SARAJEVO JOURNAL OF MATHEMATICS
Vol.AB (CD), No.y (20ab), 1–22

DOI: 10.5644/SJM.AB.0y.xx

TESTING RANDOMNESS BY MATCHING PENNIES
— DEDICATED TO MIRJANA VUKOVIĆ —

DUSKO PAVLOVIC PETER-MICHAEL SEIDEL MUZAMIL YAHIA

ABSTRACT. In the game of Matching Pennies, Alice and Bob each hold a penny, and at every tick of the clock they simultaneously display the head or the tail sides of their coins. If they both display the same side, then Alice wins Bob's penny; if they display different sides, then Bob wins Alice's penny. To avoid giving the opponent a chance to win, both players seem to have nothing else to do but to randomly play heads and tails with equal frequencies. However, while not losing in this game is easy, not missing an opportunity to win is not. Randomizing your own moves can be made easy. Recognizing when the opponent's moves are not random can be arbitrarily hard.

The notion of randomness is central in game theory, but it is usually taken for granted. The notion of outsmarting is not central in game theory, but it is central in the practice of gaming. We pursue the idea that these two notions can be usefully viewed as two sides of the same coin. The resulting analysis suggests that the methods for strategizing in gaming and security, and for randomizing in computation, can be leveraged against each other.

1. INTRODUCTION

1.1. Game of Matching Pennies

The payoff matrix for Matching Pennies is displayed in Table 1. For the convenience of using the bitstring notations, we denote the heads move as 0 and the tails move as 1. The game is repeated, and we assume that it is played long enough that even the smallest strategic advantages are captured in the outcome. Both players can win or lose arbitrarily large amounts of pennies.

2010 *Mathematics Subject Classification.* 03D32,91A26,91A26, 68Q32.

Key words and phrases. randomness, computability, algorithmic complexity, equilibrium, hypothesis testing.

		0	1
0		-1 1	1 -1
1		1 -1	-1 1

TABLE 1. Payoffs for Matching Pennies

1.2. **How not to lose Matching Pennies**

To determine her strategy, Alice might reason something like this.

> Suppose that I consistently play 1 with a frequency $p \in [0,1]$ and thus 0 with a frequency $1-p$. If I set $p < \frac{1}{2}$, then Bob can get the expected payoff $-p+(1-p) = 1-2p > 0$ by playing 1. If I set $p > \frac{1}{2}$, then Bob can get the expected payoff $p-(1-p) = 2p-1 > 0$ by playing 0. If I set $p = \frac{1}{2}$, then Bob's expected payoff is the same whether he plays 1 or 0: it is $1-2p = 2p-1 = 0$. Since Bob's winnings are my losses, the best strategy for me is to set $p = \frac{1}{2}$, and to play 0 and 1 with equal frequencies, since that minimizes my expected losses.

By the same reasoning, Bob arrives at the same conclusion, that he should set the frequency of playing 1 at $q = \frac{1}{2}$. This is the well known Nash equilibrium of the game of Matching Pennies. Both players arrive to it by minimizing the expected losses.

1.3. **Playing Matching Pennies**

In general, a mixed strategy Nash equilibrium prescribes the frequencies for both players' moves in the long run. The essential assumption is that the moves will be *randomized*. If Bob's move is predictable with some likelihood, then Alice can increase her chances to win. It seems natural to imagine that the players randomize by tossing their coins, and displaying the random outcomes. At the equilibrium, the players are just passive servants of chance, since they cannot gain anything by deviating from it. If they are rational, all they can do is toss their coins.

But suppose that Bob suddenly plays

$$0101010101010101010101010101010101010101 \tag{1.1}$$

Will Alice predict that Bob's next move is 0 and play 0 to win a penny? If she thinks probabilistically, she will probably notice that the probability of getting (1.1)

by flipping a fair coin is 2^{-40}, which is exactly the same as the chance of getting, e.g.

$$11010001001101010010111001000001000000010 \tag{1.2}$$

or any other sequence that she would accept as random. If Alice's rationality is based on probabilities, then she will not be able to distinguish any two strings of Bob's moves, since they are all equally probable if he tosses fair coins.

But if Bob knows, or even just believes, that Alice's rationality is based on probabilities, and that Alice will thus continue to randomize her moves in any case, then Bob has no reason to randomize, since playing (1.1), or (1.2), or a string of 0s, or any other string, yield the same expected payoff against Alice's random plays. On the other hand, if Alice believes that Bob's rationality is based on probabilities, then she will have no reason to randomize either, for the same reason as Bob.

So by combining their beliefs about their probabilistic reasoning, both players will become indifferent towards mixing and randomizing their moves. Their common knowledge that they may both stop randomizing, because they both know that the opponent will be unable to tell, will not change their expected payoffs. Indeed, if they both play non-randomly, one of them will almost surely win and the other will lose, but their chances to be the winner are the same, and they average out. However, while the expected payoffs remain unchanged, the higher moments will, of course, change significantly.

1.4. **How to win Matching Pennies if you can**

In order to exploit Bob's deviation from the equilibrium, or to give him an incentive to genuinely randomize his mixed equilibrium strategy, Alice must go beyond probabilities, i.e. beyond just calculating the frequency of his moves. If she just checks whether the frequencies of 0 and 1 are $\frac{1}{2}$, she will detect that the string consisting of 0s alone is not random, but not that the string (1.1) is not random; if she checks whether the frequencies of 00, 01, 10 and 11 are $\frac{1}{4}$, she will detect that (1.1) is not random, but not that the string where these four digraphs of bits alternate is not random; etc. By checking that each bitstring of length n has *in the long run* the frequency $\frac{1}{2^n}$, she will detect many non-random plays, but still miss most of them. E.g., the string

$$01101110010111011110001001101010111100 1101\ldots \tag{1.3}$$

obtained by concatenating the binary notations for the sequence of natural numbers 0,1,2,3... will pass the bias tests for all n-grams, if taken long enough, yet it is, of course, easily predictable, and obviously not random. Moreover, Bob might, e.g., randomize all even bits, and just alternate 0s and 1s at the odd positions. To recognize such opportunities, Alice will have to check that *every substring* of the string of Bob's past moves has unbiased frequencies of all n-grams. As the game goes on, Alice will thus have to keep proving that Bob's play, i.e. the ever

growing string of his past moves, is what von Mises called *Kollektiv* in his theory of probability [22]. Proving that something is a *Kollektiv* is known to be a problematic task, as specifying the substrings to be tested has led to problems that remained open for many years [2, 36].

1.5. **Randomness from equilibrium**

Scratching the surface of the basic assumption about the players' incentive to implement a mixed strategy equilibrium led us straight into the foundations of probability. There is, of course, nothing surprising about the fact that the concept of a mixed strategy, expressed in terms of probability, depends on the foundations of probability. The point is not so much that there are deep foundational problems lurking behind simple games. It seems much more useful, and more interesting, that, the other way around, there seem to be instructive ways to state the solutions of the foundational problems of probability in terms of games.

In particular, we show that the usual definition of mixed strategy equilibria based on the notion of randomness as given can be reversed, and that the notion of randomness can be defined using mixed strategy equilibria. The upshot is not just that a complicated concept of randomness is replaced by an intuitive game theoretic concept of not losing Matching Pennies at the equilibrium; the upshot is also that the effective content of both concepts, of randomness and of equilibrium, can be analyzed in terms of *computational power of testing*. This formalization brings both the basic probabilistic concepts and the basic game theoretic concepts into the logical realm of computable inductive inference [3, 9, 31, 34, 39].

1.6. **Background and related work**

We propose a simple and narrow bridge between games and probabilities. An extensive effort towards reconstructing the foundations of probability theory from a particular game has been ongoing for many years, as reported by Shafer and Vovk [33]. The work presented in this paper is not only at the opposite end of the scale in terms of its scope and technical sophistication, but it also goes in a different direction, and therefore uses an essentially different model. While the authors of [33] aim to reconstitute the full power of the diverse probabilistic tools in their rich gaming model, the point here is to illustrate how the most basic games capture the most basic probability concepts in a natural fashion. A similar analysis geared in the opposite direction of *eliminating* probabilities is provided in [38].

The bridge between games and probabilities is built using significance testing and computation. Significance testing goes back to Fisher [8, 9] and lies, of course, at the core of the method of statistical induction. The constructions sketched here are related to the computational versions of testing, developed on one hand in Martin-Löf's work [21, 24], and on the other hand in the techniques of inductive learning [3, 10, 37].

We analyze the computational content of testing. The analyses of the computational content of strategic reasoning go back to the earliest days of game theory [30], and continue through theory of bounded rationality [32], and on a broad front of algorithmic game theory [25]. The finite state machine model seems preferred for specifying strategies [13, 32], since computable strategies lead to problems with the equilibrium constructions [17, 23]. In recent work, a different family of problems, arising from the cost of strategic computations has been analyzed, including the cost of randomization [11, 12]. This led the players to not just lose the incentive to randomize, as in the little story above, but to prefer determinism. Although we are here also looking at the problem of deviating from the equilibrium into non-randomness, we are concerned with a completely different question: *How should the opponent recognize and exploit this deviation?* The present work seems to deviate from previous computational approaches to gaming in one essential aspect: we are not analyzing the computations that the players perform to construct or implement their own strategies, or the equilibrium, but the computations that they perform to test the opponents' strategies. This leads into a completely different realm of computability, that emerges from a different aspect of gaming. While the analysis goes through for most models of computation, represented by an abstract family of programmable functions, as explained in Sec 2.3, it is perhaps worth stating the obvious: that stronger notions of computation lead to stronger notions of randomness.

Although the high level models of gaming [1,14,26] are not explicitly introduced in the paper, as they are not necessary for the presented results, they were used in the earlier versions and may be gleaned in the background.

1.7. **Outline of the paper**

In Sec. 2 we spell out the preliminaries and some notations used in the paper. In Sec. 3 we motivate and explain the simplest case of randomness testing, with respect to the uniform distributions, and describe its application in gaming. Sec. 4 derives as a corollary the characterization of random strings as the equilibrium plays. In Sec. 5 we describe how to construct randomness tests for arbitrary programmable distributions. Sec. 6 closes the paper with some final comments.

2. NOTATIONS AND PRELIMINARIES

2.1. **Monoid of plays**

In the games considered in this paper, the set of *moves* is always $\mathbb{2} = \{0,1\}$. We sometimes call 0 heads and 1 tails. A *play* is a finite string (or list, or vector) of moves $\vec{x} = x_1x_2x_3\cdots x_m$, or $\vec{y} = y_1y_2y_3\cdots y_n$ played in a match of a game. The set of all bitstrings, used to represent plays, is denoted by $\mathbb{2}^*$. The empty bitstring is $()$, and the concatenation of bitstrings is $\vec{x} :: \vec{y} = x_1\cdots x_m y_1\cdots y_n$. They constitute the

monoid $\big(\mathbb{2}^*, ::, ()\big)$, freely generated by $\mathbb{2}$. The monoid structure induces the prefix ordering

$$\vec{x} \sqsubseteq \vec{y} \quad \Longleftrightarrow \quad \exists \vec{z}.\, \vec{x} :: \vec{z} = \vec{y} \tag{2.1}$$

and the *length* measure $\ell : \mathbb{2}^* \longrightarrow \mathbb{N}$, which is the unique homomorphism from the free monoid over two generators to the free monoid over one generator. The fact that the length measure is a homomorphism means that

$$\ell() = 0 \qquad \text{and} \qquad \ell(\vec{x} :: \vec{y}) = \ell(\vec{x}) + \ell(\vec{y})\,.$$

We shall also need a bijective pairing $\langle -, - \rangle : \mathbb{2}^* \times \mathbb{2}^* \longrightarrow \mathbb{2}^*$ with the projections $-_{(0)}, -_{(1)} : \mathbb{2}^* \longrightarrow \mathbb{2}^*$, which means that together they satisfy

$$\big\langle \vec{x}_{(0)}, \vec{x}_{(1)} \big\rangle = \vec{x} \qquad \big\langle \vec{x}_0, \vec{x}_1 \big\rangle_{(i)} = \vec{x}_i.$$

Using the fact that a free monoid is also cofree, a bijective pairing can be derived from any two disjoint injections $\mathbb{2}^* \hookrightarrow \mathbb{2}^*$. For simplicity, we use

$$\langle \vec{x}, \vec{y} \rangle \quad = \quad x_1 x_1 x_2 x_2 \cdots x_m x_m 01 y_1 y_2 \cdots y_n \tag{2.2}$$

where $\vec{x} = x_1 x_2 \cdots x_m$ and $\vec{y} = y_1 y_2 \cdots y_n$. The length induces the shift homomorphism

$$\ell\big(\langle \vec{x}, \vec{y} \rangle\big) \quad = \quad 2\ell(\vec{x}) + \ell(\vec{y}) + 2. \tag{2.3}$$

2.2. **Sets and functions**

$|X|$ denotes the number of elements of the set X. A function written $f : X \to Y$ is always total, whereas a partial function is written $h : X \rightharpoonup Y$. We write $h(x)\!\downarrow$ when the partial function h is defined on the input x, and $h(x)\!\uparrow$ or $h(x) = \uparrow$ when h is undefined on x.

2.3. **Programmable functions**

We say that $f : \mathbb{2}^* \rightharpoonup \mathbb{2}^*$ is *$\mathcal{L}$-programmable*, or that it is an *$\mathcal{L}$-function* when it is specified using a programming language $\mathcal{L}$. The intuitions from the reader's favorite programming language, practical or theoretical, should do. For a theoretical example, one could take $\mathcal{L}$ to be the language of finite state machines. A program could then be either a list of transitions of a Moore or Mealy machine, or a corresponding regular expression [4, 15]. The graphs of programmable functions would be regular as languages. A larger family or programmable functions would be obtained from a Turing complete programming language, like Python or Java, or from the language of Turing machines themselves. In the latter case, a program could again be a list of the transitions of the machine. A high-level formalism is based on the structure of *monoidal computer*, spelled out in [27–29].

Formally, the programming language is given by a *universal evaluator* (or *interpreter*), a partial function $\mathcal{L} : \mathbb{2}^* \times \mathbb{2}^* \rightharpoonup \mathbb{2}^*$. This function may or may not be in $\mathcal{L}$. E.g., when $\mathcal{L}$ is a Turing complete language, then $\mathcal{L}$ is an $\mathcal{L}$-function. If $\mathcal{L}$

is the language of regular expressions, then their universal evaluator is not $\mathcal{L}$-programmable.

We usually write $\mathcal{L}(x,y)$ in the form $\{x\}y$. A universal evaluator is characterized by the requirement that for every $\mathcal{L}$-function $f : \mathbb{2}^* \longrightarrow \mathbb{2}^*$ there is a bitstring $p_f :\in \mathbb{2}^*$ such that

$$f(\vec{x}) \quad = \quad \{p_f\}\vec{x}.$$

3. Randomness for uniform distributions

We focus on Alice's task to detect patterns of non-randomness in Bob's play, which she could exploit to predict his moves. Bob is assumed to be doing the same, observing Alice's play and trying to detect some patterns. But what is a pattern? And what does it mean to detect it?

Intuitively, an object has a pattern if it can be described succinctly, i.e. compressed. E.g. the string in (1.1) can be compressed to $(01)^{20}$ in mathematical notation, or to

```
for (i=0; i<20; i++) { print 01 }
```

in a Java-like programming language. The program to extend (1.1) infinitely would be $(01)^*$ or

```
for (;;) { print 01 }
```

and the program to output (1.3) would be as follows.

```
for (i=0;;i++) { print i }
```

On the other hand, a program to output the string (1.2), without a detectable pattern, would have to spell it out in full length:

```
print 11010001001101010010111001000010000001001
```

The idea that randomness can be defined as incompressibility goes back to Kolmogorov [18], and further back to the scholastic logical principle known as *Occam's Razor*, which established the priority of succinct descriptions as inductive hypotheses, as explained by Solomonoff [34].

3.1. **Testing hypotheses**

Definition 3.1. *Let $\mathcal{L}$ be a family of programmable (partial) functions. A* hypothesis *is an $\mathcal{L}$-function $h : \mathbb{2}^* \rightharpoonup \mathbb{2}^*$ such that*

$$h(\vec{x}) = \vec{y} \quad \Longrightarrow \quad \ell(\vec{x}) < \ell(\vec{y}). \tag{3.1}$$

*A string $\vec{y}$ that lies in the image of h is said to be h-*regular*. A string $\vec{x}$ on which h is defined and maps it to $\vec{y}$ is a* short description *of $\vec{y}$. A hypothesis h is* predictive *if*

$$\forall \vec{x}.\, h(\vec{x})\!\downarrow \quad \Longrightarrow \quad \exists \vec{z}.\, \vec{x} \sqsubset \vec{z} \wedge h(\vec{x}) \sqsubset h(\vec{z}) \tag{3.2}$$

where $\sqsubset$ is the prefix ordering (2.1).

The tacit idea behind predictive hypotheses is that the input data are given with some end markings, which tell the computer where the input string ends. This is the case with the data input on most real computers, but not on *"plain"* Turing machines, which leads to the restriction to *prefix-free* or *self-delimiting* machines [6, 20, 42]. For the Turing machine model, the reader should assume that there is a special symbol $\square$ denoting the end of each string, and that the string inclusion ignores that symbol. The computation $h(\vec{x})$ thus halts when it encounters $\square$ after $\vec{x}$, whereas the computation $h(\vec{z})$ proceeds longer and provides a longer output when h is predictive.

Definition 3.2. *A bitstring $\vec{y}$ is said to be h-*regular at the level $m \in \mathbb{N}$ *if*

$$\exists \vec{x}. \quad h(\vec{x}) = \vec{y} \quad \wedge \quad \ell(\vec{x}) + m \leq \ell(\vec{y}). \tag{3.3}$$

*The h-regular bitstrings at each level form the h-*regularity sets

$$H_m^n = \{\vec{y} \in 2^n | \exists \vec{x}.\, h(\vec{x}) = \vec{y} \wedge \ell(\vec{x}) + m \leq \ell(\vec{y})\} \tag{3.4}$$

$$H_m = \bigcup_{n=1}^{\infty} H_m^n. \tag{3.5}$$

Setting for convenience $H_0 = 2^*$ yields a decreasing sequence of sets:

$$H_0 \supseteq H_1 \supseteq H_2 \supseteq H_3 \supseteq \cdots \supseteq H_m \supseteq \cdots \tag{3.6}$$

*This tower of sets is the h-*test.

Note that a bitstring of length n can only be regular at the level m if $m \leq n$. The h-regularity sets H_m^n for $m > n$ are empty.

Proposition 3.1. *The size of h-regularity sets decreases exponentially with m, in the sense*

$$|H_m^n| \quad < \quad 2^{1+n-m}. \tag{3.7}$$

Proof. By (3.4), for every $\vec{y} \in H_m^n$ there is $\vec{x}$ such that $h(\vec{x}) = \vec{y}$ and $\ell(\vec{x}) + m \leq \ell(\vec{y})$, and thus $\ell(\vec{x}) \leq n - m$, because $\vec{y} \in 2^n$. The function $h : 2^* \longrightarrow 2^*$ is thus restricted to a surjection onto H_m^n from the set of strings $\vec{x}$ of lengths at most $n - m$. Hence (3.7). $\square$

Proposition 3.1 says that the chance that an observation $\vec{y}$ is h-regular at the level m decreases exponentially in m. Since this is true for all hypotheses, the implication is that most bitstrings are irregular: most hypotheses are eventually rejected, and most bitstrings are accepted as random. This is a formal expression of Laplace's observation that regular objects constitute a null set [19].

Definition 3.3. *The* h-regularity degree $\sigma_h(\vec{y})$ *is the highest* h*-regularity level that the bitstring* $\vec{y}$ *achieves, i.e.*

$$\sigma_h(\vec{y}) \quad = \quad \max\{m \leq \ell(\vec{y}) \mid \vec{y} \in H_m\}.$$

The h*-regularity degree is thus a function* $\sigma_h : 2^* \longrightarrow \mathbb{N}$.

3.2. **Alice's testing strategy**

Alice's computations of h-regularity degree follow the basic method of significance testing [8, 9]. She tests whether Bob's play $\vec{y}$ satisfies the hypothesis h. The hypothesis is rejected if $\vec{y}$ is not h-regular at a sufficiently high level. So Alice goes down the test tower $H_0 \supseteq H_1 \supseteq H_2 \supseteq \cdots$, and checks how far is it true that $\vec{y} \in H_m$. This ceases to be true when $m = \sigma_h(\vec{y})$. The hypothesis h is thus rejected if the regularity degree $\sigma_h(\vec{y})$ is below some *significance threshold* M, chosen in advance. If she wants to echo statisticians' habit to set the significance level at 1% or 5%, Alice should probably choose M to be between 4 and 7, since the indices of the test towers correspond to the negative logarithms of statistical significance levels.

But what is Alice trying to achieve by testing Bob? What will she do if she detects a significant h-regularity in his play $\vec{y}$? She wants to predict his moves, and use the prediction to take his pennies. In particular, if she finds a significantly shorter description $\vec{x}$ of $\vec{y}$ realized by h, she will try to guess a bitstring $\vec{s}$ such that $h(\vec{x} :: \vec{s})$ is defined, and extends $\vec{y}$, i.e. such that

$$\vec{y} \quad \sqsubset \quad h(\vec{x} :: \vec{s}).$$

The definition of *predictive* hypotheses requires that they always allow such extensions. So if she formulates a predictive hypothesis h, finds a short description $\vec{x}$ of Bob's play $\vec{y}$, and guesses an extension $\vec{s}$ allowing her to predict Bob's moves, Alice will match and take Bob's pennies.

3.3. **Separating regularity and randomness**

The essence of Alice's testing strategy is to separate a regular component of Bob's strategy from the random component. If Bob plays completely randomly, his play $\vec{y}$ will not have a short description, and Alice will not find a hypothesis h that $\vec{y}$ will satisfy. The regular component is then empty. If Alice finds a hypothesis h and a short description $\vec{x}$ of $\vec{y}$ proving that $\vec{y}$ satisfies h, then h captures some of the regularity of $\vec{y}$. If there is still some regularity in $\vec{x}$, then it has a still shorter description $\vec{x}'$, realized using a hypothesis h'. In other words, $\vec{x} = h'(\vec{x}')$ and there is m' such that $\ell(\vec{x}') + m' < \ell(\vec{x})$. But this means that $\vec{x}'$ is a still shorter description of Bob's play $\vec{y}$, showing that it satisfies the hypothesis $h \circ h'$ at the regularity level $m + m'$, since $h \circ h'(\vec{x}') = \vec{y}$ and $\ell(\vec{x}') + m + m' < \ell(\vec{y})$.

On the other hand, if $\vec{x}$ is incompressible, then it is random. In that case, the short description $\vec{x}$ is the random component of Bob's play $\vec{y}$, whereas all of its regularity

is captured by h. Alice can thus extrapoloate Bob's future moves by running h. She also has to expand the random part $\vec{x}$ by some additional randomness $\vec{s}$, as presumably Bob will continue doing. In that sense, Alice still has to gamble. But just like $\vec{x}$ is much shorter than $h(\vec{x}) = \vec{y}$, the chance of guessing $\vec{x} :: \vec{s}$ is greater than the chance of guessing $h(\vec{x} :: \vec{s})$. So separating out the regular component h of Bob's play and reducing the randomness of Bob's play to a significantly shorter description presents a significant advantage for Alice.

Since $\sigma_h(h(\vec{x})) \geq \ell(\vec{x})$, regularity increases with length, and the testing outcomes become more significant, and provide better fitting predictions. On the other hand, longer strings also fit more hypotheses, and the usual problems of *overfitting* in statistical inference enter scene. But testing hypotheses as $\mathcal{L}$-programmable functions turns out to have a special feature, which we consider next.

3.4. **Universal hypothesis**

The main remaining question is: *How should Alice choose her hypotheses?* She can, of course, stares at $\vec{y}$ and search for a pattern. She can try a hypothesis $h^{(1)}$, and if it gets rejected, she can try $h^{(2)}$, and $h^{(3)}$, and so on. But which one should she try first? Occam suggests: *The simplest hypotheses should be tried first*. But which ones are the simplest? Solomonoff and Kolmogorov suggest: *The simplest functions are those that have the shortest programs* [20, 31, 39].

This is where Alice comes to use the fact that her hypotheses are programmable. By enumerating all programs, she can in principle test all hypotheses. If the universal evaluator $\mathcal{L}$ is $\mathcal{L}$-programmable itself, she can in fact test a *universal hypothesis*. The idea of a universal randomness test goes back to Per Martin-Löf [21]. Since it will eventually detect any regularity, any universal hypothesis test is in fact also a universal randomness test, as random strings can be characterized as just those that pass all tests [22].

Definition 3.4. *A hypothesis* $\upsilon : \mathbb{2}^* \rightharpoonup \mathbb{2}^*$ *is* universal *if any string that is regular with respect to any hypothesis* h *is also regular with respect to* υ. *More precisely, for every hypothesis* $h : \mathbb{2}^* \rightharpoonup \mathbb{2}^*$ *there is a constant* c_h *for which every bitstring* $\vec{x}$ *satisfies*

$$\sigma_h(\vec{x}) \quad \leq \quad c_h + \sigma_\upsilon(\vec{x}). \tag{3.8}$$

Proposition 3.2. *If the universal evaluator of a family of* $\mathcal{L}$*-programmable functions is* $\mathcal{L}$*-programmable itself, then the family also contains a universal hypothesis.*

The assumption that the universal evaluator $\mathcal{L}$ is $\mathcal{L}$-programmable is satisfied not just when $\mathcal{L}$ interprets a Turing complete language, but also when it is resticted to a complexity class with complete instances.

Proof. Let $\{\} : 2^* \times 2^* \rightharpoonup 2^*$ be a universal evaluator. Recall that this means that for every computable function $f : 2^* \rightharpoonup 2^*$ there is a program $\vec{p}_f$ such that $f(\vec{x}) = \{\vec{p}_f\}\vec{x}$. Define

$$\mathfrak{v}(\vec{x}) \;=\; \begin{cases} \left\{\vec{x}_{(0)}\right\}\vec{x}_{(1)} & \text{if } \ell(\vec{x}) < \ell\left(\left\{\vec{x}_{(0)}\right\}\vec{x}_{(1)}\right) \\ \uparrow & \text{otherwise.} \end{cases} \tag{3.9}$$

Then $\mathfrak{v} : 2^* \rightharpoonup 2^*$ is a hypothesis by Def. 3.1. Any hypothesis $h : 2^* \rightharpoonup 2^*$ and any bitstring $\vec{x}$ satisfy

$$h(\vec{x}) \quad = \quad \mathfrak{v}\left(\langle \vec{p}_h, \vec{x}\rangle\right)$$

for a program $\vec{p}_h$ encoding h. For $c_h = 2\ell\left(\vec{p}_h\right) + 2$, we have the bound

$$c_h + \ell(\vec{x}) + m \;\leq\; \ell\left(\mathfrak{v}\left(\langle \vec{p}_h, \vec{x}\rangle\right)\right) \;=\; \ell\left(h(\vec{x})\right) \tag{3.10}$$

The constants 2 in c_h come from the particular definition of pairing and length that we have chosen in (2.2) and (2.3). But (3.10) means that $\mathfrak{v}$-regularity at the level m implies h-regularity at the level $m + c_h$, i.e. $U^n_m \subseteq H^n_{m+c_h}$. Hence (3.8). □

3.5. **Alice's universal strategy**

If Bob's play $\vec{y}$ is not random, then finding an $\vec{x}$ such that $\mathfrak{v}\left(\vec{x}\right) = \vec{y}$ will separate the regular component $\vec{x}_{(0)}$ and the random component $\vec{x}_{(1)}$ from his play $\vec{y}$, as explained in Sec. 3.3. Guessing Bob's moves $\vec{b}$ can then be reduced to the task of guessing a shorter random string $\vec{s}$ such that $\vec{y} :: \vec{b} = \mathfrak{v}\left(\vec{x} :: \vec{s}\right)$.

In summary, Alice's tasks are similar to her testing strategy: the **first task** is to search for inverse images along a programmable function, this time $\mathfrak{v} : 2^* \longrightarrow 2^*$; and her **second task** is again to use the detected regularity of $\vec{y}$ to predict an extension.

Concerning the first task, note that Alice will stall if she simply lists a sequence of candidates $\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}, \ldots$ and tries to compute $\mathfrak{v}\left(\vec{x}^{(i)}\right) = \left\{\vec{x}^{(i)}_{(0)}\right\}\vec{x}^{(i)}_{(1)}$ for $i = 1, 2, 3, \ldots$, one after another seeking to find an inverse image of $\vec{y}$ along $\mathfrak{v}$. That strategy will only go as far as the first $\vec{x}^{(i)}$ for which the program $\vec{x}^{(i)}_0$ diverges on the input $\vec{x}^{(i)}_1$; the next candidate will never be tested. To avoid that, the search for short descriptions $\vec{x}$ must proceed by *dovetailing*, as described e.g. in [42]. This means that the search through the sequence $\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}, \ldots$ should run a finite number of steps of each computation in a finite initial segment of the sequence, and keep extending that initial segment. In that way, each member of the sequence will eventually be reached and run. E.g., a single step of each of the computations $\mathfrak{v}\left(\vec{x}^{(i)}\right)$ can be run in following order:

$$\vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}, \vec{x}^{(1)}, \vec{x}^{(2)}, \vec{x}^{(3)}, \vec{x}^{(4)}, \vec{x}^{(1)}, \ldots$$

Once Bob's play $\vec{y}$ has been captured by a short description $\vec{x}$, i.e. decomposed in the form $\vec{y} = \left\{\vec{x}_{(0)}\right\}\vec{x}_{(1)}$, where $\vec{x}_{(0)}$ is the regular component of $\vec{y}$, and $\vec{x}_{(1)}$ is its random component, then Alice can proceed with the second task.

In summary, Alice's universal strategy can be described as the search for the earliest bitstring $\vec{b}$ which results from a shorest extension $\vec{s}$ of a shortest inverse image $\vec{x}$ of $\vec{y}$ along υ. This can be summarized as the function $\alpha : \mathbb{2}^* \multimap \mathbb{2}^*$ where

$$\begin{array}{rcl}
\alpha(\vec{y}) & = & \mu\vec{b}.\, A\left(\vec{y},\vec{b}\right) \\
 & & \text{where} \\
A\left(\vec{y},\vec{b}\right) & \iff & \exists\vec{x}\vec{s}.\, \widetilde{A}\left(\vec{x},\vec{y},\vec{s},\vec{b}\right) \wedge \\
 & & \forall\vec{x}'\vec{s}'.\, \left(\widetilde{A}\left(\vec{x}',\vec{y},\vec{s}',\vec{b}\right) \Rightarrow \right. \\
 & & \left. \ell\left(\vec{x}\right) \leq \ell\left(\vec{x}'\right) \wedge \ell\left(\vec{s}\right) \leq \ell\left(\vec{s}'\right)\right) \\
 & & \text{where} \\
\widetilde{A}\left(\vec{x},\vec{y},\vec{s},\vec{b}\right) & \iff & \upsilon\left(\vec{x}\right) = \vec{y} \wedge \left\{\vec{x}_{(0)}\right\}\left(\vec{x}_{(1)} :: \vec{s}\right) = \vec{y} :: \vec{b}.
\end{array}$$

While Alice thus seeks to predict Bob's moves $\vec{b}$ in order to play the same moves, Bob's universal strategy would be dual, in the sense that he would seek to predict Alice's moves $\vec{a}$ in order to play the opposite moves. We discuss below what happens if two universal strategies are played against each other.

4. Matching Pennies randomness

The notion of randomness as incompressibility, as formalized by Kolmogorov [18] and developed in algorithmic information theory [20], has been justified by Martin-Löf's proof that incompressible strings are just those that pass all randomness tests [20, 21, 24]. But we have seen that randomness tests are also a part of playing Matching Pennies. The players stay at the equilibrium only as long as their plays pass each other's tests. Whenever a test produces a significant outcome, the randomness hypothesis is rejected, and the players depart from the equilibrium, whether the detected pattern was a real consequence of someone's earlier deviation from the equilibrium, or whether the test overfitted a pattern onto an actually random string. The equilibrium persists only if both players' plays pass both players' tests.

Corollary 4.1. *A bitstring is uniformly random (in the sense of Kolmogorov [18, 21]) if and only if it can occur as a play of the equilibrium strategy in the game of Matching Pennies.*

Proof. If a bitstring is uniformly random, then it will pass every randomness test, and can occur as an equilibrium strategy. If a bitstring can occur in an equilibrium strategy, and thus passes every randomness test, then it is uniformly random. □

The upshot of this corollary is that randomness tests are an important aspect of the actual process of gaming, yet they are generally abstracted away from game theory. When randomness is taken for granted, the computational content of equilibrium constructions are abstracted away from game theoretic analyses, while the competitive aspects of gaming, of course, essentially depend on using randomness, and recognizing non-randomness. Taking the randomness *testing* for granted hides from sight the whole wide area of players' strategic analyses of each other's plays, which is where the essence of real gaming is played out. If Alice's play passes Bob's tests, but Alice's tests detect the regularity behind Bob's play, then Alice will win by outsmarting Bob. Randomness and outsmarting are two sides of the same coin. Taking one for granted hides the other one from sight, and separates game theory from practice.

While the concept of randomness in the above statements largely follows the approach and the ideas of Martin-Löf [6, 20, 21, 24], the abstract view of computation [27, 29], although lurking in the background in this extended abstract, allows a broader approach. When $\mathcal{L}$ is a Turing complete language, and testing is computable, then Prop. 3.2 implies that there is a universal strategy, and Prop. 4.1 thus says that a bitstring is uniformly random if it does not lose the game of Matching Pennies against the universal strategy. Using weaker programming languages $\mathcal{L}$, and thus specifying weaker randomness tests, yields weaker notions of randomness. A path towards a taxonomy of different notions of randomness obtained in this way is discussed in [24]. The point here is that all such notions can be cast in terms of games. A different approach to a similar idea has been pursued to a much greater depth in [33].

5. General randomness testing

There are many games with nonuniformly distributed mixed strategy equilibria. They require nonuniform randomness testing, more general than described in Sec. 3.5. Many familiar games can be used to motivate it. We describe a new variation on the theme of Matching Pennies. It has infinitely many mixed state equilibria, and the players must test randomness with respect to for multiple distributions, as they are seeking the equilibrium.

Moreover, we shall see in the next section that this game seems to allow a *spooky strategic interaction at a distance*[1] among the members of a team capable of sharing quantum effects — without disturbing the external randomness of their plays.

[1] Einstein famously referred to quantum entanglement itself as "spooky action at a distance" [7, Letter of March 3, 1947].

5.1. **Coordinating Pennies**

Let us consider a version of Matching Pennies played by Alice and Bob against Clare and Dave. Alice and Bob play together, but they are not allowed to communicate; Carol and Dave play together, and cannot communicate either. The task for all of them is to coordinate without communicating. That is why we call the game Coordinating Pennies. Just like Matching Pennies, the game is repeated, and played long enough for the frequencies to settle and the advantages to play out.

Carol and Dave display their two coins first, both of them at the same time. They may agree in advance about the frequencies with which they display their coins, but during the game they must sample their moves independently of one another. Alice only sees Carol's coin before her own move, and Bob only sees Dave's coin before his. All moves are public after they are made. If Carol and Dave have displayed 11, then Alice and Bob win by displaying 01 or 10. Otherwise, Alice and Bob win by displaying 00 or 11. Otherwise they lose. The game is zero sum again, i.e. Carol and Dave win whatever Alice and Bob lose, and vice versa. The payoff matrix is on Table 2. To determine their strategies

	00,01,10	11
00,11	-1 1	1 -1
01,10	1 -1	-1 1

TABLE 2. Payoffs for Coordinating Pennies

- Carol and Dave need to determine together
  - the frequency c with which Carol should play 1, and
  - the frequency d with which Dave should play 1,
- Alice and Bob need to determine together
  - the frequencies a_i for $i = 0$ or 1, with which Alice should play 1 when she sees Carol's move i,
  - the frequencies b_i for $i = 0$ or 1, with which Bob should play 1 when he sees Dave's move i.

Denoting by $q_{11} = cd$ the frequency with which Carol and Dave play 11, Alice and Bob's expected payoffs are $q_{11} - (1 - q_{11})$ if they play different sides of their coins, and $(1 - q_{11}) - q_{11}$ if they play the same sides. By reasoning just like in Sec. 1.2 for Matching Pennies, Carol and Dave can thus conclude that they must play $q_{11} = cd = \frac{1}{2}$, or else Alice and Bob can gain a positive expected payoff by

settling on a pure strategy. Carol and Dave must thus play

$$c = \frac{1}{2d} \quad \text{for} \quad d \in \left[\frac{1}{2}, 1\right]. \tag{5.1}$$

The frequencies of Carol's and Dave's joint moves will thus be as in Table 3, where q_{ij} is the probability that they will play ij.

q_{00}	q_{01}	q_{10}	q_{11}
$-d+\frac{3}{2}-\frac{1}{2d}$	$d-\frac{1}{2}$	$\frac{1}{2d}-\frac{1}{2}$	$\frac{1}{2}$

TABLE 3. Carol's and Dave's joint frequencies

On the other hand, Alice and Bob's expected payoff E_{11} when Carol and Dave play 11 is $a_1(1-b_1)+(1-a_1)b_1-a_1b_1-(1-a_1)(1-b_1)$, whereas for $ij=00,01$ or 10 the expected payoffs E_{ij} are $a_ib_j+(1-a_i)(1-b_j)-a_i(1-b_j)-(1-a_i)b_j$, which all together simplifies to

$$\begin{aligned} E_{ij} &= (2a_i-1)(2b_j-1) \text{ for } ij=00,01,10 \\ E_{11} &= -(2a_1-1)(2b_1-1). \end{aligned}$$

Carol and Dave can now maximize their payoffs by minimizing Alice and Bob's expected payoff

$$E = \sum_{i,j\in\mathbb{2}} q_{ij}E_{ij} \tag{5.2}$$

and derive that they should play

$$d = \frac{1}{2}\sqrt{2\cdot\frac{a_1-a_0}{b_1-b_0}\cdot\frac{2a_0-1}{2b_0-1}}. \tag{5.3}$$

Extracting an additional dependency from Carol and Dave's incentive for mixing, and substituting it all into (5.2) shows that there are infinitely many mixed strategy equilibria, all with the expected payoff 0. For each mixture, each team has to keep testing the randomness of the other team's plays.

This gives rise to the task of testing the randomness of the moves distributed according to arbitrary distributions, not necessarily uniform.

5.2. **Testing P-hypotheses and P-randomness**

Any probability distribution over a finite set can arise as a mixed strategy equilibrium for suitable payoffs. Moreover, iterated games and games with changing payoffs induce in the same way a family of stochastic processes that also arise in computation and cryptography. Their randomness can be defined in terms of mixed strategy equilibria again.

Definition 5.1. *A* string distribution *is an $\mathcal{L}$-programmable*[2] *function $P : \mathbb{2}^* \longrightarrow [0,1]$ such that*

$$P() = 1 \qquad\qquad P(\vec{x}) = P(\vec{x} :: 0) + P(\vec{x} :: 1).$$

Definition 5.2. *Given a string distribution $P : \mathbb{2}^* \longrightarrow [0,1]$, a P-hypothesis with respect to $\mathcal{L}$ is an $\mathcal{L}$-function $h_P : \mathbb{2}^* \rightharpoonup \mathbb{2}^*$ such that*

$$h_P(\vec{x}) = \vec{y} \implies \ell(\vec{x}) < \ell(\vec{y}) \wedge P(\vec{x}) > P(\vec{y}). \tag{5.4}$$

*A bitstring $\vec{y}$ is said to be h_p-*regular *at the level $m \in \mathbb{N}$ whenever*

$$\exists \vec{x}. \qquad h_P(\vec{x}) = \vec{y} \quad \wedge \quad \ell(\vec{x}) + m \leq \ell(\vec{y}) \quad \wedge \quad P(\vec{x}) \geq 2^m \cdot P(\vec{y}). \tag{5.5}$$

*The h_P-regular bitstrings at each level form the h_P-*regularity sets

$$H_m^n = \big\{\vec{y} \in \mathbb{2}^n \mid \exists \vec{x}.\, h_P(\vec{x}) = \vec{y} \wedge \\ \wedge\ \ell(\vec{x}) + m \leq \ell(\vec{y}) \wedge P(\vec{x}) \geq 2^m \cdot P(\vec{y})\big\}. \tag{5.6}$$

The sets $H_m = \bigcup_{n=1}^{\infty} H_m^n$ form the h_p-test

$$H_1 \supseteq H_2 \supseteq H_3 \supseteq \cdots \supseteq H_m \supseteq \cdots$$

Proposition 5.1. *The P-size of h_P-regularity sets decreases exponentially with m*

$$\sum_{\vec{y} \in H_m^n} P(\vec{y}) \quad < \quad 2^{1-m}. \tag{5.7}$$

Proof. By definition of H_m^n, for every $\vec{y} \in H_m^n$ there is $\vec{x}$ of length at most $n-m$ such that $P(\vec{y}) \leq 2^{-m} P(\vec{x})$. It follows that

$$\sum_{\vec{y} \in H_m^n} P(\vec{y}) \leq \sum_{\vec{x} \in 2^{n-m}} 2^{-m} \cdot P(\vec{x}) \ < \ 2^{1-m}.$$

□

The search for non-random patterns, deviating from a given string distribution P, proceeds just like the search for patterns deviating from the uniform distribution in Sec. 3.2. When $\mathcal{L}$ is the family of all computable functions, there is a universal P-hypothesis, defined just like in Def. 3.4.

Proposition 5.2. *If the universal evaluator $\mathcal{L}$ and the string distribution P are $\mathcal{L}$-programmable, then there is an $\mathcal{L}$-programmable universal P-hypothesis as well.*

[2]The programmability of a real function P can be defined in different ways. The idea going back to Turing [35] is to present P as a program $\vec{p}_P : \mathbb{2}^* \longrightarrow \mathbb{2}^*$ which for each $\vec{x}$ outputs a program $\vec{p}_P(\vec{x})$ which streams the digits of the real number $P(\vec{x}) \in [0,1]$.

Proof. The universal P-hypothesis is this time

$$\upsilon_P(\vec{x}) \;=\; \begin{cases} \left\{\vec{x}_{(0)}\right\}\vec{x}_{(1)} & \text{if} \quad \ell(\vec{x}) < \ell\left(\left\{\vec{x}_{(0)}\right\}\vec{x}_{(1)}\right) \\ & \text{and} \;\; P(\vec{x}) > 2^{2\ell\left(\vec{x}_{(0)}\right)+2}\cdot P\left(\left\{\vec{x}_{(0)}\right\}\vec{x}_{(1)}\right) \\ \uparrow & \text{otherwise,} \end{cases} \tag{5.8}$$

where the 2s come from (2.2) and (2.3) again. For a P-hypothesis h_P, some program $\vec{p}_h$ for it, and the length $c_h = 2\ell(\vec{p}_h)+2$ again, this time we have

$$\begin{aligned} c_h + m + \ell(\vec{x}) \;&\leq\; \ell\left(\upsilon_P\left(\langle \vec{p}_h, \vec{x}\rangle\right)\right) \;=\; \ell\left(h_P(\vec{x})\right) \\ P(\vec{x}) \;&>\; 2^m \cdot P\left(\upsilon\left(\langle \vec{p}_h, \vec{x}\rangle\right)\right) \;\geq\; 2^{c_h+m}\cdot P\left(h_P(\vec{x})\right) \end{aligned}$$

which gives $U_m^n \subseteq H_{c_h+m}^n$. By Def. 3.4, this means that υ_P is universal for all P-hypotheses h_P. □

Corollary 5.1. *A bitstring is P-random if and only if it can occur as a play in a game where the string distribution P is a component of a mixed strategy equilibrium.*

5.3. **Universal strategies beyond Matching Pennies**

Just like a universal hypothesis can be used to build a universal strategy for winning, if possible, in the game of Matching Pennies, universal P-hypotheses can be used to build universal strategies for a large family of games, where mixed strategies are expressed in terms of string distributions. A familiar example of such a game is the iterated version of Prisoners' Dilemma, where dynamically changing mixed strategies, inducing string distributions, have been played in tournaments against each other since the early days. Strategies played in such tournaments are usually described by finite state machines, and thus induce $\mathcal{L}$-programmable distributions where $\mathcal{L}$ is a language generating regular expressions. More powerful languages allow specifying not only more powerful strategies, but also games where the payoff matrices are not necessarily fixed through the iterations of the game, but may also change, in an $\mathcal{L}$-programmable way. The crucial feature that allows analyzing such games are the fixed point constructions, enabled by the universal evaluators.

We assume that the payoffs are public information, and that both Alice and Bob have both computed the equilibrium strategies, and know the distributions P_A and P_B according to which Alice and Bob, respectively, must randomly mix their moves. Alice's first task is thus to program a function $\eta_A : \mathbb{2}^* \longrightarrow \mathbb{2}^*$ to search for short descriptions $\vec{x} = \eta_A(\vec{y})$ of Bob's plays $\vec{y}$, whereas Bob's first task is to program a function $\eta_B : \mathbb{2}^* \longrightarrow \mathbb{2}^*$ to search for short descriptions $\vec{u} = \eta_B(\vec{w})$ of

Alice's plays $\vec{w}$. So they are both looking for a right inverse of the universal P-detector of the opponent's string distribution P, i.e.

$$\upsilon_B \circ \eta_A(\vec{y}) = \vec{y} \qquad\qquad \upsilon_A \circ \eta_B(\vec{w}) \;=\; \vec{w}$$

where we write υ_B to simplify υ_{P_B} and υ_A for υ_{P_A}. Their second tasks will be to program functions $\vartheta_A, \vartheta_B : 2^* \longrightarrow 2^*$ to guess the likely extensions of opponents' plays, i.e.

$$\upsilon_B \circ \vartheta_A \circ \eta_A(\vec{y}) \sqsupset \vec{y} \qquad\qquad \upsilon_A \circ \vartheta_B \circ \eta_B(\vec{w}) \;\sqsupset\; \vec{w}.$$

In summary, Alice's and Bob's tasks are thus to program strategies $\alpha = \vartheta_A \circ \eta_A$ and $\beta = \vartheta_B \circ \eta_B$ with

$$\upsilon_B \circ \alpha(\vec{y}) \sqsupset \vec{y} \qquad\qquad \upsilon_A \circ \beta(\vec{w}) \;\sqsupset\; \vec{w}.$$

Alice's universal strategy described in Sec. 3.5 is an instance of this α.

5.4. **On outsmarting and coordination**

In a zero-sum game like Matching Pennies, Alice and Bob have no incentive to cooperate. Each tries to predict opponent's moves and to hamper the opponent's efforts to predict their own moves. If there is a non-random component of their strategies, then they trying to *outsmart* each other: to design the regular component of the strategy so complex that the opponent will not be able to find it. This situation leads to increase of complexity in evolution of adversary processes.

In a non-zero-sum game like Prisoners' Dilemma, the opponents do have an incentive to cooperate, but if the cooperation is not an equilibrium of the game, then the players have an incentive to defect, and the fascinating problem of trust arises. This makes Prisoners' Dilemma into a subject of many sciences, surely more than any other game. The salient point of universal hypotheses, as tools for separating regular components of strategies from random components, as discussed in Sec. 3.3, is that they provide a framework where the players can apparently go beyond the established Nash equilibria of their games, and deviate from the randomization prescriptions in an apparently coordinated way. The example of Prisoners' Dilemma shows that there are situations where such deviations are beneficial for all. In such situations, the process of randomness testing, described in this paper, seems to provide for the players a framework to *coordinate* their hypotheses towards a *secondary equilibrium construction*: an algorithmic *learning equilibrium*, that the players may be able to jointly construct in a secure and mutually beneficial fashion, after they have jointly performed the usual Nash equilibrium construction[3].

[3]We refer to Nash equilibrium as a *joint* construction in the sense that it depends on all players knowing all players' preferences, deriving everyone's best response strategies, which allows all of them to define the best response profile, and compute the equilibria as its fixed points. It is a joint construction in the sense that all players perform the same computations in parallel; not in the

Without coordinating, the only way Alice and Bob can construct their universal strategies α and β is by exhaustive dovetailed search, as described in Sec. 3.5. If they share the same language $\mathcal{L}$, and use the same universal evaluator for it, then they can *coordinate* a construction their universal strategies as a suitable joint fixed point of the following help functions:

$$\begin{aligned}\Xi_A(p,q,\vec{y}) &= \mu\vec{\xi}.\,\upsilon_B\left(\vec{\xi}\right) \sqsupset \vec{y}\\ \Xi_B(p,q,\vec{w}) &= \mu\vec{\zeta}.\,\upsilon_A\left(\vec{\zeta}\right) \sqsupset \vec{w}.\end{aligned}$$

The players will then construct the *coordinated* help functions

$$\begin{aligned}\widehat{\Xi}_A(p,q,\vec{y}) &= \Xi_A([p]\,q,[q]\,p,\vec{y})\\ \widehat{\Xi}_B(p,q,\vec{w}) &= \Xi_B([p]\,q,[q]\,p,\vec{w}).\end{aligned}$$

6. CONCLUDING REMARKS

The starting point of this paper was the observation that finding and playing one's own strategy is often much easier than recognizing and understanding other players' strategies. In particular, randomizing is much easier than testing randomness. On the other hand, knowing that the opponent keeps an eye on how you play is necessary for the implementations of many equilibrium concepts, usually assumed implicitly. In order to stay at an equilibrium, the players must test each other. But capturing their tests opens an alley towards modeling competition, outsmarting, and deceit, which are prominent in the practice of gaming, but often ignored in game theory. We believe that the tools are readily available to tackle this interesting and important aspect of gaming.

Players' randomness testing of each other's plays turned out to be an intuitive characterization of the notion of randomness. It is perhaps worth emphasizing here that the players with different computational powers recognize different notions of randomness. More precisely, different families of programmable functions $\mathcal{L}$ induce different hypotheses, different tests, different notions of randomness, and different implementations for the mixed strategy equilibria. Restricting the family $\mathcal{L}$ to the language of regular expressions or finite state machines would give a weak but interesting notion. The hypotheses could be implemented along the lines of the familiar compression algorithms, such as those due to Lempel, Ziv and Welch [40,41]. However, since there is no such thing as a universal finite state machine, capable of evaluating all finite state machines, such tests based on regular languages would have to be specified one at a time, and sought ad hoc. In contrast,

---

sense that there is any communication or coordination between them. The construction of learning equilibrium, presented here, has the same character.

taking $\mathcal{L}$ to be a Turing complete language, such as the language of Turing machines themselves, allows constructing a universal randomness test, which Alice could implement as a universal hypothesis from Sec. 3.4. This leads to the canonical notion of randomness spelled out by Kolmogorov, Martin-Löf and Solovay, and characterized in Corollary 4.1. Although the simple dovetailing technique used to construct the universal hypothesis quickly leads beyond the realm of what is considered feasible computation, the methods of algorithmic learning and statistical induction are built upon them nevertheless [16, 31, 39]. The idea that randomness testing can be used to construct a learning equilibrium, sketched in Sec. 5.4, suggests an alternative interpretation of the presented results. It is not an afterthought, but predates the present paper, and ties its threads in a different way. Whether it is another conceptual link or a different kind of computational knot remains to be seen.

REFERENCES

[1] Samson Abramsky and Viktor Winschel. Coalgebraic analysis of subgame-perfect equilibria in infinite games without discounting. *Mathematical Structures in Computer Science*, 27(5):751–761, 2017.

[2] Laurent Bienvenu, Glenn Shafer, and Alexander Shen. On the history of martingales in the study of randomness. *Electronic Journal for History of Probability and Statistics*, 5(1):1–40, June 2009.

[3] Lenore Blum and Manuel Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28(2):125–155, 1975.

[4] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall mathematics series. Dover Publications, Incorporated, 2012.

[5] Martin Davis, editor. *The Undecidable : Basic papers on undecidable propositions, unsolvable problems, and computable functions*. Raven Press, Helwett, N.Y., 1965.

[6] Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic Randomness and Complexity*. Theory and Applications of Computability. Springer New York, 2010.

[7] Albert Einstein and Hedwig Born. *The Born-Einstein letters 1916–1955*. Walker, New York, 1971.

[8] Ronald A. Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, Edinburgh, 1925.

[9] Ronald A. Fisher. *Statistical Methods and Scientific Inference*. Oliver and Boyd, Edinburgh, UK, second edition, 1959.

[10] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[11] Joseph Y. Halpern and Rafael Pass. Game theory with costly computation: Formulation and application to protocol security. In Andrew Chi-Chih Yao, editor, *ICS*, pages 120–142. Tsinghua University Press, 2010.

[12] Joseph Y. Halpern and Rafael Pass. Sequential equilibrium in computational games. In Francesca Rossi, editor, *IJCAI*. IJCAI/AAAI, 2013.

[13] Joseph Y. Halpern, Rafael Pass, and Lior Seeman. I'm Doing as Well as I Can: Modeling People as Rational Finite Automata. In Jörg Hoffmann and Bart Selman, editors, *AAAI*. AAAI Press, 2012.

[14] Jules Hedges, Paulo Oliva, Evguenia Shprits, Viktor Winschel, and Philipp Zahn. Selection equilibria of higher-order games. In Yuliya Lierler and Walid Taha, editors, *Practical Aspects*

of Declarative Languages - 19th International Symposium, PADL 2017, Paris, France, January 16-17, 2017, Proceedings, volume 10137 of *Lecture Notes in Computer Science*, pages 136–151. Springer, 2017.

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

[16] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer-Verlag, Berlin, Heidelberg, 2010.

[17] Vicki Knoblauch. Computable strategies for repeated prisoner's dilemma. *Games and Economic Behavior*, 7(3):381–389, November 1994.

[18] Andrey N. Kolmogorov. On tables of random numbers (Reprinted from "Sankhya: The Indian Journal of Statistics", Series A, Vol. 25 Part 4, 1963). *Theor. Comput. Sci.*, 207(2):387–395, 1998.

[19] Pierre-Simon Laplace. *Essai Philosophique sur les Probabilités*. H. Remy, Brussels, 1829.

[20] Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications (2. ed.)*. Graduate texts in computer science. Springer, 1997.

[21] Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966.

[22] Richard von Mises. *Probability, Statistics, and Truth*. Dover Books on Mathematics Series. Dover Publications, 1957.

[23] John H Nachbar and William R Zame. Non-computable strategies and discounted repeated games. *Economic Theory*, 8(1):103–22, June 1996.

[24] Andre Nies. *Computability and Randomness*. Oxford University Press, Inc., New York, NY, USA, 2009.

[25] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[26] Dusko Pavlovic. A semantical approach to equilibria and rationality. In Alexander Kurz and Andzrej Tarlecki, editors, *Proceedings of CALCO 2009*, volume 5728 of *Lecture Notes in Computer Science*, pages 317–334. Springer Verlag, 2009. arxiv.org:0905.3548.

[27] Dusko Pavlovic. Monoidal computer I: Basic computability by string diagrams. *Information and Computation*, 226:94–116, 2013. arxiv:1208.5205.

[28] Dusko Pavlovic. *Programs as Diagrams: From Categorical Computability to Computable Categories*. Theory and Applications of Computability. Springer, September 2023.

[29] Dusko Pavlovic and Muzamil Yahia. Monoidal computer III: A coalgebraic view of computability and complexity. In C. Crîstea, editor, *Coalgebraic Methods in Computer Science (CMCS) 2018 — Selected Papers*, volume 11202 of *Lecture Notes in Computer Science*, pages 167–189. Springer, 2018. https://arxiv.org/abs/1704.04882.

[30] Michael O Rabin. Effective computability of winning strategies. *Contributions to the Theory of Games*, 3(39):147–157, 1957.

[31] Jorma Rissanen. *Information and Complexity in Statistical Modeling*. Information science and statistics. Springer, New York, 2007.

[32] Ariel Rubinstein. *Modeling Bounded Rationality*. Zeuthen lecture book series. MIT Press, 1998.

[33] Glen Shafer and Vladimir Vovk. *Probability and Finance: It's Only a Game!* Wiley Series in Probability and Statistics. Wiley, 2005.

[34] Ray J. Solomonoff. A formal theory of inductive inference. Part I., Part II. *Information and Control*, 7:1–22, 224–254, 1964.

[35] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society. Second Series*, 42:230–265, 1936. Reprinted in [5].

[36] Jean-André Ville. Sur la notion de collectif. *Comptes Rendus*, 203:26–27, July 1936.

[37] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
[38] Vladimir Vovk and Dusko Pavlovic. Universal probability-free prediction. *Ann. Math. Artif. Intell.*, 81(1-2):47–70, 2017. arxiv.org:1603.04283.
[39] C.S. Wallace. *Statistical and Inductive Inference by Minimum Message Length*. Information Science and Statistics. Springer, 2005.
[40] Terry A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.
[41] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theor.*, 24(5):530–536, September 2006.
[42] A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the algorithmic concepts of information and randomness. *Russian Math. Surveys*, 25(6):83–124, 1970.

(Received:)
(Revised:)

Dusko Pavlovic
University of Hawaii
e-mail: *dusko@hawaii.edu*
and
Peter-Michael Seidel
University of Hawaii
e-mail: *pseidel@hawaii.edu*
and
Muzamil Yahia
University of Hawaii
e-mail: *muzamil@hawaii.edu*