On Probabilistic Program Equivalence and Refinement*

Andrzej S. Murawski and Joël Ouaknine

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Abstract. We study notions of *equivalence* and *refinement* for probabilistic programs formalized in the second-order fragment of Probabilistic Idealized Algol. Probabilistic programs implement randomized algorithms: a given input yields a probability distribution on the set of possible outputs. Intuitively, two programs are equivalent if they give rise to identical distributions for all inputs. We show that equivalence is decidable by studying the fully abstract game semantics of probabilistic programs and relating it to probabilistic finite automata. For terms in β-normal form our decision procedure runs in time exponential in the syntactic size of programs; it is moreover fully compositional in that it can handle *open* programs (probabilistic modules with unspecified components).

In contrast, we show that the natural notion of program refinement, in which the input-output distributions of one program uniformly dominate those of the other program, is undecidable.

1 Introduction

Ever since Michael Rabin's seminal paper on probabilistic algorithms [1], it has been widely recognized that introducing randomization in the design of algorithms can yield substantial improvements in time and space complexity. There are by now dozens of randomized algorithms solving a wide range of problems much more efficiently than their 'deterministic' counterparts—see [2] for a good textbook survey of the field.

Unfortunately, these advantages are not without a price. Randomized algorithms can be rather subtle and tricky to understand, let alone prove correct. Moreover, the very notion of 'correctness' slips from the Boolean to the probabilistic. Indeed, whereas traditional deterministic algorithms associate to each input a given output, randomized algorithms yield for each input a *probabilistic distribution* on the set of possible outputs.

The main focus of this paper is the study of *probabilistic equivalence*. Intuitively, two algorithms are equivalent if they give rise to identical distributions for all inputs. To this end, we introduce (second-order) Probabilistic Idealized

* Work supported by the UK EPSRC (GR/R88861/01) and St John's College, Oxford.

Algol, a programming language which extends Idealized Algol[1] by allowing *(fair) coin-tossing* as a valid expression. It can be shown that, in the presence of loop constructs, this notion of randomization is as powerful as any other 'reasonable' one. Any randomized algorithm can therefore be coded in Probabilistic Idealized Algol. An important consequence of our work is to enable the automated comparison of different randomized algorithms against each other.

We study program equivalence through fully abstract game models, and obtain EXPTIME decidability by recasting the problem in terms of probabilistic automata. We also investigate *program refinement*, a notion intended to capture the relationship that an implementation should enjoy with respect to its specification, and prove undecidability. Our paper continues the algorithmic line of research in game semantics initiated in [4, 5], which aims to exploit the fully abstract character of the game models. These provide exact accounts of extensional behaviour and lead to models of programs that are much different from the traditional approaches to program verification. Their distinctive feature is the absence of explicit reference to state (state manipulations are hidden) which leads to precise and compact summaries of observable program behaviour.

Related work. Much previous work in probabilistic program verification has focussed on *probabilistic model checking*, whereby a probabilistic program is checked against a probabilistic temporal logic specification—see, e.g., [6, 7] and references within. Probabilistic behavioural equivalences have also been studied in the context of process algebra, both from an operational (e.g., [8]) and a denotational (e.g., [9]) perspective.

2 Probabilistic Idealized Algol

The subject of this paper is the finitary version $\mathsf{PA_f}$ of Probabilistic Idealized Algol. Its types θ are generated by the grammar

$$\beta ::= com \,|\, exp \,|\, var \qquad \theta ::= \beta \,|\, \theta \to \theta$$

where β stands for base types: *com* is the command type, *exp* is the *finite* type of expressions ($exp = \{\, 0, \cdots, max \,\}$ for $max > 0$), *var* is the type of assignable variables in which one can store values of type *exp*. The order of a type, written $\mathsf{ord}(\theta)$ is defined by: $\mathsf{ord}(\beta) = 0$, $\mathsf{ord}(\theta \to \theta') = \max(\mathsf{ord}(\theta) + 1, \mathsf{ord}(\theta'))$. $\mathsf{PA_f}$ enables probabilistic functional and imperative programming. Recursion is allowed only in the strictly restricted form of **while**-loops. The complete syntax is shown in Figure 1. We will say that a term-in-context $\Gamma \vdash M : \theta$ is of order i provided $\mathsf{ord}(\theta) \leq i$ and identifiers from Γ have types of order strictly less than i. The big-step operational semantics is defined for terms $\Gamma \vdash M : \theta$, where $\Gamma =$

[1] Devised by Reynolds [3], Idealized Algol augments a Pascal-like procedural language with functional programming constructs. We consider a finitary version in which variables range over a bounded set of integers, terms are parameterized by the allowable higher-order types of their free identifiers, and neither recursion nor pointers are allowed.

$$\frac{}{\Gamma \vdash skip : com} \qquad \frac{i \in \{0, \cdots, max\}}{\Gamma \vdash i : exp} \qquad \frac{}{\Gamma \vdash coin : exp}$$

$$\frac{}{\Gamma, x : \theta \vdash x : \theta} \qquad \frac{\Gamma \vdash M : exp}{\Gamma \vdash \mathbf{succ}(M) : exp} \qquad \frac{\Gamma \vdash M : exp}{\Gamma \vdash \mathbf{pred}(M) : exp}$$

$$\frac{\Gamma \vdash M : exp \quad \Gamma \vdash N_0 : \beta \quad \Gamma \vdash N_1 : \beta}{\Gamma \vdash \mathbf{ifzero}\, M\, N_0\, N_1 : \beta}$$

$$\frac{\Gamma \vdash M : com \quad \Gamma \vdash N : \beta}{\Gamma \vdash M; N : \beta} \qquad \frac{\Gamma \vdash M : exp \quad \Gamma \vdash N : com}{\Gamma \vdash \mathbf{while}\ M\ \mathbf{do}\ N : com}$$

$$\frac{\Gamma \vdash M : var}{\Gamma \vdash !M : exp} \qquad \frac{\Gamma \vdash M : var \quad \Gamma \vdash N : exp}{\Gamma \vdash M := N : com}$$

$$\frac{\Gamma, X : var \vdash M : com, exp}{\Gamma \vdash \mathbf{new}\ X\ \mathbf{in}\ M : com, exp}$$

$$\frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x^{\theta}.M : \theta \to \theta'} \qquad \frac{\Gamma \vdash M : \theta \to \theta' \quad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'}$$

Fig. 1. Syntax of $\mathsf{PA_f}$.

$x_1 : var, \cdots, x_n : var$, through judgments of the shape $(s, M) \Downarrow^p (s', V)$, where s, s' are functions from $\{x_1, \cdots, x_n\}$ to $\{0, \cdots, max\}$. Whenever an evaluation tree ends in $(s, M) \Downarrow^p (s', V)$ we can interpret that as "the associated evaluation of M from state s has probability p". Because of *coin*, M may have countably many evaluations from a given s. We shall write $(s, M) \downarrow^p V$ iff $p = \sum p_i$ and the sum ranges over all evaluations of the form $(s, M) \Downarrow^{p_i} (s', V)$ (for some s'). If there are no such reductions, we simply have $(s, M) \downarrow^0 V$. The judgment $(s, M) \downarrow^p V$ thus denotes the fact that the probability of evaluating M in state s to V is p. When M is closed we write $M \downarrow^p V$, because s is then trivial. For instance, we have $coin \downarrow^{0.5} 0$ and $coin \downarrow^{0.5} 1$.

We can now define the induced notion of contextual equivalence: the terms-in-context $\Gamma \vdash M_1 : \theta$ and $\Gamma \vdash M_2 : \theta$ are *equivalent* (written $\Gamma \vdash M_1 \cong M_2$) if for all contexts $\mathcal{C}[-]$ such that $\vdash \mathcal{C}[M_1], \mathcal{C}[M_2] : com$ we have $\mathcal{C}[M_1] \downarrow^p skip$ if and only if $\mathcal{C}[M_2] \downarrow^p skip$. Danos and Harmer gave a fully abstract game model for $\cong$ in [10], which we review in the following section.

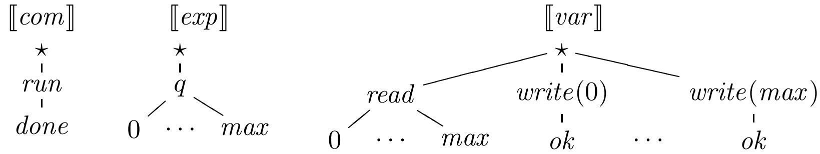
As stated in the Introduction, randomized algorithms can readily be coded in Probabilistic Idealized Algol. Under mild conditions[2], contextual equivalence then precisely corresponds to the natural notion of equivalence for randomized algorithms: identical input/output distributions.

3 Probabilistic Game Semantics

The games needed to interpret probabilistic programs are played in arenas as in the sequential case. An arena A is a triple $\langle M_A, \lambda_A, \vdash_A \rangle$, where M_A is the set of

[2] Essentially syntactic restrictions aimed at preventing undesirable side-effects.

moves, $\lambda_A : M_A \to \{\, \mathrm{O}, \mathrm{P} \,\} \times \{\, Q, A \,\}$ indicates to which of the two players (O or P) each move belongs and whether it is a question- or an answer-move, and $\vdash_A$ is the so-called *enabling* relation between $\{\, \star \,\} + M_A$ and M_A. $\vdash_A$ is required to satisfy a number of technical conditions: for instance, moves enabled by $\star$, called *initial* and collected in the set called I_A, must be O-questions and whenever one move enables another they have to belong to different players. Here are the arenas used to interpret base types:



The moves $run, q, read$ and $write(i)$ $(0 \leq i \leq max)$ are initial O-questions, the rest are P-answers enabled by the respective O-questions.

Arenas can be combined to form *product* and *arrow* arenas as follows[3].

$$\begin{aligned} M_{A\times B} &= M_A + M_B \\ \lambda_{A\times B} &= [\lambda_A, \lambda_B] \\ \vdash_{A\times B} &= \vdash_A + \vdash_B \end{aligned}$$

$$\begin{aligned} M_{A\Rightarrow B} &= M_A + M_B \\ \lambda_{A\Rightarrow B} &= [\overline{\lambda}_A, \lambda_B] \\ \vdash_{A\Rightarrow B} &= (\vdash_A \cap (M_A \times M_A)) \ + \ (I_B \times I_A) + \ \vdash_B \end{aligned}$$

The allowable exchanges of moves in an arena A are *justified sequences*, which are sequences of moves of A such that each occurrence of a non-initial move n is equipped with a pointer to an earlier move m such that $m \vdash_A n$. In order for a justified sequence to become a *play*, the moves must alternate between the two players (O necessarily begins then) and the standard conditions of visibility and bracketing must be satisfied [11, 12]. The set of plays over A will be denoted by $\mathcal{L}_A$, that of odd- and even-length ones by $\mathcal{L}_A^{\mathsf{odd}}$ and $\mathcal{L}_A^{\mathsf{even}}$ respectively.

The notion of a probabilistic strategy makes a departure from sequential game semantics. A strategy σ is a function $\sigma : \mathcal{L}_A^{\mathsf{even}} \to [0,1]$ such that $\sigma(\varepsilon) = 1$ and for any $s \in \mathcal{L}_A^{\mathsf{even}}, sa \in \mathcal{L}_A^{\mathsf{odd}}$ we have $\sigma(s) \geq \sum_{\{\, sab \in \mathcal{L}_A^{\mathsf{even}} \,\}} \sigma(sab)$. Let $\mathcal{T}_\sigma$ be the set of all even-length plays s such that $\sigma(s) > 0$. In the following they will be called the *traces* of σ. To interpret *coin* one takes the strategy $\sigma : [\![exp]\!]$ such that $\sigma(qi_1 \cdots qi_n) = (1/2)^n$ where $i_1, \cdots, i_n \in \{\, 0, 1 \,\}$ and $\sigma(s) = 0$ otherwise.

The above definition of a strategy describes the global probabilistic behaviour. Providing $s \in \mathcal{T}_\sigma$ and $sa \in \mathcal{L}_A^{\mathsf{odd}}$ the conditional (local) probability of sab given sa can be calculated by taking $\sigma(sab)/\sigma(s)$.

Probabilistic strategies are composed by considering interaction sequences between them. Suppose u is a sequence of moves from arenas A, B, C together with unique pointers from all moves except those initial in C. We define $u \restriction B, C$

[3] $\overline{\lambda}_A$ works like λ_A except that it reverses the ownership of moves.

by deleting from u all moves from A together with associated pointers. $u \upharpoonright A, B$ is defined in a similar way. $u \upharpoonright A, C$ works analogously except that whenever there was a pointer from an A-move m_A to a B-move m_B and a pointer from m_B to a C-move m_C, we introduce a pointer from m_A to m_C. u is called an *interaction sequence* of A, B, C provided $u \upharpoonright A, B \in \mathcal{L}_{A \Rightarrow B}$ and $u \upharpoonright B, C \in \mathcal{L}_{B \Rightarrow C}$. The set of interaction sequences is then denoted by $\mathcal{I}(A, B, C)$. If $s \in \mathcal{L}^{\mathsf{even}}_{A \Rightarrow C}$, the set of B-witnesses of s, written $\mathsf{wit}_B(s)$, is defined to be $\{ u \in \mathcal{I}(A, B, C) \,|\, u \upharpoonright A, C = s \}$. Finally, given $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$, one defines $\sigma; \tau : A \Rightarrow C$ by

$$(\sigma; \tau)(s) = \sum_{u \in \mathsf{wit}_B(s)} \sigma(u \upharpoonright A, B) \cdot \tau(u \upharpoonright B, C).$$

4 Inside the Game Model

In general, plays may contain several occurrences of initial moves. Such occurrences define *threads* within the play in the following way: two moves are in the same thread if they are connected via chains of pointers to the same occurrence of an initial move. Plays that contain just one occurrence of an initial move (and consequently consist of one thread only) are called *well-opened* (we write $\mathcal{L}^{\mathsf{wo}}_A$ for the set containing them)[4].

Because plays satisfy the visibility condition, responses by P are always in the same thread as the preceding O-move, a condition known as *well-threadedness.* A stricter class of *single-threaded* strategies arises when one requires that P-responses depend on the thread of play of the preceding O-move. This behaviour is formally captured by the notion of a comonoid homomorphism in [10] and can be summed up in two points. Firstly, the threads occurring in a trace are also traces, so we can regard traces as interleavings of well-opened ones (such that only O can switch between threads). Secondly, whenever a trace s is an interleaving of well-opened $s_1, \cdots, s_n$, we have $\sigma(s) = \sigma(s_1) \cdot \ldots \cdot \sigma(s_n)$. As shown in [10], arenas and single-threaded strategies, quotiented by the *intrinsic preorder*, constitute a fully abstract model for $\cong$. Next we give a more direct full abstraction result based on comparing probabilities in special kinds of plays rather than quotienting them.

Our analysis will also capture a notion of probabilistic refinement defined as follows: $\Gamma \vdash M_1 : \theta$ *refines* $\Gamma \vdash M_2 : \theta$, written $\Gamma \vdash M_1 \sqsubseteq M_2$, iff for all contexts $\mathcal{C}[-]$ such that $\vdash \mathcal{C}[M_1], \mathcal{C}[M_2] : com$ if $\mathcal{C}[M_1] \Downarrow^{p_1} skip$ then $\mathcal{C}[M_2] \Downarrow^{p_2} skip$ and $p_1 \leq p_2$. Note that $M_1 \cong M_2$ iff $M_1 \sqsubseteq M_2$ and $M_2 \sqsubseteq M_1$. We are going to show that, like in the sequential second-order case [4], $\cong$ is decidable. However, in contrast, $\sqsubseteq$ will turn out undecidable.

[4] Equivalently, one can present the model in the style of [12], where terms induce well-opened sequences only and the interpretation of function spaces adheres to the decomposition $A \Rightarrow B = {!A} \multimap B$. In this paper we stick to the presentation of [10], where the induced plays do not have to be well-opened, but the strategies are restricted to be single-threaded.

Definition 1. *A play* $s \in \mathcal{L}_A^{\mathsf{even}}$ *is* **complete** *iff all questions in* s *are answered. The set of complete plays of* A *is denoted by* $\mathcal{L}_A^{\mathsf{comp}}$*. Suppose* $\sigma_1, \sigma_2 : A$ *are single-threaded strategies. We then define:*

$$\sigma_1 \leq_\pi \sigma_2 \stackrel{\text{def}}{\iff} \sigma_1(s) \leq \sigma_2(s) \textit{ for all } s \in \mathcal{L}_A^{\mathsf{wo}} \cap \mathcal{L}_A^{\mathsf{comp}}.$$

Lemma 1. *Let* $\vdash M_1, M_2 : \theta$ *and* $\sigma_i = [\![\vdash M_i]\!]$ $(i = 1, 2)$*. Then* $\vdash M_1 \sqsubseteq M_2$ *if and only if* $\sigma_1 \leq_\pi \sigma_2$*.*

Proof. Suppose $\vdash M_1 \sqsubseteq M_2$. Let $s \in \mathcal{L}_{[\![\theta]\!]}^{\mathsf{wo}} \cap \mathcal{L}_{[\![\theta]\!]}^{\mathsf{comp}}$ and $\sigma_1(s) = p_1$. By the definability result for (sequential) Idealized Algol in [12] there exists a *deterministic* context $x : \theta \vdash \mathcal{C}[x] : com$ such that the only well-opened complete play in $[\![x : \theta \vdash \mathcal{C}[x] : com]\!]_{\mathsf{IA}}$ is *run s done*. Then in the probabilistic game model we have $[\![x : \theta \vdash \mathcal{C}[x] : com]\!](run\, s\, done) = 1$ and $[\![x : \theta \vdash \mathcal{C}[x] : com]\!](run\, s'\, done) = 0$ whenever $s \neq s'$. So:

$$[\![\vdash \mathcal{C}[M_1] : com]\!](run\ done) = (\sigma_1; [\![x : \theta \vdash \mathcal{C}[x] : com]\!])(run\ done) = p_1.$$

By adequacy (Theorem 3.2 in [10]) $\mathcal{C}[M_1] \Downarrow^{p_1} skip$. Because $\vdash M_1 \sqsubseteq M_2$ we have $\mathcal{C}[M_2] \Downarrow^{p_2} skip$ and $p_1 \leq p_2$. By soundness for evaluation (Theorem 3.2 in [10]) $\sigma_2(run\ done) = p_2$. Because $[\![\vdash \mathcal{C}[M_2]]\!] = \sigma_2; [\![x : \theta \vdash \mathcal{C}[x] : com]\!]$ and $\mathcal{C}$ is deterministic, we have $\sigma_2(s) = p_2$. Hence, $\sigma_1 \leq_\pi \sigma_2$.

Now we prove the contrapositive of the converse. Suppose we do not have $\vdash M_1 \sqsubseteq M_2$, i.e. there exists a context $\mathcal{C}[-]$ such that $\vdash \mathcal{C}[M_i] : com$, $\mathcal{C}[M_i] \Downarrow^{p_i} skip$ for $i = 1, 2$ and $p_1 > p_2$. By soundness, compositionality and the composition formula we have

$$\begin{aligned} p_i &= [\![\vdash \mathcal{C}[M_i]]\!](run\ done) = (\sigma_i; [\![x : \theta \vdash \mathcal{C}[x] : com]\!])(run\ done) = \\ &= \sum_{u \in \mathsf{wit}_{[\![\theta]\!]}(run\ done)} \sigma_i(u \upharpoonright 1, [\![\theta]\!]) \cdot [\![x : \theta \vdash \mathcal{C}[x] : com]\!](u \upharpoonright [\![\theta]\!], [\![com]\!]). \end{aligned}$$

Since $p_1 > p_2$, there must exist $u \in \mathsf{wit}_{[\![\theta]\!]}(run\ done)$ such that $\sigma_1(u \upharpoonright 1, [\![\theta]\!]) > \sigma_2(u \upharpoonright 1, [\![\theta]\!])$. Let $s = u \upharpoonright 1, [\![\theta]\!]$. Because *run done* is complete, so is s. In general s might be an interleaving of several well-opened complete plays, let us call them $s_1, \cdots, s_k$. Because σ_i is single-threaded, we then have $\sigma_i(s) = \prod_{j=1}^{k} \sigma_i(s_j)$ and, further, because $\sigma_1(s) > \sigma_2(s)$, we must have $\sigma_1(s_j) > \sigma_2(s_j)$ for some j. Consequently, $\sigma_1 \not\leq_\pi \sigma_2$. ☐

The Lemma generalizes to open terms (by observing that $\Gamma \vdash M_1 \sqsubseteq M_2$ is equivalent to $\vdash \lambda\Gamma.M_1 \sqsubseteq \lambda\Gamma.M_2$) and implies the following result for program equivalence. Given single-threaded strategies $\sigma_1, \sigma_2 : A$ we write $\sigma_1 =_\pi \sigma_2$ iff $\sigma_1(s) = \sigma_2(s)$ for all $s \in \mathcal{L}_A^{\mathsf{wo}} \cap \mathcal{L}_A^{\mathsf{comp}}$.

Lemma 2. *Let* $\vdash M_1, M_2 : \theta$ *and* $\sigma_i = [\![\vdash M_i]\!]$ $(i = 1, 2)$*. Then* $\vdash M_1 \cong M_2$ *if and only if* $\sigma_1 =_\pi \sigma_2$*.*

5 Probabilistic Automata

Probabilistic automata (PA) generalize finite automata in that probability distributions are imposed on transitions. Their first definition goes back to Rabin's work in the 1960s [13] (see Paz [14] for a textbook treatment). Various modifications of the automata have recently reappeared in research on probabilistic systems (see e.g. [15, 16]).

Let X be a finite set. A *subprobability* over X is a function $\omega : X \to [0,1]$ such that $\sum_{x\in X} \omega(x) \leq 1$. A *probabilistic distribution* over X is a subprobability such that $\sum_{x\in X} \omega(x) = 1$. We write $\mathcal{S}(X), \mathcal{P}(X)$ respectively for the sets of all subprobabilities and probabilistic distributions over X.

Definition 2. *A* **probabilistic automaton** *is a tuple* $\mathcal{A} = \langle Q, \Sigma, i, F, \delta \rangle$, *where Q is a finite set of states, Σ is the alphabet, $i \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and δ is the transition function, which can take one of the following two shapes: either $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ or $\delta : Q \to \mathcal{P}(\Sigma \times Q)$. In the former case $\mathcal{A}$ is called* **reactive**, *in the latter* **generative**.

The different shapes of the transition function reflect the typical scenarios which the two kinds of automata are used to model. Reactive automata describe probabilistic reactions to given symbols, while for generative ones the symbol is viewed as part of the probabilistic response. The automata we are going to use to model probabilistic programs will turn out to combine the features of the two. O-moves will adhere to a restricted form of the reactive framework, whereas P-moves will be generative.

We refer to transitions by writing $q \xrightarrow{x,p} q'$ whenever $\delta(q,x)(q') = p$ (the reactive case) or $\delta(q)(x,q') = p$ (the generative case). A run r of a probabilistic automaton is a sequence of transitions

$$i \xrightarrow{x_1,p_1} q_1 \xrightarrow{x_2,p_2} \dots \xrightarrow{x_k,p_k} q_k.$$

Let us write $P_{\mathcal{A}}(r)$ for the probability of the run r, i.e. $P_{\mathcal{A}}(r) = \prod_{i=1}^{k} p_i$. The word associated with the run r will be denoted by $W_{\mathcal{A}}(r)$, i.e. $W_{\mathcal{A}}(r) = x_1 \cdots x_k$. A run is accepting if $q_k \in F$. Let $Acc_{\mathcal{A}}(w)$ be the set of accepting runs r such that $W_{\mathcal{A}}(r) = w$. For a given automaton $\mathcal{A}$ we define a function $\mathcal{A} : \Sigma^\star \to [0,1]$ as follows

$$\mathcal{A}(w) = \sum_{r \in Acc_{\mathcal{A}}(w)} P_{\mathcal{A}}(r).$$

$\mathcal{A}(w)$ denotes the probability that the automaton reaches a final state and reads the string w.

¿From now on we restrict our attention to automata in which the probabilities associated with transitions are rational numbers. We shall also often consider automata where the requisite distributions are in fact only subprobabilities on the understanding that they can be extended to probability distributions by adding a dummy "sink" state and dummy transitions. In this sense generative automata can be considered special cases of reactive ones. Note also that a reactive automaton $\mathcal{A}$ can be converted to a generative one, which we denote $\mathcal{A}/|\Sigma|$, by dividing all probabilities occurring in transitions by the size of the alphabet. We introduce the following two decision problems.

Definition 3. *Suppose $\mathcal{A}_1, \mathcal{A}_2$ are probabilistic automata of the same kind.*

- Equivalence*: $\mathcal{A}_1(w) = \mathcal{A}_2(w)$ for all $w \in \Sigma^*$.*
- Refinement*: $\mathcal{A}_1(w) \leq \mathcal{A}_2(w)$ for all $w \in \Sigma^*$.*

Equivalence for reactive automata was already considered by Paz in [14] (in a slightly different setting) and shown decidable. His proof relies on the observation that in order to prove two automata equivalent it suffices to verify equivalence for strings of length $n_1 + n_2 - 1$, where n_1, n_2 are the respective numbers of states. This leads to an NP algorithm. Paz's result was subsequently refined by Tzeng [17], who presented a Ptime algorithm based on a search for a basis in a vector space. Note that the decidability of Equivalence for reactive automata implies decidability for generative automata as well.

In contrast, we next show that Refinement is undecidable by reducing the following problem to it:

> Nonemptiness with threshold: Given a reactive automaton $\mathcal{A}$ and a rational $0 \leq \lambda \leq 1$, there exists $w \in \Sigma^*$ such that $\mathcal{A}(w) > \lambda$.

Nonemptiness with threshold was introduced by Rabin [13] and proved undecidable by Paz [14]. More recently, Blondel and Canterini [18] gave a more elementary proof based on Post's Correspondence Problem. Observe that the complement of Nonemptiness with threshold reduces to Refinement (for reactive automata) by considering the automaton $\mathcal{A} = \langle \{ i, f \}, \Sigma, i, \{ f \}, \delta \rangle$ below which accepts every non-empty word with probability λ.

$$i \xrightarrow{\Sigma,\lambda} f \circlearrowright \Sigma,1$$

Then we have $\mathcal{A}(w) = \lambda$ for any $w \in \Sigma^+$, which implies undecidability of Refinement in the reactive case.

The undecidability carries over to the generative case, because the refinement of $\mathcal{A}_1$ by $\mathcal{A}_2$, where both are reactive, is equivalent to the refinement of $\mathcal{A}_1/|\Sigma|$ by $\mathcal{A}_2/|\Sigma|$ (both generative). Furthermore, refinement remains undecidable for pairs of generative automata in which all probabilities in $\mathcal{A}_1$ and $\mathcal{A}_2$ are of the form $m/2^n$. To see this, observe that refinement of $\mathcal{A}_1$ by $\mathcal{A}_2$ is equivalent to that of $v\mathcal{A}_1$ by $v\mathcal{A}_2$, where $v \in (0, 1]$ and $v\mathcal{A}$ is obtained from $\mathcal{A}$ by multiplying all probabilities on transitions by v. Now given $\mathcal{A}_1$ and $\mathcal{A}_2$, choose $v = d/2^n$, where d is the least common denominator of all the probabilities appearing in both $\mathcal{A}_1$ and $\mathcal{A}_2$, and 2^n is the smallest power of two that exceeds d. Every weight in $v\mathcal{A}_1$ and $v\mathcal{A}_2$ is now of the form $m/2^n$ for some integer m. We are interested in restricting the probabilities to this form, because later on we are going to simulate generative automata in $\mathsf{PA_f}$. Although it was shown in [10] that the strategy representing *coin* is universal, the proof relied on infinitary features such as infinite datatypes and recursion. By contrast, distributions based on probabilities of the shape $m/2^n$ can be simulated in a small fragment of $\mathsf{PA_f}$ that does not even require **while**.

Another problem from the theory of probabilistic automata that will be useful in our work concerns ε-transitions. Note that the definitions of probabilistic automata (as well as those of $P_{\mathcal{A}}(r), W_{\mathcal{A}}(r)$ and $\mathcal{A}(w)$) can easily be extended to encompass ε-transitions. Then it is natural to ask whether and how ε-transitions can be removed in such a way as to yield an equivalent automaton. This problem was investigated by Mohri in the general setting of weighted automata, where the weights come from a variety of semirings [19, 20]. The probabilistic case then falls into the case of *closed* semirings, which require a special approach based on a decomposition into strongly-connected components and a generalization of the Floyd-Warshall all-pairs shortest-path algorithm. This decomposition is designed to handle the problematic ε-cycles, as it is easier to deal with them in a strongly-connected component. A consequence of the algorithm is the fact that rationality of weights is preserved after ε-removal, which should be contrasted with the general failure of compositionality for rational strategies pointed out in [10].

6 Second-order Program Equivalence is Decidable

Recall that a $\mathsf{PA_f}$ term $\Gamma \vdash M : \theta$ is a second-order term if $\mathsf{ord}(\theta) \leq 2$ and the type of each identifier from Γ is either a base type or a first-order type. We show that program equivalence for second-order $\mathsf{PA_f}$ terms is decidable. The argument will be based on a reduction to EQUIVALENCE for reactive automata via Lemma 2. More precisely, we show that for any second-order $\mathsf{PA_f}$ term there exists a reactive automaton which accepts (the sequences of moves that occur in) well-opened complete plays with the same probabilities as those assigned to them by the corresponding probabilistic strategy. As we are interested in the second-order terms only, it is not necessary to represent pointers, because they can be uniquely reconstructed [4]. Consequently, we ignore them completely in what follows.

The automata corresponding to programs will be special instances of reactive automata. Their sets of states will be partitioned into O-states and P-states: only transitions on O-moves (respectively P-moves) will be available from O-states (respectively P-states). Moreover, at O-states, there can only be at most one transition for a given input letter (its probability is then 1). For P-states, however, the probabilities of all outgoing transitions will have to add up to at most 1, which is consistent with the generative framework. This pattern of behaviour is captured by the definition below (M_A^O and M_A^P stand for the sets of O-moves and P-moves respectively).

Definition 4. *An A-automaton is a tuple $\mathcal{A} = \langle Q, M_A, i, f, \delta \rangle$, where:*

1. *$Q = Q^O + Q^P$ is the set of states (elements of Q^O, Q^P are called O-states and P-states respectively);*
2. *i has no incoming transitions, f has no outgoing transitions;*

3. $\{ i, f \} \subseteq Q^O$;
4. $\delta = \delta^O + \delta^P$, *where* $\delta^O : Q^O \times M_A^O \rightharpoonup Q^P$ *(*$\delta(q,m) = q'$ *is taken to mean* $\delta(q,m)(q') = 1$*) and* $\delta^P : Q^P \rightharpoonup \mathcal{S}(M_A^P \times Q^O)$;
5. *sequences of moves determined by runs of* $\mathcal{A}$ *are plays of* A*, accepting runs are complete positions.*

Example 1. coin : *exp* will be interpreted by the automaton $i \xrightarrow{q,1} \circ \overset{0,\frac{1}{2}}{\underset{1,\frac{1}{2}}{\rightrightarrows}} f$.

Definition 5. *A* $[\![\Gamma \vdash \theta]\!]$*-automaton* $\mathcal{A}$ *represents* $\Gamma \vdash M : \theta$ *iff*

$$\mathcal{A}(w) = \begin{cases} [\![\Gamma \vdash M : \theta]\!](w) & w \in \mathcal{L}^{\mathsf{wo}}_{[\![\Gamma \vdash \theta]\!]} \cap \mathcal{L}^{\mathsf{comp}}_{[\![\Gamma \vdash \theta]\!]} \\ 0 & \text{otherwise} \end{cases}$$

In the rest of this section we set out to prove that any second-order $\mathsf{PA_f}$ term is represented by an automaton as specified in the definition above. It suffices to prove that this is the case for terms in β-normal form, since β-equivalent terms are also $\cong$-equivalent (the fully abstract model [10] is a cartesian-closed category). The most difficult stage in the construction is the interpretation of the application rule

$$\frac{\Gamma \vdash M : \theta \to \theta' \quad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'}$$

which will be split into two simpler rules, multiplicative application and contraction respectively:

$$\frac{\Gamma \vdash M : \theta \to \theta' \quad \Delta \vdash N : \theta}{\Gamma, \Delta \vdash MN : \theta'} \qquad \frac{\Gamma, x : \theta, y : \theta \vdash M : \theta'}{\Gamma, x : \theta \vdash M[x/y] : \theta'}.$$

The former is simply interpreted by composition, because (up to currying) $[\![\Gamma, \Delta \vdash MN : \theta']\!]$ is equal to $[\![\Delta \vdash N : \theta]\!] ; [\![\, \vdash \lambda x^\theta . \lambda \Gamma . Mx : \theta \to (\Gamma \to \theta')]\!]$. Hence, in order to interpret application we need to be able to handle composition and contraction. Other term constructs (except λ-abstraction) can also be interpreted through application by introducing special constants for each of them [12]. For instance, assignment then corresponds to the constant $(:=) : \mathit{var} \to \mathit{exp} \to \mathit{com}$ so that $((:=)M)N$ corresponds to $M := N$. For local variables one uses $\mathsf{new}_\beta : (\mathit{var} \to \beta) \to \beta$ so that **new** X **in** M is equivalent to $\mathsf{new}_\beta(\lambda X.M)$.

Theorem 1. *For any second-order term* $\Gamma \vdash M : \theta$ *there exists a* $[\![\Gamma \vdash \theta]\!]$*-automaton representing* $\Gamma \vdash M : \theta$*.*

Proof. We construct the automata by induction on the structure of β-normal terms. The base cases are ground-type constants (*coin*, *skip*, i : *exp*), free identifiers (of base type or first-order type) and the constants corresponding to **succ**, **pred**, **ifzero**, ; , **while**, !, := , **new**. The automaton for *coin* was given in Example 1. The shape of the strategies corresponding to other constants and free identifiers is already known from work on sequential Algol (see e.g. [21]). Because the strategies are all deterministic, the probabilistic automata representing them can be obtained by assigning probability 1 to all transitions of the finite automata associated with them. Thus it remains to interpret λ-abstraction and application. The former is trivial in game semantics, because currying amounts to the identity operation (up to associativity of the disjoint sum), so we only need to concentrate on application, i.e. composition and contraction.

Let $\sigma = [\![\Delta \vdash N : \theta]\!]$ and $\tau = [\![x : \theta \vdash \lambda\Gamma.Mx : \Gamma \to \theta']\!]$. Suppose $\mathcal{A}_1 = \langle Q_1, M_{[\![\Delta]\!]} + M_{[\![\theta]\!]}, i_1, f_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, M_{[\![\theta]\!]} + M_{[\![\Gamma]\!]} + M_{[\![\theta']\!]}, i_2, f_2, \delta_2 \rangle$ represent $\Delta \vdash N$ and $x \vdash \lambda\Gamma.Mx$ respectively. Because we consider β-normal terms of second order only, well-opened complete traces of $\sigma;\tau$ can arise only from interaction sequences which involve well-opened complete traces from τ and iterated well-opened complete traces from σ. Thus, as a first step, we have to construct a probabilistic automaton that accepts iterated well-complete traces from σ with the same probabilities as those assigned to them by σ. Since σ is well-threaded, we would like the probability of accepting a complete trace which is not well-opened to be equal to the product of probabilities with which the automaton accepts the constituent well-opened traces. This is achieved simply by identifying f_1 with i_1, because i_1 does not have any incoming transitions and f_1 has no outgoing ones. We write $\mathcal{A}_1^*$ for the automaton obtained from $\mathcal{A}_1$ in this way. Let $A = [\![\Delta]\!]$, $B = [\![\theta]\!]$ and $C = [\![\Gamma \to \theta']\!]$. We define another automaton

$$\mathcal{A}_{||} = \langle Q_{||}, M_A + M_B + M_C, (i_1, i_2), (f_1, f_2), \delta \rangle,$$

where $Q_{||} = (Q_1^O \times Q_2^O) + (Q_1^O \times Q_2^P + Q_1^P \times Q_2^O)$, which will model all the interactions that may result in well-opened complete plays of $\sigma;\tau$. This is done by parallel composition of $\mathcal{A}_1^*$ with $\mathcal{A}_2$ synchronized on moves from B. The function δ is defined by the transitions given below (q_1^O, q_1^P below range over O-states and P-states of $\mathcal{A}_1^*$ respectively; q_2^O, q_2^P are used analogously for $\mathcal{A}_2$):

- for any $q_1^O \in Q_1^O$ and $x \in M_C$

$$\begin{array}{ll} (q_1^O, q_2^O) \xrightarrow{x,1} (q_1^O, q_2^P) & \text{if } q_2^O \xrightarrow{x,1} q_2^P \\ (q_1^O, q_2^P) \xrightarrow{x,p} (q_1^O, q_2^O) & \text{if } q_2^P \xrightarrow{x,p} q_2^O \end{array}$$

- for any $q_2^O \in Q_2^O$ and $x \in M_A$

$$\begin{array}{ll} (q_1^O, q_2^O) \xrightarrow{x,1} (q_1^P, q_2^O) & \text{if } q_1^O \xrightarrow{x,1} q_1^P \\ (q_1^P, q_2^O) \xrightarrow{x,p} (q_1^O, q_2^O) & \text{if } q_1^P \xrightarrow{x,p} q_1^O \end{array}$$

- for any $x \in M_B$

$$\begin{array}{ll} (q_1^O, q_2^P) \xrightarrow{x,p} (q_1^P, q_2^O) & \text{if } q_1^O \xrightarrow{x,1} q_1^P \text{ and } q_2^P \xrightarrow{x,p} q_2^O \\ (q_1^P, q_2^O) \xrightarrow{x,p} (q_1^O, q_2^P) & \text{if } q_1^P \xrightarrow{x,p} q_1^O \text{ and } q_2^O \xrightarrow{x,1} q_2^P. \end{array}$$

By the structure of interaction sequences (as described, for instance, by the state diagram in Figure 2 of [10]) each run of $\mathcal{A}_{||}$ determines an interaction sequence of σ and τ. Moreover, because accepting runs of $\mathcal{A}_1^*$ and $\mathcal{A}_2$ determine complete plays in the respective games (well-opened for $\mathcal{A}_2$), the accepting runs of $\mathcal{A}_{||}$ determine interactions that, when projected onto $M_A + M_C$ yield well-opened complete plays in $A \Rightarrow C$. Note that the probability of an accepting run of $\mathcal{A}_{||}$ is the product of probabilities associated with the corresponding runs of $\mathcal{A}_1^*$ and $\mathcal{A}_2$. Because interaction sequences are uniquely determined by the constituent plays, for any $w \in \mathcal{L}^{\mathsf{wo}}_{[\![\Delta\to(\Gamma\to\theta')]\!]} \cap \mathcal{L}^{\mathsf{comp}}_{[\![\Delta\to(\Gamma\to\theta')]\!]}$ we get

$$\begin{aligned}
\mathcal{A}_{||}(w) &= \sum_{r\in Acc_{\mathcal{A}_{||}}(w)} P_{\mathcal{A}_{||}}(r) \\
&= \sum_{r_1\in Acc_{\mathcal{A}_1^*}(w\restriction A,B)} \; \sum_{r_2\in Acc_{\mathcal{A}_2^*}(w\restriction B,C)} P_{\mathcal{A}_1^*}(w\restriction A,B)\cdot P_{\mathcal{A}_2}(w\restriction B,C) \\
&= \mathcal{A}_1^*(w\restriction A,B)\cdot \mathcal{A}_2(w\restriction B,C) = \sigma(w\restriction A,B)\cdot\tau(w\restriction B,C).
\end{aligned}$$

By the composition formula for $\sigma;\tau$, in order to construct an $A \Rightarrow C$-automaton for $\Delta \vdash \lambda\Gamma.MN$, it now suffices to relabel all transitions on B-moves as ε-transitions and subsequently replace them using Mohri's algorithm [19] (in fact, the full power of the algorithm is needed to handle composition with **while** and **new**, other cases can be easily solved "by hand"). Because B-moves are only available from states in $(Q_1^O\times Q_2^P + Q_1^P\times Q_2^O)$, the automaton after the ε-removal will be an $A \Rightarrow C$-automaton with the partition of states given before.

Finally, we discuss contraction. It is interpreted simply by identifying moves originating from the two contracted copies of θ, i.e. by relabelling. To complete the argument, we only need to show that the transition function retains the shape required in A-automata. This is obvious for P-states but (at least in principle) the relabelling might produce an O-state with two outgoing transitions on the same O-move. Then we claim that the O-state is unreachable and, consequently, can be deleted. Indeed, if it were reachable, there would exist a position s such that the automaton representing the term before contraction would read both $s\,o_1$ and $s\,o_2$, where o_1 and o_2 are O-moves from the two different copies of θ. By 5. both so_1 and so_2 would be plays then, but this is impossible because only one of them can satisfy visibility (since the questions justifying o_1 and o_2 cannot appear in the O-view at the same time). Consequently, contraction can be interpreted in such a way that we get an automaton of the required shape.

By Lemma 2, the Lemma above and the decidability of Equivalence we have:

Theorem 2. $\cong$ *is decidable for second-order* $\mathsf{PA_f}$ *terms.*

Note that the size of the automaton produced in the above proof will be exponential in the size of the β-normal term. Because ε-removal and equivalence testing work in polynomial time, equivalence of β-normal second-order $\mathsf{PA_f}$ terms can be decided in Exptime.

7 Probabilistic Program Refinement Is Undecidable

Probabilistic program refinement at second order will be shown undecidable by reducing REFINEMENT for generative probabilistic automata to probabilistic program refinement. To this end, for each generative automaton where the probabilities are of the form $m/2^n$, we construct a $\mathsf{PA_f}$ term representing it in a way to be described later.

First we discuss how to model the special distributions in $\mathsf{PA_f}$. Let us define a family of terms $\mathsf{choice}_n(M_0, \cdots, M_{2^n-1})$ which evaluate to each of the terms M_i with the same probability $\frac{1}{2^n}$:

$$\begin{aligned}\mathsf{choice}_0(M_0) &= M_0\\ \mathsf{choice}_{n+1}(M_0, \cdots, M_{2^{n+1}-1}) &=\\ &\mathbf{ifzero}\; \mathit{coin}\; (\mathsf{choice}_n(M_0, \cdots, M_{2^n-1}))\, (\mathsf{choice}_n(M_{2^n}, \cdots, M_{2^{n+1}-1})).\end{aligned}$$

Observe that by using the same term M as M_i for several i's we can vary (increase) the probability of $\mathsf{choice}_n(M_0, \cdots, M_{2^n-1})$ being equivalent to M_i. Suppose that, given terms $N_1, \cdots, N_k$, we are to construct another term $\widehat{N}$ which evaluates to N_i $(1 \le i \le k)$ with probability $p_i = \frac{m_i}{2^n}$, where $m_i \in \mathbb{N}$ and $\sum_{i=1}^k p_i = 1$. This can be done by taking $\widehat{N}$ to be $\mathsf{choice}_n(M_0, \cdots, M_{2^n-1})$ where for $M_0, \cdots, M_{2^n-1}$ we take m_i copies of N_i for each $i = 1, \cdots, k$ (the order is irrelevant). Then the probability of N_i being selected is $\frac{m_i}{2^n}$.

In order to complete the encoding of the special generative automata we have to define how strings are interpreted. Suppose $\Sigma = \{x_1, \cdots, x_m\}$. A string $w = x_{j_1} \cdots x_{j_l}$ is then interpreted by the position $\widehat{w}$:

$$\mathit{run}\,\mathit{run}_f(\mathit{run}_{f,1}\mathit{run}_{x_{j_1}}\mathit{done}_{x_{j_1}}\mathit{done}_{f,1})\cdots(\mathit{run}_{f,1}\mathit{run}_{x_{j_l}}\mathit{done}_{x_{j_l}}\mathit{done}_{f,1})\mathit{done}_f\,\mathit{done}$$

in the game $[\![\mathit{com}_{x_1}, \cdots, \mathit{com}_{x_m}, \mathit{com}_{f,1} \to \mathit{com}_f \vdash \mathit{com}]\!]$, where we have used subscripts to indicate the origin of moves from the various occurrences of com.

Lemma 3. *Suppose $\mathcal{A} = \langle Q, \Sigma, i, F, \delta \rangle$ is a generative automaton and $\Sigma = \{x_1, \cdots, x_m\}$. There exists a term-in-context $\Gamma \vdash M_{\mathcal{A}} : \mathit{com}$, where*

$$\Gamma = x_1 : \mathit{com}, \cdots, x_m : \mathit{com}, f : \mathit{com} \to \mathit{com},$$

such that for all $s \in \mathcal{L}^{\mathsf{wo}}_{[\![\Gamma \vdash \mathit{com}]\!]} \cap \mathcal{L}^{\mathsf{comp}}_{[\![\Gamma \vdash \mathit{com}]\!]}$:

$$[\![\Gamma \vdash M_{\mathcal{A}}]\!](s) = \begin{cases}\mathcal{A}(w) & \exists_{w \in \Sigma^*}\, s = \widehat{w}\\ 0 & \text{otherwise.}\end{cases}$$

Proof. We will construct $M_{\mathcal{A}}$ in such a way that its induced plays will emulate runs of $\mathcal{A}$. The state of $\mathcal{A}$ will be kept in a variable ST. If $|Q| > \mathit{max}$ we will use sufficiently large tuples of variables, which we also denote by ST. Using branching we can easily define a case distinction construct $\mathsf{case}[!ST](\cdots, H_q, \cdots)$ which for each $q \in Q$ selects H_q if $!ST$ represents q. We will use the first-order identifier $f : \mathit{com} \to \mathit{com}$ for iterating the transitions. $M_{\mathcal{A}}$ has the shape

$$\mathbf{new}\; ST\; \mathbf{in}\; ST := i; f(\mathsf{case}[!ST](\cdots, H_q, \cdots)); [!ST \in F]$$

where $[condition] \equiv$ **if** *condition* **then** *skip* **else** *div* and $div \equiv$ **while** 1 **do** *skip*. The condition $!ST \in F$ can also be implemented via branching. Finally, the terms H_q will have the shape $\widehat{N}$ and will be constructed for the distribution $\delta(q) \in \mathcal{P}(\Sigma \times Q)$. Recall that in order to complete the definition of $\widehat{N}$ we need to specify the terms $N_1, \cdots, N_k$. They are defined as follows: if $\delta(q)(x_j, q') = p_i$ then $N_i \equiv (x_j; ST := q')$. $\square$

Theorem 3. $\sqsubseteq_{\approx}$ *is undecidable.*

Proof. Let $\mathcal{A}_1, \mathcal{A}_2$ be generative automata. By the above Lemma and Lemma 1 the refinement of $\mathcal{A}_1$ by $\mathcal{A}_2$ is equivalent to $\Gamma \vdash M_{\mathcal{A}_1} \sqsubseteq_{\approx} M_{\mathcal{A}_2}$. Because REFINEMENT is undecidable, so is $\sqsubseteq_{\approx}$.

Note that the terms used for simulating generative automata are of second order (the types of free identifiers have order 0 or 1) and that the encoding does not rely on **while**. Accordingly, the undecidability result applies not only to $\mathsf{PA_f}$ but also to its **while**-free fragment. Note however that the above argument did depend on free first-order identifiers. If we leave them as well as **while** out, the problem becomes decidable as the length of the induced well-opened traces is then bounded (the corresponding automata have no cycles).

8 Conclusion and Future Work

The main result of this paper is an EXPTIME algorithm for deciding probabilistic contextual equivalence of β-normal second-order Probabilistic Idealized Algol terms. Subject to mild conditions, this corresponds to the natural notion of equivalence for randomized algorithms, namely identical input/output distributions, and therefore enables the comparison of different randomized algorithms against each other.

It can be shown that probabilistic equivalence is PSPACE-hard, since it subsumes the deterministic case, which itself is PSPACE-complete [22]. We conjecture that probabilistic equivalence is in fact also PSPACE-complete. In any case, even the EXPTIME bound is quite encouraging within the realm of verification, and we therefore intend to implement our algorithm to conduct a number of experimental case studies.

An interesting alternative to (exact) probabilistic equivalence is that of *approximate* probabilistic equivalence, parameterized by some small 'tolerance margin' ε. Such a notion would allow us to quantitatively compare two randomized algorithms, or a randomized and a deterministic algorithm, by checking whether their input/output distributions remain within a predetermined small bound of each other. Unfortunately, the most natural interpretation of approximate equivalence is already undecidable for reactive probabilistic automata, by a simple reduction from NONEMPTINESS WITH THRESHOLD. Moreover, *contextual* approximate equivalence for programs seems difficult to define sensibly: if two non-divergent programs fail to be *exactly* equivalent, then a context can always be manufactured that, through some kind of 'statistical testing', can magnify the differences between the two programs to arbitrarily large values. We would like to investigate this question in greater detail, perhaps by restricting contexts to using their arguments only a bounded number of times. We would also like to discover conditions under which such problems become decidable (cf. [17]), or alternatively develop efficient semi-algorithms for them.

References

1. Rabin, M.O.: Algorithms and Complexity. Academic Press, 1976.
2. Motwani, R., Raghavan, P.: Randomized Algorithms. CUP, 1995.
3. Reynolds, J.: The essence of Algol. In Jaco W. de Bakker and J. C. van Vliet, eds.: Algorithmic Languages. North-Holland, 1981.
4. Ghica, D.R., McCusker, G.: Reasoning about Idealized Algol using regular expressions. In ICALP 2000, LNCS 1853.
5. Abramsky, S., Ghica, D.R., Murawski, A.S., Ong, C.H.L.: Applying game semantics to compositional software modelling and verification. In TACAS 2004, LNCS 2988.
6. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: a hybrid approach. In TACAS 2002, LNCS 2280.
7. Bustan, D., Rubin, S., Vardi, M.: Verifying ω-regular properties of Markov chains. In CAV 2004, LNCS 3114.
8. Giacalone, A., Jou, C., Smolka, S.A.: Algebraic reasoning for probabilistic concurrent systems. In IFIP WG 2.2/2.3 Conference on Programming Concepts and Methods, 1990.
9. Lowe, G.: Probabilistic and prioritized models of Timed CSP. Theoretical Computer Science **138(2)**, 1995.
10. Danos, V., Harmer, R.: Probabilistic game semantics. ACM Trans. on Comp. Logic **3(3)**, 2002.
11. Hyland, J.M.E., Ong, C.H.L.: On Full Abstraction for PCF. Information and Computation **163(2)**, 2000.
12. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In O'Hearn, P.W., Tennent, R.D., eds.: Algol-like languages, Birkhaüser, 1997.
13. Rabin, M.O.: Probabilistic automata. Information and Control **6(3)**, 1963.
14. Paz, A.: Introduction to Probabilistic Automata. Academic Press, 1971.
15. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, MIT. Available as Technical Report MIT/LCS/TR-676, 1995.
16. Stoelinga, M.I.A.: An introduction to probabilistic automata. In Rozenberg, G., ed.: EATCS bulletin. Volume 78, 2002.
17. Tzeng, W.G.: A polynomial-time algorithm for the equivalence of probabilistic automata. SIAM Journal on Computing **21**, 1992.
18. Blondel, V.D., Canterini, V.: Undecidable problems for probabilistic automata of fixed dimension. Theoretical Computer Science **36(3)**, 2003.
19. Mohri, M.: Generic e-removal and input e-normalization algorithms for weighted transducers. International Journal of Foundations of Computer Science **13**, 2002.
20. Mohri, M.: Semiring frameworks and algorithms for shortest-distance problems. Journal of Automata, Languages and Combinatorics **7**, 2002.
21. Abramsky, S.: Algorithmic games semantics: a tutorial introduction. In Schwichtenberg, H., Steinbruggen, R., eds.: Proof and System Reliability. Kluwer, 2002.
22. Murawski, A.: Games for complexity of second-order call-by-name programs. Theoretical Computer Science, to appear.