Probabilistic Automata with Parameters

Rastislav Lenhardt

Supervisors: Dr James Worrell, Dr Joël Ouaknine

Oriel College, Oxford, 2009

Submitted in partial fulfilment of the requirements for the
Degree of Master of Science
in
Mathematics and the Foundations of Computer Science
at the
University of Oxford

Acknowledgements

I am grateful to my supervisors James Worrell and Joël Ouaknine for the great topic and invaluable guidance, support, discussions and suggestions during my work on the dissertation.

I would also like to thank Stephan Kreutzer a lot for guiding me through this master's program.

Finally I appreciate Engineering and Physical Sciences Research Council, Konto Orange (Orange foundation), Nadácia Tatrabanky (Tatrabanka foundation) and Nadácia SPP (SPP foundation) for helping me finance my studies at Oxford.

Abstract

Probabilistic automata were introduced by Rabin in early 1960s. We extend this concept by allowing transition probabilities among the states to be parametric. Two main decision problems are considered: the language equivalence and bisimulation of probabilistic automata.

One of our main results is a new one sided error Monte Carlo randomized polynomial-time algorithm for the universal equivalence. On the other hand for the existential equivalence we show that it is NP-Hard and for the first-order logic equivalence it is PSPACE-Hard. But in both cases it is still decidable.

We present also the polynomial time algorithm for the universal bisimulation and show that the existential bisimulation is NP-Complete.

Keywords: probabilistic automata, parameters, language equivalence, bisimulation

Contents

Chapter 1

Introduction

The concept of finite state machine was first used to model processes. Soon it became very popular and used a lot everywhere. It is especially useful as a model for design and verification of algorithms and hardware.

Rabin and Scott introduced new extended version of deterministic finite automata and presented solutions to many basic decision problems [RS59]. First they added the non-determinism and then in 1963 Rabin introduced in [Rab63] the concept of probabilistic automata. Nowadays they are used a lot in natural language processing, string processing, compilers, verification and modelling of the non-deterministic systems.

A probabilistic automaton is a finite state automaton with probabilistic transitions among the states. We can say that it lies somewhere between completely deterministic finite automata (DFA) and non-deterministic finite automata (NFA). So it is reasonable to expect that decision problems for probabilistic automata will be at least as hard as for DFA, which are in fact just special case of the probabilistic automata with transition probabilities 0 and 1 only, and they might be easier than for NFA.

The most prominent decision problem, considered in the major part of this thesis, is the language equivalence. It asks if two automata accept the same language (in the case of probabilistic automata we ask if any string is accepted with the equal probability by both automata). There is a large gap between DFA and NFA. There is a polynomial-time algorithm for DFA

equivalence problem (it was shown it is NLOG SPACE-Complete [CH92]). On the other hand NFA equivalence is PSPACE-Complete [GJ90]. Now we know that the language equivalence problem for probabilistic automata is in the complexity much closer to DFA. In early nineties W.G. Tzeng presented a polynomial-time algorithm [Tze92] based on finding basis of all reachable probability distributions over the states.

Another decision problem we consider is bisimulation. It is stronger than the language equivalence, because we are not interested only in the accepted language, but also in the structure. It is one of the best tools to minimize number of states in automaton which accepts the same language. There is $O(n \log n)$ Hopcroft algorithm [Hop71] to transform DFA to minimal DFA. Since there is a unique minimal DFA automaton, we can use this algorithm also to check the language equivalence. On the other hand if we want to find minimal NFA it is much harder, it is PSPACE-Complete problem [JR93] [RS97]. In case of probabilistic automata, there is $O(mn^2)$ algorithm [Bai96] and an even faster $O(m \log n)$ algorithm [DHS$^+$03] for probabilistic bisimulation.

The next interesting problem is the language emptiness. For both DFA and NFA we can simply perform breadth first search to decide the problems. It was shown that they are NLOG SPACE-Complete [Jon75]. The same method can be used to decide if there is a string accepted by probabilistic automaton with non-zero probability. However when we ask if there is a string accepted with the probability greater than γ (where $0 < \gamma < 1$), the so called threshold problem (it is natural extension of the language emptiness for probabilistic automata), it is undecidable [Wor08]. It is discussed with the implications in Chapter 2.

Outline

In Chapter 2 we define probabilistic automata and also introduce its parametric extension. After that we present Tzeng's algorithm for the language equivalence of probabilistic automata with slightly improved time complexity. In the next chapter we give a proof that this decision problem is PTIME-

Complete and since it is widely believed that PTIME-Complete problems are inherently sequential, it is probably not parallelizable effectively.

Next we consider universal, existential and first-order logic equivalence of probabilistic automata with parameters in Chapters 4, 5 and 6. In short, the universal problem asks if probabilistic automata are equivalent for all feasible values of parameters, the existential problem asks if there exist feasible values of parameters such that automata are equivalent and the first-order logic problem asks if the given first order logic statement about the equivalence of probabilistic automata is true.

In Chapter 4 we present for universal equivalence a new one-sided error Monte Carlo randomized algorithm that is asymptotically as fast as Tzeng's algorithm for non-parametric automata. It is based on polynomial identity testing and if we fix number of parameters we can derandomize it to get polynomial-time deterministic algorithm.

Next we show that existential and first-order logic equivalence are much harder in Chapters 5 and 6. They are still decidable, but we present reductions which prove that they are respectively NP-Hard and PSPACE-Hard. We give special attention to existential equivalence over the rationals, i.e. if there exist rational feasible values of parameters such that automata are equivalent. We give a proof that it is as hard as an open extended tenth Hilbert problem which asks if a polynomial with integer coefficient has a rational root.

In Chapter 7 we examine bisimulation of probabilistic automata. After recalling the non-parametric algorithm, we give a new polynomial time algorithm in the universal case. We also show an NP algorithm in the existential case and prove that this problem is NP-Complete so we cannot hope to do much better.

We conclude with the summary of results and suggestions for future work.

In the whole thesis we assume that all basic arithmetic operations, i.e. $+$, $-$, $*$, $/$, over $\mathbb{Q}$ are performed in constant time. We also assume that all constants used in input automata are rational numbers.

Chapter 2

Probabilistic Automata

Probabilistic automata are machines with finitely many states and probabilistic transitions among them. Any input string x is accepted with a certain probability.

In this chapter we define more precisely what Probabilistic automata are and introduce necessary notation with the aim to make it as consistent as possible with [Tze92]. We show the basic properties of these automata and very important Tzeng's algorithm. Moreover we define parametric extension of Probabilistic automata.

2.1 Definitions

Definition 2.1. *A vector is stochastic if all its entries are non-negative and sum to* 1*.*

Definition 2.2. *A matrix is stochastic if all its row vectors are stochastic.*

Definition 2.3. *Let span be the function mapping a set of vectors to the vector space generated by them.*

Let $|x|$ be the length of string x, λ be an empty string and $\pi(i, j)$ be the set of all $i \times j$ dimensional stochastic matrices.

Definition 2.4. *A probabilistic automaton U is a 5-tuple (S, Σ, M, ρ, F), where $S = \{s_1, s_2, \ldots s_n\}$ is a finite set of states, Σ is an input alphabet, M*

is a transition function from Σ into $\pi(n, n)$, ρ is an initial distribution over states, and $F \subseteq S$ is a set of accepting states.

The value $M(\delta)[i, j]$ is the probability that automaton moves from the state s_i to the state s_j after reading $\delta \in \Sigma$. We can extend the domain of function M from Σ to Σ^*: $M(x\delta) = M(x)M(\delta)$ for $x \in \Sigma^*$ and $\delta \in \Sigma$. Note that then $M(\lambda)$ is the identity matrix.

We represent final states by n dimensional row vector η_F such that $\eta_F[i] = 1$ iff $s_i \in F$ and 0 otherwise.

Definition 2.5. *The state distribution induced by string $x \in \Sigma^*$ is*

$$P_U(x) = \rho M(x)$$

In other words $P_U(x)[i]$ is the probability that U moves to s_i after reading string x with the initial distribution ρ.

The probability for U to accept $x \in \Sigma^*$ is therefore $P_U(x)(\eta_F)^T$. If we want, we can represent initial state of the automaton by setting $\rho = (1, 0, \ldots, 0)$.

Definition 2.6. *The language accepted by probabilistic automaton U is*

$$L_U = \{(x, P_U(x)(\eta_F)^T) : x \in \Sigma^*\}$$

Example 2.1. Consider an automaton with two states on $\Sigma = \{0, 1\}$ with the initial distribution $\rho = (1, 0)$ and transition matrices:

$$M(0) = \begin{pmatrix} 1 & 0 \\ 1/2 & 1/2 \end{pmatrix} \quad M(1) = \begin{pmatrix} 1/2 & 1/2 \\ 0 & 1 \end{pmatrix}$$

What is the state distribution after reading string $x = \delta_1\delta_2 \ldots \delta_n$, where $\delta_i \in \Sigma$? It is $(1 - p, p)$, where $p = 0.\delta_1\delta_2 \ldots \delta_n$ in binary.

Definition 2.7 (Language equivalence)**.** *Probabilistic automata U_1 and U_2 are said to be language equivalent (for short, we use only equivalent in the rest of the text) if for all strings $x \in \Sigma^*$ the two automata accept x with the*

same probability, i.e.

$$P_{U_1}(x)(\eta_{F_1})^T = P_{U_2}(x)(\eta_{F_2})^T \text{ for all } x \in \Sigma^*$$

Language inclusion

We show later in this chapter the algorithm to solve the language equivalence problem. The language inclusion problem is more problematic.

Definition 2.8 (Language inclusion)**.** *Language accepted by automaton U_1 is said to be the subset of language accepted by automaton U_2 if*

$$P_{U_1}(x)(\eta_{F_1})^T \leq P_{U_2}(x)(\eta_{F_2})^T \text{ for all } x \in \Sigma^*$$

Let L_γ be the language of probabilistic automaton which accepts each string with the probability γ. Then the language inclusion problem $L_U \leq L_\gamma$ is equivalent to the threshold problem (natural extension of the language emptiness for probabilistic automata).

Definition 2.9. *The threshold problem is: given probabilistic automaton U and $0 < \gamma < 1$, does there exist a string $x \in \Sigma^*$ such that it is accepted with probability greater than γ, i.e. $P_U(x)(\eta_F)^T > \gamma$?*

It was shown that the threshold problem is undecidable [Wor08] and so also the language inclusion is undecidable.

2.2 Properties

Lemma 2.1. *Probabilistic automata U_1 and U_2 are equivalent if for all strings x of length at most $n_1 + n_2 - 1$ the two automata accept x with equal probability.*

Proof. This was shown in [Paz71]. We can also see it from the Tzeng's algorithm described below since we can have at most $n_1 + n_2$ internal nodes and we have to have at least one internal node corresponding to a string of length 0, one to a string of length 1, etc. □

Corollary 2.2. *Lemma 2.1 implies that the equivalence problem is in the complexity class coNP.*

2.3 Tzeng's algorithm

The algorithm was shown in [Tze92]. The basic idea is to combine two input automata into one, find the basis of a span of reachable state distributions and then to check now only for basis that the same proportion is in the accepting states in both automata.

Definition 2.10. *Let $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1)$ and $U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$ be two probabilistic automata with n_1 and n_2 states. We define combination of automata $U_1 \oplus U_2$ to have set of states the disjoint union of S_1 and S_2. Its transition function is*

$$M_{U_1 \oplus U_2}(x) = \begin{pmatrix} M_1(x) & 0_{n_1 \times n_2} \\ 0_{n_2 \times n_1} & M_2(x) \end{pmatrix}$$

so we have $M_{U_1 \oplus U_2}(x\delta) = M_{U_1 \oplus U_2}(x) M_{U_1 \oplus U_2}(\delta)$. We also define

$$P_{U_1 \oplus U_2}(x) = [\rho_1, \rho_2] M_{U_1 \oplus U_2}(x)$$

Now we can reformulate equivalence of two Probabilistic automata U_1 and U_2 to be if and only if

$$\forall x \in \Sigma^*, P_{U_1 \oplus U_2}(x)[\eta_{F_1}, -\eta_{F_2}]^T = 0$$

Definition 2.11. *Let $H(U_1, U_2) = \{P_{U_1 \oplus U_2}(x) : x \in \Sigma^*\}$.*

Lemma 2.3. *Let V be a basis for span$(H(U_1, U_2))$. Then probabilistic automata U_1 and U_2 are equivalent if and only if $\forall v \in V, v[\eta_{F_1}, -\eta_{F_2}]^T = 0$.*

Proof. It follows from the fact that each state distribution (for any input string x) is a linear combination of basis vectors. □

Note that V has at most $n_1 + n_2$ elements, because the dimension of $span(H(U_1, U_2))$ is at most $n_1 + n_2$. So if we are able to find basis V of $span(H(U_1, U_2))$ in polynomial time then we can solve the equivalence problem for probabilistic automata in polynomial time.

Description of the algorithm

Without loss of generality we consider $\Sigma = \{0, 1\}$. We define a binary tree T, such that its nodes are strings $x \in \Sigma^*$. Its root is $node(\lambda)$ and each $node(x)$ has two children: $node(x0)$ and $node(x1)$. Moreover, for each $node(x)$ we have corresponding state distribution induced by string x on automata: $P_{U_1 \oplus U_2}(x)$.

We find the basis V of $span(H(U_1, U_2))$ by pruning the tree T. Initially V is empty. We visit nodes of T in breadth-first search. If a state distribution for the visited node is linearly independent of V we add it to the set V. Otherwise we prune the subtree of the visited node. We terminate when all nodes are either visited or pruned. We prove in the next part that the resulting set V forms a basis of $span(H(U_1, U_2))$. If we choose another method of traversing we will get other basis, but it will not affect the results. Finally, at the end of the Algorithm 2.3.1 we check if the condition from Lemma 2.3 holds.

Correctness

Let T' be the tree formed by visited (i.e. not pruned) nodes of T. T' has at most $n_1 + n_2$ internal nodes (the nodes whose corresponding state distributions are in V), because $|V| \leq n_1 + n_2$ and at most $n_1 + n_2 + 1$ leaves (nodes, where subtrees were pruned, because corresponding state distributions were linearly dependent with V).

We prove that vectors in the resulting set V forms a basis of $span(H(U_1, U_2))$. Let $V_i = \{P_{U_1 \oplus U_2}(xy) : node(x) \text{ is a leaf}, |y| = i\}$. So set V_0 contains state distributions corresponding to all the leaves (of T') and sets V_i for $i \geq 1$ contains all the state distributions of nodes of T which have distance i from a leaf.

Algorithm 2.3.1 Tzeng's algorithm for equivalence of probabilistic automata

Require: $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1), U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$
Set V to be the empty set
Set Q to be the empty queue
Add $node(\lambda)$ to Q
while Q is not empty **do**
  take $node(x)$ from the queue Q
  if $P_{U_1 \oplus U_2}(x) \notin span(V)$ **then**
    add $node(x0)$ to Q
    add $node(x1)$ to Q
    add vector $P_{U_1 \oplus U_2}(x)$ to V
  end if
end while
if $\forall v \in V, v[\eta_{F_1}, -\eta_{F_2}]^T = 0$ **then**
  return 'yes'
else
  return 'no'
end if

Note that

$$span(V \cup \bigcup_{i=0}^{\infty} V_i) = span(P_{U_1 \oplus U_2}(x) : x \in \Sigma^*) = span(H(U_1, U_2))$$

We will prove that $\forall i \geq 0, V_i \subseteq span(V)$ by induction.

The base case $V_0 \subseteq span(V)$ follows from the algorithm. Let $V = \{v_1, \dots v_r\}$. Suppose that $V_i \subseteq span(V)$ and for any x such that $node(x)$ is a leaf and for any y such that $|y| = i$ and for any $\delta \in \Sigma$ we have:

$$P_{U_1 \oplus U_2}(xy\delta) = P_{U_1 \oplus U_2}(xy) M_{U_1 \oplus U_2}(\delta) = (\sum_{i=1}^{r} c_i v_i) M_{U_1 \oplus U_2}(\delta) =$$

$$= \sum_{i=1}^{r} c_i (v_i M_{U_1 \oplus U_2}(\delta)) \in span(V \cup V_0) = span(V)$$

as required.

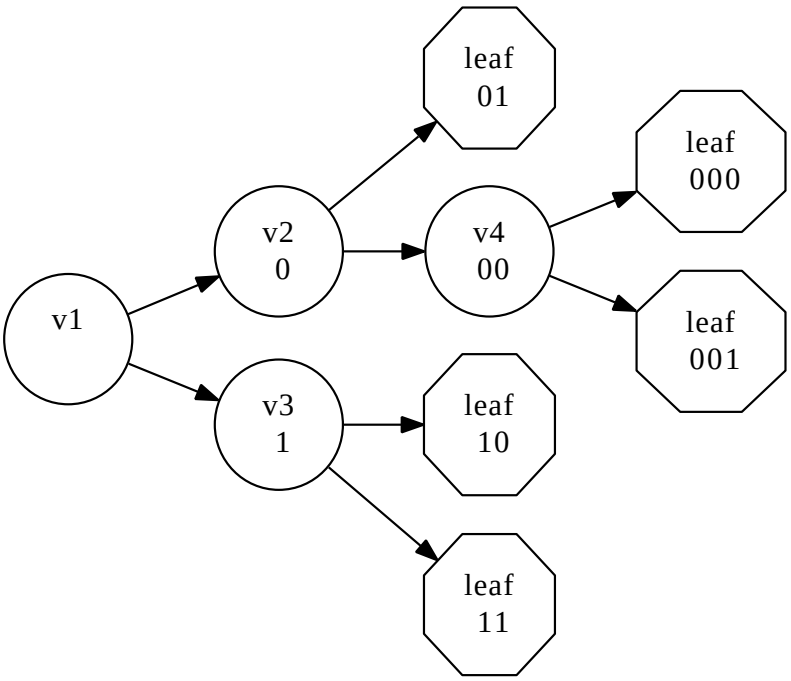


Figure 2.1: Example of tree T'

Complexity

The vector $P_{U_1 \oplus U_2}(x\delta)$ for $node(x\delta)$ is computed using the value for its parental node using the fact that $P_{U_1 \oplus U_2}(x\delta) = P_{U_1 \oplus U_2}(x) M_{U_1 \oplus U_2}(\delta)$. It can be done in time $O((n_1+n_2)^2)$ $O(n_1+n_2)$ times contributing overall by time $O((n_1+n_2)^3)$. To verify if a set of n_1+n_2 vectors is linearly independent, we can use Gaussian elimination or check if the matrix is singular. We can do it in time $O((n_1+n_2)^3)$. We need to verify linear independence $O(n_1+n_2)$ times, so the overall complexity will be $O((n_1+n_2)^4)$.

We improved this complexity result from [Tze92] to $O((n_1+n_2)^3)$. When we use Gaussian elimination then by reusing the previously computed triangular form, even if we need to verify linear independence $O(n_1+n_2)$ times, the overall complexity will stay $O((n_1+n_2)^3)$.

Remark 2.1. Tzeng's algorithm works perfectly well also if some transition weights are outside of the range $[0, 1]$, negative or not sum to 1.

2.4 Probabilistic automata with parameters

Informally, we allow transition probabilities among states to be parametric. We allow them to be linear functions (with rational coefficients) of parameters. For example $3 + 2y_1 - \frac{1}{2}y_3$.

Definition 2.12. *A probabilistic automaton with parameters U is a 5-tuple (S, Σ, M, ρ, F), where $S = \{s_1, s_2, \ldots s_n\}$ is a finite set of states, Σ is an input alphabet, M is a transition function from Σ into $\pi(n, n)$, ρ is an initial distribution over states, and $F \subseteq S$ is a set of accepting states. Transition probabilities defined by a transition function M can be linear functions (with rational coefficients) of parameters $y = (y_1, \ldots y_k)$ such that $-1 \leq y_i \leq 1$.*

With each probabilistic automaton with parameters U, we have associated a matrix A and a vector b such that $Ay \leq b$ is a set of constraints for parameters y that are directly implied by

- bounds on parameters: $-1 \leq y_i \leq 1$
- the necessity of transition probabilities being between 0 and 1
- the necessity of transition matrices $M(\delta)$ being stochastic, i.e. the outgoing probabilities for any state s after reading any input symbol δ must sum to 1.

Note that in condition $-1 \leq y_i \leq 1$ we can exchange 1 for any other constant C without changing the power of the automaton. We introduce these bounds on parameters to make things simpler.

We could also allow extending constraints forced by the automata by some additional external constraints of the form $cy^T \leq d$, where c is a row vector and d is a constant. It would not alter any results and it is possible to encode them straight into automata.

Definition 2.13. *Let U be a probabilistic automaton with parameters. We denote by $U(y)$ a non-parametric probabilistic automaton which we get from U by plugging in the values of parameters y.*

Example 2.2. We show an example of Probabilistic automaton with parameters U (see Figure 2.2) and all the necessary constraints for parameters. In our case $S = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{\delta\}$, $\rho = (1, 0, 0, 0)$, $F = \{q_3\}$ and

$$M(\delta) = \begin{pmatrix} 0 & y & 0 & 1/2 + z - y \\ 0 & 0 & 1/2 + y & 1/2 - y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Probability of accepting an empty string λ and string δ is 0 and probability of accepting string $\delta\delta \dots \delta$ for $|\delta\delta \dots \delta| \geq 2$ is $y(1/2 + y)$.

We have the following constraints:

$$\begin{aligned} -1 \leq y, z \leq 1 \\ 0 \leq 1/2 + y \leq 1 \\ 0 \leq 1/2 - y \leq 1 \\ 0 \leq y \leq 1 \\ 0 \leq 1/2 + z - y \leq 1 \\ y + 1/2 + z - y = 1 \\ y + 1 - y = 1 \end{aligned}$$

It can be seen that they are equivalent to $0 \leq y \leq 1/2$ and $z = 1/2$.

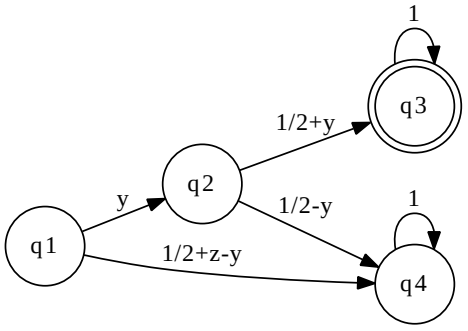


Figure 2.2: Example of Probabilistic automaton with parameters

Chapter 3

Equivalence of Probabilistic Automata is P-Complete

In this chapter we consider probabilistic automata without parameters and show that the problem of their equivalence is P-Complete. It is widely believed that P-Complete problems are inherently sequential, i.e. not in the classes NC^k of problems that can be solved by parallel RAMS in polynomial time, and so this problem is probably not parallelizable effectively.

Theorem 3.1. *The equivalence problem for Probabilistic Automata is P-Complete.*

Proof. Since we have a polynomial-time algorithm which solves this problem, it is in the complexity class P. Recall that if one problem is P-Hard and is LOGSPACE reducible to the second problem then the second problem is also P-Hard. To prove P-Hardness of equivalence of Probabilistic Automata we show a LOGSPACE reduction from the monotone boolean circuit value problem, mCVP for short, which is well known to be P-Complete (see for example [GR88]).

Definition 3.1. *A monotone boolean circuit is a directed acyclic graph that contains input nodes with indegree* 0 *and internal nodes with indegree* 2*. There is exactly one output node, with outdegree* 0*. Input nodes are assigned boolean values (i.e. either* 0 *or* 1*) and all internal nodes are labelled with either* $\vee$ *or* $\wedge$ *which describes the operation (either disjunction or conjunction)*

which is used to compute the value of this node from two input values. The value of output node is the output value of the circuit.

For example see Figure 3.1, where description of nodes is in the form "# index of node, label of node".

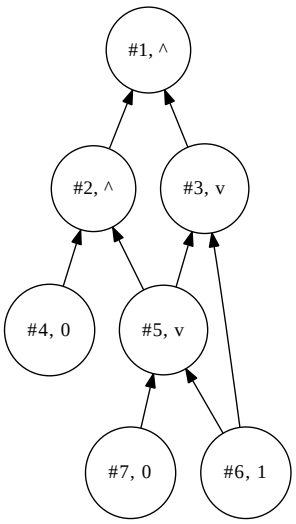


Figure 3.1: An instance of mCVP

Definition 3.2. *The monotone boolean circuit value problem is asking if the output of a given circuit is* 1.

Let us consider an instance of mCVP in the form as in [SJ01]. Let $V = \{1, \dots n\}$ be a set of nodes. For every node let $l(i)$, $r(i)$ to be nodes such that in our graph there are edges $l(i) \to i$ and $r(i) \to i$. Let an input instance of mCVP consists of number of nodes n and of values $l(i)$, $r(i)$ and $label(i)$ (i.e. one of $0, 1, \vee, \wedge$) for each node i.

Reduction

Our goal is to construct from an instance of mCVP a probabilistic automaton U_1 which is equivalent to probabilistic automaton U_2 which accepts anything with probability 0 if and only if the output value of $mCVP$ is 0.

We construct probabilistic automata from an instance of mCVP in the following way:

- The input alphabet is $\Sigma = \{\delta\}$. From now on when we say transition we mean transitions after reading an input symbol δ.
- The set of states of automaton U_1 is $S_1 = \{s_i, e_i \text{ ; for } 1 \leq i \leq n\} \cup \{q_{bad}\}$. So we have one start state s_i and one end state e_i for each node $i \in V$.
- For input nodes i we define transition probability $s_i \to e_i$ to be
  - 0 if $label(i) = 0$
  - 1 if $label(i) = 1$
- For non-input nodes i with $label(i) = \vee$ we define the following transition probabilities:
  - 1/2 for transition $s_i \to s_{l(i)}$
  - 1/2 for transition $s_i \to s_{r(i)}$
  - 1 for transition $e_{l(i)} \to e_i$
  - 1 for transition $e_{r(i)} \to e_i$
- For non-input nodes i with $label(i) = \wedge$ we define the following transition probabilities:
  - 1 for transition $s_i \to s_{l(i)}$
  - 1 for transition $e_{l(i)} \to s_{r(i)}$
  - 1 for transition $e_{r(i)} \to e_i$
- All non-defined transitions from any state go to a special state q_{bad}

- Let k be an output node. Then e_k is the only one accepting state and the whole initial distribution is in state s_k.
- We construct automaton U_2 such that it accepts any string with probability 0. For example it can have just one non-accepting state.

Correctness

First note that the above reduction is LOGSPACE. We prove by induction (from the bottom to the top of an mCVP instance) that the probability of getting from a state s_i to a state e_i (after one or more transitions) is zero if and only if the evaluated value of node i of mCVP is 0.

If $label(i) \in \{0, 1\}$ then the statement is true.

If $label(i) = \vee$ then, by the construction above, the probability of getting from the state s_i to the state e_i is non zero if and only if probability of getting from $s_{l(i)}$ to $e_{l(i)}$ is non zero or if probability of getting from $s_{r(i)}$ to $e_{r(i)}$ is non zero, which is by induction exactly when at least one of the states $l(i)$ and $r(i)$ is evaluated to 1.

Similarly if $label(i) = \wedge$ then, by the construction above, the probability of getting from the state s_i to the state e_i is non zero if and only if probability of getting from $s_{l(i)}$ to $e_{l(i)}$ is non zero and if probability of getting from $s_{r(i)}$ to $e_{r(i)}$ is non zero, which is by induction exactly when both of the states $l(i)$ and $r(i)$ are evaluated to 1.

Let k be an output node. It follows that probability of getting from the only initial state s_k to the only accepting state e_k is 0 if and only if the evaluated value of the output node (so also the circuit) is 0. Moreover since automaton U_2 does accept any string with probability 0, it is if and only if automata U_1 and U_2 are equivalent. □

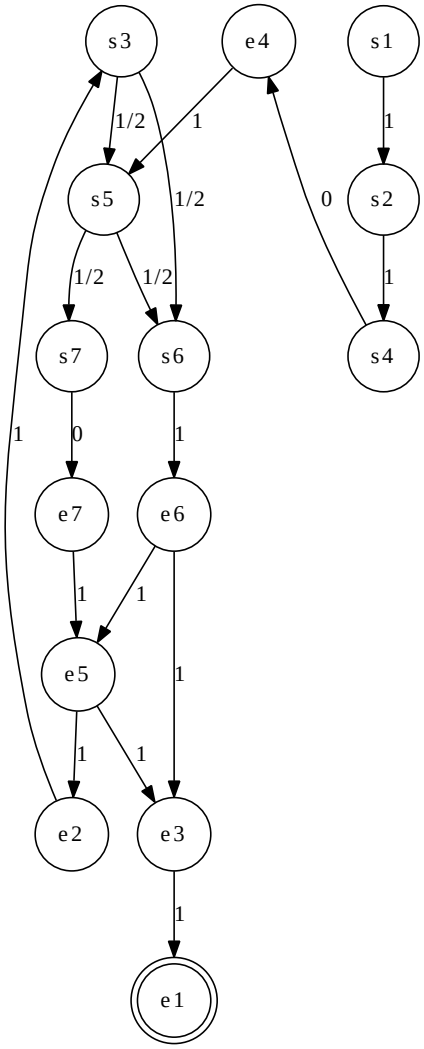


Figure 3.2: Reduction to automaton U_1 from mCVP in Figure 3.1

Chapter 4

Universal equivalence of probabilistic automata with parameters

Universal equivalence of probabilistic automata with parameters U_1 and U_2 is a decision problem asking if for all feasible values of parameters $y = (y_1, y_2, \ldots, y_k)$ by plugging them into automata we get equivalent probabilistic automata, i.e. if

$$L_{U_1(y)} = L_{U_2(y)} \text{ for all feasible values of } y$$

4.1 Preliminaries

Theorem 4.1. *Probabilistic automata with parameters U_1 and U_2 are equivalent for all feasible values of parameters $y = (y_1, y_2, \ldots, y_k)$ if and only if*

$$Q(y) = \sum_{x \in \Sigma^*, |x| \leq n_1 + n_2 - 1} (P_{U_1}(x)(\eta_{F_1})^T - P_{U_2}(x)(\eta_{F_2})^T)^2 = 0$$

for all feasible values of parameters y.

Proof. Fix an instantiation of parameters. According to Lemma 2.1 probabilistic automata U_1 and U_2 are equivalent if and only if any string x of length

at most n_1+n_2-1 is accepted by both of them with equal probability. In other words, it is true when for any such string x it is true that $P_{U_1}(x)(\eta_{F_1})^T - P_{U_2}(x)(\eta_{F_2})^T = 0$ for all feasible parameters y. Note that if this is true for all such strings x then also the whole $\sum_{x\in\Sigma^*,|x|\leq n_1+n_2-1}(P_{U_1}(x)(\eta_{F_1})^T - P_{U_2}(x)(\eta_{F_2})^T)^2$ is equal to 0. On the other hand, if there is a string x' such that $P_{U_1}(x')(\eta_{F_1})^T - P_{U_2}(x')(\eta_{F_2})^T \neq 0$ then $(P_{U_1}(x')(\eta_{F_1})^T - P_{U_2}(x')(\eta_{F_2})^T)^2 > 0$. It follows that the whole sum is then also positive. □

Theorem 4.2. *(Schwartz-Zippel Theorem) Let $Q(y_1, \dots y_k) \in F[y_1, \dots, y_k]$ be a multivariate polynomial of total degree d over the field F. Fix any finite set $S \subseteq F$, and let $r_1, \dots r_k$ be chosen independently and uniformly at random from S. Then*

$$\mathbb{P}[Q(r_1, \dots r_k) = 0 | Q(y_1, \dots y_k) \not\equiv 0] \leq \frac{d}{|S|}$$

Proof. Proof can be found for example in the textbook [MR95]. □

4.2 1MC randomized polynomial time algorithm

Our algorithm 4.2.1 for equivalence of probabilistic automata with parameters is based on the polynomial identity testing. Theorem 4.1 tells us that automata U_1 and U_2 are equivalent iff the given polynomial $Q(y) = 0$ for all feasible parameters y. So we can randomly sample feasible values of y (we show later how to do this) and then perform polynomial identity test.

Let y'' be randomly sampled feasible value of parameters y. The problem is that we cannot evaluate the polynomial $Q(y'')$ directly as it has an exponential number of terms in general. However we can run in polynomial-time Tzeng's algorithm for equivalence of probabilistic automata instead, which can determine if for the given y'' the two automata are equivalent, i.e. whether polynomial $Q(y'') = 0$.

Algorithm 4.2.1 1MC randomized polynomial time algorithm for equivalence of probabilistic automata with parameters

Require: $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1), U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$
Require: $Ay \leq b$ are constraints on parameters y from U_1, U_2
if $LinearProgramming(Ay \leq b, \text{minimize } y_1)$ has no feasible solution **then**
  return 'constraints on parameters are inconsistent'
end if
$y' \leftarrow$ feasible solution of $LinearProgramming(Ay \leq b, \text{minimize } y_1)$
Set V to be an empty set
$V \leftarrow$ neighbour vertices of vertex y' by using the method in Simplex algorithm
Set R to be an integer array of the same size as V
for all $R[i] \in R$ **do**
  $R[i] \leftarrow$ random number between 0 and $4(|S_1| + |S_2|)$, inclusive
end for
$y'' \leftarrow y' + \sum_i R[i](V[i] - y')$
Set U_1' to be U_1 given that parameter values are y''
Set U_2' to be U_2 given that parameter values are y''
if $ProbabilisticAutomataWithoutParametersEquivalent(U_1', U_2')$ **then**
  return 'equivalent'
else
  return 'not equivalent'
end if

4.2.1 Sampling

In this section we show how we can sample efficiently random feasible parameter values y''.

Recall that $y = (y_1, y_2, \ldots, y_k)$ and consider a matrix A and a vector b, which contain all linear constraints on parameters from both automata U_1 and U_2 in the form $Ay \leq b$. Let L be the subset of the k-dimensional Euclidean space spanned by the variables $y_1, \ldots, y_k$, which contains all feasible solutions of the system of linear inequalities $Ay \leq b$. Then $0 \leq dim(L) \leq k$ (number of free variables is at most k). It can be lower than k if there are some dependencies across variables (for example $2y_1 = 1 - y_3$ or $y_2 = 0.25$). Note that L is a convex polytype since it is an intersection of half-spaces.

Set $m = dim(L)$ and relabel the variables such that $y_1, y_2, \ldots y_m$ are free

variables (a variable is free if its value is not determined by setting the values of previous variables) and then $y_{m+1}, \dots y_k$ are non-free variables. There is a deterministic way how to determine these dependencies across variables (see Appendix A for details). We can find these dependencies across variables step by step to rewrite the whole system of inequalities and polynomial $Q(y)$ only in variables $y_1, \dots, y_m$.

Theorem 4.3. *Polynomial $Q(y_1, y_2, \dots y_m) = 0$ for all feasible values $y_1, \dots, y_m$ if and only if $Q(y_1, y_2, \dots y_m) \equiv 0$ (i.e. for all $y_1, \dots, y_m$).*

Proof. It follows directly from the fact that if two polynomials Q_1, $Q_2 \in \mathbb{R}[y_1, \dots y_m]$ are equal on a convex m-dimensional polytype $C \subset \mathbb{R}^m$ then they are equal on the whole space $\mathbb{R}^m$. □

Corollary 4.4. *There is a deterministic exponential time algorithm which can decide if two probabilistic automata with parameters are equivalent.*

Proof. First we eliminate non-free variables (see Appendix A) and then we check if the polynomial $Q(y_1, \dots y_m) \equiv 0$ by expanding it and comparing the coefficients. □

Linear Programming and Simplex algorithm overview

Definition 4.1. *An instance of Linear Programming (LP) consists of a set of linear inequalities $Ay \leq b$ and the objective function $f(y) = c^T y$ for some vector $c \in \mathbb{R}^n$. The goal is to find the solution of $Ay \leq b$ with minimal value of the objective function.*

There are three possible outcomes when solving an instance of Linear Programming:

1. LP has an optimal solution
2. LP has no solutions
3. LP has solutions, but none is optimal (when f is unbounded on $Ay \leq b$)

Lemma 4.5. *In our algorithm 4.2.1 only first two outcomes are possible.*

Proof. It follows from the definition of Probabilistic Automata with parameters, from the constraints on variables: $-1 \leq y_i \leq 1$. Therefore L is bounded and also values of f on it. □

The short overview of Simplex algorithm follows. The detailed reference with careful explanation and proofs can be found for example in [CLRS01].

Simplex algorithm

- The set of feasible solutions of the system of linear inequalities forms a convex polytype as it can be seen for example in Figure 4.1[1].
- The objective function $c^T y = v$ is a hyperplane which we move as we try to minimize v. Therefore the minimum is certainly achieved in one of the vertices (but it does not not have to be at this vertex only).
- The algorithm first finds a feasible solution which is also a vertex of the polytype.
- Then it tries to move to one of the neighbouring vertices, where the objective function is minimal. This process is called Pivoting. Algebraically it means trying to increase (or decrease) one variable at a time in the system of linear inequalities while maintaining the feasibility of the solution.
- Once the algorithm cannot improve objective function by going to neighbouring vertex it returns the best solution so far. This is alright, because on a convex region local optimum of a linear objective function is always also the global optimum.

Efficient sampling

Suppose we have a convex polytype $P = \{y \in \mathbb{R}^n : Ay \leq b\}$. We can find one of its vertices (call this vertex v) by running any algorithm for Linear

[1]This picture is part of Wikimedia Commons, a freely licensed media file repository

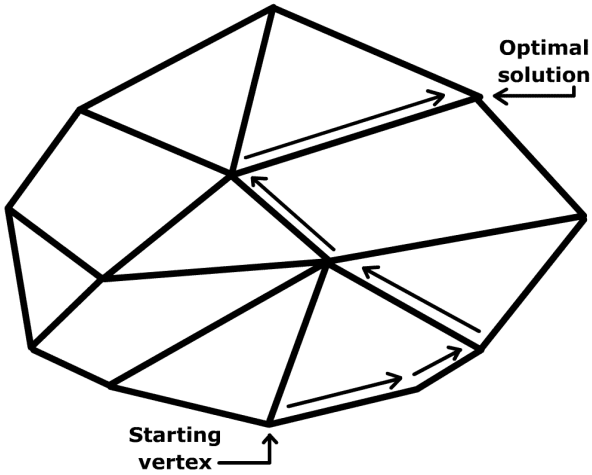


Figure 4.1: Multidimensional convex polytype during Simplex algorithm

Programming. Let V be a set of neighbour vertices of vertex v, the same set of neighbour vertices as considered by simplex algorithm during Pivoting. Let C be a "corner", i.e. the convex polytype determined by the vertex v and its neighbouring vertices in V. We refer to these defined objects (P, V, C, v) in the following theorem.

Theorem 4.6. *The subspace spanned by the convex polytype P is the same subspace as that spanned by C.*

Proof. Clearly, the subspace spanned by C is a subset of the space spanned by P. Now consider any point $p \in P \setminus C$. From the convexity of P the whole segment pv lies inside P. Let cv be a segment which is intersection of pv and C. If $c \neq v$ then the point p is part of the span of C since vector pv is just prolonged vector cv, which is part of the span. Otherwise $c = v$ and it implies that p should be a neighbour of v, which is a contradiction with $p \in P \setminus C$. □

So we can sample points y'' randomly as you can see in Algorithm 4.2.1 from the subspace spanned by a vertex v and vertices in V.

The advantage of this sampling is that it captures all the dependencies across variables. So by applying Theorem 4.3 we have that $Q(y'')$ is zero on P iff it is zero on the whole subspace of $\mathbb{R}^k$ spanned by v and V.

4.2.2 Error probability

Consider the polynomial $Q(y)$ from Theorem 4.1. And let $n = n_1 + n_2$ be overall number of states of both automata. We consider only words of length less than n and so total degree of $Q(y)$ is less than $2n$. In our Algorithm 4.2.1 the fixed subset from which we sample has size at least $4n$ (Except the degenerate case, when $dim(L) = 0$, i.e. when there exists only one feasible solution to the system of linear inequalities $Ay \leq b$. In that case it is sufficient to plug it in and we obtain the answer with no error probability). Therefore if automata are not equivalent, according to Schwartz-Zippel Theorem (Theorem 4.2), the probability of error is

$$\mathbb{P}[Q(y'') = 0 | \exists \text{ feasible } y : Q(y) \neq 0] < \frac{2n}{4n} = \frac{1}{2}$$

On the other hand if automata are equivalent then the algorithm always correctly answers that they are equivalent.

4.2.3 Complexity

To find the vertex v, the basic feasible solution of LP problem, we can use one of the fastest known Linear Programming algorithms. For example Karmakar's algorithm (see [Kar84]) runs in time $\mathrm{O}(k^{3.5})$, where k is number of variables. Then we can determine neighbours of v in $\mathrm{O}(n)$ time using Pivoting and Tzeng's algorithm for equivalence of probabilistic automata without parameters run in $\mathrm{O}(n^3)$ time. So the overall time complexity of the Algorithm 4.2.1 is $\mathrm{O}(n^3 + k^{3.5})$. Usually $k = \mathrm{O}(n)$ then the time complexity is simply $\mathrm{O}(n^{3.5})$.

4.2.4 Repeating of the algorithm

By constant number of repetitions of the Algorithm 4.2.1 we can decrease its error probability under any positive constant.

By repeating it $\log^* n$ times we get algorithm with time complexity $\mathrm{O}(n^{3.5} \log^* n)$ with error probability $\to 0$ as $n \to \infty$.

4.3 Derandomization of the algorithm

We already showed in Corollary 4.4 how we can solve deterministically universal equivalence. Another possible approach is to derandomize algorithm 4.2.1. Instead of sampling randomly from the given set, we can test if a polynomial is zero at sufficiently many points in the set. Schwartz-Zippel's Theorem tells us that if a polynomial is not identically zero then it can be zero in at most half of the sampled points. So if we check one more than half of them and the polynomial is zero in all of them we can be sure that the polynomial is a zero polynomial.

Suppose that we have k parameters, i.e. a k-variate polynomial. For each point we can run polynomial time Tzeng's algorithm to determine if the polynomial is zero at that point. It follows that if we fix k to be a constant we have a deterministic polynomial time algorithm solving universal equivalence of Probabilistic automata with parameters in time $\mathrm{O}(n^3(4n+1)^k)$.

Chapter 5

Existential equivalence of probabilistic automata with parameters

The existential equivalence of probabilistic automata with parameters U_1 and U_2 is a decision problem asking if there exist feasible values of parameters $y = (y_1, y_2, \ldots, y_k)$ such that by plugging them into automata we get equivalent probabilistic automata, i.e. if

$$\exists \text{ feasible values of } y, L_{U_1(y)} = L_{U_2(y)}$$

5.1 Existential equivalence is NP-Hard

Theorem 5.1. *The existential equivalence problem is NP-Hard.*

Proof. The basic idea is to encode 3SAT into our automata. First we encode equalities $y_i(1 - y_i) = 0$ which forces all parameters to be either 0 or 1 and then we encode any 3SAT formula into automata in such a way that the 3SAT is satisfiable if and only if there exist feasible parameters such that the constructed automata are equivalent. The in detail construction of automata follows.

- As input we have a 3SAT formula which consist from m clauses $C_1, \ldots C_m$.

Each clause is a conjunction of 3 literals, but some literals can be present in more clauses (in both positive way or in negation). So we have $t \leq 3m$ literals.

- Our two automata U_1 and U_2 will have the same finite input alphabet $\Sigma = \{a_1, a_2, \ldots a_m, b_1, \ldots b_t\}$.

- Both automata will have two special states. The state having the whole initial distribution: q_{init} (in U_1 and q'_{init} in U_2) and the state q_{bad} (in U_1 and q'_{bad} in U_2) where all remaining transition probabilities not defined elsewhere will go. For example if we define transition probability from the state q_0 to other states after reading an input symbol 'a' to be 0.75 together then with the remaining probability of 0.25 the automaton will move to state q_{bad}. All these special states are not accepting and being in state q_{bad} (or q'_{bad}) the automaton will stay there with probability 1 after reading any input symbol.

- Now we encode equalities $y_i(1 - y_i) = 0$ which forces all parameters (literals from 3SAT) to be either 0 or 1.

  For each literal y_i we will have two new states: non-accepting $q_{(i,1)}$ and accepting $q_{(i,2)}$ in U_1 with transition probability after reading input symbol b_i being y_i from state $q_{init} \to q_{(i,1)}$ and probability $1 - y_i$ from state $q_{(i,1)} \to q_{(i,2)}$.

  In U_2 we will not have any new states, so after input being $b_i b_i$ U_2 will move from the state q'_{init} directly to q'_{bad} and so accepts $b_i b_i$ with probability 0. Therefore to have automata U_1 and U_2 equivalent, $y_i(1 - y_i)$ must be 0, i.e. y_i must be either 0 or 1.

- In the last step we encode the whole formula into U_1. For each clause C_i we have three new states $q_{[i,1]}$, $q_{[i,2]}$ and $q_{[i,3]}$. Only the third of them is accepting.

  For input symbol a_i we now define transition probabilities $q_{init} \to q_{[i,1]}$, $q_{[i,1]} \to q_{[i,2]}$ and $q_{[i,2]} \to q_{[i,3]}$ according to literals in clause C_i. If the first literal is positive, say y_h, then the transition probability $q_{init} \to$

$q_{[i,1]}$ will be $1 - y_h$. If it is negative then the transition probability will be y_h. We define similarly also transition probability $q_{[i,1]} \to q_{[i,2]}$ according to the second literal and transition probability $q_{[i,2]} \to q_{[i,3]}$ according to the third literal in C_i.

Automaton U_2 does not accept anything and so automata U_1 and U_2 are equivalent if only if for all clauses there is no assignment of transition probabilities which allows $q_{init} \to q_{[i,1]} \to q_{[i,2]} \to q_{[i,3]}$, i.e. if there is no clause which is not satisfiable.

For clarification see example in Figure 5.1 for formula $(y_1 \vee y_2 \vee \neg y_3) \wedge (\neg y_1 \vee y_2 \vee y_4)$.

□

5.2 Decidability and hardness of the problem

Lemma 5.2 (Existential Theory of the Reals)**.**
The sentence $(\exists X_1) \dots (\exists X_k) F(X_1, \dots, X_k)$*, where* $F(X_1, \dots, X_k)$ *is a quantifier free formula with coefficients in a real closed field R, is decidable. There is a* $PSPACE$ *algorithm for this problem.*

Proof. A good reference containing the solution to this problem is Chapter 13 in [BPR06]. □

Theorem 5.3. *The existential equivalence problem is decidable. It is in the complexity class* $PSPACE$*.*

Proof. We reduce the problem to the Existential theory of the reals described in Lemma 5.2, which is well known to be in $PSPACE$. First we construct the polynomial $Q(y)$ as in Theorem 4.1.

Let $F'(y)$ be a quantifier free boolean formula which contains all inequalities for parameters, which are forced by automata (i.e. implied by the fact that transition probabilities must be between 0 and 1,inclusive, and sum to 1 for each state and input symbol).

Now we would like to use Lemma 5.2 for the formula $F(y) = Q(y) \wedge F'(y)$, but the problem is that the polynomial $Q(y)$ has exponentially many terms.

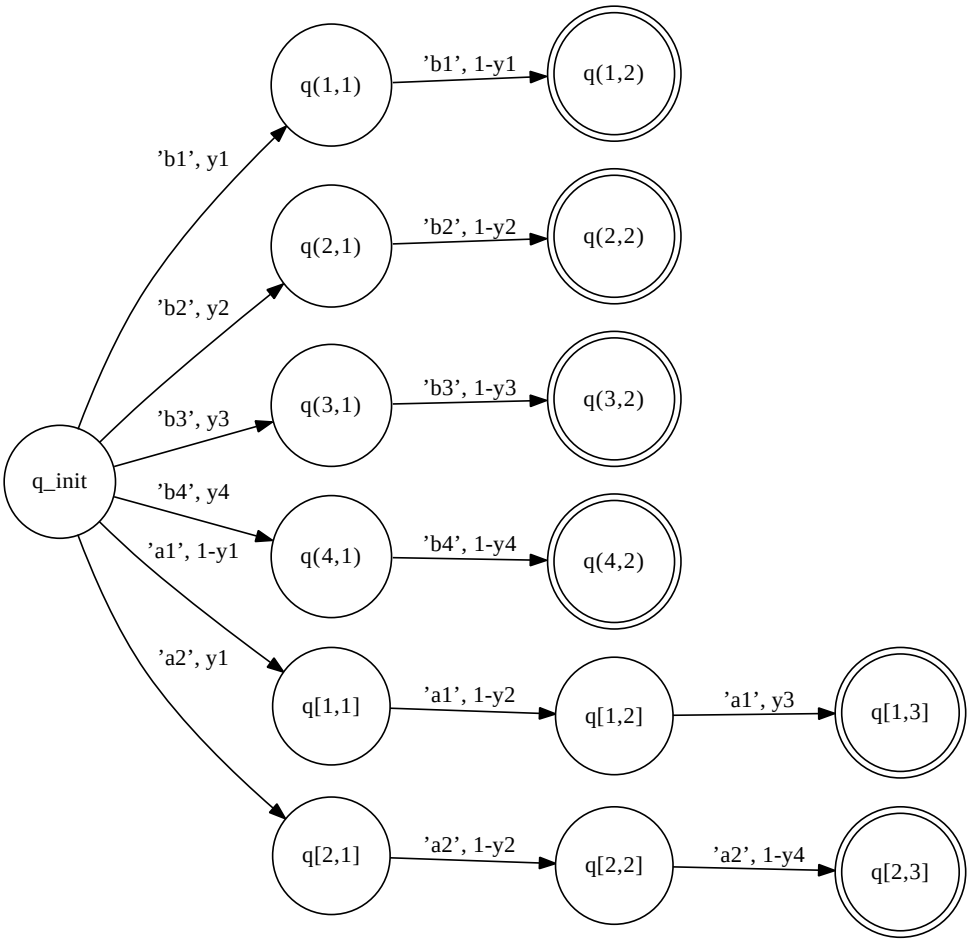


Figure 5.1: Example of encoding 3SAT formula $(y_1 \vee y_2 \vee \neg y_3) \wedge (\neg y_1 \vee y_2 \vee y_4)$ into automaton U_1

We can fix this problem by checking each term in a sum individually (only in conjunction with $F'(y)$), since the whole sum $Q(y)$ is zero if and only if each term is 0. □

Theorem 5.4. *The existential equivalence problem is at least as hard as the problem of comparing sum of square roots of integers with an integer.*

Definition 5.1. *The problem of comparing sum of square roots of integers with an integer is a decision problem such that given non-negative integers* $g_1, \dots g_k$ *and* g*, is* $\sqrt{g_1} + \ldots + \sqrt{g_k} \leq g$*?*

Classifying the complexity of this problem is a famous open problem in computational geometry. It arises when we compare the length of polygonal paths in Euclidean space. It is not known to be in NP (nor to be NP-Hard). According to [Tiw92] and [CMSC09] PSPACE is the smallest well studied complexity class that provably contains this problem.

Proof. To solve this problem by using existential equivalence of probabilistic automata we will use similar technique as in Theorem 5.1 to wire (encode) equalities into our automata.

First note that the following lines are equivalent:

$$\sqrt{g_1} + \ldots + \sqrt{g_k} \leq g \tag{5.1}$$

$$\sqrt{\frac{g_1}{g^2}} + \ldots + \sqrt{\frac{g_k}{g^2}} \leq 1 \tag{5.2}$$

$$\exists h \geq 0, \sqrt{\frac{g_1}{g^2}} + \ldots + \sqrt{\frac{g_k}{g^2}} + h = 1 \tag{5.3}$$

Let $q_1, \dots q_k \in \mathbb{Q}$ be such that $q_i = g_i/g^2$. With first k input symbols we define equalities $q_i'^2 = q_i$ and with another input symbol we define the equality $\sum_{i=0}^{k} q_i' + h = 1$. Now automata U_1 and U_2 are equivalent if and only if the equality 5.3 holds. □

Definition 5.2. *A multivariate* K*-polynomial is a multivariate polynomial such that if it has a real root then it has a real root in the multidimensional ball of radius* $\leq K$ *for* $K \in \mathbb{R}^+$*.*

Note that each multivariate K-polynomial is also a multivariate L-polynomial for all L greater than K. Moreover, for each multivariate polynomial there exists $K \in \mathbb{R}^+$ such that it is a multivariate K-polynomial.

Remark 5.1. We can change a multivariate L-polynomial $Q_L(y)$ to a multivariate K-polynomial $Q_K(y)$ by the following transformation (scaling): $Q_K(y) := Q_L(\frac{L}{K}y)$. The set of real roots of both polynomials is the same.

Lemma 5.5. *For each multivariate polynomial $Q(y)$ we can effectively find $K \in \mathbb{R}^+$ such that $Q(y)$ is a multivariate K-polynomial.*

Proof. To find such K we can use Cylindrical Algebraic Decomposition (CAD) algorithm described in the next chapter. We run CAD which gives us finite partition of the space and then K is the maximum of the distances of the outer most cells of the partition from the origin. □

Theorem 5.6. *The existential equivalence problem is at least as hard as deciding if a multivariate K-polynomial has a root for any fixed $K \in \mathbb{R}^+$.*

Proof. First we scale an input polynomial and then we construct a Probabilistic automata for which exist feasible values of parameters such that they are equivalent if and only if the input polynomial has a root.

Scaling

We can scale the input polynomial using the fact from Remark 5.1 to get a multivariate $\frac{1}{2}$-polynomial. So we can assume that the polynomial either does not have any real roots or it has a root $y = (y_1, \dots y_k)$, where $|y| \leq \frac{1}{2}$ (so then also $|y_i| \leq \frac{1}{2}$).

We assume that an input polynomial is given as a valid string which consists of characters $0, 1, \dots, 9$,), (, +, −, ∗, ^ and variables y_i. We limit only exponentiation in the way that nesting of exponents is forbidden and variables cannot be in the exponent.

Example 5.1. Examples of a bad input: $(y_1 + 4)^{y_2}$, $((2 * y_1)^3 + 1)^4$.
Example of a good input: $(y_1 + 4 + y_2)^{17} - (y_1 * y_1 + 5) * (y_1 - y_2)^6 + y_2^3$.

Now we perform another transformation which ensures that any valid part of an input polynomial $Q(y)$ is in absolute value at most $1/2$. Let L be an upper bounds for an absolute value of the polynomial. We can get it by changing all minus signs to plus signs and interchanging all variables with 1 (we have bounds on variables $1/2$ so we can do it) and all constants less than 1 also by 1. Then by evaluating the part of the input polynomial we get certainly the upper bound not only for it, but also for any sub part of it. We can prove it by induction since at each polynomial we can look as at the sum or product of two shorter polynomials, and by adding or multiplying two expressions greater than 1 the result of the operation is greater than both of them.

Now by dividing $Q(y)$ by $2L$ we get that the overall value of the polynomial is in the absolute value $\leq 1/2$. The question is how to incorporate this division by $2L$ into all parts of $Q(y)$ to get this property for any sub part. We can do it in a recursive way. Each valid part $Q'(y)$ of the input polynomial is also a polynomial and has one of the following forms:

- $Q'(y)$ is a constant or a variable
- $Q'(y)$ is a sum of two shorter polynomials: $Q_1(y) + Q_2(y)$
- $Q'(y)$ is a power of the shorter polynomial: $(Q_1(y))^k$
- $Q'(y)$ is a product of two shorter polynomials: $Q_1(y)Q_2(y)$

In the first two cases we just divide sub parts by $2L$, in the third case we can divide them by $\sqrt[k]{2L}$. In the last case the simplest way is to divide both of them by $2L$. However it causes few problems as we need to divide the whole polynomial by the same number to preserve the same roots. We can easily fix it during popping out of (on the way out of) recursion. In case of $Q'(y) = Q_1(y) + Q_2(y)$ we check by what we divided $Q_1(y)$ and by what $Q_2(y)$ and make it equal by the additional dividing of the one which was divided less so far.

Construction

We construct probabilistic automata U_1 and U_2 such that there exist feasible values of parameters $y = (y_1, \dots y_k)$ such that U_1 and U_2 are equivalent if and only if the input polynomial $Q(y)$ has a root such that $-1/2 \leq y_i \leq 1/2$ for all y_i (by scaling done above).

Example 5.2. We start with a simple example. Consider a polynomial $Q(y) = \underbrace{\overbrace{(y_1y_2 + 1/7)}^{a_3}\overbrace{(y_2 - 1/5)}^{a_4}}_{a_1} + \underbrace{(-4y_1)}_{a_2}$. We ask if it has a root. It is equivalent to asking if the following system of equations has a solution:

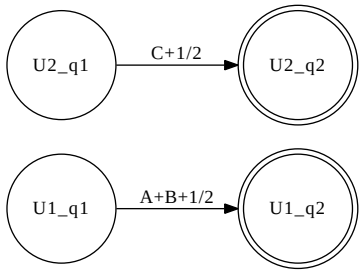
$$\begin{aligned} a_1 + a_2 &= 0 && (5.4)\\ a_2 &= -4y_1 && (5.5)\\ a_3a_4 &= a_1 && (5.6)\\ a_4 &= y_2 - 1/5 && (5.7)\\ a_5 + 1/7 &= a_3 && (5.8)\\ y_1y_2 &= a_5 && (5.9) \end{aligned}$$

Similarly as in the example above we can define the set of equalities for any input polynomial. It is because each polynomial can be written as a constant or a variable, a sum or a product of two shorter polynomials or power of shorter polynomial. First we ignore the power operation and will come back to it later. So basically (except exponentiation for now) we can write down a polynomial equation as a system of equalities of the form $A + B = C$ or $A * B = C$ by introducing new variables. Note that the input has length n and we need at most one new variable for each sum or product, so at most n new variables a_i.

To solve this problem by using existential equivalence of probabilistic automata we will once again use a similar technique as in Theorem 5.1 to encode equalities into our automata. From the scaling we have all variables y_i and also $a_i \in [-1/2, 1/2]$, so we must be careful not to limit them in any other way while encoding equalities to automata. It could be limited by

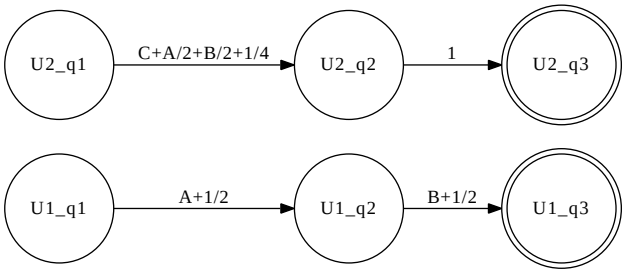
the fact that each transition probability in automaton is between 0 and 1, inclusive. Therefore straightforward encoding would not work.

However we encode the equality $A+B=C$ into automata in the following way:



For i-th such equality we have input symbol c_i and we accept the string c_i with transition probability $A+B+1/2$ in Automaton U_1 and with transition probability $C+1/2$ in Automaton U_2.

And we encode the equality $A*B=C$ into automata in the following way:



For i-th such equality we have input symbol d_i and we accept the string $d_i d_i$ with transition probability $(A+1/2)*(B+1/2)$ in Automaton U_1 and with transition probability $C+A/2+B/2+1/4$ in Automaton U_2. We have this equality $(A+1/2)*(B+1/2)=C+A/2+B/2+1/4$ if and only if $A*B=C$ for all $A,B,C\in[-1/2,1/2]$.

In the case of exponentiation A^k, we cannot just do it directly through writing it as a product $\overbrace{A*A\ldots*A}^{k \text{ times}}$ and then encoding it as above. It would make our construction exponential and we need to have it in a polynomial

time. However we can fix it by doing iterative squaring. We can do it by introducing new variables and using the fact that $A^k = A^{k/2} * A^{k/2}$ for k even and $A^k = A^{k/2} * A^{k/2} * A$ for k odd. For this we need $\mathrm{O}(\log k)$ new variables for any exponent k. For input of length n is $k < 9^n$ and therefore we need at most $\mathrm{O}(\log 9^n)$=$\mathrm{O}(n)$ new variables. We have less than n exponents in the input polynomial and so we need only polynomially many new variables and equations, as required. We encode these equations as in the case of multiplication $A * B = C$.

□

Remark 5.2. Consider famous Hilbert's tenth problem:

> "Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers"

It was finally proved in 1970 that this problem, which is equivalent to asking if a polynomial with integer coefficients has an integer root, is not decidable. The result for a similar problem asking if such polynomial has a rational root is, despite the great interest, still an open problem. Taking into account Lemma 5.5 and encoding of the polynomial into automata using the technique in Theorem 5.6 we can conclude that if we were able to decide the problem of Existential equivalence of probabilistic automata with parameters over rationals (i.e. if exist parameters $y_i \in \mathbb{Q}$ such that two automata are equivalent) we would be able to decide if the polynomial (with integer coefficients) has a rational root.

Chapter 6

First-order logic equivalence of probabilistic automata with parameters

First-order logic equivalence of probabilistic automata with parameters U_1 and U_2 is a decision problem (considering only feasible values of parameters $y = (y_1, y_2, \ldots, y_k)$) asking if the given first order logic statement about equivalence of two probabilistic automata is true.

Definition 6.1. *Consider atomic formulas:*

- $L_{U_1(y)} = L_{U_2(y)}$
- *equalities or inequalities of algebraic expressions which contain variables y_i, numbers, brackets, and operations $+$, $-$, $*$, / and $\hat{}$*

First-order logic equivalence of probabilistic automata with parameters U_1 and U_2 is a decision problem asking if the logical sentence, which is a combination of atomic formulas with boolean connectives and quantifiers, is true.

Example 6.1. The example of the problem is to decide the truth of:

$$(\exists y_1)(\forall y_2)((3y_1 \geq 2y_2) \wedge (L_{U_1(y)} = L_{U_2(y)}))$$

Lemma 6.1. *Any first-order logic statement about an equivalence of polynomials is decidable.*

It follows from the decidability of the first order theory of the reals. Currently there are many different known algorithms. Some of them have better theoretical complexity, other have better performance in practice (See [HL91] for comparison). The complexity class of the algorithms is Double Exponential. We describe in short an idea of Collins algorithm which is the most widely used, mainly because of the practical performance.

6.1 Collins algorithm overview

We give an overview of the algorithm based on the overview in [HL91]. A more detailed description of the algorithm is available in [Col75] and [BPR06].

The algorithm constructs a Cylindrical Algebraic Decomposition (CAD) of the input polynomials, where a CAD of a set of polynomials in n variables is a certain finite partitioning of the n-dimensional real space such that in each cell of the partition polynomials have constant signs. Once a CAD is constructed, the truth of the input sentence can be determined by checking its truth value in appropriate cells by examining sample points within these cells.

A CAD is constructed in three stages:

Projection: The input n-variable polynomials are projected into a set of $(n-1)$ variate polynomials in a way that a CAD of n-space can be built on a CAD of $(n-1)$-space. This continues until univariate projection polynomials are obtained.

Base stage: A CAD of 1-space is constructed by isolating real roots of the univariate projection polynomials.

Extension: A CAD of 2-space is constructed by building a stack of cells over the cells of CAD of 1-space. It continues until a CAD of n-space is obtained.

6.2 First-order logic equivalence is PSPACE-Hard

Theorem 6.2. *First-order logic equivalence of probabilistic automata with parameters is NP-Hard.*

Proof. It follows from Theorem 5.1, from NP-Hardness of existential equivalence of probabilistic automata with parameters. □

Theorem 6.3. *First-order logic equivalence of probabilistic automata with parameters is decidable and is in the complexity class Double Exponential.*

Proof. We reduce the problem to a problem in Lemma 6.1, which is well known to be in Double Exponential. First we construct the polynomial $Q(y)$ as in Theorem 4.1 and then we add to our first-order logic prefix all inequalities for parameters, which are forced by automata (i.e. implied from the fact that transition probabilities are between 0 and 1, inclusive, and sum to 1 for each state and input symbol). □

Theorem 6.4. *First-order logic equivalence of probabilistic automata with parameters is PSPACE-Hard.*

Definition 6.2. *Quantified SAT, in short QSAT, is a decision problem if a given fully quantified (with no free variables) boolean formula evaluates to true.*

Example of QSAT is: $\forall x \exists y \forall z (x \vee y) \wedge z$.

Proof. The QSAT problem is well known to be PSPACE-Complete. We show reduction of this problem to First-order logic equivalence of probabilistic automata with parameters.

It is sufficient to take an input QSAT formula and add constraint for each parameter y_i corresponding to a literal: $(y_i = 0 \vee y_i = 1)$ to force them to be either 0 or 1 and then to ask the question if Probabilistic automaton U_1 is equivalent to itself. Technically, we cannot take QSAT directly as operations $\vee$ and $\wedge$ are not well defined over the reals. So we can transcribe the formula using the following two rules, where t stands for transcribing operation :

- $t(y_i)$ goes to y_i for any atomic literal (parameter) y_i
- $t(\phi(y) \wedge \psi(y))$ goes to $t(\phi(y))t(\psi(y)) = 1$
- $t(\phi(y) \vee \psi(y))$ goes to $1 - (1 - t(\phi(y)))(1 - t(\psi(y))) = 1$

First-order logic equivalence of probabilistic automata with parameters will answer yes if and only if the first-order logic prefix is true, i.e. when an input instance of QSAT is true. □

Chapter 7

Bisimulation of Probabilistic Automata

The notion of bisimulation for probabilistic transition systems was introduced by Larsen and Skou [LS91]. It is a probabilistic version of popular notion of bisimulation on labelled transition systems. It is mainly used for state minimization as well as equivalence checking.

A bisimulation is an equivalence relation on two state transition systems, where one system simulates (matches the moves of) another and the other way round.

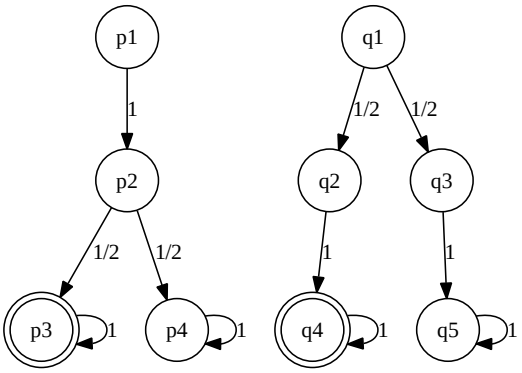


Figure 7.1: Example of automata that are language equivalent, but not bisimilar

Bisimulation is stronger than language equivalence, so it can happen that

two systems are language equivalent, but not bisimilar. On the other hand two bisimilar systems are always also language equivalent.

First we consider as the transition systems probabilistic automata without parameters. Consider two probabilistic automata $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1)$ and $U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$. Recall that (see Definition 2.10) we can combine them to the one automaton $U_1 \oplus U_2$. Now we define bisimulation on this combined automaton.

Let S be set of states of size n and $\sim$ be an equivalence relation on S. We denote by $s \sim t$ that $(s,t) \in \sim$, by $S/\sim$ the set of equivalence classes with respect to $\sim$ and by ρ_s the distribution over states such that it is 1 for the state s and 0 for the other states. Moreover recall that if we have $B \subseteq S$ then we define η_B to be an n dimensional row vector such that $\eta_B[i] = 1$ iff $s_i \in B$ and 0 otherwise.

In words, $s \sim t$ if the sum of transition probabilities from the state s after reading the input symbol δ to the part of partition B is the same as from the state t for all possible input symbols δ and all elements B of partition.

Definition 7.1. *Let $U_1 \oplus U_2$ be a combination of two probabilistic automata. A bisimulation on $S = S_1 \cup S_2$ is an equivalence relation $\sim$ on S such that if $s \sim t$ then $\forall \delta \in \Sigma$, $B \in S/\sim$ we have $\rho_s M_{U_1 \oplus U_2}(\delta)\eta_B^T = \rho_t M_{U_1 \oplus U_2}(\delta)\eta_B^T$ and both $s,t \in F = F_1 \cup F_2$ or both $s,t \in S \setminus F$.*

Definition 7.2. *We say that probabilistic automata U_1 and U_2 are bisimilar if there exists a bisimulation $\sim$ such that $[\rho_1, -\rho_2]\eta_B^T = 0$ for each equivalence class $B \in S/\sim$.*

Note 7.1. In Definition 7.2 we could exchange maximum bisimulation by if there exists bisimulation. It is the same, because bisimulations are closed under union.

Example 7.1. See Figure 7.1 as the example of two automata that are language equivalent, but not bisimilar. In that example, Σ has only one symbol and the whole initial distribution is concentrated in the states p_1 and q_1, respectively. If the states p_4 and q_5 became accepting then the two automata would be bisimilar.

7.1 Partition refinement algorithm for testing bisimulation

One way to compute the maximum bisimulation on an automaton is by *partition refinement*. The idea is to start with trivial partition and then to refine this partition step by step until a bisimulation is reached. A similar idea was introduced by Hopcroft [Hop71] for minimizing deterministic finite automata.

This approach was extended to probabilistic transition systems in [Bai96], where the algorithm running in time $O(n^2m)$ was introduced (n is number of states and m is number of transitions).

There is an even faster $O(m \log n)$ algorithm [DHS$^+$03] for probabilistic bisimulation (first especially designed for Markov Chains). The algorithm gains speedup by using splay trees to sort transition weights. Also note that probabilistic bisimulation applied to Markov Chains is the same concept as *lumpability*.

In this section we provide the description of the simpler version of the $O(n^2m)$ algorithm accommodated to the probabilistic automata defined in this thesis.

Algorithm

We start with the trivial partition $X_0 = \{S \setminus F, F\}$. In each step we refine the elements of the partition obtained so far.

Let $E_s(B, \delta) = \rho_s M_{U_1 \oplus U_2}(\delta)\eta_B$, i.e. E_s is a matrix of transition probabilities from the state s, where there is one entry for each combination of the input symbol $\delta \in \Sigma$ and element of partition $B \in X_i$ (where X_i is the current partition). Then we split each element of partition $B \in X_i$ in such a way that states $s, t \in B$ stay together if and only if $E_s(C, \delta) = E_t(C, \delta)$ for all blocks $C \in S/X_i$. We say that we obtain the partition X_{i+1} as the union of refinements of elements of partition $B \in X_i$ with respect to the relation

corresponding to X_i. I.e.

$$X_{i+1} = \bigcup_{B \in X_i} B/ \equiv_{X_i}$$

Formally, we define $s \equiv_{X_i} t$ iff $E_s(C, \delta) = E_t(C, \delta)$ for all $C \in S/X_i$ and $\delta \in \Sigma$.

Since each refinement increases the number of blocks in the partition, after at most $n - 1$ steps the partition cannot be refined any more so we are finished. The corresponding equivalence relation to the partition X_k is the maximum bisimulation $\sim$. It can be shown by induction on i that any bisimulation on S refines X_i for all i.

Algorithm 7.1.1 Algorithm for testing bisimulation of probabilistic automata

Require: $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1), U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$

Set X, PX to be the empty partitions
$X \leftarrow \{\{s_i \in S \setminus F\}, \{s_i \in F\}\}$
while $X \neq PX$ **do**
    $PX \leftarrow X$
    $X \leftarrow REFINE(X)$
end while
if $\forall \delta \in \Sigma, B \in X$, $[\rho_1, -\rho_2]\eta_B^T = 0$ **then**
    return 'yes'
else
    return 'no'
end if

Two states are bisimilar if they are in the same part of partition and we can decide if two probabilistic automata are bisimilar by verifying that $\forall \delta \in \Sigma, B \in S/\sim$ we have $[\rho_1, -\rho_2]\eta_B^T = 0$.

7.2 Universal parametric case

Universal bisimulation of probabilistic automata with parameters U_1 and U_2 is a decision problem if for all feasible values of parameters $y = (y_1, y_2, \ldots, y_k)$ by plugging them into automata we get bisimilar probabilistic automata.

Recall that introducing parameters brings naturally also all the constraints for them. Let $Ay \leq b$ be the set of constraints from the automata U_1 and U_2.

Algorithm

We would like to run the non-parametric algorithm (see Algorithm 7.1.1) with minor changes. In the non-parametric case, during partition refinement, we are comparing matrices which contain only numbers. Now if we try to run the same algorithm we need to compare linear expressions containing parameters to determine if we need to split any block of the partition.

Fix a block $B \in S/\sim$. For each pair of states $s, t \in B$, each block $C \in S/\sim$ and symbol $\delta \in \Sigma$ we have an equality $E_s(C, \delta) = E_t(C, \delta)$ on the set of parameters y. Corresponding entries of these matrices are equal if the probability to transition from the state s to block C after reading δ is the same as the probability to transition from the state t to block C after reading δ.

Consider such linear equality of corresponding entries in $E_s(C, \delta)$ and $E_t(C, \delta)$. In general we can write it in the form $cy^T = d$, where c is a row vector and d is a constant. There are two possibilities.

1. Either the equality $cy^T = d$ is already implied by the constraints $Ay \leq b$. In that case we can say that it always holds and we keep s, t in the same block.

2. Or the equality is not enforced by the constraints $Ay \leq b$. In that case there are values of parameters y such that two states are not bisimilar. So we can say that two given states will not be together. Do we have to add the constraint $cy^T \neq d$ to the set of constraints $Ay \leq b$? Fortunately not; and this keeps our algorithm simple. It is because the finite number of linear "inequalities" (meant $\neq$ only) with the system $Ay \leq b$ does not help to force any new equality which would not be forced by $Ay \leq b$ alone.

To finalize our algorithm, we need to be able to check if the equality $cy^T = d$ is forced by $Ay \leq b$ or not. We need to do it in polynomial time to

get the overall polynomial algorithm. We will use Linear Programming to do it. First we minimize cy^T given that $Ay \leq b$ and then maximize cy^T given that $Ay \leq b$. If in both cases we get d then the equality is forced, otherwise not.

7.3 Existential parametric case

Existential bisimulation of probabilistic automata with parameters U_1 and U_2 is a decision problem if there exist feasible (satisfying $Ay \leq b$) values of parameters $y = (y_1, y_2, \ldots, y_k)$ such that by plugging them into automata we get bisimilar probabilistic automata.

In this section we show two NP algorithms for this problem followed by the proof that this problem is $NP - Hard$ so we cannot hope for any polynomial time algorithm. However, if we fix the number of parameters then the second algorithm runs in polynomial time.

Trivial Algorithm

We can try all possible partitions of the set S.

Given a partition X we need to check if the relation induced by it is a bisimulation and if the condition $\forall \delta \in \Sigma, B \in X$, $[\rho_1, -\rho_2]\eta_B^T = 0$ holds. We get bunch of new linear equalities over parameters y. For each pair of states s, t in the same block $B \in X$, for each $\delta \in \Sigma$ and for each block $C \in X$ we get equations $E_s(C, \delta) = E_t(C, \delta)$.

When we combine them with the initial constraints on parameters $Ay \leq b$ and the whole system has a feasible solution then there exist parameters such that the automata are bisimilar. We can verify if the whole system has a feasible solution by Linear Programming.

Note that this algorithm is NP even if the number of parameters k is fixed since there are exponentially many partitions (in the number of states).

Faster Algorithm

Once again we would like to run the non-parametric algorithm (see Algorithm 7.1.1), but for all possible value of parameters. If one of the runs is successful we can conclude that there exist feasible values of parameters such that automata are bisimilar.

Algorithm 7.3.1 Algorithm $EXIST\_BISIMILAR(k, C, X)$, where k is number of parameters, C are constraints, X is a partition of set S

Require: $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1), U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$
Require: $C = \{Ay \leq b\}$ are constraints on parameters y from U_1, U_2
Require: $BISIMILAR$ is non-parametric algorithm 7.1.1
Require: $CHECK\_PARTITION$ determines if two automata are bisimilar for a given partition, using the method described in Trivial algorithm
Require: $COMBINE$ substitute the equation to constraints
Require: $REFINE\_NE$ does one step of partition refinement with assumption that none of non-trivial possible equalities from partition X holds

1: **if** $k = 0$ **then**
2: **return** $BISIMILAR(C)$
3: **end if**
4: **for all** non-trivial possible equalities $cy^T = d$ from partition X **do**
5: $CC \leftarrow COMBINE(C, cy^T = d)$
6: **if** $EXIST\_BISIMILAR(k-1, CC, X)$ **then**
7: **return** 'yes'
8: **end if**
9: **end for**
10: $NX \leftarrow REFINE\_NE(X)$
11: **if** $X = NX$ **then**
12: **if** $CHECK\_PARTITION(X)$ **then**
13: **return** 'yes'
14: **end if**
15: **else**
16: **if** $EXIST\_BISIMILAR(k, C, NX)$ **then**
17: **return** 'yes'
18: **end if**
19: **end if**
20: **return** 'no'

Of course there are infinitely (usually even uncountably) many possible values of parameters, so we cannot examine all of them. Fortunately for large

sets of them the algorithm behaves identically. In the non-parametric case, during partition refinement, we are comparing matrices which contain only numbers. Now if we try to run the same algorithm we need to compare linear expressions containing parameters to determine if the states stay together or have to be separated. So for each non-trivial (non trivial is if c is not a zero vector) linear equation $cy^T = d$ which we need to test, we have two options. Either it holds or not.

If it holds we can get rid of one parameter, substitute it everywhere to both automata (so also to all constraints) and call recursively the algorithm with the input that contains one parameter less. Finally for zero parameters we can run the non-parametric algorithm.

If none of the equations holds then we can do the refinement step in our algorithm with the knowledge that none of those non-trivial equations holds. As in non-parametric algorithm it makes sense to repeat this process only if after refinement step we do not get the same partition again (so at most $n-1$ times). Then we can check it for this partition as in Trivial algorithm (by Linear Programming).

This algorithm is clearly exponential. Now consider the case that the number of parameters k is fixed. Let d be the size of Σ. Then non-parametric algorithm runs in time $\mathrm{O}(ndn^3)$. Linear programming runs in time $\mathrm{O}(k^{3.5})$, but as we can have many input equations and k is constant, it is better to bound it by $\mathrm{O}(n)$. If $\mathrm{T}(n)$ is the time complexity of the algorithm then we have $\mathrm{T}(0)=\mathrm{O}(ndn^3)$ and $\mathrm{T}(n)\leq\ (n-1)dn^3\ \mathrm{T}(n-1)+\mathrm{O}(n)$. So we have $\mathrm{T}(n)=\mathrm{O}((dn^4)^{k+1})$ and therefore for the fixed number of parameters we have the polynomial time algorithm. In practice the performance of this algorithm could be improved a lot by pruning inconsistent cases sooner. E.g. equations can be not consistent with the constraint so far so it does not make sense to do the computation which assumes they hold.

NP-Completeness

Theorem 7.1. *The problem to decide if there exist feasible values of parameters $y = (y_1, y_2, \ldots, y_k)$ such that by plugging them into automata U_1 and*

U_2 we get bisimilar probabilistic automata is $NP - Hard$.

Proof. We do a reduction from $3SAT$. We construct two probabilistic automata in such a way that there exist feasible parameters y such that they are bisimilar if and only if the corresponding $3SAT$ formula can be satisfied.

So suppose we have $3SAT$ formula φ with k variables and n clauses. For each proposition variable y_i in φ we have the corresponding parameter y_i in our automata. The construction has two main parts. First we force the parameters to be only 0 or 1, i.e. for no other values would the automata be bisimilar. Then we force each clause to be satisfied. So then there exist parameters y such that automata are bisimilar if and only if the input $3SAT$ formula can be satisfied.

For each parameter we have a corresponding input symbol δ_i. Figure 7.2 illustrates the gadget we use to force each parameter to be 0 or 1.

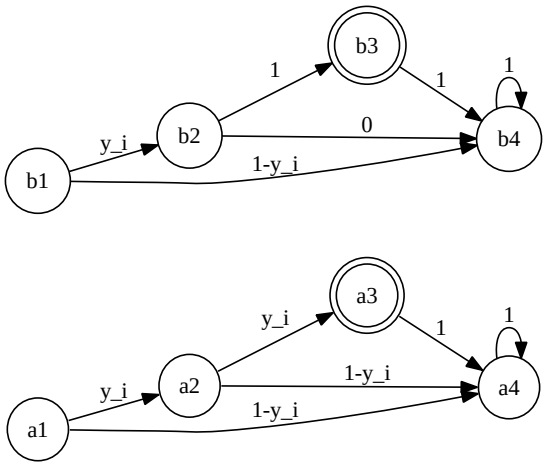


Figure 7.2: To keep automata bisimilar we force y_i to be 0 or 1

We construct automata such that among the states in Figure 7.2, only a_1 and b_1 has non-zero probability p in the initial distribution. Note that automaton U_1 accepts string $\delta_i\delta_i$ with the probability py_i^2 and automaton U_2 with the probability py_i. Recall that if automata are not equivalent then they cannot be bisimilar. Therefore the only chance for them to be bisimilar is if $y_i^2 = y_i$. It is true only for $y_i \in \{0, 1\}$. We can check that these parts are bisimilar easily. For $y_i = 1$ it is clear. If $y_i = 0$, running the algorithm 7.1.1

we get a partition $\{a_1, a_2, a_4, b_1, b_4\}, \{b_2\}, \{a_3, b_3\}$ for which it can be easily verified that automata are bisimilar.

Now we just need to encode constraints that will force each clause to be satisfied. If the sum of the parameters corresponding to literals in the clause (by which we mean that for variable y_i from $3SAT$ we have parameter y_i in automata or $1 - y_i$ if it is negated in the clause) is at least 1 then the clause is satisfied.

Since the formula is in 3-CNF, the maximum sum is 3, but since the transition probabilities are limited to be between 0 and 1, inclusive, we would sum thirds of values of parameters and check if it is at least 1/3.

So for the i^{th} clause of the form $(y_j \vee y_k \vee y_l)$ we make transition probability from the new state s_i to the state e_i with probability $(y_j + y_k + y_l)/3 + u_i$, where $u_i \leq 2/3$. We can do the same in both automata (so we do not break bisimulation, but only add new constraints) and since we do not define any other possible transitions from s_i, we must have $(y_j + y_k + y_l)/3 + u_i = 1$. Now we add the constraint $u_i \leq 2/3$ and we are done. We add this constraint, by setting transition probabilities from the new state u_i to the states u_i' and u_i'' to be $u_i + 1/3$ and $2/3 - u_i$ respectively. Once again we add it to both automata to not to break bisimulation.

If the literal y_k is negated in the clause, we use $1 - y_k$ instead. So for example for the clause $(y_j \vee y_k \vee \neg y_l)$ the transition probability is $(y_j + y_k + (1 - y_l))/3 + u$.

□

Corollary 7.2. *The problem to decide if there exist feasible values of parameters $y = (y_1, y_2, \ldots, y_k)$ such that by plugging them into automata U_1 and U_2 we get bisimilar probabilistic automata is $NP - Complete$.*

Proof. It follows from Theorem 7.1 and NP algorithms solving this problem described above. □

Chapter 8

Summary and future work

We conclude with a summary of the results and discussion of possible future work. An overview of the results is in Table 8.1. The underlined results are our new results first presented here.

In the language equivalence problem of probabilistic automata our main contributions consist of showing PTIME-Completeness of the equivalence problem for non-parametric automata and design of 1MC randomized algorithm for deciding universal equivalence for parametric automata. This algorithm is very fast with the error probability going to zero for n (the number of states of automata) going to infinity so it can be very useful for the practical applications.

For the existential equivalence of probabilistic automata we showed not only that the problem is NP-Hard, but also that it is as hard as the problem of whether the sum of square roots of a given family of integers is less than an integer, or the problem of whether the multivariate K-polynomial has a root. For both these problems there are no known better than PSPACE algorithms. Therefore we expect the existential equivalence problem to be classified somewhere between NP-Hard and PSPACE-Hard or even as PSPACE-Complete problem. This opens the space for future work. One possible way to proceed is if we were able to find more effectively K such that an input multivariate polynomial has a real root if and only if it has a real root less than K (the distance from the origin was less than K). Then we would be able to show

that the existential equivalence is as hard as the existential theory of the reals.

Problem	**Best known algorithm**	**Hardness of the problem**
Non-parametric Equivalence	$O(n^3)$	PTIME-Complete
Universal Equivalence	1MC randomized $O(n^3 + k^{3.5})$	PTIME-Hard
Existential Equivalence over $\mathbb{R}$	PSPACE	NP-Hard
Existential Equivalence over $\mathbb{Q}$	not known to be decidable	extended Hilbert tenth problem -Hard
FOL Equivalence	Double Exponential	PSPACE-Hard
Non-parametric Bisimulation	$O(dn^4)$	
Universal Bisimulation	$O(dn^4k^{3.5})$	
Existential Bisimulation	$O((dn^4)^{k+1})$	NP-Complete

Table 8.1: Summary of the results

For the first-order logic equivalence we proved that it is a PSPACE-Hard problem, so we cannot even hope for faster than PSPACE algorithm. In spite of the hardness of both the existential and the first-order logic equivalence, the positive thing is that both are solvable. Moreover recently there has been a lot of effort to make the implementations of algorithms for the existential (respectively first-order) theory of the reals as fast as possible. So in practice and for smaller automata it could be useful.

Concerning bisimulation of probabilistic automata, our main contributions are in creating a polynomial time algorithm for deciding universal bisimulation of parametric automata and an NP algorithm for deciding existential

bisimulation. However, if we fix the number of parameters then existential bisimulation algorithm runs in polynomial time. Our algorithms are based on the non-parametric bisimulation algorithm which is not asymptotically optimal. There exist faster algorithms for that problem mentioned in Chapter 7 which are based on clever use of splay trees. There is possible future work in adapting those ideas also to algorithms for universal and existential parametric bisimulation. Moreover especially in the existential bisimulation, we can hope for improvements in faster implementation by pruning the cases that do not lead to the solution. However, there is some trade off as we need to run the linear programming to decide if we can prune.

Appendix A

Elimination of non-free variables from the system of linear inequalities

Our goal is to find dependencies across variables from the system of linear inequalities. We will consider only non-strict inequalities (in standard form, i.e. described in the form something ≤ 0). Moreover we assume that there exists a solution to the system (it can be checked by Linear Programming).

Example A.1.

$$x + y + 4 \leq 0 \tag{A.1}$$
$$-x - y - 4 \leq 0 \tag{A.2}$$
$$y - 1 \leq 0 \tag{A.3}$$

We can immediately see from the first two inequalities that $x + y + 4 = 0$ and so we have dependency and we can eliminate one of the variables. For example, after eliminating y (by using the fact that $y = -4 - x$), we get the

following set of linear inequalities:

$$0 \leq 0 \tag{A.4}$$

$$0 \leq 0 \tag{A.5}$$

$$-x - 5 \leq 0 \tag{A.6}$$

If we had additional inequality $x + 5 \leq 0$ at the beginning we would be able to find also the dependency $x = -5$.

Remark A.1. By eliminating non-free variables we get the dimension of the polytype specified by the set of linear inequalities.

We try to find one dependency at a time. If we find one then we plug this dependency into the system, as in the example above, and start solving the problem again for less variables. If there are no more dependencies we terminate.

Suppose that we have the system of linear inequalities with k variables $y_1, \ldots y_k$. Let there be a dependency for variable y_1 (without loss of generality). Then there must $\exists d_2, \ldots d_k, d_{k+1} \in \mathbb{R}$ such that $y_1 + d_2y_2 + d_3y_3 + \ldots + d_ky_k + d_{k+1} = 0$, or equivalently we must have as a consequence of our system of linear inequalities the following two inequalities:

$$y_1 + d_2y_2 + d_3y_3 + \ldots + d_ky_k + d_{k+1} \leq 0$$

$$-(y_1 + d_2y_2 + d_3y_3 + \ldots + d_ky_k + d_{k+1}) \leq 0$$

This can be achieved only if there are two disjoint subsets of linear inequalities such that there exists a positive linear combination (it preserves inequality) of inequalities from one subset which is equal to negation of a positive linear combination of inequalities from the second subset. In our example it was $1(x + y + 4) \leq 0$ and $-(1(-x - y - 4)) \leq 0$.

We can check this property easily by considering all possible pairs of disjoint subsets of the whole system. If we fix these two subsets then we get the system of linear equalities since the constant in front of each variable must be the same in both positive linear combinations from the subsets. We get

additional conditions (inequalities) from the fact that all linear combinations must be positive. We can once again use Linear programming to check if there is a feasible solution to this newly created system.

Consider one more time our example. Let the first subset be just equation $x + y + 4 \leq 0$ and the second subset be just equation $-x - y - 4 \leq 0$. Then we ask if $\exists c_1, c_2 \in \mathbb{R}^+$ such that

$$c_1(x + y + 4) = -c_2(-x - y - 4)$$

So we get equations $c_1 x = c_2 x$, $c_1 y = c_2 y$ and $4c_1 = 4c_2$ which are equivalent to $c_1 = c_2$. We can see easily that for example $c_1 = c_2 = 1$ is the solution and so we found the dependence.

List of Figures

Bibliography

[Bai96] Christel Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In *CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification*, pages 50–61, London, UK, 1996. Springer-Verlag.

[BPR06] Saugata Basu, Richard Pollack, and Marie-Franoise Roy. *Algorithms in Real Algebraic Geometry*. Springer, 2nd edition, 2006.

[CH92] Sang Cho and Dung T. Huynh. The parallel complexity of finite-state automata problems. *Inf. Comput.*, 97(1):1–22, 1992.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill Book Company, The MIT Press, 2001.

[CMSC09] Qi Cheng, Xianmeng Meng, Celi Sun, and Jiazhe Chen. Bounding the sum of square roots via lattice reduction, 2009.

[Col75] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20-23, 1975*, 1975.

[DHS+03] Salem Derisavi, Holger Hermanns, William H. Sanders, William H. S, and Ers A. Optimal state-space lumping in markov chains, 2003.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[GR88] Alan Gibbons and Wojciech Rytter. *Efficient parallel algorithms*. Cambridge University Press, Cambridge, UK, 1988.

[HL91] Hoon Hong and Collins L. Comparison of several decision algorithms for the existential theory of the reals. Technical report, 1991.

[Hop71] J. E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. technical report cs-71-190. *Stanford University*, 1971.

[Jon75] N. D. Jones. Space bounded reducibility among combinatorial problems. *J. Computer and System Sci.*, pages 68–75, 1975.

[JR93] Tao Jiang and B. Ravikumar. Minimal nfa problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[LS91] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[Paz71] Azaria Paz. *Introduction to Probabilistic Automata*. Academic Press, New York, USA, 1971.

[Rab63] M. O Rabin. Probabilistic automata. *Information and Control 6*, pages 230–245, 1963.

[RS59] M. O Rabin and D Scott. Finite automata and their decision problems. *IBM journal of research and development*, 1959.

[RS97] Grzegorz Rozenberg and Arto Salomaa. *Handbook of Formal Languages*. Springer, 1997.

[SJ01] Zdenek Sawa and Petr Jancar. P-hardness of equivalence testing on finite-state processes. *Lecture Notes in Computer Science, SOFSEM 2001: Theory and Practice of Informatics*, Volume 2234/2001:326–335, 2001.

[Tiw92] Prasoon Tiwari. A problem that is easier to solve on the unit-cost algebraic ram. *J. Complex.*, 8(4):393–397, 1992.

[Tze92] Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. COMPUT.*, 21(2):216–227, 1992.

[Wor08] James Worrell. Model checking and decision procedures for probabilistic automata and markov chains. In *QEST*, 2008.