

On the Termination Problem for Probabilistic Higher-Order Recursive Programs

Naoki Kobayashi
The University of Tokyo

Ugo Dal Lago
University of Bologna

Charles Grellois
Aix-Marseille University

November 7, 2018

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 (but possibly incomplete) procedure for approximately computing the termination probability. We have implemented the procedure for order-2 PHORSs, and confirmed that the procedure works well through preliminary experiments that are reported at the end of the article.

1 Introduction

Computer science has interacted with probability theory in many fruitful ways, since its early days [20]. 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 [53] or in cryptography [28]. Probability theory is also a key tool in artificial intelligence, for instance in probabilistic graphical models [49], one of the most successful paradigms in machine learning.

This trend of a rise of probability theory’s importance in computer science has been followed by the programming languages 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, the simplest one consisting in allowing programs to sample from distributions by way of dedicated operators. Fair, binary probabilistic choice is for instance perfectly sufficient to obtain universality if the underlying programming language is itself universal [61, 18]. This has been the path followed in probabilistic λ -calculi, from the pioneering work by Saheb-Djahromi dating back to the seventies [58] to the most recent contributions (e.g., [19, 21, 17, 33]), through the seminal work, e.g., of Jones and Plotkin [38].

Probabilistic systems and programs can be analyzed by *model checking techniques*, themselves forming a very active research field (see [3, 13] for a survey). Over the last decade, there has been much interest and progress in model checking of probabilistic *recursive* programs [23, 25, 7, 6], 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, however, is capable of handling *higher-order* functions, which are a natural feature in functional languages. This is in sharp contrast to 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. [55, 32, 44, 46, 48, 31, 30, 64, 59]). There have been some studies on the termination of probabilistic higher-order programs [16], 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

by Dal Lago
& Grellois

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_coin` for generating `true` or `false` with probability $\frac{1}{2}$.

P_1 : `let rec f() = if flip_coin() then () else f() in f()`

$$X = \frac{1}{2} + \frac{1}{2}X \rightarrow X = 1$$

The program almost surely terminates (i.e., terminates with probability 1), but if we ignore the probabilistic aspect and model `flip_coin()` as a *non-deterministic* (rather than *probabilistic*) primitive, then we would conclude that the program may not terminate. The following program makes use of an interesting combination of probabilistic choice and higher-order functions:

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

$$g(x) = \frac{1}{2} + \frac{1}{2}x \cdot g(x) \quad \text{rational}$$

$$\rightarrow g(x) = \frac{1}{1-2x}$$

$$g(g(1))$$

The function `listgen` above takes a generator f of elements as an argument, and generates a list of elements generated by f . Thus, the whole program generates a list of lists of Booleans. 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, by the way, is not at all an artificial task, being central to, e.g., random test generation [56, 52].

As a model of probabilistic higher-order programs, we first introduce PHORS, a probabilistic extension of higher-order recursion schemes (HORS) [41, 55]. 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 existence of algorithmic solutions to some of the verification problems (in fact, we can decide 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 [41, 32], PHORS can be considered a higher-order extension of probabilistic pushdown systems [7, 6] and of recursive Markov chains [66], 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 [62] 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 [55], intersection types [43, 46] and Krivine machines [59]) 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 model checking of 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 safety property verification problems to (may-)termination problems (by encoding the safety property as a termination). We can also verify certain liveness properties, by encoding a good event as a termination and checking that the termination probability is 1 or close to 1. As we will see in Section 2, the two questions above on the `listgen` program can also be reduced to problems of computing the termination probability of 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 [66, 6].

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 is terminating with probability 1, is undecidable for 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 [23]. The proof of undecidability is based on a reduction from the undecidability of Hilbert's tenth problem (i.e. unsolvability of Diophantine equations) [50]. 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 $\{\mathcal{G} \mid \text{Pr}(\mathcal{G}) \geq r\}$ (where $\text{Pr}(\mathcal{G})$ denotes the termination probability of \mathcal{G}) is not recursively enumerable. Note, however, that this negative result does not preclude the possibility to compute the termination probability with arbitrary precision; there

They may still be algebraic?

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 $(r, r + \epsilon)$.

As a positive result for 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 [23] 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 of 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.

This paper’s contributions can thus be summarized as follows:

1. Formalization of probabilistic higher-order recursion schemes (PHORS) and their termination probabilities.
2. A proof of undecidability of the almost sure termination problem for PHORS (of order 2 or higher).
3. Fixpoint characterization of the termination probability of PHORS, which immediately yields the semi-decidability of the lower-bound problem.
4. A sound procedure for computing an upper-bound to the termination probability of order-2 PHORS, accompanied by an implementation and preliminary experiments with promising results.

The rest of this paper is structured as follows. Section 2 formalizes PHORS and their termination probabilities. Section 3 proves the undecidability of almost sure termination of order-2 PHORSs. Section 4 provides fixpoint characterizations of the termination probability of PHORSs. Section 5 presents a sound procedure for computing an upper-bound of the termination probability. Section 6 reports on an implementation and preliminary experiments on computing lower/upper bounds of the termination probability based on the proposed method. Section 7 discusses related work, and Section 8 concludes the paper.

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

This section introduces *probabilistic higher-order recursion schemes* (PHORS), an extension of higher-order recursion schemes [41, 55] in which programs can perform discrete probabilistic choice and proceed according to its outcome. Higher-order recursion schemes are usually treated as generators of infinite trees, but as we are interested only in the termination probability, we consider only nullary tree constructors **halt** 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:

$$\kappa ::= \mathbf{o} \mid \kappa_1 \rightarrow \kappa_2.$$

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

$$\text{order}(\mathbf{o}) = 0 \quad \text{order}(\kappa_1 \rightarrow \kappa_2) = \max(\text{order}(\kappa_1) + 1, \text{order}(\kappa_2)).$$

We often write $\mathbf{o}^\ell \rightarrow \mathbf{o}$ for $\underbrace{\mathbf{o} \rightarrow \cdots \rightarrow \mathbf{o}}_\ell \rightarrow \mathbf{o}$, and abbreviate $\kappa_1 \rightarrow \cdots \rightarrow \kappa_k \rightarrow \kappa'$ to $\tilde{\kappa} \rightarrow \kappa'$. The set of *applicative terms*, ranged over by t , is given by:

$$t ::= \mathbf{halt} \mid \Omega \mid x \mid t_1 t_2,$$

where **halt** and Ω are (the only) constants of type \circ and x ranges over a set of variables. Intuitively, **halt** 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.

$$\frac{}{\mathcal{K} \vdash \mathbf{halt} : \circ} \quad \frac{}{\mathcal{K} \vdash \Omega : \circ} \quad \frac{\mathcal{K}(x) = \kappa}{\mathcal{K} \vdash x : \kappa} \quad \frac{\mathcal{K} \vdash t_1 : \kappa_2 \rightarrow \kappa \quad \mathcal{K} \vdash t_2 : \kappa_2}{\mathcal{K} \vdash t_1 t_2 : \kappa}$$

Definition 2.1 (PHORS) A probabilistic higher-order recursion scheme (*PHORS for short*) 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, \dots) to the set of types.

2. \mathcal{R} is a map from $\text{dom}(\mathcal{N})$ to terms of the form $\lambda x_1 \dots \lambda x_k. t_L \oplus_p t_R$, where $p \in [0, 1]$, and t_L, t_R are applicative terms. If $\mathcal{N}(F) = \kappa_1 \rightarrow \dots \rightarrow \kappa_k \rightarrow \circ$, $\mathcal{R}(F)$ must be of the form $\lambda x_1 \dots \lambda x_k. t_L \oplus_p t_R$, where $\mathcal{N}, x_1 : \kappa_1, \dots, x_k : \kappa_k \vdash t_L : \circ$ and $\mathcal{N}, x_1 : \kappa_1, \dots, x_k : \kappa_k \vdash t_R : \circ$.

3. $S \in \text{dom}(\mathcal{N})$, called the start symbol, is a distinguished non-terminal that satisfies $\mathcal{N}(S) = \circ$. The order of a PHORS $(\mathcal{N}, \mathcal{R}, S)$ is the highest order of the types of its non-terminals, i.e., $\max_{F \in \text{dom}(\mathcal{N})} \text{order}(\mathcal{N}(F))$.

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

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:

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

We write $\xRightarrow{\pi,p}_{\mathcal{G}}$ for the relational composition of $\xrightarrow{d_1,p_1}_{\mathcal{G}}, \dots, \xrightarrow{d_n,p_n}_{\mathcal{G}}$, when $\pi = d_1 \dots d_n$ 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 $S \xRightarrow{\pi,p}_{\mathcal{G}} \mathbf{halt}$. For an applicative term t , we define $\mathcal{P}(\mathcal{G}, t, \pi)$ by:

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

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

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

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.

Example 2.1 Let \mathcal{G}_1 be the order-1 PHORS $(\mathcal{N}_1, \mathcal{R}_1, S)$, where:

$$\mathcal{N}_1 = \{S \mapsto \circ, F \mapsto \circ \rightarrow \circ\} \quad \mathcal{R}_1 = \{S = F \mathbf{halt} \oplus_1 \Omega, \quad F x = x \oplus_p F(F x)\}.$$

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

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

Thus, we have $S \xRightarrow{LRL, p^2(1-p)} \mathbf{halt}$. 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 reported in Figure 1, using the notational conventions from [23]. \square

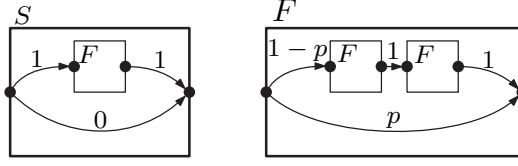


Figure 1: A Recursive Markov Chain Modeling \mathcal{G}_1 .

As the previous example suggests, there is a relationship between recursive Markov Chains and order-1 PHORS. In fact, one can prove that *for every* recursive Markov Chain A and for every reachability problem for A there is a corresponding order-1 PHORS \mathcal{G}_A which faithfully models it, see the Appendix A.1 for more details.

Example 2.2 Let \mathcal{G}_2 be the order-2 PHORS $(\mathcal{N}_2, \mathcal{R}_2, S)$ where:

$$\begin{aligned} \mathcal{N}_2 &= \{S \mapsto \circ, H \mapsto \circ \rightarrow \circ, F \mapsto (\circ \rightarrow \circ) \rightarrow \circ, D \mapsto (\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ\} \\ \mathcal{R} &= \{S = (FH) \oplus_1 \text{halt}, Hx = x \oplus_{\frac{1}{2}} \Omega, Fg = (g\text{halt}) \oplus_{\frac{1}{2}} (F(Dg)), Dgx = (g(gx)) \oplus_1 \Omega\}. \end{aligned}$$

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

$$S \xrightarrow{L,1} FH \xrightarrow{R,\frac{1}{2}} F(DH) \xrightarrow{L,\frac{1}{2}} DH\text{halt} \xrightarrow{L,1} H(H\text{halt}) \xrightarrow{L,\frac{1}{2}} H\text{halt} \xrightarrow{L,\frac{1}{2}} \text{halt}.$$

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. \square

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

Theorem 2.1 *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 $\mathcal{P}(\mathcal{G}, n) > r$ for some n , and that $\mathcal{P}(\mathcal{G}, n)$ is computable. \square

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 2.1 *Given a PHORS \mathcal{G} , by replacing each probabilistic operator \oplus_p s.t. $0 < p < 1$ with a binary tree constructor \mathbf{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}' . Then $\mathcal{P}(\mathcal{G}) = 0$ if and only if the tree generated by \mathcal{G}' has no finite path to **halt**. Thus, by [47] (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 HORS \mathcal{G}' . If the tree of \mathcal{G}' has neither Ω nor infinite path (which is decidable), then $\mathcal{P}(\mathcal{G})=1$, but the converse does not hold.*

Remark 2.2 *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 $F \tilde{x} = C[t_L \oplus_p t_R]$, we can normalize it to $F \tilde{x} = C[G \tilde{x}]$, where G is defined by $G \tilde{x} = t_L \oplus_p t_R$. Keeping this in mind, in examples, we sometimes allow probabilistic choices to allow inside terms. In fact, a PHORS can be considered as a term (of type \circ) of a probabilistic extension of the (call-by-name) λY -calculus [62]. We define the set of probabilistic λY terms by:*

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

Here, \oplus_p is a probabilistic choice operator of type $\circ \rightarrow \circ \rightarrow \circ$, and other terms are simply-typed in the usual way. Then, PHORSs 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 2.3 We adopt the call-by-name semantics, and allow probabilistic choices only on terms of type \circ . The call-by-value semantics, as well as probabilistic choices at higher-order types can be modeled by applying the CPS transformation. A PHORS does not have data other than functions, but as in ordinary HORS [44], elements of a finite set (such as Booleans) can be modeled by using Church encoding.

We provide a few more examples below.

Example 2.3 Recall the list generator example in Section 1.

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

The termination of the program above is equivalent to that of the following program, obtained by replacing the output of each function with the unit value $()$.

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

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

$$S = \text{Listgen} (\text{Listgen Boolgen}) \text{halt} \\ \text{Boolgen } k = k \oplus_{\frac{1}{2}} k \quad \text{Listgen } f \ k = k \oplus_{\frac{1}{2}} (f(\text{Listgen } f \ k))$$

It is not difficult to confirm that $\mathcal{P}(\mathcal{G}_3) = 1$ (using the fixpoint characterization given in Section 4). The following is a variation, which generates ternary trees instead of lists:

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

The corresponding PHORS (that captures the termination probability of the original program) \mathcal{G}_4 is:

$$S = \text{Treegen Boolgen} \text{halt} \quad \text{Boolgen } k = k \oplus_{\frac{1}{2}} k \\ \text{Treegen } f \ k = k \oplus_{\frac{1}{2}} (f(\text{Treegen } f (\text{Treegen } f (\text{Treegen } f \ k))))$$

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

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

```
let boolgen() = if flip_coin() then true else false in
let rec treegen p f = if flip_coinp(p) then Leaf
                     else Node(f(), treegen  $\frac{p+1}{2}$  f, treegen  $\frac{p+1}{2}$  f, treegen  $\frac{p+1}{2}$  f) in
treegen  $\frac{1}{2}$  boolgen
```

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

$$S = \text{Treegen } H \text{ Boolgen} \text{halt} \quad \text{Boolgen } k = k \oplus_{\frac{1}{2}} k \\ H \ x \ y = x \oplus_{\frac{1}{2}} y \quad G \ p \ x \ y = x \oplus_{\frac{1}{2}} (p \ x \ y) \\ \text{Treegen } p \ f \ k = p \ k \ (f(\text{Treegen } (G \ p) \ f (\text{Treegen } (G \ p) \ f (\text{Treegen } (G \ p) \ f \ k))))$$

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, and returns a probabilistic function $\lambda x.\lambda y.x \oplus_{\frac{1}{2}} (p \ x \ y)$, which chooses the first argument with probability $\frac{p+1}{2}$. As expected, \mathcal{G}_5 is almost surely terminating. \square

Example 2.4 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() = if flip_coin() then true else false in
let rec loop() = loop() in
let rec listgen_even f = if flip_coin() then () else (f();listgen_odd f) in
    and listgen_odd f = if flip_coin() then loop() else (f();listgen_even f) in
    listgen_even(boolgen)
```

Here, we have duplicated `listgen` to `list_even` and `list_odd`, which are expected to simulate the generation of even and odd lists respectively. Thus, the then-branches of `list_even` and `list_odd` 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 :

$$S = \text{ListgenEven Boolgen halt} \quad \text{Boolgen } k = k \oplus_{\frac{1}{2}} k$$

$$\text{ListgenEven } f \ k = k \oplus_{\frac{1}{2}} (f(\text{ListgenOdd } f \ k)) \quad \text{ListgenOdd } f \ k = \Omega \oplus_{\frac{1}{2}} (f(\text{ListgenEven } f \ k)).$$

The termination probability of the PHORS is

$$\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{4} + \frac{1}{2} \cdot \left(\frac{1}{4}\right)^2 + \dots = \frac{2}{3}.$$

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.

$$S = \text{ListgenEven Boolgen halt} \quad \text{Boolgen } k_1 \ k_2 = k_1 \oplus_{\frac{1}{2}} k_2$$

$$\text{ListgenEven } f \ k = k \oplus_{\frac{1}{2}} (f(\text{ListgenOdd } f \ k)(\text{ListgenEven } f \ k))$$

$$\text{ListgenOdd } f \ k = \Omega \oplus_{\frac{1}{2}} (f(\text{ListgenEven } f \ k)(\text{ListgenOdd } f \ k)).$$

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 *ListgenEven* (*ListgenOdd*, resp.) is called when the number of `true`'s generated so far is even (odd, resp.). The termination probability of the PHORS above is $\frac{3}{4}$. \square

The following example allows us to see a standard program transformation for randomised algorithms 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 uses of the scheme above, please refer to [34]).

Example 2.5 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 $1 - p$. 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 *Determinize g* almost surely terminates for $g = \lambda y. \lambda z. y \oplus_p z$, where *Determinize* is defined by:

$$\text{Determinize } g = g \text{ halt } (\text{Determinize } g).$$

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

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

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

Remark 2.4 Although PHORS does 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 2.5 realizes the average operation $\frac{p_1+p_2}{2}$. The multiplication can be represented by $Mult\ p_1\ p_2\ x\ y = p_1\ (p_2\ x\ y)\ y$.

3 Undecidability of Almost Sure Termination of Order-2 PHORS

Is it still undecidable for $p = 1/2$ in \oplus_p ?

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, even for order-2 PHORSs. The proof is by reduction from the undecidability of Hilbert's 10th problem [50] (i.e. unsolvability of Diophantine equations). Note that almost sure termination of an order-1 PHORS is decidable, as order-1 PHORSs are essentially equi-expressive with probabilistic pushdown systems and recursive Markov chains [23, 25, 7, 6]. 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 [23]. Thus, our undecidability result for order-2 PHORSs 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 multivariate polynomials $P(x_1, \dots, x_k)$ and $Q(x_1, \dots, x_k)$ with non-negative integer coefficients, whether $P(x_1, \dots, x_k) < Q(x_1, \dots, x_k)$ for some $x_1, \dots, x_k \in \mathbf{Nat}$ is undecidable. More precisely, the set of pairs of polynomials: $\{(P(x_1, \dots, x_k), Q(x_1, \dots, x_k)) \mid \forall x_1, \dots, x_k \in \mathbf{Nat}. P(x_1, \dots, x_k) \geq Q(x_1, \dots, x_k)\}$ is not recursively enumerable.*

Proof. Let $D(x_1, \dots, x_k)$ be a multivariate polynomial with integer coefficients. Then, for all natural numbers $x_1, \dots, x_k \in \mathbf{Nat}$, $D(x_1, \dots, x_k) = 0$ if and only if $(D(x_1, \dots, x_k))^2 - 1 < 0$. Any such polynomial $(D(x_1, \dots, x_k))^2 - 1$ may be rewritten as $P(x_1, \dots, x_k) - Q(x_1, \dots, x_k)$, where $P(x_1, \dots, x_k)$ and $Q(x_1, \dots, x_k)$ have only non-negative integer coefficients. Then, $D(x_1, \dots, x_k) = 0$ if and only if $P(x_1, \dots, x_k) < Q(x_1, \dots, x_k)$. Since whether $D(x_1, \dots, x_k) = 0$ for some $x_1, \dots, x_k \in \mathbf{Nat}$ is undecidable [50], it is also undecidable whether $P(x_1, \dots, x_k) < Q(x_1, \dots, x_k)$ for some $x_1, \dots, x_k \in \mathbf{Nat}$. Furthermore, since $\{(P(x_1, \dots, x_k), Q(x_1, \dots, x_k)) \mid \exists x_1, \dots, x_k \in \mathbf{Nat}. P(x_1, \dots, x_k) < Q(x_1, \dots, x_k)\}$ is recursively enumerable, its complement $\{(P(x_1, \dots, x_k), Q(x_1, \dots, x_k)) \mid \forall x_1, \dots, x_k \in \mathbf{Nat}. P(x_1, \dots, x_k) \geq Q(x_1, \dots, x_k)\}$ is not recursively enumerable. \square

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 $P(x_1, \dots, x_k) < Q(x_1, \dots, x_k)$ for some x_1, \dots, 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 $P(x_1, \dots, x_k)$ and $Q(x_1, \dots, x_k)$ be 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, \dots, n_k) , and for each tuple, spawn a process $\text{TestLt}(P(n_1, \dots, n_k))(Q(n_1, \dots, n_k))$ with non-zero probability, where $\text{TestLt } m_1 m_2$ 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:

$$\begin{aligned} S &= \text{Loop Zero} \cdots \text{Zero}. \\ \text{Loop } x_1 \cdots x_k &= (\text{TestLt}(P x_1 \cdots x_k)(Q x_1 \cdots x_k)) \\ &\quad \oplus_{\frac{1}{2}} (\text{Loop}(\text{Succ } x_1) \cdots x_k) \oplus_{\frac{1}{2}} \cdots \oplus_{\frac{1}{2}} (\text{Loop } x_1 \cdots (\text{Succ } x_k)). \end{aligned}$$

Here, for readability, we have extended the righthand sides of rules to choices on n terms:

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

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

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

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

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

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

$$\mathcal{N}(\text{Zero}) = \text{CT} \quad \mathcal{N}(\text{Succ}) = \text{CT} \rightarrow \text{CT} \quad \mathcal{N}(\text{Add}) = \mathcal{N}(\text{Mult}) = \text{CT} \rightarrow \text{CT} \rightarrow \text{CT},$$

where $\text{CT} = (\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ$. Note that the order of Church numerals is 2, and those of operations Succ , Add , and Mult are 3. By using them, we can also 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 TestLt , so that $\text{TestLt } m_1 m_2$ is non-AST if and only if $m_1 < m_2$. Since $\mathcal{G}_3^{P,Q}$ runs $\text{TestLt}(P n_1 \cdots n_k)(Q n_1 \cdots n_k)$ for each tuple of Church numerals (n_1, \dots, n_k) with non-zero probability, $\mathcal{G}_3^{P,Q}$ is non-AST if and only if $P(n_1, \dots, n_k) < Q(n_1, \dots, n_k)$ for *some* natural numbers n_1, \dots, n_k . The key ingredient used for the construction of TestLt is the function CheckHalf of type $(\circ \rightarrow \circ \rightarrow \circ) \rightarrow \circ$, defined as follows:

$$\text{CheckHalf } g = F' \ g \ \text{halt} \quad F' \ g \ x = g \ x \ (F' \ g \ (F' \ g \ x)).$$

Here, F' above is a parameterized version of F in Example 2.1: $F' \oplus_p$ (where \oplus_p is treated as a function of type $\circ \rightarrow \circ \rightarrow \circ$, which chooses the first argument with probability p and the second one with $1 - p$) corresponds to F . As discussed in Example 2.1, $F \ \text{halt}$ is non-AST if and only if $p < \frac{1}{2}$. Thus, $\text{CheckHalf } g = F' \ g \ \text{halt}$ (which is equivalent to $F \ \text{halt}$ 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 $\circ \rightarrow \circ \rightarrow \circ$ that chooses the first argument with probability smaller than $\frac{1}{2}$ if and only if $m_1 < m_2$. Then, TestLt can be defined by:

$$\text{TestLt } m_1 \ m_2 = \text{CheckHalf}(\text{CheckLt } m_1 \ m_2).$$

Finally, CheckLt can be defined by:

$$\begin{aligned} \text{CheckLt } m_1 \ m_2 \ x \ y &= (\text{NatToPr } m_1 \ x \ y) \oplus_{\frac{1}{2}} (\text{NatToPr } m_2 \ y \ x). \\ \text{NatToPr } m \ x \ y &= m \ (H \ x) \ y. & H \ x \ y &= x \oplus_{\frac{1}{2}} y. \end{aligned}$$

Let us write $[m]$ for the natural number represented by a Church numeral m . For a Church numeral m , $\text{NatToPr } m \ x \ y$ (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 $\text{CheckLt } m_1 \ m_2 \ x \ y$ 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 Appendix B 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 $P(x_1, \dots, x_k) < Q(x_1, \dots, x_k)$ holds for some $x_1, \dots, x_k \in \mathbf{Nat}$.

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 $NatToPr\ m$, 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 $\mathbf{CT} = (\mathbf{o} \rightarrow \mathbf{o}) \rightarrow \mathbf{o} \rightarrow \mathbf{o}$, whereas $NatToPr\ m$ has an order-1 type $\mathbf{o} \rightarrow \mathbf{o} \rightarrow \mathbf{o}$. This is ultimately what allows to strictly decrease the order.

Based on the idea above, we replace $TestLt$ with $TestLtPr$, which now takes probabilistic functions of type $\mathbf{o} \rightarrow \mathbf{o} \rightarrow \mathbf{o}$ as arguments:

$$TestLtPr\ g_1\ g_2 = CheckHalf\ (CheckLtPr\ g_1\ g_2). \quad CheckLtPr\ g_1\ g_2\ x\ y = (g_1\ x\ y) \oplus_{\frac{1}{2}} (g_2\ y\ x).$$

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, $TestLtPr\ (NatToPr\ (P\ n_1 \dots n_k))\ (NatToPr\ (Q\ n_1 \dots n_k))$ is non-AST if and only if $P(n_1, \dots, n_k) < Q(n_1, \dots, n_k)$.

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

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

Basically, a natural number m is encoded as a probabilistic function of type $\mathbf{o} \rightarrow \mathbf{o} \rightarrow \mathbf{o}$, which chooses the first and second arguments with probabilities $1 - \frac{1}{2^m}$ and $\frac{1}{2^m}$ respectively. Notice that $AddPr\ (NatToPr\ m_1)\ (NatToPr\ m_2)$ is equivalent to $NatToPr\ (Add\ m_1\ m_2)$, because the probability that $AddPr\ (NatToPr\ m_1)\ (NatToPr\ m_2)\ x\ y$ 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, \dots, n_k in the top-level loop, we pass around the PF encodings of the values of $n_1^{i_1} \dots n_k^{i_k}$ for each $i_1 \leq d_1, \dots, i_k \leq d_k$, where d_1, \dots, d_k respectively are the largest degrees of $P(x_1, \dots, x_k) + Q(x_1, \dots, x_k)$ in x_1, \dots, x_k . We thus define the start symbol S of $\mathcal{G}_2^{P,Q}$ by:

$$\begin{aligned} S &= LoopPr\ OnePr\ \underbrace{ZeroPr \dots ZeroPr}_{(d_1+1) \dots (d_k+1)-1 \text{ times}} \\ LoopPr\ \tilde{x} &= (TestLtPr\ (P'\ \tilde{x})\ (Q'\ \tilde{x})) \oplus_{\frac{1}{2}} (LoopPr\ (Inc_{1,(0,\dots,0)}\ \tilde{x}) \dots (Inc_{1,(d_1,\dots,d_k)}\ \tilde{x})) \\ &\quad \oplus_{\frac{1}{2}} \dots \oplus_{\frac{1}{2}} (LoopPr\ (Inc_{k,(0,\dots,0)}\ \tilde{x}) \dots (Inc_{k,(d_1,\dots,d_k)}\ \tilde{x})). \end{aligned}$$

Here, \tilde{x} denotes the sequence of $(d_1 + 1) \dots (d_k + 1)$ variables $x_{(0,\dots,0)}, \dots, x_{(d_1,\dots,d_k)}$, consisting of $x_{(i_1,\dots,i_k)}$ for each $i_1 \in \{0, \dots, d_1\}, \dots, i_k \in \{0, \dots, d_k\}$. Each variable $x_{(i_1,\dots,i_k)}$ holds (the PF encoding of) the value of $n_1^{i_1} \dots n_k^{i_k}$. The functions P' and Q' are the PF encoding version of the polynomials P and Q . Since P and Q can be represented as linear combinations of monomials $x_1^{i_1} \dots x_k^{i_k}$ for $i_1 \leq d_1, \dots, i_k \leq d_k$, 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: $P'\ \tilde{x}\ y\ z = AddPr\ x_{(2,0)}\ (AddPr\ x_{(1,1)}\ x_{(1,1)})\ y\ z$. The function $Inc_{j,(i_1,\dots,i_k)}\ \tilde{x}$ represents the PF encoding of $n_1^{i_1} \dots (n_j + 1)^{i_j} \dots n_k^{i_k}$, assuming that \tilde{x} represents (the PF encoding of) the values $n_1^0 \dots n_k^0, \dots, n_1^{d_1} \dots n_k^{d_k}$. Note that $Inc_{j,(i_1,\dots,i_k)}$ can also be defined by using $ZeroPr$ and $AddPr$, since $x_1^{i_1} \dots (x_j + 1)^{i_j} \dots x_k^{i_k}$ can be expressed as a linear combination of monomials $x_1^0 \dots x_k^0, \dots, x_1^{d_1} \dots x_k^{d_k}$. For example, if $k = 2$, then $Inc_{2,(1,2)}$ can be defined by $Inc_{2,(1,2)}\ \tilde{x}\ y\ z = AddPr\ x_{(1,2)}\ (AddPr\ x_{(1,1)}\ (AddPr\ x_{(1,1)}\ x_{(1,0)}))\ y\ z$, 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 Appendix B for the whole 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 $\{\mathcal{G} \mid \mathcal{P}(\mathcal{G}) = 1, \mathcal{G} \text{ is an order-2 PHORS}\}$ is not recursively enumerable.*

Proof. By the construction of $\mathcal{G}_2^{P,Q}$ above, $\mathcal{P}(\mathcal{G}_2^{P,Q}) = 1$ if and only if $P(x_1, \dots, x_k) \geq Q(x_1, \dots, x_k)$ holds for all $x_1, \dots, x_k \in \mathbf{Nat}$. By Lemma 3.1, the pair (P, Q) that satisfies the latter is not recursively enumerable, hence neither is the set $\{\mathcal{G} \mid \mathcal{P}(\mathcal{G}) = 1, \mathcal{G} \text{ is an order-2 PHORS}\}$. \square

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, neither the set $\{\mathcal{G} \mid \Pr(\mathcal{G}) \geq r, \mathcal{G} \text{ is an order-2 PHORS}\}$ nor the set $\{\mathcal{G} \mid \Pr(\mathcal{G}) = r, \mathcal{G} \text{ is an order-2 PHORS}\}$ are recursively enumerable.

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. \square

Recall that, in contrast, $\{\mathcal{G} \mid \Pr(\mathcal{G}) > r\}$ is r.e. (recursively enumerable) by Theorem 2.1.

Remark 3.1 *Let us write $\Psi_{?r}$ for the set of order-2 PHORSes \mathcal{G} such that $\Pr(\mathcal{G}) ? r$ where $? \in \{<, \leq, =, \geq, >\}$. By Theorems 2.1 and 3.2, we have:*

- (i) *For any rational number $r \in [0, 1]$, $\Psi_{>r}$ is r.e.*
- (ii) *For any rational number $r \in (0, 1]$, $\Psi_{\geq r}$ is not r.e. (whereas $\Psi_{\geq 0}$ is obviously recursive).*
- (iii) *For any rational number $r \in (0, 1]$, $\Psi_{=r}$ is not r.e. (whereas $\Psi_{=0}$ is recursive; recall Remark 2.1).*

It is open whether the followings propositions hold or not.

- (iv) $\Psi_{<r}$ is r.e. for every rational number r .
- (v) $\Psi_{\leq r}$ is r.e. 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 $|\Pr(\mathcal{G}) - r| < \epsilon$.*

Propositions (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 m (overlapping) intervals

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

By using procedures for (i) and (iv), one can enumerate all the order-2 PHORSs 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 $\epsilon = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots$, and output \mathcal{G} if the output r' for (\mathcal{G}, ϵ) 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 $\epsilon = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots$ \square

Remark 3.2 Theorem 3.2 also implies that, in contrast to the decidability of LTL model checking of recursive Markov chains [7, 24], the corresponding problems for order-2 PHORS are undecidable and there are even no precise approximation algorithms. Let us extend terms with events:

$$t ::= \dots \mid \mathbf{eventa}; t$$

where $\mathbf{eventa}; t$ 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 3.1. To see this, notice that by parametric $\mathcal{G}_2^{P,Q}$ with \mathbf{halt} , we can define a nonterminal $F : \circ \rightarrow \circ$ such that $F x$ almost surely reduces to x if and only if there exist no n_1, \dots, n_k such that $P(n_1, \dots, n_k) < Q(n_1, \dots, n_k)$. Consider the (extended) PHORS $\mathcal{G}^{P,Q,a^\omega}$ whose start symbol S is defined by:

$$S = \mathbf{eventa}; F(S).$$

Then

$$\mathcal{P}_{a^\omega}(\mathcal{G}_1) = \begin{cases} 0 & \text{if there exists } n_1, \dots, n_k \text{ such that } P(n_1, \dots, n_k) < Q(n_1, \dots, n_k) \\ 1 & \text{otherwise.} \end{cases}$$

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

4 Fixpoint Characterization of Termination Probability

The aim of this section is to show 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 basis 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 to characterize 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 [23], which served as a key step in the analysis of recursive Markov chains (or probabilistic pushdown systems) [23, 25, 7, 6].

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.

$$\begin{aligned} \mathcal{E} \text{ (equations)} &::= \{f_1(\tilde{x}_{1,1}) \cdots (\tilde{x}_{1,\ell_1}) = e_1, \dots, f_m(\tilde{x}_{m,1}) \cdots (\tilde{x}_{m,\ell_m}) = e_m\} \\ e \text{ (expressions)} &::= r \mid x \mid f \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e_1 e_2 \mid (e_1, \dots, e_k) \end{aligned}$$

Here, r ranges over the set of real numbers in $[0, 1]$, and (\tilde{x}) represents a tuple of variables (x_1, \dots, x_k) . In the set \mathcal{E} of equations, we require that each function symbol f_i 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 2. The *order of a system of fixpoint equations* \mathcal{E} is the largest order of the types of functions in \mathcal{E} .

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

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

It is well-typed under $f_1 : \mathbf{R}, f_2 : (\mathbf{R} \rightarrow \mathbf{R}) \rightarrow (\mathbf{R} \times \mathbf{R}) \rightarrow \mathbf{R}, f_3 : \mathbf{R} \rightarrow \mathbf{R}$. \square

τ (types) ::= $\mathbf{R} \mid \tau_1 \rightarrow \tau_2 \mid \tau_1 \times \dots \times \tau_n$.

$$\begin{array}{c}
\frac{r \in \mathbf{R}}{\Gamma \vdash r : \mathbf{R}} \qquad \frac{\Gamma \vdash e_1 : \mathbf{R} \quad \Gamma \vdash e_2 : \mathbf{R}}{\Gamma \vdash e_1 + e_2 : \mathbf{R}} \qquad \frac{\Gamma \vdash e_1 : \mathbf{R} \quad \Gamma \vdash e_2 : \mathbf{R}}{\Gamma \vdash e_1 \cdot e_2 : \mathbf{R}} \\[10pt]
\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau} \qquad \frac{\Gamma \vdash e_i : \tau_i \text{ for each } i \in \{1, \dots, k\}}{\Gamma \vdash (e_1, \dots, e_k) : \tau_1 \times \dots \times \tau_k} \\[10pt]
\frac{\Gamma, (\tilde{x}_{i,1}) : \tau_{i,1}, \dots, (\tilde{x}_{i,\ell_i}) : \tau_{i,\ell_i} \vdash e_i \quad \Gamma(f_i) = \tau_{i,1} \rightarrow \dots \rightarrow \tau_{i,\ell_i} \rightarrow \mathbf{R} \text{ (for each } i \in \{1, \dots, m\})}{\Gamma \vdash \{f_i(\tilde{x}_{i,1}) \dots (\tilde{x}_{i,\ell_i}) = e_i \mid i \in \{1, \dots, m\}\}}
\end{array}$$

Figure 2: Type system for fixpoint equations. $(x_1, \dots, x_k) : \tau$ denotes $x_1 : \tau_1, \dots, x_k : \tau_k$ where $\tau = \tau_1 \times \dots \times \tau_k$.

The semantics of fixpoint equations is defined in an obvious manner. Let \mathbf{R}_∞ 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 $\llbracket \tau \rrbracket = (X_\tau, \sqsubseteq_\tau, \perp_\tau)$, defined by induction on τ :

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

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

For a type environment Γ , we write $\llbracket \Gamma \rrbracket$ for the set of functions that map each $x \in \text{dom}(\Gamma)$ to an element of $\llbracket \Gamma(x) \rrbracket$. Given $\rho \in \llbracket \Gamma \rrbracket$ and e such that $\Gamma \vdash e : \tau$, its semantics $\llbracket e \rrbracket_\rho \in \llbracket \tau \rrbracket$ is defined by:

$$\begin{aligned}
\llbracket r \rrbracket_\rho &= r & \llbracket x \rrbracket_\rho &= \rho(x) & \llbracket f \rrbracket_\rho &= \rho(f) & \llbracket e_1 + e_2 \rrbracket_\rho &= \llbracket e_1 \rrbracket_\rho + \llbracket e_2 \rrbracket_\rho & \llbracket e_1 \cdot e_2 \rrbracket_\rho &= \llbracket e_1 \rrbracket_\rho \cdot \llbracket e_2 \rrbracket_\rho \\
\llbracket e_1 e_2 \rrbracket_\rho &= (\llbracket e_1 \rrbracket_\rho)(\llbracket e_2 \rrbracket_\rho) & \llbracket (e_1, \dots, e_k) \rrbracket_\rho &= (\llbracket e_1 \rrbracket_\rho, \dots, \llbracket e_k \rrbracket_\rho).
\end{aligned}$$

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 $\rho \in \llbracket \Gamma \rrbracket$ such that $\llbracket f(\tilde{x}_1) \dots (\tilde{x}_\ell) \rrbracket_{\rho\{\tilde{x}_1 \mapsto \tilde{y}_1, \dots, \tilde{x}_\ell \mapsto \tilde{y}_\ell\}} = \llbracket e \rrbracket_{\rho\{\tilde{x}_1 \mapsto \tilde{y}_1, \dots, \tilde{x}_\ell \mapsto \tilde{y}_\ell\}}$ for every equation $f(\tilde{x}_1) \dots (\tilde{x}_\ell) = e \in \mathcal{E}$ and $(\tilde{y}_1) \in \llbracket \tau_1 \rrbracket, \dots, (\tilde{y}_\ell) \in \llbracket \tau_\ell \rrbracket$ with $\Gamma(f) = \tau_1 \rightarrow \dots \rightarrow \tau_\ell \rightarrow \mathbf{R}$. Note that $\rho_{\mathcal{E}}$ always exists, and given by: $\rho_{\mathcal{E}} = \text{lfp}(\mathcal{F}_{\mathcal{E}}) = \bigsqcup_{i \in \omega} \mathcal{F}_{\mathcal{E}}^i(\perp_{\llbracket \Gamma \rrbracket})$, where $\mathcal{F}_{\mathcal{E}} \in \llbracket \Gamma \rrbracket \rightarrow \llbracket \Gamma \rrbracket$ is defined as the map such that

$$\mathcal{F}_{\mathcal{E}}(\rho)(f) = \lambda(\tilde{y}_1) \in \llbracket \tau_1 \rrbracket. \dots \lambda(\tilde{y}_\ell) \in \llbracket \tau_\ell \rrbracket. \llbracket e \rrbracket_{\rho\{\tilde{x}_1 \mapsto \tilde{y}_1, \dots, \tilde{x}_\ell \mapsto \tilde{y}_\ell\}}$$

for each $f(\tilde{x}_1) \dots (\tilde{x}_\ell) = e \in \mathcal{E}$ with $\Gamma(f) = \tau_1 \rightarrow \dots \rightarrow \tau_\ell \rightarrow \mathbf{R}$. Note that $\mathcal{F}_{\mathcal{E}}$ is a continuous function on the ω -cpo $\llbracket \Gamma \rrbracket$.

4.2 Order- n Fixpoint Characterization

We now give a translation from an order- n PHORS \mathcal{G} to 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 **halt** 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:

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

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

$$\mathbf{o}^\# = \mathbf{R} \qquad (\kappa_1 \rightarrow \kappa_2)^\# = \kappa_1^\# \rightarrow \kappa_2^\# \qquad (x_1 : \kappa_1, \dots, x_n : \kappa_n)^\# = x_1 : \kappa_1^\#, \dots, x_n : \kappa_n^\#.$$

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

Lemma 4.1 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ be an order- n PHORS. Then $\mathcal{N}^\# \vdash \mathcal{E}_{\mathcal{G}}$ and $\mathcal{N}^\# \vdash S : \mathbf{R}$.*

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.

Theorem 4.2 *Let \mathcal{G} be an order- n PHORS. Then $\mathcal{P}(\mathcal{G}) = \rho_{\mathcal{E}_{\mathcal{G}}}(S)$.*

Proofs of Lemma 4.1 and Theorem 4.2 are found in Appendix C.1.

Example 4.2 Recall $\mathcal{G}_1 = (\mathcal{N}_1, \mathcal{R}_1, S)$ from Example 2.1, where:

$$\mathcal{N}_1 = \{S \mapsto \mathbf{o}, F \mapsto \mathbf{o} \rightarrow \mathbf{o}\} \quad \mathcal{R}_1 = \{S = F \text{halt} \oplus_1 \Omega, \quad Fx = x \oplus_p F(Fx)\}.$$

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

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

□

Example 4.3 Recall \mathcal{G}_3 in Example 2.3.

$$S = \text{Listgen}(\text{Listgen Boolgen}) \text{halt} \\ \text{Boolgen } k = k \oplus_{\frac{1}{2}} k \quad \text{Listgen } f \ k = k \oplus_{\frac{1}{2}} (f(\text{Listgen } f \ k)).$$

The corresponding fixpoint equations are:

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

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

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

The least solution is:

$$S = 1 \quad \text{Boolgen } k = \text{ListgenList } k = \text{ListgenBool } k = k.$$

□

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 a generalization of the result on the reachability problem for Recursive Markov Chains [23].

The basic observation (that is also behind the fixpoint characterization for Recursive Markov Chains [23]) is that the termination behavior of an order-1 function of type $\mathbf{o}^\ell \rightarrow \mathbf{o}$ can be represented by a tuple of probabilities $(p_0, p_1, \dots, p_\ell)$, where (i) p_0 is the probability that the function terminates *without* using 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 \mathbf{o} , where f is an order-1 function of type $\mathbf{o}^\ell \rightarrow \mathbf{o}$. 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 $i \in \{1, \dots, \ell\}$, and t_i terminates (notice, in this case, that none of the other t_j 's are called¹). Thus, the probability that $f \ t_1 \cdots t_\ell$ terminates can be calculated by $p_0 + p_1 q_1 + \cdots p_\ell q_\ell$, where

¹Since t_i is of type \mathbf{o} , once t_i is called, the control cannot go back to f .

q_i denotes the probability that t_i terminates. The termination probability is, therefore, independent of the precise internal behavior of f ; only $(p_0, p_1, \dots, p_\ell)$ 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 through Example 2.1.

Example 4.4 Recall \mathcal{G}_1 in Example 2.1, consisting of: $S = F \text{ halt}$ and $F x = x \oplus_p F(F x)$. Here, we have two functions: S of type \circ and F of type $\circ \rightarrow \circ$. 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 equations.

$$S_0 = F_0 + F_1 \cdot 1 \quad F_0 = p \cdot 0 + (1-p)(F_0 + F_1 \cdot F_0) \quad F_1 = p \cdot 1 + (1-p)(F_1 \cdot F_1 \cdot 1).$$

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:

$$S_0 = F_0 + F_1 \quad F_0 = (1-p)(F_0 + F_1 F_0) \quad F_1 = p + (1-p)F_1^2.$$

The least solution is: $F_0 = 0 \quad S_0 = F_1 = \begin{cases} \frac{p}{1-p} & \text{if } 0 \leq p < \frac{1}{2} \\ 1 & \text{if } \frac{1}{2} \leq p \leq 1. \end{cases}$ □

For the technical convenience in formalizing the translation, we assume below that the rules of PHORS do not contain **halt**; the start symbol S (which is now a non-terminal of type $\circ \rightarrow \circ$) takes **halt** from the environment. Thus, the termination probability we consider is $\mathcal{P}(\mathcal{G}, S \text{ halt})$, where **halt** does not occur in \mathcal{R} . This is without any loss of generality, since **halt** can be passed around as an argument without increasing the order of PHORS.

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:

$$S x = F (H x) x \quad F f y = f(f y) \quad H x y = x \oplus_{\frac{1}{2}} (y \oplus_{\frac{1}{2}} \Omega).$$

Suppose we wish to characterize the termination probability of $S \text{ halt}$, 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 so that it can work in the 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, the probability that the argument f reaches the target x . Therefore, p_1 is represented as $F_0(H_1, H_2)$, where

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

In $f_0 + f_1 \cdot f_0$, f_0 on the lefthand side represents the probability that the outer call of f in $f(fx)$ reaches the target (without using (fx)), and $f_1 \cdot f_0$ represents the probability that the outer call of f uses the argument fx , 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 F_i 's and H_i 's 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 can pass around a tuple of real numbers (f_0, f_1, \dots, 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 is reached (where “the current target” is chosen by a caller).

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 $G : ((\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ$, 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

$$G_1 \ h_1 = \dots,$$

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. 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

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

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 H_2 represents the probability that $H \ z$ reaches its argument (the second argument of H).

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

Given a rule $F \ z_1 \dots z_m = t_L \oplus_p t_R$ of PHORS, we uniquely decompose z_1, \dots, z_m to two (possibly empty) subsequences z_1, \dots, z_ℓ and $z_{\ell+1}, \dots, z_m$ so that the order of z_ℓ is greater than 0 (note, however, that the orders of $z_1, \dots, z_{\ell-1}$ may be 0), and $z_{\ell+1}, \dots, z_m$ are order-0 variables (in other words, $z_{\ell+1}, \dots, z_m$ is the maximal postfix of z_1, \dots, 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 $F \ \tilde{y} \ \tilde{x} = t_L \oplus_p t_R$ for a rule of PHORS, we implicitly assume that \tilde{x} is the maximal postfix of the sequence $\tilde{y} \ \tilde{x}$ consisting of only order-0 variables. Similarly, when we write $F \ \tilde{s} \ \tilde{t}$ for a fully-applied term (of order 0), we implicitly assume that \tilde{t} is the maximal postfix of the sequence of arguments, consisting of only order-0 terms.

Consider a function definition of the form:

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

where (following the notational convention above) the sequence x_1, \dots, x_k is the maximal postfix of $y_1, \dots, y_m, x_1, \dots, x_k$ 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:

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

where \mathcal{K} is the type environment for non-terminals and y_1, \dots, y_m , and κ is the type of t with $\text{ar}(\kappa) = \ell$. The output of the translation, $(e_0, e_1, \dots, e_\ell, e_{\ell+1}, \dots, e_{\ell+k}, e_{\ell+k+1})$, represents the following information.

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

$\overline{\mathcal{K}; x_1, \dots, x_k \vdash \Omega : \circ \rightsquigarrow (0^{k+2})}$	(TR-OMEGA)
$\overline{\mathcal{K}; x_1, \dots, x_k \vdash x_i : \circ \rightsquigarrow (0^i, 1, 0^{k-i+1})}$	(TR-GVAR)
$\overline{\mathcal{K}(y) = \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ}$	(TR-VAR)
$\overline{\mathcal{K}; x_1, \dots, x_k \vdash y : \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (y_0, y_1, \dots, y_\ell, (y_{\ell+1})^{k+1})}$	
$\mathcal{K}; x_1, \dots, x_k \vdash s : \kappa_1 \rightarrow \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \rightsquigarrow (s_0, \dots, s_{\ell+k+1})$	
$\mathcal{K}; x_1, \dots, x_k \vdash t : \kappa_1 \rightsquigarrow (t_0, \dots, t_{\ell'+k+1}) \quad \mathbf{ar}(\kappa_1) = \ell'$	(TR-APP)
$\mathcal{K}; \tilde{x} \vdash st : \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ$	
$\rightsquigarrow (s_0(t_0, \dots, t_{\ell'}, t_{\ell'+k+1}), s_1(t_1, \dots, t_{\ell'}, t_{\ell'+k+1}), \dots, s_\ell(t_1, \dots, t_{\ell'}, t_{\ell'+k+1}),$	
$s_{\ell+1}(t_{\ell'+1}, t_1, \dots, t_{\ell'}, t_{\ell'+k+1}) \dots, s_{\ell+k+1}(t_{\ell'+k+1}, t_1, \dots, t_{\ell'}, t_{\ell'+k+1}))$	
$\mathcal{K}; x_1, \dots, x_k \vdash s : \circ^{\ell+1} \rightarrow \circ \rightsquigarrow (s_0, \dots, s_{k+\ell+2})$	
$\mathcal{K}; x_1, \dots, x_k \vdash t : \circ \rightsquigarrow (t_0, \dots, t_{k+1})$	(TR-APPG)
$\overline{\mathcal{K}; x_1, \dots, x_k \vdash st : \circ^\ell \rightarrow \circ \rightsquigarrow (s_0 + s_1 \cdot t_0, s_2, \dots, s_{\ell+1}, s_{\ell+2} + s_1 \cdot t_1, \dots, s_{\ell+k+2} + s_1 \cdot t_{k+1})}$	
$\mathcal{N}, y_1 : \kappa_1, \dots, y_\ell : \kappa_\ell; x_1, \dots, x_k \vdash t_d : \circ \rightsquigarrow (t_{d,0}, \dots, t_{d,k+1}) \text{ for each } d \in \{L, R\}$	
$\tilde{y}_i = (y_{i,0}, \dots, y_{i,\mathbf{ar}(\kappa_i)}) \quad \tilde{y}_i' = (y_{i,1}, \dots, y_{i,\mathbf{ar}(\kappa_i)})$	(TR-RULE)
$\mathcal{N} \vdash (F \tilde{y} x_1 \dots x_k = t_L \oplus_p t_R) \rightsquigarrow \{F_i \tilde{y}_1' \dots \tilde{y}_\ell' = pt_{L,i} + (1-p)t_{R,i} \mid i \in \{1, \dots, k\}\}$	
$\cup \{F_0 \tilde{y}_1 \dots \tilde{y}_\ell = pt_{L,0} + (1-p)t_{R,0}, \quad F_{k+1} \tilde{y}_1 \dots \tilde{y}_\ell = pt_{L,0} + (1-p)t_{R,0}\}$	
$\mathcal{E} = \bigcup \{\mathcal{E}_i \mid \mathcal{N} \vdash (F \tilde{y} x_1 \dots x_k = t_L \oplus_p t_R) \rightsquigarrow \mathcal{E}_i, (F \tilde{y} x_1 \dots x_k = t_L \oplus_p t_R) \in \mathcal{R}\}$	(TR-GRAM)
$(\mathcal{N}, \mathcal{R}, S) \rightsquigarrow (\mathcal{E}, S_1)$	

Figure 3: Translation rules for the order- $(n-1)$ fixpoint characterization

- 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).
- $e_i (i \in \{1, \dots, \ell\})$: the reachability probability to t 's i -th argument.
- $e_{\ell+i} (i \in \{1, \dots, k\})$: 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 $\tilde{\kappa} \Rightarrow \circ^m \rightarrow \circ$ is replaced by $(y_0, y_1, \dots, y_m, y_{m+1})$, which represents information analogous to $(e_0, e_1, \dots, e_\ell, e_{\ell+k+1})$: y_0 represents (a function for computing) the reachability probability to the current target, $y_i (i \in \{1, \dots, m\})$ represents the reachability probability to the i -th order-0 variable (among the last m variables), and y_{m+1} represents the reachability probability to a fresh target (to be set later). In contrast, the variables x_1, \dots, x_k will be removed by the translation.

The translation rules are given in Figure 3. In the rules, to clarify the correspondence between source terms and target expressions, we use metavariables s, t, \dots (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 **halt**; this is due to the assumption that **halt** 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, \dots, 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 those components. 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 $s_1, \dots, s_{\ell+k}$ 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 $s_{\ell+1}, \dots, s_{\ell+k}$, the reachability targets are x_1, \dots, x_k ; thus, information about how t reaches those variables is passed as the first argument of $s_{\ell+1}, \dots, s_{\ell+k}$. 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, \dots, F_{k+1} . As the body of F_i ($i \in \{1, \dots, k\}$), $t_{d,i}$ is chosen as the body of F_i , as it should represent the reachability probability to x_i . The body of F_{k+1} is the same as that of F_0 ; no target is actually set yet in F_0 . (Thus, we can actually eliminate the definition of F_{k+1} by replacing all the occurrences of F_{k+1} with F_0 ; F_{k+1} has been introduced just for technical convenience). 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 $\mathcal{N}(S) = \circ \rightarrow \circ$), we write $\mathcal{E}_{\mathcal{G}}^{\text{ref}}$ for \mathcal{E} such that $(\mathcal{N}, \mathcal{R}, S) \rightsquigarrow (\mathcal{E}, S_1)$. Such \mathcal{E} is actually unique (up to α -equivalence), given \mathcal{N}, \mathcal{R} . Note also that by the definition of the translation relation, the output of the translation always exists.

Example 4.5 Recall the order-2 PHORS \mathcal{G}_2 in Example 2.2:

$$S = F H \quad H x = x \oplus_{\frac{1}{2}} \Omega \quad F g = (g \text{halt}) \oplus_{\frac{1}{2}} (F(D g)) \quad D g x = g(g x).$$

It can be modified to the following rules so that **halt** does not occur.

$$S z = F H z \quad H x = x \oplus_{\frac{1}{2}} \Omega \quad F g z = (g z) \oplus_{\frac{1}{2}} (F(D g) z) \quad D g x = g(g x).$$

Here, **halt** can be passed around through the variable z . We have:

$$\mathcal{N}_2; z \vdash F H : \circ \rightsquigarrow (F_0(H_0, H_1, H_2), F_1(H_1, H_2), F_2(H_2, H_1, H_2), F_2(H_2, H_1, H_2)).$$

Using it, $F H z$ is translated to:

$$(F_0(H_0, H_1, H_2) + F_1(H_1, H_2) \cdot 0, F_2(H_2, H_1, H_2) + F_1(H_1, H_2) \cdot 1, F_2(H_2, H_1, H_2) + F_1(H_1, H_2) \cdot 0).$$

Thus, we have the following equations for S_0, S_1 , and S_2 .

$$S_0 = S_2 = F_0(H_0, H_1, H_2) \quad S_1 = F_2(H_2, H_1, H_2) + F_1(H_1, H_2).$$

See Appendix C.3 for details. The following equations are obtained for the other non-terminals.

$$\begin{aligned} H_0 = H_2 = 0 \quad H_1 = \frac{1}{2} \\ F_0(g_0, g_1, g_2) = F_2(g_0, g_1, g_2) = \frac{1}{2}g_0 + F_0(D_0(g_0, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)) \\ F_1(g_1, g_2) = \frac{1}{2}(g_1 + g_2) + \frac{1}{2}(F_2(D_2(g_2, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)) + F_1(D_1(g_1, g_2), D_2(g_2, g_1, g_2))) \\ D_0(g_0, g_1, g_2) = D_2(g_0, g_1, g_2) = g_0 + g_1g_0 \quad D_1(g_1, g_2) = g_2 + g_1(g_1 + g_2). \end{aligned}$$

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

$$\begin{aligned} S_0 = S_2 = F_0(\frac{1}{2}) \quad S_1 = F_0(\frac{1}{2}) + F_1(\frac{1}{2}) \quad F_0(g_1) = F_2(g_1) = F_0(D_1(g_1)) \\ F_1(g_1) = \frac{1}{2}g_1 + \frac{1}{2}(F_0(D_1(g_1)) + F_1(D_1(g_1))) \quad D_0(g_1) = D_2(g_1) = 0 \quad D_1(g_1) = g_1^2. \end{aligned}$$

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

$$S_1 = F_1(\frac{1}{2}) \quad F_1(g_1) = \frac{1}{2}g_1 + \frac{1}{2}F(g_1^2).$$

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

Example 4.6 Consider the following order-3 PHORS:

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

$$F(x) = \frac{1}{2}x + \frac{1}{2}F(x^2)$$

where

$$S : \circ \rightarrow \circ, F : ((\circ \rightarrow \circ) \rightarrow \circ) \rightarrow \circ, C : \circ \rightarrow (\circ \rightarrow \circ) \rightarrow \circ, H : \circ \rightarrow \circ.$$

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:

$$\begin{aligned} S_0 &= S_2 = F_0(C_0(0, 0), C_1(0, 0)) & S_1 &= F_1(C_1(1, 0), C_1(0, 0)) \\ F_0(g_0, g_1) &= F_1(g_0, g_1) = g_0(H_0, H_1, H_2) \\ C_0(x_0, x_1)(f_0, f_1, f_2) &= C_1(x_0, x_1)(f_0, f_1, f_2) = f_0 + f_1 \cdot x_0 \\ H_0 &= H_2 = 0 & H_1 &= \frac{1}{2}, \end{aligned}$$

where

$$\begin{aligned} S_0 &: \mathbf{R}, S_1 : \mathbf{R}, S_2 : \mathbf{R}, \\ F_0 &: (\mathbf{R} \times \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}) \times (\mathbf{R} \times \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}) \rightarrow \mathbf{R}, F_1 : (\mathbf{R} \times \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}) \times (\mathbf{R} \times \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}) \rightarrow \mathbf{R}, \\ C_0 &: (\mathbf{R} \times \mathbf{R}) \rightarrow (\mathbf{R} \times \mathbf{R} \times \mathbf{R}) \rightarrow \mathbf{R}, C_1 : (\mathbf{R} \times \mathbf{R}) \rightarrow (\mathbf{R} \times \mathbf{R} \times \mathbf{R}) \rightarrow \mathbf{R}, \\ H_0 &: \mathbf{R}, H_1 : \mathbf{R}, H_2 : \mathbf{R}. \end{aligned}$$

The order of the equations is 2 (where the largest order is that of the types of F_0 and F_1).

We have:

$$S_1 = F_1(C_1(1, 0), C_1(0, 0)) = C_1(1, 0)(H_0, H_1, H_2) = H_0 + H_1 \cdot 1 = \frac{1}{2}.$$

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

To state the well-formedness of the output of the translation, we define the translation of types and type environments as follows.

$$\begin{aligned} (\kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \circ^\ell \rightarrow \circ)^\dagger &= (\kappa_1^\dagger \rightarrow \dots \rightarrow \kappa_k^\dagger \rightarrow \mathbf{R}) \times (\kappa_1^{\dagger'} \rightarrow \dots \rightarrow \kappa_k^{\dagger'} \rightarrow \mathbf{R})^\ell \times (\kappa_1^\dagger \rightarrow \dots \rightarrow \kappa_k^\dagger \rightarrow \mathbf{R}) \\ (\kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \circ^\ell \rightarrow \circ)^{\dagger'} &= (\kappa_1^{\dagger'} \rightarrow \dots \rightarrow \kappa_k^{\dagger'} \rightarrow \mathbf{R})^\ell \times (\kappa_1^\dagger \rightarrow \dots \rightarrow \kappa_k^\dagger \rightarrow \mathbf{R}). \end{aligned}$$

We also write $(\kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \circ^\ell \rightarrow \circ)^{\dagger+m}$ for

$$(\kappa_1^\dagger \rightarrow \dots \rightarrow \kappa_k^\dagger \rightarrow \mathbf{R}) \times (\kappa_1^{\dagger'} \rightarrow \dots \rightarrow \kappa_k^{\dagger'} \rightarrow \mathbf{R})^\ell \times (\kappa_1^\dagger \rightarrow \dots \rightarrow \kappa_k^\dagger \rightarrow \mathbf{R})^{m+1}.$$

It represents the type of the tuple $(e_0, \dots, e_{\ell+m+1})$ obtained by translating a term of type $\kappa_1 \rightarrow \dots \rightarrow \kappa_k \Rightarrow \circ^\ell \rightarrow \circ$ with order-0 variables x_1, \dots, x_m . The distinction between κ_i^\dagger and $\kappa_i^{\dagger'}$ reflects the fact that in the output $(e_0, e_1, \dots, e_\ell, e_{\ell+1}, \dots, e_{\ell+m+1})$ of the translation, e_1, \dots, e_ℓ take one less argument (recall TR-APP). The translation of type environments is defined by:

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

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

Lemma 4.3 (well-typedness of the output of transformation) *Let $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ be a PHORS. If $\mathcal{N} \vdash \mathcal{R} \rightsquigarrow (\mathcal{E}, S_1)$, then $\mathcal{N}^\dagger \vdash \mathcal{E}$ and $\mathcal{N}^\dagger(S_1) = \mathbf{R}$.*

As a corollary, it follows that for any order- n PHORS \mathcal{G} (where $n > 0$), the order of $\mathcal{E}_{\mathcal{G}}^{\text{ref}}$ is $n - 1$. The following is the main theorem of this section, which states the correctness of the translation.

Theorem 4.4 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ be an order- n PHORS, Then, $\mathcal{P}(\mathcal{G}, S \text{halt}) = \rho_{\mathcal{E}_{\mathcal{G}}^{\text{ref}}}(S_1)$.*

A proof of the theorem is found in Appendix C.2. Here we sketch the proof. We first prove the theorem for *recursion-free* PHORSs (so that any term is strongly normalizing; see Appendix C.1 for the precise definition), and extend it to general PHORSs \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 PHORSs, 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 C.2 for details.

Example 4.7 Recall PHORS \mathcal{G}_5 in Example 2.3.

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

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

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

Here, Treegen_1 is the function that returns the probability that $\text{Treegen}\ p\ \text{Boolgen}\ x$ reaches x , where the parameters p_1 and p_2 represent the probability 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 $\text{Treegen}_1^{(m)}$ of $\rho(\text{Treegen}_1)$ by

$$\text{Treegen}_1^{(0)}(p_1, p_2) = 0 \quad \text{Treegen}_1^{(m+1)}(p_1, p_2) = p_1 + p_2 \cdot (\text{Treegen}_1^{(m)}(G_1(p_1, p_2), G_2(p_1, p_2)))^3.$$

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

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

For the induction step, we have

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

as required. Thus,

$$\text{Treegen}_1^{(m)}(\frac{1}{2}, \frac{1}{2}) = \frac{1}{2} + \frac{1}{2} \text{Treegen}_1^{(m-1)}(1 - \frac{1}{2^2}, \frac{1}{2^2}) \geq \frac{1}{2} + \frac{1}{2}(1 - \frac{1}{2^{2+(m-1)-1}})$$

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

5 Computing Upper-Bounds of Termination Probability

Theorems 4.2 and 4.4 immediately provide procedures for computing *lower*-bounds of the termination probability as precisely as we need (cf. Theorem 2.1). The termination probability, in other words, is a *recursively enumerable* real number (see, e.g. [10]), 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 can be easily computed, but it does not provide a good upper-bound for the *least* solution, 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 PHORSs, 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 are of the form:

$$f_1(x_1, \dots, x_{\ell_1}) = e_1, \quad \dots \quad f_k(x_1, \dots, x_{\ell_k}) = e_k, \quad (1)$$

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, \dots, x_{ℓ_i} of each function f_i can be partitioned to several groups of variables $(x_1, \dots, x_{d_{i,1}}), (x_{d_{i,1}+1}, \dots, x_{d_{i,2}}), \dots, (x_{d_{i,j-1}+1}, \dots, x_{\ell_i})$, 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 $(x_{d_{i,j-1}+1}, \dots, x_{d_{i,h}})$ 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. Suppose that (x_1, \dots, x_{ℓ_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 in 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 \perp by Kleene iterations, are *pointwise convex*, i.e., convex on each variable, i.e., $f(x_1, \dots, (1-p)x + py, \dots, x_n) \leq (1-p)f(x_1, \dots, x, \dots, x_n) + pf(x_1, \dots, y, \dots, x_n)$ 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, $\frac{1}{4} = f(\frac{1}{2}, \frac{1}{2}) = f(\frac{1}{2}(0, 1) + \frac{1}{2}(1, 0)) > \frac{1}{2}f(0, 1) + \frac{1}{2}f(1, 0) = 0$. Recall that $\mathcal{F}_{\mathcal{E}_G^{\text{ref}}}$ is the functional associated with the fixpoint equations $\mathcal{E}_G^{\text{ref}}$; we simply write \mathcal{F} for $\mathcal{F}_{\mathcal{E}_G^{\text{ref}}}$ below.

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

Proof. The pointwise convexity of and monotonicity of $\mathcal{F}^m(\perp)$ 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 $\text{lfp}(\mathcal{F})$ follows from the fact that, for every m , when \vec{x} and \vec{y} differ by at most one coordinate,

$$(1-p)\text{lfp}(\mathcal{F})(\vec{x}) + p\text{lfp}(\mathcal{F})(\vec{y}) \geq (1-p)\mathcal{F}^m(\perp)(\vec{x}) + p\mathcal{F}^m(\perp)(\vec{y}) \geq \mathcal{F}^m(\perp)((1-p)\vec{x} + p\vec{y})$$

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

5.2 Computing an Upper-Bound by Discretization

Given fixpoint equations as in (1), we can compute an upper-bound of the least solution of the equations, by overapproximating the values of f_1, \dots, 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.

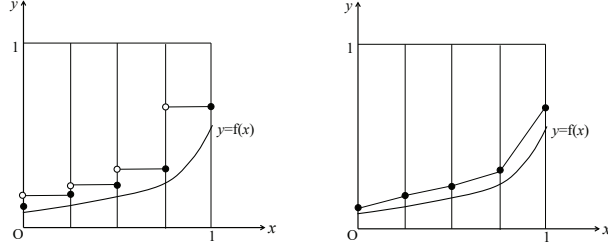


Figure 4: Overapproximation by a step-function (left) and a stepwise linear function (right)

5.2.1 Computing an Upper-Bound for a Unary Function

Suppose we are given a PHORS \mathcal{G} and that $\mathcal{E}_{\mathcal{G}}^{\text{ref}}$ 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, x , additions, multiplications, and applications of f . We abstract f to a sequence of real numbers $(r_0, \dots, r_n) \in [0, 1]^{n+1}$, where r_i represents the value of $f(\frac{i}{n})$. Thus, the abstraction function α mapping any function $f : [0, 1] \rightarrow [0, 1]$ to its abstract form $[0, 1]^{n+1}$ is defined by

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

We write γ for any concretization function γ , mapping any element of $[0, 1]^{n+1}$ back to a function in $[0, 1] \rightarrow [0, 1]$. The idea here is that *if γ satisfies certain axioms*, to be discussed later, 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 $\vec{r} = (r_0, \dots, r_n)$:

$$\vec{r} \geq \alpha(\mathcal{F}(\gamma(\vec{r}))). \quad (2)$$

Let $\hat{\mathcal{F}}$ be the functional $\lambda \vec{s}. \alpha(\mathcal{F}(\gamma(\vec{s})))$. Notice that solutions to (2) are precisely the pre-fixpoints of $\hat{\mathcal{F}}$, 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 4):

- (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 $\hat{f}(x) = r_i + \frac{x - \frac{i-1}{n}}{\frac{1}{n}}(r_{i+1} - r_i) = (i+1-nx)r_i + (nx-i)r_{i+1}$ if $x \in [\frac{i}{n}, \frac{i+1}{n}]$.

The first choice turns (α, γ) into a Galois insertion. The second choice is itself valid despite *not* being a Galois insertion: 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 arithmetics* (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, $0, \frac{1}{m}, \dots, \frac{m-1}{m}, 1$. The inequality (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 (2) is indeed a sufficient condition for $\gamma(\vec{r})$ to be an upper-bound on $\text{lfp}(\mathcal{F})$. Note that both step functions and stepwise linear functions satisfy the assumption of the lemma below.

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 $\text{lfp}(\mathcal{F})$.*

```

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] := αh(eval(e, {f ↦ r, x ↦  $\frac{i}{n}$ }));
  }
  return apply(r, c); }
apply(r, c) { (* apply the function represented by array r to c *)
  i := ⌊nc⌋; (*  $\frac{i}{n} \leq c < \frac{i+1}{n}$  *)
  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 5: Pseudo code for computing an upper-bound of $f(c)$ where $f(x) = e$ (unary function case)

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

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

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

Below we consider the combination of (b) and (d). Figure 5 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, $\alpha_h(x) = \frac{\lfloor mx \rfloor}{m}$. 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 $f \in [0, 1] \rightarrow [0, 1]$), which are satisfied by the fixpoint equations obtained from a PHORS (recall Section 5.1).

Example 5.1 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:

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

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 $0.3398 \dots$. \square

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 range over $0 \leq x_1, x_2$, 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 6 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., $(\frac{i_1}{n}, \frac{i_2}{n})$, $(\frac{i_1}{n}, \frac{i_2+1}{n})$, $(\frac{i_1+1}{n}, \frac{i_2}{n})$, $(\frac{i_1+1}{n}, \frac{i_2+1}{n})$

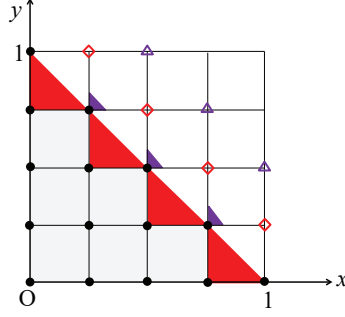


Figure 6: Discretization in the case of a binary function $f(x, y)$, whose domain is $\{(x, y) \mid 0 \leq x, 0 \leq y, x + y \leq 1\}$ (i.e., the grey and red areas). Outside the domain, the value of $f(x, y)$ may not belong to $[0, 1]$, or may be even undefined. The value at a point in the grey area 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 .

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.

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

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 discretized the codomain of f to $\{0, \frac{1}{m}, \dots, \frac{mh-1}{m}, h, \infty\}$ (instead of $\{0, \frac{1}{m}, \dots, \frac{m-1}{m}, 1\}$) 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 $\hat{f}(x_1, x_2) = 1$.

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 5, and essentially the same soundness argument as Lemma 5.2 applies.

Example 5.2 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 $\frac{1}{3}$;

```

main( $e_1, \dots, e_k, \vec{c}$ ) {
   $\rho := [f_1 \mapsto [\vec{0}], \dots, f_k \mapsto [\vec{0}]]$ ; (*  $\rho(f_i)$  is an array indexed by each element of  $D_i$  *)
   $\rho' := [f_1 \mapsto [\vec{1}], \dots, f_k \mapsto [\vec{1}]]$ ; (* dummy *)
  while not( $\rho = \rho'$ ) do {
     $\rho' := \rho$ ; (* copy the contents *)
    for each  $i \in \{0, \dots, k\}$  do
      for each  $\vec{v} \in D_i$  do  $\rho'(f_i)[\vec{v}] := \alpha_h(\text{eval}(e_i, \rho\{\vec{x} \mapsto \vec{v}\}, \vec{v} \stackrel{?}{\in} D'_i))$ ;
  }
  return apply( $\rho(f_1), \vec{c}$ ); }

eval( $e, \rho, b$ ) {
  (*  $b$  represents whether we are computing for 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
       $x \rightarrow \rho(x) \mid c \rightarrow c$ 
       $\mid f_i(\vec{e'}) \rightarrow \text{let } \vec{v} = \text{eval}(\vec{e'}, \rho, b) \text{ in if } \vec{v} \notin \overline{D_i} \text{ then } \infty \text{ else } \text{apply}(\rho(f_i), \vec{v})$ 
       $\mid e_1 + e_2 \rightarrow \text{eval}(e_1, \rho, b) + \text{eval}(e_2, \rho, b)$ 
       $\mid e_1 \cdot e_2 \rightarrow \text{eval}(e_1, \rho, b) \cdot \text{eval}(e_2, \rho, b)$ 
  in if  $b$  then return min( $r, 1$ ) else return  $r$  }

```

Figure 7: Pseudo code for computing an upper-bound for the general case

so the upper-bound for $f(0.3, 0.3)$ is sound but imprecise. By choosing $n = 16$ and $m = 256$, we obtain $0.3359 \dots$ as an upper-bound of $f(0.3, 0.3)$.

This is an example where the values of f at red points in Figure 6 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 $x_1x_2 > 0.25$; in particular $f(0.5, 1)$ and $f(1, 0.5)$ (the values of red points in Figure 6 for the case $n = 2$) are undefined. \square

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 $f_1(c_1, \dots, c_{\ell_1})$ for the least solution of the fixpoint equations:

$$f_1(x_1, \dots, x_{\ell_1}) = e_1 \quad \dots \quad f_k(x_1, \dots, x_{\ell_k}) = e_k.$$

Here, the formal arguments x_1, \dots, x_{ℓ_i} of each function f_i are partitioned to several groups $(x_1, \dots, x_{d_{i,1}}), (x_{d_{i,1}+1}, \dots, x_{d_{i,2}}), \dots, (x_{d_{i,g_i-1}+1}, \dots, x_{\ell_i})$, 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 $\{0, \frac{1}{n}, \dots, \frac{n-1}{n}, 1\}$, where the variables in each group $(x_{d_{i,j-1}+1}, \dots, x_{d_{i,j}})$ are constrained by $x_{d_{i,j-1}+1} + \dots + x_{d_{i,j}} \leq \frac{n+2}{n}$. Note that we choose $\frac{n+2}{n}$ instead of 1 as the upper-bound of the sum, to include the points \diamond and \triangle in Figure 6. 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 $x_{d_{i,j-1}+1} + \dots + x_{d_{i,j}} \leq 1$; note that $f_i(x_1, \dots, x_{\ell_i}) \in [0, 1]$ for $(x_1, \dots, x_{\ell_i}) \in D'_i$, but $f_i(x_1, \dots, x_{\ell_i})$ may be greater than 1 or undefined for $(x_1, \dots, x_{\ell_i}) \in D_i \setminus D'_i$. We also write $\overline{D_i}$ for the set $\{(x_1, \dots, x_{\ell_i}) \mid (\lceil nx_1 \rceil / n, \dots, \lceil nx_{\ell_i} \rceil / n) \in D_i\}$ (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 $f_1(c_1, \dots, c_{\ell_1})$ is given in Figure 7.

The function α_h takes a real value (or ∞) x , and returns the least element in $\{0, \frac{1}{n}, \dots, \frac{mh-1}{m}, h, \infty\}$ that is no less than x . The function **apply** in the figure takes the current approximations of values of f_i

at the points in 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:

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

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

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 [14] 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 $(C, \sqsubseteq_C, \perp_C)$ and $(A, \sqsubseteq_A, \perp_A)$ be ω -cpo. Suppose that $\alpha : (C, \sqsubseteq_C) \rightarrow (A, \sqsubseteq_A)$ and $\gamma : (A, \sqsubseteq_A) \rightarrow (C, \sqsubseteq_C)$ 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 $(C, \sqsubseteq_C, \perp_C)$ to $(C, \sqsubseteq_C, \perp_C)$. Let $\hat{\mathcal{F}} : (A, \sqsubseteq_A, \perp_A) \rightarrow (A, \sqsubseteq_A, \perp_A)$ be $\lambda x \in A. \alpha(\mathcal{F}(\gamma(x)))$, which is an “abstract version” of \mathcal{F} . Note that $\hat{\mathcal{F}}$ is also continuous. Then, we have:

Proposition 5.3 $\text{lfp}(\mathcal{F}) \sqsubseteq_C \gamma(\text{lfp}(\hat{\mathcal{F}}))$.

This result is standard (see, e.g., [14], Proposition 18) but we provide a proof in Appendix D for the convenience of the reader.

By the proposition above, to overapproximate the least fixpoint of \mathcal{F} , it suffices to find an appropriate abstract domain $(A, \sqsubseteq_A, \perp_A)$ and α, γ that satisfy the conditions above, so that the least fixpoint of $\hat{\mathcal{F}}$ is easily computable. In the case of overapproximating the termination probability of order- n PHORSs, we need to set up an abstract domain $(A, \sqsubseteq_A, \perp_A)$ 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:

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

Here, the concrete domain C^τ denotes $\llbracket \tau \rrbracket$ in Section 4.1. Then, α^τ and γ^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 $\text{lfp}(\hat{\mathcal{F}})$.

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 [42, 9, 57], 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 $\frac{1}{2} \cdot \frac{1}{n} + f(\frac{1}{2} \cdot \frac{1}{n})$, 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.

equations	#iter	#dom	#codom	l.b.	u.b.	u.b.(step)	exact	time
Ex2.3-1	12	16	512	0.333	0.336	1.0	$\frac{1}{3}$	0.001
Ex2.3-0	12	16	512	0.333	0.334	0.334	$\frac{1}{3}$	0.001
Ex2.3-v1	12	16	512	0.312	0.315	0.365	-	0.001
Ex2.3-v2	12	16	512	0.262	0.266	0.321	-	0.017
Ex2.3-v3	12	16	512	0.263	0.266	0.309	-	0.018
Ex2.4	12	16	512	0.320	0.323	0.329	-	0.004
Double	12	16	512	0.649	0.653	1.0	-	0.002
Listgen	15	16	512	0.999	1.0	1.0	1.0	0.002
Treegen	15	64	4096	0.618	0.619	1.0	$\frac{\sqrt{5}-1}{2}$	0.520
Treegenp	12	16	512	1.0	1.0	1.0	1.0	0.002
ListEven	12	32	1024	0.666	0.667	0.667	$\frac{2}{3}$	0.001
ListEven2	12	16	512	0.749	0.75	0.75	$\frac{3}{4}$	0.001
Determinize	25	16	512	0.996	1.0	1.0	1.0	20.7
Ex5.4(0,0)	12	16	512	0.0	0.0	0.0	0	0.005
Ex5.4(0.3,0.3)	12	16	512	0.333	0.336	0.35	$\frac{1}{3}$	0.005
Ex5.4(0.5,0.5)	10000	16	512	0.999	1.0	1.0	1	0.008
Discont(0,1)	12	16	512	0.0	0.0	0.0	0	0.004
Discont(0.01,0.99)	1000	16	512	0.999	1.0	1.0	1	0.004
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	3.06

Table 1: Experimental results (times are in seconds).

The step functions make use of only 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 PHORSs), exploiting an additional property like convexity would be important for obtaining a more precise method; it is left for future work.

6 Preliminary 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 result is summarized in Table 1. 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 $[0, 1]$ 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 “u.b.(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 “u.b.(step)”).

The equations “Ex2.3-1” and “Ex2.3-0” are order-1 and order-0 equations obtained from the PHORS in Example 2.1 (see also Examples 4.2 and 4.4) 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 2.2 (see also Example 4.5). The equations “Double” are those obtained from the following order-2 PHORS:

$$S = F H \quad H x y = x \oplus_{\frac{1}{2}} y \quad F g = g \text{halt}(F(D g)) \quad D g x y = g(g x y) y,$$

with manual simplifications. The equations “Listgen”, “Treegen”, and “Treegenp” are from Example 2.3, corresponding to \mathcal{G}_3 , \mathcal{G}_4 and \mathcal{G}_5 , respectively. The equations “ListEven” and “ListEven2” are from Example 2.4, and “Determinize” is from Example 2.5. The equations “Ex5.4(x, y)” (for $(x, y) \in \{(0, 0), (0.3, 0.3), (0.5, 0.5)\}$) are from Example 5.2. 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:

$$S = F G \quad F g = g \text{halt}(F g) \quad G x_0 x_1 = x_0 \oplus_p x_1.$$

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:

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

Our tool could correctly compute lower/upper bounds for both cases of $p = 0$ and $p = 0.01$ (though again, #iter had to be increased due to the slow convergence of the lower-bound computation). The equations “Incomp” and “Incomp2” consist of:

$$s = f(s) \quad f x = x^2 + 0.4x + 0.09, \quad \text{and} \quad s = f(s) \quad f x = 0.5x^2 + 2f(0.5x)$$

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

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

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 $0.5 \cdot \frac{1}{n^2} + 2 \cdot 0.5 \cdot \frac{c}{n^2} = \frac{c+0.5}{n^2}$. 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 do not help. We do not know, however, whether such situations occur in the fixpoint equations that arise from actual order-2 PHORSs; 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 “u.b.(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,³ 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 [23, 25, 6]. Our PHORS is 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 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 [23] to arbitrary orders.

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, [5, 22, 26, 12, 51, 1, 11]); 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, Kaminski and Katoen [40] have shown that almost sure termination of probabilistic imperative programs is Π_2^0 -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 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 [41, 55, 32, 44, 46, 48, 31, 30, 64, 59]. 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 a 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 [58, 38]. Most of the work in this field is concerned with operational semantics [18], denotational semantics (see, e.g., [39, 19, 21, 63, 2]), or program equivalence (see, e.g., [17, 15, 60]), which sometime becomes decidable (e.g. [54]), 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 [29] or Anglican [65]. There are some studies to analyze the termination behavior of probabilistic higher-order programs (with infinite data) by using types. Dal Lago and Grellois [16] generalized sized types [35, 4] to obtain a sound but highly incomplete technique. Breuvar and Dal Lago [8] developed systems of intersection types from which the termination probability

³As already mentioned, our method with piecewise linear functions may also be incomplete, but that does not show up in the current benchmark set.

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 [45]. Other undecidability results on probabilistic systems include the undecidability of the emptiness of probabilistic automata [27]. 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 in 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. This work was supported by JSPS KAKENHI Grant Number JP15H05706 and by ANR ELICA Grant Number ANR-14-CE25-0005.

References

- [1] M. Avanzini, U. D. Lago, and A. Yamada. On probabilistic term rewriting. In J. P. Gallagher and M. Sulzmann, editors, *Functional and Logic Programming - 14th International Symposium, FLOPS 2018, Nagoya, Japan, May 9-11, 2018, Proceedings*, volume 10818 of *Lecture Notes in Computer Science*, pages 132–148. Springer, 2018.
- [2] G. Bacci, R. Furber, D. Kozen, R. Mardare, P. Panangaden, and D. Scott. Boolean-valued semantics for the stochastic λ -calculus. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 669–678, 2018.
- [3] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- [4] G. Barthe, M. J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *MSCS*, 14(1):97–141, 2004.
- [5] O. Bournez and F. Garnier. Proving positive almost-sure termination. In J. Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2005.
- [6] T. Brázdil, V. Brozek, V. Forejt, and A. Kucera. Branching-time model-checking of probabilistic pushdown automata. *J. Comput. Syst. Sci.*, 80(1):139–156, 2014.
- [7] T. Brázdil, J. Esparza, S. Kiefer, and A. Kucera. Analyzing probabilistic pushdown automata. *Formal Methods in System Design*, 43(2):124–163, 2013.

- [8] F. Breuvar and U. Dal Lago. On intersection types and probabilistic lambda calculi. In *Proceedings of PPDP 2018*. ACM, 2018. To appear.
- [9] C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *Proceedings of CSL 2013*, volume 23 of *LIPIcs*, pages 129–148, 2013.
- [10] C. Calude. *Information and Randomness: An Algorithmic Perspective*. Springer-Verlag, Berlin, Heidelberg, 2nd edition, 2002.
- [11] A. Chakarov and S. Sankaranarayanan. Probabilistic program analysis with martingales. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, pages 511–526, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [12] K. Chatterjee, P. Novotný, and D. Zikelic. Stochastic invariants for probabilistic termination. In G. Castagna and A. D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 145–160. ACM, 2017.
- [13] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.
- [14] P. Cousot. Types as abstract interpretations. In P. Lee, F. Henglein, and N. D. Jones, editors, *Conference Record of POPL’97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997*, pages 316–331. ACM Press, 1997.
- [15] R. Crubillé and U. Dal Lago. On probabilistic applicative bisimulation and call-by-value λ -calculi. In Z. Shao, editor, *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8410 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2014.
- [16] U. Dal Lago and C. Grellois. Probabilistic termination by monadic affine sized typing. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, pages 393–419, 2017.
- [17] U. Dal Lago, D. Sangiorgi, and M. Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In Jagannathan and Sewell [37], pages 297–308.
- [18] U. Dal Lago and M. Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, 46(3):413–450, 2012.
- [19] V. Danos and R. Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.
- [20] K. de Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro. Computability by probabilistic machines. *Automata Studies*, 34:183–212, 1956.
- [21] T. Ehrhard, C. Tasson, and M. Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In Jagannathan and Sewell [37], pages 309–320.
- [22] J. Esparza, A. Gaiser, and S. Kiefer. Proving termination of probabilistic programs using patterns. In P. Madhusudan and S. A. Seshia, editors, *Computer Aided Verification*, pages 123–138, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [23] K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1):1:1–1:66, 2009.
- [24] K. Etessami and M. Yannakakis. Model checking of recursive probabilistic systems. *ACM Trans. Comput. Log.*, 13(2):12:1–12:40, 2012.

- [25] K. Etessami and M. Yannakakis. Recursive markov decision processes and recursive stochastic games. *J. ACM*, 62(2):11:1–11:69, 2015.
- [26] L. M. F. Fioriti and H. Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In S. K. Rajamani and D. Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 489–501. ACM, 2015.
- [27] H. Gimbert and Y. Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In S. Abramsky, C. Gavaille, C. Kirchner, F. M. auf der Heide, and P. G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 527–538. Springer, 2010.
- [28] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
- [29] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: A language for generative models. In *In UAI*, pages 220–229, 2008.
- [30] C. Grellois and P. Melliès. Finitary semantics of linear logic and higher-order model-checking. In Italiano et al. [36], pages 256–268.
- [31] C. Grellois and P. Melliès. Relational semantics of linear logic and higher-order model checking. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 260–276. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
- [32] M. Hague, A. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461. IEEE Computer Society, 2008.
- [33] C. Heunen, O. Kammar, S. Staton, and H. Yang. A convenient category for higher-order probability theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.
- [34] J. Hromkovic. *Design and Analysis of Randomized Algorithms - Introduction to Design Paradigms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2005.
- [35] J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In H. Boehm and G. L. S. Jr., editors, *POPL’96*, pages 410–423. ACM Press, 1996.
- [36] G. F. Italiano, G. Pighizzini, and D. Sannella, editors. *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, volume 9234 of *Lecture Notes in Computer Science*. Springer, 2015.
- [37] S. Jagannathan and P. Sewell, editors. *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL ’14, San Diego, CA, USA, January 20-21, 2014*. ACM, 2014.
- [38] C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS ’89), Pacific Grove, California, USA, June 5-8, 1989*, pages 186–195. IEEE Computer Society, 1989.
- [39] A. Jung and R. Tix. The troublesome probabilistic powerdomain. *Electronic Notes in Theoretical Computer Science*, 13:70 – 91, 1998. Comprox III, Third Workshop on Computation and Approximation.
- [40] B. L. Kaminski and J. Katoen. On the hardness of almost-sure termination. In Italiano et al. [36], pages 307–318.

- [41] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.
- [42] N. Kobayashi. Model-checking higher-order functions. In *Proceedings of PPDP 2009*, pages 25–36. ACM Press, 2009.
- [43] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of POPL*, pages 416–428. ACM Press, 2009.
- [44] N. Kobayashi. Model checking higher-order programs. *Journal of the ACM*, 60(3), 2013.
- [45] N. Kobayashi. Inclusion between the frontier language of a non-deterministic recursive program scheme and the dyck language is undecidable. submitted to a TCS special issue, 2018.
- [46] N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.
- [47] N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Logical Methods in Computer Science*, 7(4), 2011.
- [48] N. Kobayashi, R. Sato, and H. Unno. Predicate abstraction and CEGAR for higher-order model checking. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 222–233. ACM Press, 2011.
- [49] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.
- [50] Y. V. Matiyasevich. *Hilbert’s Tenth Problem*. The MIT Press, 1993.
- [51] A. McIver, C. Morgan, B. L. Kaminski, and J. Katoen. A new proof rule for almost-sure termination. *PACMPL*, 2(POPL):33:1–33:28, 2018.
- [52] A. Mista, A. Russo, and J. Hughes. Branching processes for quickcheck generators. *CoRR*, abs/1808.01520, 2018.
- [53] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.
- [54] A. S. Murawski and J. Ouaknine. On probabilistic program equivalence and refinement. In *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, pages 156–170, 2005.
- [55] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society Press, 2006.
- [56] S. M. Poulding and R. Feldt. Automated random testing in multiple dispatch languages. In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*, pages 333–344. IEEE Computer Society, 2017.
- [57] S. Ramsay, R. Neatherway, and C.-H. L. Ong. An abstraction refinement approach to higher-order model checking. In *Proceedings of POPL 2014*, pages 61–72. ACM, 2014.
- [58] N. Saheb-Djahromi. Probabilistic LCF. In J. Winkowski, editor, *Mathematical Foundations of Computer Science 1978, Proceedings, 7th Symposium, Zakopane, Poland, September 4-8, 1978*, volume 64 of *Lecture Notes in Computer Science*, pages 442–451. Springer, 1978.
- [59] S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *Proceedings of ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2011.

- [60] D. Sangiorgi and V. Vignudelli. Environmental bisimulations for probabilistic higher-order languages. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 595–607, 2016.
- [61] E. S. Santos. Probabilistic turing machines and computability. *Proceedings of the American Mathematical Society*, 22(3):704–710, 1969.
- [62] R. Statman. On the lambda Y calculus. *Ann. Pure Appl. Logic*, 130(1-3):325–337, 2004.
- [63] S. Staton, H. Yang, F. D. Wood, C. Heunen, and O. Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 525–534, 2016.
- [64] T. Tsukada and C. L. Ong. Compositional higher-order model checking via ω -regular games over böhm trees. In *Proceedings of CSL-LICS '14*, pages 78:1–78:10. ACM, 2014.
- [65] F. Wood, J. W. Meent, and V. Mansinghka. A New Approach to Probabilistic Programming Inference. In S. Kaski and J. Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 1024–1032, Reykjavik, Iceland, 22–25 Apr 2014. PMLR.
- [66] M. Yannakakis and K. Etessami. Checking LTL properties of recursive markov chains. In *Second International Conference on the Quantitative Evaluation of Systems (QEST 2005), 19-22 September 2005, Torino, Italy*, pages 155–165. IEEE Computer Society, 2005.

Appendix

A Additional Materials for Section 2

A.1 Encoding Recursive Markov Chains into Order-1 PHORSs

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 [23], Section 2.

Let us first of all fix a RMC $A = (A_1, \dots, A_k)$, 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 $i_I \in \{1, \dots, k\}$, 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 [23]). 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 $i \in \{1, \dots, k\}$, and s is a vertex A_i . The type $\mathcal{N}_A(F_{i,s})$ is $\mathfrak{o}^{|Ex_i|} \rightarrow \mathfrak{o}$. There is also a nonterminal S_A of type \mathfrak{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

$$(i_I, s_I, q_I) = S_A = F_{i_I, s_I}(\underbrace{\Omega, \dots, \Omega}_{j-1 \text{ times}}, \mathbf{halt}, \Omega, \dots, \Omega)$$

where $Ex_i = \{s_1, \dots, s_{|Ex_i|}\}$ and $q_I = s_j$.

- For every i and for every exit node $s_j \in Ex_i = \{s_1, \dots, s_{|Ex_i|}\}$, there is a rule

$$F_{i, s_j}(x_1, \dots, x_{|Ex_i|}) = x_j$$

- For each i and for each non-exit node or return port s of A_i , there is a rule

$$F_{i, s}(x_1, \dots, x) = \bigoplus_j p_s^q F_{i, q}(x_1, \dots, x)$$

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

$$F_{i, s}(\vec{x}) = F_{Y_i(b), en}(F_{i, (b, ex_1)}(\vec{x}), \dots, F_{i, (b, ex_v)}(\vec{x}))$$

and $Ex_{Y_i(b)} = \{ex_1, \dots, ex_v\}$.

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 $\langle \epsilon, s_I \rangle$.

B Additional Materials for Section 3

B.1 Rules of $\mathcal{G}_3^{P,Q}$

$S = \text{Loop Zero} \cdots \text{Zero}.$
 $\text{Loop } x_1 \cdots x_k = (\text{TestLt } (P x_1 \cdots x_k) (Q x_1 \cdots x_k))$
 $\quad \oplus_{\frac{1}{2}} (\text{TestAll } (\text{Succ } x_1) \cdots x_k) \oplus_{\frac{1}{2}} \cdots \oplus_{\frac{1}{2}} (\text{TestAll } x_1 \cdots (\text{Succ } x_k)).$
 $\text{TestLt } m_1 m_2 = \text{CheckHalf } (\text{CheckLt } m_1 m_2).$
 $\text{CheckHalf } y = F' y \text{ halt.} \quad F' g x = g x (F' g (F' g x)).$
 $\text{CheckLt } m_1 m_2 x y = (\text{NatToPr } m_1 x y) \oplus_{\frac{1}{2}} (\text{NatToPr } m_2 y x).$
 $\text{NatToPr } m x y = m (H x) y. \quad H x y = x \oplus_{\frac{1}{2}} y.$
 $\text{Zero } s z = z. \quad \text{Succ } n s z = s(n s z). \quad \text{Add } n m s z = n s(m s z). \quad \text{Mult } n m s z = n(m s) z.$
 $P x_1 \cdots x_k = \cdots (* \text{ constructed using } \text{Zero}, \text{Succ}, \text{Add}, \text{Mult} *).$
 $Q x_1 \cdots x_k = \cdots (* \text{ constructed using } \text{Zero}, \text{Succ}, \text{Add}, \text{Mult}^*).$

B.2 Rules of $\mathcal{G}_2^{P,Q}$

$S = \text{LoopPr OnePr } \underbrace{\text{ZeroPr} \cdots \text{ZeroPr}}_{(d_1+1)\cdots(d_k+1)-1}.$
 $\text{LoopPr } \tilde{x} = (\text{TestLtPr } (P' \tilde{x}) (Q' \tilde{x}))$
 $\quad \oplus_{\frac{1}{2}} (\text{LoopPr } (\text{Inc}_{1,(0,\dots,0)} \tilde{x}) \cdots (\text{Inc}_{1,(d_1,\dots,d_k)} \tilde{x}))$
 $\quad \oplus_{\frac{1}{2}} \cdots \oplus_{\frac{1}{2}} (\text{LoopPr } (\text{Inc}_{k,(0,\dots,0)} \tilde{x}) \cdots (\text{Inc}_{k,(d_1,\dots,d_k)} \tilde{x})).$
 $\text{TestLtPr } g_1 g_2 = \text{CheckHalf } (\text{CheckLtPr } g_1 g_2).$
 $\text{CheckLtPr } g_1 g_2 x y = (g_1 x y) \oplus_{\frac{1}{2}} (g_2 y x).$
 $\text{CheckHalf } y = F' y \text{ halt.} \quad F' g x = g x (F' g (F' g x)).$
 $\text{ZeroPr } x y = y. \quad \text{OnePr } x y = x \oplus_{\frac{1}{2}} y.$
 $\text{SuccPr } g x y = x \oplus_{\frac{1}{2}} (g x y). \quad \text{AddPr } g_1 g_2 x y = g_1 x (g_2 x y).$
 $P' \tilde{x} = \cdots (* \text{ PF encoding of } P, \text{ defined using } \text{ZeroPr} \text{ and } \text{AddPr} *).$
 $Q' \tilde{x} = \cdots (* \text{ PF encoding of } Q, \text{ defined using } \text{ZeroPr} \text{ and } \text{AddPr} *).$
 $\text{Inc}_{j,(i_1,\dots,i_k)} = \cdots (* \text{ for each } j \in \{1, \dots, k\}, i_1 \leq d_1, \dots, i_k \leq d_k *).$

C Additional Materials for Section 4

C.1 Proofs for Section 4.2

C.1.1 Proof of Lemma 4.1

We first prove the following lemma:

Lemma C.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 4.1. $\mathcal{N}^\# \vdash S : \mathbf{R}$ follows immediately from $\mathcal{N}(S) = \mathbf{o}$, Lemma C.1, and $\mathbf{o}^\# = \mathbf{R}$. Let \mathcal{R} be $\{F_i x_1 \cdots x_{\ell_i} = t_i \mid i \in \{1, \dots, m\}\}$. Then, by the definition of PHORS, we have $\mathcal{N}, x_1 : \kappa_{i,1}, \dots, x_{\ell_i} : \kappa_{i,\ell_i} \vdash t_i : \mathbf{o}$, with $\mathcal{N}(F) = \kappa_{i,1} \rightarrow \cdots \rightarrow \kappa_{i,\ell_i} \rightarrow \mathbf{o}$. We need to show that

$$\mathcal{N}^\#, x_1 : \kappa_{i,1}^\#, \dots, x_i : \kappa_{i,\ell_i}^\# \vdash t_i^\# : \mathbf{R}$$

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

C.1.2 Proof of Theorem 4.2

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 $\llbracket t^{\#} \rrbracket_{\rho}$.

Lemma C.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 $\mathcal{N} \vdash t : \circ$, 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 finite 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 **halt** (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

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

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

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

as required. \square

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

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

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

Lemma C.3 1. $\mathcal{G}^{(k)}$ is recursion-free.

2. $\mathcal{P}(\mathcal{G}) = \bigsqcup_{k \in \omega} \mathcal{P}(\mathcal{G}^{(k)})$.

3. $\mathcal{F}_{\mathcal{E}_{\mathcal{G}}}^k(\perp_{\llbracket \mathcal{N} \rrbracket})(F) = \mathcal{F}_{\mathcal{E}_{\mathcal{G}^{(k)}}}^k(\perp_{\llbracket \mathcal{N}^{(k)} \rrbracket})(F^{(k)}) = \text{lfp}(\mathcal{F}_{\mathcal{E}_{\mathcal{G}^{(k)}}})(F^{(k)})$ 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 $\{(\pi, p) \mid S \xrightarrow{\pi, p}_{\mathcal{G}} \text{halt}\}$ and $P^{(k)}$ be $\{(\pi, p) \mid S^{(k)} \xrightarrow{\pi, p}_{\mathcal{G}^{(k)}} \text{halt}\}$. 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 $S \xrightarrow{\pi, p}_{\mathcal{G}} \text{halt}$, then $S^{(|\pi|)} \xrightarrow{\pi, p}_{\mathcal{G}^{(|\pi|)}} \text{halt}$, because non-terminals are unfolded at most $|\pi|$ times in $S \xrightarrow{\pi, p}_{\mathcal{G}} \text{halt}$. Therefore, $P = \bigcup_k P^{(k)}$, from which the result follows.

3. We show that $\mathcal{F}_{\mathcal{E}_{\mathcal{G}}}^k(\perp_{\llbracket \mathcal{N} \rrbracket})(F) = \mathcal{F}_{\mathcal{E}_{\mathcal{G}(\ell)}}^k(\perp_{\llbracket \mathcal{N}(\ell) \rrbracket})(F^{(k)})$ holds for any $\ell \geq k$, 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:

$$\begin{aligned} \mathcal{F}_{\mathcal{E}_{\mathcal{G}(\ell)}}^k(\perp_{\llbracket \mathcal{N}(\ell) \rrbracket})(F^{(k)}) &= \llbracket ([F_1^{(k-1)} / F_1, \dots, F_m^{(k-1)} / F_m] \mathcal{R}(F))^{\#} \rrbracket_{\mathcal{F}_{\mathcal{E}_{\mathcal{G}(\ell)}}^{k-1}(\perp_{\llbracket \mathcal{N}(\ell) \rrbracket})} \\ &= \llbracket \mathcal{R}(F)^{\#} \rrbracket_{\{F_1 \mapsto \mathcal{F}_{\mathcal{E}_{\mathcal{G}(\ell)}}^{k-1}(\perp_{\llbracket \mathcal{N}(\ell) \rrbracket})(F_1^{(k-1)}), \dots, F_m \mapsto \mathcal{F}_{\mathcal{E}_{\mathcal{G}(\ell)}}^{k-1}(\perp_{\llbracket \mathcal{N}(\ell) \rrbracket})(F_m^{(k-1)})\}} \\ &= \llbracket \mathcal{R}(F)^{\#} \rrbracket_{\mathcal{F}_{\mathcal{E}_{\mathcal{G}}}^{k-1}(\perp_{\llbracket \mathcal{N} \rrbracket})} \text{ (by the induction hypothesis)} \\ &= \mathcal{F}_{\mathcal{E}_{\mathcal{G}}}^k(\perp_{\llbracket \mathcal{N} \rrbracket})(F), \end{aligned}$$

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

For the second equality, we can show that $\mathcal{F}_{\mathcal{E}_{\mathcal{G}(\ell)}}^k(\perp_{\llbracket \mathcal{N}(\ell) \rrbracket})(F^{(k)}) = \mathbf{lf}(\mathcal{F}_{\mathcal{E}_{\mathcal{G}(\ell)}})(F^{(k)})$ holds for any $\ell \geq k$, by straightforward induction on k .

□

Theorem 4.2 follows as a corollary of the above lemmas.

Proof of Theorem 4.2. By Lemmas C.2 and C.3, we have

$$\mathcal{P}(\mathcal{G}) = \bigsqcup_k \mathcal{P}(\mathcal{G}^{(k)}) = \bigsqcup_k \mathbf{lf}(\mathcal{F}_{\mathcal{E}_{\mathcal{G}^{(k)}}})(S^{(k)}) = \bigsqcup_k \mathcal{F}_{\mathcal{E}_{\mathcal{G}}}^k(\perp)(S) = \mathbf{lf}(\mathcal{F}_{\mathcal{E}_{\mathcal{G}}})(S)$$

as required.

□

C.2 Proofs for Section 4.3

C.2.1 Proof of Lemma 4.3

Lemma C.4 *If $\mathcal{K}, \tilde{x} : \tilde{\circ} \vdash t : \kappa$ and $\mathcal{K}; \tilde{x} \vdash t : \kappa \rightsquigarrow e$, then $\mathcal{K}^{\dagger} \vdash e : \kappa^{\dagger + |\tilde{x}|}$,*

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

□

We also prepare the following lemma on the syntactic property of the translation result, which is important for Lemma 4.3 and the substitution lemma (Lemma C.10 below) proved later.

Lemma C.5 *If $\mathcal{K}; \tilde{z} \vdash t : \kappa \rightsquigarrow (t_0, \dots, t_{\mathbf{ar}(\kappa)}, t_{\mathbf{ar}(\kappa)+1}, \dots, t_{\mathbf{ar}(\kappa)+|\tilde{z}|+1})$, then for each $y_i \in \text{dom}(\mathcal{K})$, $y_{i,0}$ occurs only in t_0 .*

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

□

Proof of Lemma 4.3. $\mathcal{N}^{\dagger}(S_1) = \mathbf{R}$ follows immediately from $\mathcal{N}(S) = \circ \rightarrow \circ$ 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 $\mathcal{N}(F) = \kappa_1 \rightarrow \cdots \kappa_{\ell} \Rightarrow \circ^k \rightarrow \circ$. Suppose also

$$\mathcal{N}, y_1 : \kappa_1, \dots, y_{\ell} : \kappa_{\ell}; x_1, \dots, x_k \vdash t_d : \circ \rightsquigarrow (t_{d,0}, \dots, t_{d,k+1}).$$

We need to show

$$\mathcal{N}^{\dagger}, (\tilde{y}_1) : \kappa_1^{\dagger}, \dots, (\tilde{y}_{\ell}) : \kappa_{\ell}^{\dagger} \vdash t_{d,0} : \mathbf{R}$$

and

$$\mathcal{N}^{\dagger}, (\tilde{y}_1') : \kappa_1^{\dagger'}, \dots, (\tilde{y}_{\ell}') : \kappa_{\ell}^{\dagger'} \vdash t_{d,i} : \mathbf{R}$$

for $i \in \{1, \dots, k\}$, where \tilde{y}_i and \tilde{y}_i' are as given in the premises of rule TR-RULE. The former follows immediately from Lemma C.4. For the latter, by Lemma C.4, we have

$$\mathcal{N}^{\dagger}, (\tilde{y}_1) : \kappa_1^{\dagger}, \dots, (\tilde{y}_{\ell}) : \kappa_{\ell}^{\dagger} \vdash t_{d,i} : \mathbf{R}.$$

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

$$\mathcal{N}^\dagger, (\tilde{y}_1') : \kappa_1^{\dagger'}, \dots, (\tilde{y}_\ell') : \kappa_\ell^{\dagger'} \vdash t_{d,i} : \mathbf{R}$$

as required. \square

C.2.2 Proof of Theorem 4.4

Given two expressions e_1, e_2 and fixpoint equations $\mathcal{E}_G^{\text{ref}}$, we write $e_1 \cong_{\mathcal{E}_G^{\text{ref}}} e_2$ if $\llbracket e_1 \rrbracket_{\rho_{\mathcal{E}_G^{\text{ref}}}} = \llbracket e_2 \rrbracket_{\rho_{\mathcal{E}_G^{\text{ref}}}}$. We often omit the subscript.

As sketched in Section 4.3, we first prove the theorem for recursion-free PHORSs. 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, $\mathfrak{b}(t) > \mathfrak{b}(t_L), \mathfrak{b}(t_R)$, where $\mathfrak{b}(t)$ 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 $\xrightarrow{d,p}_G$ 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.

$$t \text{ (extended terms)} ::= \Omega \mid x \mid t_1 t_2 \mid \{t_1/x\}t_2$$

Here, $\{t_1/x\}t_2$ is an *explicit* substitution; the intended meaning is the same as the ordinary substitution $[t_1/x]t_2$, but the substitution is delayed until x becomes necessary. Note that we have omitted **halt**; we consider an open term Sx as the initial term instead of $S\mathbf{halt}$. The type judgment relation for terms is extended by adding the following rule:

$$\frac{\mathcal{K} \vdash t_1 : \circ \quad \mathcal{K}, x : \circ \vdash t_2 : \circ}{\mathcal{K} \vdash \{t_1/x\}t_2 : \circ}$$

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:

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

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

$$\begin{array}{c} \frac{x \neq z}{E[\{t/z\}x] \xrightarrow{\epsilon,1}_{\text{es},G} E[x]} \\[10pt] \frac{}{E[\{t/x\}x] \xrightarrow{\epsilon,1}_{\text{es},G} E[t]} \\[10pt] \frac{\mathcal{N}(F) = \tilde{\kappa} \Rightarrow \circ^\ell \rightarrow \circ \quad \mathcal{R}(F) = \lambda \tilde{y}. \lambda \tilde{z}. u_L \oplus_p u_R \quad \ell = |\tilde{z}| = |\tilde{t}| \quad |\tilde{y}| = |\tilde{s}| \quad \tilde{z} \text{ do not occur in } E[F \tilde{s} \tilde{t}]}{E[F \tilde{s} \tilde{t}] \xrightarrow{L,R}_{\text{es},G} E[\{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u_L]} \end{array}$$

$$\frac{\mathcal{R}(F) = \lambda \tilde{y}. \lambda \tilde{z}. u_L \oplus_p u_R \quad \mathcal{N}(F) = \tilde{\kappa} \Rightarrow \mathbf{o}^\ell \rightarrow \mathbf{o} \quad \ell = |\tilde{z}| = |\tilde{t}| \quad |\tilde{y}| = |\tilde{s}| \quad \tilde{z} \text{ do not occur in } E[F \tilde{s} \tilde{t}]}{E[F \tilde{s} \tilde{t}] \xrightarrow{R, 1}_{\text{es}, \mathcal{G}}^p E[\{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u_R]}$$

We call reductions by the first two rules (i.e., reductions labeled by $\xrightarrow{\epsilon, p}_{\text{es}, \mathcal{G}}$) *administrative reductions*. In the last two rules, we assume that α -conversion is implicitly applied so that \tilde{z} do not clash with variables that are already used in $E[F \tilde{s} \tilde{t}]$. In those rules, recall also our notational convention that when we write $F \tilde{s} \tilde{t}$, the second sequence \tilde{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 C.6 *Let s be an extended term.*

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

Proof. Trivial from the definitions of $\xrightarrow{d, p}$ and $\xrightarrow{d, p}_{\text{es}}$. \square

We define $\xrightarrow{\pi, p}_{\text{es}, \mathcal{G}}$ in an analogous manner to $\xrightarrow{\pi, p}_{\mathcal{G}}$, 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 $P_{\text{es}}(\mathcal{G}, t, x)$ for the set $\{(\pi, p) \mid t \xrightarrow{\pi, p}_{\text{es}, \mathcal{G}} x\}$, and write $\mathcal{P}_{\text{es}}(\mathcal{G}, t, x)$ for $\sum_{(\pi, p) \in P_{\text{es}}(\mathcal{G}, t, x)} p$, based on the new reduction relation. The following lemma follows immediately from the above definitions and Lemma C.6.

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

Proof. By Lemma C.6, $t \xrightarrow{\pi, p}_{\text{es}} x$ if and only if $t^* \xrightarrow{\pi, p} x$, if and only if $[\text{halt}/x]t^* \xrightarrow{\pi, p} \text{halt}$ (for the second “if and only if”, recall the assumption that **halt** does not occur in \mathcal{R}), from which the result follows. \square

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

$$\frac{\mathcal{K}; x_1, \dots, x_k \vdash s : \mathbf{o} \rightsquigarrow (s_0, \dots, s_{k+1}) \quad \mathcal{K}; z, x_1, \dots, x_k \vdash t : \mathbf{o} \rightsquigarrow (t_0, \dots, t_{k+2})}{\mathcal{K}; x_1, \dots, x_k \vdash \{s/z\}t : \mathbf{o} \rightsquigarrow (t_0 + t_1 \cdot s_0, t_2 + t_1 \cdot s_1, \dots, t_{k+2} + t_1 \cdot s_{k+1})} \quad (\text{TR-SUB})$$

We shall prove that the translation relation is preserved by the new reduction relation (Lemmas C.12, C.13, and C.14 below).

Lemma C.8 (weakening) *1. If $\mathcal{K}; x_1, \dots, x_k \vdash t : \kappa \rightsquigarrow e$, then $\mathcal{K}, y : \kappa_y; x_1, \dots, x_k \vdash t : \kappa \rightsquigarrow e$.*

2. If $\mathcal{K}; x_1, \dots, x_k \vdash t : \kappa \rightsquigarrow (t_0, \dots, t_\ell)$, then $\mathcal{K}; x_1, \dots, x_k, x_{k+1} \vdash t : \kappa \rightsquigarrow (t_0, \dots, t_\ell, t_\ell)$.

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

Lemma C.9 (exchange)

If $\mathcal{K}; x_1, \dots, x_i, x_{i+1}, \dots, x_k \vdash t : \kappa \rightsquigarrow (t_0, \dots, t_{\text{ar}(\kappa)+i}, t_{\text{ar}(\kappa)+i+1}, \dots, t_{\text{ar}(\kappa)+k+1})$, then $\mathcal{K}, y : \kappa_y; x_1, \dots, x_{i+1}, x_i, \dots, x_k \vdash t : \kappa \rightsquigarrow (t_0, \dots, t_{\text{ar}(\kappa)+i+1}, t_{\text{ar}(\kappa)+i}, \dots, t_{\text{ar}(\kappa)+k+1})$.

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

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 C.10 (substitution) *Suppose t does not contain any explicit substitutions (i.e., any subterms of the form $\{u/x\}s$). If $\mathcal{K}, \tilde{y} : \kappa_{\tilde{y}}; \tilde{z} \vdash t : \kappa \rightsquigarrow (\tilde{t}, t_{m+1})$ and $\mathcal{K}; x_1, \dots, x_k \vdash s_i : \kappa_{y,i} \rightsquigarrow (\tilde{s}_i, s_{i,\ell_i+1}, \dots, s_{i,\ell_i+k+1})$ with $\{x_1, \dots, x_k\} \cap \{\tilde{z}\} = \emptyset$ and $\ell_i = \mathbf{ar}(\kappa_{y,i})$, then:*

$$\mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash [\tilde{s}/\tilde{y}]t : \kappa \rightsquigarrow (\theta_0 \tilde{t}, \theta_{\ell+1} t_0, \dots, \theta_{\ell+k} t_0, \theta_0 t_{m+1}),$$

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

$$\begin{aligned} \theta_j &= \theta_{1,j} \cdots \theta_{|\tilde{s}|,j} \text{ for } j \in \{1, \dots, |\tilde{x}|\} \\ \theta_{i,0} &= [s_{i,0}/y_{i,0}, \dots, s_{i,\mathbf{ar}(\kappa_i)}/y_{i,\mathbf{ar}(\kappa_i)}, s_{i,\mathbf{ar}(\kappa_i)+|\tilde{x}|+1}/y_{i,\mathbf{ar}(\kappa_i)+1}] \text{ for } i \in \{1, \dots, |\tilde{s}|\} \\ \theta_{i,j} &= [s_{i,\mathbf{ar}(\kappa_i)+j}/y_{i,0}, \dots, s_{i,\mathbf{ar}(\kappa_i)}/y_{i,\mathbf{ar}(\kappa_i)}, s_{i,\mathbf{ar}(\kappa_i)+|\tilde{x}|+1}/y_{i,\mathbf{ar}(\kappa_i)+1}] \\ &\quad \text{for } i \in \{1, \dots, |\tilde{s}|\}, j \in \{1, \dots, |\tilde{x}|\}. \end{aligned}$$

Here, the part $\theta_{\ell+1} t_0, \dots, \theta_{\ell+k} t_0$ accounts for information about reachability to the newly introduced variables x_1, \dots, x_k .

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

- Case TR-OMEGA: In this case, $t = \Omega$ with $\tilde{t} = \tilde{0}$ and $t_{m+1} = 0$. Thus, the result follows immediately from the rule TR-OMGA.
- Case TR-GVAR: In this case, $t = z_i$ with $\tilde{t} = 0^i, 1, 0^{|\tilde{z}|-i}$ and $t_{m+1} = 0$. By using TR-GVAR, we obtain

$$\mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash [\tilde{s}/\tilde{y}]t (= z_i) : \kappa \rightsquigarrow (0^i, 1, 0^{|\tilde{z}|+k-i+1}).$$

Since

$$(\theta_0 \tilde{t}, \theta_{\ell+1} t_0, \dots, \theta_{\ell+k} t_0, \theta_0 t_{m+1}) = (0^i, 1, 0^{|\tilde{z}|+k-i+1}),$$

we have the required result.

- Case TR-VAR: If $t \notin \{\tilde{y}\}$, then the result follows immediately from TR-VAR. If $t = y_i$, then $\tilde{t} = y_{i,0}, \dots, y_{i,\ell_i}, y_{i,\ell_i+1}^{|\tilde{z}|}$, and $t_{m+1} = y_{i,\ell_i+1}$. By applying Lemmas C.8 and C.9 to $\mathcal{K}; x_1, \dots, x_k \vdash s_i : \kappa_{y,i} \rightsquigarrow (\tilde{s}_i, s_{i,\ell_i+1}, \dots, s_{i,\ell_i+k+1})$, we obtain:

$$\mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash s_i : \kappa_{y,i} \rightsquigarrow (\tilde{s}_i, s_{i,\ell_i+k+1}^{|\tilde{z}|}, s_{i,\ell_i+1}, \dots, s_{i,\ell_i+k+1}).$$

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

$$(\theta_0 \tilde{t}, \theta_1 t_0, \dots, \theta_k t_0, \theta_0 t_{m+1}) = (\tilde{s}_i, s_{i,\ell_i+k+1}^{|\tilde{z}|}, s_{i,\ell_i+1}, \dots, s_{i,\ell_i+k+1}).$$

Thus we have the required result.

- Case TR-APP: In this case, we have $t = uv$ and

$$\begin{aligned} \mathcal{K}, \tilde{y} : \kappa_{\tilde{y}}; \tilde{z} \vdash u : \kappa_v &\rightarrow \kappa \rightsquigarrow (u_0, \tilde{u}, u_{\ell'+1}, \dots, u_{\ell'+|\tilde{z}|+1}) \\ \mathcal{K}, \tilde{y} : \kappa_{\tilde{y}}; \tilde{z} \vdash v : \kappa_v &\rightsquigarrow (v_0, \tilde{v}, v_{\ell''+1}, \dots, v_{\ell''+|\tilde{z}|+1}) \\ \tilde{t} &= u_0(v_0, \tilde{v}, v_{\ell''+|\tilde{z}|+1}), \tilde{u}(\tilde{v}, v_{\ell''+|\tilde{z}|+1}), \\ &\quad u_{\ell'+1}(v_{\ell''+1}, \tilde{v}, v_{\ell''+|\tilde{z}|+1}), \dots, u_{\ell'+|\tilde{z}|}(v_{\ell''+|\tilde{z}|}, \tilde{v}, v_{\ell''+|\tilde{z}|+1}) \\ t_{m+1} &= u_{\ell'+|\tilde{z}|+1}(v_{\ell''+|\tilde{z}|+1}, \tilde{v}, v_{\ell''+|\tilde{z}|+1}) \\ \ell' = |\tilde{u}| = \mathbf{ar}(\kappa) \quad m &= \ell' + |\tilde{z}| \quad \ell'' = \mathbf{ar}(\kappa_v) = |\tilde{v}|. \end{aligned}$$

By the induction hypothesis, we have:

$$\begin{aligned} \mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash [\tilde{s}/\tilde{y}]u : \kappa_v \rightarrow \kappa \rightsquigarrow \\ (\theta_0(u_0, \tilde{u}, u_{\ell'+1}, \dots, u_{\ell'+|\tilde{z}|}), \theta_1 u_0, \dots, \theta_k u_0, \theta_0 u_{\ell'+|\tilde{z}|+1}) \\ \mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash [\tilde{s}/\tilde{y}]v : \kappa_v \rightsquigarrow \\ (\theta_0(v_0, \tilde{v}, v_{\ell'+1}, \dots, v_{\ell'+|\tilde{z}|}), \theta_1 v_0, \dots, \theta_k v_0, \theta_0 v_{\ell'+|\tilde{z}|+1}) \end{aligned}$$

By applying TR-APP, we obtain:

$$\begin{aligned} \mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash ([\tilde{s}/\tilde{y}]u)([\tilde{s}/\tilde{y}]v) : \kappa \rightsquigarrow \\ ((\theta_0 u_0)(\theta_0 v_0, \theta_0 \tilde{v}, \theta_0 v_{\ell'+|\tilde{z}|+1}), (\theta_0 \tilde{u})(\theta_0 \tilde{v}, \theta_0 v_{\ell'+|\tilde{z}|+1}), \\ (\theta_0 u_{\ell'+1})(\theta_0 v_{\ell'+1}, \theta_0 \tilde{v}, \theta_0 v_{\ell'+|\tilde{z}|+1}), \dots, (\theta_0 u_{\ell'+|\tilde{z}|})(\theta_0 v_{\ell'+|\tilde{z}|}, \theta_0 \tilde{v}, \theta_0 v_{\ell'+|\tilde{z}|+1}), \\ (\theta_1 u_0)(\theta_1 v_0, \theta_0 \tilde{v}, \theta_0 v_{\ell'+|\tilde{z}|+1}), \dots, (\theta_k u_0)(\theta_k v_0, \theta_0 \tilde{v}, \theta_0 v_{\ell'+|\tilde{z}|+1}) \\ (\theta_0 u_{\ell'+|\tilde{z}|+1})(\theta_0 v_{\ell'+|\tilde{z}|+1}, \theta_0 \tilde{v}, \theta_0 v_{\ell'+|\tilde{z}|+1})). \end{aligned}$$

Since y_0 does not occur in \tilde{v} and $v_{\ell'+|\tilde{z}|+1}$ (Lemma C.5), $\theta_0 \tilde{v}$ and $\theta_0 u_{\ell'+|\tilde{z}|+1}$ are equivalent to $\theta_j \tilde{v}$ and $\theta_j u_{\ell'+|\tilde{z}|+1}$ respectively for any $j \in \{1, \dots, k\}$. Therefore, the whole output of transformation is equivalent to:

$$\begin{aligned} (\theta_0(u_0(v_0, \tilde{v}, v_{\ell'+|\tilde{z}|+1})), \theta_0(\tilde{u}(\tilde{v}, v_{\ell'+|\tilde{z}|}))), \\ \theta_0(u_{\ell'+1}(v_{\ell'+1}, \tilde{v}, v_{\ell'+|\tilde{z}|+1})), \dots, \theta_0(u_{\ell'+|\tilde{z}|}(v_{\ell'+|\tilde{z}|}, \tilde{v}, v_{\ell'+|\tilde{z}|+1})), \\ \theta_1(u_0(v_0, \tilde{v}, v_{\ell'+|\tilde{z}|+1})), \dots, \theta_k(u_0(v_0, \tilde{v}, v_{\ell'+|\tilde{z}|+1})), \\ \theta_0(u_{\ell'+|\tilde{z}|+1}(v_{\ell'+|\tilde{z}|+1}, \tilde{v}, v_{\ell'+|\tilde{z}|+1}))) \end{aligned}$$

Thus, we have the required result.

- Case TR-APPG: In this case, we have $t = uv$ and:

$$\begin{aligned} \mathcal{K}, \tilde{y} : \tilde{\kappa}_{\tilde{y}}; \tilde{z} \vdash u : \mathbf{o}^{\ell'+1} \rightarrow \mathbf{o} \rightsquigarrow (u_0, \dots, u_{\ell'+|\tilde{z}|+2}) \\ \mathcal{K}, \tilde{y} : \tilde{\kappa}_{\tilde{y}}; \tilde{z} \vdash v : \mathbf{o} \rightsquigarrow (v_0, \dots, v_{|\tilde{z}|+1}) \\ (t, t_{m+1}) = (u_0 + u_1 \cdot v_0, u_2, \dots, u_{\ell'+1}, u_{\ell'+2} + u_1 \cdot v_1, \dots, u_{\ell'+|\tilde{z}|+2} + u_1 \cdot v_{|\tilde{z}|+1}) \\ \kappa = \mathbf{o}^{\ell'} \rightarrow \mathbf{o} \end{aligned}$$

By the induction hypothesis, we have:

$$\begin{aligned} \mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash [\tilde{s}/\tilde{y}]u : \mathbf{o}^{\ell'+1} \rightarrow \mathbf{o} \rightsquigarrow \\ (\theta_0 u_0, \dots, \theta_0 u_{\ell'+|\tilde{z}|+1}, \theta_1 u_0, \dots, \theta_k u_0, \theta_0 u_{\ell'+|\tilde{z}|+k+2}) \\ \mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash [\tilde{s}/\tilde{y}]v : \mathbf{o} \rightsquigarrow (\theta_0 v_0, \dots, \theta_0 v_{|\tilde{z}|}, \theta_1 v_0, \dots, \theta_k v_0, \theta_0 v_{|\tilde{z}|+1}). \end{aligned}$$

By applying TR-APPG, we obtain:

$$\begin{aligned} \mathcal{K}; \tilde{z}, x_1, \dots, x_k \vdash ([\tilde{s}/\tilde{y}]u)([\tilde{s}/\tilde{y}]v) : \mathbf{o}^{\ell'} \rightarrow \mathbf{o} \rightsquigarrow \\ (\theta_0 u_0 + \theta_0 u_1 \cdot \theta_0 v_0, \theta_0 u_2, \dots, \theta_0 u_{\ell'+1}, \\ \theta_0 u_{\ell'+2} + \theta_0 u_1 \cdot \theta_0 v_1, \dots, \theta_0 u_{\ell'+|\tilde{z}|+1} + \theta_0 u_1 \cdot \theta_0 v_{|\tilde{z}|}, \\ \theta_1 u_0 + \theta_0 u_1 \cdot \theta_1 v_0, \dots, \theta_k u_0 + \theta_0 u_1 \cdot \theta_k v_0, \theta_0 u_{\ell'+|\tilde{z}|+2} + \theta_0 u_1 \cdot \theta_0 v_{|\tilde{z}|+1}) \end{aligned}$$

Since u_1 does not contain any occurrence of y_0 (Lemma C.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:

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

Thus, we have the required result. □

The following is a “substitution lemma” for explicit substitutions.

Lemma C.11 (explicit substitution) *If $\mathcal{K}; x_1, \dots, x_k \vdash t : \mathbf{o} \rightsquigarrow (t_0, \dots, t_{k+1})$ and $\mathcal{K}; s : \mathbf{o} \rightsquigarrow (s_0, s_1)$, then $\mathcal{K}; x_2, \dots, x_k \vdash \{s/x_1\}t : \mathbf{o} \rightsquigarrow (t_0 + t_1 \cdot s_0, t_2 + t_1 \cdot s_1, \dots, t_{k+1} + t_1 \cdot s_1)$.*

Proof. By applying Lemma C.8 to $\mathcal{K}; \vdash s : \circ \rightsquigarrow (s_0, s_1)$, we obtain $\mathcal{K}; x_2, \dots, x_k \vdash s : \circ \rightsquigarrow (s_0, s_1, \dots, s_1)$. By using TR-SUB, we obtain the required result. \square

We are now ready to prove that the translation relation is preserved by reductions (Lemmas C.12, C.13, and C.14 below).

Lemma C.12 (subject reduction) *If $\mathcal{K}; \tilde{x} \vdash F \tilde{s} \tilde{t} : \circ \rightsquigarrow (v_0, \dots, v_{|\tilde{x}|+1})$ and $\mathcal{R}(F) = \lambda \tilde{y}. \lambda \tilde{z}. u_L \oplus_p u_R$ (where $\{\tilde{x}\} \cap \{\tilde{z}\} = \emptyset$), then there exist $w_{L,i}, w_{R,i}$ ($i \in \{0, \dots, |\tilde{x}| + 1\}$) such that $\mathcal{K}; \tilde{x} \vdash \{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u_d \rightsquigarrow (w_{d,0}, \dots, w_{d,|\tilde{x}|+1})$ and $v_i \cong pw_{L,i} + (1-p)w_{R,i}$ for each $i \in \{0, \dots, |\tilde{x}| + 1\}$.*

Proof. By the assumptions, we have:

$$\begin{aligned}
& \mathcal{K}; \tilde{x} \vdash s_i : \kappa_i \rightsquigarrow (s_{i,0}, \dots, s_{i,\text{ar}(\kappa_i)+|\tilde{x}|+1}) \text{ for each } i \in \{1, \dots, |\tilde{s}|\} \\
& \mathcal{K}; \tilde{x} \vdash t_i : \circ \rightsquigarrow (t_{i,0}, \dots, t_{i,|\tilde{x}|+1}) \text{ for each } i \in \{1, \dots, |\tilde{z}|\} \\
& v'_0 = F_0(s_{1,0}, \dots, s_{1,\text{ar}(\kappa_1)}, s_{1,\text{ar}(\kappa_1)+|\tilde{x}|+1}) \cdots (s_{|\tilde{s}|,0}, \dots, s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})}, s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})+|\tilde{x}|+1}) \\
& v'_j = F_j(s_{1,1}, \dots, s_{1,\text{ar}(\kappa_1)}, s_{1,\text{ar}(\kappa_1)+|\tilde{x}|+1}) \cdots (s_{|\tilde{s}|,1}, \dots, s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})}, s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})+|\tilde{x}|+1}) \\
& \hspace{15em} \text{for each } j \in \{1, \dots, |\tilde{z}|\} \\
& v'_{|\tilde{z}|+j} = F_0(s_{1,\text{ar}(\kappa_1)+j}, s_{1,1}, \dots, s_{1,\text{ar}(\kappa_1)}, s_{1,\text{ar}(\kappa_1)+|\tilde{x}|+1}) \cdots \\
& \hspace{10em} (s_{|\tilde{s}|,\text{ar}(\kappa_1)+j}, s_{|\tilde{s}|,1}, \dots, s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})}, s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})+|\tilde{x}|+1}) \\
& \hspace{15em} \text{for each } j \in \{1, \dots, |\tilde{x}|\} \\
& v'_{|\tilde{z}|+|\tilde{x}|+1} = F_{|\tilde{z}|+1}(s_{1,\text{ar}(\kappa_1)+|\tilde{x}|+1}, s_{1,1}, \dots, s_{1,\text{ar}(\kappa_1)}, s_{1,\text{ar}(\kappa_1)+|\tilde{x}|+1}) \cdots \\
& \hspace{10em} (s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})+|\tilde{z}|+1}, s_{|\tilde{s}|,1}, \dots, s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})}, s_{|\tilde{s}|,\text{ar}(\kappa_{|\tilde{s}|})+|\tilde{x}|+1}) \\
& v_0 \cong v'_0 + v'_1 \cdot t_{1,0} + \cdots + v'_{|\tilde{z}|} \cdot t_{|\tilde{z}|,0} \\
& v_{|\tilde{z}|+i} \cong v'_{|\tilde{z}|+i} + v'_1 \cdot t_{1,i} + \cdots + v'_{|\tilde{z}|} \cdot t_{|\tilde{z}|,i} \text{ for each } i \in \{1, \dots, |\tilde{x}| + 1\} \\
& \mathcal{K}, \tilde{y} : \tilde{\kappa}; \tilde{z} \vdash u_d : \circ \rightsquigarrow (u_{d,0}, \dots, u_{d,|\tilde{z}|+1}) \text{ for } d \in \{L, R\}
\end{aligned}$$

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

$$\mathcal{K}; \tilde{z}, \tilde{x} \vdash [\tilde{s}/\tilde{y}]u_d : \circ \rightsquigarrow (w'_{d,0}, \dots, w'_{d,|\tilde{z}|+|\tilde{x}|+1})$$

where

$$\begin{aligned}
& (w'_{d,0}, \dots, w'_{d,|\tilde{z}|+|\tilde{x}|+1}) = \\
& (\theta_0 u_{d,0}, \dots, \theta_0 u_{d,|\tilde{z}|}, \theta_1 u_{d,0}, \dots, \theta_{|\tilde{x}|} u_{d,0}, \theta_0 u_{d,|\tilde{z}|+1}) \\
& \theta_j = \theta_{1,j} \cdots \theta_{|\tilde{s}|,j} \text{ for } j \in \{1, \dots, |\tilde{x}| + 1\} \\
& \theta_{i,0} = [s_{i,0}/y_{i,0}, \dots, s_{i,\text{ar}(\kappa_i)}/y_{i,\text{ar}(\kappa_i)}, s_{i,\text{ar}(\kappa_i)+|\tilde{x}|+1}/y_{i,\text{ar}(\kappa_i)+1}] \text{ for } i \in \{1, \dots, |\tilde{s}|\} \\
& \theta_{i,j} = [s_{i,\text{ar}(\kappa_i)+j}/y_{i,0}, \dots, s_{i,\text{ar}(\kappa_i)}/y_{i,\text{ar}(\kappa_i)}, s_{i,\text{ar}(\kappa_i)+|\tilde{x}|+1}/y_{i,\text{ar}(\kappa_i)+1}] \\
& \hspace{15em} \text{for } i \in \{1, \dots, |\tilde{s}|\}, j \in \{1, \dots, |\tilde{x}| + 1\}.
\end{aligned}$$

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

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

Then, the required result is obtained by repeated applications of TR-SUB to

$$\mathcal{K}; \tilde{z}, \tilde{x} \vdash [\tilde{s}/\tilde{y}]u_d : \circ \rightsquigarrow (w'_{d,0}, \dots, w'_{d,|\tilde{z}|+|\tilde{x}|+1}).$$

\square

Lemma C.13 (subject reduction (for administrative step)) *If $\mathcal{K}; \tilde{x} \vdash \{s/z\}x_i : \circ \rightsquigarrow (t_0, \dots, t_{|\tilde{x}|+1})$ with $x_i \neq z$, then $\mathcal{K}; \tilde{x} \vdash x_i : \circ \rightsquigarrow (u_0, \dots, u_{|\tilde{x}|+1})$ for some u_j ($j \in \{0, \dots, |\tilde{x}| + 1\}$) such that $t_j \cong u_j$ for each $j \in \{0, \dots, |\tilde{x}| + 1\}$.*

Proof. By the assumption $\mathcal{K}; \tilde{x} \vdash \{s/z\}x_i : \circ \rightsquigarrow (t_0, \dots, t_{|\tilde{x}|+1})$, we have:

$$\begin{aligned} \mathcal{K}; z, \tilde{x} \vdash x_i : \circ &\rightsquigarrow (0^{i+1}, 1, 0^{|\tilde{x}|-i+1}) \\ \mathcal{K}; \tilde{x} \vdash s : \circ &\rightsquigarrow (s_0, \dots, s_{|\tilde{x}|+1}) \\ t_i &= 1 + 0 \cdot s_i \cong 1 \quad t_j \cong 0 \text{ for } j \neq i \end{aligned}$$

Since $\mathcal{K}; \tilde{x} \vdash x_i : \circ \rightsquigarrow (0^i, 1, 0^{|\tilde{x}|-i+1})$, we have the required condition for: $u_i = 1$ and $u_j = 0$ for $j \neq i$. \square

Lemma C.14 (subject reduction (for administrative step (ii))) *If $\mathcal{K}; \tilde{x} \vdash \{s/z\}z : \circ \rightsquigarrow (t_0, \dots, t_{|\tilde{x}|+1})$, then $\mathcal{K}; \tilde{x} \vdash s : \circ \rightsquigarrow (u_0, \dots, u_{|\tilde{x}|+1})$ for some u_j ($j \in \{0, \dots, |\tilde{x}|+1\}$) such that $t_j \cong u_j$ for each $j \in \{0, \dots, |\tilde{x}|+1\}$.*

Proof. By the assumption $\mathcal{K}; \tilde{x} \vdash \{s/z\}z : \circ \rightsquigarrow (t_0, \dots, t_{|\tilde{x}|+1})$, we have:

$$\begin{aligned} \mathcal{K}; z, \tilde{x} \vdash z : \circ &\rightsquigarrow (0, 1, 0^{|\tilde{x}|+1}) \\ \mathcal{K}; \tilde{x} \vdash s : \circ &\rightsquigarrow (s_0, \dots, s_{|\tilde{x}|+1}) \\ t_i &= 0 + 1 \cdot s_i \text{ for each } i \in \{0, \dots, |\tilde{x}|+1\}. \end{aligned}$$

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

We are now ready to prove Theorem 4.4, restricted to recursion-free PHORSs (the definition of recursion-free PHORSs is found in Section C.1).

Lemma C.15 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{R}, S)$ be a recursion-free PHORS, and ρ be the least solution of $\mathcal{E}_{\mathcal{G}}^{\text{ref}}$. Then, $\mathcal{P}(\mathcal{G}, S \text{halt}) = \rho(S_1)$.*

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

Suppose $b(t) = 0$. 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 $b(t) > 0$, we perform case analysis on a reduction of t . If $t \xrightarrow{\epsilon, 1}_{\text{es}} t'$, then by Lemmas C.13 and C.14, $\mathcal{N}; x \vdash t \rightsquigarrow (t'_0, t'_1, t'_2)$ and $t_1 \cong t'_1$ for some (t'_0, t'_1, t'_2) . By the induction hypothesis, $\mathcal{P}_{\text{es}}(\mathcal{G}, t', x) \cong t'_1$. Therefore, we have $\mathcal{P}_{\text{es}}(\mathcal{G}, t, x) = \mathcal{P}_{\text{es}}(\mathcal{G}, t', x) \cong t'_1 \cong t_1$ as required.

If $t \xrightarrow{d,p}_{\text{es}} t'$ for $d \in \{L, R\}$, then t must be of the form $E[F \tilde{s} \tilde{t}]$, $\mathcal{R}(F) = \lambda \tilde{y}. \lambda \tilde{z}. u_L \oplus_p u_R$, $t \xrightarrow{L,p}_{\text{es}} t_L$, and $t \xrightarrow{R, 1-p}_{\text{es}} t_R$ with $t_L = E[\{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u_L]$ and $t_R = E[\{\tilde{t}/\tilde{z}\}[\tilde{s}/\tilde{y}]u_R]$. By Lemma C.12, there exist $(t_{L,0}, t_{L,1}, t_{L,2})$ and $(t_{R,0}, t_{R,1}, t_{R,2})$ such that $\mathcal{N}; x \vdash t_L \rightsquigarrow (t_{L,0}, t_{L,1}, t_{L,2})$ and $\mathcal{N}; x \vdash t_R \rightsquigarrow (t_{R,0}, t_{R,1}, t_{R,2})$ with $t_1 \cong pt_{L,1} + (1-p)t_{R,1}$. Thus, we have $\mathcal{P}_{\text{es}}(\mathcal{G}, t, x) = p\mathcal{P}_{\text{es}}(\mathcal{G}, t_L, x) + (1-p)\mathcal{P}_{\text{es}}(\mathcal{G}, t_R, x) \cong pt_{L,1} + (1-p)t_{R,1} \cong t_1$ as required. \square

We are now ready to prove Theorem 4.4.

Proof of Theorem 4.4. Consider the finite approximation $\mathcal{G}^{(k)}$ (defined in Section C.1). Then, we have

$$\begin{aligned} \mathcal{P}(\mathcal{G}, S \text{halt}) &= \bigsqcup_k \mathcal{P}(\mathcal{G}^{(k)}, S^{(k)} \text{halt}) = \bigsqcup_k \mathbf{lfp}(\mathcal{F}_{\mathcal{G}^{(k)}})(S_1^{(k)}) \\ &= \bigsqcup_k \mathcal{F}_{\mathcal{G}}^k(\perp)(S_1) = \mathbf{lfp}(\mathcal{F}_{\mathcal{G}})(S_1) = \rho_{\mathcal{E}_{\mathcal{G}}^{\text{ref}}}(S_1) \end{aligned}$$

as required. Here $\mathcal{F}_{\mathcal{G}}$ is the functional associated with fixpoint equations $\mathcal{E}_{\mathcal{G}}^{\text{ref}}$, as defined in Section 4.1, and we can use essentially the same reasoning as in the proofs of Lemma C.3 and Theorem 4.2. \square

C.3 Examples of derivations of order- $(n - 1)$ fixpoint equations

Example C.1 Recall the following PHORS considered in Example 4.5.

$$S z = F H z \quad H x = x \oplus_{\frac{1}{2}} \Omega \quad F g z = (g z) \oplus_{\frac{1}{2}} (F(D g) z) \quad D g x = g(g x).$$

Consider the body $F H z$ of S . F and H are translated as follows.

$$\begin{aligned} \mathcal{N}; z \vdash F : (\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ &\rightsquigarrow (F_0, F_1, F_2, F_2) \\ \mathcal{N}; z \vdash H : \circ \rightarrow \circ &\rightsquigarrow (H_0, H_1, H_2, H_2) \end{aligned}$$

By applying TR-APP, we obtain:

$$\mathcal{N}; z \vdash F H : \circ \rightarrow \circ \rightsquigarrow (F_0(H_0, H_1, H_2), F_1(H_1, H_2), F_2(H_2, H_1, H_2), F_2(H_2, H_1, H_2)).$$

Using TR-GVAR, z can be translated as follows.

$$\mathcal{N}; z \vdash z : \circ \rightsquigarrow (0, 1, 0).$$

Thus, by applying TR-APPG, we obtain:

$$\begin{aligned} \mathcal{N}; z \vdash F H z : \circ &\rightsquigarrow \\ (F_0(H_0, H_1, H_2) + F_1(H_1, H_2) \cdot 0, &F_2(H_2, H_1, H_2) + F_1(H_1, H_2) \cdot 1, F_2(H_2, H_1, H_2) + F_1(H_1, H_2) \cdot 0). \end{aligned}$$

By simplifying the output, we obtain:

$$\mathcal{N}; z \vdash F H z : \circ \rightsquigarrow (F_0(H_0, H_1, H_2), F_2(H_2, H_1, H_2) + F_1(H_1, H_2), F_2(H_2, H_1, H_2)).$$

Let us also consider the translation of $F(D g) z$, which occurs on the righthand side of the definition of F . D and g are translated as follows.

$$\begin{aligned} \mathcal{N}, g : \circ \rightarrow \circ; z \vdash D : (\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ &\rightsquigarrow (D_0, D_1, D_2, D_2) \\ \mathcal{N}, g : \circ \rightarrow \circ; z \vdash g : \circ \rightarrow \circ &\rightsquigarrow (g_0, g_1, g_2, g_2) \end{aligned}$$

By using TR-APP, we obtain:

$$\mathcal{N}, g : \circ \rightarrow \circ; z \vdash D g : \circ \rightarrow \circ \rightsquigarrow (D_0(g_0, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2), D_2(g_2, g_1, g_2))$$

By using TR-APP and

$$\mathcal{N}, g : \circ \rightarrow \circ; z \vdash F : (\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ \rightsquigarrow (F_0, F_1, F_2, F_2),$$

we obtain

$$\begin{aligned} \mathcal{N}, g : \circ \rightarrow \circ; z \vdash F(D g) : \circ \rightarrow \circ &\rightsquigarrow \\ (F_0(D_0(g_0, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)), &F_1(D_1(g_1, g_2), D_2(g_2, g_1, g_2)), \\ F_2(D_2(g_2, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)), &F_2(D_2(g_2, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2))) \end{aligned}$$

By using TR-APPG and

$$\mathcal{N}, g : \circ \rightarrow \circ; z \vdash z : \circ \rightsquigarrow (0, 1, 0),$$

we obtain:

$$\begin{aligned} \mathcal{N}, g : \circ \rightarrow \circ; z \vdash F(D g) z : \circ \rightarrow \circ &\rightsquigarrow \\ (F_0(D_0(g_0, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)) + &F_1(D_1(g_1, g_2), D_2(g_2, g_1, g_2)) \cdot 0, \\ F_2(D_2(g_2, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)) + &F_1(D_1(g_1, g_2), D_2(g_2, g_1, g_2)) \cdot 1, \\ F_2(D_2(g_2, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)) + &F_1(D_1(g_1, g_2), D_2(g_2, g_1, g_2)) \cdot 0). \end{aligned}$$

It can be simplified to:

$$\begin{aligned} \mathcal{N}, g : \circ \rightarrow \circ; z \vdash F(D g) z : \circ \rightarrow \circ &\rightsquigarrow \\ (F_0(D_0(g_0, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)), & \\ F_2(D_2(g_2, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2)) + &F_1(D_1(g_1, g_2), D_2(g_2, g_1, g_2)), \\ F_2(D_2(g_2, g_1, g_2), D_1(g_1, g_2), D_2(g_2, g_1, g_2))) & \end{aligned}$$

□

D Additional materials for Section 5

Proof of Proposition 5.3. By the monotonicity of \mathcal{F} and $\widehat{\mathcal{F}}$, we have: $\perp_C \sqsubseteq_C \mathcal{F}(\perp_C) \sqsubseteq_C \mathcal{F}^2(\perp_C) \sqsubseteq_C \dots$ and $\perp_A \sqsubseteq_A \widehat{\mathcal{F}}(\perp_A) \sqsubseteq_A \widehat{\mathcal{F}}^2(\perp_A) \sqsubseteq_A \dots$; hence both $\bigsqcup_C \{\mathcal{F}^i(\perp_C) \mid i \in \omega\}$ and $\bigsqcup_A \{\widehat{\mathcal{F}}^i(\perp_A) \mid i \in \omega\}$ 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 $\mathcal{F}^i(\perp_C) \sqsubseteq_C \gamma(\widehat{\mathcal{F}}^i(\perp_C))$. The proof proceeds by induction on i . The base case $i = 0$ is trivial. If $i > 0$, we have:

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

□