Learning to Win Process-Control Games Watching Game-Masters

John Case[1], Matthias Ott[2,⋆], Arun Sharma[3,⋆⋆], and Frank Stephan[4,⋆⋆⋆]

[1] Department of CIS, University of Delaware, Newark, DE 19716, USA, case@cis.udel.edu.

[2] Institut für Logik, Komplexität und Deduktionssysteme, Universität Karlsruhe, 76128 Karlsruhe, Germany, m_ott@ira.uka.de.

[3] School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia, arun@cse.unsw.edu.au.

[4] Mathematisches Institut, Universität Heidelberg, Im Neuenheimer Feld 294, 69120 Heidelberg, Germany, fstephan@math.uni-heidelberg.de.

Abstract. The present paper focuses on some interesting classes of *process-control games*, where *winning* essentially means successfully controlling the process. A *master* for one of these games is an agent who plays a winning-strategy. In this paper we investigate situations, in which even a complete model (given by a program) of a particular game does not provide enough information to synthesize — even in the limit — a winning strategy. However, if in addition to getting a program, a machine may also watch masters play winning strategies, then the machine is able to learn in the limit a winning strategy for the given game. Studied are successful learning from *arbitrary* masters and from pedagogically useful *selected* masters. It is shown that selected masters are strictly more helpful for learning than are arbitrary masters. Both for learning from arbitrary masters and for learning from selected masters, though, there are cases where one can learn programs for winning strategies from masters but not if one is required to learn a program for the master's strategy itself. Both for learning from arbitrary masters and for learning from selected masters, one can learn strictly more watching $m+1$ masters than one can learn watching only m. Lastly a simulation result is presented where the presence of a selected master reduces the complexity from infinitely many semantic mind changes to finitely many syntactic ones.

1 Introduction

To learn to win games such as chess, besides exploring the game tree with many practice games, it is also useful, or may even be necessary, to study the games

⋆ Supported by the Deutsche Forschungsgemeinschaft (DFG) Graduiertenkolleg "Beherrschbarkeit komplexer Systeme" (GRK 209/2-96).

⋆⋆ Supported by Australian Research Council Grant A49600456.

⋆⋆⋆ Supported by the Deutsche Forschungsgemeinschaft (DFG) Grant Am 60/9-1.

of master players.[1] We do not have much access to the masters' actual strategic programs mostly stored in their subconscious wetware. We have, instead, access to their game-playing *behavior*. It is also apparently useful to study the (game-playing) behavior of masters who play with very different styles. For example, it is likely better to study the behavior of *both* Kasparov and Deep Blue[2] than to study only one of them.

In machine learning, the *behavioral cloning* approach to process-control, surveyed in [1], involves using data from the behavior of master or expert human controllers, in order to make complex control learning problems feasible. For example, it has been used successfully to teach an autopilot to fly an aircraft simulator [1, 3, 12, 21, 22] and to teach a machine to operate efficiently a free-swinging shipyard crane [1, 25]. Behavioral cloning partly motivates the present paper.

For us the *masters* are players of *winning strategies* for the classes of process-control games described in Section 1.1 just below. Of course the experts behaviorally cloned in the machine learning experiments mentioned just above aren't necessarily playing exactly the same kinds of process-control games as we study herein, nor are they necessarily playing perfect, complete, winning strategies. Nonetheless, some of the parallels we describe, in the rest of this subsection, between these experimental machine learning results and our main theorems are very interesting and, we hope, instructive for the future.

In this paper we study situations in which the learnability of strategies *necessarily* depends on the fact that the learner, in addition to exploring a complete description of the game, may also watch the behaviour of master players. For pedagogical purposes, some masters may be better to watch than others. In [1, 3, 12, 21, 22] it is noted that better results were obtained using the data from some pilots rather than others. Theorem 6 in Section 2 below implies that some masters are strictly more helpful than others. Hence, we distinguish between whether we are using *arbitrary* or carefully *selected* masters.[3]

In [1, 3, 12, 21, 22] the learning program employed, C4.5 [19], did not merely learn to copy identically each pilot modeled. We show in Section 2, for both arbitrary (Theorem 3) and selected masters (Theorem 4), that there are cases where one can learn winning strategies for process-control games from masters *but not if one is required to copycat the master*. An interestingly contrasting theorem in the same section (Theorem 2) implies that, if a class of process-control games can be learned *incrementally*, i.e., after finitely many trial and error rounds, from *arbitrary* masters, then it can be incrementally learned by copycatting *selected* masters.

[1] In this paragraph, by *master players* we mean players who win, not players formally designated as Masters (as opposed to Grand Masters in chess, ...).

[2] In principle, in the case of Deep Blue, we could look at its actual strategic program, but, even Kasparov learned from watching Deep Blue's *behavior*.

[3] Formally, this distinction is handled definitionally by universal versus existential quantifiers over masters in positive assertions (see Definition 1 in Section 2 below).

In the learning-to-fly project [1, 3, 12, 21, 22] it was discovered that C4.5 got confused if it received data from more than one pilot at a time. Seemingly contrasting with this, in Section 3 below, we show, for both arbitrary (Theorem 8) and selected masters (Theorem 9), for each $m \geq 1$, surprisingly, one can learn strictly more watching $m + 1$ masters than one can learn watching only m.[4] Interestingly, the separation between learning from two and learning from one selected master(s) is witnessed by a class of games, which is essentially specified by the *natural* class of all trees that contain infinitely many infinite computable branches.

1.1 The Process-Control Games

In the present paper we focus on the learning of (programs for) winning strategies for two kinds of *process-control games*. The two kinds turn out to be, for all our purposes, mathematically *equivalent* [8]! The second kind is mathematically elegantly simple, so we state and prove our results in terms of it, but, although, this second kind *is* interesting in its own right, more of our motivation comes from the first kind. Again: all of our results straightforwardly carry over *mutatis mutandis* to the first kind of process-control game.

The process-control games of the first kind are called *closed computable games*. These games nicely model *reactive* process-control problems. The second are the *one-player immortality games* (synonymously: *branch games*). We describe each in turn, the first informally (with references) and the second in more detail.

To explain closed computable games, we show how to model an *archetypal* process-control problem as a closed computable game. Suppose we wish to keep the temperature t in a particular room between $t_{\text{min}} = 18\,°\text{C}$ and $t_{\text{max}} = 22\,°\text{C}$, inclusive, where the initial temperature is $t_0 = 20\,°\text{C}$. A temperature controller, which can sense the temperature in the room, and an *unseen* physical disturbance each act at discrete times $n = 0, 1, 2, \ldots$ on the temperature of the room as follows. At time n, the controller and the disturbance can and do choose respective actions a_n and d_n each in $\{-1, 0, 1\}$, where the resultant temperature, in degrees Celsius, at time $n+1$, is given by $t_{n+1} = f(t_n, a_n, d_n) = t_n + a_n + d_n$. The controller sees the temperature, not the disturbance, and, from its perspective, the temperature behaves indeterministically, yet, the *controller has to do well against all possible behaviors of the temperature and disturbance.* Equivalently, the controller needs a winning strategy for the associated two-player closed computable game we describe next. Player I is the controller and Player II is the *temperature*. Of course *we* know that Player II is a mere puppet of Player I *and the unseen disturbance*, but Player I can *see* the temperature, so it is better to model Player II as the temperature. A *play* of the game is just an alternating infinite sequence $a_0 t_1 a_1 t_2 \ldots$ of controller actions and temperatures, i.e., of moves of Players I and II. Player I *wins* the play $a_0 t_1 a_1 t_2 \ldots$ iff $(\forall n)[t_{\text{min}} \leq t_n \leq t_{\text{max}}]$.

[4] In the project on teaching an autopilot, a separate attribute distinguishing one pilot from another was not used; hence, this may explain the contrast.

The *goal set* for Player I is (by definition) the set of all plays $a_0 t_1 a_1 t_2 \ldots$ where Player I wins. In topology, *closed sets* (by definition) contain their limit points. The game we have described is called *closed* since the goal set for Player I is a closed set. I.e., *if* every finite initial segment $\sigma_n = a_0 t_1 a_1 \ldots a_{n-1} t_n$ of an infinite play yields no loss for the controller (i.e., if, for each $n \geq 1$, the temperature is between $t_{\min}$ and $t_{\max}$ at times $m = 1, \ldots, n$), *then* the *limit point* $a_0 t_1 a_1 \ldots$ of the sequence $(\sigma_n)_{n \in \omega}$ is a win for the controller, i.e., is in the goal set for Player I.[5] An example *winning strategy* for Player I, the controller, is as follows:

$$a_n = \begin{cases} +1 & \text{if } t_n \leq 20 \text{ or } n = 0, \\ -1 & \text{if } t_n > 20 \text{ and } n > 0. \end{cases} \tag{1}$$

We have defined the winning strategy (1) by an informal algorithm, or program; hence, it's clearly computable. A human *master* playing strategy (1) would have stored in his/her head this or an equivalent program. Formally, the watchable *behavior* would be an enumeration of the pairs (t, a) such that t is a temperature that could be observed and a is this master's response.

Next we describe the mathematically equivalent *one player immortality games*. As an informal example, consider a robot which is placed in a (finite or infinite) environment. The robot's job is to keep exploring its environment yet not get trapped or destroyed. To help it, it has a model of its environment, from which it can generate, for example, a map showing the dangerous spots. If we model finite environments as deterministic finite automata [20], then, in these cases, the one-player immortality game can be modeled as follows: Given a finite automaton, a winning strategy is an infinite word such that the finite automaton never visits a rejecting state when run on this word.

Formally, and in general, a *one player immortality game* is (by definition) a computable tree containing at least one infinite (computable) branch. The player starts at the root, and its moves must take it successively further from the root. The winning strategies are exactly the infinite branches of the trees. The conventional, master-free, strategy learning scenario is: given an enumeration of the graph, or even a program, of the game tree, incrementally synthesize a program for following *some* such winning strategy, i.e., for traveling along some *infinite* branch. Death or entrapment is modeled, then, by the player getting stuck on a *finite* branch.

1.2 The Power of Watching Masters

As shown in [8], there are classes $\mathcal{C}$ of immortality games such that no machine can synthesize a winning strategy for every game $G \in \mathcal{C}$ *in the limit*, given an enumeration of the graph, or even a program, of G as input. However, it is reasonable that one can overcome such limitations by presenting to the machine an enumeration of a winning strategy as additional input, that is, the learner may *watch a master*. In this work we study the power of (several variants of) this new learning notion. It is *important* to note that for all our results in Sections 2 and 3,

[5] For more formal treatment, see [5, 8, 10, 24].

where we compare different models of learning from masters, it *does not matter* whether the game tree is presented by a program, or by an enumeration of its graph! On the other hand, this matters when one compares master learning to conventional strategy learning. This comparison is the topic of Section 4, and there, we will also discuss the effect of the two different input models on this comparison.

In order to demonstrate the additional power, which a learner may gain from watching a master, we consider the following illustrative example [16]. Let the tree T_e consist of branches $f_k = ka_0a_1\ldots$ for every natural number $k \in \omega$, that is, T_e has an infinitely branching root such that every successor k of the root is only extended by the branch f_k. The branches f_k may be finite or infinite, depending on the e-th recursively enumerable set W_e (of an standard enumeration of all r.e. sets). More precisely, we let $0a_0\ldots a_n$ be "on the tree" T_e iff a_m is the smallest number such that $|W_{e,a_m}| \geq m$, for $m = 0,\ldots,n$. Here, $(W_{e,n})_{n\in\omega}$ denotes a finite approximation of W_e. For $k > 0$, $k0^n$ is on the tree, iff $|W_{e,n}| < k$. It is not difficult to see, that the branch f_0 is infinite iff W_e is infinite, and that, for $k > 0$, f_k is infinite iff $|W_e| < k$. Thus, every tree T_e has an infinite branch.

Assume now, that M is a machine, which computes, for all e, from a program of T_e a sequence $i_0i_1\ldots i_m iii\ldots$ which stabilizes on a program i for an infinite branch $f = f_k$ of T_e. Then it holds that W_e is infinite iff $k = 0$. Note that $k = f_k(0) = f(0)$ can be computed using the program i. Thus, we have a procedure which decides the index set $\mathit{Inf} = \{e\colon W_e \text{ infinite}\}$ in the limit, that is, Inf is Δ_1 with respect to the arithmetical hierarchy. However, as well known, Inf is Π_2-complete, which is a contradiction.

Thus, a conventional branch learning machine can not synthesize infinite branches for all T_e even if it gets a program of T_e as input (instead of just an enumeration of T_e).

Now, consider a learner, who gets a program j for T_e and who watches an enumeration $f(0)f(1)\ldots$ of an infinite branch of T_e. Clearly, having seen $f(0)$ the learner knows *that the branch extending* $k = f(0)$ *is infinite* and can then, using the program j of T_e, compute a program for the infinite branch f. This demonstrates that learning by watching masters may be extremely more powerful than conventional branch learning. It even allows the learner to find a branch *without* any mind changes. Moreover, the learner can even *identify* the input master, instead of just learning any infinite branch of T_e. In this work we analyze this additional power gained by watching masters, and compare several interesting variants of this learning notion.

Note that we could easily code e into the beginning of the tree T_e. Then it is still impossible to synthesize strategies for this class in the limit, however, a learner who watches a master can still be successful on this class, *even* if the learner gets only an enumeration of the graph of T_e as input. By similar reasons, all our results in Sections 2 and 3 are, as already noted, independent of the input model for the trees. Therefore, we will base our, now following, more formal treatment on the easier model, in which the learner gets a program for the game tree as input.

2 Learning from a Master

The natural numbers are denoted by ω. ω^* is the set of all *finite* sequences from ω, and ω^ω is the set of all *infinite* sequences from ω. We are using an acceptable programming system $\varphi_0, \varphi_1, \ldots$; the function computed by the e-th program within s steps is denoted by $\varphi_{e,s}$. REC is the set of all total computable functions. For strings $\sigma, \tau \in \omega^* \cup \omega^\omega$, $\sigma \preceq \tau$ means that σ is an initial segment of τ. $|a_1 \ldots a_n| = n$ denotes the length of a string $a_1 \ldots a_n \in \omega^*$. If $\sigma \in \omega^*$ and $\tau \in \omega^* \cup \omega^\omega$ then $\sigma\tau$ is the concatenation of the two strings. Let $\langle\cdot\rangle$ be a coding of ω^*, i.e., a bijective computable function $\langle\cdot\rangle : \omega^* \to \omega$, which is monotone with respect to subsequences:

$$(\forall \sigma, \tau \in \omega^*)[\sigma \text{ is a subsequence of } \tau \implies \langle\sigma\rangle \leq \langle\tau\rangle].$$

We identify finite strings with their code numbers. Total functions $f : \omega \to \omega$ are identified with the infinite string $f(0)f(1)\ldots$. We write $f[n]$ for the initial segment $f(0)\ldots f(n-1)$ of f.

$T \subseteq \omega^*$ is a *tree* if T is closed under initial segments.[6] Elements of a tree are called *nodes*. If $A \subseteq \omega^* \cup \omega^\omega$ is a set of finite and infinite strings, then the prefix closure, $\{\sigma \in \omega^* : \sigma \preceq \alpha \text{ for some } \alpha \in A\}$, is a tree. We often will define trees by specifying only such a set A. A total function $f : \omega \to \omega$ is an infinite branch of T if $f[n] \in T$ for all $n \in \omega$.

For background from inductive inference see, e.g., [14]. Remaining computability theoretic notation is from [13].

We are interested only in the class *Tree* of all computable trees which contain at least one infinite computable branch. If $f \in REC$ is an infinite computable branch of T we also say that f is *on* T. Moreover, in the context, when an f on T is given as input to a learner, such a branch is called a *master*.

In what follows, for convenience, we will say *branch* to refer only to infinite computable branches. Furthermore, also for convenience, we will sometimes speak of learning a branch when we mean learning a program for the branch.

Definition 1. *A Turing machine*[7] *M learns a branch* from an arbitrary master *for a tree $T \in$ Tree, if* for all *masters f on T and for all e with $\varphi_e = T$, the sequence $(M(e, f[n]))_{n\in\omega}$* converges *to an infinite computable branch of T, i.e., there exists an i such that φ_i is an infinite branch on T and $M(e, f[n]) = i$ for almost all n.*[8] *For $\mathcal{C} \subseteq$ Tree we write $\mathcal{C} \in$* **ArbMa** *if there exists a Turing machine M which learns branches from an arbitrary master for every $T \in \mathcal{C}$.*

[6] The theory remains the same if it is based on trees over a finite alphabet, e.g., on binary trees $T \subseteq \{0,1\}^*$ [16].

[7] By a well known argument, we can assume, without loss of generality, that all learning machines considered in this paper are total [14].

[8] Note that the sequence $M(e, f[n])$ converges *syntactically* to a (program for an) infinite computable branch, i.e., our notion of learning corresponds to the version of learning in the limit, which is called **Ex**-style (or incremental) learning in the literature.

A Turing machine M learns a branch from a selected master *for a tree $T \in$ Tree, if there* exists *a master f on T such that for all e with $\varphi_e = T$ the sequence $(M(e, f[n]))_{n\in\omega}$ converges to a (program for an) infinite computable branch of T. For $\mathcal{C} \subseteq$ Tree we write $\mathcal{C} \in$* **SelectMa** *if there exists a Turing machine M which learns branches from a selected master for every $T \in \mathcal{C}$.*

If there exists an **ArbMa**- *or* **SelectMa**-*learner which converges to a program* for the input master *(instead of just to a program for* any *infinite computable branch of T) for every tree T of a class $\mathcal{C}$, we say that $\mathcal{C}$ is learnable from arbitrary/selected masters* identically. *The corresponding classes are denoted by* **ArbMaId** *and* **SelectMaId**.

The definitions directly imply **ArbMa** $\subseteq$ **SelectMa**, **ArbMaId** $\subseteq$ **ArbMa** and **SelectMaId** $\subseteq$ **SelectMa**. One can prove that these inclusions are proper. Thus, to identify a master is a proper restriction for both learning from arbitrary and learning from selected masters. This shows that the advantage in watching one master (rather than none) comes from ones creating ones own winning strategy, and not from being a copycat. This result is not as surprising for **ArbMa** since one can imagine masters who go out of their way to avoid being figured out. But for the selective version of master learning this result is much more interesting. It says that regardless of how skilled pedagogically is the selected master you are watching, if one can learn a winning strategy from him/her/it, then this is, in general, only possible by creating a new strategy which differs from that of the master.

The noninclusion **SelectMa** $\not\subseteq$ **ArbMa** shows that not all masters are equally helpful for a learner. We are even able to prove **SelectMaId** $\not\subseteq$ **ArbMa**. Thus, while watching some masters provides enough information to identify these masters, watching others may be absolutely useless. Surprisingly, the other direction of the inclusion, **ArbMa** $\subseteq$ **SelectMaId**, holds. I.e., if every master allows the learner to at least find some winning strategy, then there exists one master which can even be identified by the learner. In summary, this establishes the proper linear chain **ArbMaId** $\subset$ **ArbMa** $\subset$ **SelectMaId** $\subset$ **SelectMa**. Moreover, it holds that *Tree* $\notin$ **SelectMa**. I.e., even if a learner can watch "a most helpful" master and is only required to output any winning strategy, it is still not possible to learn such strategies for all games, which have one.

One complexity measure of a learning task is the number of mind changes[9] which a machine needs to stabilize on a program for the target object. With respect to this complexity measure, learning without any mind changes at all provides the strongest positive results one may obtain. Zero mind change learning is also called *finite* learning in the literature. All the separation results which we give in this section are established by classes of trees so that the positive half of the separation result is witnessed by a machine which makes *no* mind changes!

[9] In the formal definition of *mind change* one allows the machine to output initially a special symbol "?" to indicate that it has yet not seen enough data to make up its mind for its first conjecture. So a mind change is said to happen, if $? \neq M(e, f[n]) \neq M(e, f[n+1])$.

Theorem 2. **ArbMa** $\subseteq$ **SelectMaId**.

Proof Sketch. This simulation proof is related to the proof of Freivalds and Wiehagen that every computable function can be learned from an upper bound of any of its indices [6,7].

Let M be an **ArbMa**-learner. Furthermore, for any given tree select that infinite branch which has among all computable infinite branches the smallest minimal index, say e. Now this branch f can be inferred as follows:

The new machine N emulates M on $f[n]$ and receives a sequence $(e_n)_{n\in\omega}$ of hypotheses which converges to an index e' of some branch of the tree. By choice, e' is greater or equal to e. At each stage, N amalgamates all programs i below e_n which are consistent with the input master f on $\{0,\ldots,n\}$ during their first n computation steps. This algorithm amalgamates in the limit all programs with indices below e' which are consistent with f and thus, identifies the master f in the limit. Hence N witnesses that the class of trees **ArbMa**-learned by M can also be **SelectMaId**-learned where the selected master is the one with the smallest index. $\square$

Theorem 3. **ArbMa** $\not\subseteq$ **ArbMaId**. *Moreover, this noninclusion can be witnessed by a class of trees which is* **ArbMa**-*learnable without any mind change.*

Theorem 4. **SelectMa** $\not\subseteq$ **SelectMaId**. *Moreover, this noninclusion can be witnessed by a class of trees which is* **SelectMa**-*learnable without any mind change.*

Proof Sketch. We will build trees T_e for all $e \in \omega$. Each tree T_e has the form $T_e = \bigcup_{i\in\omega} U_e^i$ such that $ei \preceq \sigma$ for all $\sigma \in U_e^i$, $|\sigma| \geq 2$, and each tree U_e^i contains at most one infinite computable branch f. In every tree U_e^i we will try to diagonalize against φ_e as a potential **SelectMaId**-learner for T_e. We will diagonalize against φ_e by securing that φ_e makes enough "prediction errors" on f, that is, $\varphi_e(f[n])$ is undefined or does not equal $f(n)$ for infinitely many n. Since every **SelectMaId**-learner M would yield a predictor $\lambda\sigma \in \omega^*.\varphi_{M(j,\sigma)}(|\sigma|)$ for f on $\varphi_j = U_e^i$, which is at most finitely often undefined or incorrect, this will suffice to diagonalize against all **SelectMaId**-learners. T_e will be uniformly computable from e. Since e is encoded into the master, it is therefore not necessary to give an index of T_e as input to the diagonalized machine.

Furthermore, we organize this diagonalization in such a way that it will only succeed in U_e^0 and in all trees U_e^i, for $i > 0$, such that φ_i is an infinite branch of T_e. All other trees U_e^i will be finite. This implies that for $i > 0$ every tree U_e^i contains an infinite branch iff φ_i is an infinite branch of T_e. This idea is the key to achieve $\mathcal{C} \in$ **SelectMa**, since the branch f of such an infinite tree U_e^i with $i > 0$ fulfills $f(1) = i$, that is, $f(1)$ is a program for an infinite branch of T_e. Therefore, these branches can be used as selected masters.

Construction of $T_e = \{\sigma_s^i : i, s \in \omega\}$:

Stage 0:

For all $i \in \omega$: $\sigma_0^i = \tau_0^i = ei$, $x_0^i = 2$.

Stage s+1:

1. For all $i > 0$:
   If $(\forall x < x^i_s)[\varphi_{i,s}(x)\downarrow]$ and $(\exists t \le s)(\exists j)[\varphi_i[x^i_s] \preceq \sigma^j_t]$ then let
   $x^i_{s+1} = x^i_s + 1$.
   Else let
   $x^i_{s+1} = x^i_s$.
   {*Check whether φ_i seems to become a branch of T_e so that U^i_e may be extended in step 2.*}
2. For all i:
   If $i = 0$ or $x^i_{s+1} > x^i_s$ then
   if $(\exists \tau_i)[\tau^i_s \preceq \tau_i \preceq \sigma^i_s$ and $\varphi_{e,s}(\tau_i)\downarrow]$, then choose the smallest such τ_i and let
   $a_i = s + 1 + \varphi_e(\tau_i)$,
   $\sigma^i_{s+1} = \tau^i_{s+1} = \tau_i a_i$.
   Else let
   $a_i = s + 1$,
   $\sigma^i_{s+1} = \sigma^i_s a_i$, $\tau^i_{s+1} = \tau^i_s$.
   Else let
   $\sigma^i_{s+1} = \sigma^i_s$, $\tau^i_{s+1} = \tau^i_s$.

End of Construction.

□

Theorem 5. *Tree* $\notin$ **SelectMa**.

Theorem 6. **SelectMaId** $\not\subseteq$ **ArbMa**. *Moreover, this noninclusion can be witnessed by a class of trees which is* **SelectMaId**-*learnable without any mind change.*

3 Hierarchies for Learning from Many Masters

Definition 7. *In the following we write $icb(T)$ for the number of infinite computable branches of a tree $T \in$ Tree. Note that T may have infinitely many infinite computable branches, in which case we write $icb(T) = \infty$.*

A Turing machine M learns a branch from arbitrary m masters *for a tree $T \in$ Tree, if* for all *masters $f_1, \ldots, f_m$ on T with*

$$|\{f_1, \ldots, f_m\}| \ge \min\{m, icb(T)\} \tag{2}$$

and for all e with $\varphi_e = T$ the sequence $(M(e, f_1[n], \ldots, f_m[n]))_{n\in\omega}$ converges to a (program for an) infinite computable branch of T. M learns a branch from selected m masters *for a tree $T \in$ Tree, if* there exist *masters $f_1, \ldots, f_m$ on T such that for all e with $\varphi_e = T$ the sequence $(M(e, f_1[n], \ldots, f_m[n]))_{n\in\omega}$ converges to a (program for an) infinite computable branch of T.*

The corresponding classes are denoted by $\mathbf{ArbMa}^m$ *and* $\mathbf{SelectMa}^m$.

Requirement (2) just above *on the input masters of an* **ArbMa**-*learner* ensures that:

1. If there are at least m distinct infinite branches on T, then, the m masters are pairwise distinct;
2. if the number k of distinct infinite branches on T is $< m$, then exactly k of the m masters are pairwise distinct; and,
3. hence, both $\mathbf{ArbMa}^m \subseteq \mathbf{ArbMa}^{m+1}$ and $\mathbf{SelectMa}^m \subseteq \mathbf{SelectMa}^{m+1}$.

Theorem 8. *For all $m \geq 1$: $\mathbf{ArbMa}^m \subset \mathbf{ArbMa}^{m+1}$. Moreover, the noninclusion can be witnessed by a class of trees which is $\mathbf{ArbMa}^{m+1}$-learnable without any mind change.*

The separation of $\mathbf{ArbMa}^{m+1}$ and $\mathbf{ArbMa}^m$ can also be witnessed by the natural class $\mathcal{C}_{m+1}$ of all computable binary trees from *Tree* which have at most $m+1$ (arbitrary) infinite branches. However, the $\mathbf{ArbMa}^{m+1}$ learner for $\mathcal{C}_{m+1}$ needs generally $\log(m)$ mind changes (instead of zero mind changes).

The analogous hierarchy result to Theorem 8 also holds for branch learning from *selected* masters:

Theorem 9. *For all $m \geq 1$: $\mathbf{SelectMa}^m \subset \mathbf{SelectMa}^{m+1}$.*

This result, in particular, implies that even our most powerful notion of master learning, namely, learning an arbitrary branch from several selected masters, is not strong enough to learn a branch for every tree:

Corollary 10. *For all $m \geq 1$: $Tree \notin \mathbf{SelectMa}^m$.*

Theorem 9 can be proven using a team learning result from [4]. However, interestingly, the separation of $\mathbf{SelectMa}^2$ and $\mathbf{SelectMa}^1$ is witnessed by *natural* classes of trees, in particular, by the class *TreeInf* of all trees which contain infinitely many infinite computable branches. Since such separation results, which are witnessed by natural classes, are particularly interesting for inductive inference, we present here the proof of this result instead of proving the general statement from Theorem 9.

Besides *TreeInf*, also the following natural classes witness $\mathbf{SelectMa}^2 \not\subseteq \mathbf{SelectMa}^1$ (see, e.g., [4] for definitions):

- the binary trees which contain only computable infinite branches,
- the binary trees of bounded width,
- the binary trees of bounded rank, and
- the binary trees of bounded variation.

Theorem 11. *The class TreeInf of all trees, which contain infinitely many infinite computable branches, is in $\mathbf{SelectMa}^2 - \mathbf{SelectMa}^1$.*

Proof. We first prove $TreeInf \in \mathbf{SelectMa}^2$. So, let an arbitrary tree $T \in TreeInf$ be given, and let e' be the smallest e such that φ_e is on T. Then there exists an x' such that for all $e < e'$, either $\varphi_e(x)\uparrow$ for some $x < x'$, or $\varphi_e[x'] \notin T$. Since T contains infinitely many branches, T contains an infinitely branching node, or, for every n, there exist two different branches f_1, f_2 on T with $f_1[n] = f_2[n]$. This implies that there exist an n' and two different branches f_1, f_2 on T

with $f_1[n'] = f_2[n']$, $f_1(n') \neq f_2(n')$ and $n' + f_1(n') + f_2(n') > x'$. We choose these branches f_1 and f_2 as selected masters.

Now, the machine M witnessing $\mathit{TreeInf} \in \mathbf{SelectMa}^2$ works as follows. On input T, f_1, f_2 the machine M waits until it finds the first n with $f_1(n) \neq f_2(n)$. Then it computes $y = n + f_1(n) + f_2(n)$. Note that $n = n'$ and $y > x'$. From now on, M outputs in stage n the smallest $e \leq n$ such that

$$(\forall x < y)[\varphi_{e,n}(x)\downarrow] \text{ and } \varphi_e[y] \in T. \tag{3}$$

If no such e exists, then M outputs 0.

By definition, there exists an $n \geq e'$ such that $e = e'$ and n satisfy (3). On the other hand, no $e < e'$ can satisfy (3) for any n. Thus, from some point on, the machine will always output e' which is a program for a branch of T. Hence, M is a correct $\mathbf{SelectMa}^2$-learner for the class *TreeInf*.

We only *sketch* the proof of the negative statement. Assume by way of contradiction that the Turing machine M witnesses $\mathit{TreeInf} \in \mathbf{SelectMa}$. Let η_n denote the unique string with $n = \langle \eta_n \rangle$. We will construct a tree T such that M fails to **SelectMa**-learn a branch for T as follows:

Construction of $T = \bigcup_{s\in\omega} T_s$:

Stage 0: $T_0 = \emptyset$, $\mathit{queue}_0 = (\epsilon)$.

Stage s+1: Assume that $\mathit{queue}_s = (\sigma_0, \ldots, \sigma_q)$.
Compute $j = M(e, \sigma_0)$, where $\varphi_e = T$. $\quad$ {*Recursion Theorem!*}
Check whether η_s satisfies the following conditions:
(a) $\sigma_0 \prec \eta_s$,
(b) $(\forall \tau, \sigma_0 \prec \tau \prec \eta_s)[\tau \in T_s]$,
(c) $(\exists x < |\sigma_0|)[\varphi_{j,s}(x)\uparrow]$ or $(\exists x < |\eta_s|)[\varphi_{j,s}(x)\downarrow \neq \eta_s(x)]$.

If η_s satisfies (a) – (c) then
let $T_{s+1} = T_s \cup \{\eta_s\}$;
if $M(e, \eta_s) \neq j$ then let $\mathit{queue}_{s+1} = (\sigma_1, \ldots, \sigma_q, \eta_s 0, \eta_s 1)$, $\quad$ (4)
else let $\mathit{queue}_{s+1} = \mathit{queue}_s$.
If η_s does not satisfy (a) – (c) then let $T_{s+1} = T_s$, $\mathit{queue}_{s+1} = \mathit{queue}_s$.

End of Construction.

One can show that T is actually in *TreeInf*. In order to prove that M does not **SelectMa**-learn a branch of T one distinguishes the following two cases.

Case 1: There exists a stage s with $\mathit{queue}_s = \{\sigma_0, \ldots, \sigma_q\}$ such that σ_0 is *never* removed from the queue in any stage $t > s$. This implies that, for all masters f on T, f extends σ_0, and, furthermore, $M(e, f[n]) = M(e, \sigma_0)$ for all $n \geq |\sigma_0|$. Thus, M converges on all masters to $j = M(e, \sigma_0)$. Then, φ_j is total. However, in this case, from some point on, we will only extend branches which are inconsistent with φ_j due to condition (c). Hence, φ_j is no branch of T. Contradiction.

Case 2: Every string is eventually removed from queue_s. Then, by construction, M will make infinitely many mind changes on every master of T due to line (4). Thus, for every master f of T, M fails to converge to a single program on input f. Contradiction. $\square$

4 Master Learning versus Branch Learning

How is ordinary branch learning [8, 16], where the learner can only inspect the tree but has no access to a master, related to master learning? As already noted in the introduction, this comparison is, in contrast to the results of Sections 2 and 3, effected by the input model for the tree. In order to make the differences clear, we denote, for **Crit** $\in$ {**ArbMaId**, **ArbMa**, **SelectMaId**, **SelectMa**}, the version of **Crit**, where $M(e, f[n])$ in Definition 1 is replaced with $M(T[n], f[n])$, by **Enum-Crit**.

It is well known that, in general, enumerations of graphs are far less useful then programs. Therefore, it is not so surprising that **ArbMaId** $\not\subseteq$ **Enum-SelectMa**. In other words, if one is working with an enumeration of the tree, then even the most powerful master learning notion fails to capture the most restrictive master learning notion working with programs of the trees. This result, in particular, implies that **Enum-Crit** $\subset$ **Crit** for all master learning criteria **Crit**.

The most powerful branch learning notion from [8] is called *weak* **Bc**-*learning*.[10] A class $\mathcal{C} \subseteq$ *Tree* is in **BranchWBc**, if there exists a Turing machine M such that for all $T \in \mathcal{C}$ and for almost all n, the function $\varphi_{M(T[n])}$ is an infinite computable branch of T, where the tree $T \subseteq \omega^*$ is identified with its characteristic function. Note that a **BranchWBc**-learner is allowed to make infinitely many *semantic* mind changes.

A machine *synthesizes* branches for a class $\mathcal{C} \subseteq$ *Tree in the limit* if it computes from every index of a tree $T \in \mathcal{C}$ a sequence of programs, which converges (syntactically) to a (program for an) infinite branch of T. The corresponding class is denoted by **SynthLim**.[11]

By using the trees from Section 1.2 one can show that no "pure branch learning class" captures any "master learning classes", demonstrating the extreme power of learning from masters:

Theorem 12. **Enum-ArbMaId** $\not\subseteq$ (**BranchWBc** $\cup$ **SynthLim**).

What can we say about the other direction, that is, which master learning classes capture which branch learning classes? First, one can show, that the master learning notions which are working with enumerations of trees are too restrictive:

Theorem 13.
BranchWBc $\not\subseteq$ **Enum-SelectMa** *and* **SynthLim** $\not\subseteq$ **Enum-SelectMa**.

However, by definition, **SynthLim** is a subset of **ArbMa**. One can show that this inclusion is proper. Hence, we have **SynthLim** $\subset$ **ArbMa** $\subset$ **SelectMaId** $\subset$ **SelectMa**. The classes **SynthLim** and **BranchWBc** are incomparable [8], that is, there are classes in **BranchWBc** for which one cannot synthesize branches in the limit. Thus, it is interesting to see whether one of the three master learning

[10] The criterion is not very restrictive (weak), but, then, many things can be learned with respect to it, so, from that perspective, it is powerful.

[11] **SynthLim** is equivalent to the learning criterion called **Uni**$[K]$ in [8, 16].

notions extending **SynthLim** are powerful enough to capture **BranchWBc**. It turns out that while **ArbMa** is still too restrictive to cover **BranchWBc**, the power of a **SelectMaId**-learner suffices. Thus, if a learner gets the index of the tree and can watch *the right* master, then the learner can improve its weak performance of infinitely many semantic mind changes to the very strong syntactic convergence to the master in the limit. The next and the last theorem of the paper formalizes this positive mind change complexity reduction result, which has an interesting simulation proof:

Theorem 14. **BranchWBc** $\subseteq$ **SelectMaId**.

Proof. Let $\mathcal{C}$ be in **BranchWBc**. Given an index of a tree $T \in \mathcal{C}$, the set E of all programs output by the learner is uniformly enumerable and depends only on the graph of T and not on the index of T. The indices in E define the following subtree $T' \subseteq T$:

$$\sigma \in T' \Leftrightarrow \sigma \in T \wedge (\exists e \in E)\,(\forall x \in dom(\sigma))\,[\varphi_e(x)\downarrow = \sigma(x)].$$

T' is enumerable but may not be computable. Furthermore, for all indices of T, the strings in T' are enumerated in the same order. In addition, T' has only finitely many finite branches, which cannot be extended to infinite branches, that is, almost all nodes of T' lie on an infinite branch of T'. So, knowing the enumeration of T', it is possible to extend almost every node σ of T' to an infinite branch of T'. This branch u_σ is defined inductively after initializing it with σ. Let now n be the first value not yet defined, then let

$$u_\sigma(n) = \begin{cases} a & \text{if } a \text{ is the first number such that } u_\sigma(0)u_\sigma(1)\ldots u_\sigma(n-1)\,a \\ & \text{is enumerated into } T'; \\ \uparrow & \text{otherwise, if there is no such } a. \end{cases}$$

Every function u_σ is either an infinite branch of T' or is a finite branch which cannot be extended and differs from all infinite branches of T' at some value. So, the following two propositions hold:

$$\tau \prec \sigma \text{ and } u_\tau[n] \preceq u_\sigma \implies u_\tau(n)\downarrow \tag{5}$$

$$\sigma \preceq \tau \preceq u_\sigma \implies u_\sigma = u_\tau \tag{6}$$

Since the **BranchWBc**-learner produces at least one infinite branch, T' has at least one infinite branch. Moreover, if σ is a node of T' which is longer than every finite branch, then the function u_σ is infinite. Taking the master $f = u_\sigma$, the following algorithm learns this branch. The learner starts with u_λ. Whenever for the current τ an argument n is found with $u_\tau(n)\downarrow \neq f(n)$ then τ is extended to $f(0)f(1)\ldots f(n)$ and a mind change to this new u_τ is made. After finitely many mind changes, $u_\tau = u_\sigma$ and the **SelectMaId**-learner succeeds. □

5 Future Work

As already mentioned in the introduction, the experts, which have been observed in the behavioral cloning experiments, are not playing necessarily winning strategies. Also master chess players do not win every play: instead of being classified into winners and losers, real players are ranked, more or less continuously, from bad to very good players. In the present basic study of watching masters we abstracted from this fact. But in future work it would be interesting to study learning from imperfect masters. At first instance, the crucial question for this research is, how to model imperfect masters. For example, one may consider masters which are playing finite variants of winning strategies. Or one may assume, that one of m input masters, or a majority of them, knows the best move in each situation. Moreover, for imperfect masters the problem of on-line learning is no longer trivial [8]! It would be interesting to investigate probabilistic learning from imperfect masters. The performance of an on-line learner can be measured by the number of lost plays, until it is eventually playing perfectly. Is there a connection between the quality of the input masters and the number of plays which an on-line learner loses?

The one-player immortality games given by a deterministic finite automaton, as described in Section 1.1, are just a special case of the well known two-player finite-state games [2, 11, 24]. In such games there always exist winning strategies which can be executed by a finite automaton. In [15] it is investigated whether one can *efficiently* learn strategies for one- and two-player *closed* finite-state games from membership and play queries, where membership queries involve asking whether a certain position is already a loss, and play queries involve asking whether a certain finite automaton implements a winning strategy. It would be interesting to apply the master learning concept to this situation. Can the time and query complexity of a learner be improved if the learner can ask queries about a winning automaton?

In Section 3 we have stated natural examples witnessing the separation of $\mathbf{ArbMa}^{m+1}$ and $\mathbf{ArbMa}^{m}$, and of $\mathbf{SelectMa}^{2}$ and $\mathbf{SelectMa}^{1}$. It would be interesting to look whether further natural examples separate some other levels of the $\mathbf{SelectMa}^{m}$ hierarchy.

References

1. M. Bain and C. Sammut. A framework for behavioural cloning. In K. F. S. Muggleton and D. Michie, editors, *Machine Intelligence 15*. Oxford University Press, 1996.
2. J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
3. C. Sammut, S. Hurst, D. Kedzier and D. Michie. Learning to fly. In D. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Conference on Machine Learning*. Morgan Kaufmann, 1992.
4. J. Case, S. Kaufmann, E. Kinber, and M. Kummer. Learning recursive functions from approximations. *Journal of Computer and System Sciences*, 55:183–196, 1997.
5. D. Cenzer and J. Remmel. Recursively presented games and strategies. *Mathematical Social Sciences*, 24:117–139, 1992.
6. R. V. Freivalds and R. Wiehagen. Inductive inference with additional information. *Elektronische Informationsverarbeitung und Kybernetik*, 15:179–185, 1979.

7. S. Jain and A. Sharma. Learning with the knowledge of an upper bound on program size. *Information and Computation*, 102(1):118–166, Jan. 1993.
8. M. Kummer and M. Ott. Learning branches and learning to win closed games. In *Proceedings of Ninth Annual Conference on Computational Learning Theory*, pages 280–291, New York, 1996. ACM.
9. M. Kummer and F. Stephan. On the structure of degrees of inferability. *Journal of Computer and System Sciences*, 52(2):214–238, Apr. 1996.
10. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proceedings of the Annual Symposium on the Theoretical Aspects of Computer Science*, volume 900 of *LNCS*, pages 229–242. Springer-Verlag, 1995.
11. R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.
12. D. Michie and C. Sammut. Machine learning from real-time input-output behaviour. In *Proceedings of the International Conference on Design to Manufacture in Modern Industry*, pages 363–369, 1993.
13. P. Odifreddi. *Classical Recursion Theory*. North-Holland, Amsterdam, 1989.
14. D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn*. MIT Press, Cambridge, Massachusetts, 1986.
15. M. Ott and F. Stephan. The complexity of learning branches and strategies from queries. In *Proceedings of the Eighth Annual International Symposium on Algorithms and Computation*, volume 1350 of *LNCS*, pages 283–292. Springer, 1997.
16. M. Ott and F. Stephan. Structural measures for games and process control in the branch learning model. In S. Ben-David, editor, *Proceedings of the Third European Conference on Computational Learning Theory*, volume 1208 of *LNAI*, pages 94–108. Springer, 1997.
17. L. Pitt. Probabilistic inductive inference. *Journal of the ACM*, 36:383–433, 1989.
18. L. Pitt and C. Smith. Probability and plurality for aggregations of learning machines. *Information and Computation*, 77:77–92, 1988.
19. J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.
20. R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, Apr. 1993.
21. C. Sammut. Acquiring expert knowledge by learning from recorded behaviours. In *Japanese Knowledge Acquisition Workshop*, 1992.
22. C. Sammut. Automatic construction of reactive control systems using symbolic machine learning. *Knowledge Engineering Review*, 11(1):27–42, 1996.
23. C. Smith. The power of pluralism for automatic program synthesis. *Journal of the ACM*, 29:1144–1165, 1982.
24. W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the Annual Symposium on the Theoretical Aspects of Computer Science*, volume 900 of *LNCS*, pages 1–13. Springer-Verlag, 1995.
25. T. Urbančič and I. Bratko. Reconstructing human skill with machine learning. In A. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence*. John Wiley & Sons, 1994.