Probabilistic Finite Automaton Emptiness is undecidable, or: Who is afraid of small probabilities?

Günter Rote

February 11, 2022

Abstract

It is undecidable whether the language recognized by a probabilistic finite automaton is empty.

This lecture note gives a self-contained proof of this important result, on which many other undecidability results are based.

1 Probabilistic finite automata (PFA)

We give a formal definition of a probabilistic finite automaton (PFA) later, in Section 7. Informally, we can think of a PFA in terms of an algorithm that reads an input string from left to right, having only finite memory. That is, it can manipulate a finite number of variables with bounded range, just like an ordinary finite automaton. In addition, a probabilistic finite automaton may make coin flips. The question whether the PFA arrives in an accepting state and thus accepts a given input word is not a yes/no decision, but it happens with a certain probability. The language *recognized* by a PFA is defined by specifying a probability threshold or *cut point* λ. The language consists of all words for which the probability of acceptance exceeds λ.

The *PFA Emptiness Problem* is the problem of deciding whether this language is empty.

2 History

The study of probabilistic finite automata was initiated by Michael Rabin in 1963 [10]. The first proof that PFA Emptiness is undecidable is due to Masakazu Nasu and Namio Honda from 1969 [8, Theorem 3]. It proceeds through a cascade of lemmas that involve tricky constructions, showing that more and more classes of languages, including certain types of context-free languages, can be recognized by a PFA, and it uses strong results from a previous paper of the authors. Eventually, the undecidability of the PFA Emptiness Problem is derived from Post's Correspondence Problem (PCP). The proof is reproduced in the final part of a monograph by Azaria Paz from 1971 [9, Theorem IIIB.6.17] (but without naming the PCP). This proof is often erroneously attributed to Paz, although Paz gives appropriate credit to Nasu and Honda [9, Section IIIB.7].

Another proof was sketched by Anne Condon and Richard Lipton in 1989 [3]. It arose as a side result of their investigation of space-bounded interactive proofs. Condon and Lipton based their reduction on the undecidability of the Halting Problem for Two-Counter Machines (2CM), see Section 3 below. The main ideas of this proof go back to Rūsiņš Freivalds [4], who studied the emptiness problem for probabilistic *2-way* finite automata in 1981 (unaware of Nasu and Honda's earlier work). In particular, Freivalds developed the idea of a competition between two players to recognize the language $\{\, \mathtt{a}^i\mathtt{b}^i \mid i \geq 0 \,\}$ (see Section 4 below), and aggregating the results of these competitions into "macrocompetitions" (see Section 5). A 2-way automaton can move the input head back and forth over the input, and thus process the input as often as it likes. Freivalds claimed that the emptiness problem for such automata is undecidable [4, Theorem 4]; he only mentions that the reduction should be from the PCP but gives no details how to connect "macrocompetitions" with the PCP. I have not been able to come up with an idea how the proof would proceed. For our case of a 1-way finite automaton, the repeated scan of the input is not possible; it is replaced by providing an input which consists of many repetitions of the same string.

In 2000, Blondel and Tsitsiklis [1] complained that "a complete proof that PFA Emptiness is undecidable cannot be found in its entirety in the published literature". Since then, Condon and Lipton's proof has been published in sufficient detail in other papers, for example by Madani, Hanks, and Condon [6, Sec. 3.1 and Appendix A] in 2003. Moreover, in the publication list on Anne Condon's homepage, the entry for the Condon–Lipton paper [3] from 1989 links to a 22-page manuscript, dated November 29, 2005[1]. According to the metadata, the file was generated on that date by the dvips program from a file called "journalsub.dvi". This manuscript also gives the proof in great detail.

Condon and Lipton's proof, which is based on Freivalds' ideas, is conceptually simple and illuminating, and it can be presented in a self-contained way.

The two proof have different merits: Condon and Lipton's proof leads to an arbitrarily large gap between accepting and rejecting probabilities, and it allows to bound the size of the input alphabet to 2. Nasu and Honda's proof allows to bound the number of states of the PFA by 12. Moreover, one can even show undecidability of the emptyness problem for a fixed PFA with 12 states and an input alphabet of size 53, where the only variable is the starting distribution.

3 2-Counter machines

A counter machine has a finite control, represented by state q from a finite set Q, and a number of nonnegative counters. Two states are designated as the start state and the halting state. Such a machine operates as follows. At each step, it checks which counters are zero, and depending on this and the current state q, it can increment or decrement its counters, and it enters a new state.

A counter machine with as few as two counters (a 2CM) is as powerful as a Turing machine. This was first proved by Marvin Minsky [7] in 1961 and is by now textbook knowledge [5, Theorem 7.9].[2] The question whether such a two-counter machine halts if it is started with both counter values at 0 is undecidable.

Denoting by q_i, l_i, r_i the state and the values of the counters after i steps, an accepting computation of length n can be written as follows:

$$l_0, r_0, q_0, l_1, r_1, q_2, l_2, r_2, q_3, \ldots, l_{n-1}, r_{n-1}, q_n$$

[1] `https://www.cs.ubc.ca/~condon/papers/condon-lipton89.pdf` accessed 2022-02-05.

[2] The usual way to simulate a Turing machine by a 2CM proceeds in several easy steps: (i) A two-sided infinite tape can be simulated by two push-down stacks. (ii) A push-down stack can be simulated by two counters, interpreting the stack contents as digits in an appropriate radix; two counters are necessary to perform multiplication and division by the radix. (iii) Any number of counters can be simulated by two counters, representing the values $a, b, c, d, \ldots$ of the counters as a product $2^a 3^b 5^c 7^d \ldots$ of prime powers. See `https://en.wikipedia.org/wiki/Counter_machine#Two-counter_machines_are_Turing_equivalent_(with_a_caveat)`, accessed 2022-02-05.

As an input for a finite automaton, we encode it as a string U over the alphabet $Q \cup \{\mathtt{0}, \mathtt{1}, \mathtt{\#}\}$ with an end marker $\mathtt{\#}$:

$$U = \mathtt{0}^{l_0}\mathtt{1}^{r_0}q_0\mathtt{0}^{l_1}\mathtt{1}^{r_1}q_1\mathtt{0}^{l_2}\mathtt{1}^{r_2}q_2\ldots\mathtt{0}^{l_n}\mathtt{1}^{r_n}q_n\mathtt{\#} \tag{1}$$

There are some conditions for an accepting computation that a deterministic finite automaton can easily check: Does the string conform to this format? Do the state transitions follow the rules? Is $l_0 = r_0 = 0$? Is the initial and the final state correct? We refer to these checks as the *formal checks.*

The only thing that a finite automaton cannot check is the consistency of the counters, for example, whether $l_{i+1} = l_i$ (or $l_i + 1$, or $l_i - 1$, as appropriate).

For this task, we use the probabilistic capacities of the PFA. If there is an accepting computation U for the counter machine of the form (1), we feed this computation as input to the PFA again and again. In other words, we input the string U^m for a large enough m. We will set up the PFA in such a way that it will accept this input with probability at least 0.99. On the other hand, if there is no accepting computation, then the PFA will reject any input with probability at least 0.99.

4 The Equality Checker

As an auxiliary procedure, we study a PFA that reads words of the form $\mathtt{a}^i\mathtt{b}^j\mathtt{\#}$, and the goal is to "decide" whether $i = j$. We call this procedure the *Equality Checker.* There are three possible outcomes, "Same", "Different", or "Undecided".

The PFA simulates a competition between two players S and D ("Same" and "Different", or "Sum" and "Double"). There are four coins of different colors.

- Player D flips the red coin twice for each $\mathtt{a}$ and the orange coin twice for each $\mathtt{b}$.
- Player S flips the blue coin and the green coin for each $\mathtt{a}$ and each $\mathtt{b}$.

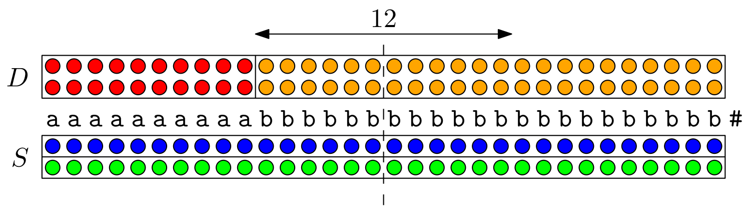
12
D
a a a a a a a a a a b b b b b b b b b b b b b b b b b b b b b b #
S

Figure 1: The coins flipped for the input $\mathtt{a}^{10}\mathtt{b}^{22}\mathtt{\#}$

At the same time, the PFA keeps track of the difference $i - j$ modulo 12. If $i \not\equiv j \pmod{12}$, we declare the outcome to be "Different".

If $i \equiv j \pmod{12}$, the outcome of the game is defined as follows. We call a coin *lucky* if it always came up heads.

- If D has a lucky coin and S has no lucky coin, declare "Different".
- If S has a lucky coin and D has no lucky coin, declare "Same".
- Otherwise, declare "Undecided".

For large i and j, lucky means *extremely lucky.* Thus, the first two events are very rare, and the outcome will almost always be "Undecided".

Lemma 1.

- *If* $i = j$*,* $\Pr[\text{“Different”}] = \Pr[\text{“Same”}]$*.*
- *If* $i \neq j$*,* $\Pr[\text{“Different”}] \geq 2^{11} \cdot \Pr[\text{“Same”}]$*.*

Figure 2 illustrates the outcome of the Equality Checker.

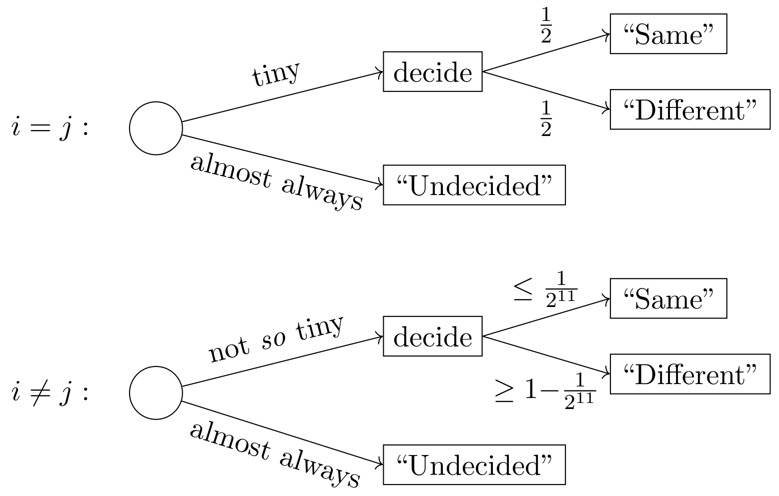

Figure 2: The behavior of the Equality Checker, assuming $i \equiv j \pmod{12}$

Proof. The first statement is clear, since the situation between D and S is symmetric.

Assume that $i \neq j$. If $i \not\equiv j \pmod{12}$, then $\Pr[\text{“Different”}] = 1$.

Otherwise, $|i - j| \geq 12$, and the smaller of i and j, say i, is at most $i \leq \frac{i+j}{2} - 6$, and so the red coin is flipped at most $2i \leq i + j - 12$ times. Thus,

$$Pr[D \text{ has a lucky coin}] \geq Pr[\text{the red coin was lucky}] \geq 1/2^{i+j-12} \tag{2}$$

The blue and the green coin was each flipped $i + j$ times, and hence

$$Pr[S \text{ has a lucky coin}] \leq Pr[\text{the blue coin was lucky}] \tag{3}$$
$$+ Pr[\text{the green coin was lucky}] \leq 2/2^{i+j} \tag{4}$$

The ratio $Pr[D \text{ lucky}]/Pr[S \text{ lucky}]$ between (2) and (3) is at least 2^{11}. From each of these probabilities, we have to subtract the (small) probability that both S and D have a lucky coin, but this tilts the ratio between “Different” and “Same” in D's favor. Formally:

$$\frac{\Pr[\text{“Different”}]}{\Pr[\text{“Same”}]} = \frac{Pr[D \text{ lucky}] - Pr[D \text{ lucky and } S \text{ lucky}]}{Pr[S \text{ lucky}] - Pr[D \text{ lucky and } S \text{ lucky}]} \geq \frac{Pr[D \text{ lucky}]}{Pr[S \text{ lucky}]} \geq 2^{11} \qquad \square$$

5 Correctness Test: Checking a computation

Recall that we wish to check a description of a computation of the following form.

$$U = 0^{l_0} 1^{r_0} q_0 0^{l_1} 1^{r_1} q_1 0^{l_2} 1^{r_2} q_2 \ldots 0^{l_n} 1^{r_n} q_n \#$$

The Equality Checker can be adapted to look at, say, two consecutive zero blocks 0^{l_i} and $0^{l_{i+1}}$ of a computation that represent the values of the counter l and check whether

$l_{i+1} = l_i$. It can also be adapted to check $l_{i+1} = l_i + 1$, or $l_{i+1} = l_i - 1$, as appropriate. The guarantees of Lemma 1 about the outcome remain valid.

We run independent Equality Checkers for each relation between two consecutive values l_i and l_{i+1}, as well as r_i and r_{i+1}, of a computation U. In total, these are $2n$ Equality Checkers. In the schematic drawing of Figure 3, the outcomes of the Equality Checkers are shown as a row of boxes. Typically, most of them will be "Undecided", with a few interspersed "Same" and "Different" results (proportionately much fewer than shown in the first example row). We are interested in the rare cases when all outcomes are "Same", or all "Different".

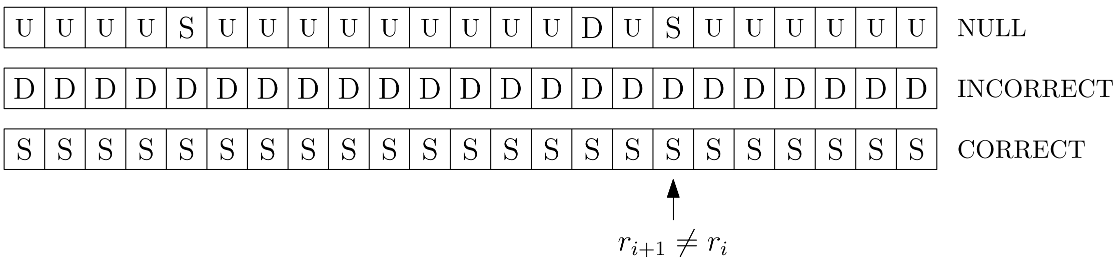
U U U U S U U U U U U U U U D U S U U U U U U
NULL
D D D D D D D D D D D D D D D D D D D D D D D
INCORRECT
S S S S S S S S S S S S S S S S S S S S S S S
CORRECT
$r_{i+1} \neq r_i$

Figure 3: The Correctness Test for a computation U

The output of these Equality Checkers is aggregated into a *Correctness Test* as follows: We report the output "CORRECT" if *all* Equality Checkers report "Same", and we report the output "INCORRECT" if *all* Equality Checkers report "Different". Otherwise, we report "NULL".

To compute this result, only four independent Equality Checkers have to run simultaneously: The current block lengths l_i and r_i have to be compared with the preceding and the next values. Thus, the computation can be implemented by a PFA. (Looking more carefully, one sees that actually, only three Equality Checkers are active at the same time.)

Lemma 2. *Suppose that a computation U of the form* (1) *passes all formal checks.*
If U represents an accepting computation,

$$\Pr[\text{"INCORRECT"}] = \Pr[\text{"CORRECT"}].$$

If U does not represent an accepting computation,

$$\Pr[\text{"INCORRECT"}] \geq 2^{11} \cdot \Pr[\text{"CORRECT"}].$$

Proof. The probability for "CORRECT" is the product of the probabilities that each Equality Checker results in "Same", and analogously, for "INCORRECT".

If U represents an accepting computation, then all Equality Checkers are balanced between "Same" and "Different", and the result is clear. Otherwise, there is at least one position (marked by an arrow in Figure 3) where an error occurs, and the probability for "Different" is at least 2^{11} times larger than for "Same". In all other Equality Checkers, the probability is either balances or it gives a further advantage for "Different". Thus, the product of the probabilities is at least 2^{11} times larger for "all Different" than for "all Same". $\square$

6 Processing the whole input

As mentioned, we feed the PFA with sufficiently many copies of an accepting computation U. Each copy of U is subjected to the Correctness Test.

If we take the first definite result ("CORRECT" or "INCORRECT") as an indication whether to accept or reject the input, we get an acceptance probability close to $1/2$ on an accepting input. (It is slightly less than $1/2$ because of the small chance that the input runs out before a definite answer.) On the other hand, if there is no accepting computation, the algorithm will recognize and reject any "fake" input with probability at least $1 - 1/2^{11}$.

6.1 Boosting the acceptance probability

We can modify the rules to make the acceptance probability larger, at the expense of the rejection probability. We determine the overall result as follows. As soon as a Correctness Test yields "CORRECT", we accept the input. However, we reject the input only if we receive 10 answers "INCORRECT" before receiving an answer "CORRECT". If the end of the input is reached before any of these events happens, we also reject the input. Of course, we also reject the input immediately if any of the formal checks fail.

Theorem 1. *If there is an accepting computation U for 2-CM, then the PFA accepts the input U^m, for sufficiently high m, with probability at least* 0.99.

If there is no accepting computation, then the PFA rejects any input with probability at least 0.99.

Proof. If U is an accepting computation, the distribution between "CORRECT" and "INCORRECT" is fair. Thus, the probability of receiving 10 outputs "INCORRECT" before receiving an output "CORRECT" is $1/2^{10} < 0.001$. To this we must add the probability of rejection because the input runs out before receiving an output "CORRECT", but this can be made arbitrarily small by increasing m.

If there is no accepting computation, then "INCORRECT" has an advantage over "CORRECT" by a factor at least 2^{11}. Thus, the probability of receiving 10 outputs "INCORRECT" before receiving an output "CORRECT" is at least

$$\left(\frac{2^{11}}{2^{11}+1}\right)^{10} = \left(1 - \frac{1}{2^{11}+1}\right)^{10} \geq (1 - \tfrac{1}{2000})^{10} \approx 1 - \tfrac{1}{200} = 0.995. \qquad \square$$

If the 2CM halts, there is an accepting computation U, and since the 2CM is deterministic, U is unique. In this case, the language recognized by the PFA with cutpoint $\lambda = \frac{1}{2}$ is $\{\, U^m \mid m \geq m_0 \,\}$ for some large m_0. Otherwise, the language is empty.

As a consequence, checking whether the language accepted by a PFA is empty is undecidable.

It is a rewarding exercise to calculate the necessary number m of repetitions. Suppose that there is an accepting computation U of length n. Then the counter values l_i and r_i are also bounded by n. The probability of the outcome "Same" in the Equality Checker is roughly 2^{-n}, and the probability that all $2n$ Equality Checkers for the computation U yield "Same", leading to the answer answer "CORRECT", is roughly $(2^{-n})^{2n} = 4^{-n^2}$. We want the probability that none of m experiments gets the answer "CORRECT" to be ≤ 0.009:

$$(1 - 4^{-n^2})^m \leq 0.009$$

Thus we need m to be roughly of the order $4^{n^2} + 5$.

6.2 Further boosting the decision probabilities

We can boost the decision probabilities beyond 0.99 to become arbitrarily close to 1 by running an odd number of copies of the PFA simultaneously and taking a majority vote.

Alternatively, the number K of times that one waits for "INCORRECT" before rejecting the input can be increased beyond $K = 10$. As a compensation, one has to increase the modulus Q (in our case, $Q = 12$) by which i and j are compared in the Equality Checker. The acceptance probability in case of a valid input increases to become arbitrarily close to $1 - 1/2^K$, and the rejection probability for an invalid input is at least $(1 - 1/2^{Q-1})^K$.

7 Formal statement of the result

Formally, a PFA is given by a sequence of stochastic *transition matrices* M_σ, one for each letter σ from the input alphabet Σ. The matrices are $n \times n$ matrices if the PFA has n states. Our PFA can be constructed to have a single accepting state.

If we assume that the starting state is the first state and the accepting state is the last state, the acceptance probability for an input word is the upper right entry in the product of the corresponding transition matrices.

Thus the PFA Emptiness Problem with cutpoint λ, whose undecidability we have shown, can be formally described as follows.

> PFA Emptiness. Given a finite set of $n \times n$ rational stochastic matrices $\mathcal{M}$, is there a product $M_1 M_2 \dots M_m$, with $M_i \in \mathcal{M}$ for all $i = 1, \dots, m$, whose upper right corner is larger than 1/2:
>
> $$(M_1 M_2 \dots M_m)_{1,n} > \lambda \ ? \tag{5}$$

The most natural choice is $\lambda = \frac{1}{2}$, but we have seen that we can choose any cutpoint λ with $0 < \lambda < 1$, and we can also ask $\geq \lambda$ instead of $> \lambda$.

Since we have only used fair coin tosses, the denominators of the entries of our transition matrices are powers of two. Moreover, by padding the input, we can ensure that the PFA algorithm needs to toss at most one coin per input symbol, and thus the entries of the matrices can be restricted to $0, \frac{1}{2}, 1$. In the algorithm as described, only 12 coin tosses are necessary per input character (four coins per Equality Checker running at any point in time). Thus we simply add 11 padding symbols `*` after each `0` and `1` in the input U that encodes an accepting computation (1). In addition, we can encode the input U over a binary alphabet, by using a binary code for the original input alphabet $Q \cup \{\texttt{0}, \texttt{1}, \texttt{\#}, \texttt{*}\}$. This means that the set $\mathcal{M}$ can be restricted to only two matrices.

Theorem 2. *For any fixed λ with $0 < \lambda < 1$, the PFA Emptiness Problem with cutpoint λ is undecidable, even if $\mathcal{M}$ consists only of two transition matrices, all of whose entries are from the set $\{0, \frac{1}{2}, 1\}$.*

As we pointed out, for any $\varepsilon > 0$ we can construct the PFA in such a way that it either accepts *some word* with probability at least $1-\varepsilon$, or it accepts *no word* with probability larger than ε. This does not mean that there cannot be words whose acceptance probability is between those ranges, for example close to 1/2. The words U^m where m is "too small" candidates for such words.

Some more general definitions of a PFA allow an arbitrary starting probability distribution π over the states instead of a fixed starting state. It is also possible to have several accepting states. In this case, the acceptance probability that should take the place of (5) is $\pi^T M_1 M_2 \dots M_m \eta$, where $\eta \in \{0,1\}^n$ is the characteristic vector of the accepting state set.

8 The other proof

I found a readable account of Nasu and Honda's result in the textbook of Volker Claus [2, Satz 28, p. 157].

8.1 Post's Correnspondence Problem (PCP)

We are given a list of pairs of strings $(v_1, w_1), (v_2, w_2), \dots (v_k, w_k)$ over the alphabet $\{\mathtt{0}, \mathtt{1}\}$. The problem is to decide if there is a nonempty sequence $a_1 a_2 \dots a_m$ of indices $a_i \in \{1, 2, \dots, k\}$ such that

$$v_{a_1} v_{a_2} \dots v_{a_m} = w_{a_1} w_{a_2} \dots w_{a_m}$$

This is one of the well-known undecidable problems.

8.2 The binary PFA

For a string $u \in \{0, 1\}^*$, we denote by $(u)_2$ the integer value of u when it is interpreted as a binary number, and we write $|u|$ for the length of u. Then we define the following stochastic matrix:

$$B(u) := \begin{pmatrix} 1 - \frac{(u)_2}{2^{|u|}} & \frac{(u)_2}{2^{|u|}} \\ 1 - \frac{(u)_2+1}{2^{|u|}} & \frac{(u)_2+1}{2^{|u|}} \end{pmatrix}$$

Note that the top right entry $\frac{(u)_2}{2^{|u|}}$ of this matrix is the value $0.u$ when interpreted as a binary fraction, and we will freely use this notation. A straightforward calculation confirms the following multiplication law:

$$B(u)B(v) = B(vu) \tag{6}$$

Let us look at the first sequence of strings $v_1, \dots, v_k$. We construct the PFA with input alphabet $\{1, 2, \dots, k\}$, two states Φ_0 and Φ_1, and transition matrices are $M_i = B(v_i)$. If we take Φ_0 as the starting state and Φ_1 as the accepting state, then the acceptance probability can be found in the upper right corner of the product of the transition matrices, and it is clear from (6) that the acceptance probability of the string $a = a_1 a_2 \dots a_m$ is

$$\phi(a) = 0.v_{a_m} v_{a_{m-1}} \dots v_{a_2} v_{a_1}. \tag{7}$$

We can build an analogous PFA for the other sequence of strings $w_1, \dots, w_k$, and its acceptance probability will be

$$\psi(a) = 0.w_{a_m} w_{m_{n-1}} \dots w_{a_2} w_{a_1}. \tag{8}$$

Due to the reversal of the factors in (6), the order in which the words are concatenated (7) and (8) in the reverse order compared to the usual convention, but this does not change the solvability of the PCP. Thus the PCP comes down to the question whether there is a nonempty string a with $\phi(a) = \psi(a)$. We have to be careful because trailing zero don't change the probabilities (7) and (8). An easy solution is to add a `1` after every symbol of every word, thus doubling the length of the words. This ensures that there are no trailing zeros that could be ignored.

The is a construction to build a PFA that recognizes these strings, based of the formula

$$\tfrac{1}{4}(1 - \phi^2) + \tfrac{1}{4}(1 - \psi^2) + \tfrac{1}{2}\phi\psi = \tfrac{1}{2} - \tfrac{1}{4}(\phi - \psi)^2. \tag{9}$$

It is straightforward to build a PFA with acceptance probability $\phi(u)\psi(u)$, see Figure 4: This PFA simulates the two PFAs for $v_1, \ldots, v_k$ and for $w_1, \ldots, w_k$ simultanously and accepts only if both these PFAs accept. This PFA has four states. Similarly, we can build a PFA with acceptance probability $\phi(u)^2$: We simulate two *independent* copies of the PFAs for $v_1, \ldots, v_k$. Again this uses four states. However, one can see that the states (Φ_0, Ψ_1) and (Φ_1, Ψ_0) of Figure 4 become indistinguishable when $\phi = \psi$. Thus, they can be merged into one state, and we need only three states.

To get acceptance probability $1 - \phi(u)^2$, we complement the set of accepting states. The PFA for $1 - \psi(u)^2$ follows the same principle. Finally, we mix the three PFAs in the ratio $\frac{1}{4} : \frac{1}{4} : \frac{1}{2}$, as shown in Figure 5a.

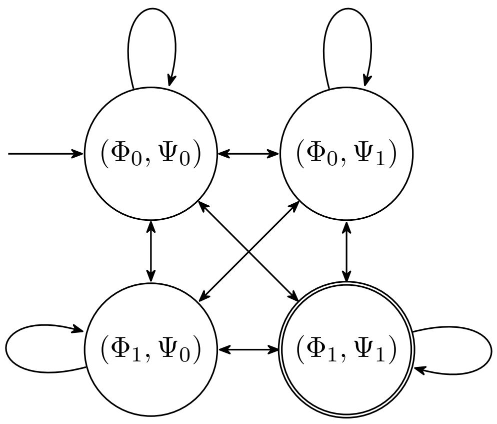

Figure 4: Acceptance probability $\phi\psi$.

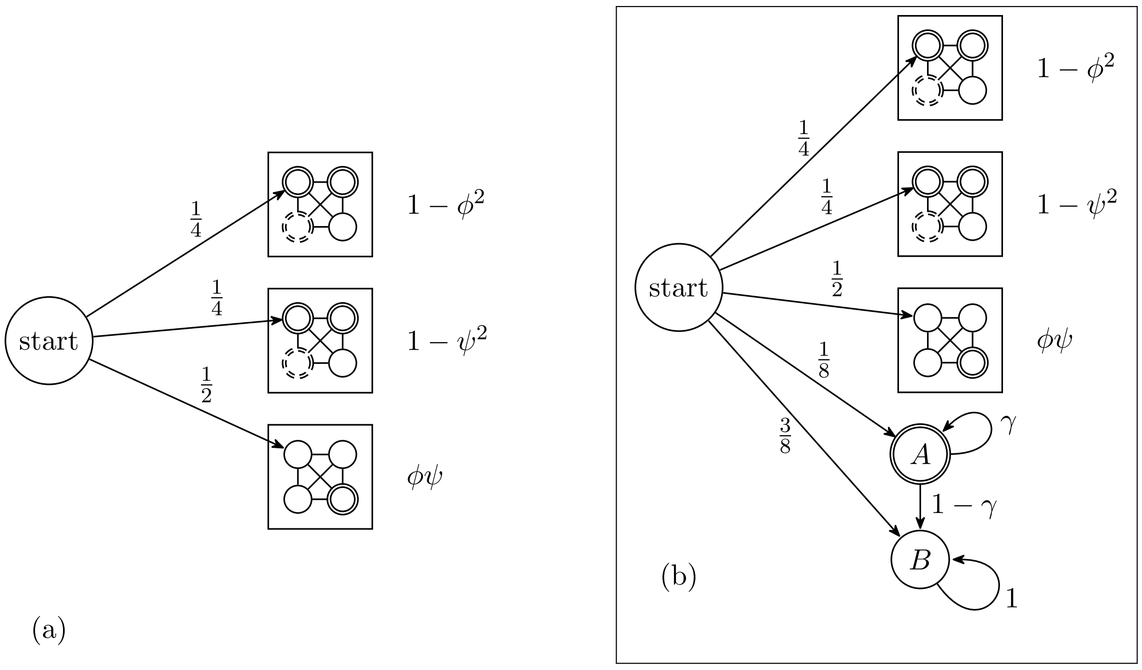

Figure 5: (a) Acceptance probability $\frac{1}{2} - \frac{1}{4}(\phi - \psi)^2$. (b) $\frac{1}{4} - \frac{1}{8}(\phi - \psi)^2 + \varepsilon$

The random transitions from the "start" state can be thought of as taking place before the algorithm starts to read its input symbols. In the actual PFA, these transition are carried out together with the transition for the first symbol. The introduction of the new start state has also eliminated the empty word, which would otherwise have satisfied the equation $\phi(a) = \psi(a)$.

Proposition 1. *Given a finite set of* 11×11 *stochastic matrices* $\mathcal{M}$ *with rational entries whose denominator is a power of 2, it is undecidable if there a product* $M_1 M_2 \dots M_m$*, with* $M_i \in \mathcal{M}$ *for all* $i = 1, \dots, m$*, whose upper right corner is* $\geq 1/2$.

This is almost a PFA, except that the convention for such an automaton to recognize a word is strict inequality ($> \lambda$). We thus have to raise the probability just a tiny little bit, without raising any of the values $< 1/2$ to become bigger than $1/2$.

Since all probabilities are rational, this can be done as follows, see Figure 5b. Suppose that all entries in the transition matrix are multiples of some number γ. (We can set $\gamma = 4^{-\max\{|v_i|,|w_i|:1\leq i\leq k\}}$.) The original PFA is carried out with probability 1/2. We create a new accepting state A that is entered initially with probability 1/8. Whenever a symbol is read, the PFA stays in that state with probability γ, and otherwise it moves to some absorbing state B.

The new part contributes $\varepsilon := \frac{1}{8}\gamma^{|a|}$ to the acceptance probability of every nonempty word a. From the old part we have $\frac{1}{4} - \frac{1}{8}(\phi - \psi)^2$, and we know that the probability must be a multiple of $\frac{1}{4}\gamma^{|a|}$. Thus, if the probability is less than 1/4, it cannot become greater than 1/4 by adding ε. If it was 1/4 (i.e., if a is a solution to the PCP), it becomes greater than 1/4.

Proposition 2. *It is undecidable whether the language recognized by a PFA with 13 states with cutpoint* $\lambda = 1/4$ *is empty.*

This PFA has a fixed starting state. The cutpoint can also adjusted to any value between 0 and 1, at the expense of adding another state. (For achieving a cutpoint value between 0 and 1/2, it is sufficient to adjust the initial split probability between the original PFA of Figure 5a and the states A and B.)

We can save a state by using the *Modified Post Correspondence Problem* (MPCP). The MPCP is often used as an intermediate problem when reducing the Halting Problem for Turing machines to the PCP, see for example [5, Theorem 8.8].

It differs from the PCP as follows: The pair (v_1, w_1) must be used as the *starting pair*, and in cannot be used in any other place. In other words: $a_1 = 1$, and $a_i > 1$ for $i = 2, \dots, m$. The idea is therefore to apply the transition for the first letter a_i right away, and use the resulting distribution on the states as the starting distribution.

There is still a technical discrepancy: In (7) and (8), the first letter of a to be read will determine the *last* words to be concatenated. Thus we must reverse all strings v_i and w_i and turn the MPCS into a *reversed* MPCP, where the *last* pair in the concatenation is prescribed to be the pair (v_1, w_1):

> The Reversed Modified Post Correspondence Problem.
>
> We are given a list of pairs of strings $(v_1, w_1), (v_2, w_2), \dots (v_k, w_k)$, and the problem is to decide if there is a (possibly empty) sequence $a_2 \dots a_m$ of indices $a_i \in \{2, \dots, k\}$ such that
>
> $$v_{a_m} v_{a_{m-1}} \dots v_{a_2} v_1 = w_{a_m} w_{a_{m-1}} \dots w_{a_2} w_1 \ ?$$

For this problem, the translation of (7) and (8) applies directly.

Proposition 3. *It is undecidable whether the language recognized by a PFA with 12 states, and with a probability distribution over the starting states, with cutpoint* $\lambda = 1/4$ *is empty.*

Proof. The construction for Proposition 2 gives a set of 13×13 matrices such that the index sequence $a_1 a_2 \dots a_m$ is a solution of the PCP iff

$$e_1^T M_{a_m} M_{a_{m-1}} \dots M_{a_2} M_{a_1} f = \tfrac{1}{4},$$

where e_1 is the first unit vector $(1, 0, \dots, 0)$. Since the last letter a_m and hence the first matrix M_{a_m} is fixed, the product $e_1^T M_{a_m}$ has a fixed value π^T, and we can replace it by this value:

$$\pi^T M_{a_{m-1}} \dots M_{a_2} M_{a_1} f$$

This is the expression for the acceptance probility starting from an initial probability distribution π.

The PFA goes from the start state to the 12 other states and never returns; thus we can eliminate the start state after the first step, and only use the 12×12 submatrices without the start state. □

We can even achieve a stronger result, by using a Universal Turing Machine. A Universal Turing Machine is a fixed Turing Machine that can simulate any other Turing Machine. In particular, the Halting Problem for such a machine is undecidable. Given some input tape, does the machine halt? In the standard translation from the Halting problem to the MPCP, the transition rules of the TM are translated into pairs (v_i, w_i). The input for the TM is translated into the starting word (v_1, w_1). In our translation to the PFA emptyness, the starting word (v_1, w_1) affects only the starting distribution π, whereas the transition matrices M_i depend only on the rules of the Universal TM, which are fixed!

Since we want a MPCP with as few pairs as possibly, we slightly modify the construction of the MPCP from the TM. We ensure that the configuration is padded with sufficiently many blank symbols. This eliminates the need to deal with special cases when the TM "touches the bounday".

If the input word for the TM is w, we start with the pair $(v_1, w_1) = (\texttt{\#}, \texttt{\#}Bq_0wB\texttt{\#})$. The other pairs are as follows: (s, s) for each $s \in \Gamma$. The following pair is different from the standard construction: $(\texttt{\#}, B\texttt{\#}B)$. We emit an additional blank symbol at both ends of the configuration in each step.

For each right-moving rule (q, s, q', s', R), the pair $(qs, s'q')$. For each halting rule $(q, s, -)$, the pair (qs, H). For each left-moving rule (q, s, q', s', R) and for each $t \in \Gamma$, the pair $(tqs, q'ts')$.

For each $s \in \Gamma$, the pairs (Hs, H) and (sH, H), and finally the pair $(H\texttt{\#\#}, \texttt{\#})$. In total, $3|\Gamma| + 2$ pairs, plus one pair for each right-moving or halting rule, plus $|\Gamma|$ pairs for each left-moving rule (q, s, q', s', R), plus the starting pair.

Theorem 3. *There is a fixed set of 53* 12×12 *rational stochastic matrices* $\mathcal{M}$ *whose entries are multiples of* 2^{-22}, *and a fixed 0-1-vector* $f \in \{0,1\}^{12}$, *for which the following question is undecidable*:

Given a probability distribution $\pi \in \mathbb{R}^{12}$ *whose entries are rational numbers with denominators that are powers of* 2, *is there a product* $M_1 M_2 \dots M_m$, *with* $M_i \in \mathcal{M}$ *for all* $i = 1, \dots, m$, *such that*

$$\pi^T M_1 M_2 \dots M_m f > \tfrac{1}{4}\ ?$$

In other words, is the language recognized by the PFA with starting distribution π *and cutpoint* $\lambda = \frac{1}{4}$ *empty?*

There is another fixed set $\mathcal{M}'$ *of 53* 12×12 *rational stochastic matrices and a starting distribution* π, *all with rational entries whose denominators are powers of* 2, *for which the following question is undecidable*:

Given a vector $f \in \mathbb{R}^{12}$ whose entries are rational numbers with denominators that are powers of 2, is there a product $M_1 M_2 \dots M_m$, with $M_i \in \mathcal{M}'$ for all $i = 1, \dots, m$, such that

$$\pi^T M_1 M_2 \dots M_m f > \tfrac{1}{4} \ ?$$

Proof. The construction gives a set of 13×13 matrices such that the index sequence $a_1 a_2 \dots a_m$ is a solution of the PCP iff

$$e_1^T M_{a_m} M_{a_{m-1}} \dots M_{a_2} M_{a_1} f = \tfrac{1}{4},$$

and otherwise it is $> \frac{1}{4}$, where e_1 is the first unit vector $(1, 0, \dots, 0)$. Since the last letter a_m and hence the first matrix M_{a_m} is fixed, we can compute $\pi^T = e_1^T M_{a_m}$ and substitute $e_1^T M_{a_m}$ by π^T. The PFA goes from the start state to the 12 other states and never returns; thus we can eliminate the start state after the first step, and only use the 12×12 submatrices without the start state.

IN FACT, the last word pair in the PCP is also known!: $(H\#\#, \#$.

In the other case $M_{a_1} f = f'$ is fixed, and we replace

There is one technicality that needs to be resolved. γ was supposed to be a common divisor of the matrix entries, which depend on the maximum lengths $|v_i|$ and $|w_i|$ of the input strings. However, the lengths $|v_1|$ and $|w_1|$ depend on the input tape, and thus are unbounded. The solution is to carry out the imagined first transition (which is not encoded into a transition matrix in $\mathcal{M}$, but determines the starting distribution) with a sufficiently high small value of γ, namely $\gamma = 4^{-\max\{|v_1|, |w_1|\}}$. The other transition at the state A can be carried out with the value γ that is sufficient for those entries. □

state q	π_q	state q	π_q	state q	π_q
(Φ_0, Ψ_0)	$\frac{1}{4}(1 - 0.v_1)(1 - 0.w_1)$				
(Φ_0, Ψ_1)					
(Φ_0, Ψ_1)					
(Φ_1, Ψ_1)	$\frac{1}{4} \cdot 0.v_1 \cdot 0.w_1$				

Table 1: starting probabilities

With the weak inequality $\geq \frac{1}{2}$ instead of $> \frac{1}{4}$, the statement holds for 10×10 matrices.

It would be interesting to look at some UTMs and check how many pairs (v_i, w_i) are actually produced by the translation, in order to see how small $\mathcal{M}$ can be taken.

The pairs (v_i, w_i) into which the TM rules are translated are actually quite short: at most 3 letters long. If the TM has states Q and tape alphabet Γ, the alphabet for the words v_i and w_i is $Q \cup \{H\} \cup \Gamma \cup \{\#\}$, where H is a halting state, with an extra separation symbol #. There are universal Turing machines with 4 states and 6 tape symbols.

Rogozhin's (4, 6) machine uses only 22 instructions, Rogozhin, Yurii (1996), "Small Universal Turing Machines", Theoretical Computer Science, 168 (2): 215–240, doi:10.1016/S0304-3975(96)00077-1

maybe go to PCP from TAG directly?

TAG → UTM → MPCP. string-matching → machine → string-matching

states/symbols = (15, 2), (9, 3), (6, 4), (5, 5), (4, 6), (3, 9), and (2, 18)

$U_{15,2}$ has 15 states and 2 symbols

This means that the alphabet can be coded in 5 bits. (The halting state has to be added.) The input words have then at most 15 bits, and entries of the transition matrices are multiples of $1/4^{15}$. A variable-length code might be more efficient, each word contains at most one state, but several letters. Using 5-letter codes of the form

`0****` for the 15 states plus the halting state leaves 3-letter codes `1**` for the 4 symbols $\Gamma \cup \{\texttt{#}\}$, leading to word lengths bounded by $5 + 3 + 3 = 11$.

We only need to take care that the code for `#` is not all zeros, and then the trailing zero problem cannot arise.

e.g. (9, 3), 2 bits per symbol 2+4 bits per letter (sometimes less)

UTM(7,4) Section 8, Rogozhin, 7 states and 4 symbols

UTM(10,3) Section 8, Rogozhin, 7 states and 4 symbols

Fundamenta Informaticae 91 (2009) 105–126 105 DOI 10.3233/FI-2009-0008 IOS Press Four Small Universal Turing Machines Turlough Neary Damien Woods `http://mural.maynoothuniversity.ie/12416/1/Woods_FourSmall_2009.pdf`

(15, 2),

(9, 3), Section 3.2 $U_{9,3}$. 13 right-moving rules, 1 halting rule, 13 left-moving rules. $|\Gamma| = 3$

$9 + 2 + 13 \times 3 + 14 = 63$

(5, 5), Section 3.3 $U_{5,5}$. 11 right-moving rules, 3 halting rules, 11 left-moving rules. $|\Gamma| = 5$

$15 + 2 + 11 \times 5 + 14 = 86$

(6, 4), Section 3.4 $U_{6,4}$. 13 right-moving rules, 1 halting rule, 10 left-moving rules. $|\Gamma| = 4$

$12 + 2 + 10 \times 4 + 14 = 68$

Section 3.5 $U_{15,2}$. 14 right-moving rules, 1 halting rule, 15 left-moving rules. $|\Gamma| = 2$

$6 + 2 + 15 \times 2 + 15 = 53$

Section 3.2 $U_{9,3}$. 13 right-moving rules, 1 halting rule, 13 left-moving rules. $|\Gamma| = 3$

$9 + 2 + 13 \times 3 + 14 = 64$

Universal Turing Machines for which the Halting Problem is Undecidable.

R It is a rewarding exercise to see how the method of equality testing by the formula (9) would apply to the Equality Checker problem of Section 4: It is easy to set up a PFA that accepts $\texttt{a}^i\texttt{b}^j\texttt{#}$ with probability $\phi = 1/2^i$, and another that accepts it with probability $\psi = 1/2^j$. The construction of Figure 5a, translated into the language of the Equality Checker, runs as follows. The coins are flipped as usual. In the end, when reading the symbol `#`, the PFA flips two more coins, and

- With probability 1/4, it accepts if the red coin was unlucky.
- With probability 1/4, it accepts if the orange coin was unlucky.
- With probability 1/2, it accepts if the blue coin was lucky.

The resulting probability of acceptance is $\frac{1}{4}(1 - 1/4^i) + \frac{1}{4}(1 - 1/4^j) + \frac{1}{2} \cdot 1/2^{i+j} = \frac{1}{2} - \frac{1}{4}(1/2^i - 1/2^j)^2$, which reaches its maximum value 1/2 iff $i = j$.

[reference ??? This is proved in Minsky's book (Computation, 1967, p. 255–258), Marvin Minsky (1967). Computation: Finite and Infinite Machines (1st ed.). Englewood Cliffs, N. J.: Prentice-Hall, Inc. In particular see chapter 11: Models Similar to Digital Computers and chapter 14: Very Simple Bases for Computability. In the former chapter he defines "Program machines" and in the later chapter he discusses "Universal Program machines with Two Registers" and "...with one register", etc.]

References

[1] Vincent D. Blondel and John N. Tsitsiklis. The boundedness of all products of a pair of matrices is undecidable. *Systems & Control Letters*, 41(2):135–140, 2000. `doi:10.1016/S0167-6911(00)00049-9`.

[2] Volker Claus. *Stochastische Automaten.* Teubner Studienskripten. B. G. Teubner, Stuttgart, 1971.

[3] A. Condon and R. J. Lipton. On the complexity of space bounded interactive proofs. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, page 462–467, USA, 1989. IEEE Computer Society. doi:10.1109/SFCS.1989.63519.

[4] Rūsiņš Freivalds. Probabilistic two-way machines. In Jozef Gruska and Michal Chytil, editors, *Mathematical Foundations of Computer Science 1981 (MFCS)*, volume 118 of *Lecture Notes in Computer Science*, pages 33–45. Springer, 1981. doi:10.1007/3-540-10856-4_72.

[5] John E. Hopcroft and Jeffrey E. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[6] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147(1–2):5–34, 2003. doi:10.1016/S0004-3702(02)00378-8.

[7] Marvin L. Minsky. Recursive unsolvability of Post's problem of 'tag' and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961. doi:10.2307/1970290.

[8] Masakazu Nasu and Namio Honda. Mappings induced by PGSM-mappings and some recursively unsolvable problems of finite probabilistic automata. *Information and Control*, 15(3):250–273, 1969. doi:10.1016/S0019-9958(69)90449-5.

[9] Azaria Paz. *Introduction to Probabilistic Automata.* Computer Science and Applied Mathematics. Academic Press, New York, 1971.

[10] Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963. doi:10.1016/S0019-9958(63)90290-0.