Information and Computation 300 (2024) 105196

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

Secretive interaction Players and strategies

Eduardo Skapinakis [a,b,*]

[a] Center for Mathematics and Applications (NOVA Math), NOVA School of Science and Technology (NOVA FCT), Portugal
[b] Carl Friedrich von Weizsäcker-Zentrum, Universität Tübingen, Germany

ARTICLE INFO

Article history:
Received 27 October 2023
Received in revised form 7 May 2024
Accepted 16 July 2024
Available online 19 July 2024

Keywords:
Inductive inference
Interactive learning
Secrecy
Learning strategies

ABSTRACT

We study the behaviour of n players engaged in a conversation, each attempting to predict the behaviour of the others. We prove that, under a certain notion of interaction, it is possible to learn all other players, but, generally, not without being learned back. We then focus on the interaction of two players and consider, for different methodological restrictions based on the philosophy of science, whether some form of secretive learning can be achieved.

© 2024 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction and motivation

Learning theory emerged in the 1960s [32,37,23,7] and studies the process of an abstract scientist mapping evidence into hypotheses. Fix a set of objects, an environment through which data from the objects can be transmitted, conditions that a scientist must meet and a criterion for successful learning. We are interested in the question "can the scientist identify an object after a finite amount of data?" [30,18].

We will concern ourselves with recursive scientists attempting to discover (possibly different) codes of recursive functions, a learning criterion introduced by Bārzdiņš [19,5] and Case and Smith [9,11].

The learning theory of identification descending from Gold [23] does not take into account the possibility that the object being learned interacts with the learner. Montagna and Osherson [29] thus introduced a learning paradigm aimed at coordination between two players, where each attempts to predict the behaviour of the other, using only the history of earlier predictions (see also Case, Jain, Montagna, Simi and Sorbi [14]).

This work extends Case and Kötzing [16], who considered the additional task of predicting the behaviour of the other player, without having our own behaviour predicted. We consider the question of whether some form of *secretive learning* is possible when there are more than two players interacting and when players are restricted by some form of behaviour rooted in the philosophy of science.

Sample results: coordination of n players is always possible; there is a player that can *protect* others, by learning back any player that learns them; there are consistent players that cannot be secretively learned; simpleminded players can be secretively learned.

* Correspondence to: Center for Mathematics and Applications (NOVA Math), NOVA School of Science and Technology (NOVA FCT), Portugal.
E-mail address: e.scapinakis@campus.fct.unl.pt.

https://doi.org/10.1016/j.ic.2024.105196
0890-5401/© 2024 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

2. Background

Background from *computability* can be found in Rogers [33], from *learning theory* in Osherson, Stob and Weinstein [30] and from *interactive learning* in Montagna and Osherson [29] and Case and Kötzing [16]. We consider partial recursive functions, in the sense of the Church-Turing thesis [35], defined over the set of natural numbers $\mathbb{N} = \{0, 1, \dots\}$. We call recursive functions those that, besides being partial recursive, are total, i.e., are defined for every input. We write $f(x)\downarrow$ to mean that the computation of f on input x halts, and $f(x)\uparrow$ otherwise. From now on, fix some encoding of Turing machines $\{\phi_x : x \in \mathbb{N}\}$ and, for $i \geq 1$, let $\mathtt{R}_i$ be the set of recursive functions with arity i. For a recursive function f and $m \geq 1$, we denote by $f[m]$ the array containing $f(0), \dots, f(m-1)$. Given an array σ, we denote by $\text{last}(\sigma)$ the last entry in σ and by $\sigma(x)$ the xth entry of σ (the first being the 0th). We denote by $\langle x_1, \dots, x_n \rangle$ some standard encoding of $x_1, \dots, x_n$ and let $\langle x_1, \dots, x_n \rangle_i = x_i$, for $i \in \{1, \dots, n\}$.

We study a learning paradigm where $n \geq 2$ players (recursive functions) interact, each producing an output depending on all previous outputs of the other $n-1$ players. We represent this interaction via a *crossfeeding* operator[1]:

$$\begin{aligned} X^n : \mathtt{R}_{n-1}^n &\longrightarrow \mathtt{R}_1^n \\ (h_1, \dots, h_n) &\longmapsto (p_1, \dots, p_n) \end{aligned}$$

where for each $i \in \{1, \dots, n\}$ and $m \geq 1$, $p_i(m) = h_i(p_1[m], \dots, p_{i-1}[m], p_{i+1}[m], \dots, p_n[m])$. For each $m \in \mathbb{N}$, $p_i(m)$ is the conjecture of the *player* h_i when it receives all previous m outputs of all other $n-1$ players. The time when all p_i's have halted on input m is the *stage* m of the interaction. By convention, $p_i(0) = 0$, for each $i \in \{1, \dots, n\}$, representing an output when no data has yet been received. From now on, we will always omit the arity in X.

As players interact with each other, the notion of learnability here is different from that in identification theory, where one is required to reason about the code of the function producing the data received. In this interactive setting, to predict the behaviour of a player is to predict what he will output at each stage m, which, in general, is not what the player outputs when given m as input. In the notation above, this is the difference between being required to conjecture the code of some h_i, or of the function p_i, that represents the outputs of h_i at each stage. The latter is the notion adopted: a player *learns* another if, after a given time, he can predict what the other will output in any future stage of the interaction.

If players learn a subset of the players involved in the interaction, we use indices to identify which ones are being learned. In sentences with multiple occurrences of the word "player", we may use the expression "learner" to refer to a player that is learning another.

The central notion of learnability is the following:

Definition 2.1. Let $h_1, \dots, h_n$ be recursive functions (the *players*) and let $X(h_1, \dots, h_n) = (p_1, \dots, p_n)$. Let $i \in \{1, \dots, n\}$ (the learner) be given and, for $k \in \{1, \dots, n-1\}$, let $\alpha_1, \dots, \alpha_k$ be different elements of $\{1, \dots, n\} \setminus \{i\}$ (the players being learned). We say that h_i $\mathtt{XBc}$*-learns* $h_{\alpha_1}, \dots, h_{\alpha_k}$ if there is $N \in \mathbb{N}$ such that, for all $m \geq N$, $p_i(m) = \langle e_1^m, \dots, e_{n-1}^m \rangle$ and, for each $j \in \{1, \dots, k\}$, if $\alpha_j < i$, $e_{\alpha_j}^m$ is a code for p_{α_j}, and, if $\alpha_j > i$, $e_{\alpha_j - 1}^m$ is a code for p_{α_j}.[2]

The suffix $\mathtt{Bc}$ in $\mathtt{XBc}$ stands for *Behaviourally correct* [19,5,9,11], since, as indicated by the superscript m in the codes e_i^m, the conjectures of the player h_i can change in each stage, as long as they are always correct codes for the target functions.

If the conjectures of h_i, for each target function, stabilize to a single code, we say that h_i $\mathtt{XEx}$*-learns* $h_{\alpha_1}, \dots, h_{\alpha_k}$. The suffix $\mathtt{Ex}$ stands for *Explanatory* [23], as then h_i is required to give a single *explanation* for the data received. If the conjectures of h_i oscillate between a finite group of correct codes, we say that h_i $\mathtt{XFEx}$*-learns* $h_{\alpha_1}, \dots, h_{\alpha_k}$. The suffix $\mathtt{FEx}$ stands for *Vacillatory* [3,13], as h_i vacillates between different codes.

We say that a group of players *coordinate* if each of them learns all others, for any of the learning criteria mentioned above. We add the suffix "$\mathtt{s}$" to a learning criterion when a player learns the others without being learned back. We may write the name of the players together with the learning criterion to emphasise the players involved. For example, "h_1 $\mathtt{XBcs}(h_1, \dots, h_n)$-learns h_2" means that, when $h_1, \dots, h_n$ interact, h_1 $\mathtt{XBc}$-learns h_2, but h_2 does not $\mathtt{XBc}$-learn h_1 back.

There will be two types of simulations of the interaction between the players $h_1, \dots, h_n$. When the goal of the simulation is to return $p_i(m)$, the interaction is simulated until the stage m and $p_i(m)$ is obtained. When the goal of the simulation is to know what $p_i(m)$ would be, to then trick the conjecture of h_i, the simulation is run until the stage $m-1$ (for $m \geq 1$) and the mth stage is simulate with different possible outputs.

The results we present have essentially two forms. We either show that there is a player that can learn back any player that learns him, or that, for any player g with some particular *restriction*, there is another that learns him and is not learned back. For the second type of result, we will construct players which, from the code of g, simulate it on different possible inputs and return one that guarantees that g does not learn them back. Although this is somewhat strange, to say that

[1] The letter X comes from *cross*, as short for *crossfeeding*.

[2] The distinction between $\alpha_j < i$ and $\alpha_j > i$ is required because the index of a player in an interaction is not necessarily the index in which it appears in the conjectures of another player. For example, suppose that h_2 $\mathtt{XBc}(h_1, h_2, h_3)$-*learns* h_1 and h_3. If, for $m \in \mathbb{N}$, $p_2(m) = \langle e_1, e_2 \rangle$, the conjecture for the code of p_1 is e_1, but, for the code of p_3, it is e_2.

g cannot be secretive against some player that already has access to his code, it shows that given strategies cannot work against *every* player.

2.1. Recursion theorems

A beautiful aspect of formal learning theory is that it provides a means to apply recursion theoretic results to concrete scenarios of machines attempting to discover some phenomena [15]. We will require the following tools that we present below: the s-m-n theorem, Kleene's Second Recursion Theorem (KRT), the Extended Recursion Theorem (ERT), the Generalized Delayed Fixed Point Theorem (DFT) and the Operator Recursion Theorem (ORT).

Theorem 2.2 *(s-m-n [28]). For every $n, m \geq 0$, there is a primitive recursive function s_m^{n+m} such that $\phi_{s_m^{n+m}(e,x_1,\ldots,x_m)}(y_1,\ldots,y_n) = \phi_e(x_1,\ldots,x_m,y_1,\ldots,y_n)$.*

The theorem states that, given the code of a function f, one can construct the code of another function that receives fewer inputs and behaves as f, when the missing inputs are specific constants.

We require ERT of Smullyan whenever we are dealing with more than 2 players, since each players' action influences all other players, whose output then influences the original players again.

Theorem 2.3 *(ERT [34]). Let $n > 0$ and let $F_1,\ldots,F_n$ be partial recursive functions of arity $n+1$. Then there are codes $e_1,\ldots,e_n \in \mathbb{N}$ such that, for all $i \in \{1,\ldots,n\}$, $\phi_{e_i}(y) = F_i(e_1,\ldots,e_n,y)$.*

KRT is the special case of ERT with $n = 1$. The previous theorems allow us to take partial recursive functions and construct others that run their own code on the functions we chose. However, we will also want to exhibit some form of behaviour, not on a function, but on the function it conjectures. To reach this form of recursion, we will use a special case of DFT and ORT of Case.

Theorem 2.4 *(DFT [8]). If F is a partial recursive function of arity 3, there is a recursive function ϕ_e such that $\phi_{\phi_e(x)}(y) = F(e,x,y)$.*

ORT applies to recursive operators, which we define as in [8] (see also [33]). A *recursive operator* is a total map Θ, from the set of partial recursive functions to itself, such that there is a recursive function which, when given any enumeration of the graph of a partial recursive function f, outputs an enumeration of the graph of the function $\Theta(f)$.

Since we can always reduce greater arities, by considering functions that interpret an input as the encoding of two, we state the result as it is originally formulated.

Theorem 2.5 *(ORT [8]). If Θ is a recursive operator mapping functions of arity 1 into functions of arity 2, there is a strictly increasing recursive function h such that $\phi_{h(n)}(x) = \Theta(h)(n,x)$.*

Taking $\Theta(f)(x,n) = f(x)$, for example, we get a function that outputs ever increasing codes for itself. The fact that h can be taken to be strictly increasing guarantees that many players can be protected in the sense of Theorem 3.4.

3. Multi-player interaction

We start by considering $n \geq 2$ players interacting, adapting Theorems 4.1 and 4.3 of [16] to this multi-player scenario. For simplicity, we will always prove the results for $n = 3$ and discuss how they can be adapted to more players.

3.1. Coordination

We show that, given some fixed players, one can define others that learn them all. These players can indeed be constructed to be a constant functions, whence, for suitably chosen players, coordination of all players involved follows. We start with the case where there are only two players involved, to exemplify the structure of the proofs.

Theorem 3.1 *([29,16]). Every player can be XBc-learned.*

Proof. Let g be a player and define H as the function specified in Algorithm 3.1.

By DFT, there is a code e such that $\phi_{\phi_e(\sigma)}(x) = H(e,\sigma,x)$. Define $h = \phi_e$ and let $X(h,g) = (p,q)$. Then, since $h = \phi_e$, the algorithm $H(e,\sigma,x)$ simulates the interaction between h and g until the stage x and outputs $q(x)$. Thus, for every $m \geq 1$,

$$\phi_{p(m)}(x) = \phi_{h(q[m])}(x) = \phi_{\phi_e(q[m])}(x) = H(e,q[m],x) = q(x)$$

whence h XBc-learns g. □

Algorithm 3.1: XBc-learning a player.

Input y, σ, x;
Simulate $X(\phi_y, g) = (p_y, q_y)$ until the stage x;
return $q_y(x)$

This process of, knowing the code of g, simulating the interaction between itself and g, will be a key aspect throughout the rest of the paper and is what requires the use of the recursion theorems.

We show below that, not only can the above result be generalized to $n > 2$ functions, but also that the players designed to learn can be taken to be constant functions.

Theorem 3.2. *For any set of recursive functions $g_1, \ldots, g_k$, there are recursive functions $h_1, \ldots, h_n$ such that each h_i XEx$(h_1, \ldots, h_n, g_1, \ldots, g_k)$-learns all g's and all other h's.*

Proof. We prove the result for $k = 1$ and $n = 2$. Let g be a player and denote by C_y the constant function that only outputs y. For $i \in \{1, 2, 3\}$, define the function H_i as follows:

Algorithm 3.2: Coordinating with other learners.

Input w, y, z, x;
Simulate $X(C_{\langle y,z\rangle}, C_{\langle w,z\rangle}, g) = (p_1, p_2, q)$ until the stage x;
return the ith coordinate of $(p_1(x), p_2(x), q(x))$

Now, from ERT, there are three codes, $e_1, e_2, e_3 \in \mathbb{N}$, such that, for each $i \in \{1, 2, 3\}$, $\phi_{e_i}(x) = H_i(e_1, e_2, e_3, x)$. The codes e_1, e_2 and e_3 will represent the behaviour of the players h_1, h_2 and g, respectively, in the interaction.

Define $h_1 = C_{\langle e_2,e_3\rangle}$ and $h_2 = C_{\langle e_1,e_3\rangle}$ and let $X(h_1, h_2, g) = (p_1, p_2, q)$. We show that h_2 XEx(h_1, h_2, g)-learns h_1, g. For each $m \geq 1$,

$$\phi_{p_2(m)_1}(x) = \phi_{h_2(p_1[m],q[m])_1}(x) = \phi_{\langle e_1,e_3\rangle_1}(x) = \phi_{e_1}(x) = H_1(e_1, e_2, e_3, x)$$

The function $H_1(e_1, e_2, e_3, x)$ then simulates the interaction between $C_{\langle e_2,e_3\rangle}$, $C_{\langle e_1,e_3\rangle}$ and g, until the stage x, and outputs $p_1(x)$, so h_2 XEx(h_1, h_2, g)-learns h_1. Analogously,

$$\phi_{p_2(m)_2}(x) = \phi_{h_2(p_1[m],q[m])_2}(x) = \phi_{\langle e_1,e_3\rangle_2}(x) = \phi_{e_3}(x) = H_3(e_1, e_2, e_3, x) = q(x)$$

So h_2 XEx(h_1, h_2, g)-learns g as well. □

One may note that a different strategy was employed in the two proofs above. In Theorem 3.1 we used DFT to obtain (only) the code of the learner, whereas in Theorem 3.2 we used ERT to obtain three codes, one for each player. The difference is that, while in the first case the learner was constructed to simulate *itself* interacting with g, using this simulation to predict what the behaviour of g would be, in the second case the learners are no longer simulating *themselves*, but rather constant functions, whose outputs should be correct predictions for the behaviour of the other players. This means that the codes for the behaviour of the players being learned have to be determined a priori, which is why three codes as obtained in the proof.

If we wanted to include more players to be learned, we would just have to add them in the cross-feeding operator simulation, since their code is fixed. If we wanted to add more players learning, we would just have to consider constant functions that output encodings of more codes.

The result above shows that coordination is always possible in an "open environment", where everyone can read everyone else's predictions. The rest of the paper regards the notion of being able to learn other players without being learned back.

3.2. Secrecy fails

We prove two results that show that, in general, it is not possible to secretively learn other players: there is a player that learns back any player that learns him (so secretively learning an arbitrary group of players is impossible) and there is a player that "protects" others, by learning back any player that learns them (so, in general, secretively learning any player in an "open environment" is impossible).

Intuitively, secrecy fails because of the nature of *crossfeeding*. If a player h makes a prediction about what g will output in a stage $x \geq 1$, then h must implicitly also predict what it will *itself* output at stage $x - 1$, as the behaviour of g is (or can be made to be) dependent on the behaviour of h. One can then construct a player g which, "removing" his own behaviour from the predictions made by h, uses the conjectures received to learn h back.

Theorem 3.3. *There is a player that cannot be* XBcs*-learned.*

Proof. Let h_1 and h_2 be two players. We will define a recursive function g such that, if h_i XBc(h_1, h_2, g)-learns g, then g XBc(h_1, h_2, g)-learns h_i back. For $i \in \{1, 2\}$, define the recursive operator[3]

$$\Phi_i(f)(x, \sigma_1, \sigma_2) = \text{last}\left(\left(f^{-1}\left(\left(\phi_{(\text{last}(\sigma_i))_2}(x+1)\right)_i\right)\right)_i\right)$$

Then, by ORT, there are strictly increasing functions g_1 and g_2, such that, for each $i \in \{1, 2\}$,

$$\phi_{g_i(\sigma_1,\sigma_2)}(x) = \Phi(g_i)(x, \sigma_1, \sigma_2)$$

Define $g(\sigma_1, \sigma_2) = \langle g_1(\sigma_1, \sigma_2), g_2(\sigma_1, \sigma_2)\rangle$ and let $X(h_1, h_2, g) = (p_1, p_2, q)$.

The inputs σ_1 and σ_2 represent the history of the outputs of p_1 and p_2, respectively, which are given to g as input at each stage. The code conjectured by each h_i, for the behaviour of g, is $(\text{last}(\sigma_i))_2$, so, if h_i correctly learns g, then $\phi_{(\text{last}(\sigma_i))_2}(x+1)$ is the output g gives, when it receives $(p_1(0), \ldots, p_1(x), p_2(0), \ldots, p_2(x))$ as input. The inverse in the operator then represents the act of removing the behaviour of g, which leaves only information from which $p_1(x)$ and $p_2(x)$ can be recovered. In the construction of the operator, the second to last i is used to choose which of the components of the output of g to return (to cancel out with the respective inverse function) and the last one to choose between $p_1(x)$ and $p_2(x)$.

Below is the proof that g cannot be XBcs-learned. In the equalities where only part of the left-hand side is rewritten we use a bold font to emphasize which of its components "triggers" the equality. Suppose that h_1 XBc(h_1, h_2, g)-learns g. Then, there is $N \in \mathbb{N}$ such that, for every $m \geq N$, $\phi_{p_1(m)_2}(x) = q(x)$. Thus, for $m \geq N+1$,

$$\begin{aligned}
\phi_{\boldsymbol{q(m)}_1}(x) &= \phi_{g(p_1[m], p_2[m])_{\mathbf{1}}}(x) = \phi_{g_1(p_1[m], p_2[m])}(x) = \text{last}\left(\left(g_1^{-1}\left(\left(\phi_{\mathbf{(last(}\boldsymbol{p_1}\mathbf{[}\boldsymbol{m}\mathbf{]))_2}}(x+1)\right)_1\right)\right)_1\right) \\
&= \text{last}\left(\left(g_1^{-1}\left(\left(\boldsymbol{\phi_{(p_1(m-1))_2}}(x+1)\right)_1\right)\right)_1\right) = \text{last}\left(\left(g_1^{-1}\left(\left(\boldsymbol{q(x+1)}\right)_1\right)\right)_1\right) \\
&= \text{last}\left(\left(g_1^{-1}\left(\left(g(p_1[x+1], p_2[x+1])\right)_{\mathbf{1}}\right)\right)_1\right) = \text{last}\left(\left(\boldsymbol{g_1^{-1}}\left(\boldsymbol{g_1}(p_1[x+1], p_2[x+1])\right)\right)_1\right) \\
&= \text{last}\left((p_1[x+1], p_2[x+1])_{\mathbf{1}}\right) = \textbf{last}\left(p_1[x+1]\right) = p_1(x)
\end{aligned}$$

Analogously, if h_2 XBc(h_1, h_2, g)-learns g, there is $N \in \mathbb{N}$ such that, for every $m \geq N$, $q(m)_2$ is a code for p_2. □

These results imply that any attempt at secrecy between n players fails whenever one of them is defined as in the proof above. Any function that is required to learned g will be learned back and, once codes for its behaviour are outputted, all other functions will have access to it.

If a player is not defined as in the proof above, one can still show that secrecy cannot always be achieved.

Theorem 3.4. *For every injective player g_1, there is a protector g_2 that* XBc*-learns any player that* XBc*-learns g_1.*

Proof. Let g_1 and h be any players, with g_1 injective. We define g_2 such that, if h XBc(h, g_1, g_2)-learns g_1, then g_2 XBc(h, g_1, g_2)-learns h. Define

$$G(x, \sigma_1, \sigma_2) = \text{last}\left(\left(g_1^{-1}\left(\phi_{(\text{last}(\sigma_1))_1}(x+1)\right)\right)_1\right)$$

Let e be the code of G. Then, by the s-m-n theorem, there is a function s_2^3 such that $\phi_{s_2^3(e,\sigma_1,\sigma_2)}(x) = G(x, \sigma_1, \sigma_2)$. Define $g_2(\sigma_1, \sigma_2) = \langle s_2^3(e, \sigma_1, \sigma_2), 1\rangle$, let $X(h, g_1, g_2) = (p, q_1, q_2)$ and suppose that h XBc(h, g_1, g_2)-learns g_1. Then, there is $N \in \mathbb{N}$ such that, for every $m \geq N$,

$$\begin{aligned}
\phi_{\boldsymbol{q_2(m)}_1}(x) &= \phi_{s_2^3(e, p[m], q_1[m])}(x) = \text{last}\left(\left(g_1^{-1}\left(\boldsymbol{\phi_{(\textbf{last}(\boldsymbol{p}[\boldsymbol{m}]))_1}}(x+1)\right)\right)_1\right) = \text{last}\left(\left(g_1^{-1}\left(\boldsymbol{q_1(x+1)}\right)\right)_1\right) \\
&= \text{last}\left(\left(\boldsymbol{g_1^{-1}}\left(\boldsymbol{g_1}(p[x+1], q_2[x+1])\right)\right)_1\right) = \text{last}\left((p[x+1], q_2[x+1])_{\mathbf{1}}\right) = \textbf{last}(p[x+1]) = p(x) \quad \square
\end{aligned}$$

If, in the statement above, the player g_1 was not injective, then, while the *protector* could still compute the "inverse" of g_1, looking for the smallest input that maps to the correct output, this strategy would no longer guarantee that $p(x)$ is obtained, whenever x is greater than the current stage. Note that, if, for any $y \in \mathbb{N}$, g_1 maps at most a finite number of points to y, then the *protector* can still learn h in some form, by outputting k possible conjectures, one of which must be correct.

[3] The inverse is interpreted as looking for the smallest x such that $f(x) = y$.

A possible way to circumvent the proofs above is by relaxing the definition of XBc-learning, so that players may not have to encode their predictions in a fixed order, or by allowing the predictions to be split between players that can communicate secretly. For the first option, the player constructed in both proofs above fails to learn the target player back, because he does not know which part of the learner's output to use. These ways of attempting secrecy only emerge when there are more than two players interacting.

To conclude this section, consider the set $\mathtt{SD} = \{f \in \mathtt{R}_1 : f(0) \text{ is a code of } f\}$. It appears in Blum and Blum [7], who show that no player can Ex-identify all functions from $\mathtt{SD} \cup \mathtt{AEZ}$, where AEZ is the set of all recursive functions f, for which there is $N \in \mathbb{N}$ such that, for every $n \geq N$, $f(n) = 0$. This *non-union theorem* also holds for Bc-identification [11,5]. As remarked in [29], this implies that no player can XBc-learn every player from $\mathtt{SD} \cup \mathtt{AEZ}$, since, for players that ignore the other players' output, XBc-learning reduces to Bc identification.

We will show that, while there are players that can XBc-learn all players in SD, no player can XBcs-learn them all. The first part is simple: as a players' first input is 0, those in SD will output their own code in stage 1, whence learnability of all players in SD follows from Theorem 3.1, where all that is required is access to the code of the player being learned. For the second part, we will construct a player in SD that cannot be XBcs-learned.

Theorem 3.5. *There is a player in SD that cannot be XBcs-learned.*

Proof. We adapt the construction of the operator from Theorem 3.3. Define

$$\Phi(f)(x,\sigma) = \begin{cases} \sigma(0) & \text{if } x = 0 \\ \text{last}\left(f^{-1}\left(\phi_{\text{last}(\sigma)}(x+1)\right)\right) & \text{otherwise} \end{cases}$$

From ORT, there is a strictly increasing function G such that $\phi_{G(\sigma)}(x) = \Phi(G)(x,\sigma)$. Consider

$$G'(y,\sigma) = \begin{cases} y & \text{if } \sigma = 0 \\ G(\sigma) & \text{otherwise} \end{cases}$$

From KRT, consider the function $g(\sigma) = \phi_e(\sigma) = G'(e,\sigma)$. This is the function from the statement. First, $g(0) = G'(e,0) = e$, so $g \in \mathtt{SD}$. Let h be any learner, let $X(h,g) = (p,q)$ and suppose that h XBc-learns g. Then, there is $N \in \mathbb{N}$ such that, for every $m > N$ (after which $p[m] \neq 0$}), we have

$$\phi_{q(m)}(x) = \phi_{g(p[m])}(x) = \phi_{G'(e,p[m])}(x) = \phi_{G(p[m])}(x) = \Phi(G)(x, p[m])$$

If $x = 0$, we have that $\phi_{q(m)}(0) = p[m](0) = p(0)$, so the prediction made by g is correct. For the remaining points,

$$\phi_{q(m)}(x) = \Phi(G)(x, p[m]) = \text{last}\left(G^{-1}\left(\boldsymbol{\phi_{p(m-1)}}(x+1)\right)\right) = \text{last}\left(G^{-1}\left(\boldsymbol{q(x+1)}\right)\right) = \text{last}\left(G^{-1}\left(g(p[x+1])\right)\right)$$

Since $x \neq 0$, $p[x+1] \neq 0$, so $g(p[x+1]) = G'(e, p[x+1]) = G(p[x+1])$, whence

$$\phi_{q(m)}(x) = \text{last}\left(G^{-1}\left(G(p[x+1])\right)\right) = \text{last}(p[x+1]) = p(x) \qquad \square$$

In the multi-player scenario, this would be a player that "announces" he will make secrecy fail and no other player, even with access to his code, can learn him without being learned back.

4. Special players

Montagna and Osherson [29] consider three types of special players: "forgiving", which coordinate with a player iff they coordinate with any of that player's finite variants, "blind", which ignore the other player's output, and "memory limited", which only remember the last $k \in \mathbb{N}$ previous outputs of the other player. The function g from Theorem 3.3 is an example of a memory limited player, since it only needs the last output from the player learning him. We consider whether some form of secretive learning is possible for different restrictions on the players' behaviour.

4.1. The players

Research on inductive inference often has a close connection with considerations from the philosophy of science [32,23, 5,7,21,25,15,22]. Observing the proof of Theorem 3.3, one can wonder whether, imposing some methodological restriction on the behaviour of a player, he can behave as the function that makes secrecy fail, or whether this restriction implies that he can be secretively learned. We consider some restrictions rooted in philosophical principles of scientific discovery and show that they always entail some loss of "security". Further examples of such *strategies* in learning theory are discussed in Chapter 4 of Osherson, Stob and Weinstein [30].

Consistent players [4,7,38,1] are restricted to only conjecture functions that agree with the data received so far. This seems to be a reasonable restriction, since, in theory, it makes no sense to make conjectures that contradict data we have already

acquired. However, note that the player from Theorem 3.3 is not, in general, consistent, since his conjectures only become correct after he has been learned. In the paradigm of identification, SD is not Ex-identifiable by consistent scientists.

Popperian players [2,7,9,10,12] are restricted to only conjecture total functions. Since we consider that players are recursive functions, we know that, if the learning process is successful, the conjectures will have to represent total functions at some point. This restriction imposes that, also during the learning process, the player only conjectures total functions. This means that a player's conjectures can always be tested against future data, which is in conformity with Popper's refutability principle [31]. Note that, given a Popperian player, one can construct a consistent one, which tests the conjectures for the data that has already appeared and fixes any errors that might appear.

A generalisation of Popperianism is the notion of accountability. *Accountable* players [30] are restricted to conjecture functions which are defined for more points than those that have already been observed. This means that, although they do not need to conjecture total functions, their hypothesis can always be tested against data that has not yet been received.

Finally, for any function f, define $\min(f) = \min\{e : \phi_e = f\}$. The code $\min(f)$ can be regarded as the smallest code for the function f. This is a particular case of a *measure* of the size of a machine, a notion which was introduced by Blum [6]. For b a recursive function, *b-simpleminded* players [26,20,27,17,24] are restricted to converge to codes of the target function p that are bounded by $b(\min(p))$. The idea is thus that a simpleminded player does not learn versions of the target function that are unnecessarily large, which formalizes the heuristic known as the razor of Ockham [36]. In the paradigm of identification, AEZ is not Ex-identifiable by simpleminded scientists.

4.2. Secretive learning

We will consider the special players mentioned above and prove that, for each of them, some form of secretive learning is possible. The idea is to construct learners that explore the players' limitations from identification learning theory, simulating them and outputting different codes for their behaviour, in a way that prevents them from learning the learners back.

Theorem 4.1. *Every consistent player can be XBc-learned by some player he cannot XFEx-learn.*

Proof. The key aspect of the proof is that, if g is a consistent player, he cannot conjecture the same code when given two different inputs, as only one of them can be consistent with his previous conjecture. Then, at each stage, one chooses a set of possible codes of g (different codes for the same function can be obtained, for example, by introducing irrelevant constants in its program) and outputs one that forces g to make a conjecture that still has not appeared.

Let H as the function specified in Algorithm 4.1.

Algorithm 4.1: Secretively XBc-learning a consistent player.

Input: y, σ;
Simulate $X(\phi_y, g)$ until the stage $|\sigma| - 1$;
Let k be the number of different conjectures made by g;
Calculate $k+1$ different codes $e_1, \ldots, e_{k+1}$ for the following program:
"**Input** x;
Simulate $X(\phi_y, g) = (p_y, q_y)$ until the stage x;
return $q_y(x)$";
return the e_i such that $g(p_y[|\sigma|] \cdot e_i)$ is different from all previous conjectures of g

By KRT, there is a code e such that $\phi_e(\sigma) = H(e, \sigma)$. Let $h(\sigma) = \phi_e(\sigma)$ and $X(h, g) = (p, q)$. Then, for each $m \geq 1$,

$$\phi_{p(m)}(x) = \phi_{h(q[m])}(x) = \phi_{\phi_e(q[m])}(x) = \phi_{H(e,q[m])}(x) = q(x)$$

where the last equality holds since $H(e, q[m])$ returns codes for the function that simulates ϕ_e and g until the stage x and returns $q(x)$. This simulation corresponds to the actual interaction between h and g, whence h XBc-learns g.

Moreover, since g is consistent, at any stage $m \geq 1$, the test of $g(p[m-1] \cdot e_i)$ ensures that, whatever it is that g was expecting to receive from h, he will receive something else, whence $q(m)$ must be different from $q(m-1)$. Since h forces g to change his conjectures infinitely often, to a conjecture that has not appeared before, g does not XFEx-learn h. □

We show below that, in general, when learning consistent players, XFEx cannot be replaced with XBc-learning.

Theorem 4.2. *There is a consistent player that cannot be XBcs-learned.*

Proof. We adapt the proof of Theorem 3.3, ensuring the consistency of the player g by using a list containing the history of the interaction. This will imply that the conjectures of g never stabilize, which is why we move from XFEx to XBc-learning. Let h be a player and define the operator

$$\Phi(f)(x,\sigma) = \begin{cases} \sigma(x) & \text{if } x < |\sigma| \\ \text{last}\left(f^{-1}\left(\phi_{\text{last}(\sigma)}(x+1)\right)\right) & \text{otherwise} \end{cases}$$

From ORT, let g be the function such that $\phi_{g(\sigma)}(x) = \Phi(g)(x,\sigma)$ and let $X(h,g) = (p,q)$.

At stage $m \geq 1$, the data already received by g is $p[m] = p(0)\dots p(m-1)$. Then, regardless of whether h has already learned g, we have that, for any $x < m$,

$$\phi_{q(m)}(x) = \phi_{g(p[m])}(x) = p[m](x) = p(x)$$

whence g is a consistent player. Note that, as the list σ changes at each stage, so does the code conjectured by g.

If h XBc-learns g, then eventually the second branch of $\Phi(g)(x,\sigma)$ will always give the correct answer for $p(x)$, whence g also XBc-learns h. □

When we consider accountable players (and thus Popperian players as well) we can employ another strategy for changing the conjectures of the player being learned, by simulating the predictions made by their conjectures and acting differently than predicted.

Theorem 4.3. *Every accountable player can be XFExs-learned.*

Proof. Let g be an accountable player. Then, at each stage, the conjecture of g is a function that is defined for some input that still has not appeared. We will thus construct a function that searches for this input and returns a code of g that is different from the output predicted. Define H as the function specified in Algorithm 4.2.

Algorithm 4.2: XFExs-learning an accountable player.

Input: y, σ;
Calculate two different codes e_1, e_2 for the following program:
"**Input** x;
Simulate $X(\phi_y, g) = (p_y, q_y)$ until the stage x;
return $q_y(x)$";
Simulate $X(\phi_y, g) = (p_y, q_y)$ until the stage $|\sigma| - 1$;
Look for the smallest number $m \geq |\sigma|$ such that $\phi_{q_y(|\sigma|-1)}(m)\downarrow$ and let d be the result;
If $m = |\sigma|$, **return** the e_i such that $e_i \neq d$;
Otherwise, **return** e_1

By KRT, there is a code e such that $h(\sigma) = \phi_e(\sigma) = H(e,\sigma)$. Let $X(h,g) = (p,q)$. Since h oscillates between two conjectures only, both of which are correct, h XFEx-learns g. Indeed, for any $m \geq 1$:

$$\phi_{p(m)}(x) = \phi_{\phi_e(q[m])}(x) = \phi_{H(e,q[m])}(x) = q(x)$$

Consider a stage $m \geq 1$. If $\phi_{q(m-1)}(m)\uparrow$, then, since $p(m)$ gives some output, $q(m-1)$ is a wrong conjecture for the code of p. If $\phi_{q(m-1)}(m)\downarrow$, then $p(m)$ gives an output that is different from $\phi_{q(m-1)}(m)$, so, once again, $q(m-1)$ is wrong. Thus, at each stage $m \geq 1$, h outputs a correct code for q that makes the conjecture $q(m-1)$ wrong, whence h XFExs-learns g.[4] □

We conclude with simpleminded players. Since they are restricted to make conjectures with a bounded size, they inherently perform at most XFEx-learning. The intuition for the possibility of secretive learning is that, since a simpleminded player can only make a finite number of different conjectures, a player can go through all of them and ensure that each of them is wrong.

Theorem 4.4. *Every simpleminded player can be XFExs-learned.*

Proof. For b a recursive function, let g be a b-simpleminded player. Define H as the function specified in Algorithm 4.3.

By KRT, there is a code e such that $h(\sigma) = \phi_e(\sigma) = H(e,\sigma)$. Let $X(h,g) = (p,q)$. We will observe that, for each scenario of how the interaction between h and g may go, there is a player that XFExs-learns g.

Let $m \geq 1$ be a stage in the interaction. In the first block of the algorithm, two correct codes for q are calculated: e_1 and e_2. To explore the simplemindedness of g, $q(m) = g(p[m])$ is compared with $b(e)$, as e is a lower bound for $\min(h)$. The first **if** condition thus ensures that, if it holds, $q(m)$ is *too large* or *new*, so we set $p(m) = e_1$. As g is a b-simpleminded player and the number of available conjectures below $b(e)$ is finite, h will eventually go through all of them. When an old conjecture is discovered, it will then ensure that it is wrong.

[4] Note that, in this case, g does not even XBc-learn h, as its conjectures are wrong at each stage.

Algorithm 4.3: XFExs-learning a simpleminded player.

Input: y, σ;
Calculate two different codes e_1, e_2 for the following program:
"**Input** x;
Simulate $X(\phi_y, g) = (p_y, q_y)$ until the stage x;
return $q_y(x)$";
$m := |\sigma|$;
Simulate $X(\phi_y, g) = (p_y, q_y)$ until the stage $m-1$;
$(*)$
If $q_y(m) > b(y)$ or $q_y(m)$ is a new conjecture, **return** e_1;
Otherwise, simulate $\phi_{q_y(m)}(|\sigma|)$ for m steps;
If the computation finishes, **return** the e_i that is different from $\phi_{q_y(m)}(|\sigma|)$;
Otherwise, $m := m+1$, simulate giving e_1 to g and go back to $(*)$

So suppose that, at stage m, h finds that $q(m)$ will be an old conjecture smaller than $b(e)$. Then, h simulates $\phi_{q(m)}(m)$ for m steps, to see what g thinks that $p(m)$ will be. If the computation finishes in m steps, h outputs a correct code for q that was not predicted. If the computation does not finish in m steps, h simulates giving e_1 to g and, if $q(m+1)$ is still an old conjecture smaller than $b(e)$, tests it on m again, but this time for $m+1$ steps. This process repeats itself until either g conjectures something *new* (which can only happen a finite number of times) or his conjecture *halts*. There are now two cases to consider.

Suppose that there is $m' \geq m$ such that $\phi_{q(m')}(m)$ halts in m' steps. Then, $p(m)$ is a code for q that is different from $\phi_{q(m')}(m)$, so $q(m')$ is a wrong conjecture for the code of p. We have thus eliminated one of the finitely many possible conjectures that g can make for the code of p. Eventually, g will not be able to produce a new conjecture that is still compatible with the data received, whence h XFExs-learn g.

Now suppose that $\phi_{q(m')}(m)$ never halts, for any $m' > m$, when g is constantly given e_1 after stage m. Then, we replace h with a player h', which behaves as h up until the stage $m-1$, after which it only outputs e_1. The player h' thus XExs-learn g. □

5. Conclusion

We have expanded the paradigm of interaction in two distinct directions, considering both the interaction of $n \geq 2$ players, each attempting to predict the behaviour of all others, and then, focusing on secretive learning, the interaction between players who are restricted by some methodology motivated by the philosophy of science. Each of these directions poses some interesting questions that we discuss below.

In the multi-player scenario, we considered that each player has access to the same information: the entire history of the interaction. A natural question one can pose is the effect on these results of giving different players a different access to previous prediction. We can consider, for example, teams of players which communicate amongst themselves in secret, playing against other teams, each with different limitations on how to access information. We can also define various notions of successful learning. We considered that a player learns others when his outputs encode correct predictions for those players' behaviour. When we have multiple players interacting, one may consider scenarios where a team of players learns another when they produce outputs which can be used, amongst themselves, to predict the player's behaviour. We also saw that we can define a *protector*, which learns back any player that succeeds in learning the player he is protecting, but we can consider other roles for different players. Players that might, for example, be designed to make the learning process harder for some player and easier for others. These scenarios seem especially interesting when we start considering players with methodological or complexity restrictions.

Regarding the study of special players, we saw that techniques from the learning theory of identification can be adapted to the interactive paradigm by designing players that simulate themselves and then, exploring the weaknesses of the players from identification theory, act as functions that they cannot learn. There are still many forms of restrictions we did not cover, which may exhibit interesting phenomena when carried over to the paradigm of interaction. Combining with the previous paragraph, one may consider the effect on interaction of different players, each following some methodological restriction. Are there combinations for which some form of secretive learning can be achieved?

Declaration of competing interest

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was funded by national funds through the FCT – Fundação para a Ciência e a Tecnologia, I.P., under the scope of the projects UIDB/00297/2020 (https://doi.org/10.54499/UIDB/00297/2020) and UIDP/00297/2020 (https://doi.org/10.54499/UIDP/00297/2020) (Center for Mathematics and Applications) and the FCT scholarship, reference 2022.10596.BD.

I am grateful to José Félix Costa for introducing me to learning theory and to the anonymous referees for their comments and suggestions.

References

[1] Dana Angluin, Inductive inference of formal languages from positive data, Inf. Control 45 (2) (1980) 117–135.
[2] Janis Bārzdiņš, Rusins Freivald, On the prediction of general recursive functions, Sov. Math. Dokl. 13 (1972) 1224–1228.
[3] Janis Bārzdiņš, Karli Podnieks, Towards a theory of inductive inference, in: Proceedings of 2nd Symposium and Summer School on Mathematical Foundations of Computer Science, 1973, pp. 9–15.
[4] Janis Bārzdiņš, Inductive inference of automata, functions and programs, in: Proceedings of the International Congress of Mathematicians, 1974, pp. 455–560.
[5] Janis Bārzdiņš, Two theorems on the limiting synthesis of functions, in: Theory of Algorithms and Programs, vol. 1(210), 1974, pp. 82–88.
[6] Manuel Blum, On the size of machines, Inf. Control 11 (3) (1967) 257–265.
[7] Lenore Blum, Manuel Blum, Toward a mathematical theory of inductive inference, Inf. Control 28 (2) (1975) 125–155.
[8] John Case, Periodicity in generations of automata, Math. Syst. Theory 8 (1) (1974) 15–32.
[9] John Case, Carl Smith, Anomaly hierarchies of mechanized inductive inference, in: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 314–319.
[10] John Case, Suzanne Ngo Manguelle, Refinements of inductive inference by Popperian machines, Technical report, Department of Computer Science, SUNY, Buffalo, 1979.
[11] John Case, Carl Smith, Comparison of identification criteria for machine inductive inference, Theor. Comput. Sci. 25 (2) (1983) 193–220.
[12] John Case, Sanjay Jain, Suzanne Ngo Manguelle, Refinements of inductive inference by Popperian and reliable machines, Kybernetika 30 (1) (1994) 23–52.
[13] John Case, The power of vacillation in language learning, SIAM J. Comput. 28 (6) (1999) 1941–1969.
[14] John Case, Sanjay Jain, Franco Montagna, Giulia Simi, Andrea Sorbi, On learning to coordinate: random bits help, insightful normal forms, and competency isomorphisms, J. Comput. Syst. Sci. 71 (3) (2005) 308–332.
[15] John Case, Directions for computability theory beyond pure mathematical, Math. Probl. Appl. Log. II (2007) 53–98.
[16] John Case, Timo Kötzing, Learning secrets interactively. Dynamic modeling in inductive inference, Inf. Comput. 220 (2012) 60–73.
[17] Keh-Jiann Chen, Tradeoffs in the inductive inference of nearly minimal size programs, Inf. Control 52 (1) (1982) 68–86.
[18] José Félix Costa, On discovering scientific laws, Int. J. Unconv. Comput. 14 (2019) 285–318.
[19] Jerome Feldman, Some decidability results on grammatical inference and complexity, Inf. Control 20 (1972) 244–262.
[20] Rusins Freivald, Minimal Gödel numbers and their identification in the limit, in: International Symposium on Mathematical Foundations of Computer Science, vol. 32, 1975, pp. 219–225.
[21] Mark Fulk, Sanjay Jain, Approximate inference and scientific method, Inf. Comput. 114 (2) (1994) 179–191.
[22] Konstantin Genin, Kevin T. Kelly, Theory choice, theory change, and inductive truth-conduciveness, Stud. Log. 107 (2019) 949–989.
[23] E. Mark Gold, Language identification in the limit, Inf. Control 10 (5) (1967) 447–474.
[24] Anthony Guilherme, A Study of Simpleminded Scientists, Master's thesis, Instituto Superior Técnico, 2023.
[25] Kevin Kelly, The Logic of Reliable Inquiry, Oxford University Press, Oxford, 1996.
[26] Efim Kinber, On the synthesis in the limit of almost minimal Gödel numbers, in: Theory of Algorithms and Programs, vol. 1, 1974, pp. 221–223.
[27] Efim Kinber, On a theory of inductive inference, Lect. Notes Comput. Sci. 56 (1977) 435–440.
[28] Stephen Cole Kleene, On notation for ordinal numbers, J. Symb. Log. 3 (4) (1938) 150–155.
[29] Franco Montagna, Daniel Osherson, Learning to coordinate; A recursion theoretic perspective, Synthese 118 (1999) 363–382.
[30] Daniel Osherson, Michael Stob, Scott Weinstein, Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists, The MIT Press, 1986.
[31] Karl Raimund Popper, Logik der Forschung: zur Erkenntnistheorie der moderner Naturwissenschaft, Verlag von Julius Springer, 1935.
[32] Hilary Putnam, Probability and confirmation. Voice of America, Forum on Philosophy of Science, 1963.
[33] Hartley Rogers, Theory of Recursive Functions and Effective Computability, MIT Press, 1987.
[34] Raymond M. Smullyan, Recursion Theory for Metamathematics, Oxford University Press on Demand, 1993.
[35] Robert I. Soare, The history and concept of computability, in: Studies in Logic and the Foundations of Mathematics, vol. 14, 1999, pp. 3–36.
[36] Elliot Sober, Ockham's Razors: A User's Manual, Cambridge University Press, 2015.
[37] Ray Solomonoff, A formal theory of inductive inference, Inf. Control 7 (1) (1964) 1–22.
[38] Rolf Wiehagen, Limes-Erkennung rekursiver Funktionen durch spezielle Strategien, Elektron. Inf.verarb. Kybern. 12 (1976) 93–99.