Formal Methods in System Design manuscript No.
(will be inserted by the editor)

Algorithmic probabilistic game semantics

Playing games with automata

Stefan Kiefer · Andrzej S. Murawski · Joël Ouaknine · Björn Wachter · James Worrell

Received: date / Accepted: date

Abstract We present a detailed account of a translation from probabilistic call-by-value programs with procedures to Rabin's probabilistic automata. The translation is fully abstract in that programs exhibit the same computational behaviour if and only if the corresponding automata are language-equivalent. Since probabilistic language equivalence is decidable, we can apply the translation to analyse the behaviour of probabilistic programs and protocols. We illustrate our approach on a number of case studies.

1 Introduction

Ever since Michael Rabin's seminal paper on probabilistic algorithms [38], it has been widely recognised that introducing randomisation 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 [33] 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.

Research funded by EPSRC (EP/G069158/1).

S. Kiefer · J. Ouaknine · B. Wachter · J. Worrell
Dept of Computer Science, Univ. of Oxford, Parks Road, Oxford OX1 3QD, UK

A. S. Murawski
Dept of Computer Science, Univ. of Leicester, University Road, Leicester LE1 7RH, UK

Our work exploits some of the latest modelling techniques developed by the denotational semantics community to handle probabilistic programs. Their distinctive feature is that they capture observational program behaviour, as opposed to individual computational steps that a program can make during execution, as in operational semantics. Technically speaking, the models we rely on are *fully abstract* [31]: two programs have the same denotations if and only if they are *contextually equivalent*. The notion of contextual equivalence relies on comparing the outcomes of deploying programs in all possible contexts. In the probabilistic case, the outcomes are measured through probability distributions.

Contextual equivalence is an expressive and powerful technique, as it subsumes many other properties. As will be seen in the Dining Cryptographers case study, it is particularly suitable to analysing anonymity protocols, as by its very definition it provides an account of what can be observed about a protocol and hides unobservable actions, such as histories of private variables. In general, contextual equivalence is very difficult to reason about due to the universal quantification over *all* syntactic contexts in which the program behaviours need to be considered.

Game semantics [2,20,36] has emerged as a versatile technique for building models that pass the criterion of full abstraction. Its development has proceeded along all axes of programming paradigms, and is by now applicable to higher-order types, concurrency, polymorphism, probability and references.

In our work we rely on a game model of an ML-like higher-order language RML, for which a game model was constructed by Abramsky and McCusker [3]. Equivalently, the model can be presented in the Honda-Yoshida style [17], which we shall follow because it is more direct and relies on a more economical alphabet of moves. In order to model a probabilistic variant of the language, we adopt the probabilistic framework of Danos and Harmer [11]. In contrast to previous expository papers on game semantics [4,1], our account of probabilistic game semantics uses a *call-by-value* evaluation strategy, which underpins most modern programming languages. Note that our earlier paper [28] was based on a *call-by-name* probabilistic game semantics and was missing many implementation-related details. As well as being more self-contained, the present approach enables more direct and transparent encodings of our various case studies.

For the purpose of deriving a formalism compatible with finite-state modelling methods we consider a fragment of the language, in which programs have finite datatypes and the type system is restricted to first-order types (this will mean that programs have first-order types as well as being able to use free first-order identifiers, which correspond to unknown first-order components). For the fragment in question we define a compositional translation from terms to probabilistic automata. Our main result, Theorem 2, states that the automata capture the semantics, i.e. the associated probabilistic language is a faithful representation of the program's game semantics. Consequently, probabilistic language equivalence of the derived automata characterizes contextual

program equivalence, and the associated algorithms can be applied to solve contextual equivalence problems.

We have implemented a tool APEX that takes a program (in the above-defined fragment) as input and returns a probabilistic automaton capturing the game semantics of the program. This paper describes a number of case studies illustrating the use of APEX to verify anonymity and almost-sure termination, and to perform average-case analysis of data types. These examples highlight some of the distinctive features of our approach, including the ability to describe models in a high-level language, to model *open* programs with free identifiers and to check contextual equivalence.

Related Work Most probabilistic model checkers use temporal logic to express specifications. Among this class are PRISM [27], ProbVerus [14], RAPTURE [12], ETMCC [15], APNN-toolbox [5], MRMC [23], Ymer [46], and LiQuor [9]. For the most part, these tools use probabilistic and continuous-time variants of Computation Tree Logic, although Linear Temporal Logic is also supported in [9,5,6]. The class of models treated includes discrete and continuous-time Markov chains, Markov decision processes (which feature both probabilistic and nondeterministic branching), stochastic Petri nets, and process algebras. The model-checking algorithms implemented in these tools combine state-of-the-art techniques from non-probabilistic model checking, including symbolic representations [27,5], symmetry reduction and bisimulation minimisation [23], abstraction /refinement [12], partial-order reduction [9], together with techniques for handling large systems of linear equations and linear programs, as well as statistical techniques, such as sampling [46].

The approach put forward with APEX is set apart by its ability to check equivalence between programs and their specification, which is enabled by the ability of game semantics to hide internal behavior. Hiding, in turn, can also be a means to produce a small semantics of a program because the program captures just its input and output behavior. The more compact automaton can then be interpreted more easily and analysed more efficiently. We have demonstrated this approach in a variety of case studies.

Verification of probabilistic programs has also been studied in the context of program logics and theorem proving. McIver and Morgan [29] give a proof system to reason about probabilistic properties of programs; a logic-based approach has also been pursued by Hurd [19].

2 Programming Language

We consider a probabilistic ML-like language pRML over a finite ground datatype $\{0, \cdots, max\}$ ($max > 0$). pRML is a *probabilistic* and *finitary* variant of RML [3], obtained by augmenting RML with coin-tossing, restricting the datatypes to finite ones and replacing general recursion with loops. Ultimately, for the purpose of obtaining a correspondence with probabilistic automata, we shall consider a fragment of pRML in Subsection 2.2.

$$\frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{n \in \{0, \cdots, max\}}{\Gamma \vdash n : \mathsf{int}} \qquad \frac{p_0, \cdots, p_{max} \in \mathbb{Q} \qquad \sum_{i=0}^{max} p_i = 1}{\Gamma \vdash \mathsf{coin}[0 : p_0, \cdots, max : p_{max}] : \mathsf{int}}$$

$$\frac{\Gamma \vdash M : \mathsf{int} \qquad \Gamma \vdash M_i : \theta \qquad i = 0, \cdots, max}{\Gamma \vdash \mathsf{case}(M)[M_0, \cdots, M_{max}] : \theta} \qquad \frac{\Gamma \vdash M : \mathsf{int} \quad \Gamma \vdash N : \mathsf{unit}}{\Gamma \vdash \mathsf{while}\ M\ \mathsf{do}\ N : \mathsf{unit}}$$

$$\frac{(x : \theta) \in \Gamma}{\Gamma \vdash x : \theta} \qquad \frac{\Gamma, x : \theta_1 \vdash M : \theta_2}{\Gamma \vdash \lambda x^{\theta_1}.M : \theta_1 \to \theta_2} \qquad \frac{\Gamma \vdash M : \theta_1 \to \theta_2 \quad \Gamma \vdash N : \theta_1}{\Gamma \vdash MN : \theta_2}$$

$$\frac{\Gamma \vdash M : \mathsf{var}}{\Gamma \vdash !M : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{var} \quad \Gamma \vdash N : \mathsf{int}}{\Gamma \vdash M := N : \mathsf{unit}} \qquad \frac{\Gamma, x : \mathsf{var} \vdash M : \theta}{\Gamma \vdash \mathsf{new}\ x\ \mathsf{in}\ M : \theta}$$

$$\frac{\Gamma \vdash M : \mathsf{unit} \to \mathsf{int} \qquad \Gamma \vdash N : \mathsf{int} \to \mathsf{unit}}{\Gamma \vdash \mathsf{mkvar}(M, N) : \mathsf{var}}$$

Fig. 1 Syntax of pRML.

2.1 pRML

Formally, pRML is based on types θ generated according to the grammar given below.

$$\theta \quad ::= \quad \mathsf{unit} \quad | \quad \mathsf{int} \quad | \quad \mathsf{var} \quad | \quad \theta \to \theta$$

Terms of pRML along with typing judgments of the form

$$x_1 : \theta_1, \cdots, x_k : \theta_k \vdash M : \theta$$

are presented in Figure 1.

Remark 1 When referring to $\mathsf{coin}[0 : p_0, \cdots, max : p_{max}]$, we shall often omit the entries with zero probabilities and write, for instance, $\mathsf{coin}[0 : 0.5,\ 1 : 0.5]$. We restrict the probabilities $p_0, \cdots, p_{max}$ to be rational to achieve decidability.

Remark 2 $\mathsf{mkvar}(M, N)$ is the so-called *bad-variable constructor*. It allows one to construct terms of type var, which are unlike standard (good) variables. As the reduction rules will make clear, $\mathsf{mkvar}(M, N)$ is best viewed as an "object" of type var equipped with its own reading (M) and writing (N) methods. We further discuss the role of bad variables in Remark 3.

The operational semantics of pRML is defined in Figure 2 for terms of the shape

$$x_1 : \mathsf{var}, \cdots, x_k : \mathsf{var} \vdash M : \theta.$$

It relies on statements of the form $(s, M) \Downarrow^p (s', V)$, where s, s' are states (formally, functions from $\{x_1, \cdots, x_k\}$ to $\{0, \cdots, max\}$) and V ranges over values.

One can interpret a concrete derivation of $(s, M) \Downarrow^p (s', V)$ as saying that the associated evaluation of program M and state s to value V and state s' has probability p. Because of the term coin, M may have countably many evaluations from a given state. We shall write $(s, M) \downarrow^p V$ iff $p = \sum p_i$ and the sum ranges over all derivations of $(s, M) \Downarrow^{p_i} (s', V)$ for some s'. If there are no

$$\frac{}{s, V \Downarrow^1 s, V} \qquad \frac{}{s, \mathsf{coin}[0 : p_0, \cdots, max : p_{max}] \Downarrow^{p_n} s, n}$$

$$\frac{s, M \Downarrow^p s', n \quad s', M_n \Downarrow^q s'', V}{s, \mathsf{case}(M)[M_0, \cdots, M_{max}] \Downarrow^{pq} s'', V} \qquad \frac{(s \,|\, x \mapsto 0), M \Downarrow^p s', V}{s, \mathsf{new}\, x \,\mathsf{in}\, M \Downarrow^p s', V}$$

$$\frac{s, M \Downarrow^p s', x}{s, !M \Downarrow^p s', s'(x)} \qquad \frac{s, M \Downarrow^p s', x \quad s', N \Downarrow^q s'', n}{s, M := N \Downarrow^{pq} s''(x \mapsto n), ()}$$

$$\frac{s, M \Downarrow^p s', \lambda x.M' \quad s', N \Downarrow^q s'', V \quad s'', M'[V/x] \Downarrow^r s''', V'}{s, MN \Downarrow^{pqr} s''', V'}$$

$$\frac{s, M \Downarrow^p s', 0}{s, \mathsf{while}\, M \,\mathsf{do}\, N \Downarrow^p s', ()}$$

$$\frac{s, M \Downarrow^p s', n \quad (n > 0) \quad s', N \Downarrow^q s'', () \quad s'', \mathsf{while}\, M \,\mathsf{do}\, N \Downarrow^r s''', ()}{s, \mathsf{while}\, M \,\mathsf{do}\, N \Downarrow^{pqr} s''', ()}$$

$$\frac{s, M \Downarrow^p s', V_r \quad s', N \Downarrow^q s'', V_w}{s, \mathsf{mkvar}(M, N) \Downarrow^{pq} s'', \mathsf{mkvar}(V_r, V_w)}$$

$$\frac{s, M \Downarrow^p s', \mathsf{mkvar}(V_r, V_w) \qquad s', V_r() \Downarrow^q s'', n}{s, !M \Downarrow^{pq} s'', n}$$

$$\frac{s, M \Downarrow^p s', \mathsf{mkvar}(V_r, V_w) \quad s', N \Downarrow^q s'', n \quad s'', V_w n \Downarrow^r s''', ()}{s, M := N \Downarrow^{pqr} s''', ()}$$

V ranges over values which can take one of the following shapes: $()$, n, x^{var}, $\lambda x^{\theta}.M$, $\mathsf{mkvar}(\lambda x^{\mathsf{unit}}.M, \lambda y^{\mathsf{int}}.N)$. $s(x \mapsto n)$ stands for s in which $s(x)$ has been changed to n. $s' = (s \,|\, x \mapsto n)$ is an extension of s, defined only if $x \notin \mathsf{dom}(s)$, which coincides with s on $\mathsf{dom}(s)$ and satisfies $s'(x) = n$.

Fig. 2 Operational semantics of pRML.

such derivations, we simply write $(s, M) \downarrow^0 V$. The judgment $(s, M) \downarrow^p V$ thus denotes the fact that the probability of evaluating M in state s to value V is p. When M is closed, i.e. $\vdash M : \theta$, we write $M \downarrow^p V$, because s is then empty. For instance, we have $\mathsf{coin}[0 : 0.5, 1 : 0.5] \downarrow^{0.5} 0$ and $\mathsf{coin}[0 : 0.5, 1 : 0.5] \downarrow^{0.5} 1$.

We now define the notion of contextual equivalence induced by the operational semantics. A context $\mathcal{C}[-]$ is simply a term of the language containing a placeholder $[-]$ that can be instantiated with a term of pRML to yield another term.

Definition 1 Terms $\Gamma \vdash M_1 : \theta$ and $\Gamma \vdash M_2 : \theta$ are *contextually equivalent* (written $\Gamma \vdash M_1 \cong M_2 : \theta$) if for all contexts $\mathcal{C}[-]$ such that $\vdash \mathcal{C}[M_1], \mathcal{C}[M_2] : \mathsf{unit}$ we have $\mathcal{C}[M_1] \downarrow^p ()$ if and only if $\mathcal{C}[M_2] \downarrow^p ()$.

While contextual equivalence of closed terms $\vdash M_1 : \mathsf{int}$, $\vdash M_2 : \mathsf{int}$ coincides with the equality of induced subdistributions on values[1], equivalences involving terms of higher-order types as well as terms with free identifiers (open program phrases) are difficult to reason about directly. This is because contextual equivalence relies, on universal quantification over contexts, i.e.

[1] Note that we need to talk about subdistributions because of divergence.

all possible instantiations of the unknown identifiers. The main approaches to handling such equivalences for languages with rich type structures are currently logical relations (see e.g. [13]), environmental bisimulation (see e.g. [41]) and game semantics. As far as we know, the former two have not been applied in the probabilistic context.

Example 1
- A loop that terminates with probability one is equivalent to (), e.g., $\vdash \mathsf{while}\,(\mathsf{coin}[0:0.7, 1:0.3])\,\mathsf{do}\,() \cong ()$.
- It can be shown that $x : \mathsf{int} \vdash M_0 \cong M_1 : \mathsf{int}$ holds exactly when $M_0[m/x] \downarrow_p n$ if and only if $M_1[m/x] \downarrow_p n$ for all $m, n \in \{0, \cdots, max\}$. That is to say, $\cong$ generalizes the concept of equality of input/output functions.
- We have $f : \mathsf{int} \to \mathsf{unit} \vdash f(\mathsf{coin}[1:0.5,\ 2:0.5]) \cong M : \mathsf{unit}$, where M is the term

$$
\begin{array}{l}
\mathsf{new}\,x\,\mathsf{in}\,(\\
\quad x := 1; \\
\quad \mathsf{while}\,(!x)\,\mathsf{do} \\
\qquad \mathsf{case}(\mathsf{coin}[0:p_0, 1:p_1, 2:p_2])[\ (),\ x := 0; f(1)\ ,\ x := 0; f(2)] \\
)
\end{array}
$$

  and p_0, p_1, p_2 are chosen so that $p_0 + p_1 + p_2 = 1$ and $p_1 = p_2$.

Remark 3 In [40] Reynolds advocated modelling the type var as $(\mathsf{unit} \to \mathsf{int}) \times (\mathsf{int} \to \mathsf{unit})$. While this approach is conceptually elegant, many elements of the product space do not behave like memory cells (good variables). Consequently, to achieve full abstraction, so-called "bad variables" have to be admitted into the language, e.g. in the form of the mkvar constructor. However, if no type in Γ or θ contains an occurrence of var, contextual equivalence is completely unaffected by the inclusion of mkvar [34]. This will be true in all of our case studies.

Despite the absence of general recursion and the restriction to a finite ground type, contextual equivalence in pRML is already undecidable in the deterministic case without loops, e.g. at type $(\mathsf{unit} \to \mathsf{unit}) \to (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$ [34]. The probabilistic setting is known to be a conservative extension [11, Corollary 3.7], thus to yield decidability results pRML has to be restricted further. In the deterministic case, research in that direction has been carried out using regular [34] and visibly pushdown languages [18].

2.2 A Restricted Language

In this section we focus on a restriction of pRML to basic first-order types, which we call $\mathcal{L}$. It features terms with first-order procedures, also indeterminate ones. The latter are listed on the left-hand side of the turnstile in the typing rules.

$$\frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{n \in \{0, \cdots, max\}}{\Gamma \vdash n : \mathsf{int}} \qquad \frac{p_0, \cdots, p_{max} \in \mathbb{Q} \qquad \sum_{i=0}^{max} p_i = 1}{\Gamma \vdash \mathsf{coin}[0 : p_0, \cdots, max : p_{max}] : \mathsf{int}}$$

$$\frac{(x : \mathsf{int}) \in \Gamma \qquad \Gamma \vdash M_i : \theta \qquad i = 0, \cdots, max}{\Gamma \vdash \mathsf{case}(x)[M_0, \cdots, M_{max}] : \theta} \qquad \frac{(f : \mathsf{unit} \to \mathsf{int}), (g : \mathsf{unit} \to \mathsf{unit}) \in \Gamma}{\Gamma \vdash \mathsf{while}\, f()\, \mathsf{do}\, g() : \mathsf{unit}}$$

$$\frac{(x : \beta) \in \Gamma}{\Gamma \vdash x : \beta} \qquad \frac{\Gamma, x : \beta_1 \vdash M : \beta_2}{\Gamma \vdash \lambda x^{\beta_1}.M : \beta_1 \to \beta_2} \qquad \frac{(f : \beta_1 \to \beta_2), (x : \beta_1) \in \Gamma}{\Gamma \vdash fx : \beta_2}$$

$$\frac{(x : \mathsf{var}) \in \Gamma}{\Gamma \vdash !x : \mathsf{int}} \qquad \frac{(x : \mathsf{var}), (y : \mathsf{int}) \in \Gamma}{\Gamma \vdash x := y : \mathsf{unit}} \qquad \frac{\Gamma, x : \mathsf{var} \vdash M : \theta}{\Gamma \vdash \mathsf{new}\, x\, \mathsf{in}\, M : \theta}$$

$$\frac{(x : \mathsf{unit} \to \mathsf{int}), (y : \mathsf{int} \to \mathsf{unit}) \in \Gamma}{\Gamma \vdash \mathsf{mkvar}(x, y) : \mathsf{var}} \qquad \frac{\Gamma \vdash M : \theta_1 \qquad \Gamma, x : \theta_1 \vdash N : \theta_2}{\Gamma \vdash \mathsf{let}\, x = M\, \mathsf{in}\, N : \theta_2}$$

Fig. 3 Syntax of $\mathcal{L}$.

$\mathcal{L}$ uses types θ generated according to the grammar given below.

$$\theta \quad ::= \quad \beta \quad | \quad \mathsf{var} \quad | \quad \beta \to \beta \qquad\qquad \beta \quad ::= \quad \mathsf{unit} \quad | \quad \mathsf{int}$$

Its syntax is presented in Figure 3. The $\mathsf{let}\, x = M\, \mathsf{in}\, N$ term corresponds to $(\lambda x^{\theta}.N)M$ from pRML. Note that, as a special case, $\mathsf{let}\, x = M\, \mathsf{in}\, N$ allows the user to define a procedure M and link it to the code that uses it.

Our syntax is minimalistic in order to simplify the meta-theory: we have isolated the simplest instances of every construct, e.g. $\mathsf{case}(x)[\cdots]$, $x := y$, fx, $\mathsf{while}\, f()\, \mathsf{do}\, g()$, so that their more structured (and commonly used) variants can easily be compiled into $\mathcal{L}$ as "syntactic sugar". For instance, assuming $\Gamma \vdash M : \mathsf{unit}$, the term $M; N$ corresponds simply to $\mathsf{let}\, x = M\, \mathsf{in}\, N$. Here is a selection of other cases:

- $\mathsf{case}(M)[M_0, \cdots, M_{max}]$ corresponds to $\mathsf{let}\, x = M\, \mathsf{in}\, \mathsf{case}(x)[M_0, \cdots, M_{max}]$;
- $x := N$ corresponds to $\mathsf{let}\, y = N\, \mathsf{in}\, x := y$;
- fN corresponds to $\mathsf{let}\, x = N\, \mathsf{in}\, fx$ and, more generally, MN is $\mathsf{let}\, f = M\, \mathsf{in}\, \mathsf{let}\, x = N\, \mathsf{in}\, fx$;
- $\mathsf{while}\, M\, \mathsf{do}\, N$ corresponds to

$$\mathsf{let}\, f = \lambda x^{\mathsf{unit}}.M\, \mathsf{in}\, \mathsf{let}\, g = \lambda y^{\mathsf{unit}}.N\, \mathsf{in}\, (\mathsf{while}\, f()\, \mathsf{do}\, g()).$$

The reader may be puzzled by the minimalistic rule for while, which involves $f : \mathsf{unit} \to \mathsf{int}$, $g : \mathsf{unit} \to \mathsf{unit}$. It is necessitated by the fact that, under call-by-value evaluation, $\mathsf{let}\, x = M\, \mathsf{in}\, \mathsf{let}\, y = N\, \mathsf{in}\, (\mathsf{while}\, x\, \mathsf{do}\, y)$ and $\mathsf{while}\, M\, \mathsf{do}\, N$ are not equivalent: in the first case M and N are evaluated only once. Hence, the somewhat complicated shape of our rule, which allows us to handle the general case through a basic case and let.

3 Games

In this section we give a self-contained account of the probabilistic game semantics of $\mathcal{L}$ obtained by combining the Honda-Yoshida approach to modelling call-by-value evaluation [17] with the Danos-Harmer probabilistic games framework [11].

Game semantics views computation as a dialogue (exchange of moves) between the environment (Opponent, O) and the program (Proponent, P). The dialogues are called *plays*. In our setting, the players will always alternate. Programs are then interpreted by strategies, which tell P whether and how to extend a play after an O-move. Danos and Harmer introduced probabilistic game semantics: a probabilistic strategy assigns a probability to each even-length play, subject to several technical constraints [11]. Intuitively, the strategy provides P with a (sub)distribution on responses at each stage of the game at which P is to make a move. It was shown that a certain class of plays, called *complete*, suffice to account for probabilistic program equivalence [35]. Hence we shall only concentrate on these in what follows.

Because we intend to relate the model to probabilistic automata, we have made our account very direct and restricted the use of the usual technical machinery needed to set up a game model (arena, justification, strategy, composition) to a minimum. Let us begin with some basic definitions specifying what moves are available in our games and how they can be used in a play. We start off by defining an auxiliary concept of tags.

Definition 2 Let Γ be a context and θ a type. If x occurs in Γ, we write θ_x for the corresponding type. Let us define the set $\mathcal{T}_{\Gamma,\theta}$ of associated *tags* to be

$$\begin{aligned}
&\{\mathsf{c}_x, \mathsf{r}_x \quad | \quad (x:\theta_x) \in \Gamma, \quad \theta_x \equiv \beta_1 \to \beta_2\} \quad \cup \\
&\{\mathsf{c}_x^r, \mathsf{r}_x^r, \mathsf{c}_x^w, \mathsf{r}_x^w \quad | \quad (x:\theta_x) \in \Gamma, \quad \theta_x \equiv \mathsf{var}\} \quad \cup \\
&\{\mathsf{r}_\downarrow\} \quad \cup \quad \{\mathsf{c}, \mathsf{r} \quad | \quad \theta \equiv \beta_1 \to \beta_2\} \quad \cup \quad \{\mathsf{c}^r, \mathsf{r}^r, \mathsf{c}^w, \mathsf{r}^w \quad | \quad \theta \equiv \mathsf{var}\}.
\end{aligned}$$

Thus, for each function-type identifier x in Γ, we have tags c_x and r_x representing calls and returns related to that identifier. $\mathsf{r}_\downarrow$ can be taken to correspond to the fact that evaluation was successfully completed. If θ is a function type then c and r refer respectively to calling the corresponding value and obtaining a result. The intuitions behind $\mathsf{c}_x^r, \mathsf{r}_x^r, \mathsf{c}_x^w, \mathsf{r}_x^w$ are analogous except that they refer respectively to the reading and writing actions associated with $x : \mathsf{var}$. Similarly, $\mathsf{c}^r, \mathsf{r}^r, \mathsf{c}^w, \mathsf{r}^w$ are used if $\theta \equiv \mathsf{var}$, to account for reads and writes for the corresponding value.

Next we define *initial moves*, which simply contain values for all free integer-typed identifiers and $\star$ for other identifiers. For the definition to make sense we need to assume that the identifiers in Γ are listed in some order. We write $\mathbb{V}$ for the set $\{\star\} \cup \{0, \cdots, max\}$.

Definition 3 (Initial moves) Given $\Gamma = [x_1 : \theta_1, \cdots, x_k : \theta_k]$, we define the set I_Γ of *initial moves* to be

$$\{(v_1, \cdots, v_k) \in \mathbb{V}^k \quad | \quad \forall_{1 \le i \le k} \ (v_i = \star \iff \theta_i \not\equiv \mathsf{int})\}.$$

We shall use ι to range over initial moves and write $\iota(x_i)$ to refer to v_i.

Given θ, let us define $\mathbb{V}_\theta$ to be $\{0, \cdots, max\}$ if $\theta \equiv \mathsf{int}$ and $\{\star\}$ otherwise.

Definition 4 Let $t \in \mathcal{T}_{\Gamma,\theta}$ and $v \in \mathbb{V}$. We shall say that (t, v) is a *non-initial move* if it has one of the shapes listed below.

Shape	Condition
$(\mathsf{c}_x, v_1), (\mathsf{r}_x, v_2)$	$\theta_x \equiv \beta_1 \to \beta_2,\ v_1 \in \mathbb{V}_{\beta_1},\ v_2 \in \mathbb{V}_{\beta_2}$
$(\mathsf{r}_\downarrow, v)$	$v \in \mathbb{V}_\theta$
$(\mathsf{c}, v_1), (\mathsf{r}, v_2)$	$\theta \equiv \beta_1 \to \beta_2,\ v_1 \in \mathbb{V}_{\beta_1},\ v_2 \in \mathbb{V}_{\beta_2}$
$(\mathsf{c}^r_x, \star), (\mathsf{r}^r_x, v)$	$\theta_x \equiv \mathsf{var},\ v \in \mathbb{V}_{\mathsf{int}}$
$(\mathsf{c}^w_x, v), (\mathsf{r}^w_x, \star)$	
$(\mathsf{c}^r, \star), (\mathsf{r}^r, v)$	$\theta \equiv \mathsf{var},\ v \in \mathbb{V}_{\mathsf{int}}$
$(\mathsf{c}^w, v), (\mathsf{r}^w, \star)$	

Next we introduce complete plays, as these suffice to reason about equivalence in our case [35].

Definition 5 Let Γ be a context and θ a type. A *complete play*[2] over Γ, θ is a (possibly empty) sequence of moves $\iota(t_1, v_1) \cdots (t_k, v_k)$ such that $\iota \in I_\Gamma$, $t_i \in \mathcal{T}_{\Gamma,\theta}$ $(i = 1, \ldots, k)$ and the corresponding sequence of tags $t_1 \cdots t_k$ matches the regular expression

$$X\, \mathsf{r}_\downarrow\, (\mathsf{c}\, X\, \mathsf{r} + \mathsf{c}^r\, X\, \mathsf{r}^r + \mathsf{c}^w\, X\, \mathsf{r}^w)^*$$

where

$$X = (\sum_{\substack{(x:\theta_x)\in\Gamma \\ \theta_x \equiv \beta_1 \to \beta_2}} (\mathsf{c}_x\, \mathsf{r}_x) + \sum_{\substack{(x:\theta_x)\in\Gamma \\ \theta_x \equiv \mathsf{var}}} ((\mathsf{c}^r_x\, \mathsf{r}^r_x) + (\mathsf{c}^w_x\, \mathsf{r}^w_x))\ )^*.$$

We write $\mathcal{C}_{\Gamma,\theta}$ to refer to the set of complete plays over Γ, θ.

The shape of complete plays can be thought of as a record of successful computation. First, calls are made to free identifiers of non-ground types in the context (expression X) then a value is reached ($\mathsf{r}_\downarrow$) and, if the value is of a non-ground type, we have a series of calls and matching returns corresponding to uses of the value, possibly separated by external calls with matching returns.

Remark 4 In game semantics moves are assigned ownership: O (environment) or P (program). ι and those with tags $\mathsf{r}_x, \mathsf{r}^r_x, \mathsf{r}^w_x, \mathsf{c}, \mathsf{c}^r, \mathsf{c}^w$ belong to O and the rest (tags $\mathsf{c}_x, \mathsf{c}^r_x, \mathsf{c}^w_x, \mathsf{r}_\downarrow, \mathsf{r}, \mathsf{r}^r, \mathsf{r}^w$) to P. We shall write M^O and M^P to refer to O- and P-moves respectively.

Following [11], we shall interpret terms $\Gamma \vdash M : \theta$ by functions $\sigma : \mathcal{C}_{\Gamma,\theta} \to [0, 1]$, which we call $[0, 1]$-*weighted complete plays on* Γ, θ. We write $\sigma : \Gamma, \theta$ to refer to them.

[2] Readers familiar with game semantics will notice that we omit justification pointers. This is because they are uniquely recoverable in the sequences of moves under consideration.

4 Interpretation of $\mathcal{L}$ through weighted complete plays

What we describe below amounts to a more concrete reformulation of the corresponding probabilistic game semantics [17,11], aimed to highlight elements relevant to the subsequent translation to probabilistic automata. Below for each term of $\mathcal{L}$ we define the corresponding σ by listing all complete plays with non-zero weights.

- $\Gamma \vdash () : \mathsf{unit}$
$$\sigma(\ \iota\,(\mathsf{r}_\downarrow, \star)\ ) = 1$$
- $\Gamma \vdash n : \mathsf{int}$
$$\sigma(\ \iota\,(\mathsf{r}_\downarrow, n)\ ) = 1$$
- $\Gamma \vdash \mathsf{coin}[0 : p_0, \cdots, max : p_{max}] : \mathsf{int}$
$$\sigma(\ \iota\,(\mathsf{r}_\downarrow, n)\ ) = p_n \qquad n = 0, \cdots, max$$
- $\Gamma \vdash \mathsf{case}(x)[M_0, \cdots, M_{max}]$

  Let σ_n correspond to the terms M_n $(n = 0, \cdots, max)$.
$$\sigma(\ \iota\, s\ ) = \sigma_{\iota(x)}(\ \iota\, s\,)$$
- $\Gamma \vdash \mathsf{while}\, f()\, \mathsf{do}\, g() : \mathsf{unit}$

  Let s range over sequences matching $(\ (\sum_{n=1}^{max}(\mathsf{c}_f, \star)\,(\mathsf{r}_f, n))\,(\mathsf{c}_g, \star)\,(\mathsf{r}_g, \star)\ )^*$.
$$\sigma(\ \iota\, s\,(\mathsf{c}_f, \star)\,(\mathsf{r}_f, 0)\,(\mathsf{r}_\downarrow, \star)\ ) = 1$$
- $\Gamma \vdash x : \beta$
$$\sigma(\ \iota\,(\mathsf{r}_\downarrow, \iota(x))\ ) = 1$$
- $\Gamma \vdash \lambda x^{\beta_1}.M : \beta_1 \to \beta_2$

  Let σ' correspond to $\Gamma, x : \beta_1 \vdash M : \beta_2$. Suppose $u_i \in \mathbb{V}_{\beta_1}$, $v_i \in \mathbb{V}_{\beta_2}$ $(i = 1, \cdots, k)$.
$$\sigma(\ \iota\,(\mathsf{r}_\downarrow, \star)\ ) = 1$$
$$\sigma(\ \iota\,(\mathsf{r}_\downarrow, \star)\,(\mathsf{c}, u_1)s_1(\mathsf{r}, v_1)\ \cdots\ (\mathsf{c}, u_k)s_k(\mathsf{r}, v_k)\ ) = \textstyle\prod_{i=1}^{k} \sigma'(\ (\iota, u_i)\, s_i\,(\mathsf{r}_\downarrow, v_i)\ )$$
- $\Gamma \vdash fx : \beta_2$
$$\sigma(\ \iota\,(\mathsf{c}_f, \iota(x))(\mathsf{r}_f, v)(\mathsf{r}_\downarrow, v)\ ) = 1 \qquad v \in \mathbb{V}_{\beta_2}$$
- $\Gamma \vdash !x : \mathsf{int}$
$$\sigma(\ \iota\,(\mathsf{c}_x^r, \star)(\mathsf{r}_x^r, v)(\mathsf{r}_\downarrow, v)\ ) = 1 \qquad v \in \mathbb{V}_{\mathsf{int}}$$
- $\Gamma \vdash x := y : \mathsf{unit}$
$$\sigma(\ \iota\,(\mathsf{c}_x^w, \iota(y))(\mathsf{r}_x^w, \star)(\mathsf{r}_\downarrow, \star)\ ) = 1$$

– $\Gamma \vdash \mathsf{mkvar}(x, y) : \mathsf{var}$

$$\sigma(\ \iota\,(\mathsf{r}_\downarrow, \star)s\ ) = 1$$

s ranges over sequences matching

$$(\sum_{n=0}^{max} (\mathsf{c}^r, \star)(\mathsf{c}^r_x, \star)(\mathsf{r}^r_x, n)(\mathsf{r}^r, n) + (\mathsf{c}^w, n)(\mathsf{c}^w_y, n)(\mathsf{r}^w_y, \star)(\mathsf{r}^w, \star))^*$$

To interpret $\mathsf{new}\, x\, \mathsf{in}\, M$ and $\mathsf{let}\, x = M\, \mathsf{in}\, N$ we need to introduce a notion of composition on complete plays, which is a specialisation of the composition of probabilistic strategies [11, Section 2.4] to types occurring in $\mathcal{L}$.

First we analyse a particular way in which two complete plays can be combined. Let S be a special tag.

– Given $s_1 \in \mathcal{C}_{\Gamma,\theta}$ let us write $s_1[t_x/t, (S, v)/(\mathsf{r}_\downarrow, v)]$, or s'_1 for short, for the sequence s_1 in which each tag from $\{\mathsf{c}, \mathsf{r}, \mathsf{c}^r, \mathsf{r}^r, \mathsf{c}^w, \mathsf{r}^w\}$ has been replaced with a corresponding tag subscripted with x and the (unique) move $(\mathsf{r}_\downarrow, v)$ with (S, v).
– Given $s_2 \in \mathcal{C}_{\Gamma+\{x:\theta\},\theta'}$ let $s_2[\iota\,(S, v)/(\iota, v)]$, or s'_2 for short, be the sequence s_2 in which the initial move (ι, v) was replaced with ι followed by (S, v).

Let $\mathcal{B} = I_\Gamma \cup (\{S, \mathsf{c}_x, \mathsf{r}_x, \mathsf{c}^r_x, \mathsf{r}^r_x, \mathsf{c}^w_x, \mathsf{r}^w_x\} \times \mathbb{V})$. We say that s_1 and s_2 are *compatible* if $s'_1 \restriction \mathcal{B} = s'_2 \restriction \mathcal{B}$, where $s \restriction \mathcal{B}$ denotes the subsequence of s restricted to moves in $\mathcal{B}$. We use $s \setminus \mathcal{B}$ to refer to the dual subsequence of s consisting of moves that do not belong to $\mathcal{B}$. Given compatible s_1 and s_2, we define another sequence $s_1 \lightning s_2$ by

$$s_1 \lightning s_2 = s'_1 \underset{\mathcal{B}}{|||}\ s'_2\,,$$

where $|||_{\mathcal{B}}$ is the selective synchronised parallel construct of CSP. This produces an interleaving of s'_1 and s'_2 subject to the requirement that they synchronise on events from $\mathcal{B}$. Because the sequences of tags underlying s'_1 and s'_2 have the following respective shapes (for clarity we ignore tags with superscripts):

$$\begin{array}{ll} \iota\, X^*\, S\, (\mathsf{c}_x\, X^*\, \mathsf{r}_x)^* & X = \sum_y \mathsf{c}_y\, \mathsf{r}_y \\ \iota\, S\, Y^*\, \mathsf{r}_\downarrow\, (\mathsf{c}\, Y^*\, \mathsf{r})^* & Y = X + \mathsf{c}_x \mathsf{r}_x \end{array}$$

there is actually only one way of synchronising s'_1 and s'_2: both must first synchronise on ι, then s_1 can proceed with X^*, after which synchronisation on S must take place, s_2 can then proceed (Y^*) but, once it encounters c_x, s_1 will regain control until it eventually reaches r_x, when s_2 can continue, etc. Thus the synchronised sequence has to match the following expression

$$\iota\, X_1^*\, S\, (X_2 + \mathsf{c}_x\, X_1^*\, \mathsf{r}_x)^*\, \mathsf{r}_\downarrow\, (\mathsf{c}\, (X_2 + \mathsf{c}_x\, X_1^*\, \mathsf{r}_x)^*\, \mathsf{r})^*$$

where the subscripts $1, 2$ indicate whether the X segment originates from s_1 or s_2. So long as s_1 and s_2 are compatible, they can always be synchronised in a unique way.

Definition 6 Given $\sigma_1 : \Gamma, \theta$ and $\sigma_2 : \Gamma + \{x : \theta\}, \theta'$ we define $\sigma_2[\sigma_1/x] : \Gamma, \theta$ as follows:

$$\sigma_2[\sigma_1/x](s) = \sum_{\substack{s_1 \in \mathcal{C}_{\Gamma,\theta},\ s_2 \in \mathcal{C}_{\Gamma\cup\{x:\theta\},\theta'} \\ s_1, s_2 \text{ compatible} \\ s = (s_1 \lightning s_2) \setminus \mathcal{B}}} \sigma_1(s_1) \cdot \sigma_2(s_2),$$

where $\setminus\mathcal{B}$ corresponds to hiding all actions from $\mathcal{B}$.

Note that this sum may diverge in general. This is guaranteed not to happen in our case, though, thanks to compatibility with the Danos-Harmer framework [11, Proposition 2.2]. The remaining constructs of the language can now be assigned weighted complete plays in the following way.

– $\Gamma \vdash \mathsf{new}\, x \,\mathsf{in}\, M : \theta$

  Let σ_1 correspond to $\Gamma, x : \mathsf{var} \vdash M : \theta$ and let $\mathit{cell} : \Gamma, \mathsf{var}$ be defined by

$$\mathit{cell}(\, \iota\, (\mathsf{r}_\downarrow, \star)\, s) = 1,$$

  where s ranges over sequences matching

$$((\mathsf{c}^r, \star)(\mathsf{r}^r, 0))^* (\sum_{n=0}^{max} (\, (\mathsf{c}^w, n)(\mathsf{r}^w, \star)\, (\, (\mathsf{c}^r, \star)(\mathsf{r}^r, n)\,)^*\,))^*.$$

  The expression above captures the behaviour of a memory cell initialised to 0: initially $(\mathsf{c}^r, \star)$ is followed by $(\mathsf{r}^r, 0)$ and afterwards by (r^r, n), where n is the value occurring in the preceding move with tag c^w. Then the interpretation σ of $\mathsf{new}\, x \,\mathsf{in}\, M$ is specified as follows.

$$\sigma = \sigma_1[\mathit{cell}/x]$$

– $\Gamma \vdash \mathsf{let}\, x = M \,\mathsf{in}\, N : \theta_2$

  Let σ_1, σ_2 correspond to $\Gamma \vdash M : \theta_1$ and $\Gamma, x : \theta_1 \vdash N : \theta_2$ respectively. Then the interpretation σ of $\mathsf{let}\, x = M \,\mathsf{in}\, N$ is specified as follows.

$$\sigma = \sigma_2[\sigma_1/x]$$

Theorem 1 ([17,11,35]) *Suppose $\Gamma \vdash M_i : \theta$ $(i = 1, 2)$ are $\mathcal{L}$-terms and $\sigma_i : \Gamma, \theta$ are the corresponding interpretations (as specified above). Then $\Gamma \vdash M_1 \cong M_2 : \theta$ if and only if $\sigma_1 = \sigma_2$.*

5 Automata

We represent the weighted complete plays associated with terms $\Gamma \vdash M : \theta$ using a special class of probabilistic automata, called (Γ, θ)-automata. These are based on the classical notion of Rabin [37] and can be viewed as a special instance of the so-called *reactive model* of probabilistic choice [45]. We write $\mathcal{S}(X)$ for the set of probability subdistributions on X.

Definition 7 A probabilistic automaton is a tuple $\mathcal{A} = (\Sigma, Q, i, \delta, F)$, where

- Σ is a finite alphabet;
- Q is a finite set of states;
- $i \in Q$ is the initial state;
- $\delta : Q \times \Sigma \to \mathcal{S}(Q)$ is a transition function;
- $F \subseteq Q$ is the set of final states.

Given $f \in F$ and $w = w_1 \cdots w_n \in \Sigma^n$ let $\mathcal{A}^f(w)$ be the sum of weights of all paths from i to f, i.e.

$$\mathcal{A}^f(w) = \sum_{\substack{(q_0,\cdots,q_n)\in Q^{n+1} \\ q_0=i,\, q_n=f}} \prod_{i=1}^{n} \delta(q_{i-1}, w_i)(q_i).$$

The automata used to interpret terms will be probabilistic automata with extra structure. More precisely, transitions corresponding to the environment will be *deterministic*, while transitions for program-moves will be *generative*. The latter means that the sum of weights associated with outgoing transitions from program-states will never exceed 1. Also, for technical convenience, the automata will not read initial moves and, if $\theta \equiv \beta$, the final moves will also be suppressed. In the latter case, the information needed to recover them will be provided in a final state in the form of a subdistribution.

As our alphabet we shall use the set $\mathcal{T}_{\Gamma,\theta} \times \mathbb{V}$, which contains all non-initial moves (Definition 4). Recall from Definition 2 that $\mathcal{T}_{\Gamma,\theta}$ contains tags which intuitively represent term evaluation, calls/returns associated with the resultant value as well as calls/returns related to the indeterminate components of a term. Elements of $\mathbb{V} = \{\star\} \cup \{0, \cdots, max\}$ are then used to represent parameters occurring in the respective calls/returns.

Definition 8 A (Γ, θ)-automaton is a tuple (Q, i, δ, F, η) such that

- $(\mathcal{T}_{\Gamma,\theta} \times \mathbb{V}, Q, i, \delta, F)$ is a probabilistic automaton;
- the set of states is partitioned into Q^O and Q^P (that will be used for processing O- and P-moves respectively);
- $i \in Q^P$;
- δ is partitioned into δ^O and δ^P such that:
  - (O-**determinacy**) if $\delta^O(q, m)(q') \neq 0$ then $m \in M^O$, $q \in Q^O$, $q' \in Q^P$ and $\delta^O(q, m)(q') = 1$,

- (P-**generativity**) if $\delta^P(q,m)(q') \neq 0$ then $m \in M^P$, $q \in Q^P$, $q' \in Q^O$ and
$$\sum_{m \in M^P, q'' \in Q} \delta^P(q,m)(q'') \leq 1;$$

- F and η are subject to the following conditions.
  - If $\theta \equiv \beta$ then $F \subseteq Q^P, \eta : F \to \mathcal{S}(\mathbb{V}_\beta)$ and for all $f \in F$
$$\sum_{q,m} \delta(f,m)(q) + \sum_{v \in \mathbb{V}_\beta} \eta(f)(v) \leq 1.$$
  - If $\theta \not\equiv \beta$ then $F \subseteq Q^O$ (η will be immaterial in this case).

Next we spell out what it means for (Γ, θ)-automata to represent weighted complete plays.

Definition 9 Let $\sigma : \Gamma, \theta$ and let $\{\mathcal{A}_\iota\}_{\iota \in I_\Gamma}$ be a collection of (Γ, θ)-automata. We shall say that $\{\mathcal{A}_\iota\}_{\iota \in I_\Gamma}$ represents σ if the conditions below are satisfied for all $\iota \in I_\Gamma$.

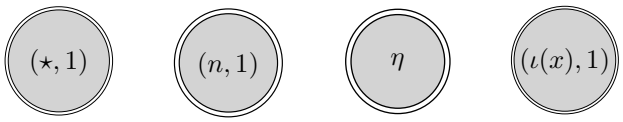
- If $\theta \equiv \beta$ then $\sigma(\iota s(\mathsf{r}_\downarrow, v)) = \sum_{f \in F} \mathcal{A}_\iota^f(s) \cdot \eta(f)(v)$ for all $\iota\, s\, (\mathsf{r}_\downarrow, v) \in \mathcal{C}_{\Gamma,\theta}$.
- If $\theta \not\equiv \beta$ then $\sigma(\iota\, s) = \sum_{f \in F} \mathcal{A}_\iota^f(s)$ for all $\iota\, s \in \mathcal{C}_{\Gamma,\theta}$.

Finally we show how to construct collections of automata representing the weighted complete plays induced by terms of $\mathcal{L}$.

Theorem 2 *Let $\Gamma \vdash M : \theta$ be a $\mathcal{L}$-term and $\sigma : \Gamma, \theta$ the corresponding set of weighted complete plays defined in Section 4. Then there exists a collection of (Γ, θ)-automata representing σ.*

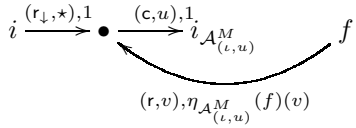
Proof We shall denote the collection of automata referred to in the statement of the Theorem by $\{\mathcal{A}_\iota^{\Gamma \vdash M}\}_{\iota \in I_\Gamma}$ and will most often omit the typing context $\Gamma \vdash$ for brevity. The construction of this family will proceed by structural induction for $\mathcal{L}$-terms.

- The automata $\mathcal{A}_\iota^{()}$, $\mathcal{A}_\iota^n$, $\mathcal{A}_\iota^{\mathsf{coin}[0:p_0,\cdots,max:p_{max}]}$ and $\mathcal{A}_\iota^{x:\beta}$ are given by the respective one-state automata listed below, where $\eta : \{0, \cdots, max\} \to [0,1]$ is the subdistribution $\eta(n) = p_n$.



  We use double circling to indicate that a state is final (the states above are also initial, which we highlight by filling their background). Additionally, whenever the distribution η is relevant, we shall write it inside the final state. In each of the cases considered above, the empty word will carry weight 1, so the given automata do represent the requisite complete plays from Section 4 as stipulated in Definition 9.

- For $\mathcal{A}_\iota^{\mathsf{case}(x)[M_0,\cdots,M_{max}]}$ we can simply take $\mathcal{A}_\iota^{M_{\iota(x)}}$.
- The automata $\mathcal{A}_\iota^{fx}$, $\mathcal{A}_\iota^{!x}$, $\mathcal{A}_\iota^{x:=y}$, $\mathcal{A}_\iota^{\mathsf{mkvar}(x,y)}$ can be constructed directly from the definition of the corresponding complete plays given in Section 4.
- $\mathcal{A}_\iota^{\lambda x^{\beta_1}.M}$ can be obtained as follows, where u, v, f range over $\mathbb{V}_{\beta_1}$, $\mathbb{V}_{\beta_2}$ and $F_{\mathcal{A}^M_{(\iota,u)}}$ respectively, and $\bullet$ is the only final state in the new automaton.



Note that $(\mathsf{r}_\downarrow,\star)$ is accepted with probability 1 to correspond to the requirement $\sigma(\ \iota\,(\mathsf{r}_\downarrow,\star)\ ) = 1$ from Section 4. Moreover, the new transition between f and $\bullet$ allows us to model the desired multiplication effect on weights:

$$\sigma(\ \iota\,(\mathsf{r}_\downarrow,\star)\,(\mathsf{c},u_1)s_1(\mathsf{r},v_1)\ \cdots\ (\mathsf{c},u_k)s_k(\mathsf{r},v_k)\ ) = \prod_{i=1}^{k}\sigma'(\ (\iota,u_i)\,s_i\,(\mathsf{r}_\downarrow,v_i)\ ).$$

- We proceed to discuss the automata-theoretic account of the $\sigma_2[\sigma_1/x]$ construction, which is used to interpret the remaining syntactic constructs $\mathsf{let}\,x\,=\,M\,\mathsf{in}\,N$ and $\mathsf{new}\,x\,\mathsf{in}\,M$. The argument will be divided into two cases: $\theta \equiv \beta$ and $\theta \not\equiv \beta$. The former can be treated more directly by concatenation, while for the latter we shall use a product-like construction. Suppose $\sigma_1 : \Gamma, \theta$ and $\sigma_2 : \Gamma + \{x : \theta\}, \theta'$. Let the families $\{\mathcal{A}^1_\iota\}_\iota$, $\{\mathcal{A}^2_{\iota,v}\}_{\iota,v}$ represent σ_1 and σ_2 respectively. Let us fix ι and assume that $\mathcal{A}^1_\iota = (Q_1, i_1, \delta_1, F_1, \eta_1)$ and $\mathcal{A}^2_{\iota,v} = (Q_2^v, i_2^v, \delta_2^v, F_2^v, \delta_2^v)$, where $v \in \mathbb{V}_\theta$.
  - Suppose $\theta \equiv \beta$. In this case the corresponding sequences s_1 (here we refer to s_1 from Definition 6) will not contain the c, r tags. Hence, the pattern of tags that arises during synchronisation degenerates to

$$\iota\ X_1^*\ S\ X_2^*\ \mathsf{r}_\downarrow\ (\mathsf{c}\ X_2^*\ \mathsf{r})^*.$$

    Consequently, in order to simulate it, it suffices to link final states of $\mathcal{A}^1_\iota$ with appropriate copies of $\mathcal{A}^2_{\iota,v}$. To that end we define a (Γ,θ)-automaton $\mathcal{A}' = (Q', i', \delta', F', \eta')$ as follows, where $\bigoplus$ stands for the disjoint sum of sets.
    - $Q' = Q_1 + \bigoplus_{v\in\mathbb{V}_\beta} Q_2^v$
    - $i' = i_1$
    - $\delta' = \delta_1 + \delta_{extra} + \bigoplus_{v\in\mathbb{V}_\beta}\delta_2^v$

$$\forall_{f\in F_1, v\in\mathbb{V}_\beta, q'\in Q_2^v}\quad \delta_{extra}(f,m)(q') = \eta(f)(v)\cdot\delta_2^v(i_2^v,m)(q')$$

    - $F' = F_1 + \bigoplus_{v\in\mathbb{V}_\beta} F_2^v$

- $\eta' = \eta_{extra} + \bigoplus_{v \in \mathbb{V}_\beta} \eta_2^v$

  η_{extra} is defined only if θ' is a base type. Then we have:

$$\forall_{f \in F_1, u \in \mathbb{V}_{\theta'}} \quad \eta_{extra}(f)(u) = \sum_{v \in \mathbb{V}_\beta, i_2^v \in F_2^v} \eta_1(f)(v) \cdot \eta_2^v(i_2^v)(u).$$

  Note that by linking the automata through $\delta_{extra}(f, m)(q') = \eta(f)(v) \cdot \delta_2^v(i_2^v, m)(q')$ we obtain the requisite multiplication on weights from Definition 6. No erasure is necessary in this case because the alphabet $\mathcal{B}$ corresponds to initial and final moves, which are not represented explicitly in automata.

– Now we tackle the other case ($\theta \not\equiv \beta$). We shall simply write $(Q_2, i_2, \delta_2, F_2, \eta_2)$ instead of $(Q_2^v, i_2^v, \delta_2^v, F_2^v, \eta_2^v)$, as $v = \star$ in this case. Next we introduce the product construction that will allow the two automata to engage in interaction according to the definition of $\sigma_2[\sigma_1/x]$ (and synchronize on moves in $\mathcal{B}$). First we apply the relabelling $[t_x/t, (S, v)/(\mathsf{r}_\downarrow, v)]$ to $\mathcal{A}_\iota$. In order to allow for synchronisation on $(S, \star)$ we shall add a new state i_2^- along with a transition $i_2^- \xrightarrow{(S,\star),1} i_2$. The automaton $\mathcal{A}^{|||_\mathcal{B}} = (Q, i, \delta, F, \eta)$ capturing the requisite interactions is defined as follows.

  - $\Sigma = (\mathcal{T}_{\Gamma \cup \{x:\theta\}, \theta'} \times \mathbb{V}) \cup \{(S, \star)\}$
  - $Q = Q_1 \times (Q_2 \cup \{i_2^-\})$
  - $i = (i_1, i_2^-)$
  - δ is defined below, where we let m, b range over $\mathcal{T}_{\Gamma,\theta'} \times \mathbb{V}$ and $\{\mathsf{c}_x, \mathsf{r}_x, \mathsf{c}_x^r, \mathsf{r}_x^r, \mathsf{c}_x^w, \mathsf{r}_x^w\} \times \mathbb{V}$ respectively and $q_i, q_i' \in Q_i$ $(i = 1, 2)$.

$$\begin{aligned}
\delta((q_1, i_2^-), m)(q_1', i_2^-) &= \delta_1(q_1, m)(q_1') \\
\delta((q_1, i_2^-), (S, \star))(q_1', i_2) &= \delta_1(q_1, (S, \star))(q_1') \\
\delta((q_1, q_2), m)(q_1', q_2) &= \delta_1(q_1, m)(q_1') \\
\delta((q_1, q_2), m)(q_1, q_2') &= \delta_2(q_2, m)(q_2') \\
\delta((q_1, q_2), b)(q_1', q_2') &= \delta_1(q_1, b)(q_1') \cdot \delta_2(q_2, b)(q_2')
\end{aligned}$$

  - $F = F_1 \times F_2$
  - $\eta(f_1, f_2) = \eta_2(f_2)$

  Note that in the last clause for δ the probabilities are multiplied (in fact one of them will be equal to 1), as in the definition of $\sigma_2[\sigma_1/x]$. Accordingly, $\mathcal{A}^{|||_\mathcal{B}}$ will be consistent with Definition 6 in the following sense.

  - If $\theta' \equiv \beta$ for any $f \in F$ we shall have $(\mathcal{A}^{|||_\mathcal{B}})^f(s) \cdot \eta(f)(v) \neq 0$ only if $\iota\, s\, (\mathsf{r}_\downarrow, v) = s_1 \not\downarrow s_2$ for compatible s_1, s_2. This is because the constituent automata have to synchronise exactly on elements of $\mathcal{B}$. Conversely, if $\iota\, s\, (\mathsf{r}_\downarrow, v) = s_1 \not\downarrow s_2$ for compatible s_1, s_2 then $\sum_{f \in F} (\mathcal{A}^{|||_\mathcal{B}})^f(s) \cdot \eta(f)(v) = \sigma_1(s_1) \cdot \sigma_2(s_2)$, because during synchronisation the weights on synchronised transitions are multiplied.

- Similarly, if $\theta' \not\equiv \beta$, for any $f \in F$, we have $(\mathcal{A}^{|||_{\mathcal{B}}})^f(s) \neq 0$ only if $\iota\, s = s_1 \nleftrightarrow s_2$ and $\sum_{f \in F} (\mathcal{A}^{|||_{\mathcal{B}}})^f(s) = \sigma_1(s_1) \cdot \sigma_2(s_2)$, as required.

However, $\mathcal{A}^{|||_{\mathcal{B}}}$ is not a (Γ, θ')-automaton yet, due to the presence of transitions involving tags associated with x as well as S. We can remove them by turning them into ϵ-transitions (thus implementing the erasure $\backslash\mathcal{B}$ from the definition of $\sigma_2[\sigma_1/x]$) and applying one of the standard ϵ-elimination algorithms for weighted automata, e.g. [32]. For completeness we give the partitioning of states in $\mathcal{A}^{|||_{\mathcal{B}}}$ into O-states and P-states:

$$\begin{aligned} Q^O &= Q_1^O \times (\{i_2^-\} \cup Q_2^O), \\ Q^P &= Q_1^O \times Q_2^P \;\cup\; Q_1^P \times (\{i_2^-\} \cup Q_2^O). \end{aligned}$$

Other states can be removed as they are not reachable from (i_1, i_2^-).

Now that we have an automata-theoretic construction corresponding to $\sigma_2[\sigma_1/x]$, the theorem for $\mathsf{let}\, x = M \,\mathsf{in}\, N$ and $\mathsf{new}\, x \,\mathsf{in}\, M$ follows by applying the construction as explained in Section 4.

□

5.1 Some special cases more concretely

The syntax of $\mathcal{L}$ that we introduced was very economical and many commonly used idioms had to be compiled into the language via the let construct. Next we give four constructions that cover selected special cases. They yield equivalent automata to those constructed by following the general recipe given above, but are more suitable for implementation and also more intuitive.

– Let $(f : \beta_1 \to \beta_2) \in \Gamma$ and suppose $\mathcal{A}_\iota^M = (Q, i, \delta, F, \eta)$ corresponds to $\Gamma \vdash M$. Then the automaton for $\mathsf{let}\, x = M \,\mathsf{in}\, fx$ (normally written as fM) can be constructed as follows.

$$\begin{aligned} Q' &= Q + \{q\} + \{f_v \mid v \in \mathbb{V}_{\beta_2}\} \\ i' &= i \\ \delta' &= \delta + \{f \xrightarrow{(\mathsf{c}_f, u),\, \eta(f)(u)} q \mid f \in F,\, u \in \mathbb{V}_{\beta_1}\} + \{q \xrightarrow{(\mathsf{r}_f, v),1} f_v \mid v \in \mathbb{V}_{\beta_2}\} \\ F' &= \{f_v \mid v \in \mathbb{V}_{\beta_2}\} \\ \eta' &= \{(f_v, (v, 1)) \mid v \in \mathbb{V}_{\beta_2}\} \end{aligned}$$

– Let $(x : \mathsf{var}) \in \Gamma$ and suppose $\mathcal{A}_\iota = (Q, i, \delta, F, \eta)$ corresponds to $\Gamma \vdash M : \mathsf{int}$. Then the automaton for $\mathsf{let}\, y = M \,\mathsf{in}\, x := y$ (typically written as $x := M$) can be obtained as follows.

$$\begin{aligned} Q' &= Q + \{q\} + \{f_\star\} \\ i' &= i \\ \delta' &= \delta + \{f \xrightarrow{(\mathsf{c}_x^w, u),\, \eta(f)(u)} q \mid f \in F,\, u \in \mathbb{V}_{\mathsf{int}}\} + \{q \xrightarrow{(\mathsf{r}_x^w, \star),1} f_\star\} \\ F' &= \{f_\star\} \\ \eta' &= \{(f_\star, (\star, 1))\} \end{aligned}$$

– Suppose $\mathcal{A}_{\iota,\star} = (Q, i, \delta, F, \eta)$ corresponds to $\Gamma, x : \mathsf{var} \vdash M : \theta$. Then the automaton $\mathcal{A}'_\iota$ for $\Gamma \vdash \mathsf{new}\, x\, \mathsf{in}\, M : \theta$ can be constructed by creating $(\max +1)$ copies of M (tracking the intermediate values of x), in which transitions related to x are appropriately redirected or erased. Formally

$$\mathcal{A}_\iota = (Q \times \{0, \cdots, max\}, (i, 0), F \times \{0, \cdots, max\}, \delta', \eta'),$$

where

$$\delta'((q_1, i), m)(q_2, j) = \begin{cases} \delta(q_1, m)(q_2) & m = (t, v),\ \ i = j,\ \ t \neq \mathsf{r}^r_x, \mathsf{c}^w_x \\ \delta(q_1, m)(q_2) & m = (t, v),\ \ v = j,\ \ t = \mathsf{r}^r_x, \mathsf{c}^w_x \\ 0 & \text{otherwise} \end{cases}$$

$$\eta'(f, i) = \eta(f) \qquad f \in F,\ i \in \{0, \cdots, max\}$$

Now it suffices to designate all transitions involving tags $\mathsf{c}^r_x, \mathsf{r}^r_x, \mathsf{c}^w_x, \mathsf{r}^w_x$ as ϵ-transitions and to perform ϵ-removal.

– Let $(f : \mathsf{unit} \to \mathsf{int}), (g : \mathsf{unit} \to \mathsf{unit}) \in \Gamma$, $(x : \mathsf{unit}) \notin \Gamma$ and suppose $\mathcal{A}^1_\iota = (Q_1, i_1, \delta_1, F_1, \eta_1)$ and $\mathcal{A}^2_\iota = (Q_2, i_2, \delta_2, F_2, \eta_2)$ correspond to $\Gamma \vdash M : \mathsf{int}$ and $\Gamma \vdash N : \mathsf{unit}$. Then the automaton for $\mathsf{let}\, f = \lambda x^{\mathsf{unit}}.M\, \mathsf{in}\, \mathsf{let}\, g = \lambda x^{\mathsf{unit}}.N\, \mathsf{in}\, \mathsf{while}\, f()\, \mathsf{do}\, g()$ (which corresponds to $\mathsf{while}\, M\, \mathsf{do}\, N$) can be constructed by eliminating ϵ-transitions in the automaton given below.

$$\begin{aligned} Q' &= Q_1 + Q_2 \\ i' &= i_1 \\ \delta' &= \delta_1 + \delta_2 + \{f_1 \xrightarrow{\epsilon, \sum_{n=1}^{max} \eta_1(f_1)(n)} i_2 \mid f_1 \in F_1\} + \{f_2 \xrightarrow{\epsilon, \eta_2(f_2)(\star)} i_1 \mid f_2 \in F_2\} \\ F' &= F_1 \end{aligned}$$

in which $\eta'(f_1)(\star) = \eta_1(f)(0)$ for any $f_1 \in F'$.

5.2 Case studies

Next we present a series of case studies in which the automata have been generated in an automated way by our tool APEX. In order to reduce the number of transitions, consecutive transitions tagged c_x and r_x respectively (the same applies to $\mathsf{c}^r_x, \mathsf{r}^r_x$ and $\mathsf{c}^w_x, \mathsf{r}^w_x$) are lumped together. In order to retrieve the correspondence with game semantics, the reader should refer to the table below.

game semantics	automata
$\bullet_0 \xrightarrow{(\mathsf{c}_x, \star)} \bullet \xrightarrow{(\mathsf{r}_x, \star)} \bullet_1$	$\bullet_0 \xrightarrow{run_x} \bullet_1$
$\bullet_0 \xrightarrow{(\mathsf{c}_x, u)} \bullet \xrightarrow{(\mathsf{r}_x, \star)} \bullet_1$	$\bullet_0 \xrightarrow{x(u)} \bullet_1$
$\bullet_0 \xrightarrow{(\mathsf{c}_x, \star)} \bullet \xrightarrow{(\mathsf{r}_x, v)} \bullet_1$	$\bullet_0 \xrightarrow{v_x} \bullet_1$
$\bullet_0 \xrightarrow{(\mathsf{c}_x, u)} \bullet \xrightarrow{(\mathsf{r}_x, v)} \bullet_1$	$\bullet_0 \xrightarrow{(u,v)_x} \bullet_1$
$\bullet_0 \xrightarrow{(\mathsf{c}^w_x, u)} \bullet \xrightarrow{(\mathsf{r}^w_x, \star)} \bullet_1$	$\bullet_0 \xrightarrow{write(u)_x} \bullet_1$
$\bullet_0 \xrightarrow{(\mathsf{c}^r_x, \star)} \bullet \xrightarrow{(\mathsf{r}^r_x, v)} \bullet_1$	$\bullet_0 \xrightarrow{v_x} \bullet_1$

For flexibility, we use a variety of int- and var-types with different ranges. In the code we shall write, for example, `int%5` or `var%7` to reflect that. We also use arrays: an array a of size N is translated into $\mathcal{L}$ by declaring N variables using new. Assuming the variables are $x_0, \cdots, x_{N-1}$, array accesses $a[M]$ can then be desugared to $\mathsf{case}(M)[x_0, \cdots, x_{N-1}]$. Note that, because we work with finite datatypes, various arithmetic operations can be handled via $\mathsf{case}(\cdots)[\cdots]$. Subdistributions associated with final nodes will be written inside nodes. In the figures we use 0 to stand for $\star$, so $(0, 1)$ denotes the distribution returning $\star$ (termination) with probability 1.

6 Experimental Results

We describe the architecture of the tool APEX, which generates automata by following the constructions described earlier. Further, we give experimental results with statistics on running time and automata sizes. We profile the different automata operations involved to identify the automata operations that are most critical for performance.

From a given program, APEX builds an automaton and writes it to disk. The automaton can then be viewed as a graph or compared for equivalence to another automaton. The architecture of APEX is split into a language front-end, implemented in OCaml, which translates the program into a sequence of operations on automata, and a back-end automata library, implemented in C, which represents automata as adjacency graphs.

We have run APEX on a collection of benchmark case studies using a 3.07 GHz workstation with 48GB of memory and a time-out limit of 10 minutes. The benchmarks consist of probabilistic programs studied in previous publications [28,24], and one new case study, the Crowds protocol [39]. In this subsection we briefly review the case studies and give performance statistics.

Crowds protocol. Assume that there is a sender who would like to transmit a message to a receiver. The sender is a member of a crowd of participants. The sender is aware that there are corrupt members—potentially even the receiver—who seek to intercept the identity of the sender. The Crowds protocol [39] forwards the message via a random route to the receiver, so that the sender can plead probable innocence to be the originator of the message. As crowd memberships may change, the protocol may run for several rounds R leading to path reformulations.

The algorithm works as follows (we provide the APEX code in Figure 4). Each honest crowd member probabilistically either forwards the message to another randomly chosen member or sends it directly to the recipient. It is assumed that, when corrupt crowd members receive a message from another member, they just announce the identity of the other member. In this way, the protocol dynamically builds a random route to the recipient. The identity of the sender cannot be determined with absolute certainty. However, if a corrupt member receives a message along this route, it leaks the identity of the

```
const USERS := 30;
const CORRUPT := 3;
const ROUNDS := 2;

announce : int%USERS -> unit |-
  var%2 corrupt[USERS];
  var%USERS i := USERS - 1;
  while(USERS - i < CORRUPT) do {
    corrupt[i] := 1;    // initialise corrupt users
    i := i - 1;
  } ;
  var%(ROUNDS+1) round :=0;
  while(round < ROUNDS) do {
   var%USERS current;
   var%USERS previous;
   var%2 done;
   // create path
   while(done = 0) do {
     if( corrupt[current] = 1 ) then {
        announce(previous);
        done := 1
      } else {
      if(coin[0:1/4,1:3/4]) then { // forward or not
              previous := current;  // save the sender
             current := rand[USERS];   // forward to whom ?
       } else {
              done := 1
       }   //end of if
     }  // end of if
  }; // end of current path
  round := round + 1;
}:unit // end of path reformulations
```

Fig. 4 Crowds protocol.

member it has received the message from. We assume that corrupt members can determine when two different messages originate from the same sender, e.g., due to similarity in content. Thus over several path reformulations the more often the identity of a particular member is announced, the likelier he is to be the actual sender of given class of messages. Thus the concept of anonymity can be quantified probabilistically. Different degrees of probabilistic innocence exist: one variant considers whether the sender has higher probability than any other member to be announced, another variant considers the probability for the sender to be announced is larger than zero.

Shmatikov [43] modelled the Crowds protocol in the probabilistic model checker PRISM using one counter for each member. The counter variables recorded the number of times a member has been announced by a corrupt member. Anonymity could thus be quantified by computing the probability that a certain relation between counter values held, e.g., the probability that the identity of the sender is announced more often than any other member. The downside of this modelling approach was that the counter values led to an exponential blow up in the state space.

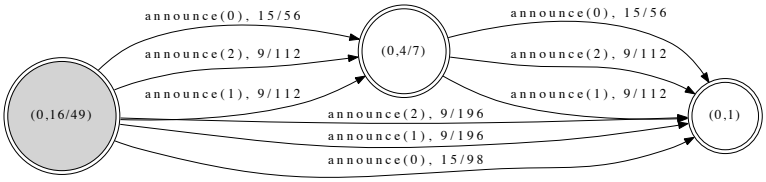


Fig. 5 Crowds: 3 members, 1 corrupt.

In APEX, however, the Crowds protocol can be conveniently modeled without instrumenting the model with counters. Instead the announcements of members are labels in the automaton. Figure 5 shows the automaton obtained with 3 members and 1 corrupt member.

In this example game semantics is used for automatic extraction of an abstraction of the protocol that captures exactly the information needed to investigate its anonymising power. The sizes of the extracted models grow only linearly in the number of rounds and members. Thus, model-checking them will be more efficient than model checking the full state space, as could be done with other tools. Here our goal is not to prove equivalence with a specification, but to demonstrate how game semantics can be used as a technique for compact model building.

Dining cryptographers [28]. A classical example illustrating anonymity is that of the Dining Cryptographers protocol [8], which is a popular case study that very nicely illustrates the benefits of game semantics. Imagine that a certain number of cryptographers are sharing a meal at a restaurant around a circular table. At the end of the meal, the waiter announces that the bill has already been paid. The cryptographers conclude that it is either one of them who has paid, or the organisation that employs them. They resolve to determine which of the two alternatives is the case, with the proviso that for the former the identity of the payer should remain secret.

A possible solution goes as follows. A coin is placed between each pair of adjacent cryptographers. The cryptographers flip the coins and record the outcomes for the two coins that they can see, i.e., the ones that are to their immediate left and right. Each cryptographer then announces whether the two outcomes agree or disagree, except that the payer (if there is one) says the opposite. When all cryptographers have spoken, they count the number of disagreements. If that number is odd, then one of them has paid, and otherwise, their organisation has. Moreover, if the payer is one of the cryptographers, then no other cryptographer is able to deduce who it is.

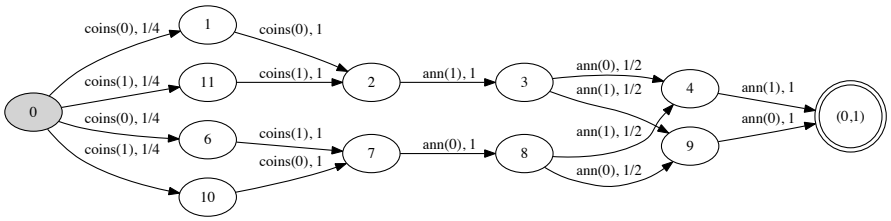
We show how to model the Dining Cryptographers protocol in our probabilistic programming language, and verify anonymity using APEX. Let us consider the case of three cryptographers, numbered 1, 2, and 3, from the point of view of the first cryptographer; the open program in Figure 6 enacts the protocol. This program has a local variable `whopaid` that can be set

```
const N:= 4;  // number of cryptographers = N-1
coins: int%2-> unit, ann: int%2 -> unit |-
  var%N whopaid :=2;
  var%2 first :=coin;
  var%2 right :=first ;
  var%2 left;
  var%2 parity;

  var%N i:=1;
  while (i) do {
    if (i=N-1) then {
      left:=first;
    }
    else {
      left:=coin[0:1/2, 1:1/2];
    }
    if (i=1) then {
      coins(right); coins(left)
    }
    ann ( (left=right)+(whopaid=i) ) ;
    right := left;
    i:=i+1
  } : unit
```

Fig. 6 Dining cryptographers.

to 2 or 3, to model the appropriate situation. All events meant to be visible to the first cryptographer, i.e., the outcomes of his two adjacent coins, as well as the announcements of all cryptographers, are made observable by respectively calling the free procedures `coins` and `ann` with the corresponding values. Thus, the observable behaviour of the program will correspond to the perspective of the first cryptographer. (Probabilistic) anonymity with respect to the first cryptographer then corresponds to the assertion that the program in which `whopaid` has been set to 2 is contextually equivalent to the program in which it is set to 3. From the code in Figure 6, APEX produces the following probabilistic automaton.



The variable `whopaid` is set to 2 in Figure 6. It turns out that setting `whopaid` to 3 yields precisely the same automaton. The two programs are therefore contextually equivalent, which establishes anonymity of the protocol with three cryptographers.

One can easily investigate larger instances of the protocol by changing the value of the constant N in line 1. It is interesting to note that the size of the state space of the automata grows only linearly with the number of cryptographers, despite the fact that the raw cryptographers state space is ostensibly exponential (due to the set of possible outcomes of the coin flips). Note however that this complexity is in our case reflected in the number of paths of the automata rather than in the number of their states. In fact, in our experiments (see Table 1), the state spaces of the intermediate automata as well as the total running times grew linearly as well. This unexpected outcome arose partly from APEX's use of bisimulation reduction throughout the construction, in which most symmetries were factored out. The automaton for 20 cryptographers is shown below to illustrate that the automaton size grows linearly with the number of cryptographers:

We can also show that probabilistic anonymity fails when the coins are biased. Note that thanks to full abstraction, whenever two probabilistic programs are not equivalent, their corresponding probabilistic automata will disagree on the probability of accepting some particular word. This word, whose length need be at most the total number of states of both automata, can be thought of as a counterexample to the assertion of equivalence of the original programs, and can be used to debug them. Using a counterexample word, we are able to track down the exact program locations that lead to inequivalence and also a scenario in the protocol where anonymity is breached. For example, if tossing 0 has probability $\frac{1}{3}$, APEX produces the following counterexample for the protocol instance with 4 cryptographers:

$$coins(1) \quad coins(1) \quad ann(1) \quad ann(1) \quad ann(0)$$

which gives a probability $\frac{1}{9}$ in the automaton where cryptographer 2 has paid and $\frac{2}{9}$ if cryptographer 3 has paid. Counterexample words are generated automatically using the techniques from [24].

The current version of APEX can analyse instances of up to 800 philosophers in the given time limit. (In previous experiments [28], instances of the protocol with up to 100 cryptographers could be handled before the tool would time out.)

The Dining Cryptographers protocol is a popular benchmark for evaluating frameworks for analysing anonymity and information hiding. One approach is to use probabilistic temporal-logic model checking. For example, the protocol has been used as a case study for the PRISM probabilistic model checker [27]. Van der Meyden and Su [44] use an algorithm for model checking a temporal logic with epistemic modalities. In this setting the specification can refer to subjective probabilities, permitting a more explicit formalisation of anonymity properties that in a pure temporal logic. Kacprzak *et al.* [22] have investigated the performance of epistemic model checking algorithms on this case study

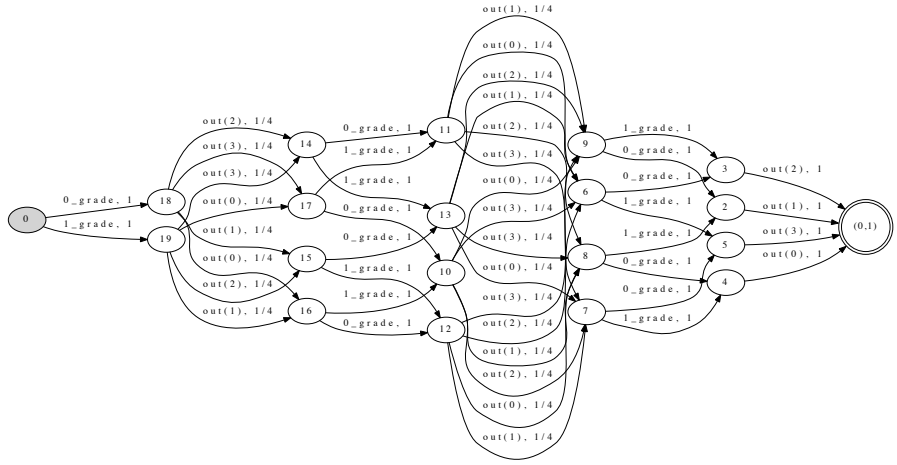


Fig. 7 Grades: specification automaton

using both BDD- and SAT-based approaches. The above papers consider only finite instances of the protocol. They report that the complexity of model checking grows exponentially with the number of cryptographers, and for almost all properties their experiments involve instances with fewer that 20 cryptographers.

Our approach does not directly formalise the knowledge and beliefs of different agents. Instead this is modelled indirectly via the ability of game semantics to handle programs with free identifiers, modelling what is visible to the environment. Consequently we formalise anonymity as a global property (equivalence in all contexts) rather than a collection of different temporal logic assertions. In this respect our work is similar to the process-algebraic approaches to verifying anonymity (see in particular [7,42]). As reported above, the automata we produce seem to scale linearly with the number of cryptographers. In essence we model the runs of the protocol as a language, and our automata are an exponentially compact representation of this language.

McIver and Morgan [30] prove the correctness of arbitrarily large instances of the Dining Cryptographers protocol via an approach integrating non-interference and stepwise refinement. Their proof is comprised of assertions in a Hoare-style program logic. Whilst being completely formal, to the best of our knowledge, the proof has not been automated.

Grades protocol. The grades protocol [24] gives a randomised algorithm to determine, for a given group of students, a sum of their grades, e.g. to compute the average, without revealing their individual grade in the process. The algorithm works as follows: Let $S \in \mathbb{N}$ be the number of students, and let $\{0, \cdots, G-1\}$ ($G \in \mathbb{N}$) be the set of grades. Define $N = (G-1) \cdot S + 1$. Further we assume that the students are arranged in a ring and that each pair of adjacent students shares a random integer between 0 and $N-1$. Thus a

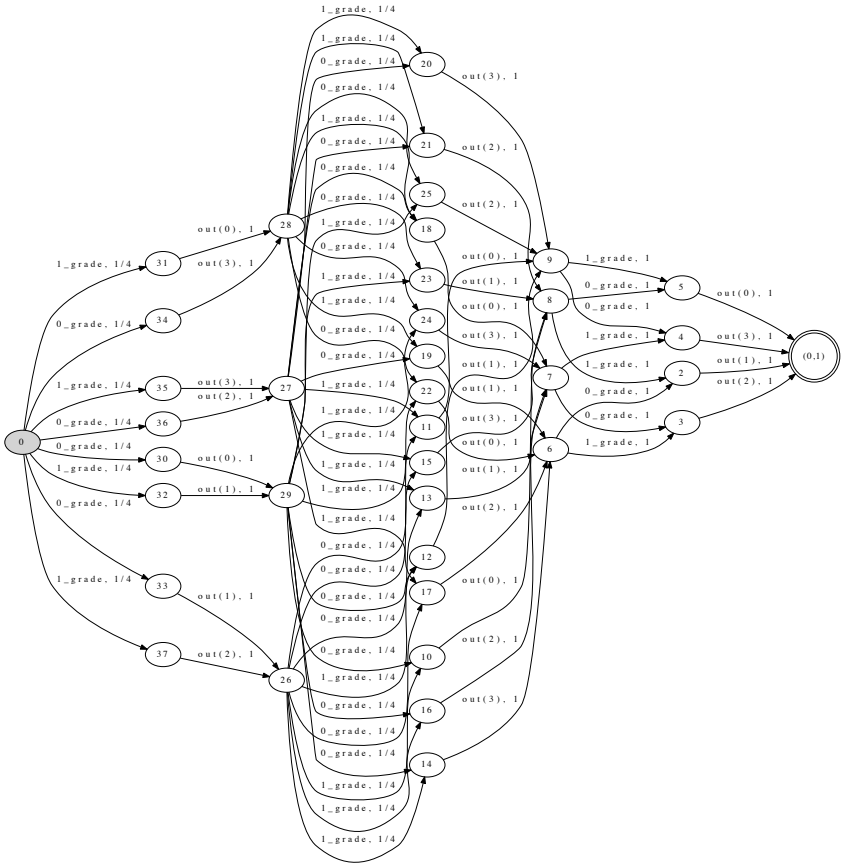


Fig. 8 Grades: implementation automaton

student shares a number l with the student on the left and a number r with the student on the right, respectively. Denoting the student's grade by g, the student announces the number $(g + l - r)\ mod\ N$. Because of the ring structure, each number will be reported twice, once as l and once as r, so the sum of all announcements (modulo N) will be the same as the sum of all grades. We can verify that only this sum can be gleaned from the announcements by an external observer, i.e. an observer who does not know about the individual grades. This correctness condition can be formalised by a specification, in which the students make random announcements subject to the condition that their sum equals the sum of their grades. This means effectively that the check for anonymity of the Grades protocol reduces to a language equivalence check of implementation automaton and specification automaton. This holds because we know that the specification automaton is anonymous and hence is the implementation if it is indistinguishable from the specification.

For two students and two grades, the automaton for the specification and the implementation, respectively, are shown in Figures 7 and 8.

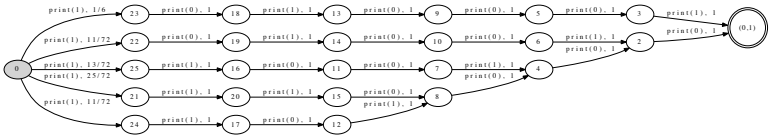
Hibbard's algorithm. We analyse the average shape of binary search trees generated by sequences of random insertions and deletions. In his seminal paper on random deletion in binary search trees, Hibbard [16] proved that: if $n+1$ items are inserted into an initially empty binary tree, in random order, and if one of those items (selected at random) is deleted, the probability that the resulting binary tree has a given shape is the same as the probability that this tree shape would be obtained by inserting n items into an initially empty tree, in random order.

For more than a decade it was subsequently believed that Hibbard's theorem in fact proved that trees obtained through arbitrary sequences of random insertions and deletions are automatically random, i.e., have shapes whose distribution is the same as if the trees had been generated directly using random insertions only; see [16,26]. It turns out that this intuition was wrong. In 1975, Knott showed that, although Hibbard's theorem establishes that $n+1$ random insertions followed by a deletion yield the same distribution on tree shapes as n insertions, we cannot conclude that a subsequent random insertion yields a tree whose shape has the same distribution as that obtained through $n+1$ random insertions [26].

As Jonassen and Knuth [21] point out, this result came as a surprise. In [25], they gave a counterexample (based on Knott's work) using trees having size no greater than three. Despite the small sizes of the trees involved and the small number of random operations performed, their presentation showed that the analysis at this stage is already quite intricate. This suggests a possible reason as to why an erroneous belief was held for so long: carrying out even small-scale experiments on discrete distributions is inherently difficult and error-prone. For example, it would be virtually impossible to carry out by hand Jonassen and Knuth's analysis for trees of size no greater than five (i.e., five insertions differ from five insertions followed by a deletion and then another insertion), and even if one used a computer it would be quite tricky to correctly set up a bespoke exhaustive search. With APEX such analyses can be carried out almost effortlessly. It suffices to write programs that implement the relevant operations and subsequently print the shape of the resultant tree, and then ask whether the programs are equivalent or not. As an example, we describe how to use APEX to reproduce Jonassen and Knuth's counterexample, i.e., three insertions differ from three insertions followed by a deletion and an insertion. Since APEX does not at present support pointers, we represent binary trees of size n using arrays of size $2n-1$, following a standard encoding (see, e.g., [10]): the left and right children of an i-indexed array entry are stored in the array at indices $2i+1$ and $2i+2$ respectively. It is then possible to write a short program that inserts three elements at random into a tree, then sequentially prints out the tree shape in breadth-first manner using a free printing procedure. The automaton for tree size 15 and 3 insertions is shown below.

Table 1 Experiments: running times and automata size.

benchmark		peak		final		time
case study	param	S	T	S	T	total
crypto	N 100	2728	119808	205	816	1.06
crypto	N 200	5428	479608	405	1616	7.17
crypto	N 500	13528	2999008	1005	4016	119.68
crypto	N 800	21628	7678408	1605	6416	597.31
crypto	N 900	-	-	-	-	-
crowd	U 10 R 2	1136	2648	3	147	0.03
crowd	U 10 R 3	1137	3147	4	276	0.05
crowd	U 20 R 1	8465	18466	2	112	0.14
crowd	U 20 R 2	8466	18488	3	317	0.37
crowd	U 20 R 3	8467	19319	4	596	0.76
crowd	U 40 R 1	65725	137726	2	232	2.36
crowd	U 40 R 2	65726	137768	3	657	9.15
crowd	U 40 R 3	-	-	-	-	-
herman	N 7	1725	3956	1	0	0.03
herman	N 9	5685	19908	1	0	0.14
herman	N 11	21801	122260	1	0	0.92
herman	N 13	86377	881252	1	0	8.93
herman	N 15	344489	7066164	1	0	153.04
herman	N 17	-	-	-	-	-
tree	S 15 I 3	72063	142905	26	94	3.50
tree	S 15 I 4	82683	329742	37	257	13.12
tree	S 27 I 3	-	-	-	-	-
grade-spec	S 20 G 3	4183	43624	1563	43624	0.11
grade-spec	S 20 G 4	6223	94245	2324	94245	0.24
grade-spec	S 20 G 5	8263	164106	3085	164106	0.40
grade-spec	S 100 G 3	100903	4282104	39803	4282104	14.42
grade-spec	S 100 G 4	151103	9543205	59604	9543205	29.67
grade-spec	S 100 G 5	201303	16884506	79405	16884506	50.84
grade-impl2	S 20 G 3	38749	536239	31203	536239	1.81
grade-impl2	S 20 G 4	82049	1554585	68444	1554585	7.17
grade-impl2	S 20 G 5	141349	3389931	120125	3389931	20.25
grade-impl2	S 100 G 3	4161709	64585119	3980003	64585119	347.22
grade-impl2	S 100 G 4	-	-	-	-	-



Statistics. Table 1 summarises running times and automata sizes obtained during our experiments. In the course of generating the final automaton, APEX generates intermediate automata. We record the peak automata size, i.e. the maximal number of states and transitions, respectively, of any (intermediate) automaton that was generated by APEX. These peak sizes may exceed the ones of the final automaton. An extreme example is the herman case study where the final automaton, shown in Figure 9, has only a single state and no transition.

Fig. 9 Herman's protocol: final automaton

While the intermediate automata grow exponentially with the parameters. The reason why only instances of up to $N = 17$ can be handled is the size of the intermediate automata. The table explores the parameter range which can still be handled within the timeout limit.

With typically over 98% of the running time, across all case studies, the new operator is the dominating semantic operation. As input, the new operator takes an automaton and a variable to be hidden. In the input automaton, the variable is essentially an uninterpreted function. In an exploration phase, the variable is interpreted as a state variable, and then subsequently its transitions are replaced by ε-transitions. Removing the labels of transitions by turning them into ε-transitions creates a lot of redundancy in the automaton, which makes many states bisimilar and creates an opportunity for state-space reduction. Therefore the automaton is passed to a bisimulation reduction routine before ε-transitions are removed. Running bisimulation before ε-removal is advantageous because ε-removal has cubic worst-case complexity in the number of states, while bisimulation reduction can be done in $m \log(n)$ time, where m is the number of edges and n the number of states.

Bisimulation reduction leads to a dramatic reduction in automata size, so that subsequent steps operate on smaller automata. Because bisimulation is applied to larger automata than the other steps (the peak number of transitions recorded in Table 1), bisimulation is typically the most expensive step which consumes at least 70% of the running time of the new operator (and hence also a significant portion of the overall running time).

The following table gives the relative runtime contribution of the different phases of the new operator in percent. Percentages are given per case study, as instances of the same case study show very similar behaviour:

case study	bisimulation	exploration	ε-removal
crowd	50%	25%	25%
crypto	72%	8%	20%
herman	71%	14%	15%
grade-spec	80%	12%	8%
grade-impl	78%	10%	12%

References

1. S. Abramsky. Algorithmic games semantics: a tutorial introduction. In H. Schwichtenberg and R. Steinbruggen, editors, *Proof and System Reliability*, pages 21–47. Kluwer Academic Publishers, 2002. Proceedings of the NATO Advanced Study Institute, Marktoberdorf.
2. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.
3. S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of CSL*, volume 1414 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 1997.

4. S. Abramsky and G. McCusker. Game semantics. In H. Schwichtenberg and U. Berger, editors, *Logic and Computation*. Springer-Verlag, 1998. Proceedings of the 1997 Marktoberdorf Summer School.
5. F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEDS. In *Proceedings of Computer Performance Evaluation (Tools)*, volume 1469 of *LNCS*, 1998.
6. M. Bernardo, R. Cleaveland, S. Sims, and W. Stewart. TwoTowers: A tool integrating functional and performance analysis of concurrent systems. In *Proceedings of FORTE*, volume 135 of *IFIP Conference Proceedings*, 1998.
7. M. Bhargava and C. Palamidessi. Probabilistic anonymity. In *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2005.
8. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.
9. F. Ciesinski and C. Baier. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proceedings of QEST*. IEEE Computer Society, 2006.
10. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.
11. V. Danos and R. Harmer. Probabilistic game semantics. *ACM Transactions on Computational Logic*, 3(3):359–382, 2002.
12. P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proceedings of PAPM-PROBMIV*, volume 2165 of *LNCS*, 2001.
13. D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *Proceedings of ICFP*, pages 143–156. ACM, 2010.
14. V. Hartonas-Garmhausen, S. Vale Aguiar Campos, and E. M. Clarke. ProbVerus: Probabilistic symbolic model checking. In *Proceedings of ARTS*, volume 1601 of *LNCS*, 1999.
15. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In *Proceedings of TACAS*, volume 1785 of *LNCS*, 2000.
16. T. N. Hibbard. Some combinatorial properties of certain trees with applications to searching and sorting. *J. ACM*, 9(1):13–28, 1962.
17. K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. *Theoretical Computer Science*, 221(1–2):393–456, 1999.
18. D. Hopkins, A. S. Murawski, and C.-H. L. Ong. A fragment of ML decidable by visibly pushdown automata. In *Proceedings of ICALP*, volume 6756 of *Lecture Notes in Computer Science*, pages 149–161. Springer, 2011.
19. J. Hurd. *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, 2002.
20. J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation*, 163(2):285–408, 2000.
21. A. T. Jonassen and D. E. Knuth. A trivial algorithm whose analysis isn't. *J. Comput. Syst. Sci.*, 16(3):301–322, 1978.
22. M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing bdd and sat based techniques for model checking chaum's dining cryptographers protocol. *Fundam. Inform.*, 72(1-3):215–234, 2006.
23. J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Proceedings of QEST*. IEEE Computer Society, 2005.
24. S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 526–540. Springer, 2011.
25. G. D. Knott. *Deletion in Binary Storage Trees*. PhD thesis, Stanford University, 1975. Computer Science Technical Report STAN-CS-75-491.
26. D. E. Knuth. Sorting and searching. In *Volume 3 of The Art of Computer Programming (first printing)*. Addison-Wesley, 1973.
27. M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

28. A. Legay, A. S. Murawski, J. Ouaknine, and J. Worrell. On automated verification of probabilistic programs. In *Proceedings of TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 2008.
29. A. McIver and C. Morgan. Abstraction and refinement in probabilistic systems. *SIGMETRICS Performance Evaluation Review*, 32(4):41–47, 2005.
30. A. McIver and C. Morgan. The thousand-and-one cryptographers. In *Reflections on the Work of C.A.R. Hoare*. Springer, 2010.
31. R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4(1):1–22, 1977.
32. M. Mohri. Generic e-removal and input e-normalization algorithms for weighted transducers. *Internation Journal of Foundations of Computer Science*, 13(1):129–143, 2002.
33. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
34. A. S. Murawski. Functions with local state: regularity and undecidability. *Theoretical Computer Science*, 338(1/3):315–349, 2005.
35. A. S. Murawski and J. Ouaknine. On probabilistic program equivalence and refinement. In *Proceedings of CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2005.
36. H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium of Logical Foundations of Computer Science*. Springer-Verlag, 1994. LNCS.
37. M. O. Rabin. Probabilistic automata. *Information and Control*, 6 (3):230–245, 1963.
38. M. O. Rabin. Probabilistic algorithms. In *Algorithms and Complexity: New Directions and Results*, pages 21–39. Academic Press, 1976.
39. M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.
40. J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J.C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North Holland, 1978.
41. D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5, 2011.
42. S. Schneider and A. Sidiropoulos. Csp and anonymity. In *ESORICS*, volume 1146 of *Lecture Notes in Computer Science*, pages 198–218. Springer, 1996.
43. V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3/4):355–377, 2004.
44. R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In *CSFW*, pages 280–. IEEE Computer Society, 2004.
45. R. J. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121(2):285–408, 1995.
46. H. L. S. Younes. Ymer: A statistical model checker. In *Proceedings of CAV*, volume 3576 of *LNCS*, 2005.