Probabilistic Finite Automaton Emptiness is Undecidable

Günter Rote

December 19, 2023

Abstract

It is undecidable whether the language recognized by a probabilistic finite automaton is empty. Several other undecidability results are based on this important theorem. We present two proofs of this theorem from the literature in a self-contained way, and we derive some strengthenings.

Contents

1 Probabilistic finite automata (PFA)

A probabilistic finite automaton (PFA) combines characteristics of a finite automaton and a Markov chain. We give a formal definition below. 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 PFA may make coin flips. As a consequence, 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* (or *represented*) by a PFA is defined by specifying a probability threshold or *cut-point* λ. By convention, the language consists of all words for which the probability of acceptance strictly exceeds λ.

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

This problem is undecidable. There are two independent proofs of this theorem in the literature, by Masakazu Nasu and Namio Honda [14] from 1969, and by Anne Condon and Richard J. Lipton [8] from 1989, based on ideas from Freivalds [10] from 1981. The somewhat intricate history is described in Section 3.

We will present these two proofs, which use very different approaches, see Sections 5 and 4, respectively. (The chains of reductions are shown in Figure 10 in Section 9.) A self-contained proof of the basic undecidability result (Proposition 2) takes less than 3 pages, see Section 5. The rest of the paper is devoted to different sharpenings of the undecidability statement, where certain parameters of the PFA are restricted (Theorems 1–4).

1.1 Formal problem definition

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 $d \times d$ matrices if the PFA has d states. The starting state is chosen according to a given probability distribution $\pi \in \mathbb{R}^d$. The set of accepting states is characterized by a 0-1-vector $f \in \{0,1\}^d$.

In terms of these data, the PFA Emptiness Problem with cut-point λ, whose undecidability we will show, can be formally described as follows.

> PFA Emptiness. Given a finite set of stochastic matrices $\mathcal{M} \subset \mathbb{Q}^{d\times d}$, a probability distribution $\pi \in \mathbb{Q}^d$, and a 0-1-vector $f \in \{0,1\}^d$, is there a sequence $M_1, M_2, \ldots, M_m$ with $M_j \in \mathcal{M}$ such that
>
> $$\pi^T M_1 M_2 \ldots M_m f > \lambda \,? \tag{1}$$

The most natural choice is $\lambda = \frac{1}{2}$, but the problem is undecidable for any fixed (rational or irrational) cut-point λ with $0 < \lambda < 1$. We can also ask $\geq \lambda$ instead of $> \lambda$.

2 Statement of results

The PFA Emptiness Problem is undecidable even if the starting state is a fixed (deterministic) state, and there is a single accepting state (different from the starting state). In that case, π is a standard unit vector, consisting of a single 1 and otherwise zeros, and likewise, f is a standard unit vector. The acceptance probability is found in a specific entry (say, the upper right corner) of the product $M_1 M_2 \ldots M_m$.

Theorem 1. *For any fixed λ with $0 < \lambda < 1$, the PFA Emptiness Problem* (1) *with cut-point λ is undecidable, even when restricted to instances where $\mathcal{M}$ consists of only two transition matrices, all of whose entries are from the set $\{0, \frac{1}{2}, 1\}$, and π and f are standard unit vectors.*

The proof is given in Section 4.

We mention that we don't have to rely on a sharp distinction between $\leq \lambda$ and $> \lambda$, because the PFA that is constructed in the proof exhibits is a strong separation property (see Theorem 5 in Section 4.3.1, and Section 4.3.3): Either there is a sequence of matrices for which the product $\pi^T M_1 M_2 \ldots M_m f$ exceeds $1 - \varepsilon$, or, for any sequence, the value is below ε, where ε be chosen arbitrarily close to 0.

The remaining results deal with the case where all matrices in $\mathcal{M}$ are fixed.

Definition 1. *By a* binary fraction*, we mean a rational number whose denominator is a power of* 2*.*

Theorem 2.

(a) *There is a fixed set $\mathcal{M}$ of 53 stochastic matrices of size 11×11, all of whose entries are multiples of $1/2^{48}$, for which the following question is undecidable*:

Given a probability distribution $\pi \in \mathbb{Q}^{11}$ whose entries are binary fractions, is there a product $M_1 M_2 \dots M_m$, with $M_j \in \mathcal{M}$ for all $j = 1, \dots, m$, such that

$$\pi^T M_1 M_2 \dots M_m e_1 > \tfrac{1}{4}$$

In other words, is the language recognized by the PFA with starting distribution π and cut-point $\lambda = \frac{1}{4}$ nonempty?

(b) *There is a fixed set $\mathcal{M}'$ of 52 stochastic matrices of size 18×18 with positive entries that are multiples of $1/2^{47}$, and a fixed vector $f \in \{0,1\}^{18}$, for which the following question is undecidable*:

Given a probability distribution $\pi \in \mathbb{Q}^{18}$ whose entries are positive binary fractions, is there a product $M_1 M_2 \dots M_m$, with $M_j \in \mathcal{M}''$ for all $j = 1, \dots, m$, with

$$\pi^T M_1 M_2 \dots M_m f \geq \tfrac{1}{2} \ ?$$

The proof is given in section 7.4. In part (a), e_1 denotes the first unit vector in $\mathbb{R}^{11}$, meaning that there is a single accepting state.

Part (a) of the theorem has the acceptance criterion $> \frac{1}{4}$, in line with the conventions for a PFA. Part (b) deviates from this convention by using a weak inequality $\geq \frac{1}{2}$, but this is rewarded by allowing a stronger assumption: All matrices in $\mathcal{M}$ are strictly positive.

The distinction between the cut-points $\frac{1}{4}$ and $\frac{1}{2}$ in parts (a) and (b) is only ephemeral. In fact, for all of the Theorems 2–4, the cut-point can be set to any fixed rational value within some range, but then the assumption that all entries are binary fractions must be given up.

An easier version of Theorem 2a, but with matrices of size 12×12, is proved in Section 6.3 (Proposition 5).

The input alphabet can be reduced to two symbols at the expense of the number of states. The proof will be given in section 7.5.

Theorem 3. *There is a PFA with 572 states, two input symbols with fixed transition matrices, all of whose entries are multiples of $1/2^{48}$, and with a single accepting state, for which the following question is undecidable*:

Given a probability distribution $\pi \in \mathbb{Q}^{572}$ whose entries are binary fractions, is the language recognized by the PFA with starting distribution π and cut-point $\lambda = \frac{1}{4}$ nonempty?

More general acceptance. If each state q is allowed to have an arbitrary probability f_q as an "acceptance degree" instead of just 0 or 1, we can also turn things around and fix the starting distribution π, but let the values f_q be part of the input. The following theorem will be proved in Section 7.3.

Theorem 4.

(a) *There is a fixed set $\mathcal{M}''$ of 52 stochastic matrices of size 11×11 and a fixed starting distribution π, all of whose entries are multiples of $1/2^{45}$, for which the following question is undecidable*:

Given a vector $f \in \mathbb{Q}^{11}$ whose entries are binary fractions from the interval $[0, 1]$, is there a product $M_1 M_2 \dots M_m$, with $M_j \in \mathcal{M}'$ for all $j = 1, \dots, m$, such that

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

(b) *There is a fixed set $\mathcal{M}'''$ of 52 positive stochastic matrices of size 9×9 and a fixed starting distribution π, all with positive entries that are multiples of $1/2^{44}$, for which the following question is undecidable*:

Given a vector $f \in \mathbb{Q}^9$ whose entries are binary fractions from the interval $[\frac{1}{4}, \frac{5}{8}]$, is there a product $M_1 M_2 \dots M_m$, with $M_j \in \mathcal{M}'''$ for all $j = 1, \dots, m$, with

$$\pi^T M_1 M_2 \dots M_m f \geq \tfrac{1}{2} \ ?$$

The distinction between parts (a) and (b) is analogous to Theorem 2. This time, part (b) has an additional advantage: In addition to the positivity of all data in $\mathcal{M}$, π, and f, the dimension is reduced from 11 to 9.

In the PFA instances constructed to prove Theorems 2–4, the solution is unique if it exists.

3 Preface, and history

The study of probabilistic finite automata was initiated by Michael Rabin in 1963 [21]. While this was a very active research area in the 1960's, PFA's are less known today. The first proof that PFA Emptiness is undecidable is due to Masakazu Nasu and Namio Honda from 1969 [14, Theorem 21, p. 270]. It proceeds through a series 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. 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 [20, Theorem 6.17 in Section IIIB, p. 190]. The presentation is quite close to the original, but very much condensed (and without ever mentioning the PCP by name!). I suppose, as the result was still recent when the book was written, it was the culmination point of the treatment. It appears as part of the last theorem of the book, before a brief final chapter on applications and generalizations. The result has often been erroneously attributed to Paz, although Paz gave appropriate credit to Nasu and Honda in the closing remarks of the chapter [20, Section IIIB.7, p. 193]. A simpler version of this proof appears in the textbook of Volker Claus from 1971 [7, Satz 28, p. 157] in German.

An independent proof was sketched by Anne Condon and Richard Lipton in 1989 [8]. It arose as an auxiliary result for 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 4 below.

Interlude: Other problems on matrix products. As the formulation (1) shows, the PFA emptiness problem is about products of matrices that can be taken from a given

set $\mathcal{M}$. There are other problems of this type, whose undecidability comes down to PFA emptiness: For example, the *joint spectral radius* of a set $\mathcal{M}$ of $d \times d$ matrices is

$$\limsup_{m\to\infty} \max_{A_1,A_2,\ldots,A_m\in\mathcal{M}} \sqrt[m]{\|A_1 A_2 \ldots A_m\|},$$

where $\|\cdot\|$ denotes an arbitrary norm. In 2000, Blondel and Tsitsiklis [1] proved, based on the PFA emptiness problem, that it is undecidable whether the joint spectral radius of a finite set of rational matrices exceeds 1.

This has recently been generalized in the analysis of the growth rate of *bilinear systems*, see Matthieu Rosenfeld [22] and Vuong Bui [6]. The study of bilinear systems was initiated for a special case of such a system in Rote [24] in the context of a combinatorial counting problem. Corresponding decidability questions are discussed in Rosenfeld [23] and [6, Chapter 6].

see also [5] [2, 3, 4]

why these theorems became interesting for me. Bui [2] LAA, Bui [4]

. . . back to the proofs of PFA emptiness: In 2000, Blondel and Tsitsiklis [1] could arguably complain 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 [12, Sec. 3.1 and Appendix A] in 2003. Moreover, in the publication list on Anne Condon's homepage, the entry for the Condon–Lipton conference paper [8] 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 detail. Condon and Lipton's proof, which is based on ideas of Freivalds, is conceptually simple and illuminating. The current article originated from lecture notes about this proof.

Meanwhile, I struggled with Nasu and Honda's article and tried to penetrate their rendition in Paz [20], which proceeds through a cascade of definitions and lemmas that stretch over the whole book. When I had acquired a rough understanding of some crucial ideas, I was lucky to find the undecidability proof in the textbook of Claus [7, Satz 28, p. 157], which is considerably simplified. The result in [7] is weaker, because the number of input symbols is the number k of word pairs of the PCP, whereas Nasu and Honda establish undecidability already for an input alphabet of size 2. It is, however, easy to reduce the input alphabet, see Lemma 3. (Nasu and Honda's technique for achieving this reduction is considerably more involved, see Section B and Appendix A.)

In this paper, I am trying present the best parts of both proofs in a self-contained way. I use slightly different terminology, and some details vary from the constructions found elsewhere. I have preferred concrete formulations with particular values of the parameters, illustrating them with examples. Generalizations to arbitrary parameters are treated as an afterthought. I have made an effort to streamline the proofs. In particular, the complete Nasu–Honda–Claus proof leading to the main undecidability result of Proposition 2 takes only 3 pages (Section 5), and I encourage the reader to jump directly to this section. In later sections, I will incrementally introduce new ideas that decrease the number of states or deal with variants of the problem, and the treatment becomes more technical. For reference, I review the original Nasu–Honda proof in Appendix A.

Comparison of the proofs. The two proofs use different ideas, and they have different merits: Condon and Lipton's proof leads to an arbitrarily large gap between accepting

[1] `https://www.cs.ubc.ca/~condon/papers/condon-lipton89.pdf`, accessed 2022-08-31.

and rejecting probabilities (Theorem 5 and Section 4.3.3) and it is easy to restrict the input alphabet to 2 symbols (Theorem 1). While the constructions in Condon and Lipton's proof use a the high-level description of a PFA as a randomized algorithm, the proof of Nasu and Honda encourages to work with the transition matrices directly, and consequently, allows a finer control over the number of states. Moreover, by looking at the reductions in detail, one can even show undecidability of the emptiness problem for a *fixed* PFA with 11 states and an input alphabet of size 53, where the only variable is the starting distribution (Theorem 2a). This sharpening of the undecidability statement is the new contribution of this paper in terms of results, and we hope it might find other applications. Another variation of the problem allows as few as 9 states (Theorem 4b).

The distinction between the two proof approaches is highlighted for a particular example, the language $\{\, \mathtt{a}^i\mathtt{b}^i\mathtt{\#} \mid i \geq 0 \,\}$, in Section 9.1.

4 The Condon–Lipton proof via 2-counter machines

This section presents the proof of Condon and Lipton [8] from 1989, leading to Theorem 1.

A counter machine has a finite control, represented by a state q from a finite set Q, and a number of nonnegative counters. There is a designated start state and a designated halting state. Such a machine operates as follows. At each step, it checks which counters are zero. Depending on the outcome of these tests and the current state q, it may increment or decrement each counter by 1, 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 [13] in 1961 and is by now textbook knowledge [11, 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 two counters after i steps, an accepting computation with m steps can be written as follows:

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

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

$$A = \mathtt{0}^{l_0}\mathtt{1}^{r_0} q_0 \mathtt{0}^{l_1}\mathtt{1}^{r_1} q_1 \mathtt{0}^{l_2}\mathtt{1}^{r_2} q_2 \ldots \mathtt{0}^{l_m}\mathtt{1}^{r_m} q_m \mathtt{\#} \tag{2}$$

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 (halting) 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} is equal to 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 A of the form (2) for the counter machine, we feed this computation as

[2] The usual way to simulate a Turing machine by a 2CM proceeds in three 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 that is large enough to accommodate the stack alphabet; 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.

input to the PFA again and again. In other words, we input the string A^t for a large enough t. We will set up the PFA in such a way that there is a strong separation of probabilities: It will accept this input with probability at least 0.99. On the other hand, if there is no accepting computation, then any input will be rejected with probability at least 0.99.

4.1 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 D and S ("Different" and "Same", or "Double" and "Sum"), as shown in Figure 1. There are four unbiased coins of different colors.

- Player D flips the red coin twice for each `a` and the orange coin twice for each `b`.
- Player S flips the blue coin and the green coin for each input symbol (`a` or `b`).

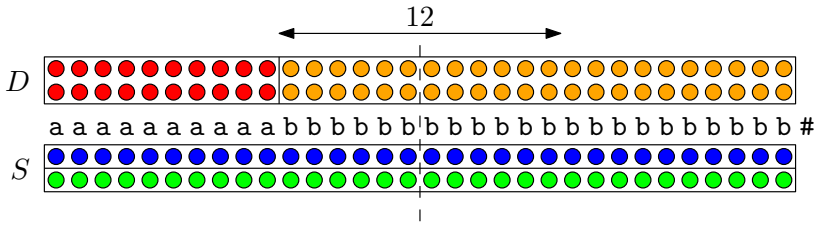

Figure 1: The coin flips for the input $\mathtt{a}^{10}\mathtt{b}^{22}\mathtt{\#}$

In addition, the PFA keeps track of the difference $i - j$ modulo 12. If $i \not\equiv j \pmod{12}$, the PFA declares 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".

Since i and j are usually large, lucky actually means *extremely lucky*. Thus, the first two events are very rare, and the outcome will almost always be "Undecided". Figure 2 illustrates and the following lemma describes the outcome of the Equality Checker.

Lemma 1.

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

Proof. The first statement is clear, since each coin is flipped $2i$ times, and the situation between D and S is symmetric.

Assume that $i \neq j$. If $i \not\equiv j \pmod{12}$, then $\Pr[\text{"Different"}] = 1$, and we are done.

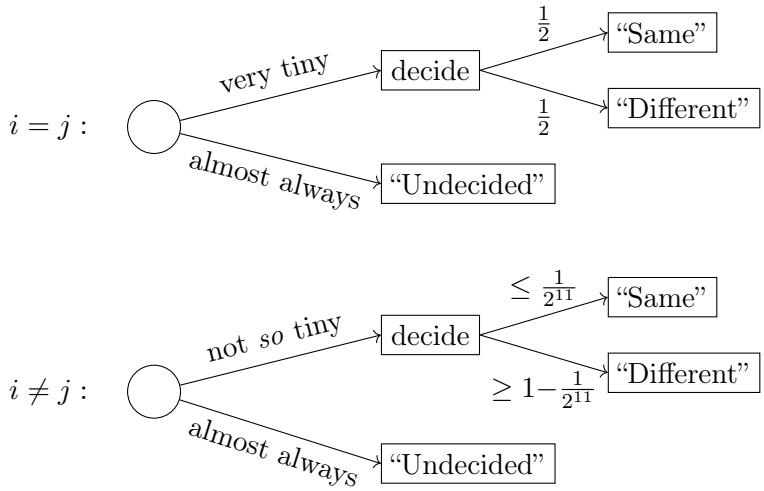

Figure 2: The behavior of the Equality Checker, assuming $i \equiv j \pmod{12}$

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

$$Pr[D \text{ has a lucky coin}] \geq Pr[\text{the red coin was lucky}] \geq 1/2^{i+j-12} \tag{3}$$

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

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

The ratio $Pr[D \text{ lucky}]/Pr[S \text{ lucky}]$ between (3) and (4) 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" even more in D's favor. Formally:

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

Since the algorithm only needs to count up to 11 and maintain a few flags, it is clear that it can be carried out by a PFA.[3]

4.2 Correctness Test: checking a 2CM computation

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

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

[3]As an exercise, the reader may try to work out the number of states. The outcomes should be represented by a partition of the states into four classes, including a category "Rejected" for inputs that don't adhere to the format $\mathtt{a}^i\mathtt{b}^j\mathtt{\#}$. A literal and naive implementation that simply keeps track of every lucky and unlucky coin and sets a flag when a `b` is seen (this is the only thing that needs to be remembered in order to check the syntax, except for a final state change on reading `#`) would need $2^5 \times 12 + 4 = 388$ states. By excluding impossible combinations of flags and with some other tricks like merging states whose distinction is irrelevant (see Section 5.6), I managed to do it with 173 states. If the PFA can trust that the input has the correct format, 108 states suffice.

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 for the state q_i and the results of the zero test of l_i and r_i. 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 A. 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".

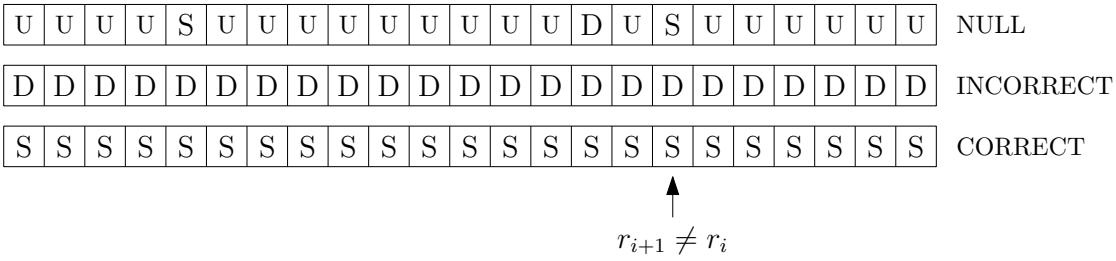

Figure 3: The Correctness Test for a computation A, and a hypothetical position where equality does not hold.

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: While reading the input, 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, with finitely many states. (Looking more carefully, one sees that actually, only three Equality Checkers are active at the same time: For example, when reading 1^{r_i}, the Equality Check between $0^{l_{i-1}}$ and 0^{l_i} has already been completed.)

Lemma 2. *Suppose that a computation A of the form* (2) *passes all formal checks.*

If A represents an accepting computation,

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

If A 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" and "Different".

If A 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", according to Lemma 1. In all other Equality Checkers, the probability is either balanced 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". □

4.3 Second-level aggregation: processing the whole input

We have aggregated the result of many Equality Checkers (with output "Same", "Different", or "Undecided") into a Correctness Test for the string A (with output "CORRECT", "INCORRECT", or "NULL"). We add another level of aggregation in order to decide whether the PFA should accept the input string. As mentioned, we feed the PFA with sufficiently many copies of an accepting computation A. Each copy of A 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 a valid input. (It is not precisely $1/2$ 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 $1 - 1/2^{11}$.

4.3.1 Increasing the acceptance probability

We modify the rules to make the acceptance probability larger, at the expense of the rejection probability for fake inputs. We determine the overall result as follows. As soon as a Correctness Test yields "CORRECT", we accept the input. However, in order to reject the input, we wait until we have received 10 answers "INCORRECT" before receiving an answer "CORRECT". If the end of the input is reached before any of these events happens, this also leads to rejection. Of course, we also reject the input right away if any of the formal checks fails.

Theorem 5. *If there is an accepting computation A for the 2-CM, then the PFA accepts the input A^t, for sufficiently large t, with probability more than* 0.99.

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

Proof. If A 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 t.

If there is no accepting computation, then "INCORRECT" has an advantage over "CORRECT" by a factor at least 2^{11}. If the input runs out before a decision is reached, this is in the favor of rejection. Otherwise, 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 A. (A is unique since the 2CM is deterministic.) In this situation, the language recognized by the PFA with cut-point $\lambda = \frac{1}{2}$ contains the set $\{ A^t \mid t \geq t_0 \}$ for some large t_0. Otherwise, the language is empty.

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

4.3.2 Who is afraid of small probabilities?

As an exercise, we estimate the necessary number t of repetitions of A. Suppose that the accepting computation A has m transitions. Then the counter values l_i and r_i are

also bounded by m. The probability of the outcome "Same" in the Equality Checker is roughly 2^{-m}, and the probability that all $2m$ Equality Checkers for the computation A yield "Same", leading to the answer answer "CORRECT", is roughly $(2^{-m})^{2m} = 4^{-m^2}$.

We want the probability that none of t experiments gets the answer "CORRECT" to be ≤ 0.009 (the difference between the bound $0.001 > 1/2^{10}$ established in the proof of Theorem 5 and the target tolerance 0.01):

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

Since $1 - 4^{-m^2} \approx \exp(-4^{-m^2})$, we need t to be roughly $5 \cdot 4^{m^2}$.

This dependence on the runtime m of the 2-counter machine does not appear so terrible; however, when considering the overhead of simulating a Turing machine (see footnote 2), the dependence blows up to a triply-exponential growth in terms of the runtime of a Turing machine.

4.3.3 Boosting the decision probabilities

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

Alternatively, the number K of times that one waits for "INCORRECT" before rejecting the input can be increased above $K = 10$. As a compensation, one has to increase the modulus G (we have chosen $G = 12$) 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 $(1 - 1/2^{G-1})^K$.

In summary, 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 there is *no word* that it accepts 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 $1/2$. Candidates for such words are the words A^t where t is slightly too small.[4]

4.4 Summing up the proof of Theorem 1

We have described the algorithm for the PFA verbally as a probabilistic algorithm, keeping in mind the finiteness constraints of a finite automaton. Eventually, this algorithm must be translated into a set of states and transition matrices. Theorem 1 puts some extra constraints on the PFA's whose emptiness is undecidable.

Theorem 1. *For any fixed λ with $0 < \lambda < 1$, the PFA Emptiness Problem* (1) *with cut-point λ is undecidable, even when restricted to instances where $\mathcal{M}$ consists of only two transition matrices, all of whose entries are from the set $\{0, \frac{1}{2}, 1\}$, and π and f are standard unit vectors.*

Proof. The extra constraints can be easily fulfilled:

(a) We encode the input A with a fixed-length binary code for the original input alphabet $Q \cup \{\mathtt{0}, \mathtt{1}, \mathtt{\#}\}$. This means that the set $\mathcal{M}$ can be restricted to only two matrices. (Lemma 3 in Section 7.5 below treats this transformation more formally.)

[4] In fact, it is impossible to avoid the neighborhood of $1/2$ except for very simple languages: Rabin [21] showed in 1963 that a gap interval $[p_1, p_2]$ of positive length, such that the acceptance probability is never in this interval, can only exist if the recognized language is regular, see also [20, Theorem 2.3 in Section IIIB, p. 160] or [7, §3.2.2, pp. 112–115].

(b) 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 16 coin tosses are necessary per input character (four coins per Equality Checker running at any point in time). Thus we simply pad each code-word in the binary code with 15 zeros.

(c) Our algorithm does not need to make any coin flips before reading the first symbol. Thus, we can fix the starting state to be a deterministic state.

(d) Finally, a single accepting state is enough: As soon as the algorithm has decided to accept the input, it will stay committed to this decision. The accepting state is an absorbing state, and there is another absorbing state for rejection. (Alternatively, we could allow a transition from the accepting state to rejection in case the algorithm is still checking the formal conditions to the very end.) In terms of vectors, both the starting distribution π and the characteristic vector f of accepting states are standard unit vectors. (Since the empty input is not accepted, the accepting state is distinct from the starting state, and we can arrange the states so that the acceptance probability is found in the upper right corner of the product $M_1 M_2 \dots M_m$.) □

4.5 History of ideas

Condon and Lipton credit the main ideas of their proof to Rūsiņš Freivalds [10], 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 $\{\, \mathtt{a}^i\mathtt{b}^i \mid i \geq 0 \,\}$ (Section 4.1), and aggregating the results of these competitions into "macrocompetitions" (Section 4.2). A 2-way automaton can move the input head back and forth over the input, and thus process the input as often as it wants. Freivalds claimed that the emptiness problem for such automata is undecidable [10, Theorem 4]; he gives only a hint that the reduction should be from the PCP (Post's Correspondence Problem), without any 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 present 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.

5 The Nasu–Honda–Claus proof via Post's Correspondence Problem

This section presents the proof of Nasu and Honda [14] from 1969 in the version of Claus [7] from 1971, leading to the undecidability results in Propositions 1–4, which are then specialized to Theorems 2–4 in the rest of the paper.

5.1 The binary PFA

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

$$B(u) := \begin{pmatrix} 1 - \frac{(u)_2}{2^{|u|}} & \frac{(u)_2}{2^{|u|}} \\ 1 - \frac{(u)_2+1}{2^{|u|}} & \frac{(u)_2+1}{2^{|u|}} \end{pmatrix}, \text{ for example } B(\mathtt{00110}) = \begin{pmatrix} \frac{26}{32} & \frac{6}{32} \\ \frac{25}{32} & \frac{7}{32} \end{pmatrix}.$$

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; for example, $\frac{6}{32} = (0.00110)_2$. It will be convenient to use the notation $0.u$ for this. These matrices fulfill the remarkable multiplication law:

$$B(u)B(u') = B(u'u), \tag{5}$$

which can be confirmed by a straightforward calculation. Note the reversed order of the factors.

5.2 Post's Correspondence Problem (PCP)

In the *Post Correspondence Problem* (PCP), we are given a list of pairs of strings $(v_1, w_1), (v_2, w_2), \ldots, (v_k, w_k)$ over some alphabet. The problem is to decide if there is a nonempty sequence $a_1 a_2 \ldots a_m$ of indices $a_i \in \{1, 2, \ldots, k\}$ such that

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

This is one of the well-known undecidable problems. It is no restriction to fix the alphabet to $\{\mathtt{0}, \mathtt{1}\}$, since any alphabet can be encoded in binary.

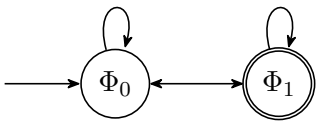

Figure 4: The binary automaton with acceptance probability ϕ

Let us look at the first sequence of strings $v_1, \ldots, v_k$. We construct a PFA with input alphabet $\{1, 2, \ldots, k\}$ and two states Φ_0 and Φ_1, see Figure 4. The transition matrices are $M_i = B(v_i)$. We take Φ_0 as the starting state and Φ_1 as the accepting state. Then the acceptance probability of the string $a = a_1 a_2 \ldots a_m$ is found in the upper right corner of the product $M_{a_1} M_{a_2} \ldots M_{a_{m-1}} M_{a_m}$ of the corresponding transition matrices, and it follows from (5) that this is

$$\phi(a) = 0.v_{a_m} v_{a_{m-1}} \ldots v_{a_2} v_{a_1}. \tag{6}$$

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

$$\psi(a) = 0.w_{a_m} w_{a_{m-1}} \ldots w_{a_2} w_{a_1}. \tag{7}$$

Due to the swapping of the factors in the multiplication law (5), the words are concatenated in (6) and (7) in reverse order, but this does not affect 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 of the *trailing zeros issue*: Trailing zeros don't change the probabilities (6) and (7). An easy way to circumvent this problem is to add a $\mathtt{1}$ after every symbol of every word, thus doubling the length of the words. This ensures that there are no trailing zeros that could go unnoticed.

5.3 Testing equality of probabilities

For recognizing the strings a with $\phi(a) = \psi(a)$, there is a construction of a PFA that does this job. It is based on the identity

$$\tfrac{1}{2}\phi\psi + \tfrac{1}{4}(1-\phi^2) + \tfrac{1}{4}(1-\psi^2) = \tfrac{1}{2} - \tfrac{1}{4}(\phi-\psi)^2. \tag{8}$$

We will build a PFA for each term $\phi\psi$, $1-\phi^2$, $1-\psi^2$ on the left, and we will mix them in the right proportion. As the right-hand side shows, we have then (almost) achieved our goal: The acceptance probability achieves its maximum value $\frac{1}{2}$ only for $\phi(a) = \psi(a)$.

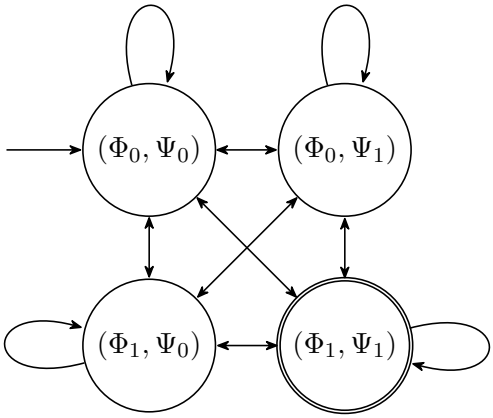

Figure 5: Acceptance probability $\phi\psi$

It is straightforward to build a PFA whose acceptance probability is the product $\phi(a)\psi(a)$, see Figure 5: This PFA simulates the two PFA's for $v_1, \ldots, v_k$ and for $w_1, \ldots, w_k$ simultaneously and accepts if both PFA's accept. The resulting *product PFA* has four states $\{\Phi_0, \Phi_1\} \times \{\Psi_0, \Psi_1\}$. Similarly, we can build a PFA with acceptance probability $\phi(a)^2$: We simulate two *independent* copies of the PFA for $v_1, \ldots, v_k$. This leads again to four states. To get acceptance probability $1-\phi(a)^2$, we complement the set of accepting states. The PFA for $1-\psi(a)^2$ follows the same principle. Finally, we mix the three PFA's in the ratio $\frac{1}{2} : \frac{1}{4} : \frac{1}{4}$, as shown in Figure 6a.

The random transition from the start state can be thought of as taking place before the algorithm starts to read its input. In the actual PFA, this transition is carried out as part of the transition for the first symbol. The introduction of the new start state has the beneficial side effect of eliminating the empty word ϵ from the recognized language, which would otherwise satisfy the equation $\phi(a) = \psi(a)$, because $\phi(\epsilon) = \psi(\epsilon) = 0$. In total, we have now 13 states, 7 of which are accepting.

As an intermediate undecidability result, we can now state:

Proposition 1. *The following problem is undecidable:*

Given a finite set $\mathcal{M}$ of stochastic matrices of size 13×13 with binary fractions as entries, is there a product $M_1 M_2 \ldots M_m$, with $M_i \in \mathcal{M}$ for all $i = 1, \ldots, m$, such that the sum of the 7 rightmost entries in the top row is $\geq \frac{1}{2}$? □

5.4 Achieving strict inequality

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

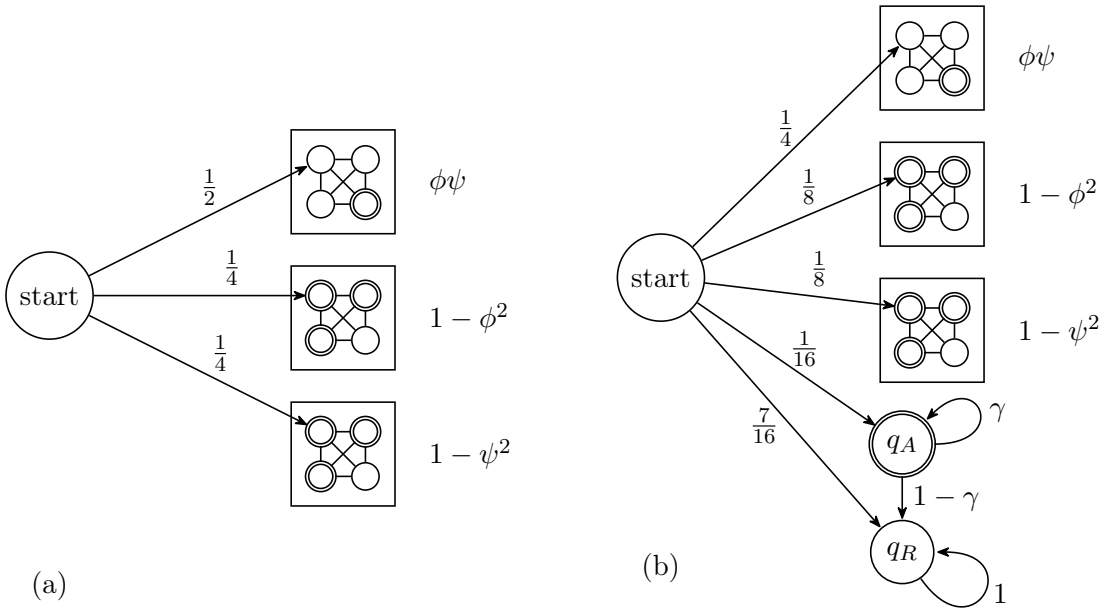

Figure 6: (a) Acceptance probability $\frac{1}{2} - \frac{1}{4}(\phi - \psi)^2$. (b) $\frac{1}{4} - \frac{1}{8}(\phi - \psi)^2 + \varepsilon$

Since all probabilities are rational, this can be done as follows, see Figure 6b. All transition probabilities within the square boxes are multiples of some small unit $\gamma := 4^{-\max\{|v_i|,|w_i|:1\le i\le k\}}$.

The original PFA is entered with probability 1/2. The transition probabilities from the start state into the original PFA are now multiples of $\gamma/8$. We create a new accepting state q_A that is entered initially with probability 1/16. Whenever a symbol is read, the PFA stays in that state with probability γ, and otherwise it moves to some absorbing state q_R.

The new part contributes $\varepsilon := \frac{1}{16}\gamma^{|a|}$ 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 this probability is a multiple of $\frac{1}{8}\gamma^{|a|}$. Thus, if the probability is less than 1/4, it cannot become greater than 1/4 by adding ε. If it was equal to 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 15 states with cut-point $\lambda = 1/4$ is empty.* □

This PFA has a fixed starting state. The cut-point can be changed to any rational value between 0 and 1/2 (exclusive) by adjusting the initial split probability between the original PFA of Figure 6a and the states q_A and q_R. Cut-points between 1/2 and 1 can be achieved at the expense of adding another accepting state.

According to Neary [15], the PCP is already undecidable with as few as five word pairs. Therefore, the size of the input alphabet in Proposition 2, or the number of matrices $\mathcal{M}$ in Proposition 1 can be restricted to 5.

5.5 History of ideas

The binary automaton (Section 5.1) and its generalization to other radices appears already in Rabin's 1963 paper [21], and it is credited to E. F. Moore. (The basic m-ary automaton processes single digits from $\{0, \ldots, m-1\}$. The binary automaton matrix

in Section 5.1 for variable-length input words u is the product of several such single-digit matrices. Instead of the binary automaton, Nasu and Honda use ternary (triadic) automata with input alphabet $\{1, 2\}$ in order to avoid the trailing zeros issue.)

The equality test for probabilities (constructing a PFA to accept words a with $\phi(a) = \psi(a)$ from two PFA's with acceptance probabilities $\phi(a)$ and $\psi(a)$, respectively, Section 5.3), including the method of adding a small probability to change $\geq \lambda$ into $> \lambda$ (Section 5.4) is given in Nasu and Honda [14, Lemma 11, pp. 259–260]. The authors credit H. Matuura, Y. Inagaki, and T. Hukumura for the key ideas (a technical report and a conference record, both from 1968 and in Japanese) [14, p. 261].

Claus already observed [7, p. 158, remark after the proof of Satz 28] that the construction leads to a bounded number of states. The details have been worked out above.

As I haven't been able to survey the rich literature on probabilistic automata, I may very well have overlooked some earlier roots of these ideas.

Nasu and Honda [14], in a footnote to Theorem 12, their main undecidability result, write that "it reduces to a statement in p. 150" of a paper of Marcel Schützenberger [25] from 1963[5] (see [14, footnote 6, p. 270, referring to the remark before Lemma 12, p. 261]). In that paper, Schützenberger derives some undecidability results, using, among others, the PCP, but I haven't been able to see the connection.

Nasu and Honda prove also other undecidable questions in connection with the language recognized by a PFA: whether the language is regular, or whether the language is context-free [14, Theorem 22, p. 270], see Appendix A.1.

5.6 Saving two states

In the PFA with acceptance probability $\phi(a)^2$, where we simulate two independent copies of the same PFA, we can see that the states (Φ_0, Ψ_1) and (Φ_1, Ψ_0) of Figure 5 become indistinguishable when $\phi = \psi$. Thus, they can be merged into one state, denoted by $\{\Phi_0, \Phi_1\}$, and we reduce the number of states by one, see Figure 7a–b.

More precisely, if we denote the the transition probabilities of the original binary automaton by

$$B(u) = \begin{array}{c} \\ \Phi_0 \\ \Phi_1 \end{array} \begin{array}{c} \begin{array}{cc} \Phi_0 & \Phi_1 \end{array} \\ \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} \end{array},$$

the 3-state PFA has the following transition matrix:

$$\begin{array}{c} \\ (\Phi_0, \Phi_0) \\ \{\Phi_0, \Phi_1\} \\ (\Phi_1, \Phi_1) \end{array} \begin{array}{c} \begin{array}{ccc} (\Phi_0, \Phi_0) & \{\Phi_0, \Phi_1\} & (\Phi_1, \Phi_1) \end{array} \\ \begin{pmatrix} p_{00}^2 & 2p_{00}p_{01} & p_{01}^2 \\ p_{00}p_{10} & p_{01}p_{10} + p_{00}p_{11} & p_{01}p_{11} \\ p_{10}^2 & 2p_{10}p_{11} & p_{11}^2 \end{pmatrix} \end{array} \tag{9}$$

When the reduced automaton is in the state $\{\Phi_0, \Phi_1\}$, we can think of the original automaton being in one of the states (Φ_0, Φ_1) or (Φ_1, Φ_0), each with probability $1/2$.

For the PFAs in Propositions 1 and 2, the number of states can thus be reduced by 2, as stated in the following proposition. Figure 7c illustrates the automaton for Proposition 3b. We show some explicit examples of transition matrices for this automaton below, in Section 6.5.

[5] `https://monge.univ-mlv.fr/~berstel/Mps/Travaux/A/A/1963-4ElementaryFamAutomataSympThAut.pdf`

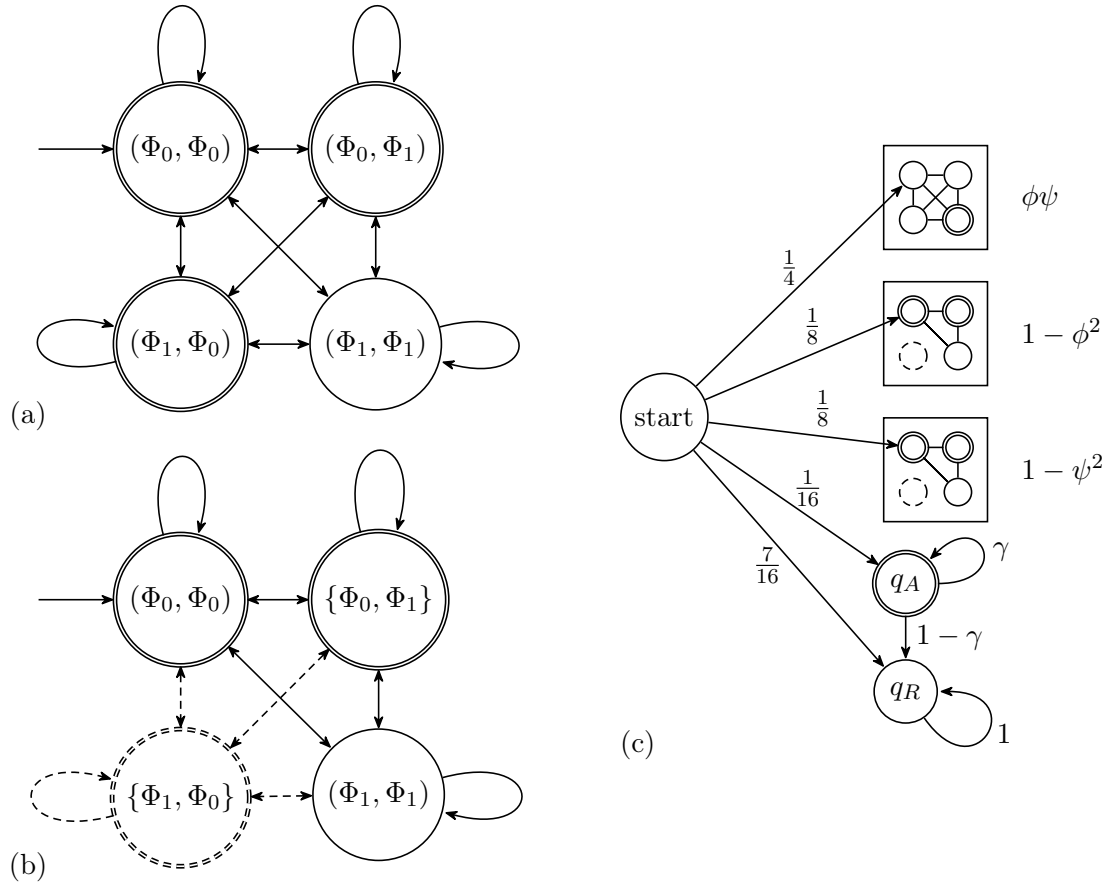

Figure 7: (a) Acceptance probability $1-\phi^2$ with 4 states (b) with 3 states. (c) Acceptance probability $\frac{1}{4} - \frac{1}{8}(\phi - \psi)^2 + \varepsilon$ with 13 states

Proposition 3.

(a) *The following problem is undecidable:*

Given a finite set $\mathcal{M}$ of stochastic matrices of size 11×11 with binary fractions as entries, 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 the sum of the 5 rightmost entries in the top row is $\geq \frac{1}{2}$?

(b) *It is undecidable whether the language recognized by a PFA with 13 states with cut-point $\lambda = 1/4$ is empty.* □

5.7 Saving the starting state by the Modified Post Correspondence Problem

We can eliminate the starting state by using the *Modified Post Correspondence Problem* (MPCP). It differs from the PCP in one detail: The pair (v_1, w_1) must be used as the *starting pair*, and it cannot be used in any other place. In other words: $a_1 = 1$, and $a_i > 1$ for $i = 2, \dots, m$. The MPCP is often used as an intermediate problem when reducing the Halting Problem for Turing machines to the PCP, and it takes some extra effort to reduce the MPCP to the PCP, see for example [11, Theorem 8.8]. For our reduction, the MPCP is actually the more convenient version of the problem.

The idea is to apply the transition for the first letter a_1 right away, and use the resulting distribution on the states as the starting distribution π.

There is still a small technical discrepancy: In the formulas (6) and (7) for the acceptance probability, the first letter of a determines the *last* words to be concatenated. Thus we must reverse all strings v_i and w_i and turn the MPCP 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 (RMPCP).
>
> We are given a list of pairs of strings $(v_1, w_1), (v_2, w_2), \ldots (v_k, w_k)$ over the alphabet $\{0, 1\}$ such that v_1 and w_1 end with 1. The problem is to decide if there is a sequence $a_2, \ldots, a_m$ of indices $a_i \in \{2, \ldots, k\}$ such that
>
> $$v_{a_m} v_{a_{m-1}} \ldots v_{a_2} v_1 = w_{a_m} w_{a_{m-1}} \ldots w_{a_2} w_1 \ .$$

This is of course just a trivial variation of the MPCP. The translation of (6) and (7) can now be applied directly. Moreover, the trailing zeros issue disappears, since v_1 and w_1 end with 1. This extra condition can be easily fulfilled by appending a 1 to v_1 and w_1 if necessary.

Proposition 4. *It is undecidable whether the language recognized by a PFA with 12 states with cut-point $\lambda = 1/4$ is empty.*

Proof. The above construction that has led to Proposition 3b gives a set of 13×13 matrices such that the index sequence $a_1 a_2 \ldots a_m$ is a solution of the PCP if and only if

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

where e_1 is the first unit vector $(1, 0, \ldots, 0)$ and f is a vector with 6 zeros and 7 ones. For any other index sequence, the value of the expression is $< \frac{1}{4}$.

For the reversed MPCP the first matrix $M_{a_1} = M_1$ is fixed. Thus the product $e_1^T M_1$ has a fixed value π^T, and we can replace it by this vector:

$$\pi^T M_{a_2} \ldots M_{a_{m-1}} M_{a_m} f$$

This is the expression for the acceptance probability starting from an initial probability distribution π. The remaining matrix product is not allowed to use M_1, and this is easily ensured by removing M_1 from $\mathcal{M}$.

The original PFA goes from the start state to the 12 other states and never returns to the start state; thus we can eliminate the start state and only use the 12×12 submatrices for the remaining states. □

We mention that with cut-point $\frac{1}{2}$ and the weak inequality $\geq \frac{1}{2}$ as acceptance criterion instead of $> \frac{1}{4}$, we don't need the extra states q_A and q_R, and the number of states is reduced to 10.

6 Fixing the set of matrices by using a universal Turing machine

We can achieve stronger and more specific results by tracing back the undecidability of the PCP to the Halting Problem. In particular, we will look at a universal Turing machine and derive from it a "universal" PCP. 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? Sticking to one fixed machine allows us to choose a fixed set of matrices that represents the PFA. The only variation is the starting distribution π, or, in another variation, the vector f of output values.

6.1 Constructing an MPCP for a Turing machine

In order to adhere to the usual practice, we describe the translation to the MPCP and not to the reversed MPCP. (The words simply have to be reversed for applying the RMPCP.) Also, we temporarily use a general alphabet for the word pairs of the MPCP. In the end, this alphabet will be encoded into the binary alphabet $\{\mathtt{0},\mathtt{1}\}$ in order to be translated into a PFA.

The word $v_{a_1}v_{a_2}\dots v_{a_m} = w_{a_1}w_{a_2}\dots w_{a_m}$ is built as a concatenation of successive *configurations* of the Turing machine, separated by the marker #. The words are built incrementally in such a way that the partial word $v_1v_{a_2}\dots v_{a_n}$ lags one step (of the Turing machine) behind the partial word $w_1w_{a_2}\dots w_{a_n}$. Figure 8 shows an example. Following

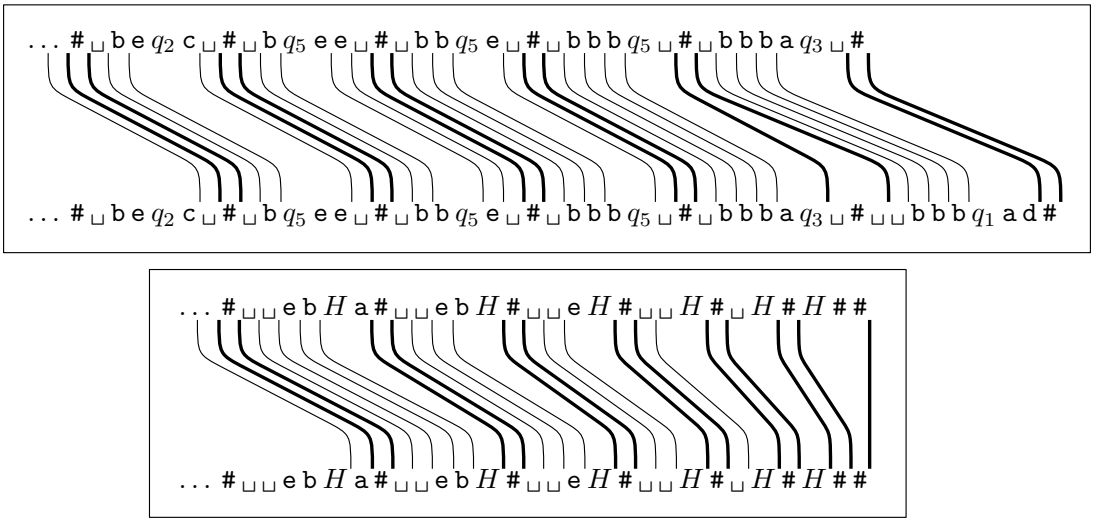

Figure 8: Top: Building two partial words $v_1v_{a_2}\dots v_{a_n}$ (upper row) and $w_1w_{a_2}\dots w_{a_n}$ (lower row) for a Turing machine with transition rules $(q_2, \mathtt{c}, \mathtt{e}, L, q_5)$, $(q_5, \mathtt{e}, \mathtt{b}, R, q_5)$, $(q_5, ␣, \mathtt{a}, R, q_3)$, $(q_3, ␣, \mathtt{d}, L, q_1)$, among others. For better visibility, the correspondences involving the separator # are highlighted. Near the right end, the padding pair $(\mathtt{\#}, ␣\mathtt{\#}␣)$ is used once in order to produce extra blanks at the ends of the tape, preventing the state symbol q_3 from becoming adjacent to the marker #. Bottom: After the Turing machine has halted, the tape is cleared and the common word is finished.

the common convention, a string such as $\mathtt{\#}␣\mathtt{be}q_2\mathtt{c}␣\mathtt{\#}$ denotes the configuration where the Turing machine is in state q_2, the tape contains the symbols bec padded by infinitely many blank symbols on both sides (of which two are shown), and the Turing machine is positioned over the third occupied cell, the one with the symbol c.

The transition rules of the Turing machine are translated into pairs (v_i, w_i). The important feature of this translation is shown in Figure 9: The input for the Turing machine is translated into the starting pair (v_1, w_1). In the above translation to a PFA, leading to Proposition 4, the starting pair (v_1, w_1) affects only the starting distribution π, whereas the transition matrices M_i depend only on the rules of the Turing machine, which, for a universal Turing machine, are fixed!

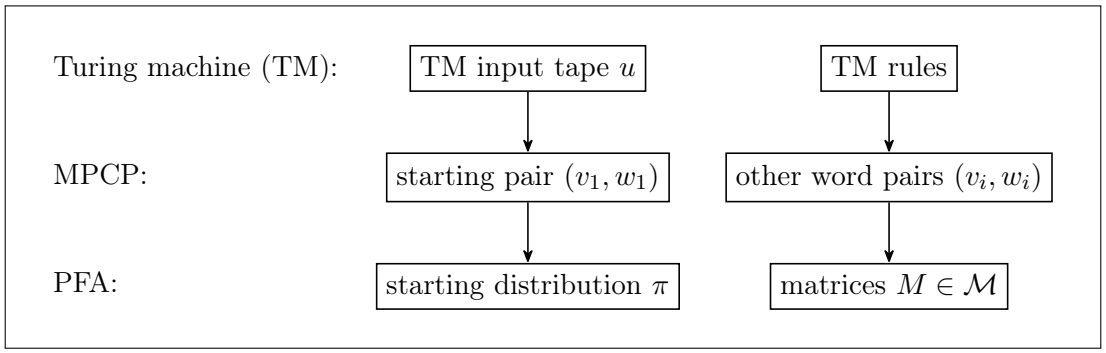

Figure 9: How the PFA is constructed from a Turing machine via an MPCP

6.2 List of word pairs of the MPCP

Since we want a MPCP with as few pairs as possible, we review the construction of the MPCP from the Turing machine in detail. We follow the construction from Sipser [27, Section 5.2, Part 5, p. 187] to ensure that the configurations are padded with sufficiently many blank symbols. This eliminates the need to deal with special cases when the Turing machine reaches the "boundary" of the tape in the string representation.

Let Γ denote the tape alphabet including the blank symbol $\sqcup$, and let Q denote the set of states of the Turing machine. The words v_i and w_i of the MPCP use the alphabet $\Gamma \cup Q \cup \{\#, H\}$ with two extra symbols: a separation symbol $\#$ and a halting symbol H.

- If the input word for the Turing machine is $u \in \{0, 1\}^*$, we define the *starting pair* $(v_1, w_1) = (\#, \#\sqcup q_0 u \sqcup \#)$, where q_0 is the start state of the Turing machine.

The other pairs (v_i, w_i) are as follows:

- For *copying* from the more advanced word to the shorter word, we have the pairs (s, s) for all $s \in \Gamma$.
- We have another copying pair $(\#, \#)$, and the *padding pair* $(\#, \sqcup\#\sqcup)$. We are allowed to (nondeterministically) emit an additional blank symbol at both ends of the configuration.
- For each *right-moving rule* (q, s, s', R, q'), the pair $(qs, s'q')$. (Such a rule means that when the Turing machine is in state q and reads the tape symbol s, it writes the symbol s', moves one step to the right on the tape, and changes to state q'.)
- For each *left-moving rule* (q, s, s', L, q') and for each $t \in \Gamma$, the pair $(tqs, q'ts')$.
- For each *halting rule* $(q, s, -)$, the pair (qs, H). The character H represents the fact that the machine has halted.
- For each $s \in \Gamma$, the *erasing pairs* (Hs, H) and (sH, H). The halting symbol absorbs all symbols on the tape one by one.
- Finally, the *finishing pair* $(H\#\#, \#)$. This is the only way how the strings can come to a common end.

In total, these are $3|\Gamma| + 3$ pairs, plus $|\Gamma|$ pairs for each left-moving rule, plus one pair for each right-moving or halting rule, plus the starting pair.

6.3 Using a universal Turing machine

We have looked at the parameters of various universal Turing machines in the literature in order to see which ones give the smallest number of pairs (v_i, w_i) for our PCP. The best result is obtained from the machine $U_{15,2}$ of Neary and Woods [16, Section 3.5].[6] Its tape alphabet, including the blank symbol, has size $|\Gamma| = 2$. It has 15 states, not counting the halting state. It has 15 left-moving rules, 14 right-moving rules, and 1 halting rule. In terms of PCP pairs, left-moving rules are more costly than right-moving rules, but we have the freedom to swap left-moving with right-moving rules by flipping the Turing machine's tape. We have to switch to the nonstandard convention of starting the Turing machine over the rightmost input character, but this is easily accomplished in the construction of the starting pair (v_1, w_1). Thus, with 14 left-moving rules and 16 right-moving and halting rules, we get $3 \times 2 + 3 + 14 \times 2 + 16 = 53$ pairs, plus the starting pair (v_1, w_1) that encodes the input.

In some sense, this can be regarded as a *universal* MPCP: all pairs except the starting pair are fixed.

We can now establish a weaker version of Theorem 2, with matrices of dimension 12×12 instead of 11×11.

Proposition 5. *There is a fixed set of 53 stochastic matrices $\mathcal{M}''''$ of dimension 12×12, 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 $\pi \in \mathbb{Q}^{12}$ whose entries are binary fractions, 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 cut-point $\lambda = \frac{1}{4}$ nonempty?

Proof. We specialize the proof of Proposition 4 to the current setting. The important point, as discussed above and shown in Figure 9, is that the matrices in $\mathcal{M}$ depend only on the word pairs that reflect the rules of the universal Turing machine $U_{15,2}$, which are fixed, and we have already calculated that there are 53 of these matrices.

We must not forget that the symbols of the alphabet $\Gamma \cup Q \cup \{\#, H\}$, in which the word pairs (v_i, w_i) of the MPCP are written, have to be encoded somehow into the binary alphabet $\{0, 1\}$ in order to define the matrices of the PFA, and we have to ensure that the codes of v_1 and w_1 end with 1, for example by letting the code for # end with 1.

There is one technicality that needs to be resolved. The quantity γ was required to be a common divisor of the matrix entries, and it depends on the maximum lengths $|v_i|$ and $|w_i|$ of the input strings. However, the strings v_1 and w_1 depend on the input tape, and thus, their lengths $|v_1|$ and $|w_1|$ cannot be bounded in advance. (The remaining strings depend only on the Turing machine.) 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 π) with a sufficiently small value of γ, namely $\gamma_1 = 4^{-\max\{|v_1|,|w_1|\}}$, where the lengths $|v_1|$ and $|w_1|$ are measured in the binary encoding. The other transitions from the state q_A can be carried out with the fixed value γ that is sufficient for those entries. Table 1 shows the starting distribution π resulting from this construction. Since π is allowed to depend on the input, we have solved the problem.

We have now established the existence of 53 fixed matrices $\mathcal{M}$ and a finishing 0-1-vector f for which the decision problem of Proposition 5 is undecidable.

[6] see `http://mural.maynoothuniversity.ie/12416/`, with incorrect page numbers

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, Φ_0)	$\frac{1}{8}(1-0.v_1)^2$	$\{\Psi_0, \Psi_1\}$	$\frac{1}{4}(1-0.w_1)0.w_1$
(Φ_0, Ψ_1)	$\frac{1}{4}(1-0.v_1)0.w_1$	$\{\Phi_0, \Phi_1\}$	$\frac{1}{4}(1-0.v_1)0.v_1$	(Ψ_1, Ψ_1)	$\frac{1}{8}(0.w_1)^2$
(Φ_1, Ψ_0)	$\frac{1}{4}\cdot 0.v_1(1-0.w_1)$	(Φ_1, Φ_1)	$\frac{1}{8}(0.v_1)^2$	q_A	$\frac{1}{16}\gamma_1$
(Φ_1, Ψ_1)	$\frac{1}{4}\cdot 0.v_1\cdot 0.w_1$	(Ψ_0, Ψ_0)	$\frac{1}{8}(1-0.w_1)^2$	q_R	$\frac{1}{2}-\frac{1}{16}\gamma_1$

Table 1: Starting probabilities π

6.4 An efficient code

In order to say something about the entries of these matrices, we have to be more specific about the way how the alphabet $\Gamma \cup Q \cup \{\#, H\}$ is encoded. The words v_i and w_i that come from the Turing machine rules are actually quite short: they have at most 3 letters; more precisely, they consist of at most one "state" symbol from $Q \cup \{H\}$, plus at most two letters from the tape alphabet $\Gamma \cup \{\#\}$. The Turing machine $U_{15,2}$ has $|Q| = 15$ states and a tape alphabet of size $|\Gamma| = 2$.

In this situation, a variable-length code is more efficient than a fixed-length code. We can use 5-letter codes of the form `0****` for the 15 states plus the halting state H. This leaves the 3-letter codes `1**` for the 3 symbols $\Gamma \cup \{\#\}$, leading to word lengths bounded by $5 + 3 + 3 = 11$. In the binary automaton, the transition probabilities are therefore multiples of 2^{-11}. Since each box carries out two binary automata simultaneously, the transition probabilities are multiples of 4^{-11}. □

With a weak inequality like $\geq \frac{1}{2}$ instead of $> \frac{1}{4}$ as acceptance criterion, we don't need the extra states q_A and q_R, and the size of the matrices for which Proposition 5 holds can be reduced to 10×10.

As mentioned after Proposition 2, the cut-point can be changed to a different value; in that case, the constraint that the input distribution π consists of binary fractions must be abandoned. Since the change only affects the very first transition, the fixed matrix set $\mathcal{M}$ remains unchanged.

The above variable-length code seems to be pretty efficient, but it wastes one of the four code-words `1**`. By looking at the actual rules of the machine $U_{15,2}$ and fiddling with the code, it might be possible to improve the power 22 in the denominator of the binary fractions.

6.5 Example matrices

For illustration, we compute some matrices of $\mathcal{M}''''$ explicitly. We use the binary code $\# \doteq \texttt{101}$, $\sqcup \doteq \texttt{100}$. The copying pair $(\sqcup, \sqcup) \doteq (\texttt{100}, \texttt{100})$ is then translated into the block

diagonal matrix

$$M_{(\sqcup,\sqcup)} = \begin{pmatrix} \frac{1}{64}\begin{pmatrix} 16 & 16 & 16 & 16 \\ 12 & 20 & 12 & 20 \\ 12 & 12 & 20 & 20 \\ 9 & 15 & 15 & 25 \end{pmatrix} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{64}\begin{pmatrix} 16 & 32 & 16 \\ 12 & 32 & 20 \\ 9 & 30 & 25 \end{pmatrix} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{64}\begin{pmatrix} 16 & 32 & 16 \\ 12 & 32 & 20 \\ 9 & 30 & 25 \end{pmatrix} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2^{22}} & 1-\frac{1}{2^{22}} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

where the rows and columns correspond to the states as they are ordered in Table 1. (Since $v_i = w_i$ in this case, we have a chance to compare the straightforward 4×4 construction of the probability ϕ^2 (the upper left block) with the condensed representation with 3 states, in the two middle blocks.)

Let us look at the erasing pair $(\mathtt{\#}\sqcup, \mathtt{\#})$. Here we have to take into account that we are dealing with the *reversed* MPCP, and therefore the strings are actually $(\sqcup\mathtt{\#}, \mathtt{\#}) \doteq (\mathtt{100\,101}, \mathtt{101})$. (The code-words don't have to be reversed.) With these data, the matrix $M_{(\mathtt{\#}\sqcup,\mathtt{\#})}$ looks as follows:

$$\begin{pmatrix} \frac{1}{512}\begin{pmatrix} 81 & 135 & 111 & 185 \\ 54 & 162 & 74 & 222 \\ 78 & 130 & 114 & 190 \\ 52 & 156 & 76 & 228 \end{pmatrix} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4096}\begin{pmatrix} 729 & 1998 & 1369 \\ 702 & 1988 & 1406 \\ 676 & 1976 & 1444 \end{pmatrix} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{64}\begin{pmatrix} 9 & 30 & 25 \\ 6 & 28 & 30 \\ 4 & 24 & 36 \end{pmatrix} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2^{22}} & 1-\frac{1}{2^{22}} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Finally, as our most elaborate example, we consider a left-moving rule of the Turing machine $U_{15,2}$ from [16]: $(q_9, \sqcup, \sqcup, L, q_1)$. This was originally a right-moving rule, but has been converted into a left-moving rule by flipping the tape. It produces two word pairs, since $|\Gamma| = 2$. One of these pairs is $(\mathtt{b}q_9\sqcup, q_1\mathtt{b}\sqcup)$, where $\mathtt{b}$ is the other letter of the tape alphabet besides $\sqcup$. Coding this letter as $\mathtt{b} \doteq \mathtt{110}$ and the states in the most straightforward way as $q_1 \doteq \mathtt{00001}$ and $q_9 \doteq \mathtt{01001}$, we get, after reversal, the binary word pair $(\sqcup q_9\mathtt{b}, \sqcup\mathtt{b}q_1) \doteq (\mathtt{100\,01001\,110}, \mathtt{100\,110\,00001})$ and the following transition matrix:

$$\begin{pmatrix} \frac{1}{2^{22}}\begin{pmatrix} 786126 & 1151282 & 915762 & 1341134 \\ 785180 & 1152228 & 914660 & 1342236 \\ 785295 & 1150065 & 916593 & 1342351 \\ 784350 & 1151010 & 915490 & 1343454 \end{pmatrix} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2^{22}}\begin{pmatrix} 894916 & 2084984 & 1214404 \\ 893970 & 2084828 & 1215506 \\ 893025 & 2084670 & 1216609 \end{pmatrix} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2^{22}}\begin{pmatrix} 690561 & 2022654 & 1481089 \\ 689730 & 2022268 & 1482306 \\ 688900 & 2021880 & 1483524 \end{pmatrix} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2^{22}} & 1-\frac{1}{2^{22}} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

7 Output values instead of a set of accepting states

We will now fix even the starting distribution π. However, this makes sense only if we allow more general values f_q. In the classic model of a PFA, f is a 0-1-vector. Once the input has been read and all probabilistic transitions have been made, acceptance is a yes/no decision: the state that has been reached is either accepting or not.

In the expression for the acceptance probability in (1), π and f play symmetric roles, and hence is makes sense to relax the 0-1-restriction on the values f_q.

We can think of f_q as a probability in a final acceptance decision, after the input has been read. Another possibility is that f_q represents a *prize* or *value* that that is gained when the process stops in state q, as in game theory. Then f_q does not need to be restricted to the interval $[0, 1]$. In this view, instead of the acceptance probability, we compute the *expected* gain (or loss) of the automaton. Following Carl Page [18], who was the first to consider this generalization, we call f the *output vector* and f_q the *output values*. Mathematically, it make sense to take the outputs even from some (complex) vector space (quantum automata?).

In our results, the values f_q are restricted to $[0, 1]$, and in fact, they have an interpretation as probabilities.

Turakainen [29] considered the most general setting, allowing arbitrary positive or negative entries for matrices $M \in \mathcal{M}$ and the vectors π and f. He showed that, the condition (1) with these more general data does not define a more general class of languages than a classic PFA, see also [7, §3.3.2, pp. 120–126] or [20, Proposition 1.1 in Section IIIB, p. 153].

7.1 Saving one more state by maintaining four binary variables

The PFA of Figure 7c mixes the PFA's for the three terms $\phi\psi$, $1 - \phi^2$, and $1 - \psi^2$ by deciding *in advance* which sub-automaton they should enter. As an alternative approach when arbitrary output values f_q are allowed, we can delay this decision to the end, when we decide whether to (probabilistically) accept the input, and this will allow us to further reduce the number of states by one.

The idea is to maintain four independent binary state variables $\Phi', \Phi'', \Psi', \Psi''$ throughout the process. Such a pool of variables is sufficient for any of the terms $\phi\psi$, $1 - \phi^2$, and $1 - \psi^2$. This would normally require $2^4 = 16$ states. As discussed above, the combinations (Φ'_0, Φ''_1) and (Φ'_1, Φ''_0) need not be distinguished and can be merged into one state, denoted by $\{\Phi_0, \Phi_1\}$, and similarly for the Ψ variables. Thus, the overall number of states is reduced from 16 to $3 \times 3 = 9$ combinations q, one less than the 10 states of Figure 7c. We have to set the nine entries $\hat{f}_q$ of the output vector to the following values, which we can conveniently arrange in tabular form:

	(Ψ'_0, Ψ''_0)	$\{\Psi_0, \Psi_1\}$	(Ψ'_1, Ψ''_1)
(Φ'_0, Φ''_0)	1/2	1/2	1/4
$\{\Phi_0, \Phi_1\}$	1/2	5/8	1/2
(Φ'_1, Φ''_1)	1/4	1/2	1/2

(10)

These output values result from the contributions to the three terms $\frac{1}{2}\phi\psi$, $\frac{1}{4}(1 - \phi^2)$, $\frac{1}{4}(1 - \psi^2)$ of the overall acceptance probability as shown below, where the states are arranged in the same matrix form as in (10):

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1/4 & 1/2 \\ 0 & 1/2 & 1 \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \frac{1}{4}\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1/2 & 1/2 & 1/4 \\ 1/2 & 5/8 & 1/2 \\ 1/4 & 1/2 & 1/2 \end{pmatrix}$$

The fractional values in the first matrix appear for the following reason. We have reduced the states for generating the acceptance probability ϕ^2 from 4 to 3 by merging two states into one. Thus, when the PFA is, for example, in the state $(\{\Phi_0, \Phi_1\}, (\Psi'_1, \Psi''_1))$, it is "really" in one of the two states $(\Phi'_0, \Phi''_1, \Psi'_1, \Psi''_1)$ or $(\Phi'_1, \Phi''_0, \Psi'_1, \Psi''_1)$, each with a share of 50 %. If we consider the product $\phi\psi$ as built, say, from the conjunction (Φ'_1, Ψ'_1), only the second of these two states should lead to acceptance, and therefore we get the fractional output value 1/2.

We can change the cut-point (for the original automaton, without the extra states q_A and q_R) from $\lambda = 1/2$ to any rational value λ strictly between 0 and 1 by modifying the output values $\hat{f}_q$ in (10): By scaling both $\hat{f}$ and λ down by the same factor, λ can be brought arbitrarily close to 0. On the other hand, by applying the transformation $x \mapsto 1 - \alpha(1 - x)$ for some constant $0 < \alpha \leq 1$ to $\hat{f}$ and λ, the cut-point λ can be moved arbitrarily close to 1 (see also [20, Proposition 1.4 of Section IIIB, p. 153]).

7.2 All transitions have positive probability.

By using an appropriate binary code, we can ensure that all transition matrices are strictly positive. Rabin calls such PFA's *actual automata* and studies their properties [21, Sections IX–XII, p. 242–245], see also [7, §3.2.3, pp. 115–118]. One can easily check that the transition matrix $B(u)$ for the binary automaton is positive except when the string u consists only of zeros or only of ones.

With only 3 symbols $\Gamma \cup \{\#\}$ using the 4 code-words `1**`, we can avoid the all-ones code-word `111` (as in the code used for the examples in Section 6.5).

A state symbol other than H never appears alone in a word v_i or w_i. Thus, we can use the code-word `00000` for one of the original states, and thereby ensure that the transition matrices $B(v_i)$ and $B(w_i)$ are always positive. As discussed earlier, their entries are multiples of 2^{-11}. The entries of the 3×3 transition matrix (9) are sums and products of entries of the 2×2 matrices $B(v_i)$ or $B(w_i)$, respectively, and are therefore positive multiples of 2^{-22}. Each entry of the 9×9 transition matrix is obtained by multiplying appropriate entries of the two 3×3 matrices, and is hence a positive multiple of 2^{-44}. (To say it more concisely, the matrix is the Kronecker product, or tensor product, of the two 3×3 matrices.)

7.3 Fixing everything except the output vector, proof of Theorem 4

For this version, we use the original (unreversed) MPCP, where the *first* word pair in the solution, and hence the *last* matrix in the matrix product, is fixed.

We can save a matrix by observing that the *last* word pair in the PCP is also known: It is the finishing pair $(H\#\#, \#)$, and like the starting pair, this pair is used nowhere else. (This is the only pair, beside the starting pair, that has a different number of #'s in the two components, and it is the only possibility how the word $v_1 v_{a_2} \dots v_{a_n}$ can catch up with the word $w_1 w_{a_2} \dots w_{a_n}$.)

For clarity, we formulate the (unreversed) Doubly-Modified Post Correspondence Problem (2MPCP), with two special pairs: a starting pair (v_1, w_1) and a finishing pair (v_2, w_2):

> 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 $\{0, 1\}$ such that v_2 and w_2 end with a 1. The problem is to decide if there is a sequence $a_2, \dots, a_{m-1}$ of indices $a_i \in \{3, \dots, k\}$ such that
>
> $$v_1 v_{a_2} v_{a_3} \dots v_{a_{m-1}} v_2 = w_1 w_{a_2} w_{a_3} \dots w_{a_{m-1}} w_2 \ .$$

The PFA starts deterministically in the state $(\Phi'_0, \Phi''_0, \Psi'_0, \Psi''_0)$. Thus, the 2MPCP has a solution if and only if the following inequality can be solved:

$$e_1^T M^2 M^{a_{m-1}} \dots M^{a_2} M^1 \hat{f} \geq \tfrac{1}{2},$$

where $\hat{f}$ is the output vector defined in (10). For clarity, we will from now on use superscripts like M^i or $M^{(v_i,w_i)}$ for the matrices that are associated to the word pairs (v_i, w_i), in order to distinguish this from the notation M_j in the theorem below, where they are numbered in the order in which they are used in the matrix product. The matrix $M^2 = M^{(H\#\#,\#)}$ comes from the finishing pair $(H\#\#, \#)$ and is fixed, and M^1 depends on the input tape u of the Turing machine. With the substitutions $e_1^T M^2 =: \pi^T$ and $M^1 \hat{f} =: f$, we can remove M^2 from the set of matrices $\mathcal{M}$, and this directly leads to part (b) of the following theorem:

Theorem 4.

(a) *There is a fixed set $\mathcal{M}''$ of 52 stochastic matrices of size 11×11 and a fixed starting distribution π, all of whose entries are multiples of $1/2^{45}$, for which the following question is undecidable*:

Given a vector $f \in \mathbb{Q}^{11}$ whose entries are binary fractions from the interval $[0, 1]$, is there a product $M_1 M_2 \dots M_m$, with $M_j \in \mathcal{M}'$ for all $j = 1, \dots, m$, such that

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

(b) *There is a fixed set $\mathcal{M}'''$ of 52 positive stochastic matrices of size 9×9 and a fixed starting distribution π, all with positive entries that are multiples of $1/2^{44}$, for which the following question is undecidable*:

Given a vector $f \in \mathbb{Q}^9$ whose entries are binary fractions from the interval $[\frac{1}{4}, \frac{5}{8}]$, is there a product $M_1 M_2 \dots M_m$, with $M_j \in \mathcal{M}'''$ for all $j = 1, \dots, m$, with

$$\pi^T M_1 M_2 \dots M_m f \geq \tfrac{1}{2} \ ?$$

Proof. For part (b), everything has been said except for observing that the entries of $f = M^1 \hat{f}$ are in the interval $[\frac{1}{4}, \frac{5}{8}]$ because M^1 is a stochastic matrix and the entries of $\hat{f}$ are in that interval.

For part (a), we add the same two states q_A and q_R as in Figure 6b and Figure 7c, with $\gamma = 2^{-44}$. Initially, we enter the original starting state $(\Phi'_0, \Phi''_0, \Psi'_0, \Psi''_0)$ and the state q_A each with probability $\frac{1}{2}$. Denoting by π_0 the corresponding vector with two $\frac{1}{2}$ entries, the initial distribution π is then defined as $\pi_0^T M^2 =: \pi^T$, and its entries are multiples of $\frac{1}{2^{45}}$.

The matrix M^1 is constructed from the starting pair (v_1, w_1), and it uses the value $\gamma_1 = 1/16^{\max\{|v_1|,|w_1|\}}$, where $|v_1|$ and $|w_1|$ are the lengths after the binary encoding.

The extra output values are defined as $\hat{f}_{q_A} = 1/16$ and $\hat{f}_{q_R} = 0$. Since the remaining output values in $\hat{f}$ are multiples of $1/8$, the value $\hat{f}_{q_A} = 1/16$ is small enough to ensure that it does not turn an acceptance probability $< \frac{1}{4}$ into a probability $> \frac{1}{4}$. □

We mention that in both cases, the solution is unique if it exists. This holds because the Turing machine is deterministic and, in addition to the starting pair, also the finishing pair (v_2, w_2) is fixed in the 2MPCP. (In the normal (M)PCP, a solution could be extended by appending arbitrary copying pairs.)

To give a concrete example, here is the transition matrix $M^{(\#_\sqcup,\#)} \in \mathcal{M}''$ for the erasing pair $(\#_\sqcup, \#) \doteq (\mathtt{101\,100}, \mathtt{101})$:

$$\frac{1}{2^{18}}\left(\begin{array}{ccc|ccc|ccc|c|c}
3600 & 12000 & 10000 & 15840 & 52800 & 44000 & 17424 & 58080 & 48400 & 0 & 0 \\
2400 & 11200 & 12000 & 10560 & 49280 & 52800 & 11616 & 54208 & 58080 & 0 & 0 \\
1600 & 9600 & 14400 & 7040 & 42240 & 63360 & 7744 & 46464 & 69696 & 0 & 0 \\
\hline
3420 & 11400 & 9500 & 15624 & 52080 & 43400 & 17820 & 59400 & 49500 & 0 & 0 \\
2280 & 10640 & 11400 & 10416 & 48608 & 52080 & 11880 & 55440 & 59400 & 0 & 0 \\
1520 & 9120 & 13680 & 6944 & 41664 & 62496 & 7920 & 47520 & 71280 & 0 & 0 \\
\hline
3249 & 10830 & 9025 & 15390 & 51300 & 42750 & 18225 & 60750 & 50625 & 0 & 0 \\
2166 & 10108 & 10830 & 10260 & 47880 & 51300 & 12150 & 56700 & 60750 & 0 & 0 \\
1444 & 8664 & 12996 & 6840 & 41040 & 61560 & 8100 & 48600 & 72900 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2^{26}} & 2^{18}-\frac{1}{2^{26}} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2^{18}
\end{array}\right)$$

(If the words v_i and w_i weren't reversed between the MPCP to the RMPCP, the upper left 9×9 block would be the Kronecker product of the two middle 3×3 blocks in the corresponding 12×12 matrix $M_{(\#_\sqcup,\#)} \in \mathcal{M}''''$ of Proposition 5 for this pair, which was shown on p. 24 in Section 6.5. If we substitute this Kronecker product, we get the matrix $M^{(\sqcup\#,\#)} \in \mathcal{M}''''$ of the opposite erasing pair.)

7.4 Eliminating the output vector, proof of Theorem 2

We will transfer these results to the classic setting with a *set* of accepting states instead of an output vector f. The characteristic vector f will be fixed, and the input should come through the starting distribution π. Thus, the Turing machine input should be coded, via the first matrix in the matrix product, into the starting distribution π, and hence we use the *reversed* MPCP again, as in Sections 5.7 and 6, where the *last* word pair in the PCP is fixed.

We will use two methods to convert a PFA with an output vector f with positive entries to into one with a characteristic vector f. The first method doubles the number of states, but it does not change the recognized language, and it maintains positivity.[7] This is formulated as part (b) in the following theorem. As an alternative, will take the liberty to change the language by adding a symbol to the end of each word. This works without adding extra states beyond the states q_A and q_R that are already there, and it will lead to part (a) of the following theorem.

Theorem 2.

(a) *There is a fixed set $\mathcal{M}$ of 53 stochastic matrices of size 11×11, all of whose entries are multiples of $1/2^{48}$, for which the following question is undecidable*:

Given a probability distribution $\pi \in \mathbb{Q}^{11}$ whose entries are binary fractions, is there a product $M_1 M_2 \dots M_m$, with $M_j \in \mathcal{M}$ for all $j = 1, \dots, m$, such that

$$\pi^T M_1 M_2 \dots M_m e_1 > \tfrac{1}{4}$$

In other words, is the language recognized by the PFA with starting distribution π and cut-point $\lambda = \frac{1}{4}$ nonempty?

(b) *There is a fixed set $\mathcal{M}'$ of 52 stochastic matrices of size 18×18 with positive entries that are multiples of $1/2^{47}$, and a fixed vector $f \in \{0,1\}^{18}$, for which the following question is undecidable*:

[7] There is a method in the the literature with the same effect, but it *squares* the number of states, see [29, proof of Theorem 1, p. 308] or [7, Step V, pp. 123–124].

Given a probability distribution $\pi \in \mathbb{Q}^{18}$ whose entries are positive binary fractions, is there a product $M_1 M_2 \dots M_m$, with $M_j \in \mathcal{M}''$ for all $j = 1, \dots, m$, with

$$\pi^T M_1 M_2 \dots M_m f \geq \tfrac{1}{2} \; ?$$

Proof. (a) We start with the set $\mathcal{M}'' \cup \{M^2\}$ of 53 matrices of Theorem 4a before the matrix $M^2 = M^{(H\#\#,\#)}$ for the finishing pair was removed. We add an extra "final" transition matrix M^∞ to these 53 matrices. As mentioned, this will change the language by adding the corresponding symbol to the end of each word, but it will not affect the emptiness question.

We declare q_A to be the unique accepting state. In the transition M^∞, each state q goes to q_A with probability $\hat{f}_q$, and to q_R with the complementary probability $1 - \hat{f}_q$. This rule applies equally to the state q_A, which goes to itself with probability $\hat{f}_{q_A} = 1/16$, and otherwise goes to q_R. The state q_R remains an absorbing state.

If we never use the matrix M^∞ in the matrix product, the only chance of reaching q_A comes from entering q_A at the beginning and staying there, and the probability for this is negligibly small, too small to be in the language recognized by the PFA.

On the other hand, when we use the matrix M^∞, the PFA will arrive in states q_A or q_R, and afterwards it makes no sense, regarding the question of PFA emptiness, to add further matrices, because this can only reduce the probability of staying in q_A.

Thus we can assume without loss of generality that M^∞ is the last matrix in the product, and that it is used only in that position. The acceptance probability is then the same as if the output vector $\hat{f}$ had been used instead of M^∞. (Algebraically, $M^\infty e_1 = \hat{f}$, assuming that q_A is the first state and therefore $f = e_1$.)

We can save one matrix by remembering that the last word pair is always the finishing pair $(v_2, w_2) = (H\#\#, \#)$, and this is used nowhere else. We can therefore assume without loss of generality that the corresponding matrix $M^2 = M^{(H\#\#,\#)}$ is the last matrix in the product before M^∞, and this matrix is used nowhere else. Hence we can replace M^2 and M^∞ by one matrix $M^2 M^\infty$, reducing the number of matrices back to 53. Since the entries of M^∞ are multiples of $\frac{1}{16} = \frac{1}{2^4}$, the entries of the new matrix are multiples of $\frac{1}{2^{48}}$.

(b) We start with the set $\mathcal{M}'''$ of 52 positive 9×9 matrices of Theorem 4b. We think of the output values $f_q \in [\frac{1}{4}, \frac{5}{8}]$ as probabilities. If we arrive in state q after reading the input, we still have to make a random decision whether to accept the input. The idea is to generate the randomness for making this acceptance decision already *when* each symbol is read, and not *afterwards* at the end. Every state q of the original PFA comes now in two versions, q^+ and q^-. The transitions to q^+ are multiplied by f_q, and the transitions to q^- are multiplied by $1 - f_q$. The accepting states are the states q^+.

In terms of matrices, this can be expressed as follows. Let $M \in \mathcal{M}'''$ be written in column form as

$$M = \begin{pmatrix} m_1 & m_2 & \cdots & m_9 \end{pmatrix}.$$

This is converted to the following 18×18 matrix for the set $\mathcal{M}'$, arranging the states in the order $q_1^+, \dots, q_9^+, q_1^-, \dots, q_9^-$:

$$\begin{pmatrix} f_1 m_1 & f_2 m_2 & \cdots & f_9 m_9 & (1-f_1) m_1 & (1-f_2) m_2 & \cdots & (1-f_9) m_9 \\ f_1 m_1 & f_2 m_2 & \cdots & f_9 m_9 & (1-f_1) m_1 & (1-f_2) m_2 & \cdots & (1-f_9) m_9 \end{pmatrix}$$

Similarly, the starting distribution $\pi^T = (\pi_1, \pi_2, \dots, \pi_9)$ is replaced by $(f_1\pi_1, f_2\pi_2, \dots, f_9\pi_9, (1-f_1)\pi_1, (1-f_2)\pi_2, \dots, (1-f_9)\pi_9)$. The matrix consists of two equal 9×18 blocks, in accordance with the fact that the distinction between q^+ and q^- has no influence on the next transition.

As the output values $f_q \in \{\frac{1}{4}, \frac{1}{2}, \frac{5}{8}\}$ are multiples of $\frac{1}{8}$, all resulting probabilities are multiples of $\frac{1}{2^{47}}$. □

7.5 Reduction to 2 input symbols, proof of Theorem 3

We have already used the reduction to a binary alphabet in the proof of Theorem 1 (Section 4.4), but now we will look at the explicit construction.

Lemma 3. *Consider a PFA A with input alphabet Σ of size $k = |\Sigma| > 2$, and let $\tau\colon \Sigma^* \to \{\mathtt{a},\mathtt{b}\}^*$ be a coding function using the code-words $\mathtt{b}, \mathtt{ab}, \mathtt{aab}, \ldots, \mathtt{a}^{k-2}\mathtt{b}, \mathtt{a}^{k-1}$.*

Then there is a PFA A' with input alphabet $\{\mathtt{a},\mathtt{b}\}$, for which the acceptance probability of each word $\tau(u)$ is the same as for $u \in \Sigma^*$ and A. Words that are not of the form $\tau(u)$ are accepted with probability 0.

The number of states is multiplied by $k-1$ in this construction.

Proof. Suppose A has transition matrices $M_1, \ldots, M_k$ corresponding to the k input symbols. The PFA A' maintains a *decoding counter* i in the range $1 \le i \le k-1$, and in addition it maintains the state $q \in Q$ of the original PFA A. Thus, the state set is $Q' = \{1, \ldots, k-1\} \times Q$. Initially, $i = 1$ and q is chosen according to the starting distribution of A.

- If A' reads the letter $\mathtt{b}$, it changes the state q to a random new state according to M_i, and resets $i := 1$.
- If A' reads an $\mathtt{a}$ and $i < k-1$, it increments the counter: $i := i+1$.
- If A' reads an $\mathtt{a}$ and $i = k-1$, it changes the state q to a random new state according to M_k, and resets $i := 1$.

An input is accepted if $i = 1$ and q is an accepting state of A.

The transition matrices for the symbols $\mathtt{a}$ and $\mathtt{b}$ can be written in block form as

$$\begin{pmatrix} 0 & I & & & & \\ & 0 & I & & & \\ & & 0 & & & \\ & & & \ddots & & \\ & & & & 0 & I \\ M_k & & & & & 0 \end{pmatrix} \text{ and } \begin{pmatrix} M_1 & 0 & \cdots & 0 \\ M_2 & 0 & \cdots & 0 \\ M_3 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ M_{k-1} & 0 & \cdots & 0 \end{pmatrix}. \qquad \square$$

The construction works more generally for any prefix-free code. The set of states Q' will have the form $K \times Q$, where the states in K do the decoding.

Applying this to Theorem 2a, we get:

Theorem 3. *There is a PFA with 572 states, two input symbols with fixed transition matrices, all of whose entries are multiples of $1/2^{48}$, and with a single accepting state, for which the following question is undecidable*:

Given a probability distribution $\pi \in \mathbb{Q}^{572}$ whose entries are binary fractions, is the language recognized by the PFA with starting distribution π and cut-point $\lambda = \frac{1}{4}$ nonempty?

The number 572 of states is an overcount. For example, the absorbing state q_R can be left as is and need not be multiplied with 52.

If we are more ambitious, we can achieve that all matrix entries are from the set $\{0, \frac{1}{2}, 1\}$, as in Theorem 1, instead of multiples of 2^{-48}. We apply the technique from item (b) in the proof of that theorem (Section 4.4): We simply add a block of 47 padding $\mathtt{a}$'s after every code-word.

We can roughly estimate the required number of states as follows. Let $|Q| = 11$ be the number of states of the original automaton, and $k = |\Sigma| = 53$ its number of

symbols. For each combination in $Q \times \Sigma$, whenever the algorithm in Lemma 3 asks to "change the state q to a random new state according to M_i", we have to set up a binary decision tree of height 48 to determine the next state. We can think of this tree as determining which of $|Q|$ intervals $[0, c_1], (c_1, c_2], (c_2, c_3], \ldots, (c_{|Q|-1}, 1]$ contains a random number $0.x_1x_2 \ldots x_{48}$, by looking at the successive bits x_j of that number. This tree has at most $(|Q|-1) \times 47$ nodes where the outcome has not been decided: each such node lies on a root-to-leaf path to some interval endpoint c_i. In addition we need up to $47 \times |Q|$ states for the situation when the next state has been decided and the algorithm only needs to count to the end of the padding block. In total, this gives an upper bound of $(k-1)|Q| + |Q|k(|Q|-1)47 + 47|Q|$ states, which is $572 + 47 \times 11 \times (53 \times 10 + 1) = 275\,099$.

8 Alternative universal Turing machines

In the literature, some "universal" Turing machines with smaller numbers of states and symbols are proposed. We review these machines and discuss whether they could possibly be used to decrease the number of matrices in Theorems 2–4.

8.1 Watanabe, weak and semi-weak universality

A universal Turing machine U_W with 3 symbols and 7 states was published by Watanabe [31] in 1972, but I haven't been able to get hold of this paper. According to the survey [33, Fig. 1], this is a *semi-weakly* universal Turing machine. In *semi-weakly* and *weakly* universal machines, the empty parts of the tape on one or both sides of the input are initially filled with some repeating pattern instead of uniformly blank symbols. Such a repeating pattern can be easily accommodated in the translation to the MPCP by modifying the padding pair of words $(\mathtt{\#}, \sqcup\mathtt{\#}\sqcup)$.

In the worst case, the 21 rules contain only one halting rule and the remaining 20 rules are balanced between left- and right-moving rules. Then, with 10 left-moving rules, 1 halting rule, 10 right-moving rules, and $|\Gamma| = 3$, we get $3 \times 3 + 3 + 10 \times 3 + 11 = 53$ matrices, the same number as from the machine $U_{15,2}$ of Neary and Woods. Any imbalance in the distribution of left-moving and right-moving rules would allow to reduce the number of matrices in Theorem 2a from 53 to 51 or less.

This speculative improvement depends on an assumption, which would need to be verified. According to [33, Section 3.1], Watanabe's weak machine U_W simulates other Turing machines T directly. What would be most useful for us is that the periodic pattern that initially fills the tape of U_W is a fixed pattern that is specified as part of the definition of U_W and does not depend on T or its input.

If this is the case, we can use them for our construction, where only the first (or last) pair of the PCP should depend on the input.

We could even accommodate some weaker requirement, namely that the periodic pattern depends on the Turing machine T that is being simulated, as long as it is independent of the input u to that Turing machine. In that case we could let U_W simulate a fixed (standard) universal Turing machine T_0, and then the periodic pattern would also be fixed.

8.2 Wolfram–Cook, Rule 110

There are some smaller machines, which are based on simulating a particular cellular automaton, the so-called *rule-110 automaton* of Stephen Wolfram. These machines are given in [9, Fig. 1, p. 3] and [17], see also the survey [33]. The rule-110 automaton was

shown to be universal by Cook [9], see also Wolfram [32, Section 11.8, pp. 675–689][8]. The universality of the rule-110 automaton comes from the fact that rule 110 can simulate *cyclic tag systems*. *Tag systems* are a special type of string rewriting systems, where symbols are deleted from the front of a string, and other symbols are appended to the end of a string, according to certain rules. *Cyclic tag systems* are a particularly simple variation of tag systems. Tag systems as well as cyclic tag systems are known to be universal.

The primary reason why these machines are not useful for our purposes is that the repeating patterns by which the empty parts of the tape are filled are not fixed, but depend on the tag system, in a complicated way, see [32, Note on initial conditions, p. 1116][9]. As a consequence, we don't have a fixed replacement for the padding pair $(\#, \sqcup\#\sqcup)$. Thus we cannot use them for the proof of Theorems 2 and 4, where only the first (or last) pair of the PCP should depend on the input.

There is another reason why we cannot use these machines directly: they have no provision for *halting*, or for otherwise determining some set of inputs that they accept[10]. This is natural in the context of a cellular automaton, which performs an infinite process, but a cyclic tag system, which the automaton supposedly simulates, *does* have a way of terminating, namely when the string on which it operates becomes empty. Fortunately, Cook gives a few hints about termination and about undecidable questions for the corresponding Turing machines: *Questions about their behavior, such as "Will this sequence of symbols ever appear on the tape?", are undecidable* [9, Note [7], p. 38]. More specifically, Cook mentions some particular undecidable questions for so-called *glider systems*. Some consequences of this discussion for Rule 110 are briefly touched upon in [9, Section 4.6, p. 37]: *So another specific example of an undecidable question for Rule 110 is: Given an initial middle segment, will there ever be an F?* Here, an F is a particular type of "glider", a cyclic sequence of patterns that moves at constant speed through the cellular automaton as long as it does not meet other patterns or irregularities. I suppose the presence if such a glider could be detected by the occurrence of a particular pattern $\hat{F}$ in the cellular automaton, or on the Turing machine tape. If this is indeed the case, such a criterion could be translated into a word pair $(\hat{F}, H)$ that introduces the halting symbol H, and this would lead to small undecidable instances of the MPCP.

8.3 Wolfram's 2, 3 Turing machine

An even smaller Turing machine with only 2 symbols and 3 states was proposed by Wolfram [32, Section 11.12, p. 709][11] and was shown to be universal, in a certain sense, by Smith [28][12]. As above, the proof uses cyclic tag systems, and again, this machine does not halt. Smith showed that the 2, 3 Turing machine can simulate cyclic tag systems.

[8] on-line at `https://www.wolframscience.com/nks/p675--the-rule-110-cellular-automaton/`

[9] `https://www.wolframscience.com/nks/notes-11-8--initial-conditions-for-rule-110/`

[10] Curiously, while the survey of Woods and Neary [33] carefully distinguishes *semi-weak* and *weak* universality, the fundamental characteristic whether the Turing machine has a provision for halting is treated only as an afterthought. Incidentally, in Turing's original article [30] from 1937, where Turing machines were first defined, the good machines are those that *don't* halt or go into a loop (the *circle-free* ones), because they are capable of producing an infinite sequence of zeros and ones on dedicated "output cells" on the tape, forming the fractional bits of a *computable number*.

[11] on-line at `https://www.wolframscience.com/nks/p709--universality-in-turing-machines-and-other-systems/`

[12] The reader should be warned that the journal version [28] is partly incomprehensible, since the vertical alignment in the tabular presentation of the Turing machines has been destroyed in the typesetting process. Understandable versions can be found elsewhere on the web, see for example `https://www.wolframscience.com/prizes/tm23/TM23Proof.pdf`.

Unfortunately, it is not addressed at all what happens when the operation of the simulated tag system terminates. Hypothesizing that such a halting computation would lead to some repeating cycle of movements of the Turing machine, such a configuration could be recognized by the occurrence of a certain pattern on the tape, and an appropriate pair (v_i, w_i) could be created that introduces the halting state H.

9 Outlook

9.1 Equality testing

For the reader who has well digested the ideas in the different proofs of PFA emptiness undecidability, it is an instructive exercise to see how Nasu and Honda's method of testing acceptance probabilities for equality by formula (8) (Section 5.3) would apply to the Equality Checker problem of Section 4.1 for the string $\mathtt{a}^i\mathtt{b}^j\mathtt{\#}$: It is straightforward to set up a PFA with two states that accepts $\mathtt{a}^i\mathtt{b}^j\mathtt{\#}$ with probability $\phi = 1/2^i$, and another that accepts it with probability $\psi = 1/2^j$.

The construction of Figure 6a, translated into the language of the Equality Checker, leads to the following algorithm: 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

$$\tfrac{1}{4}(1 - 1/4^i) + \tfrac{1}{4}(1 - 1/4^j) + \tfrac{1}{2} \cdot 1/2^{i+j} = \tfrac{1}{2} - \tfrac{1}{4}(1/2^i - 1/2^j)^2,$$

which reaches its maximum value $1/2$ if and only if $i = j$. (To save coin flips, one would of course take the decision between the three branches in advance.)

We can notice a sharp contrast between the character of the outcome in the two cases. The equality test by formula (8) capitalizes on the capability of a PFA to detect a tiny fluctuation of the acceptance probability above or below the cut-point. On the other hand, the Equality Checker, as illustrated in Figure 2, almost always leaves the answer "Undecided", but if it makes a decision, the probabilities of the two outcomes, in case of inequality, differ by several orders of magnitude.

9.2 Shortcutting the reduction

Figure 10 illustrates the chain of reductions leading to the two undecidability proofs of PFA emptiness. In both cases, undecidability ultimately stems from the Halting Problem for Turing machines.

The earliest universal Turing machines simulate general Turing machines directly, as indicated by the dotted arrow. However, the smallest universal Turing machines known today do not simulate Turing machines directly, but they simulate arbitrary tag systems (Section 8.2). In particular, this is true for the machine $U_{15,2}$ of Neary and Woods, on which Theorems 2–4 are based. This has the somewhat curious effect that our construction of specialized undecidable instances of PFA emptiness proceeds by reduction from tag systems, which operate on *strings*, via universal Turing *machines*, to another problem on *strings*: the PCP. It would seem natural to shortcut this detour and try to go from tag systems to the PCP directly. Tag systems are universal in the sense that every Turing machine can be simulated by some tag system. What might be useful for us is a tag system with a (small) fixed set of rules for which the halting problem is undecidable,

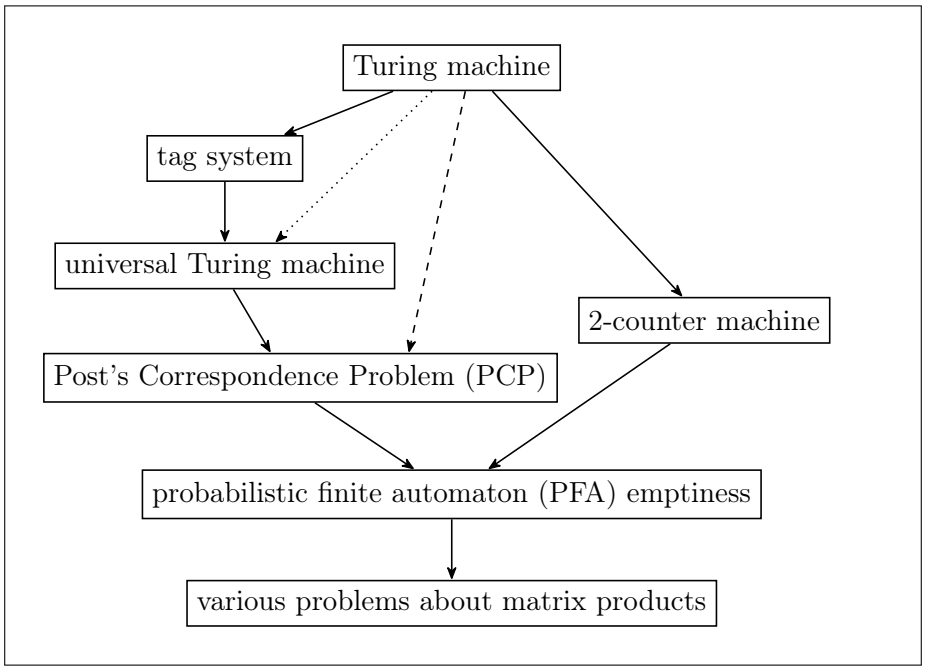

Figure 10: Reductions for proving undecidability. The four topmost boxes concern the *Halting Problem* for the respective systems. The dashed arrow represents the reduction that is sufficient for the plain undecidability result of PFA emptiness (Propositions 1–4), without the specializations of Theorems 2–4.

depending on the starting word. It seems that such tag systems have not been studied in their own right. (Of course, one can take a tag system that simulates a universal Turing machine, but that would add another round to the detour.)

9.3 Strictly positive matrices

We have established a couple of undecidability results where all transition probabilities are constrained to be positive (Theorems 2b and 4b), but only for the case when acceptance is by the criterion $\geq \lambda$. We don't know whether this can be achieved with the classic acceptance criterion $> \lambda$.

9.4 The number of states

It is a natural question to ask for the smallest number of states for which the PFA emptiness problem is undecidable. Depending on the precise technical formulation of the question, we could reduce the number of states to 11 (Theorem 4a or to 9 (Theorem 4b). Even without insisting on a fixed set $\mathcal{M}$ of transition matrices, this is the best upper bound that we have. We are not aware of any lower bound, except for the trivial observation that the emptiness question can be decided for one-state PFA's. As the example of binary automata shows, already a 2-state PFA can be very powerful.

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] Vuong Bui. Growth of bilinear maps. *Linear Algebra and its Applications*, 624:198–213, 2021. arXiv:2005.09540, doi:10.1016/J.LAA.2021.04.010.

[3] Vuong Bui. Growth of bilinear maps II: Bounds and orders, 2021. arXiv:2110.15060.

[4] Vuong Bui. Growth of bilinear maps III: Decidability, 2022. arXiv:2201.09850.

[5] Vuong Bui. On the joint spectral radius of nonnegative matrices. *Linear Algebra and its Applications*, 654:89–101, 2022.

[6] Vuong Bui. *Growth of bilinear maps*. PhD thesis, Freie Universität Berlin, Institut für Informatik, 2023. (submitted).

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

[8] 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.

[9] Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15:1–40, 2004. doi:10.25088/ComplexSystems.15.1.1[13] URL: http://www.complex-systems.com/abstracts/v15_i01_a01.html.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] Turlough Neary. Undecidability in binary tag systems and the Post correspondence problem for five pairs of words. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 649–661, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2015.649.

[13] As of August 7, 2023, the DOI of this publication did not work.

[16] Turlough Neary and Damien Woods. Four small universal Turing machines. *Fundamenta Informaticae*, 91:123–144, 2009. doi:10.3233/FI-2009-0036.

[17] Turlough Neary and Damien Woods. Small weakly universal Turing machines. In Mirosław Kutyłowski, Witold Charatonik, and Maciej Gębala, editors, *Fundamentals of Computation Theory*, volume 5699 of *Lecture Notes in Computer Science*, pages 262–273, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. arXiv:0707.4489, doi:10.1007/978-3-642-03409-1_24.

[18] Carl V. Page. Equivalences between probabilistic and deterministic sequential machines. *Information and Control*, 9(5):469–520, 1966. doi:10.1016/S0019-9958(66)80012-8.

[19] A. Paz. Some aspects of probabilistic automata. *Information and Control*, 9(1):26–60, 1966. doi:10.1016/S0019-9958(66)90092-1.

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

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

[22] Matthieu Rosenfeld. The growth rate over trees of any family of sets defined by a monadic second order formula is semi-computable. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 776–795. SIAM, 2021. doi:10.1137/1.9781611976465.49.

[23] Matthieu Rosenfeld. It is undecidable whether the growth rate of a given bilinear system is 1. *Linear Algebra and its Applications*, 651:131–143, 2022. arXiv:2201.07630, doi:10.1016/j.laa.2022.06.022.

[24] Günter Rote. The maximum number of minimal dominating sets in a tree. In Timothy Chan, editor, *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA19), San Diego*, pages 1201–1214. SIAM, 2019. doi:10.1137/1.9781611975482.73.

[25] M. P. Schutzenberger. Certain elementary families of automata. In Jerome Fox, editor, *Mathematical Theory of Automata*, volume 12 of *Microwave Research Symposia Series*, pages 139–153. Polytechnic Press of the Polytechnic Institute of Brooklyn, Brooklyn, N.Y., 1963.

[26] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

[27] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[28] Alex Smith. Universality of Wolfram's 2, 3 Turing machine. *Complex Systems*, 29:1–44, 2020. doi:10.25088/ComplexSystems.29.1.1.

[29] Paavo Turakainen. Generalized automata and stochastic languages. *Proc. Amer. Math. Soc.*, 21:303–309, 1969. doi:10.1090/S0002-9939-1969-0242596-1.

[30] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. doi:10.1112/plms/s2-42.1.230.

[31] Shigeru Watanabe. 4-symbol 5-state universal Turing machine. *Information Processing Society of Japan Magazine*, 13(9):588–592, 1972.

[32] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002. URL: https://www.wolframscience.com/nks/.

[33] Damien Woods and Turlough Neary. The complexity of small universal Turing machines: A survey. *Theoretical Computer Science*, 410(4):443–450, 2009. Computational Paradigms from Nature. arXiv:1110.2230, doi:10.1016/j.tcs.2008.09.051.

A The original Nasu–Honda proof in a nutshell

For comparison, we summarize the proof originally given by Nasu and Honda in [14] and reproduced in the monograph of Paz [20, Chapter IIIB, Section 6]. The presentation of Paz quite faithful to the original, to the point of using identical notations for many notions. It is more condensed, slightly simplified, but defaced by the occasional typographic or editing error, such as a mysterious passage in a proof that should obviously belong to another proof.

The notation here is changed and adapted to our paper. All references to [20] are to Chapter IIIB. (Items in that book are numbered separately in each chapter.)

Post's Correspondence Problem. The PCP with k word pairs (v_i, w_i) over $\{0, 1\}$ is represented by the language $L(v, w)$ of words of the form

$$0^{a_m} 1 \dots 0^{a_2} 1 0^{a_1} 1 + v_{a_1} v_{a_2} \dots v_{a_m} X w_{b_n}^R \dots w_{b_2}^R w_{b_1}^R + 1 0^{b_1} 1 0^{b_2} \dots 1 0^{b_n}$$

for sequences $a_1 \dots a_m$ and $b_1 \dots b_n$ with $a_i, b_j \in \{1, \dots, k\}$, where the superscript ^{R} denotes reversal, see [14, p. 265, before Lemma 16], [20, Lemma 6.13, p. 189]. This language is intersected with the set $L_s = \{\, y + z X z^R + y^R \mid y, z \in \{0, 1\}^* \,\}$ of palindromes with central symbol X and two occurrences of the separator symbol "+" [14, p. 265, Lemma 15]. The intersection $L(v, w) \cap L_s$ is well-known to represent the PCP solutions [14, p. 270, Lemma 20], because the palindrome property ensures that both the index sequences and the produced words match between the two sides. Its emptiness (apart from the single word +X+) is therefore undecidable [20, Lemma 6.16, p. 190].

Actually, the alphabet of these languages is $\Sigma = \{0, 1, +, X, \bar{0}, \bar{1}, \bar{+}\}$, because, for technical reasons, every symbol σ in the right half, after the X, is replaced by another version, its "complement symbol" $\bar{\sigma}$.

Rational automata, P-sets, and E-sets.

Definition 2 ([14, Definition 17, p. 259])**.** *A* rational PFA *is a PFA where all components of π, f, and the transition matrices are rational numbers.*

An E-set *is the set of words u with $\phi(u) = \psi(u)$, where $\phi(u)$ and $\psi(u)$ are the acceptance probabilities of two rational PFA's.*

A P-set *is a language recognized by a rational PFA with some rational cut-point λ, or in other words, the set of words u with $\phi(u) > \lambda$.*

We have already seen in Section 5.3 that every E-set is a P-set [14, Lemma 11, p.261], [20, Corollary 6.4, p. 183].[14]

Deterministic linear languages. The goal is to show that $L(v,w)$, L_s, and finally $L(v,w) \cap L_s$ are E-sets (and therefore P-sets).

This is done by appealing to the fact that a certain class of context-free languages, the so-called *deterministic linear languages* ([14, Definition 18, p. 263], [20, Definition 6.3, p. 187]) are E-sets [14, Lemma 14, pp. 263–265], [20, Lemma 6.11, pp. 187–188]. The proof relies on m-ary automata.

It is easy to see that this language class includes L_s, and also the language $L(v)$ of words of the form

$$\mathtt{0}^{a_m}\mathtt{1}\ldots\mathtt{0}^{a_2}\mathtt{1}\mathtt{0}^{a_1}\mathtt{1}\mathtt{+}v_{a_1}v_{a_2}\ldots v_{a_m},$$

which form the left half of the words of $L(v,w)$, up to the $\mathtt{X}$ symbol. Similarly, the right halves form a deterministic linear language.

Putting the two halves together to get the language $L(v,w)$ takes more effort. The proof in [14, Lemma 17, p. 265–269] is cumbersome and stretches over more than three pages. Paz formulates the argument as a separate lemma [20, Lemma 6.12, p. 188]:

Lemma 4. *If L_1 and L_2 are deterministic linear languages over disjoint alphabets Σ_1 and Σ_2 not containing the letter $\mathtt{X}$, then $L_1\mathtt{X}L_2$ is an E-set.*

The trick is to mix two m-ary automata (one for L_1 and one for L_2) into an m^2-ary automaton: one automaton uses the digits $\frac{m}{m^2}, \frac{2m}{m^2}, \ldots, \frac{(m-1)m}{m^2}$; the other automaton uses the digits $\frac{1}{m^2}, \frac{2}{m^2}, \ldots, \frac{m-1}{m^2}$. The acceptance probability is constructed as an m^2-ary number, but when it is viewed in the m-ary expansion, the digits alternate between the digits for L_1 and the digits for L_2. Equality of acceptance probabilities thus means equality for both L_1 and L_2.[15]

Intersections of E-sets. To get from $L(v,w)$ and L_s to $L(v,w) \cap L_s$, one uses the property that E-sets are closed under intersection. This is easy to prove by comparing the acceptance probabilities $\frac{1}{2} - \frac{1}{4}(\phi_1 - \psi_1)^2$ and $\frac{1}{2} + \frac{1}{4}(\phi_2 - \psi_2)^2$, which are constructed along the lines of formula (8) on p. 15 [14, Lemma 12, p. 261].[16]

Actually, this intersection property leads to a simple proof of generalized version of Lemma 4, where L_1 and L_2 can be arbitrary E-sets: The languages $L_1\mathtt{X}\Sigma_2^*$ and $\Sigma_1^*\mathtt{X}L_2$ are certainly E-sets (a PFA can simply ignore all symbols before or after the $\mathtt{X}$) and their intersection is $L_1\mathtt{X}L_2$.

---

[14] In [20, p. 182], these sets are renamed to *E-events* and *P-events*, respectively. I find this choice of terminology, which goes back to Rabin [21, p. 233] and pervades much of the literature, unfortunate, since an event is rather something whose probability is to be measured. The terms E-language and P-language might have been even more specific.

[15] The requirement of disjoint alphabets is the reason for introducing the complemented symbols $\bar{\mathtt{0}}, \bar{\mathtt{1}}, \bar{\mathtt{+}}$. Technically, it should not be necessary; any PFA could do this conversion implicitly as it reads the input from left to right.

I believe the whole proof would have been simpler with the more straightforward representation

$$\mathtt{0}^{a_1}\mathtt{1}\mathtt{0}^{a_2}\mathtt{1}\ldots\mathtt{0}^{a_m}\mathtt{1}\mathtt{+}v_{a_1}v_{a_2}\ldots v_{a_m}\mathtt{X}\mathtt{0}^{b_1}\mathtt{1}\mathtt{0}^{b_2}\mathtt{1}\ldots\mathtt{0}^{b_n}\mathtt{1}\mathtt{+}w_{b_1}w_{b_2}\ldots w_{b_n},$$

replacing L_s by the set $\{\, y\mathtt{+}z\mathtt{X}y\mathtt{+}z \mid y,z \in \{\mathtt{0},\mathtt{1}\}^* \,\}$. One could not have applied the results about deterministic linear languages, but one would get rid of all reversals in the proofs.

[16] Here, Paz [20, Proof of Lemma 6.14, p. 190], appeals to "Exercise 4.a.3", but in that exercise [20, p. 172], closure under intersection is only proved for E-sets that are defined by the condition $\phi(u) = \lambda$ for a *constant* λ.

Coding with 2 symbols. Finally, the 7-character alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{+}, \mathtt{X}, \bar{\mathtt{0}}, \bar{\mathtt{1}}, \bar{\mathtt{+}}\}$ is converted to binary by a coding function τ that uses the code-words $\mathtt{a}^i\mathtt{b}$ for $i = 1, \ldots, 7$. We have already seen in Lemma 3, for a very similar code, that this can be done while preserving the acceptance probabilities.

Paz appeals to a general statement that acceptance probabilities are preserved under GSM-mappings (mappings induced by a *generalized sequential machine*) [20, Definition 6.2 and Theorem 6.10, p. 186]. (A generalized sequential machine is a finite automaton with output. The machine is generalized in the sense that the output at any step is from Δ^*, i.e., the machine can produce several symbols or no output at all.) Nasu and Honda use their even more general statement about PGSM-mappings (mappings induced by a *probabilistic GSM* [14, Definition 13 and Theorem 6, pp. 253–255]), which are an object of study in the paper [14] and figure prominently in its title.

Nasu and Honda use this to show that $\tau(L(v, w) \cap L_s)$ is an E-set [14, Lemma 19, p.269]. Paz proves the weaker statement that $\tau(L(v, w) \cap L_s)$ is a P-set [20, Lemma 6.15, p. 190] and is therefore able to shortcut the proof.[17]

Since every E-set is a P-set (Section 5.3) and since PFA emptiness is about P-sets, the undecidability of PFA emptiness is established.

A.1 Deciding whether the recognized language is a regular language, or whether it is context-free

It is also shown to be undecidable whether a P-set is a regular language, or whether it is context-free [14, Theorem 22, p. 270], [20, Theorem 6.17, p. 190]. This can be established by the following arguments, which are not given explicitly in [14] or [20], but only through unspecific references to the theory of context-free languages.

If the PCP has a solution, then the language $L(v, w) \cap L_s$ contains some word of the form

$$y\mathtt{+}z\mathtt{X}\hat{z}\bar{\mathtt{+}}\hat{y}$$

with nonempty $y, z \in \{\mathtt{0}, \mathtt{1}\}^*$, where $\hat{y} = \bar{y}^R$ denotes simultaneous complementation and reversal. Since a PCP solution can be repeated arbitrarily, all words

$$y^i\mathtt{+}z^i\mathtt{X}\hat{z}^i\bar{\mathtt{+}}\hat{y}^i \tag{11}$$

for $i \geq 0$ are also in the language. Moreover, intersecting with the regular language $\{y\}^*\mathtt{+}\{z\}^*\mathtt{X}\{\hat{z}\}^*\bar{\mathtt{+}}\{\hat{y}\}^*$ leaves *exactly* the words of the form (11), but for such a language it is known that it is not context-free.[18] Since the intersection of a context-free language with a regular language is context-free, the language $L(v, w) \cap L_s$ cannot be context-free in this case. We conclude that the recognized language is context-free, or regular, (namely, the one-word language $\{\mathtt{+X+}\}$) if and only if the PCP has no solution.

To get the result for a binary input alphabet, this whole chain of arguments must be transferred from $L(v, w) \cap L_s$ to the *encoded* language $\tau(L(v, w) \cap L_s)$, but this does not change the situation.

There is an alternative proof. Following an exercise in Claus [7, p. 158, Aufgabe], we can derive the undecidability of testing whether a given P-set is a regular language

[17] This proof has a small technical mistake [20, Proof of Lemma 6.15, p. 190, line 8]: It claims that *one can easily construct a GSM mapping* Ψ^A to do the decoding. Some words x are not decodable for the reason that they contain more than 7 $\mathtt{a}$'s in a row or do not end with $\mathtt{b}$. For these words, it cannot be ensured that $\Psi^A(x) = e$ (the empty word, cf. [20, p. xix, and the footnote on p. 155]) as Paz claims. The mistake is of no consequence because such ill-formed strings x are filtered out in a subsequent step. In the original proof of Nasu and Honda [14, p. 269], a GSM for decoding is described explicitly.

[18] In fact, a language like $\{\, \mathtt{a}^i\mathtt{b}^i\mathtt{c}^i \mid i \geq 0 \,\}$ with just *three* blocks of equal length is the prime example of a language that is not context-free.

as an easy corollary of the emptiness question. The hint for the solution suggests to take the language L of nonempty solution sequences $a_1 a_2 \dots a_m$ of the PCP, and append some nonregular P-set $\tilde{L}$ to it. This can be done by appealing to the easy version of Lemma 4 that has been mentioned above, which applies not only to deterministic linear languages, but to any E-sets L_1 and L_2. We have already established that L is an E-set (Section 5.1). Taking some nonregular E-set $\tilde{L}$, for example the language $\{\, \mathtt{a}^i\mathtt{b}^i\mathtt{\#} \mid i \geq 0 \,\}$ of Section 9.1, we can form the E-set $L' = L\mathtt{X}\tilde{L}$. If the PCP has no solution, L' is empty and therefore regular. If the PCP has a solution, L' is not regular because $\tilde{L}$ is not regular.

This construction can be extended for context-free languages by taking the language $\tilde{L} = \{\, \mathtt{a}^i\mathtt{b}^i\mathtt{c}^i\mathtt{\#} \mid i \geq 0 \,\}$. It is not context-free, but it is the intersection of two E-sets $\{\, \mathtt{a}^i\mathtt{b}^i\mathtt{c}^n\mathtt{\#} \mid i, n \geq 0 \,\} \cap \{\, \mathtt{a}^n\mathtt{b}^i\mathtt{c}^i\mathtt{\#} \mid i, n \geq 0 \,\}$ and therefore an E-set.

B Epilogue: How to present things, general or special

LETZTER ABSCHNITT VOR APPENDIX?

Struggling through the literature and writing this article has prompted me to reflect on the different possible ways of presenting things.

(1) Should one make a proof self-contained, or

(2) should one apply the tools and the theory that has been established?

(A) formal framework

like programming in assembly language.

does have its advantages. utmost efficiency.

(B) description of an algorithm.

machine does several things while processing the input ... product of the state sets.

As an example, let me discuss Lemma 3 in Section 7.5 about encoding the input alphabet in binary before presenting it to the PFA. Binary coding is a very fundamental and common procedure in Computer Science, and when one thinks about it in terms of a (randomized or deterministic) algorithm, it is hardly worth mentioning.

Only for working out the effect on the number of states, and hence on the size of the transition matrices, it is necessary to become more concrete.

Example 1. Coding in binary. The readers are encouraged to familiarize themselves with the statement and the proof of Lemma 3 in Section 7.5.

The proof is self-contained and short.

I have not bothered to give a formal proof, letting the construction stand for itself. left for the reader to see that A' simulates the original automaton A faithfully,

in terms of an algorithm,

Writing the two explicit transition matrices is an add-on, making things more concrete.

For comparison, let us look at the treatment in the original sources, Paz [20, Lemma 6.15, p. 190]. and NH ..., see Appendix A. First, they appeal to a general statement that acceptance probabilities are preserved under GSM-mappings (mappings induced by a *generalized sequential machine*) [20, Definition 6.2 and Theorem 6.10, p. 186]. (A generalized sequential machine is a finite automaton with output. The machine is generalized in the sense that the output at any step is from Δ^*, i.e., the machine can produce several symbols or no output at all. DUP!!? OK?)

the decoding function from to binary alphabet $\{\mathtt{a}, \mathtt{b}\}$ to the original alphabet Σ can be easily carried out by a GSM.

Some 7-letter alphabet Σ (see Appendix A) τ that uses the code-words $\mathtt{a}^i\mathtt{b}$ for $i = 1, \ldots, 7$ using a straightforward variant of the code of Lemma 3. (The code of Lemma 3 is optimized to minimize the number of states.

proof is eight lines in total.

... like looking at q and ignoring the counter i in the acceptance decision.

Words $x \in \{\mathtt{a}, \mathtt{b}\}^*$ that are not of the form $\tau(u)$ are accepted with probability 0.

comes with a cost

The method of assigning

Would also assign a positive acceptance probability to the "unfinished" strings

Words that are not of the form $\tau(u)$ are accepted with the same probability as the partially decoded word u'.

It would correspond to accepting the input based solely on the state q, without taking into account the counter i.

Of course, it does not make sense to set up a whole body of lemmas if one does not use it.

In a separate step the probabilities have to be patched up to correct this. A finite automaton, and in particular a PFA, can easily check the wellformedness of x, i.e., membership in $\{\mathtt{ab}, \mathtt{aab}, \ldots, \mathtt{a}^7\mathtt{b}\}^*$. This is expressed very concisely

in two line

[20, p. 190, lines 12–13]).

in terms of a characteristic function χ of the above regular language ("event") and the operation $g' = g \wedge \chi$.

Intersection $f \wedge g$ of two fuzzy events is defined in [14, Definition 2, p. 251].

Closure under intersection with a *regular* set in [14, Theorem 4, p. 252]. in etwas verklausulierter Form, (cited from their earlier paper 1968). in the same form in [20, Proposition 1.9 of Chapter IIIA, pp. 148–9] with proper credit to NasuHonda1968. [20, Section IIIA.3, p. 152]).

the required closure property is not even stated directly [19] but it is easily derivable as a corollary.

appealing to another closure property of P-sets.

(and a typo, $\tau(g, \lambda)$ should be replaced by $T(g, \lambda)$.) (see [20, Definition 1.1 of Section IIIB, p. 153]).

In [14, p. 269, last line]

and, following them, Paz ...

In [14, p. 269, last line], the probabilities have to be patched up in two different ways, because of the more ambitious goal of establishing that the encoded language is an E-set: In one PFA, the ill-formed words are accepted with probability 1, in the other, with probability 0.

(In Paz is the same thing also, when treating the det. lin. languages.)

[20]

notation and abbreviations

terse and perhaps elegant,

to the point of becoming almost hermetic.

[19] [14, Theorem 4, p. 251], [20, Proposition 1.9 of Chapter IIIA, pp. 148–9]: Formulated in our language: If $\phi(u)$ and $\psi(u)$ are acceptance probabilities of (rational)(?) PFA's and the set $\{ u \in \Sigma^* \mid \phi(u) > \psi(a) \}$ is a regular language, [in Paz: $\phi(u) \geq \psi(a)$] then $\max\{\phi(u), \psi(u)\}$ and $\min\{\phi(u), \psi(u)\}$ are also acceptance probabilities of (rational)(?) PFA's.

[20] Actually, since we are concerned with emptiness, the inaccuracy wouldn't really matter. It we assign to a string x that is not of the form $\tau(u)$ the acceptance probability of some other u', this does not change the answer to the question whether probabilities $> \frac{1}{4}$ exist.

On the other hand,

especially if it is coupled with a lack of precision, it makes a proof practically inaccessible to a reader who is interested only in a particular result.

might be on the too verbose side abbreviation for being an acceptance probability of a (rational) PFA

Example 1. Restricting the acceptance degrees f to a 0-1-vector. 2nd example. converting f to 0-1.

I have emphasized the duality between the intuitive view of a PFA as a randomizes algorithm, subject to the finite-memory restriction, versus the formal view via transition matrices.

"matrices" versus "algorithm"

Paz (see also [20, p. 151]). very elegantly. (As part of a more general statement, Theorem 2.4 of Section IIIA Convex combination of 0-1-vectors f_i. [20, Proposition 1.7 of Section IIIA, p. 148]). The number of states of the discussion is not discussed, generality leads to an exponential blowup

[7, Step V, pp. 123–124], following Turakainen ....

Idea After ensuring that $\sum f_i = 1$, throws generates a random variable with d different outcomes according to the distribution f_i simultaneously with every move.

This allows to make

proof of Theorem 2b in Section 7.4.

writes down the $d^2 \times d^2$ transition matrix without explaining

see footnote 7)

"Man verifiziert nun leicht"

our approach generates a customized random variable for each state

instead of

a uniform variable that is good for every state.

and is thus less wasteful of states.

There is an analogous dilemma in writing software. Should one use library functions, or should one simply program everything from scratch? Using a library often comes with an overhead, because the functions that it offers don't usually fit exactly and require adaptation.

There is a crucial difference between writing software and writing mathematics: In programming, the primary goal is to get things done, as effectively as possible. The target is the computer, and the machine does not have to understand what it is doing, and why. (But yes, software needs to be maintained, and a different programmer will have to read and understand the code.)

In writing mathematics, one also wants to get things proved, but the primary target audience is the mathematical reader. Besides convincing the reader of correctness, one often wishes to elicit some understanding.

In this article, I have enjoyed the freedom of concentrating on a single result and writing everything from scratch. It gave me the occasion to think about the constructions in terms of explicit transition matrices, and to come up with the specializations and strengthenings that I have presented.

C Small UTMs, Overview

`https://www.thi.uni-hannover.de/fileadmin/thi/abschlussarbeiten/2020/ba_strieker.pdf` Universität Hannover, Bachelor of Science, Laura Strieker. 39 pp.

https://link.springer.com/chapter/10.1007/978-3-642-03409-1_24 https://arxiv.org/pdf/0707.4489.pdf https://arxiv.org/abs/0707.4489 Small weakly universal Turing machines. Turlough Neary, Damien Woods. [17] We give small universal Turing machines with state-symbol pairs of (6, 2), (3, 3) and (2, 4). These machines are weakly universal, which means that they have an infinitely repeated word to the left of their input and another to the right. They simulate Rule 110 and are currently the smallest known weakly universal Turing machines.

Recently, Woods and Neary [29] have given 3-state, 7-symbol and 4-state, 5-symbol semiweakly universal machines that simulate cyclic tag systems.

29. Damien Woods and Turlough Neary. Small semi-weakly universal Turing machines. In Jérôme Durand-Lose and Maurice Margenstern, editors, Machines, Computations, and Universality (MCU), volume 4664 of LNCS, pages 306–323, Orléans, France, September 2007. Springer.

Another in Fundamenta Informaticae 91 (2009) 179-195. 10.3233/FI-2009-0011 http://doi.org/10.3233/FI-2009-0011, https://mural.maynoothuniversity.ie/12415/ with incorrect page numbers. https://mural.maynoothuniversity.ie/12415/1/Woods_SmallSemi_2009.pdf This work is an extended version of [34] and contains new results. 34 = MNU paper.

3-state, 7-symbol machine The cyclic tag system's list of appendants is reversed and encoded to the left of the input. This encoded list is repeated infinitely often to the left.

1 halting rule, 9 left, 11 right rules.

It is also known from the work of Margenstern [8], Michel [10], and Baiocchi [1] that the region between the non-universal curve and the smallest standard universal machines contains (standard) machines that simulate the $3x + 1$ problem and other related problems.

[8] M. Margenstern. Frontier between decidability and undecidability: a survey. Theoretical Computer Science, 231(2):217–251, Jan. 2000

[10] P. Michel. Small Turing machines and generalized busy beaver competition. Theoretical Computer Science, 326:45–56, Oct. 2004.

[1] tech. report 1998, U. di Roma

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

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

"The initial state is u1 and the blank symbol is c." (Def 3.1)

T. Neary and D. Woods / Four Small Universal Turing Machines p.121(wrong)

Table 16. Table of behavior for U15,2.

	u1	u2	u3	u4	u5	u6	u7	u8
c:	cRu2	bRu3	cLu7	cLu6	bRu1	bLu4	cLu8	bLu9
b:	bRu1	bRu1	cLu5	bLu5	bLu4	bLu4	bLu7	bLu7

	u9	u10	u11	u12	u13	u14	u15
c:	cRu1	bLu11	cRu12	cRu13	cLu2	cLu3	cRu14
b:	bLu10		bRu14	bRu12	bRu12	cRu15	bRu1

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 $\Gamma \cup \{\#\}$, 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 situation 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

Four Small Universal Turing Machines

Neary and Woods [16, Section 3.5] [21]

(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 $U_{15,2}$. originally, 14 right-moving rules, 1 halting rule, 15 left-moving rules. $|\Gamma| = 2$. After swapping: 15 right-moving rules, 1 halting rule, 14 left-moving rules. $|\Gamma| = 2$.

$2 \times 3 + 3 + 14 \times 2 + 16 = 53$ (sic!)

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

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

We always have the freedom to swap left-moving with right-moving rules, by flipping the Turing machine's tape and starting over the rightmost input character!

Universal Turing Machines for which the Halting Problem is Undecidable.

Watanabe. Shigeru Watanabe, 4-symbol 5-state universal Turing machine, Information Processing Society of Japan Magazine 13 (9) (1972) 588–592

also another one: 3 symbols, 7 states.

Suppose the 21 rules are balanced between left and right Section 3.2 $U_{9,3}$. 10 right-moving rules, 1 halting rule, 10 left-moving rules. $|\Gamma| = 3$

$9 + 3 + 10 \times 3 + 11 = 53$ ONE smaller.

Wolfram–Cook, Rule 110. [9, Fig. 1]

5 symbols, 2 states, 4 L, 6 R

4 symbols, 3 states, 2 L, 6 R, 4 transitions unused

3 symbols, 4 states, 3 L, 7 R, 2 unused

2 symbols, 7 states, 3 L, 9 R, 2 unused

[8] Thanks to David Eppstein for figuring out that the two-symbol machine can be achieved with only seven states. (personal communication, 1998)

[9, Note [7], p. 38]

depends (in a complicated way) on the tag system that is simulated, and thus it is not useful for our purpose.

[21] see `http://mural.maynoothuniversity.ie/12416/`, with incorrect page numbers

questions about their behavior, such as "Will this sequence of symbols ever appear on the tape?", are undecidable.

Margenstern and Pavlotskaya. Margenstern and Pavlotskaya [9] gave a 2-state, 3-symbol Turing machine that is universal when coupled with a finite automaton. This machine uses only 5 instructions.

9. Maurice Margenstern and Liudmila Pavlotskaya. On the optimal number of instructions for universality of Turing machines connected with a finite automaton. International Journal of Algebra and Computation, 13(2):133–202, April 2003.

`https://www.semanticscholar.org/paper/On-the-Optimal-Number-of-Instructions-for-Universal-Margenstern-Pavlotska%C3%AFa/95ab0fc3122aa196187bff9610b8e71f2a33ac6e`

Wolfram's 2, 3 Turing Machine

On universality. Intermediate Turing degrees.

`https://en.wikipedia.org/wiki/Universal_Turing_machine`

D Further literature

D.1 Turlough Neary

Small universal Turing machines. `https://www.ini.uzh.ch/~tneary/tneary_Thesis.pdf` A thesis submitted for the degree of Doctor of Philosophy Department of Computer Science National University of Ireland, Maynooth

Supervisors: Dr. Damien Woods and Dr. J. Paul Gibson

External Examiner: Prof. Maurice Margenstern

October 2008

Turlough Neary

Institute for Neuroinformatics, University of Zürich and ETH Zürich, Switzerland. tneary@ini.phys.ethz.ch

`https://www.ini.uzh.ch/~tneary/`

I am a research scientist in the group of Matthew Cook.

now alumnus (PostDoc)

Last entry 2019

STACS 2015: The PCP is undecidable already with as few as five word pairs [15]: "Using Cook's [6] reduction of tag systems to cyclic tag systems, it is a straightforward matter to give a binary cyclic tag system program that is universal and contains only two 1 symbols. We also use our binary tag system construction to improve the bound for the number of pairs of words for which the Post correspondence problem [18] is undecidable, and the bounds for the simplest sets of matrices for which the mortality problem [16] is undecidable."

What is a *binary cyclic tag system program*?

"By applying the reductions in [8] and [3] to P in Theorem 11 we get Corollary 12.

Corollary 12. The matrix mortality problem is undecidable for sets with six 3×3 matrices and for sets with two 18×18 matrices."

D.2 Wolfram, A New Kind of Science

no termination mentioned (Chap. 11, around p. 690)

p. 667 tag systems of a certain kind are universal.

p.704 7 states 4 colors

"halting" is somehow treated as an exotic feature of Turing machines

[Actually, the "good" Turing machines of Turing 1936 don't halt, ("non-circular"), except by mistake.] Review in `https://www.cambridge.org/core/journals/journal-of-symbolic-logic/article/abs/a-m-turing-on-computable-numbers-with-an-application-to-the-entscheidungs-problcm-proceedings-of-the-london-mathematical-society-2-s-vol-42-19361937-pp-230265/4DFCA89035F7F7C5BF4DB5129B8BB09E`

2 states 5 colors → rule 110

p.94 tag system CAN halt: case (c): all elements are eventually removed.

So-called "register machines" p.97 (Chapter 3) are really 2-counter machines. German "Registermaschine" is something else, namely RAM (random-access-machine). But Minsky also says "Universal Program machines with Two Registers".

See also notes: p. 1115 History

p. 1116 . (The initial conditions in the note below quite soon become too large to run explicitly on any existing computer.)

The discussion in the main text and the construction above require a cyclic tag system with blocks that are a multiple of 6 long, and in which at least one block is added at some point in each complete cycle. By inserting k = 6 Ceiling[Length[subs]/6] in the definition of TS1ToCT from page 1113 one can construct a cyclic tag system of this kind to emulate any one-element-dependence tag system.

D.3 On universality

It seems that such a system can simulate a machine that resembles a Turing machine except that it misses the most important state: a halting state. (Another, less crucial difference is initializing the "empty" parts of the "tape" in both sides of the input not with blanks, but with some other periodic pattern.)

Unfortunately, the author never clearly define the concept of universality that he uses. .. The same vagueness and confusion ... permeates the literature in which

Wikipedia. ...

A basic fact about a universal Turing machine in the classic sense, is that the halting problem is undecidable. Which problem .. about

Cook [9, p. 11] Universality in Elementary Cellular Automata

Cook states some undecidable problem

... any program in just the tape data portion of the initial arrangement, while always using a fixed repeating pattern on the right and on the left. To find a specific example of an undecidable question in such a glider system, we can easily suppose that one of the Turing machine states represented in the tag system has the peculiar property that all of its H, L, and R symbols lead to empty appendants for the end of the tape. Then, if this state is ever entered, the tag system will stop appending symbols to the tape, and the glider system will stop sending moving data to the end of the tape, causing ossifiers to hit tape data after all. This can lead to a new kind of glider being produced. So one undecidable question is, "Will the following glider ever appear?"

We could also easily suppose that one of the Turing machine states represented in the tag system has the property that it halves both the L and R symbols of the tape, and always stays in the same state. In this case, the tape will eventually be zeroed out, and

the tag system's behavior will become periodic with period one, and the behavior of the glider system will likewise become periodic, with a specific predeterminable (space, time) period. So another example of an undecidable question is, "Will the behavior become periodic with the following period?"

If the glider system becomes periodic at all, then the emulated tag system must be periodic as well, meaning that the Turing machine is in an infinite loop where it keeps entering the same state, with the same tape to the left and right. Conversely, if the Turing machine enters such a loop, then the glider system must become periodic. Since it is undecidable whether a Turing machine will enter such a loop, another example of an undecidable question for the glider system is, "Will the behavior become periodic at all?"

"Rechnender Raum", Elektronische Datenverarbeitung, vol. 8, pages 336–344, 1967
http://www.mathrix.org/zenil/ZuseCalculatingSpace-GermanZenil.pdf
https://www.informationphilosopher.com/solutions/scientists/zuse/Rechnender_Raum.pdf 9pp.
https://link.springer.com/book/10.1007/978-3-663-02723-2
http://www.horst-zuse.homepage.t-online.de/rechnender-raum.html

D.4 Tag systems and universality

However, for our purposes, this would require some notion of universality for an individual tag system,

and such a notion does not seem to exist? seem to have been studied.

[It exists for cellular automata. see Wolfram, NKS: universal in a certain sense, it can simulate any cellular automaton, with proper setup.]

One could look at universal tag systems, with a fixed set of rules, but an arbitrary starting word. (e.g. a tag system simulating a UTM? add another round of the detour!)

In particular, a 2-tag system can be constructed to emulate a Universal Turing machine, as was done by Wang 1963 and by Cocke & Minsky 1964. https://en.wikipedia.org/wiki/Tag_system

Cocke, John; Minsky, Marvin (1964). "Universality of Tag Systems with P=2". J. Assoc. Comput. Mach. 11: 15–20. doi:10.1145/321203.321206. hdl:1721.1/6107. S2CID 2799125. https://doi.org/10.1145/321203.321206

deletion number P = 2.

Wang, H.: "Tag Systems and Lag Systems", Math. Annalen 152, 65–74, 1963. https://doi.org/10.1007/BF01343730

Common review of both papers in

The Journal of Symbolic Logic , Volume 36 , Issue 2 , June 1971 , pp. 344 DOI: https://doi.org/10.2307/2270314

D.5 Marvin Minsky

[reference ??? This (WHAT?) 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.]

A 2-way automaton can move forward and backward on the input tape, and thus process the input as often as it wants. Such an algorithm can boost the correctness probability, and thus it is possible to make a correct decision (accept or reject) with

probability at least $1 - \varepsilon$, for any ε. (By contrast, for a (1-way) probabilistic finite automaton this is impossible: If there is a gap interval $[p_1, p_2]$ of positive length such that the acceptance probability is never in this interval, the accepted language is regular. Michael O. Rabin 1963, Probabilistic Automata, Information and Control 6 230–245. [21] [20, Theorem IIIB.2.3]

D.6 Blondel and Tsitsiklis (2000)

Blondel and Tsitsiklis [1] 2000: PFA EMPTINESS is undecidable cannot be found in its entirety in the published literature. A proof (stated with a different terminology) is given in Theorem 6:17 in p. 190 of [17]. The proof given there is a few lines long and refers to a long cascade of lemma that appear at various places in the book. A full proof is hard to reconstruct. Sketches of an alternative proof can be found in several recent references; see, e.g., [6,14,15]. Finally, a full proof can be found in the expanded version of [6]; see Theorem 3:2 in [7].

[14] O. Madani, On the computability of infinite-horizon partially observable Markov decision processes, AAAI98 Fall Symposium on Planning with POMDPs, Orlando, FL, 1998.

[15] O. Madani, S. Hanks, A. Condon, On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems, Proceedings of the Sixteenth National Conference on Artificial Intelligence, Orlando, FL, July 1999.

D.7 Others

Rūsiņš Mārtiņš Freivalds

?? RELATION TO sequential test (A. Wald)

D.8 Literature about stochastic automata and languages

Rabin [21, Sections IX–XII, p. 242–245] calls PFA's in which all transition probabilities are positive *actual automata* and studies their *very special properties*.

WHAT ARE THOSE PROPERTIES? Well, not so special. for ISOLATED cut-points, only very special languages.

[14, p.261]

//Lemma 11: An E-event ($\phi(a) = \psi(a)$) is a P-event (stochastic language).

...

"Matuura et al. and Turakainen studied linear space automata (or generalized automata) and found independently the same fact that the family of stochastic languages is the same as the family of languages ac- cepted by linear space automata (or generalized automata). Matuura et al. gave us a key idea with respect to Lemma 11 in this viewpoint. P-sets, E-sets and D-sets can be studied generally in this viewpoint, and E-sets and D-sets can be related to the work of Schüzenberger (sic!) (1961). The next remark and Lemma 12 follows from Schüzenberger's results but as for Lemma 12 we will give a straightforward proof."

//Remark: neither E-set nor D-set ($\phi(a) \neq \psi(a)$) contains the other.

LEMMA 12. ...

What are "linear space automata"? With arbitrary matrices M, working on a *linear vector space* with linear operators [26].

"(c) The unbounded part of the memory, V_N, is the finite dimensional vector space of the vectors with N integral coordinates; this part of the memory plays only a passive role and all the control of the automaton is performed by the finite part."

Acceptance is by NON-containment in a Subspace. (or union of several subspaces) [14, p.270]

// Thm.21 is the undecidability of PFA emptiness.

footnote 6: As for Theorem 21, it reduces to the statement in p. 150 of Schüzenberger (1962) from the viewpoint that we said before Lemma 12 of this paper.

footnote → Schützenberger 1961/63: Which of this is in Paz? only Sch 1961 (I&C) [26]

"... grateful to Dr. Y. Inagaki and Mr. H. Matuuta (sic!) in Nagoya University ..."

MATUURA,H., INAGAKI, Y., AND HUKUMURA, T. (1968), Generalization of finite automata and its analysis. Papers of Technical Group on Automata. IECE, Japan. 1968—1 (in Japanese).

MATUURA, H., INAGAKI, Y., AND HUKUMURA,T. (1968), Linear space automata and probabilistic automata. Records of 1968 National Convention. IECE, Japan. S 8–4. (in Japanese).

TURAKAINEN, P. (1968), On probabilistic automata and their generalizations. Annales Academiae Scientiarum Fenicae, Series A I Mathematica 429.

SCHÜTZENBERGER, M. P. (1961), On the definitions of a family of automata, Inform. Control 4, 245-270. [26]

SCHÜTZENBERGER, M. P. (1962), Certain elementary families of automata. In "Mathematical Theory of Automata." (J. Fox, ed.) pp. 139-153. Polytechnic Press of the Polytechnic Institute of Brooklyn, 1963.

Marcel Schützenberger [25] [22]

review of Schützenberger's papers by Michael O. Rabin in The Journal of Symbolic Logic , Volume 34 , Issue 2 , 25 July 1969 , pp. 296 - 297 DOI: `https://doi.org/10.2307/2271115`

"The third paper is mainly expository and is based on the first two papers."

M. P. SCHUTZENBERGER. Certain elementary families of automata. Proceedings of the Symposium of Mathematical Theory of Automata, New York, N.Y., Microwave Research Symposia series vol. 12, Polytechnic Press of the Polytechnic Institute of Brooklyn, New York 1963, pp. 139-153.

closed under complement,

This is credited to "(Paz)" in [14]

cites 2 papers of Paz

Some aspects of probabilistic automata [19] 1966:

Paz, A. (1967), Fuzzy star functions, probabilistic automata and their approximation by nonprobabilistic automata. IEEE Conference Record, 1967 Eighth Annual Symposium on Switching and Automata Theory, 280-290.

also in the other, 1968 Nasu-Honda paper, p.288: "THEOREM 4.1 (due to Paz)." (closed under complement)

BUT THIS REFERS to probabilistic event == some fuzzy event. complementation means just complementing the acceptance probabilities. (trivial)

cites ONLY [19] 1966, and gives an independent proof.

C. V. Page [18]:

F: output vector, a n-component column vector whose entries are *real* numbers. F_i is the output from state S_i

arbitrary output values in the interval $[0, 1]$ could be accommodated by introducing an additional accepting state C. Upon reading a special termination symbol, the PFA makes one probabilistic step to C or the rejecting state B, reflecting the output values

[22] `https://monge.univ-mlv.fr/~berstel/Mps/Travaux/A/A/1963-4ElementaryFamAutomataSympThAut.pdf`

f_q. From C, the PFA can only go to the rejecting state B. This ensures that the corresponding transition matrix can be used only once, at the end.

However, as this would introduce an additional state, this cannot be used to improve Theorem 2 for the "classic" setting.

Generalized automata and stochastic languages, by Paavo Turakainen, Proc. Amer. Math. Soc. 21 (1969), 303-309. DOI: `https://doi.org/10.1090/S0002-9939-1969-0242596-1`

"the initial vector", "the final vector" f, cut-point = f!

Turakainen [29]

arbitrary positive or negative matrices, and π and f. Does not define a more general class of languages. [7, §3.3.2, Der Satz von Turakainen, p.120].

INFORMATION AND CONTROL 10, 215–219 (1967), On m-Adic Probabilistic Automata, ARTO SALOMAA, proves regular IFF $\lambda \in \mathbb{Q}$, cites Paz [19] about m-adic automata.

Paz [19] 1966:

E X A M P L E S (p.35)

1. An "m-adic two state p.a." is obtained by prescribing the following conditions: ... use all m different matrices.

Rabin [21] already defined THE binary automaton, with 2 states and 2 transitions.

"The following matrices were suggested by E. F. Moore."

Rabin [21]: IX. ACTUAL AUTOMATA, p.242. all transition probabilities are POSITIVE.

Adding a small probability to change $\geq \lambda$ into $> \lambda$. The idea of adding a small probability to change $\geq \lambda$ into $> \lambda$ is can also be used to prove that the languages recognized by rational PFA's are closed under complement. But: Sometimes proved with the techniques of ???Turakainen [29]? Paz cites other Turakainen papers! see below

This is crucial for for CLOSED UNDER COMPLEMENT for rational PFA's, where is this in the literature? [Closure properties must have been well-known.. ? In particular for the complement? Turakainen [29]? No the complement is not mentioned in [29], only the mirror. And moreover that appeared in the same year. Who is credited in Paz: Paz uses integral pseudo-stuff for complementation. cites Paavo Turakainen [1969b Ann. Fenn. rational probabilistic] `https://www.acadsci.fi/mathematica/1969/no429pp01-53.pdf` 51 pp. ANNALES ACADEMIAE SCIENTIARUM FENNICAE Series A I. MATHEMATICA 429 On probabilistic automata and their generalizations. `https://doi.org/10.5186/aasfm.1969.429`

–53.

(53 pp.), No 429 (1968)

Claus [7, Hilfssatz 31.ii, p. 131] is closed under complement for P-sets. Proof as an exercise; for the *rational* generalized acceptors of Turakainen [29].

The proof in Paz 1971. What the heck is a and b in [20, p. 188, line 1]? What is m? Is this erroneously copied from a different part of the proof? (Lemma 6.12)? A similar line is [20, p. 188, line -6]?

disfigured, defaced, spoiled, bungled, daubed (beschmieren), botched (Flickwerk, Pfuscherei)

defaced by occasional typos and editing errors (such as passages in a proof that obviously belong to another proof)

[14, Lemma 14, pp. 264] requires PFA for which the acceptance probability is an injective mapping from Σ^*, "for example .. in the proof of Proposition 13": These are the m-ary automata.

Paz (last page 193) gives precise credit for Section 6. Section 6 up to Theorem 6.8 is from Turakainen [1969b Ann. Fenn. rational probabilistic] `https://www.acadsci.fi/mathematica/1969/no429pp01-53.pdf` 51 pp. (NOT [29]!) and 1970a (Akad. Turku, not closed under ...) and 1970b (tech.rep. "some closure properties"), using the language of Nasu and Honda as a starting point. Everything from Exercise 6.9 to the end is from Nasu and Honda [14]

Turakainen [1969b Ann. Fenn. rational probabilistic] `https://www.acadsci.fi/mathematica/1969/no429pp01-53.pdf` Theorem 15 (p. 37 is about complement but ONLY when $L_=$ is regular!).

Theorem 16 on p.38 is about reducing to 0-1 f. equation (9.1) on p. 39. the number of states is squared as in Claus.

Mention Bucharev and Starke?

The proof in Claus 1971. Volker Claus from 1971 [7, Satz 28, p. 157],

Claus says: proved in Nasu–Honda (1969) for a fixed alphabet of size ≥ 2 *without being able to bound the number of states* , "wobei nun aber die Zustandszahl der SAkz nicht beschränkt ist." Nasu–Honda is a different proof after all!

Hilfssatz 33, S. 131. closed under complement if rational. Exercise, using generalized acceptors VAkz (S. 119). This seems to be also how it is done in Paz.

$L_{\phi=\psi}$: p. 155, using the formula (and some lemmas about product automata) (very nice notation!)

Using the PCP, encode with *ternary* automaton, and alphabet $\{1, 2\}$, avoiding the trailing zeros issue.

Terminology. "realized" is used only for fuzzy events. realizable fuzzy events = probabilistic events.

represented (Salomaa, Theory of automata, p.76) is indeed used for P-sets.

Re. "events". I find this choice of terminology, which pervades much of the literature, (in particular the whole Chapter III of the book [20]) unfortunate since an event is rather something whose probability is to be measured.

Turakainen [29] speaks only of *stochastic languages*. (+)

Claus [7, Definition 39, p. 148] "unbestimmtes Ereignis" "= translation of fuzzy set" according to Zadeh, actually = fuzzy *language*. Special cases: [7, Definition 40, p. 148]: "stochastisches Ereignis" = acceptance probability of a PFA, "reguläres Ereignis" = (characteristic function of a) regular language. But at least stochastic *languages* = stoch. Sprachen: [7, Definition 26–27, p. 101]

[14, Definition 1, p.251]: *fuzzy event*. (from Σ^*. "Subsets of Σ^* will sometimes be called events or languages."

Rabin [21, p. 233]: Subsets of Σ^* (i.e., sets of tapes) will sometimes be called *events*. ... *regular event*. Rabin [21, Definition 11, p. 243]: definite event.

Old proof of weaker version of Eliminating the output vector, proof of Theorem 2b We will use an adaptation of this procedure to prove part (b) of the following theorem.

(b) We first explain a simple version, where the number of states is multiplied by 8.

We start with the set $\mathcal{M}'''$ of 52 9×9 matrices of Theorem 4b. We simulate the corresponding automaton, but simultaneously, with every input symbol, we throw an octahedral die. In other words, we determine a random number r between 1 and 8. The acceptance criterion is as follows: If we are in state q and the output value is $f_q = \frac{i}{8}$, we accept if the input if $r \in \{1, \dots, i\}$. In other words, the randomness for making the acceptance decision is already generated *before* the last symbol is read, and we simply need to use it by looking up the value r. It is clear that the input is accepted with the correct probability $f_q = \frac{i}{8}$. The number of states is multiplied by 8, and each transition matrix $M \in \mathcal{M}'''$ is converted to the matrix

$$\begin{pmatrix}
1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M \\
1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M \\
1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M \\
1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M \\
1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M \\
1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M \\
1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M \\
1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M & 1/8\cdot M
\end{pmatrix}.$$

(A more straightforward procedure would be to throw the die once at the beginning and remember the result r, but then the transition matrix would not be positive.)

We can save states by making a cleverer, irregular die. All that is needed is that every output value $f_q \in \{\frac{1}{4}, \frac{1}{2}, \frac{5}{8}\}$ can be combined from the face probabilities of the die. ... □

E Notations

A accepting computation

A, A' PFA's

a_i ... a_m, $b_1, \dots, b_n$ *index sequence*(!) for PCP solution

`a,b` symbols for equality checker

$B(u)$ binary PFA

$[0, c_1], (c_1, c_2], (c_2, c_3], \dots$ interval boundaries

γ, γ_1 small constant (power of 2)

Γ tape alphabet, including the blank symbol ␣

$d \times d$ size of matrix in joint spectral radius

$d \times d$ size of matrix in PFA = number of states (previously n)

e_1 first unit vector (local usage)

$\varepsilon > 0$. probabilities close to 0 or 1.

ϵ empty word (local usage)

f, f_q characteristic vector (or more general, output vector) of accepting states q. η^F is used in Nasu and Honda, where F is the set of accepting states. [7] and most others uses f instead of η. Paz p. 150 uses η. Blondel and Tsitsiklis 2000 use 0-1-vector η (following Paz).

G modulus (formerly Q)

H halting state

i, j lengths for equality checker

k number of input pairs for PCP

$K = 10$ repetitions until declaring outcome "INCORRECT"

λ cut-point

l_i, r_i counter values in 2CM
L, R movements of TM, as part of the rules (quite local usage)
MPCP Modified Post Correspondence Problem
2MPCP doubly Modified Post Correspondence Problem (not used much)
$\mathcal{M}$ set of matrices
M, M_i matrices
m length of matrix product in PFA
m length of PCP solution
m length of matrix product in j.spectral radius
m length of accepting computation in 2CM (formerly n)
n length of *partial* PCP solution
PFA probabilistic finite automaton
PCP Post's Correspondence Problem
$p_{00}, p_{01}, ...$ transition probabilities in 2x2 automaton $B(u)$
π, π_q starting distribution
Q TM states, 2CM states
q_i ... q_n states in accepting computation for 2CM
q, q' states of TM (in rules)
q state of PFA
q_A, q_R extra states for PFA
RMPCP Reversed Modified Post Correspondence Problem (used only once)
r_i, l_i counter values in 2CM
x^R string reversal
$s, s', t \in \Gamma$ tape symbols
$\sigma \in \Sigma$ (letter from) the input alphabet (of PFA). Currently not used as alphabet of PCP
t number of repetitions of A to boost the acceptance. Previously m, $\rightarrow b?C?c??$
T Turing machine (local)
u input to TM
u, u', (v) binary strings
U, U_W, U_0 various universal Turing machines (local)
(v_i, w_i) ... (v_k, w_k) PCP pairs
$0.x_1x_2 \ldots x_{48}$ random bits
Φ, Ψ states
$\phi = \phi(a)$, ψ acceptance probabilities
separator symbol
$\sqcup$ blank space