{ar}(\kappa_1)}, s_{1,\mathtt{ar}(\kappa_1)+|\widetilde{x}|+1}) \cdots \\ &\qquad\qquad (s_{|\widetilde{s}|,\mathtt{ar}(\kappa_1)+j}, s_{|\widetilde{s}|,1}, \dots, s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})}, s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})+|\widetilde{x}|+1}) \\ &\qquad && \text{for each } j \in \{1, \dots, |\widetilde{x}|\} \\ &v'_{|\widetilde{z}|+|\widetilde{x}|+1} = F_0(s_{1,\mathtt{ar}(\kappa_1)+|\widetilde{x}|+1}, s_{1,1}, \dots, s_{1,\mathtt{ar}(\kappa_1)}, s_{1,\mathtt{ar}(\kappa_1)+|\widetilde{x}|+1}) \cdots \\ &\qquad\qquad (s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})+|\widetilde{x}|+1}, s_{|\widetilde{s}|,1} \cdots, s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})}, s_{|\widetilde{s}|,\mathtt{ar}(\kappa_{|\widetilde{s}|})+|\widetilde{x}|+1}) \\ &v_0 \cong v'_0 + v'_1 \cdot t_{1,0} + \cdots + v'_{|\widetilde{z}|} \cdot t_{|\widetilde{z}|,0} \\ &v_{|\widetilde{z}|+i} \cong v'_{|\widetilde{z}|+i} + v'_1 \cdot t_{1,i} + \cdots + v'_{|\widetilde{z}|} \cdot t_{|\widetilde{z}|,i} \text{ for each } i \in \{1, \dots, |\widetilde{x}| + 1\} \\ &\widetilde{y} : \widetilde{\kappa}; \widetilde{z} \vdash_{\mathcal{N}} u_d : \mathtt{o} \rightsquigarrow (u_{d,0}, \dots, u_{d,|\widetilde{z}|+1}) \text{ for } d \in \{L, R\}\end{aligned}$$

By applying the substitution lemma (Lemma B.10) to the last condition, we obtain:

$$\emptyset; \widetilde{z}, \widetilde{x} \vdash_{\mathcal{N}} [\widetilde{s}/\widetilde{y}]u_d : \mathtt{o} \rightsquigarrow (w'_{d,0}, \dots, w'_{d,|\widetilde{z}|+|\widetilde{x}|+1})$$

where

$$\begin{aligned}&(w'_{d,0}, \dots, w'_{d,|\widetilde{z}|+|\widetilde{x}|+1}) = (\theta_0 u_{d,0}, \dots, \theta_0 u_{d,|\widetilde{z}|}, \theta_1 u_{d,0}, \dots, \theta_{|\widetilde{x}|} u_{d,0}, \theta_0 u_{d,|\widetilde{z}|+1}) \\ &\theta_j = \theta_{1,j} \cdots \theta_{|\widetilde{s}|,j} \text{ for } j \in \{0, \dots, |\widetilde{x}|\} \\ &\theta_{i,0} = [s_{i,0}/y_{i,0}, \dots, s_{i,\mathtt{ar}(\kappa_i)}/y_{i,\mathtt{ar}(\kappa_i)}, s_{i,\mathtt{ar}(\kappa_i)+|\widetilde{x}|+1}/y_{i,\mathtt{ar}(\kappa_i)+1}] \text{ for } i \in \{1, \dots, |\widetilde{s}|\} \\ &\theta_{i,j} = [s_{i,\mathtt{ar}(\kappa_i)+j}/y_{i,0}, \dots, s_{i,\mathtt{ar}(\kappa_i)}/y_{i,\mathtt{ar}(\kappa_i)}, s_{i,\mathtt{ar}(\kappa_i)+|\widetilde{x}|+1}/y_{i,\mathtt{ar}(\kappa_i)+1}] \\ &\qquad\qquad \text{for } i \in \{1, \dots, |\widetilde{s}|\}, j \in \{1, \dots, |\widetilde{x}|\}.\end{aligned}$$

Then, we have $v'_j \cong pw'_{L,j'} + (1-p)w'_{R,j'}$. (Here, the only non-trivial case is for $j = |\widetilde{z}| + |\widetilde{x}| + 1$, where we need to show that $v'_j \cong p\theta_{|\widetilde{x}|+1}u_{L,0} + (1-p)\theta_{|\widetilde{x}|+1}u_{R,0}$ is equivalent to $pw'_{L,j'} + (1-p)w'_{R,j'}$; in this case, by induction on the structure of u_d, it follows that $\theta_{|\widetilde{x}|+1}u_{d,0} = \theta_0 u_{d,|\widetilde{z}|+1}$, which implies the required property) Let $\widetilde{w}_d$ be $(w_{d,0}, \ldots, w_{d,|\widetilde{x}|+1})$, where:

$$\begin{aligned} w_{d,0} &= w'_{d,0} + w'_{d,1} \cdot t_{1,0} + \cdots + w'_{d,|\widetilde{z}|} \cdot t_{|\widetilde{z}|,0} \\ w_{d,i} &= w'_{d,|\widetilde{z}|+i} + w'_{d,1} \cdot t_{1,i} + \cdots + w'_{d,|\widetilde{z}|} \cdot t_{|\widetilde{z}|,i} \\ &\qquad \text{for } i \in \{1, \ldots, |\widetilde{x}| + 1\}. \end{aligned}$$

Then, the required result is obtained by applying Tr-Sub to

$$\emptyset; \widetilde{z}, \widetilde{x} \vdash_{\mathcal{N}} [\widetilde{s}/\widetilde{y}]u_d : \mathtt{o} \rightsquigarrow (w'_{d,0}, \ldots, w'_{d,|\widetilde{z}|+|\widetilde{x}|+1}). \qquad \square$$

Lemma B.12 (Subject Reduction (for administrative steps))**.** *If $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} \{\widetilde{s}/\widetilde{z}\}x_i : \mathtt{o} \rightsquigarrow (t_0, \ldots, t_{|\widetilde{x}|+1})$ with $x_i \notin \{\widetilde{z}\}$, then $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} x_i : \mathtt{o} \rightsquigarrow (u_0, \ldots, u_{|\widetilde{x}|+1})$ for some u_j ($j \in \{0, \ldots, |\widetilde{x}| + 1\}$) such that $t_j \cong u_j$ for each $j \in \{0, \ldots, |\widetilde{x}| + 1\}$.*

Proof. By the assumption $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} \{\widetilde{s}/\widetilde{z}\}x_i : \mathtt{o} \rightsquigarrow (t_0, \ldots, t_{|\widetilde{x}|+1})$, we have:

$$\begin{aligned} &\emptyset; \widetilde{z}, \widetilde{x} \vdash_{\mathcal{N}} x_i : \mathtt{o} \rightsquigarrow (0^{i+|\widetilde{z}|+1}, 1, 0^{|\widetilde{x}|-i+1}) \\ &\emptyset; \widetilde{x} \vdash_{\mathcal{N}} s_j : \mathtt{o} \rightsquigarrow (s_{j,0}, \ldots, s_{j,|\widetilde{x}|+1}) \text{ (for each } j \in \{1, \ldots, |\widetilde{s}|\}) \\ &t_i = 1 + \Sigma_{j=1}^{|\widetilde{s}|} 0 \cdot s_{j,i} \cong 1 \qquad t_{i'} \cong 0 \text{ for } i' \neq i \end{aligned}$$

Since $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} x_i : \mathtt{o} \rightsquigarrow (0^i, 1, 0^{|\widetilde{x}|-i+1})$, we have the required condition for: $u_i = 1$ and $u_j = 0$ for $j \neq i$. $\square$

Lemma B.13 (Subject Reduction (for administrative steps (ii)))**.** *If $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} \{\widetilde{s}/\widetilde{z}\}z_i : \mathtt{o} \rightsquigarrow (t_0, \ldots, t_{|\widetilde{x}|+1})$, then $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} s_i : \mathtt{o} \rightsquigarrow (u_0, \ldots, u_{|\widetilde{x}|+1})$ for some u_j ($j \in \{0, \ldots, |\widetilde{x}| + 1\}$) such that $t_j \cong u_j$ for each $j \in \{0, \ldots, |\widetilde{x}| + 1\}$.*

Proof. By the assumption $\emptyset; \widetilde{x} \vdash_{\mathcal{N}} \{\widetilde{s}/\widetilde{z}\}z_i : \mathtt{o} \rightsquigarrow (t_0, \ldots, t_{|\widetilde{x}|+1})$, we have:

$$\begin{aligned} &\emptyset; \widetilde{z}, \widetilde{x} \vdash_{\mathcal{N}} z_i : \mathtt{o} \rightsquigarrow (0^i, 1, 0^{|\widetilde{z}|-i+|\widetilde{x}|+1}) \\ &\emptyset; \widetilde{x} \vdash_{\mathcal{N}} s : \mathtt{o} \rightsquigarrow (s_0, \ldots, s_{|\widetilde{x}|+1}) \\ &t_j \cong 0 + 1 \cdot s_j \cong s_j \text{ for each } j \in \{0, \ldots, |\widetilde{x}| + 1\}. \end{aligned}$$

Thus, the result holds for $u_j = s_j$. $\square$

We are now ready to prove Theorem 4.12, restricted to recursion-free PHORS (the definition of recursion-free PHORS is found in Section B.1).

Lemma B.14. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ be a recursion-free PHORS, and ρ be the least solution of $\mathcal{E}^{\mathtt{ref}}_{\mathcal{G}}$. Then, $\mathcal{P}(\mathcal{G}, S\,\mathtt{e}) = \rho(S_1)$.*

Proof. By Lemma B.7, it suffices to show that $\emptyset; x \vdash_{\mathcal{N}} t \rightsquigarrow (t_0, t_1, t_2)$ implies $\mathcal{P}_{\mathtt{es}}(\mathcal{G}, t, x) \cong t_1$. This follows by induction on $\flat(t)$, the length of the longest reduction sequence from t by $\xrightarrow{d,p}_{\mathtt{es}}$; note that $\flat(t)$ is well defined because t is finitely branching and strongly normalizing with respect to $\xrightarrow{d,p}_{\mathtt{es}}$; the strong normalization follows from Lemma B.6 and the fact that there can be no infinite sequence of consecutive administrative reductions (in fact, each administrative reduction consumes one explicit substitution).

Suppose $\flat(t) = 0$. Then, t is either x (in which case, $t_1 = 1$) or Ω (in which case, $t_1 = 0$). Thus, the result follows immediately.

If $\flat(t) > 0$, we perform a case analysis on a reduction of t. If $t \xrightarrow{\epsilon,1}_{\mathtt{es}} t'$, then by Lemmas B.12 and B.13, $\emptyset; x \vdash_{\mathcal{N}} t \rightsquigarrow (t'_0, t'_1, t'_2)$ and $t_1 \cong t'_1$ for some (t'_0, t'_1, t'_2). By the induction hypothesis, $\mathcal{P}_{\mathtt{es}}(\mathcal{G}, t', x) \cong t'_1$. Therefore, we have $\mathcal{P}_{\mathtt{es}}(\mathcal{G}, t, x) = \mathcal{P}_{\mathtt{es}}(\mathcal{G}, t', x) \cong t'_1 \cong t_1$ as required.

If $t \xrightarrow{d,p}_{\mathtt{es}} t'$ for $d \in \{L, R\}$, then t must be of the form $E[F\, \widetilde{s}\, \widetilde{t}]$, $\mathcal{R}(F) = \lambda\widetilde{y}.\lambda\widetilde{z}.u_L \oplus_p u_R$, $t \xrightarrow{L,p}_{\mathtt{es}} t_L$, and $t \xrightarrow{R,1-p}_{\mathtt{es}} t_R$ with $t_L = E[\{\widetilde{t}/\widetilde{z}\}[\widetilde{s}/\widetilde{y}]u_L]$ and $t_R = E[\{\widetilde{t}/\widetilde{z}\}[\widetilde{s}/\widetilde{y}]u_R]$. By Lemma B.11, there exist $(t_{L,0}, t_{L,1}, t_{L,2})$ and $(t_{R,0}, t_{R,1}, t_{R,2})$ such that $\emptyset; x \vdash_{\mathcal{N}} t_L \rightsquigarrow (t_{L,0}, t_{L,1}, t_{L,2})$ and $\emptyset; x \vdash_{\mathcal{N}} t_R \rightsquigarrow (t_{R,0}, t_{R,1}, t_{R,2})$ with $t_1 \cong pt_{L,1} + (1-p)t_{R,1}$. Thus, we have $\mathcal{P}_{\mathtt{es}}(\mathcal{G}, t, x) = p\mathcal{P}_{\mathtt{es}}(\mathcal{G}, t_L, x) + (1-p)\mathcal{P}_{\mathtt{es}}(\mathcal{G}, t_R, x) \cong pt_{L,1} + (1-p)t_{R,1} \cong t_1$ as required. □

We are now ready to prove Theorem 4.12.

Proof of Theorem 4.12. Consider the finite approximation $\mathcal{G}^{(k)}$ (defined in Section B.1). Then, we have

$$\begin{aligned}&\mathcal{P}(\mathcal{G}, S\,\mathsf{e}) = \textstyle\bigsqcup_k \mathcal{P}(\mathcal{G}^{(k)}, S^{(k)}\,\mathsf{e}) = \bigsqcup_k \mathbf{lfp}(\mathcal{F}_{\mathcal{G}^{(k)}})(S_1^{(k)})\\ &= \textstyle\bigsqcup_k \mathcal{F}_{\mathcal{G}}^k(\bot)(S_1) = \mathbf{lfp}(\mathcal{F}_{\mathcal{G}})(S_1) = \rho_{\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}}(S_1)\end{aligned}$$

as required. Here $\mathcal{F}_{\mathcal{G}}$ is the functional associated with fixpoint equations $\mathcal{E}_{\mathcal{G}}^{\mathtt{ref}}$, as defined in Section 4.1, and we can use essentially the same reasoning as in the proofs of Lemma B.3 and Theorem 4.4. □