On the Complexity of the Equivalence Problem for Probabilistic Automata*

Stefan Kiefer[1], Andrzej S. Murawski[2], Joël Ouaknine[1], Björn Wachter[1], and James Worrell[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Department of Computer Science, University of Leicester, UK

Abstract. Deciding equivalence of probabilistic automata is a key problem for establishing various behavioural and anonymity properties of probabilistic systems. In recent experiments a randomised equivalence test based on polynomial identity testing outperformed deterministic algorithms. In this paper we show that polynomial identity testing yields efficient algorithms for various generalisations of the equivalence problem. First, we provide a randomized **NC** procedure that also outputs a counterexample trace in case of inequivalence. Second, we consider equivalence of probabilistic cost automata. In these automata transitions are labelled with integer costs and each word is associated with a distribution on costs, corresponding to the cumulative costs of the accepting runs on that word. Two automata are equivalent if they induce the same cost distributions on each input word. We show that equivalence can be checked in randomised polynomial time. Finally we show that the equivalence problem for probabilistic visibly pushdown automata is logspace equivalent to the problem of whether a polynomial represented by an arithmetic circuit is identically zero.

1 Introduction

Probabilistic automata were introduced by Michael Rabin [20] as an extension of deterministic finite automata. Nowadays probabilistic automata, together with associated notions of refinement and equivalence, are widely used in automated verification and learning. Two probabilistic automata are said to be equivalent if each word is accepted with the same probability by both automata. Checking two probabilistic automata for equivalence has been shown to becrucial for efficiently establishing various behavioural and anonymity properties of probabilistic systems, and is the key algorithmic problem underlying the APEX tool [18, 16, 12].

It was shown by Tzeng [27] that equivalence for probabilistic automata is decidable in polynomial time. By contrast, the natural analog of language inclusion, that one automaton accepts each word with probability at least as great as

* Research supported by EPSRC grant EP/G069158. The first author is supported by a postdoctoral fellowship of the German Academic Exchange Service (DAAD).

another automaton, is undecidable [6] even for automata of fixed dimension [4]. It has been pointed out in [8] that the equivalence problem for probabilistic automata can also be solved by reducing it to the minimisation problem for weighted automata and applying an algorithm of Schützenberger [23].

In [12] we suggested a new *randomised* algorithm which is based on *polynomial identity testing*. In our experiments [12] the randomised algorithm compared well with the Schützenberger-Tzeng procedure on a collection of benchmarks. In this paper we further explore the connection between polynomial identity testing and the equivalence problem of probabilistic automata. We show that polynomial identity testing yields efficient algorithms for various generalisations of the equivalence problem.

In Section 3 we give a new randomised **NC** algorithm for deciding equivalence of probabilistic automata. Recall that **NC** is the subclass of **P** containing those problems that can be solved in polylogarithmic parallel time [11] (see also Section 2). Tzeng [28] considers the path equivalence problem for nondeterministic automata which asks, given nondeterministic automata $\mathcal{A}$ and $\mathcal{B}$, whether each word has the same number of accepting paths in $\mathcal{A}$ as in $\mathcal{B}$. He gives a deterministic **NC** algorithm for deciding path equivalence which can be straightforwardly adapted to yield an **NC** algorithm for equivalence of probabilistic automata. Our new randomised algorithm has the same parallel time complexity as Tzeng's algorithm, but it also outputs a word on which the automata differ in case of inequivalence, which Tzeng's algorithm cannot. Our algorithm is based on the *Isolating Lemma*, which was used in [17] to compute perfect matchings in randomised **NC**. The randomised algorithm in [12], which relies on the Schwartz-Zippel lemma, can also output a counterexample, exploiting the self-reducibility of the equivalence problem—however it does not seem possible to use this algorithm to compute counterexamples in **NC**. Whether there is a deterministic **NC** algorithm that outputs counterexamples in case of inequivalence remains open.

In Section 4 we consider equivalence of probabilistic automata with one or more cost structures. Costs (or rewards, which can be considered as negative costs) are omnipresent in probabilistic modelling for capturing quantitative effects of probabilistic computations, such as consumption of time, (de-)allocation of memory, energy usage, financial gains, etc. We model each cost structure as an integer-valued *counter*, and annotate the transitions with counter changes.

In nondeterministic cost automata [2, 14] the cost of a word is the minimum of the costs of all accepting runs on that word. In probabilistic cost automata we instead associate a probability distribution over costs with each input word, representing the probability that a run over that word has a given cost. Whereas equivalence for nondeterministic cost automata is undecidable [2, 14], we show that equivalence of probabilistic cost automata is decidable in randomised polynomial time (and in deterministic polynomial time if the number of counters is fixed). Our proof of decidability, and the complexity bounds we obtain, involves a combination of classical techniques of [23, 27] with basic ideas from polynomial identity testing.

We present a case study in which costs are used to model the computation time required by an RSA encryption algorithm, and show that the vulnerability of the algorithm to timing attacks depends on the (in-)equivalence of probabilistic cost automata. In [13] two possible defenses against such timing leaks were suggested. We also analyse their effectiveness.

In Section 5 we consider pushdown automata. Probabilistic pushdown automata are a natural model of recursive probabilistic procedures, stochastic grammars and branching processes [10, 15]. The equivalence problem of deterministic pushdown automata has been extensively studied [25, 26]. We study the equivalence problem for *probabilistic visibly pushdown automata (VPA)* [3]. In a visibly pushdown automaton, whether the stack is popped or pushed is determined by the input symbol being read.

We show that the equivalence problem for probabilistic VPA is logspace equivalent to *Arithmetic Circuit Identity Testing (ACIT)*, which is the problem of determining equivalence of polynomials presented via arithmetic circuits [1]. Several polynomial-time randomized algorithms are known for **ACIT**, but it is a major open problem whether it can be solved in polynomial time by a deterministic algorithm. The inter-reducibility of probabilistic VPA equivalence and **ACIT** is reminiscent of the reduction of the positivity problem for arithmetic circuits to the reachability problem for recursive Markov chains [10]. However in this case the reduction is only in one direction—from circuits to recursive Markov chains.

In the technical development below it is convenient to consider $\mathbb{Q}$-weighted automata, which generalise probabilistic automata. All our results and examples are stated in terms of $\mathbb{Q}$-weighted automata. Some proofs have been moved to an appendix.

2 Preliminaries

2.1 Complexity Classes

Recall that **NC** is the subclass of **P** comprising those problems considered efficiently parallelisable. **NC** can be defined via *parallel random-access machines (PRAMs)*, which consist of a set of processors communicating through a shared memory. A problem is in **NC** if it can be solved in time $(\log n)^{O(1)}$ (polylogarithmic time) on a PRAM with $n^{O(1)}$ (polynomially many) processors. A more abstract definition of **NC** is as the class of languages which have **L**-uniform Boolean circuits of polylogarithmic depth and polynomial size. More specifically, denote by $\mathbf{NC}^k$ the class of languages which have circuits of depth $O(\log^k n)$. The complexity class **RNC** consists of those languages with randomized **NC** algorithms. We have the following inclusions none of which is known to be strict:

$$\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}^2 \subseteq \mathbf{NC} \subseteq \mathbf{RNC} \subseteq \mathbf{P}\,.$$

Problems in **NC** include directed reachability, computing the rank and determinant of an integer matrix, solving linear systems of equations and the tree-

isomorphism problem. Problems that are **P**-hard under logspace reductions include circuit value and max-flow. Such problems are not in **NC** unless **P** = **NC**. Problems in **RNC** include matching in graphs and max flow in 0/1-valued networks. In both cases these problems have resisted classification as either in **NC** or **P**-hard. See [11] for more details about **NC** and **RNC**.

2.2 Sequence Spaces

In this section we recall some results about spaces of sequences [22].

Given $s > 0$, define the following space of *formal power series*:

$$\ell_1(\mathbb{Z}^s) := \{f : \mathbb{Z}^s \to \mathbb{R} : \textstyle\sum_{\boldsymbol{v}\in\mathbb{Z}^s} |f(\boldsymbol{v})| < \infty\} \ .$$

Then $\ell_1(\mathbb{Z}^s)$ is a complete vector space under the norm $||f|| = \sum_{\boldsymbol{v}\in\mathbb{Z}^s} |f(\boldsymbol{v})|$. We can moreover endow $\ell_1(\mathbb{Z}^s)$ with a Banach algebra structure with multiplication

$$(f * g)(\boldsymbol{v}) := \sum_{\substack{\boldsymbol{u},\boldsymbol{w}\in\mathbb{Z}^s \\ \boldsymbol{u}+\boldsymbol{w}=\boldsymbol{v}}} f(\boldsymbol{u})g(\boldsymbol{w}) \, .$$

Given $n > 0$ we also consider the space $\ell_1(\mathbb{Z}^s)^{n\times n}$ of $n \times n$ matrices with coefficients in $\ell_1(\mathbb{Z}^s)$. This is a complete normed linear space with respect to the infinity matrix norm

$$||M|| := \max_{1\le i\le n} \sum_{1\le j\le n} ||M_{i,j}|| \, .$$

If we define matrix multiplication in the standard way, using the algebra structure on $\ell_1(\mathbb{Z}^s)$, then $||MN|| \le ||M||||N||$. In particular, if $||M|| < 1$ then we can define a Kleene-star operation by $M^* := (I - M)^{-1} = \sum_{k=0}^{\infty} M^k$.

3 Weighted Automata

To permit effective representation of automata we assume that all transition probabilities are rational numbers. In our technical development it is convenient to work with $\mathbb{Q}$-*weighted automata* [23], which are a generalisation of Rabin's probabilistic automata.

A $\mathbb{Q}$-*weighted automaton* $\mathcal{A} = (n, \Sigma, M, \boldsymbol{\alpha}, \boldsymbol{\eta})$ consists of a positive integer $n \in \mathbb{N}$ representing the number of states, a finite alphabet Σ, a map $M : \Sigma \to \mathbb{Q}^{n\times n}$ assigning a transition matrix to each alphabet symbol, an initial (row) vector $\boldsymbol{\alpha} \in \mathbb{Q}^n$, and a final (column) vector $\boldsymbol{\eta} \in \mathbb{Q}^n$. We extend M to Σ^* as the matrix product $M(\sigma_1 \dots \sigma_k) := M(\sigma_1) \cdot \ldots \cdot M(\sigma_k)$. The automaton $\mathcal{A}$ assigns each word w a *weight* $\mathcal{A}(w) \in \mathbb{Q}$, where $\mathcal{A}(w) := \boldsymbol{\alpha}M(w)\boldsymbol{\eta}$. An automaton $\mathcal{A}$ is said to be *zero* if $\mathcal{A}(w) = 0$ for all $w \in \Sigma^*$. Two automata $\mathcal{B}, \mathcal{C}$ over the same alphabet Σ are said to be *equivalent* if $\mathcal{B}(w) = \mathcal{C}(w)$ for all $w \in \Sigma^*$. In the remainder of this section we present a randomised $\mathbf{NC}^2$ algorithm for deciding equivalence of $\mathbb{Q}$-weighted automata and, in case of inequivalence, outputting a counterexample.

Given two automata $\mathcal{B}, \mathcal{C}$ that are to be checked for equivalence, one can compute an automaton $\mathcal{A}$ with $\mathcal{A}(w) = \mathcal{B}(w) - \mathcal{C}(w)$ for all $w \in \Sigma^*$. Then $\mathcal{A}$ is zero if and only if $\mathcal{B}$ and $\mathcal{C}$ are equivalent. Given $\mathcal{B} = (n^{(\mathcal{B})}, \Sigma, M^{(\mathcal{B})}, \boldsymbol{\alpha}^{(\mathcal{B})}, \boldsymbol{\eta}^{(\mathcal{B})})$ and $\mathcal{C} = (n^{(\mathcal{C})}, \Sigma, M^{(\mathcal{C})}, \boldsymbol{\alpha}^{(\mathcal{C})}, \boldsymbol{\eta}^{(\mathcal{C})})$, set $\mathcal{A} = (n, \Sigma, M, \boldsymbol{\alpha}, \boldsymbol{\eta})$ with $n := n^{(\mathcal{B})} + n^{(\mathcal{C})}$ and

$$M(\sigma) := \begin{pmatrix} M^{(\mathcal{B})}(\sigma) & 0 \\ 0 & M^{(\mathcal{C})}(\sigma) \end{pmatrix}, \qquad \boldsymbol{\alpha} := (\boldsymbol{\alpha}^{(\mathcal{B})}, -\boldsymbol{\alpha}^{(\mathcal{C})}), \qquad \boldsymbol{\eta} := \begin{pmatrix} \boldsymbol{\eta}^{(\mathcal{B})} \\ \boldsymbol{\eta}^{(\mathcal{C})} \end{pmatrix}.$$

This reduction allows us to focus on *zeroness*, i.e., the problem of determining whether a given $\mathbb{Q}$-weighted automaton $\mathcal{A} = (n, \Sigma, M, \boldsymbol{\alpha}, \boldsymbol{\eta})$ is zero. (Since transition weights can be negative, zeroness is not the same as emptiness of the underlying unweighted automaton.) Note that a witness word $w \in \Sigma^*$ against zeroness of $\mathcal{A}$ is also a witness against the equivalence of $\mathcal{B}$ and $\mathcal{C}$. The following result from [27] is crucial.

Proposition 1. *If $\mathcal{A}$ is not equal to the zero automaton then there exists a word $u \in \Sigma^*$ of length at most $n-1$ such that $\mathcal{A}(u) \neq 0$.*

Our randomised $\mathbf{NC}^2$ procedure uses the Isolating Lemma of Mulmuley, Vazirani and Vazirani [17]. We use this lemma in a very similar way to [17], who are concerned with computing maximum matchings in graphs in **RNC**.

Lemma 2. *Let $\mathcal{F}$ be a family of subsets of a set $\{x_1, \ldots, x_N\}$. Suppose that each element x_i is assigned a weight w_i chosen independently and uniformly at random from $\{1, \ldots, 2N\}$. Define the weight of $S \in \mathcal{F}$ to be $\sum_{x_i \in S} w_i$. Then the probability that there is a unique minimum weight set in $\mathcal{F}$ is at least $1/2$.*

We will apply the Isolating Lemma in conjunction with Proposition 1 to decide zeroness of a weighted automaton $\mathcal{A}$. Suppose $\mathcal{A}$ has n states and alphabet Σ. Given $\sigma \in \Sigma$ and $1 \leq i \leq n$, choose a weight $w_{i,\sigma}$ independently and uniformly at random from the set $\{1, \ldots, 2|\Sigma|n\}$. Define the weight of a word $u = \sigma_1 \ldots \sigma_k$, $k \leq n$, to be $\mathrm{wt}(u) := \sum_{i=1}^{k} w_{i,\sigma_i}$. (The reader should not confuse this with the weight $\mathcal{A}(u)$ assigned to u by the automaton $\mathcal{A}$.) Then we obtain a univariate polynomial P from automaton $\mathcal{A}$ as follows:

$$P(x) = \sum_{k=0}^{n} \sum_{u \in \Sigma^k} \mathcal{A}(u) x^{\mathrm{wt}(u)}.$$

If $\mathcal{A}$ is equivalent to the zero automaton then clearly $P \equiv 0$. On the other hand, if $\mathcal{A}$ is non-zero, then by Proposition 1 the set $\mathcal{F} = \{u \in \Sigma^{\leq n} : \mathcal{A}(u) \neq 0\}$ is non-empty. Thus there is a unique minimum-weight word $u \in \mathcal{F}$ with probability at least $1/2$ by the Isolating Lemma. In this case P contains the monomial $x^{\mathrm{wt}(u)}$ with coefficient $\mathcal{A}(u)$ as its smallest-degree monomial. Thus $P \not\equiv 0$ with probability at least $1/2$.

It remains to observe that from the formula

$$P(x) = \boldsymbol{\alpha} \left(\sum_{i=0}^{n} \prod_{j=1}^{i} \sum_{\sigma \in \Sigma} M(\sigma) x^{w_{j,\sigma}} \right) \boldsymbol{\eta}$$

and the fact that iterated products of matrices of univariate polynomials can be computed in $\mathbf{NC}^2$ [7] we obtain an **RNC** algorithm for determining zeroness of weighted automata.

It is straightforward to extend the above algorithm to obtain an **RNC** procedure that not only decides zeroness of $\mathcal{A}$ but also outputs a word u such that $\mathcal{A}(u) \neq 0$ in case $\mathcal{A}$ is non-zero. Assume that $\mathcal{A}$ is non-zero and that the random choice of weights has isolated a unique minimum-weight word $u = \sigma_1 \dots \sigma_k$ such that $\mathcal{A}(u) \neq 0$. To determine whether $\sigma \in \Sigma$ is the i-th letter of u we can increase the weight $w_{i,\sigma}$ by 1 while leaving all other weights unchanged and recompute the polynomial $P(x)$. Then σ is the i-th letter in u if and only if the minimum-degree monomial in P changes. All of these tests can be done independently, yielding an **RNC** procedure.

Theorem 3. *Given two weighted automata $\mathcal{A}$ and $\mathcal{B}$, there is an **RNC** procedure that determines whether or not $\mathcal{A}$ and $\mathcal{B}$ are equivalent and that outputs a word w with $\mathcal{A}(w) \neq \mathcal{B}(w)$ in case $\mathcal{A}$ and $\mathcal{B}$ are inequivalent.*

4 Weighted Cost Automata

In this section we consider weighted automata with costs. Each transition has a cost, and the cumulative cost of a run is recorded in a tuple of counters. Transitions can also have negative costs, which can be considered as rewards. Note though that the counters do not affect the control flow of the automata. In Example 9 we use costs to record the passage of time in an encryption protocol. We explicitly include ε-transitions in our automata because they are convenient for applications (cf. Example 8) and we cannot rely on existing ε-elimination results in the presence of costs.

Let Σ be a finite alphabet not containing the symbol ε. A $\mathbb{Q}$-*weighted cost automaton* is a tuple $\mathcal{A} = (n, s, \Sigma, M, \boldsymbol{\alpha}, \boldsymbol{\eta})$, where $n \in \mathbb{N}$ is the number of states; $s \in \mathbb{N}$ is the number of counters; $M : \Sigma \cup \{\varepsilon\} \to (C \to \mathbb{Q})^{n \times n}$ is the transition function, where $C = \{-1, 0, 1\}^s$ is the set of *elementary cost vectors*; $\boldsymbol{\alpha} \in \mathbb{Q}^n$ is an initial (row) vector; $\boldsymbol{\eta} \in \mathbb{Q}^n$ is a final (column) vector. In this definition, $M(\sigma)_{i,j}(\boldsymbol{v})$ represents the weight of a σ-transition from state i to j with cost vector $\boldsymbol{v} \in C$. For the semantics to be well-defined we assume that the total weight of all outgoing ε-labelled transitions from any given state is strictly less than 1.

In order to define the semantics of weighted cost automata it is convenient to use results on matrices of formal power series from Section 2. We can regard $M(\sigma)$ as an $n \times n$ matrix whose entries are elements of the space $\ell_1(\mathbb{Z}^s)$ of formal power series, where $M(\sigma)_{i,j}(\boldsymbol{v}) = 0$ for $\boldsymbol{v} \in \mathbb{Z}^s \setminus C$. Our convention on the total weight of ε-transitions is equivalent to the requirement that $||M(\varepsilon)|| < 1$. We next extend M to a map $M : \Sigma^* \to (\ell_1(\mathbb{Z}^s))^{n \times n}$ such that, given a word $w \in \Sigma^*$ and states i, j, $M(w)_{i,j}(\boldsymbol{v})$ is the total weight of all w-labelled paths from state i to state j with accumulated cost $\boldsymbol{v} \in \mathbb{Z}^s$. Given a word $w = \sigma_1 \sigma_2 \dots \sigma_m \in \Sigma^*$,

we define

$$M(w) := M(\varepsilon)^* M(\sigma_1) M(\varepsilon)^* \cdots M(\sigma_m) M(\varepsilon)^* . \tag{1}$$

Finally, given $w \in \Sigma^*$ we define $\mathcal{A}(w) := \boldsymbol{\alpha}\, M(w)\, \boldsymbol{\eta}$. Then $\mathcal{A}(w)$ is an element of $\ell_1(\mathbb{Z}^s)$ such that $\mathcal{A}(w)(\boldsymbol{v})$ gives the total weight of all accepting runs with accumulated cost $\boldsymbol{v} \in \mathbb{Z}^s$.

Let $\boldsymbol{x} = (x_1, \ldots, x_s)$ be a vector of variables, one for each counter. Our equivalence algorithm is based on a representation of $\mathcal{A}(w)$ as a rational function in $\boldsymbol{x}$, following classical ideas [19]. Given $\boldsymbol{v} \in \mathbb{Z}^s$ we denote by $\boldsymbol{x}^{\boldsymbol{v}}$ the monomial $x_1^{v_1} \cdots x_s^{v_s}$. (Note that we allow negative powers in monomials.) We say that $f \in \ell_1(\mathbb{Z}^s)$ has *finite support* if $f(\boldsymbol{v}) = 0$ for all but finitely many $\boldsymbol{v} \in \mathbb{Z}^s$. We identify such an f with the polynomial $\sum_{\boldsymbol{v} \in \mathbb{Z}^s} f(\boldsymbol{v}) \boldsymbol{x}^{\boldsymbol{v}}$. We furthermore say that $f \in \ell_1(\mathbb{Z}^s)$ is *rational* if there exist $g, h : \mathbb{Z}^s \to \mathbb{Q}$ with finite support such that $f * h = g$. We then identify f with the rational function

$$\sum_{\boldsymbol{v} \in \mathbb{Z}^s} g(\boldsymbol{v}) \boldsymbol{x}^{\boldsymbol{v}} \Big/ \sum_{\boldsymbol{v} \in \mathbb{Z}^s} h(\boldsymbol{v}) \boldsymbol{x}^{\boldsymbol{v}} .$$

Note that we can clear all negative exponents from the numerator and denominator of such an expression. Note also that sums and products of rational functions correspond to sums and products in $\ell_1(\mathbb{Z}^s)$ in the above representation.

Proposition 4. *$M(w)$ can be represented as a matrix of rational functions in $\boldsymbol{x}$ such that the numerator and denominator in each matrix entry have degrees at most $2n(s+1) \cdot |w|$.*

Proof. From equation (1) it suffices to show that $M(\varepsilon)^*$ can be represented as a matrix of rational functions with appropriate degree bounds. Recall that $M(\varepsilon)^* = (I - M(\varepsilon))^{-1}$, so it suffices to show that $I - M(\varepsilon)$ (considered as a matrix of polynomials) has an inverse that can be represented as a matrix of rational functions. But the determinant formula yields that $\det(I - M(\varepsilon))$ is a (non-zero) polynomial in $\boldsymbol{x}$, thus the cofactor formula for inverting matrices yields a representation of $(I - M(\varepsilon))^{-1}$ as a matrix of rational functions in $\boldsymbol{x}$ of degree at most $2ns$. □

An automaton $\mathcal{A}$ is said to be *zero* if $\mathcal{A}(w) \equiv 0$ for all $w \in \Sigma^*$. Two automata $\mathcal{B}, \mathcal{C}$ over the same alphabet Σ with the same number of counters are said to be *equivalent* if $\mathcal{B}(w) \equiv \mathcal{C}(w)$ for all $w \in \Sigma^*$. As in Section 3, the equivalence problem can be reduced to the zeroness problem, so we focus on the latter.

The following proposition states that if there is a word witnessing that $\mathcal{A}$ is non-zero, then there is a "short" such word.

Proposition 5. *$\mathcal{A}$ is zero if and only if $\mathcal{A}(w) \equiv 0$ for all $w \in \Sigma^*$ of length at most $n - 1$.*

The proof, given in full in Appendix A, is similar to the linear algebra arguments from [23, 27], but involves an additional twist. The key idea is to substitute concrete values for the variables $\boldsymbol{x}$, thereby transforming from the setting

of infinite-dimensional vector spaces of rational functions in $\boldsymbol{x}$ to a finite dimensional setting where the arguments of [23, 27] apply.

The decidability of zeroness (and hence equivalence) for weighted cost automata follows immediately from Proposition 5. However, using polynomial identity testing, we arrive at the following theorem.

Theorem 6. *The equivalence problem for weighted cost automata is decidable in randomised polynomial time.*

Proof. We have already observed that the equivalence problem can be reduced to the zeroness problem. We now reduce the zeroness problem to polynomial identity testing.

Given an automaton $\mathcal{A} = (n, s, \Sigma, M, \boldsymbol{\alpha}, \boldsymbol{\eta})$, for each word $w \in \Sigma^*$ of length at most n we have a rational expression $\mathcal{A}(w)$ in variables $\boldsymbol{x} = (x_1, \ldots, x_s)$ which has degree at most $d := 2n(s+1) \cdot n$ by Proposition 4.

Now consider the set $R := \{1, 2, \ldots, 2d\}$. Suppose that we pick $\boldsymbol{r} \in R^s$ uniformly at random. Denote by $\mathcal{A}(w)(\boldsymbol{r})$ the result of substituting $\boldsymbol{r}$ for $\boldsymbol{x}$ in the rational expression $\mathcal{A}(w)$. Clearly if $\mathcal{A}$ is a zero automaton then $\mathcal{A}(w)(\boldsymbol{r}) = 0$ for all $\boldsymbol{r}$. On the other hand, if $\mathcal{A}$ is non-zero then by Proposition 5 there exists a word $w \in \Sigma^*$ of length at most n such that $\mathcal{A}(w) \not\equiv 0$. Since the degree of the rational expression $A(w)$ is at most d it follows from the Schwartz-Zippel theorem [9, 24, 29] that the probability that $\mathcal{A}(w)(\boldsymbol{r}) = 0$ is at most $1/2$.

Thus our randomised procedure is to pick $\boldsymbol{r} \in R^s$ uniformly at random and to check whether $\mathcal{A}(w)(\boldsymbol{r}) = 0$ for some $w \in \Sigma^*$. It remains to show how we can do this check in polynomial time. To achieve this we show that there is a $\mathbb{Q}$-weighted automaton $\mathcal{B}$ with no counters such that $\mathcal{A}(w)(\boldsymbol{r}) = \mathcal{B}(w)$ for all $w \in \Sigma^*$, since we can then check $\mathcal{B}$ for zeroness using, e.g., Tzeng's algorithm [27]. The automaton $\mathcal{B}$ has the form $\mathcal{B} = (n^{(\mathcal{B})}, \Sigma, M^{(\mathcal{B})}, \boldsymbol{\alpha}^{(\mathcal{B})}, \boldsymbol{\eta}^{(\mathcal{B})})$, where $n^{(\mathcal{B})} = n$, $\boldsymbol{\alpha}^{(\mathcal{B})} = \boldsymbol{\alpha}$, $\boldsymbol{\eta}^{(\mathcal{B})} = \boldsymbol{\eta}$ and $M^{(\mathcal{B})}(\sigma) = \sum_{\boldsymbol{v} \in \mathbb{Z}^s} M(\sigma)(\boldsymbol{v}) \boldsymbol{r}^{\boldsymbol{v}}$ for all $\sigma \in \Sigma$. □

Corollary 7. *For each fixed number of counters the equivalence problem for weighted cost automata is decidable in deterministic polynomial time.*

See Appendix A for a proof.

Example 8. We consider probabilistic programs that randomly increase and decrease a single counter (initialised with 0) so that upon termination the counter has a random value $X \in \mathbb{Z}$. The programs should be such that X is a random variable with $X = Y - Z$ where Y and Z are independent random variables with a geometric distribution with parameters $p = 1/2$ and $p = 1/3$, respectively. (By that we mean that $\Pr(Y = k) = (1-p)^k p$ for $k \in \{0, 1, \ldots\}$, and similarly for Z.) Figure 1 shows code in the syntax of the APEX tool.

The program on the left consecutively runs two while loops: it first increments the counter according to a geometric distribution with parameter $1/2$ and then decrements the counter according to a geometric distribution with parameter $1/3$, so that the final counter value is distributed as desired. The program on the right is more efficient in that it runs only one of two while loops, depending

```
inc:com, dec:com |-
  var%2 flip;
  flip := 0;
  while (flip = 0) do {
    flip := coin[0:1/2,1:1/2];
    if (flip = 0) then {
     inc;
    };
  };
  flip := 0;
  while (flip = 0) do {
    flip := coin[0:2/3,1:1/3];
    if (flip = 0) then {
     dec;
    };
  }
:com
```

```
inc:com, dec:com |-
  var%2 flip;
  flip := coin[0:1/2,1:1/2];
  if (flip = 0) then {
    while (flip = 0) do {
      flip := coin[0:1/2,1:1/2];
      if (flip = 0) then {
       inc;
      };
    };
  } else {
    flip := 0;
    while (flip = 0) do {
      dec;
      flip := coin[0:2/3,1:1/3];
    };
  }
:com
```

Fig. 1. Two APEX programs for producing a counter that is distributed as the difference between two geometrically distributed random variables.

on a single coin flip at the beginning. It may not be obvious though that the final counter value follows the same distribution as in the left program. We used the APEX tool to translate the programs to the probabilistic cost automata $\mathcal{B}$ and $\mathcal{C}$ shown in Figure 2. Since the input alphabets are empty, it suffices to consider the input word ε when comparing $\mathcal{B}$ and $\mathcal{C}$ for equivalence. If we construct the difference automaton $\mathcal{A} = (5, 1, \emptyset, M, \boldsymbol{\alpha}, \boldsymbol{\eta})$ and invert the matrix of polynomials $I - M(\varepsilon)$, we obtain

$$\mathcal{A}(\varepsilon)(x) = \left(\frac{2}{x-2}, \frac{2}{(3x-2)(x-2)}, 1, \frac{-x}{2(x-2)}, \frac{3}{2(3x-2)}\right)\boldsymbol{\eta} \equiv 0\,,$$

which proves equivalence of $\mathcal{B}$ and $\mathcal{C}$. Notice that the actual algorithm would not compute $\mathcal{A}(\varepsilon)(x)$ as a polynomial, but it would compute $\mathcal{A}(\varepsilon)(r)$ only for a few concrete values $r \in \mathbb{Q}$. □

Example 9. RSA [21] is a widely-used cryptographic algorithm. Popular implementations of the RSA algorithm have been shown to be vulnerable to timing attacks that reveal private keys [13, 5]. The preferred countermeasures are blinding techniques that randomise certain aspects of the computation, which are described in, e.g., [13]. We model the timing behaviour of the RSA algorithm using probabilistic cost automata, where costs encode time. These automata are produced by APEX, and are then used to check for timing leaks with and without blinding.

At the heart of RSA decryption is a modular exponentiation, which computes the value $m^d \mod N$ where $m \in \{0, \ldots, N-1\}$ is the encrypted message, $d \in \mathbb{N}$

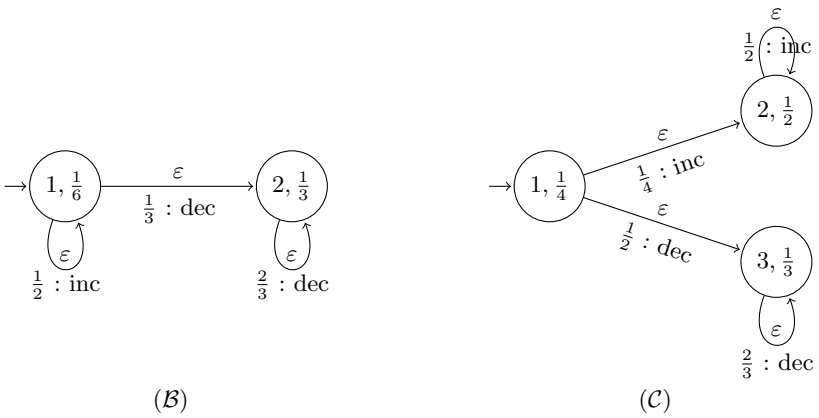

Fig. 2. Automata produced from the code in Figure 1. The states are labelled with their number and their "acceptance probability" ($\boldsymbol{\eta}$-weight). In both automata, state 1 is the only initial state ($\boldsymbol{\alpha}_1 = 1$ and $\boldsymbol{\alpha}_i = 0$ for $i \neq 1$). The transitions are labelled with the input symbol ε, with a probability (weight) and a counter action (i.e. cost).

is the private decryption exponent and $N \in \mathbb{N}$ is a modulus. An attacker wants to find out d. We model RSA decryption in APEX by implementing modular exponentiation by iterative squaring (see Figure 3). We consider the situation where the attacker is able to control the message m, and tries to derive d by observing the runtime distribution over different messages m. Following [13] we assume that the running time of multiplication depends on the operand values (because a source-level multiplication typically corresponds to a cascade of processor-level multiplications). By choosing the 'right' input message m, an attacker can observe which private keys are most likely.

We consider two blinding techniques mentioned in Kocher [13]. The first one is base blinding, i.e., the message is multiplied by r^d before exponentiation where d is a random number, which gives a result that can be fixed by dividing by r but makes it impossible for the attacker to control the basis of the exponentiation. The second one is exponent blinding, which adds a multiple of the group order $\varphi(N)$ of $\mathbb{Z}/N\mathbb{Z}$ to the exponent, which doesn't change the result of the exponentiation[3] but changes the timing behaviour.

Figure 4 shows the automaton for $N = 10$, and private key $0, 1, 0, 1$ with message blinding enabled. The APEX program is given in Figure 3.

We investigate the effectiveness of blinding. Two private keys are indistinguishable if the resulting automata are equivalent. The more keys are indistinguishable the safer the algorithm. We analyse which private keys are identified by plain RSA, RSA with a blinded message and RSA with blinded exponent.

For example, in plain RSA, the following keys $0, 1, 0, 1$ and $1, 0, 0, 1$ are indistinguishable, keys $0, 1, 1, 0$ and $0, 0, 1, 1$ are indistinguishable with base blinding, lastly $1, 0, 0, 1$ and $1, 0, 1, 1$ are equivalent only with exponent blinding. Overall

[3] Euler's totient function φ satisfies $a^{\varphi(N)} \equiv 1 \mod N$ for all $a \in \mathbb{Z}$.

9 different keys are distinguishable with plain RSA, 7 classes with base blinding and 4 classes with exponent blinding.

```
const N := 10;    // modulus
const Bits := 4 ; // number of bits of the key

m :int%N, inc:com |-
var%2 exponent[Bits] = [0,1,0,1];
com power(x:int%N) {
   var%N s := 1;
   var%N R;
   for(var%(Bits + 1) k; k < Bits; ++k) do {
      R:=s;
      if(exponent[k]) then {
         R := R*x;
         if(5<=R) then { inc; inc } else { inc }
      }
      s := R*R;
   }
}
var%N message := m*rand[N]; // blinding
power(message) : com
```

Fig. 3. APEX code for RSA.

5 Pushdown Automata and Arithmetic Circuits

In a visibly pushdown automaton [3] the stack operations are determined by the input word. Consequently VPA have a more tractable language theory than ordinary pushdown automata. The main result of this section shows that the equivalence problem for weighted VPA is logspace equivalent to the problem **ACIT** of determining whether a polynomial represented by an arithmetic circuit is identically zero.

A *visibly pushdown alphabet* $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$ consists of a finite set of *calls* Σ_c, a finite set of *returns* Σ_r, and a finite set of *internal actions* Σ_{int}. A visibly pushdown automaton over alphabet Σ is restricted so that it pushes onto the stack when it reads a call, pops the stack when it reads a return, and leaves the stack untouched when reading internal actions. Due to this restriction visibly pushdown automata only accept words in which calls and returns are appropriately matched. Define the set of *well-matched words* to be $\bigcup_{i\in\mathbb{N}} L_i$, where $L_0 = \Sigma_{int} + \{\varepsilon\}$ and $L_{i+1} = \Sigma_c L_i \Sigma_r + L_i L_i$.

A $\mathbb{Q}$-*weighted visibly pushdown automaton* on alphabet Σ is a tuple $\mathcal{A} = (n, \boldsymbol{\alpha}, \boldsymbol{\eta}, \Gamma, M)$, where n is the number of *states*, $\boldsymbol{\alpha}$ is an n-dimensional *initial* (row) vector, $\boldsymbol{\eta}$ is an n-dimensional *final* (column) vector, Γ is a finite *stack*

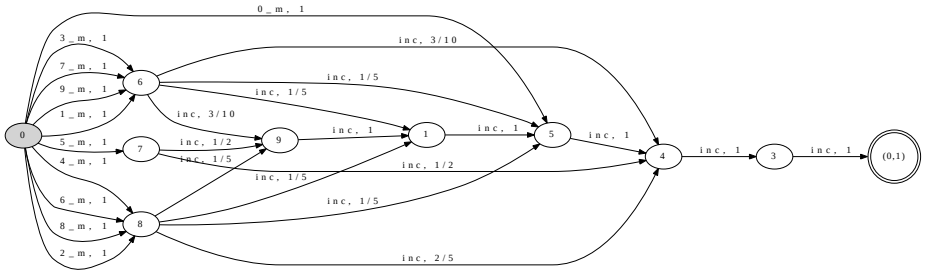

Fig. 4. Modeling RSA decryption with APEX.

alphabet, and $M = (M_c, M_r, M_{int})$ is a tuple of *matrix-valued transition functions* with types $M_c : \Sigma_c \times \Gamma \to \mathbb{Q}^{n\times n}$, $M_r : \Sigma_r \times \Gamma \to \mathbb{Q}^{n\times n}$ and $M_{int} : \Sigma_{int} \to \mathbb{Q}^{n\times n}$. If $a \in \Sigma_c$ and $\gamma \in \Gamma$ then $M_c(a,\gamma)_{i,j}$ gives the weight of an a-labelled transition from state i to state j that pushes γ on the stack. If $a \in \Sigma_r$ and $\gamma \in \Gamma$ then $M_r(a,\gamma)_{i,j}$ gives the weight of an a-labelled transition from state i to j that pops γ from the stack.

For each well-matched word $u \in \Sigma^*$ we define an $n \times n$ rational matrix $M^{(\mathcal{A})}(u)$ whose (i,j)-th entry denotes the total weight of all paths from state i to state j along input u. The definition of $M^{(\mathcal{A})}(u)$ follows the inductive definition of well-matched words. The base cases are $M^{(\mathcal{A})}(\varepsilon) = I$ and $M^{(\mathcal{A})}(a)_{i,j} = M_{int}(a)_{i,j}$. The inductive cases are

$$M^{(\mathcal{A})}(uv) = M^{(\mathcal{A})}(u) \cdot M^{(\mathcal{A})}(v)$$
$$M^{(\mathcal{A})}(aub) = \sum_{\gamma\in\Gamma} M_c(a,\gamma) \cdot M^{(\mathcal{A})}(u) \cdot M_r(b,\gamma)\,,$$

for $a \in \Sigma_c$, $b \in \Sigma_r$.

The weight assigned by $\mathcal{A}$ to a well-matched word w is defined to be $\mathcal{A}(w) := \boldsymbol{\alpha} M^{(\mathcal{A})}(u)\boldsymbol{\eta}$. We say that two weighted VPA $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* if for each well-matched word w we have $\mathcal{A}(w) = \mathcal{B}(w)$.

An *arithmetic circuit* is a finite directed acyclic multigraph whose vertices, called *gates*, have indegree 0 or 2. Vertices of indegree 0 are called *input gates* and are labelled with a constant 0 or 1, or a variable from the set $\{x_i : i \in \mathbb{N}\}$. Vertices of indegree 2 are called *internal gates* and are labelled with one of the arithmetic operations $+$, $*$ or $-$. We assume that there is a unique gate with outdegree 0 called the *output*. Note that C is a multigraph, so there can be two edges between a pair of gates, i.e., both inputs to a given gate can lead from the same source. We call a circuit *variable-free* if all inputs gates are labelled 0 or 1.

The *Arithmetic Circuit Identity Testing (**ACIT**)* problem asks whether the output of a given circuit is equal to the zero polynomial. **ACIT** is known to be in **coRP** but it remains open whether there is a polynomial or even subexponential algorithm for this problem [1]. Utilising the fact that a variable-free arithmetic circuit of size $O(n)$ can compute 2^{2^n}, Allender *et al.* [1] give a

logspace reduction of the general **ACIT** problem to the special case of variable-free circuits. Henceforth we assume without loss of generality that all circuits are variable-free. Furthermore we recall that **ACIT** can be reformulated as the problem of deciding whether two variable-free circuits using only the arithmetic operations $+$ and $*$ compute the same number [1].

The proof of the following proposition is given in Appendix B.

Proposition 10. ACIT *is logspace reducible to the equivalence problem for weighted visibly pushdown automata.*

In the remainder of this section we give a converse reduction: from equivalence of weighted VPA to **ACIT**. The following result gives a decision procedure for the equivalence of two weighted VPA $\mathcal{A}$ and $\mathcal{B}$.

Proposition 11. *$\mathcal{A}$ is equivalent to $\mathcal{B}$ if and only if $\mathcal{A}(w) = \mathcal{B}(w)$ for all words $w \in L_{n^2}$, where n is the sum of the number of states of $\mathcal{A}$ and the number of states of $\mathcal{B}$.*

Proof. Recall that for each balanced word $u \in \Sigma^*$ we have rational matrices $M^{(\mathcal{A})}(u)$ and $M^{(\mathcal{B})}(u)$ giving the respective state-to-state transition weights of $\mathcal{A}$ and $\mathcal{B}$ on reading u. These two families of matrices can be combined into a single family

$$\mathcal{M} = \left\{ \begin{pmatrix} M^{(\mathcal{A})}(u) & \mathbf{0} \\ \mathbf{0} & M^{(\mathcal{B})}(u) \end{pmatrix} : u \text{ well-matched} \right\}$$

of $n \times n$ matrices. Let us also write $\mathcal{M}_i$ for the subset of $\mathcal{M}$ generated by those well-matched words $u \in L_i$.

Let $\boldsymbol{\alpha}^{(\mathcal{A})}, \boldsymbol{\eta}^{(\mathcal{A})}$ and $\boldsymbol{\alpha}^{(\mathcal{B})}, \boldsymbol{\eta}^{(\mathcal{B})}$ be the respective initial and final-state vectors of $\mathcal{A}$ and $\mathcal{B}$. Then $\mathcal{A}$ is equivalent to $\mathcal{B}$ if and only if

$$(\, \boldsymbol{\alpha}^{(\mathcal{A})} \; \boldsymbol{\alpha}^{(\mathcal{B})} \,) M \begin{pmatrix} \boldsymbol{\eta}^{(\mathcal{A})} \\ -\boldsymbol{\eta}^{(\mathcal{B})} \end{pmatrix} = 0 \tag{2}$$

for all $M \in \mathcal{M}$. It follows that $\mathcal{A}$ is equivalent to $\mathcal{B}$ if and only if (2) holds for all M in $\mathrm{span}(\mathcal{M})$, where the span is taken in the rational vector space of $n \times n$ rational matrices. But $\mathrm{span}(\mathcal{M}_i)$ is an ascending sequence of vector spaces:

$$\mathrm{Span}(\mathcal{M}_0) \subseteq \mathrm{Span}(\mathcal{M}_1) \subseteq \mathrm{Span}(\mathcal{M}_2) \subseteq \ldots$$

It follows from a dimension argument that this sequence stops in at most n^2 steps and we conclude that $\mathrm{span}(\mathcal{M}) = \mathrm{span}(\mathcal{M}_{n^2})$. □

Proposition 12. *Given a weighted visibly pushdown automaton $\mathcal{A}$ and $n \in \mathbb{N}$ one can compute in logarithmic space a circuit that represents $\sum_{w \in L_{n^2}} \mathcal{A}(w)$.*

Proof. From the definition of the language L_i and the family of matrices $M^{(\mathcal{A})}$ we have:

$$\sum_{w \in L_{i+1}} M^{(\mathcal{A})}(w) = \sum_{a \in \Sigma_c} \sum_{b \in \Sigma_r} \sum_{\gamma \in \Gamma} M^{(\mathcal{A})}(a, \gamma) \left(\sum_{u \in L_i} M^{(\mathcal{A})}(u) \right) M^{(\mathcal{A})}(b, \gamma) + \left(\sum_{u \in L_i} M^{(\mathcal{A})}(u) \right) \left(\sum_{u \in L_i} M^{(\mathcal{A})}(u) \right) .$$

The above equation implies that we can compute in logarithmic space a circuit that represents $\sum_{w \in L_n} M^{(\mathcal{A})}(w)$. The result of the proposition immediately follows by premultiplying by the initial state vector and postmultiplying by the final state vector. □

A key property of weighted VPA is their closure under product.

Proposition 13. *Given weighted VPA $\mathcal{A}$ and $\mathcal{B}$ on the same alphabet Σ one can define a* synchronous-product automaton*, denoted $\mathcal{A} \times \mathcal{B}$, such that $(\mathcal{A} \times \mathcal{B})(w) = \mathcal{A}(w)\mathcal{B}(w)$ for all $w \in \Sigma^*$.*

The proof of Proposition 13, given in Appendix B, exploits the fact that the stack height is determined by the input word, so the respective stacks of $\mathcal{A}$ and $\mathcal{B}$ operating in parallel can be simulated in a single stack.

Proposition 14. *The equivalence problem for weighted visibly pushdown automata is logspace reducible to **ACIT**.*

Proof. Let $\mathcal{A}$ and $\mathcal{B}$ be weighted visibly pushdown automata with a total of n states between them. Then

$$\begin{aligned}\sum_{w \in L_n} (\mathcal{A}(w) - \mathcal{B}(w))^2 &= \sum_{w \in L_n} \mathcal{A}(w)^2 + \mathcal{B}(w)^2 - 2\mathcal{A}(w)\mathcal{B}(w) \\ &= \sum_{w \in L_n} (\mathcal{A} \times \mathcal{A})(w) + (\mathcal{B} \times \mathcal{B})(w) - 2(\mathcal{A} \times \mathcal{B})(w)\end{aligned}$$

Thus $\mathcal{A}$ is equivalent to $\mathcal{B}$ iff $\sum_{w \in L_n} (\mathcal{A} \times \mathcal{A})(w) + (\mathcal{B} \times \mathcal{B})(w) = 2 \sum_{w \in L_n} (\mathcal{A} \times \mathcal{B})(w)$. But Propositions 12 and 13 allow us to translate the above equation into an instance of **ACIT**. □

The trick of considering sums-of-squares of acceptance weights in the above proof is inspired by [28, Lemma 1].

References

1. E.E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.
2. S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *ATVA*, volume 6996 of *LNCS*, pages 482–491. Springer, 2011.
3. R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th Annual ACM Symposium on Theory of Computing STOC*, pages 202–211. ACM, 2004.
4. V. D. Blondel and V. Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theoretical Computer Science*, 36 (3):231–245, 2003.
5. D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
6. A. Condon and R. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *Proceedings of FOCS*, pages 462–467, 1989.
7. S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–22, 1985.

8. C. Cortes, M. Mohri, and A. Rastogi. On the computation of some standard distances between probabilistic automata. In *Proc. of CIAA*, pages 137–149, 2006.
9. R. DeMillo and R. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.
10. K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1):1:1–1:66, 2009.
11. R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.
12. S. Kiefer, A.S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *CAV*, volume 6806 of *LNCS*, pages 526–540, 2011.
13. P.C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
14. D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Int. Journal of Alg. and Comp.*, 4(3):232–249, 1994.
15. A. Kučera, J. Esparza, and R. Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1):1–31, 2006.
16. A. Legay, A. S. Murawski, J. Ouaknine, and J. Worrell. On automated verification of probabilistic programs. In *TACAS*, volume 4963 of *LNCS*, pages 173–187. 2008.
17. K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. In *STOC*, pages 345–354, 1987.
18. A. S. Murawski and J. Ouaknine. On probabilistic program equivalence and refinement. In *CONCUR*, volume 3653 of *LNCS*, pages 156–170. 2005.
19. I. Niven. Formal power series. *American Mathematical Monthly*, 76(8):871–889, 1969.
20. M. O. Rabin. Probabilistic automata. *Inf. and Control*, 6 (3):230–245, 1963.
21. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.
22. W. Rudin. *Functional analysis*. International Series in Pure and Applied Mathematics. McGraw-Hill Inc., New York, second edition, 1991.
23. M.-P. Schützenberger. On the definition of a family of automata. *Inf. and Control*, 4:245–270, 1961.
24. J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.
25. G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *ICALP*, volume 1256 of *LNCS*. Springer, 1997.
26. C. Stirling. Deciding DPDA equivalence is primitive recursive. In *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 821–832. Springer, 2002.
27. W. Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21(2):216–227, 1992.
28. W. Tzeng. On path equivalence of nondeterministic finite automata. *Inf. Process. Lett.*, 58(1):43–46, 1996.
29. R. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

A Proofs of Section 4

Proposition 5. *$\mathcal{A}$ is zero if and only if $\mathcal{A}(w) \equiv 0$ for all $w \in \Sigma^*$ of length at most $n-1$.*

Proof. Suppose that $\mathcal{A}$ is non-zero. Then there exists some word $w \in \Sigma^*$ such that $\boldsymbol{\alpha} M(w) \boldsymbol{\eta}$ is a non-zero rational expression in $\boldsymbol{x}$. Thus we can pick $\boldsymbol{r} \in \mathbb{Q}^s$ such that $\boldsymbol{\alpha} M(w)(\boldsymbol{r}) \boldsymbol{\eta} \neq 0$. Now define

$$V_i := \mathrm{Span}\{\boldsymbol{\alpha} M(w)(\boldsymbol{r}) \in \mathbb{Q}^n : |u| \leq i\}\,.$$

Then $V_0 \subseteq V_1 \subseteq V_2 \subseteq \ldots$ is an increasing family of subspaces of $\mathbb{Q}^n$. Considering the dimension of each V_i there exists $i_0 < n$ with $V_{i_0} = V_{i_0+1}$. But for all i we have

$$V_{i+1} = \mathrm{Span}(V_i \cup \{(\boldsymbol{v} M(a) M(\varepsilon)^*)(\boldsymbol{r}) : a \in \Sigma, \boldsymbol{v} \in V_i\})$$

From this characterisation it is clear that $V_{i_0+1} = V_{i_0}$ entails that $V_i = V_{i_0}$ for all $i \geq i_0$.

From the assumption that $\mathcal{A}(w) \not\equiv 0$ as a rational function there exists $\boldsymbol{r} \in \mathbb{Q}^s$ such that $\mathcal{A}(w)(\boldsymbol{r}) \neq 0$. By the above we have that $\boldsymbol{\alpha} M(w)(\boldsymbol{r}) \in V_{i_0}$ and thus $\boldsymbol{\eta}$ is not orthogonal to V_{i_0}. In particular, there exists some word u of length at most $i_0 \leq n-1$ such that $\boldsymbol{\alpha} M(u)(\boldsymbol{r}) \eta \neq 0$. But then $\mathcal{A}(u) \not\equiv 0$. □

Corollary 7. *For each fixed number of counters the equivalence problem for weighted cost automata is decidable in deterministic polynomial time.*

Proof. Consider a counter automaton $\mathcal{A}$ to be checked for zeroness. If the number s of counters is fixed, then the set of sample points R^s in the proof of Theorem 6 has polynomial size. Thus we can in polynomial time test whether $\mathcal{A}(w)(\boldsymbol{r}) = 0$ for all $w \in \Sigma^*$ and $\boldsymbol{r} \in R^s$. If $\mathcal{A}$ is non-zero then the proof of Theorem 6 guarantees the existence of $w \in \Sigma^*$ and $\boldsymbol{r} \in R^s$ such that $\mathcal{A}(w)(\boldsymbol{r}) \neq 0$. □

B Proofs of Section 5

Proposition 10. **ACIT** *is logspace reducible to the equivalence problem for weighted visibly pushdown automata.*

Proof. Let C and C' be two circuits over basis $\{+, *\}$. Without loss of generality we assume that in each circuit the inputs of a depth-i gate both have depth $i+1$, $+$-nodes have even depth, $*$-nodes have odd depth, and input nodes all have the same depth d. Notice that in either circuit any path from an input gate to an output gate has length d.

We define two automata $\mathcal{A}$ and $\mathcal{A}'$ that are equivalent if and only if C and C' have the same output. Both automata are defined over the alphabet $\{c, r, \iota\}$,

with c a call, r a return and ι an internal event. We explain how $\mathcal{A}$ arises from C; the definition of $\mathcal{A}'$ is entirely analogous.

Suppose that C has set of gates $\{g_0, g_1, \ldots, g_n\}$, with g_0 the output gate. For each gate g_i of C we include a state s_i of $\mathcal{A}$ and a stack symbol γ_i. The initial state of $\mathcal{A}$ is s_0, and all states are accepting. The transitions of $\mathcal{A}$ are defined as follows:

- For each $+$-gate $g_i := g_j + g_k$ in C we include an internal transition from s_i that goes to s_j with probability $1/2$ and to s_k with probability $1/2$.
- For each $*$-gate $g_i := g_j * g_k$ we include a probability-1 call transition from s_i to s_j that pushes γ_k onto the stack.
- An input gate g_i with label 0 contributes no transitions.
- For each input gate g_i with label 1 and each stack symbol γ_j, we include a return transition from s_i that pops γ_j off the stack and ends in state s_j with probability 1.

Recall that acceptance is by empty stack and final state. By construction $\mathcal{A}$ only accepts a single word, as we now explain. Define a sequence of words $w_n \in \{c, r, \iota\}^*$ by $w_0 = \iota$, $w_{n+1} = \iota w_n$ for n even, and $w_{n+1} = c w_n r w_n$ for n odd. Furthermore, write $M_0 = 1$, $M_{n+1} = 2M_n$ for n even, and $M_{n+1} = M_n^2$ for n odd. Then $\mathcal{A}$ accepts w_d with probability N/M_d, where d is the depth of the circuit C and N is output of C. All other words are accepted with probability 0. We conclude that C and C' have the same value if and only if $\mathcal{A}$ and $\mathcal{A}'$ are equivalent.

Proposition 13. *Given weighted VPA $\mathcal{A}$ and $\mathcal{B}$ on the same alphabet Σ one can define a* synchronous-product automaton*, denoted $\mathcal{A} \times \mathcal{B}$, such that $(\mathcal{A} \times \mathcal{B})(w) = \mathcal{A}(w)\mathcal{B}(w)$ for all $w \in \Sigma^*$.*

Proof. Let $\mathcal{A} = (n^{(\mathcal{A})}, \Sigma, \Gamma^{(\mathcal{A})}, M^{(\mathcal{A})}, \boldsymbol{\alpha}^{(\mathcal{A})}, \boldsymbol{\eta}^{(\mathcal{A})})$ and $\mathcal{B} = (n^{(\mathcal{B})}, \Sigma, \Gamma^{(\mathcal{A})}, M^{(\mathcal{B})}, \boldsymbol{\alpha}^{(\mathcal{B})}, \boldsymbol{\eta}^{(\mathcal{B})})$. We define a product automaton $\mathcal{C}$. Note that since the stack height is determined by the input word we can simulate the respective stacks of $\mathcal{A}$ and $\mathcal{B}$ using a single stack in $\mathcal{C}$ whose alphabet is the product of the respective stack alphabets of $\mathcal{A}$ and $\mathcal{B}$.

The number of states of $\mathcal{C}$ is $n^{(\mathcal{A})} \cdot n^{(\mathcal{B})}$. The initial vector $\boldsymbol{\alpha}^{(\mathcal{C})}$ has (i,j)-th component $\boldsymbol{\alpha}_i^{(\mathcal{A})} \cdot \boldsymbol{\alpha}_j^{(\mathcal{B})}$. The final vector $\boldsymbol{\eta}^{(\mathcal{C})}$ is defined likewise. The stack alphabet of $\mathcal{C}$ is $\Gamma^{(\mathcal{A})} \times \Gamma^{(\mathcal{B})}$. Given $a \in \Sigma_c \cup \Sigma_r$ we define the $((i,j),(k,l))$-th component of the transition matrix $M^{(\mathcal{C})}(a, (\gamma, \gamma'))$ to be the product of $M^{(\mathcal{A})}(a, \gamma)$ and $M^{(\mathcal{B})}(a, \gamma')$. □