Structural Measures for Games and Process Control in the Branch Learning Model

Matthias Ott*
Universität Karlsruhe

Frank Stephan†
Universität Heidelberg

Abstract

Process control problems can be modeled as closed recursive games. Learning strategies for such games is equivalent to the concept of learning infinite recursive branches for recursive trees. We use this branch learning model to measure the difficulty of learning and synthesizing process controllers. We also measure the difference between several process learning criteria, and their difference to controller synthesis. As measure we use the information content (i.e. the Turing degree) of the oracle which a machine need to get the desired power.

The investigated learning criteria are finite, EX-, BC-, Weak BC- and online learning. Finite, EX- and BC-style learning are well known from inductive inference, while weak BC- and online learning came up with the new notion of branch (i.e. process) learning. For all considered criteria — including synthesis — we also solve the questions of their trivial degrees, their omniscient degrees and with some restrictions their inference degrees. While most of the results about finite, EX- and BC-style branch learning can be derived from inductive inference, new techniques had to be developed for online learning, weak BC-style learning and synthesis, and for the comparisons of all process learning criteria with the power of controller synthesis.

1 Introduction

Kummer and Ott [13] have developed a theoretical model of learning winning strategies for closed recursive games [7]. Closed recursive games are games of infinite duration and a special kind of Gale-Stewart games (see e.g. [28]). These kind of games are especially interesting since process control problems can be interpreted as such games [18, 27, 29]. Closed games correspond to control problems with safety conditions, which say that the process may never reach a "bad" state [30]. An example of such a control problem is a temperature controller which has to hold the temperature in a room between $t_{\min}$ and $t_{\max}$.

*Institut für Logik, Komplexität und Deduktionssysteme, Universität Karlsruhe, D-76128 Karlsruhe, Germany, Email: m_ott@ira.uka.de. Supported by the Deutsche Forschungsgemeinschaft (DFG) Graduiertenkolleg "Beherrschbarkeit komplexer Systeme" (GRK 209/2-96).

†Mathematisches Institut, Universität Heidelberg, D-69120 Heidelberg, Germany, Email: fstephan@math.uni-heidelberg.de. Supported by the Deutsche Forschungsgemeinschaft (DFG) grant Am 60/9-1.

Luzeaux, Martin and Zavidovique[17, 19, 20] have developed a different theoretical model of learning to control processes. An advantage of the game approach is that the setting can be shown to be equivalent to branch learning [13]. Here the learner has to find an infinite recursive branch of an infinite recursive tree. This yields a very easy model which allows a clearer theory. A further difference between the two models is that Kummer and Ott use the standard model of data input and the well known learning criteria from inductive inference while Luzeaux et al. introduce new settings for this.

The classical approach to process control is *synthesis* [8]: First, develop a complete mathematical model of the process. From this model compute the corresponding controller. The efforts to write chess programs, for example, can be classified as a synthesis problem, since the rules of the game (i.e. a program for the game tree) are completely known in advance. The synthesis problem has also been investigated theoretically for infinite games, e.g. in [4, 12, 15, 21, 29]. This classical approach fails for the control problems appearing in modern applications from, for example, robotics and manufacturing [2, 16, 23, 31]: E.g. very often the tasks to be controlled are too complex or just not completely known (e.g. robots in unknown environment, a chemical plant where not everything is accessible to measurement or completely modeled, ...) so that a complete mathematical model cannot be developed. Additionally, the synthesis of controllers only works well for more easy control problems. This has led to the application of machine learning techniques in process control [2, 22, 26, 31], taking into account that one can get more and more data over time about the processes to control.

Our concern is the theoretical foundation of these phenomena, i.e. the power of learning in process control, and the comparison of learning and controller synthesis. Here, the game model — or even better the more easy and equivalent branch learning model — allows a rigorous mathematical study of these phenomena. In recursion theoretic terminology controller synthesis is called *uniform computation* (*Uni*). In [13] it was shown that to uniformly compute and to (EX-)learn controllers are incomparable tasks. Moreover, there are processes for which one can learn controllers, but it is not possible to learn a complete model of the process, and vice versa. But how big is the gap between learning and uniform computation? Is there a possibility to measure the difference between these two constructive approaches?

In this paper we answer these questions in terms of oracle measures [9, 14]. Oracles improve the power of machines. Which information content do oracles need such the oracle learning machines capture uniform computation, and vice versa? The information content of a oracle is its Turing degree. We study this question for different learning criteria: finite (FIN-), EX-, BC- and Weak BC- (WBC-) style learning. The above are *offline* versions of learning, i.e. the learner outputs programs intended to control a process. We also study an *online* version of process learning — introduced in [13] — in which the learning machine directly outputs control actions.

Besides the comparison of different criteria — like uniform computation versus learning, or offline versus online learning — we also investigate the classical question of oracle learning: which oracles are trivial, i.e., which oracles do not help; which oracles are omniscient, i.e., which allow to find an infinite recursive branch on every tree which has one; how do the inference degrees look like, i.e., for which A and B does $Crit[A] \subseteq Crit[B]$ hold. (For the meaning of $Crit[A]$ see the definitions below in this paper). Many learning criteria have direct counterparts in branch learning. The inference degrees of the counterparts of FIN and EX are very similar to the original ones. But this is already different at BC: BC has two counterparts ($BranchBC$ and $BranchWBC$) and furthermore the inference-degrees of both are a very different from that of BC: Other than BC none of them has a low omniscient oracle and $BranchWBC$ has even only recursive trivial oracles. The new criteria $BranchOnl$ behaves similar as finite learning. For Uni and $BranchWBC$ new techniques had to be developed to answer the above questions.

It is fundamental that uniform computation is not captured by learning, since the identity function is trivially computable, while it is one of the most fundamental problems in function learning (namely, the $REC \in Crit$? problem). This is confirmed by our results providing exact oracle measures: We show for all learning criteria that if it is possible to capture uniform computation by using an oracle, then this oracle has to be very powerful: It is impossible for finite and online branch learning at all. For EX- and BC-style branch learning we need oracles which are omniscient for this branch learning criteria. And in the case of Weak BC-branch learning the oracles have to be omniscient for the class BC in the classical setting of learning functions.

On the other side EX-, BC- and WBC-learning are not included in uniform computation for more involved reasons. We will see that an $\emptyset'$-oracle, which is a whole Turing jump below the omniscient Uni-degree, suffice to capture EX and BC-style learning. Nevertheless, this also shows that the advantage of learning over computation can be measured to correspond to a whole Turing jump. And weak BC-learning is in fact so powerful that the distance corresponds to two Turing jumps, which means that only omniscient oracles give synthesizing machines as much power.

Many of our results imply the separation of different learning criteria, e.g. $BranchBC$ versus $BranchWBC$. Because, if $Crit_1 \subseteq Crit_2[A]$ only holds for noncomputable oracles A, then clearly $Crit_1 \not\subseteq Crit_2$. Thus, by using oracles one gets alternative proofs for several of the separation results in [13]. Interestingly, these proofs are often easier than the direct proofs which do not use oracles.

We have already mentioned the technical advantage of the branch learning model [13]. Therefore the body of this paper is written in the terminology of branch learning. The relation between branch, game and process learning is the following: game learning is just a mathematical model of process learning, and branch learning is equivalent to strategy learning for closed recursive games. The following figure shows the correspondence between the different notions:

Problem:	Process	Game	Tree
Solution:	Controller	Strategy	Branch

The problem of finding infinite recursive branches of recursive trees is of independent interest in recursion theory [7, 24]. In [11] it is studied to which extend (in the sense of so called k-selectors) infinite recursive branches of trees can be computed uniformly. This approach was combined with inductive inference in [5]. Here the learner receives input/output examples of f and as additional information an index of a tree T such that f is a branch of T.

2 Notation and Definitions

The natural numbers are denoted by ω. We identify sets $A \subseteq \omega$ with their characteristic function. $\#A$ denotes the cardinality of $A \subseteq \omega$.

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 $\varphi_{e,s}$. $W_e := dom(\varphi_e)$ is the e-th recursively enumerable set. We write $W_{e,s}$ for $dom(\varphi_{e,s}) \cap \{0, \ldots, s\}$. REC is the set of all total recursive functions. Turing reducibility is denoted by $\leq_T$. If A is a set, then A' is the halting problem relative to A, that is $\{e\colon \varphi_e^A(e)\downarrow\}$. The halting problem $\emptyset'$ is denoted by K. A is *high* iff $K' \leq_T A'$. A is *low* if $A' \leq_T K$. A is called *PA-complete* relative to B if every partial B-recursive $0,1$-valued function has a total A-recursive extension. For $B \equiv_T \emptyset$ this is equivalent to the original definition which states that A is in the Turing degree of a complete extension of Peano Arithmetic (see [24]).

For strings $\sigma, \tau \in \omega^*$, $\sigma \preceq \tau$ means that σ is an initial segment of τ. $|a_1 \ldots a_n| = n$ denotes the length of a string $a_1 \ldots a_n \in \omega^*$. Strings $\sigma \in \omega^*$ are identified with their "code numbers" according to some fixed coding $\langle\cdot\rangle$ of ω^*, which satisfies $(\forall \sigma, \tau)[\sigma \preceq \tau \implies \langle\sigma\rangle \leq \langle\tau\rangle]$. Total Functions $f\colon \omega \to \omega$ are identified with the infinite string $f(0)f(1)\ldots$. We write $f\restriction n$ for the string $f(0)\ldots f(n-1)$.

$T \subseteq \omega^*$ is a *tree* if T is closed under initial segments. If $T \subseteq \{0,1\}^*$ then T is called a *binary* tree. Elements of a tree are called *nodes*. If $M \subseteq \omega^* \cup \omega^\omega$ is a set of finite and infinite strings, then the prefix closure $Pref(M) := \{\sigma \preceq \alpha\colon \alpha \in M\}$ is a tree. We often will define trees by only specifying such a set M. $\alpha \in \omega^\omega$ is an infinite branch of T, if $\{\sigma\colon \sigma \preceq \alpha\} \subseteq T$. In this paper we are only interested in the class $TREE$ of all recursive trees which have an infinite recursive branch. Note that according to our conventions an infinite recursive branch of T is just an recursive function f with $\{f\restriction n\colon n \in \omega\} \subseteq T$.

The branch learning model in [13] uses binary trees. One can show that the theory remains the same if it is based on recursive trees over ω:

Theorem 2.1. *For all criteria $Crit_1, Crit_2$ which we consider in this paper and all oracles A, B, if $Crit_1[A] \not\subseteq Crit_2[B]$, then there is a class of recursive binary trees witnessing this fact.*

The proof will be given in Section 8. Since we have discovered that it makes the proofs in this paper more simple, we base the definitions on arbitrary trees.

EX, *FIN*, and *BC* denote the classes of sets $S \subseteq REC$ which are identifiable by explanation, finitely identifiable by explanation and behaviorally correctly identifiable, respectively. The exact definitions for the different learning criteria are the direct counterparts to those given shortly in the context of branch learning. For background from inductive inference see e.g. [3, 9, 10, 25]. Remaining recursion theoretic notation is from [24].

3 Finite and Online Branch Learning

At first we define the notion of branch learning machines [13]. In the world of process control you may think of a learner which has two copies of the process to control. The first one is for experimentation (P_E) the second for application of the guessed controllers. We are considering machines without time and space bounds. Therefore we can assume that the machine may in the limit try all possible action sequences infinitely often on P_E. A sensor signals the respond of P_e to the learner. P_E may respond in different ways on the same action sequence due to indeterminism or disturbance by the environment. As a kind of fairness condition we assume that as long as there are possible respond sequences these will eventually appear. As a consequence we can assume that the learner gets an enumeration of all action/respond-sequences as input.

This assumption may seem a little bit strong, since the learner gets in the limit the whole information about the process. But note that the main content of our theorems is that something is *not* learnable. Thus, the significance of these results even grow if we base them on this strong input model.

While games such as chess and Go have finite game trees, these game trees are too large for exhaustive search. Thus, one has to come up with a strategy by only inspecting some part of the game tree. Similarly, the following definitions fix the question whether one can find a controller by only inspecting a finite amount of the process' behaviour (i.e. the corresponding infinite game tree).

We emphasize again that by the equivalence theorems in [13] in the following definitions the infinite recursive trees correspond to control problems (or infinite games), and the infinite recursive branches to the correct controllers (or winning strategies).

Definition 3.1. As learner we consider Turing machines M^A which have access to an oracle A and converge for every oracle and every input. These machines are intended to learn an infinite recursive branch of a tree $T \in \mathit{TREE}$. As input we feed the characteristic function of T into M^A such that M^A outputs a sequence of guesses $h_0 h_1 \ldots$, where each h_n is computed from $T \restriction n$, i.e., $h_n = M^A(T \restriction n)$. The guesses h_n should describe some infinite recursive branch of T according to the given learning criterion. The machine may also output a special symbol "?" to indicate that it has yet not seen enough data to make up its mind.

In the offline versions of branch learning (e.g. *BranchFin* below) the output of the learner is interpreted as a program for an infinite recursive branch of T, while in the online version the output is directly interpreted as nodes of an infinite recursive branch of T:

Definition 3.2. M^A *finitely A-branch learns* a tree T if on input T the machine M^A produces a sequence of guesses $? \ldots ? eee \ldots$ such that φ_e is an infinite recursive branch of T.

A class of trees $\mathcal{C}$ is *finitely A-branch learnable* ($\mathcal{C} \in \mathit{BranchFin}[A]$) if there is a machine M^A which finitely branch learns every $T \in \mathcal{C}$.

We write *BranchFin* for *BranchFin*[$\emptyset$]. Analogously, for the other criteria considered in this paper we write *Crit* instead of *Crit*[$\emptyset$].

Definition 3.3. M^A *online A-branch learns* a tree T if on input T the machine M^A produces a sequence of guesses $?\ldots?b_0?\ldots?b_1?\ldots$ such that $b_0 b_1 \ldots$ is an infinite recursive branch of T. We will say that the machine *enumerates* the branch $b_0 b_1 \ldots$.

A class of trees $\mathcal{C}$ is *finitely A-online learnable* ($\mathcal{C} \in \mathit{BranchOnl}[A]$) if there is a machine M^A which online branch learns every $T \in \mathcal{C}$.

The following observation holds for all criteria which we consider in this paper, since queries to an oracle A can be simulated by any oracle $B \geq_T A$. But we only state it explicitly for *BranchOnl*:

Fact 3.4. $A \leq_T B \implies \mathit{BranchOnl}[A] \subseteq \mathit{BranchOnl}[B]$.

From [13] we know that $\mathit{BranchFin} \subset \mathit{BranchOnl}$. The following theorems show that this relation relativizes. This also indicates that online learning behaves in some sense similar as finite learning.

Theorem 3.5. $\mathit{BranchFin}[A] \subseteq \mathit{BranchOnl}[B] \iff A \leq_T B$.

Proof. Assume $A \leq_T B$ and $\mathcal{C} \in \mathit{BranchFin}[A]$ via M^A. Then the following procedure online A-branch learns every $T \in \mathcal{C}$ which implies $\mathcal{C} \in \mathit{BranchFin}[B]$ by Fact 3.4:

> *On input* $T\restriction 0, T\restriction 1, \ldots$ *wait until* M^A *outputs it's first real guess* e. *Then enumerate the branch* φ_e.

For the other direction consider the class of trees

$$\mathcal{C} := \{xT \colon T \in \mathit{TREE} \wedge A(x)^\omega \text{ is an infinite branch of } T\}.$$

Clearly, $\mathcal{C} \in \mathit{BranchFin}[A]$: Having seen x output a program for $xA(x)^\omega$. Now assume that $\mathcal{C}$ is in $\mathit{BranchOnl}[B]$ via M^B. We claim that the following procedure decides A in B:

> *On input* x *apply the tree* $T := x(0^\omega + 1^\omega)$ *to* M^B. *Wait until* M^B *enumerates the second node* b_1. *Output* b_1.

Since $T \in \mathcal{C}$ the machine M^B will eventually enumerate a second node b_1. Then the output b_1 is correct, i.e. $b_1 = A(x)$: Otherwise M^B would fail on some tree $x((1 - A(x))^n + A(x)^\omega)$ which is in $\mathcal{C}$. $\square$

From Theorem 3.5 it follows that the *BranchFin* and *BranchOnl* inference degrees coincide with the Turing degrees:

Corollary 3.6. $A \leq_T B \iff \mathit{BranchFin}[A] \subseteq \mathit{BranchFin}[B] \iff \mathit{BranchOnl}[A] \subseteq \mathit{BranchOnl}[B]$

Proof. If $\mathit{BranchFin}[A] \subseteq \mathit{BranchFin}[B]$ or $\mathit{BranchOnl}[A] \subseteq \mathit{BranchOnl}[B]$ then $\mathit{BranchFin}[A] \subseteq \mathit{BranchOnl}[B]$ by Theorem 3.5 and again by Theorem 3.5 we get $A \leq_T B$. The other implications follow from Fact 3.4. $\square$

Thus, the trivial degree of *BranchFin* and *BranchOnl* is the degree of $\emptyset$.

In contrast to Theorem 3.5 there is no oracle A such that *BranchFin*$[A]$ captures *BranchOnl*. This demonstrates that besides some similarities online learning is still a much more powerful concept than finite learning:

Definition 3.7. For $f \in REC$ let $T_f := \{f(0)f(1)\ldots\}$ be the tree which consists exactly of the infinite branch f.

Theorem 3.8. *BranchOnl* $\not\subseteq$ *BranchFin*$[A]$ *for all* $A \subseteq \omega$.

Proof. The theorem follows from inductive inference since $\{T_f\colon f \in REC\}$ is in *BranchOnl*: Given $T\restriction(n+1)$ such that n codes the string $b_0 \ldots b_k$, enumerate b_k if $T(n) = 1$, otherwise enumerate "?". But from $\{T_f : f \in REC\} \in$ *BranchFin*$[A]$ it follows that $REC \in FIN[A]$ which is known to be impossible. □

As a corollary from Theorem 3.8 it follows that *BranchFin* has no omniscient degree.

4 Uniform computation

In this section we study the synthesis of controllers from complete models of the processes:

Definition 4.1. Infinite recursive branches can be *computed uniformly in* A for a class $\mathcal{C} \subseteq TREE$ ($\mathcal{C} \in Uni[A]$), if there is a partial A-recursive function g such that

$$(\forall e)(\forall T \in \mathcal{C})[T = \varphi_e \implies g(e)\downarrow \wedge\ \varphi_{g(e)} \text{ is an infinite branch of } T].$$

In [13] it was shown that *BranchOnl* is strictly included in *Uni*. We now prove that it is impossible to overcome this gap by any oracle A. For the proof we introduce certain families of trees which will also be used later in this paper:

Definition 4.2. For $f \in REC$ we define the tree

$$R_f := \{ef(e)a_0a_1\ldots a_n\colon (\forall m \leq n)[a_m = \mu s[\varphi_{e,s}(m)\downarrow = f(m)]]\}.$$

For $S \subseteq REC$ we set $\mathcal{B}(S) := \{R_f\colon f \in S\}$. Note, that eb is in R_f iff $b = f(e)$.

Lemma 4.3. *For every f the tree R_f is recursive and has infinite recursive branches extending e iff $\varphi_e = f$. Indices for f, R_f and infinite recursive branches of R_f can be computed uniformly from each other. Moreover, enumerations of f and R_f can be translated effectively into each other, i.e., there are computable functions $g_1, \ldots, g_4$ such that $f(n) = g_1(R_f\restriction g_2(n))$ and $R_f(n) = g_3(f\restriction g_4(n))$.*

Theorem 4.4. *Uni* $\not\subseteq$ *BranchOnl*$[A]$ *for all* $A \subseteq \omega$.

Proof. From Lemma 4.3 it follows that $\mathcal{B}(REC)$ is in *Uni*. If $\mathcal{B}(REC)$ were in *BranchOnl*$[A]$ via M^A then REC would be in $FIN[A]$ by the following algorithm, which yields a contradiction:

From the input $f(0)f(1)\ldots$ compute an enumeration of R_f and feed it into M^A. Wait until M^A enumerates the first node b_0 of an infinite recursive branch of R_f. Output b_0.

By Lemma 4.3 the output b_0 is an index of f. $\square$

Thus, like finite learning, *BranchOnl* also has no omniscient degree.

Compared to *BranchFin*$[A]$ uniform computation behaves similar as online learning, at least for $A \leq_T K$:

Theorem 4.5. *For all $A \leq_T K$: $BranchFin[A] \subseteq Uni[B] \iff A \leq_T B$.*

Proof. The proof of *BranchFin* $\subseteq$ *Uni* in [13] relativizes for $A \leq_T B$.

So, it remains to show the *only if* part. Since $A \leq_T K$, by the Limit Lemma there exists a computable $u : \omega^2 \to \omega$ such that $A = \lambda x. \lim_{s\to\infty} u(x,s)$. Consider the class $\mathcal{C}$ of all trees T_x where

$$T_x := \{xia_0a_1\ldots a_n : i \in \{0,1\} \wedge a_0 < a_1 < \ldots < a_n \wedge (\forall m \leq n)[u(x,a_m) = i]\}.$$

Obviously, $\mathcal{C}$ is in *BranchFin*$[A]$. Now assume $\mathcal{C} \in Uni[B]$ via some partial B-recursive function g. We choose an $h \in REC$ with $T_x = \varphi_{h(x)}$ for all x. Then $\lambda x.\varphi_{g(h(x))}(1)$ decides A relative in B. $\square$

The omniscient degree of *Uni* has already been solved in [13]:

Fact 4.6. *$TREE \in Uni[A] \iff A \geq_T K'$.*

The more difficult part ($\Rightarrow$) follows also from Theorem 7.8 below.

A corollary of Theorem 4.5 is that the degree-structure of *Uni* below K coincides with the Turing degrees:

Corollary 4.7. *For all $A \leq_T K$: $Uni[A] \subseteq Uni[B] \iff A \leq_T B$.*

This result can even be strengthened as the following Theorems show:

Theorem 4.8. *(1) $Uni[A] = Uni \iff A$ recursive.*

(2) If $B \not\geq_T K$ then: $Uni[A] \subseteq Uni[B] \iff A \leq_T B$.

Proof. Since (1) follows from (2), we only have to prove (2):

($\Leftarrow$): Follows from Fact 3.4.

($\Rightarrow$): Since every Turing degree contains a retracable set, we assume w.l.o.g. that A is retracable. Now, to get $A \leq_T B$ it suffices to show that A is r.e. in B. We set

$$T_{x,y} := \{xy0^s : y \notin K_s\} \cup \{xyz^\omega : z > 0 \wedge y \in K_z\}$$

and consider the class

$$\mathcal{C} := \{T_{x,y} : (x \in A \wedge y \notin K) \vee (x \notin A \wedge y \in K)\}.$$

C is in $Uni[A]$:

From an index e of a tree $T_{x,y} \in \mathcal{C}$ first extract x and y. If $x \in A$ then we know that $y \notin K$. Thus $xy0^\omega$ is an infinite recursive branch of $T_{x,y}$. If $x \notin A$ it follows $y \in K$. Then xyz^ω with $z = \mu s[y \in K_s]$ is an infinite recursive branch of $T_{x,y}$.

Since $Uni[A] \subseteq Uni[B]$ the class $\mathcal{C}$ is also in $Uni[B]$ via some (partial) machine M^B. We choose an computable $h\colon \omega^2 \to \omega$ with $(\forall x, y)[\varphi_{h(x,y)} = T_{x,y}]$. Consider the set

$$X := \{x\colon (\exists s, y)[y \in K_s \wedge xy0 \preceq \varphi_{M_s^B(h(x,y)),s}]\}$$

of all x such that for some $y \in K$ the machine M^B on input $T_{x,y}$ outputs a branch (or a partial function) beginning with $xy0$. X is recursively enumerable in B. We claim that $A = X$:

If $x \notin A$ then for all $y \in K$ the machine M^B will on input $T_{x,y}$ output a branch beginning with xyz for some $z > 0$ since $T_{x,y} \in \mathcal{C}$ and $T_{x,y}$ has no infinite branch beginning with $xy0$. Therefore x is not in X. It follows that $X \subseteq A$.

Now let $x \in A$. Then for all $y \notin K$ the machine M^B will on input $T_{x,y}$ output a branch beginning with $xy0$ since $T_{x,y} \in \mathcal{C}$ and $xy0^\omega$ is the only infinite recursive branch of $T_{x,y}$. Assume that there is no $y \in K$ such that M^B on input $T_{x,y}$ outputs an i with $xy0 \preceq \varphi_i$. Then $\overline{K} = \{y\colon (\exists s)[xy0 \preceq \varphi_{M_s^B(h(x,y)),s}]\}$ and thus $\overline{K}$ is r.e. in B. Since K is r.e. it follows $K \leq_T B$ which contradicts the assumption $K \not\leq_T B$. Hence, there is an $y \in K$ with $xy0 \preceq \varphi_{M^B(h(x,y))}$ which implies $x \in X$. We get $A \subseteq X$ which completes the proof of $A = X$.

Thus, A is r.e. in B and, since A is retracable, A is in fact Turing reducible to B. $\square$

By using different trees the above proof can be adapted to cover the case $A, B \geq_T K$:

Theorem 4.9. *For all $A, B \geq_T K$: $Uni[A] \subseteq Uni[B] \iff A \leq_T B$ or $B \geq_T K'$.*

Proof. ($\Leftarrow$): Follows from Fact 3.4 and Fact 4.6.

($\Rightarrow$): Assume that $B \not\geq_T K'$ and w.l.o.g. that A is retracable. We define

$$\begin{aligned} T^e_{\mathrm{fin}} &:= \{k^n\colon \#W_{e,n} < k\}, \\ T^e_{\mathrm{inf}} &:= \{0a_0a_1\ldots a_n\colon (\forall m \leq n)[\#W_e > m \wedge a_m = \mu s[\#W_{e,s} > m]]\}, \\ T_e &:= T^e_{\mathrm{fin}} \cup T^e_{\mathrm{inf}}. \end{aligned}$$

T^e_{fin} has an infinite recursive branch iff W_e is finite and T^e_{inf} has an infinite branch iff W_e is infinite. Thus, all T_e have an infinite recursive branch. Moreover, if β is an infinite recursive branch of T_e then $\beta(0) = 0$ iff W_e is infinite.

Set $T_{x,y} := xyT_y$. $Fin := \{e\colon W_e \text{ finite}\}$ and $Inf := \{e\colon W_e \text{ infinite}\}$ are the index sets of the finite and infinite r.e. sets, respectively. We consider the class

$$\mathcal{C} := \{T_{x,y}\colon (x \in A \wedge y \in Fin) \vee (x \notin A \wedge y \in Inf)\}.$$

C is in $Uni[A]$:

From an index e of a tree $T_{x,y} \in \mathcal{C}$ first extract x and y. If $x \in A$ then we know that $y \in Fin$. Since $A \geq_T K$, we can use a K-oracle to compute a number k with $(\forall n)[\#W_{y,n} < k]$. Then xyk^ω is an infinite recursive branch of $T_{x,y}$. If $x \notin A$ it follows that $y \in Inf$. Then $xya_0a_1\ldots$ with $a_m = \mu s[\#W_{y,s} > m]$ for $m = 0, 1, \ldots$ is an infinite recursive branch of $T_{x,y}$.

Since $Uni[A] \subseteq Uni[B]$ the class $\mathcal{C}$ is also in $Uni[B]$ via some (partial) machine M^B. We choose an computable $h\colon \omega^2 \to \omega$ with $(\forall x, y)[\varphi_{h(x,y)} = T_{x,y}]$. Since Fin is r.e. in K there is a B-recursive approximation $(Fin_s)_{s\in\omega}$ of Fin. Consider the set

$$X := \{x\colon\ (\exists s, y)[y \in Fin_s \wedge xy0 \preceq \varphi_{M_s^B(h(x,y)),s}]\}$$

of all x such that for some $y \in Fin$ the machine M^B on input $T_{x,y}$ outputs a branch (or a partial function) beginning with $xy0$. X is recursively enumerable in B. We claim that $A = X$:

If $x \notin A$ then for all $y \in Fin$ the machine M^B will on input $T_{x,y}$ output a branch beginning with xyk for some $k > 0$ since $T_{x,y} \in \mathcal{C}$ and $T_{x,y}$ has no infinite branch beginning with $xy0$. Therefore x is not in X. It follows that $X \subseteq A$.

Now let $x \in A$. Then for all $y \in Inf$ the machine M^B will on input $T_{x,y}$ output a branch beginning with $xy0$ since $T_{x,y} \in \mathcal{C}$ and $xy0^\omega$ is the only infinite recursive branch of $T_{x,y}$. Assume that there is no $y \in Fin$ such that M^B on input $T_{x,y}$ outputs an i with $xy0 \preceq \varphi_i$. Then $Inf = \{y\colon\ (\exists s)[xy0 \preceq \varphi_{M_s^B(h(x,y)),s}]\}$ which implies that Inf is r.e. in B. Since Fin is r.e. in K, and thus r.e. in B, it follows $K' \equiv Inf \leq_T B$ which contradicts the assumption $K' \not\leq_T B$. Hence, there is an $y \in K$ with $xy0 \preceq \varphi_{M^B(h(x,y))}$ which implies $x \in X$. We get $A \subseteq X$ which completes the proof of $A = X$.

Thus, A is r.e. in B and, since A is retracable, A is in fact Turing reducible to B. □

In summary, except the case $(A|_T K \wedge B \geq_T K)$, we were able to prove that $Uni[A] \subseteq Uni[B]$ iff $A \leq_T B$ or $B \geq_T K'$. The following theorem shows that this proposition indeed does not hold for arbitrary A, B:

Theorem 4.10. *There are oracles A and B not above K' such that $Uni[A] \subseteq Uni[B]$ but $A \not\leq_T B$. In fact, even $A \not\leq_T B'$ can be achieved.*

Proof. There are uncountably many Turing degrees above K which are hyperimmune-free relative to K, i.e., every total function computed relative to such a degree is dominated by a total function computable relative to K. Thus there is some oracle A' which is hyperimmune-free relative to K but is not below K''. A' is (Turing equivalent to) the jump of some oracle A by the jump inversion Theorem [24, Theorem V.2.24]; moreover, this A can be chosen such that $A' \equiv_T A \oplus K$. Thus, $A \not\leq_T K''$ since otherwise $A' \equiv_T A \oplus K \leq_T K''$.

By an relativization of the Low Basis Theorem to K [24, Theorem V.5.32] there is an oracle B such that B is PA-complete relative to K and $K <_T B <_T K'$. Now these oracles A and B satisfy the required properties:

(i) A and B are not above K': K' and any degree above it are hyperimmune relative to K; thus $A \not\geq_T K$. Furthermore $B \not\geq_T K'$ by the choice of B.

(ii) $A \not\leq_T B$: This also follows from the choice of A and B. A is not below K'' by the choice of A and since $B \leq_T K'$, A is not below B and also not below B'.

(iii) $Uni[A] \subseteq Uni[B]$: Let $\mathcal{C} \in Uni[A]$ via a partial A-recursive function. Then this function has a total extension f relative to A'. By the choice of A', f has a K-recursive majorant g. Thus, every tree $T \in \mathcal{C}$ given as φ_e has an infinite branch with an index below $g(e)$. Now let

$$h(i,s) = \max\{x : (\forall y < x)[\varphi_{i,s}(y)\downarrow] \wedge \varphi_i{\restriction}x \in \varphi_e\}.$$

Note that φ_i is an infinite branch of T iff $h(i,s)$ converges to ∞ for $s \to \infty$. Since B is PA-complete in K, the oracle B has an algorithm $B(e,j)$ which solves the following problem:

> $B(e,j)$ outputs always a number $i \leq j$ such that $h(i,s)$ converges to ∞ whenever some index below j has this property.

So $B(e,j)$ finds always an index of some infinite branch of T whenever this tree has such a branch with index below j. Taking now $B(e,g(e))$, this algorithm outputs an index for an infinite branch of φ_e whenever some index below $g(e)$ identifies such a branch — and this is true for all trees $\varphi_e \in S$. Thus, $S \in Uni[B]$ and $Uni[A] \subseteq Uni[B]$. □

From Theorems 3.5 and 4.5 we get:

Corollary 4.11. *For all $A \leq_T K$: $BranchOnl[A] \subseteq Uni[B] \implies A \leq_T B$.*

We will now show that the other direction in Corollary 4.11 does not hold in general, i.e. that the inclusion *BranchOnl* $\subseteq$ *Uni* from [13] does not relativize. The intuitive reason is that the *Uni*-machine can only ask finitely many queries to its oracle while the *BranchOnl*-machine may ask infinitely many queries during the enumeration of a branch.

Theorem 4.12. *For all PA-complete A: $BranchOnl[A] \subseteq Uni[B] \iff K' \leq_T B$, i.e. $TREE \in Uni[B]$.*

Proof. Kummer and Stephan [14] have constructed a family of 0,1-valued functions $\{\varphi_{g(i)}\}_{i\in\omega}$ ($g \in REC$) such that

- $1^i0 \preceq \varphi_{g(i)}$,
- $\varphi_{g(i)}(x)$ is undefined for at most one x,
- if W_i is finite and φ_e is a total extension of $\varphi_{g(i)}$ then $e \geq \#W_i$.

For every i we define a recursive *binary* tree T_i according to

$$T_i := \{a_0 \ldots a_n\colon\ 1^i0 \preceq a_0 \ldots a_n \wedge (\forall m \leq n)[\neg(\varphi_{g(i),n}(m)\downarrow \neq a_m)]\}.$$

Note that the only infinite recursive branches of T_i are the total recursive 0,1-valued extensions of $\varphi_{g(i)}$.

Consider the recursive function $f(i,\sigma)$ which checks simultaneously whether the subtrees above $\sigma 0$ or $\sigma 1$ in T_i are finite and outputs $(1-j)$ if it detects first that the subtree above σj is finite for $j \in \{0,1\}$. $f(i,\sigma)$ is undefined if none of the two subtrees is finite.

Since A is PA-complete there is a $0,1$-valued A-recursive extension h of $f(i,\sigma)$. Thus, we have for all σ:

$$\{\tau \in T_i\colon \sigma \preceq \tau\} \text{ infinite} \implies \{\tau \in T_i\colon \sigma h(i,\sigma) \preceq \tau\} \text{ infinite}.$$

Now, $\mathcal{C} := \{T_i : i \in \omega\}$ is in *BranchOnl*$[A]$ via a machine M^A which simply follows the A-recursive function h after it has decoded i from the beginning of the input tree.

Assume now that $\mathcal{C}$ is in $Uni[B]$ via some partial B-recursive function ψ. Then the index set $Inf := \{i \in \omega : W_i \text{ infinite}\}$ is r.e. in B, since $Inf = \{i : (\exists s)[\#W_{i,s} > \psi(u(i))]\}$ where $u \in REC$ with $(\forall i)[T_i = \varphi_{u(i)}]$. Note, that $range(u) \subseteq dom(\psi)$. If W_i is infinite then clearly there is an s with $\#W_{i,s} > \psi(u(i))$. And if there exists an s with $\#W_{i,s} > \psi(u(i))$ then W_i must be infinite since $\varphi_{\psi(u(i))}$ is a total extension of $\varphi_{g(i)}$.

It remains to show that $Fin = \{i\colon W_i \text{ finite}\}$ — the complement of Inf — is also r.e. in B. It is well known that Fin is r.e. in K. Therefore it suffices to prove $K \leq_T B$.

Let $W_{v(x)} := \{s\colon x \notin K_s\}$ $(v \in REC)$. Then for all x the $Uni[B]$-procedure ψ computes an index $e := \psi(u(v(x)))$ for an infinite recursive branch of $T_{v(x)}$ with:

$$x \in K \iff W_{v(x)} \text{ finite} \iff \#W_{v(x)} \leq e \iff x \in K_e. \qquad \square$$

5 Uniform Computation via Total Functions

In the definition of $Uni[A]$ (Definition 4.1) we allowed the uniform computation procedure to be partial, i.e. the procedure may not converge on inputs which are not indices for a tree in the class under consideration. Certainly, this is a natural approach since it abstracts from strange inputs. Actually, for the class *Uni* (i.e. A recursive) there is no difference if we require that the uniform computation procedures should be total. In this case, every uniform computation procedure is simply a computable program transformation, which can always be made total by using, e.g., the S^m_n-Theorem. But for non-recursive oracles the two notions does not coincide as we will see shortly. We also give further examples in this section which show that the variant of *Uni*, where the uniform computation procedures are required to be total, behaves very differently than *Uni*.

Definition 5.1. $\mathcal{C} \subseteq TREE$ is in $TUni[A]$ if $\mathcal{C} \in Uni[A]$ via some *total* A-recursive function.

Let us first write down some simple observations:

Fact 5.2. *(1)* $TUni[A] \subseteq Uni[A]$,

(2) $TUni = Uni$,

(3) $Uni[A] \subseteq TUni[A']$.

(3) holds since for every partial A-recursive function g one can check relative in A' whether $g(e)\downarrow$.

The following theorem shows that the omniscient degree of $TUni$ is one Turing jump higher than that of Uni:

Theorem 5.3. *$TREE \in TUni[A] \iff A \geq_T K''$.*

Proof. ($\Leftarrow$): By Fact 4.6 the class $TREE$ is in $Uni[K']$ via some partial function $f \leq_T K'$. We set

$$g := \lambda e. \begin{cases} 0 & \text{if } f(e)\uparrow, \\ f(e) & \text{otherwise.} \end{cases}$$

Then, g is total, recursive in K'' and g uniformly computes infinite recursive branches for $TREE$.

($\Rightarrow$): Define

$$T_e := \{ka_0 \dots a_n \colon (\forall m \leq n)[a_m = \mu s[\{k, \dots, k+m\} \in W_{e,s}]]\}.$$

T_e contains infinite recursive branches iff W_e is cofinite. Moreover, if $\beta \in REC$ is an infinite branch of T_e then $\max \overline{W_e} < \beta(0)$. It follows that $\mathcal{C} := \{T_e : W_e \text{ cofinite}\}$ is a subclass of $TREE$. Assume, that $TREE \in TUni[A]$ via the total function $f \leq_T A$. Then, we have also $\mathcal{C} \in TUni[A]$ via f. Define $g \in REC$ according to $\varphi_{g(k)} := \lambda m.\mu s[\{k, \dots, k+m\} \in W_{e,s}]$. Thus, $\max \overline{W_e} < k$ iff $\varphi_{g(k)}$ is total. We choose an $h \in REC$ with $(\forall e)[\varphi_{u(e)} = T_e]$. Now, if W_e is cofinite then $\varphi_{f(u(e))}$ is an infinite recursive branch of T_e, thus, $k := \varphi_{f(u(e))}(0)$ is defined and $\varphi_{g(k)}$ is total. Otherwise, if W_e is coinfinite then either $k := \varphi_{f(u(e))}(0)$ is undefined or, if it is defined, then $\varphi_{g(k)}$ is not total. At all we get

$$(\forall e)[W_e \text{ is cofinite} \iff k := \varphi_{f(u(e))}(0) \text{ is defined and } \varphi_{g(k)} \text{ is total.}]$$

The test on the right side is recursive in $A \oplus K'$. Note, that we use the totality of f here. But K' is Turing reducible to A by Fact 4.6, since $TREE$ is also in $Uni[A]$ via f, i.e. $A \oplus K' \equiv_T A$. At all we get $K'' \equiv_T \{e : W_e \text{ cofinite}\} \leq_T A \oplus K' \equiv_T A$. $\square$

It follows from Theorem 5.3 that $Uni[K'] \not\subseteq TUni[K']$. The corresponding result also holds for the oracle K:

Theorem 5.4. *$Uni[K] \not\subseteq TUni[K]$.*

Proof. Let $f \leq_T K$ be a partial 0, 1-valued function which has no total K-recursive extension. Since f is partial recursive in K, by the Limit Lemma there is a total recursive $h \colon \omega^2 \to \omega$ with $f = \lambda x. \lim_{s\to\infty} h(x, s)$ and a so called modulus function $m \leq_T K$ satisfying $(\forall s \geq m(x))[h(x, s) = f(x)]$. Let

$$T_x := \{xik^n \colon i \in \{0, 1\} \wedge (\forall m \leq n)[h(x, k+m) = i]\}.$$

For all $x \in dom(f)$ the tree T_x has an infinite recursive branch $\beta \succeq xi$ iff $i = f(x)$. We set $\mathcal{C} := \{T_x : x \in dom(f)\}$. Since $xf(x)m(x)^\omega$ is an infinite recursive branch of T_x for all $x \in dom(f)$, the class $\mathcal{C}$ is in $Uni[K]$.

Assume now that $\mathcal{C} \in TUni[K]$ via some total function $g \leq_T K$. Let $h \in REC$ with $(\forall x)[\varphi_{h(x)} = T_x]$ be given. Consider the function

$$F := \lambda x. \begin{cases} \varphi_{g(h(x))}(1) & \text{if } \varphi_{g(h(x))}(1)\downarrow, \\ 0 & \text{otherwise.} \end{cases}$$

Since g is total, the test whether $\varphi_{g(h(x))}(1)$ is defined is recursive in K. Thus, F is a total K-recursive function. If $x \in dom(f)$ then $\varphi_{g(h(x))}$ is an infinite recursive branch of T_x which implies $\varphi_{g(h(x))}(1)\downarrow = f(x)$. I.e. F is a total K-recursive extension of f, which is a contradiction. □

Note that $Uni[K''] = Uni[K']$ while $TUni[K''] \not\subseteq TUni[K']$. Furthermore, Theorem 4.10 generalizes to $TUni$ since it actually shows that $TUni[A']$ is contained in $TUni[B]$. Thus it provides a counterexample to a generalization of Theorem 4.9 to $TUni$. So the following holds:

Theorem 5.5. *The structures "$Uni[A] \subseteq Uni[B]$" and "$TUni[A] \subseteq TUni[B]$" are incomparable: There are A_1, B_1 with $Uni[A_1] \subseteq Uni[B_1]$, $TUni[A_1] \not\subseteq TUni[B_1]$ and A_2, B_2 with $Uni[A_2] \not\subseteq Uni[B_2]$ and $TUni[A_2] \subseteq TUni[B_2]$.*

Nevertheless, Theorem 4.8 can be generalized to $TUni$:

Theorem 5.6. *(1) $TUni[A] = TUni \iff A$ recursive.*

(2) If $B \not\geq_T K$ then: $TUni[A] \subseteq TUni[B] \iff A \leq_T B$.

Sketch of proof. If $x \geq 1$ we write $\overline{\log}(x)$ for the unique n with $2^n \leq x < 2^{n+1}$. Adapt the trees $T_{x,y}$ and the class $\mathcal{C}$ of Theorem 4.8 in the following way:

$$T_{x,y} := \{xy0^s \colon \overline{\log}(y) \notin K_s\} \cup \{xyz^\omega \colon z > 0 \wedge \overline{\log}(y) \in K_z\},$$

$$\mathcal{C} := \{T_{x,y} \colon A(\overline{\log}(x)) \neq K(\overline{\log}(y)) \wedge \overline{\log}(x) \leq 2^{e+100} \text{ for all programs } e \text{ of } T_{x,y}\}.$$

Note, that the $Uni[A]$ procedure in the proof of Theorem 4.8 on input $T_{x,y}$ uses A only to compute $A(x)$. There is a total function $g \in REC$ such that $x \leq g(e)$ for all indices e of trees $T_{x,y} \in \mathcal{C}$. This allows to adapt the old procedure of Theorem 4.8 to $TUni[A]$: The $TUni[A]$-learner generates an index which contains the finite string $A{\restriction}g(e)$ and simulates the whole $Uni[A]$-learning procedure as part of the program execution. If this simulation fails by querying some x beyond $g(e)$ then it knows that e is not an index of some tree in $\mathcal{C}$ and just stops to do anything, in the other case it simulates the function whose index is generated by the $Uni[A]$-learner.

The rest of the proof can be adapted to work also for the new definitions of $T_{x,y}$ and $\mathcal{C}$. Hereby, the set X is now defined as follows:

$$X := \{a : (\exists s, b)[b \in K_s \wedge xy0 \preceq \varphi_{M_s^B(h(x,y)),s} \text{ for the majority of all } x, y \text{ with } a = \overline{\log}(x) \wedge b = \overline{\log}(y)]\}.$$

□

6 EX-style Branch Learning

Of course, finite learning is a very restricted kind of learning. The learner gets more power if he only has to (syntactically) learn a controller *in the limit* [10]:

Definition 6.1. M^A *EX*[*A*]*-branch learns* a tree T if on input T the machine M^A produces a sequence of guesses $h_0 h_1 \dots h_n eee \dots$ such that φ_e is an infinite recursive branch of T.

A class of trees $\mathcal{C}$ is *EX*[*A*]*-branch learnable* ($\mathcal{C} \in \mathit{BranchEx}[A]$) if there is a machine M^A which $EX[A]$-branch learns every $T \in \mathcal{C}$.

The following results can be obtained by modifying the proofs from the corresponding results in inductive inference [1, 9, 14]. As in [9] we write $\mathcal{G}(A)$ if $A \leq_T G \leq_T K$ for some 1-generic set G, i.e. if A is either recursive or has the same degree as a 1-generic Turing degree below K.

Fact 6.2.

1. $A \leq_T K \implies \mathit{BranchFin}[A] \subset \mathit{BranchEx}$.
2. *For all A:* $\mathit{BranchEx}_1 \not\subseteq \mathit{BranchFin}[A]$, *where* $\mathit{BranchEx}_1$ *means EX-branch learnable with at most one mind change.*
3. $\mathit{BranchEx}[A] = \mathit{BranchEx} \iff \mathcal{G}(A)$.
4. $\mathit{TREE} \in \mathit{BranchEx}[A] \iff A$ *is high.*
5. *For all r.e. A:* $\mathit{BranchEx}[A] \subseteq \mathit{BranchEx}[B] \iff A \leq_T B$ *or B is high.*

From [13] we know that *BranchEx* is incomparable with *BranchOnl* and *Uni*. In analogy to the case of *Uni* (Theorem 4.4) it is also impossible to capture $\mathit{BranchEx}_1$ by $\mathit{BranchOnl}[A]$ for any oracle A:

Theorem 6.3. *For all A:* $\mathit{BranchEx}_1 \not\subseteq \mathit{BranchOnl}[A]$.

Proof. Consider the class $\mathcal{C} := \{T \in \mathit{TREE} : 0^\omega$ or 1^ω is a branch of $T\}$. $\mathcal{C}$ is in $\mathit{BranchEx}_1$: output a program for 0^ω until you find an m with $T(0^m) = 0$. Then output a program for 1^ω.

But a $\mathit{BranchOnl}[A]$-learner M^A for $\mathcal{C}$ will eventually output a first node on input $0^n + 1^n$ because $0^\omega + 1^\omega \in \mathcal{C}$. This node will be the same for $0^n + 1^\omega$ and $0^\omega + 1^n$. Thus, on one of the two trees M^A fails to enumerate an infinite recursive branch. □

It is fundamental that *Uni* and *BranchOnl* are not included in *BranchEx*: The class $\{T_f : f \in \mathit{REC}\}$ (see Definition 3.7) is in $\mathit{Uni} \cap \mathit{BranchOnl}$ but not in *BranchEx* since *REC* is not in *EX*. This fundamental difference between learning in the limit on the one side and online learning and uniform computation on the other side is emphasized by the following result, which shows that only omniscient oracles enable *BranchEx* to overcome this difference.

Theorem 6.4. *A high* $\iff \mathit{Uni} \subseteq \mathit{BranchEx}[A] \iff \mathit{BranchOnl} \subseteq \mathit{BranchEx}[A]$.

Proof. If A is high then $Uni \subseteq BranchEx[A] = TREE$ by Fact 6.2.4 and if $Uni \subseteq BranchEx[A]$ then $BranchOnl \subseteq BranchEx[A]$ since $BranchOnl \subset Uni$ [13].

If $BranchOnl$ is included in $BranchEx[A]$ then $\{T_f : f \in REC\}$ is in $BranchEx[A]$. This implies $REC \in EX[A]$ and thus A high [9]. □

On the other side there are EX-branch learnable classes of trees for which infinite recursive branches cannot be computed uniformly. But in contrast to Theorem 6.4 Uni does not need an omniscient oracle to capture $BranchEx$:

Corollary 6.5. *$BranchEx \subseteq Uni[A] \iff K \leq_T A$.*

Proof. If $BranchEx \subseteq Uni[A]$ then $BranchFin[K] \subseteq Uni[A]$ (Fact 6.2.1) which implies $K \leq_T A$ by Theorem 4.5. The other direction follows from [13, Proposition 19]. □

In analogy to the results about omniscient degrees, the oracles which give $TUni$ as much power as $BranchEx$ are again one Turing jump higher than that of Uni:

Theorem 6.6. *$BranchEx \subseteq TUni[A] \iff K' \leq_T A$.*

Proof. ($\Leftarrow$) follows from Corollary 6.5 and Fact 5.2.3:

$$BranchEx \subseteq Uni[K] \subseteq TUni[K'].$$

For the other direction we slightly modify the trees T^e_{fin} of Theorem 5.3 and code e into the beginning of the trees:

$$T_e := \{ek^s\colon \#W_{e,s} \leq k\}.$$

Recall that T_e has an infinite recursive branch iff W_e is finite, and $\#W_e \leq \beta(1)$ for every infinite branch of T_e. The class $\mathcal{C} := \{T_e\colon W_e \text{ is finite}\}$ is in $BranchEx$:

> *First, decode e from the input. Then, in stage n output a program for ek^ω where $k = \#W_{e,n}$*

If W_e is finite — i.e. $T_e \in \mathcal{C}$ — then there exists an s with $W_e = W_{e,s}$. Thus, in all stage $n \geq s$ the procedure will output the same program which computes an infinite recursive branch of T_e.

By hypothesis we get $\mathcal{C} \in TUni[A]$ via some total $g \leq_T A$. Now the following algorithm decides $Fin = \{e\colon W_e \text{ finite}\}$:

(i) Input: e

(ii) Compute an index i of T_e.

(iii) Let $j := g(i)$, i.e. an index of an infinite branch of T_e in the case that W_e is finite.

(iv) Compute $k := \varphi_j(1)$ if defined.

(v) If $k = \varphi_j(1)$ is undefined or there is an s with $\#W_{e,s} > k$ then output "W_e is infinite", otherwise output "W_e is finite".

If W_e is finite then $k = \varphi_j(1)$ is defined and $k \geq \#W_e$. Thus, in this case the algorithm terminates with output "W_e is finite".

If W_e is infinite then, if $k = \varphi_j(1)$ defined, there is an s with $\#W_{e,s} > k$. Thus, in this case the algorithm terminates with output "W_e is infinite".

Let us now analyze the complexity of the algorithm. Step (iii) uses the oracle A, and the steps (iv) and (v) are recursive in K. Since $\mathit{BranchEx} \subseteq \mathit{TUni}[A] \subseteq \mathit{Uni}[A]$ we can conclude from Corollary 6.5 that $K \leq_T A$. Thus, the algorithm is recursive in A and $K' \equiv_T \mathit{Fin} \leq_T A$. □

Note, that Theorem 5.4 can also be obtained as a direct conclusion from Corollary 6.5 and Theorem 6.6.

7 BC- and Weak BC-style Branch Learning

In contrast to EX-style learning in BC-style learning the learner has only to converge *semantically* to a correct controller. Note, that there may be many correct controllers. This is the reason why there are two notions of BC-style branch learning, while there exists only one notion of BC-style function learning.

Definition 7.1. M^A $BC[A]$-*branch learns* a tree T if on input T the machine M^A produces a sequence of guesses $h_0 h_1 \ldots$ such that there is an infinite recursive branch f of T with $\varphi_{h_n} = f$ for almost all n.

M^A *weakly* $BC[A]$- or $WBC[A]$-*branch learns* T if φ_{h_n} is an infinite recursive branch of T for almost all n.

$\mathcal{C} \in \mathit{BranchBC}[A]$ and $\mathcal{C} \in \mathit{BranchWBC}[A]$ for classes $\mathcal{C} \subseteq \mathit{TREE}$ are defined similar to the previous definitions (e.g. $\mathcal{C} \in \mathit{BranchEx}[A]$ in Definition 6.1).

As in the case of finite versus EX-style branch learning it follows directly from $EX \subset BC$ that $\mathit{BranchEx} \subset \mathit{BranchBC}$. The weak version of BC-style learning does not appear in classical inductive inference since there is only one target object — namely the input object itself. It was proven in [13] that $\mathit{BranchBC} \subset \mathit{BranchWBC}$.

For the omniscient BC degrees there is no nice characterization known in inductive inference. And the results in [9] suggest that there exists no nice one. Therefore it is remarkable that such a characterization exists for $\mathit{BranchBC}$ and $\mathit{BranchWBC}$:

Theorem 7.2. *A is high* $\iff$ $\mathit{TREE} \in \mathit{BranchBC}[A]$ $\iff$ $\mathit{TREE} \in \mathit{BranchWBC}[A]$.

Proof. Fact 6.2.4 already states: If A is high then $\mathit{TREE} \in \mathit{BranchEx}[A]$. Since the inclusion $\mathit{BranchEx} \subseteq \mathit{BranchBC} \subseteq \mathit{BranchWBC}$ relativizes to A, any high oracle is omniscient for BranchBC and BranchWBC, too. The trees from [13, Proposition 21] can be used to prove the reverse directions. □

We now summarize the facts which follow from results in inductive inference by modifications of the corresponding proofs [9, 14]:

Fact 7.3. *1. A high $\iff$ BranchBC $\subseteq$ BranchEx$[A]$.*

2. *BranchEx$[A] \subseteq$ BranchBC $\iff \mathcal{G}(A)$.*

3. *BranchBC$[A] =$ BranchBC $\iff \mathcal{G}(A)$.*

4. *For all r.e. A, B: BranchBC$[A] \subseteq$ BranchBC$[B] \iff A \leq_T B$ or B high.*

In Theorem 6.4 we have seen that we need omniscient oracles A to capture *Uni* and *BranchOnl* by *BranchEx*$[A]$. The following result shows that oracles A with $REC \in BC[A]$ suffices to capture *BranchOnl* by *BranchBC*$[A]$ and both, *BranchOnl* and *Uni*, by *BranchWBC*$[A]$. This demonstrates the power of *BC* and even more the power of *BranchWBC* (capturing both), since only high oracles are omniscient for *BranchBC* and *BranchWBC* (Theorem 7.2) — but there are low sets A with $REC \in BC[A]$ ([9]).

Theorem 7.4. *The following are equivalent:*

(*1*) $REC \in BC[A]$,

(*2*) *BranchOnl $\subseteq$ BranchBC$[A]$,*

(*3*) *BranchOnl $\subseteq$ BranchWBC$[A]$,*

(*4*) *Uni $\subseteq$ BranchWBC$[A]$.*

Proof. (1) $\Rightarrow$ (2): Assume $\mathcal{C} \in$ *BranchOnl* via M. On input $T \in \mathcal{C}$ enumerate an infinite branch of T via M and $BC[A]$-learns an index for it.

(2) $\Rightarrow$ (3): Obvious, since *BranchBC*$[A] \subseteq$ *BranchWBC*$[A]$.

(3) $\Rightarrow$ (1): From $\{T_f \colon f \in REC\} \in$ *Uni* $\subseteq$ *BranchWBC*$[A]$ (see Definition 3.7) we directly get $REC \in BC[A]$.

(1) $\Rightarrow$ (4): Assume $\mathcal{C} \in$ *Uni* via g. On input $T \in \mathcal{C}$ we $BC[A]$-learn an index for T, say by the sequence of guesses $h_1 h_2 \dots$. Then $g(h_1)g(h_2)\dots$ is a sequence of guesses such that almost all compute an infinite recursive branch of T.

(4) $\Rightarrow$ (3): Obvious, since *BranchOnl* $\subset$ *Uni*. □

Note that in the proof of (1) $\Rightarrow$ (4) we can not conclude *Uni* $\subseteq$ *BranchBC* since the sequence $h_1, h_2, \dots$ may converge to different indices for T, and the branch computed by g may depend on the indices for T, which g receives as input.

The following theorem shows that we actually need an omniscient oracle A to capture *Uni* by *BranchBC*$[A]$. This gives a measure for the advantage of *WBC*-style over *BC*-style branch learning. This advantage of *WBC*-style over *BC*-style branch learning is additionally demonstrated by the result that for capturing *BranchWBC* by *BranchBC*$[A]$ also an omniscient oracle A is needed. As a corollary we get the existence of classes in *Uni* such that the uniform computation of branches depends on the index of the input tree.

Theorem 7.5. *A is high $\iff$ Uni $\subseteq$ BranchBC$[A] \iff$ BranchWBC $\subseteq$ BranchBC$[A]$.*

Proof. Since high oracles are omniscient for *BranchBC* (Theorem 7.2) we only have to show that $Uni \subseteq BranchBC[A]$ and $BranchWBC \subseteq BranchBC[A]$ imply A high.

Assume that A is not high. Then there is a family of recursive functions $S \in BC - EX[A]$ ([14]). We consider the class $\mathcal{B}(S)$ (see Definition 4.2). $\mathcal{B}(S)$ is in *Uni* by Lemma 4.3. $\mathcal{B}(S)$ is also in *BranchWBC*:

From the input $R_f(0), R_f(1), \dots$ extract an enumeration $f(0), f(1), \dots$ for f. Apply the *BC*-learner on f which yields a sequence $h_0 h_1 \dots$ of guesses for f such that almost all guesses are correct. By Lemma 4.3 there is a $g \in REC$ with $(\forall e)(\forall f)[\varphi_e = f \implies \varphi_{g(e)}$ is an infinite recursive branch of $R_f]$. Thus, $g(h_0)g(h_1)\dots$ is a sequence of guesses such that almost all compute an infinite recursive branch or R_f.

Assume $\mathcal{B}(S) \in BranchBC[A]$ via M^A. We will show that this implies $S \in EX[A]$ which is a contradiction:

Translate the input sequence $f(0), f(1), \dots$ for $f \in S$ into an enumeration $R_f(0), R_f(1), \dots$ (Lemma 4.3). By Applying M^A to $R_f \restriction 0, R_f \restriction 1, \dots$ we get a sequence of guesses $h_0 h_1 \dots$ which $BC[A]$-converges to an infinite recursive branch of R_f. Let $k(n) := \max\{m \leq n : \varphi_{h_m, n}(0)\downarrow\}$. Then $(\varphi_{h_{k(n)}}(0))_{n \in \omega}$ $EX[A]$-converges to an index for f by Lemma 4.3. □

Corollary 7.6. *There exists a class $\mathcal{C} \in Uni$ such that for all g with $\mathcal{C} \in Uni$ via g:*

$$(\exists T \in \mathcal{C})(\exists i, j)[i \neq j \wedge \varphi_i = \varphi_j = T \wedge \varphi_{g(i)} \neq \varphi_{g(j)}].$$

In Corollary 6.5 we have seen that $BranchEx \subseteq Uni[A]$ iff $A \geq_T K$. This result also holds for *BC*-branch learning by the analogous proof, since Proposition 19 from [13] actually states $BranchBC \subseteq Uni[K]$:

Corollary 7.7. $BranchBC \subseteq Uni[A] \iff A \geq_T K$.

What oracle do we need to capture *BranchWBC* by $Uni[A]$? The power of *WBC*-style branch learning appears most clearly under this "*Uni*-oracle measure". The gap between capturing *BranchBC* and *BranchWBC* by $Uni[A]$ is a whole Turing jump. *BranchWBC* is so powerful that only omniscient oracles give $Uni[A]$ as much power:

Theorem 7.8. $BranchWBC \subseteq Uni[A] \iff A \geq_T K'$.

Proof. ($\Leftarrow$) follows by Fact 4.6. For the direction ($\Rightarrow$) we define

$$\begin{aligned} T_e := \{eka_0 \dots a_n \colon\ & (\forall m \leq n)[(\#W_{e,m} \leq k \implies a_m = 0) \wedge \\ & (\#W_{e,m} > k \implies \#W_e \geq m \wedge a_m = \mu t[\#W_{e,t} \geq m])]\}. \end{aligned}$$

Let $U_k^e := \{\sigma \colon ek\sigma \in T_e\}$ be the subtree above ek. If $\#W_e \leq k$ then $U_k = 0^\omega$. If $\#W_e > k$ then U_k contains an infinite recursive branch iff W_e is infinite.

The class $\mathcal{C} := \{T_e \colon e \in \omega\}$ is in *BranchWBC*:

> *Wait until you can decode e from the enumeration of T_e. Then output in stage s a program for $eka_0a_1\dots$ where $k = \#W_{e,s}$ and $a_m = \mu t[\#W_{e,m} \leq k \vee \#W_{e,t} \geq m]$.*

If W_e is finite then there is an s with $W_e = W_{e,s}$. Thus, for $k = \#W_{e,s}$ we have $U_k = 0^\omega$ and all guesses from stage s on will compute the branch $ek0^\omega$. If W_e is infinite then every ekU_k contains an infinite recursive branch and every guess computes such a branch. (Note that in this case the learner produces infinitely many different branches.)

Since $\mathit{BranchWBC} \subseteq \mathit{Uni}[A]$ there is a partial A-recursive g with $\mathcal{C} \in \mathit{Uni}[A]$ via g. We choose an $h \in \mathit{REC}$ with $T_e = \varphi_{h(e)}$ for all e. Note, that $\mathit{range}(h) \subseteq \mathit{dom}(g)$. Then $\mathit{Inf} = \{e : W_e \text{ infinite}\}$ is recursively enumerable in A, since $\mathit{Inf} = \{e : (\exists s)[\#W_{e,s} > \varphi_{g(h(e))}(1)]\}$. Let $k = \varphi_{g(h(e))}(1)$. If $k > \#W_{e,s}$ then ekU_k contains an infinite recursive branch, since $\mathcal{C} \in \mathit{Uni}$ via g. Then W_e must be infinite because of $\#W_{e,s} > k$. And if W_e is infinite then there certainly exists an s with $\#W_{e,s} > \varphi_{g(h(e))}(1)$.

Since $\mathit{BranchBC} \subseteq \mathit{BranchWBC} \subseteq \mathit{Uni}[A]$ we get $K \leq_T A$ by Corollary 7.7. The index set $\mathit{Fin} = \{e : W_e \text{ finite}\}$ is recursively enumerable in K and thus also r.e. in A. I.e. Inf and the complement of Inf are r.e. in A which implies $A \geq_T \mathit{Inf} \equiv_T K'$. □

A similar result can be obtained for the comparison of *BranchWBC* with *TUni*:

Theorem 7.9. *BranchWBC* $\subseteq$ *TUni*$[A] \iff A \geq_T K''$.

Proof. ($\Leftarrow$) follows by Theorem 5.3. For the other direction we slightly modify the trees of Theorem 5.3 by coding e into the beginning of T_e:

$$T_e := \{eka_0 \dots a_n : (\forall m \leq n)[a_m = \mu s[\{k, \dots, k+m\} \in W_{e,s}]]\}.$$

Recall that T_e contains infinite recursive branches iff $\overline{W_e}$ is cofinite, and that every infinite recursive branch β of T_e satisfies $\max \overline{W_e} < \beta(1)$. The class $\mathcal{C} := \{T_e : W_e \text{ cofinite}\}$ is in *BranchWBC*:

> *Decode e from the input. Then in stage n output a program for $\beta_n := ena_0a_1 \dots$ where $a_m = \mu s[\{n, \dots, n+m\} \in W_{e,s}]$.*

If the input tree T_e is in $\mathcal{C}$ — i.e. $\overline{W_e}$ is cofinite — then almost all β_n are infinite branches of T_e.

By hypothesis we get $\mathcal{C} \in \mathit{TUni}[A]$ via some total $g \leq_T A$. Since $\mathit{TUni}[A] \subseteq \mathit{Uni}[A]$ it follows $A \geq_T K'$ by Fact 4.6. Let $h \in \mathit{REC}$ with $(\forall e)[\varphi_{h(e)} = T_e]$ be given. Then W_e is cofinite iff

(i) $\varphi_{g(h(e))}$ is total and

(ii) $\varphi_{g(h(e))}$ is an branch of T_e.

Note, that we use the totality of g since $g(h(e))$ converges for all e not only for e with W_e cofinite, i.e. $T_e \in \mathcal{C}$. The index $g(h(e))$ is computable in A. Step (i) is recursive in $K' \leq_T A$ and step (ii) is recursive in $K <_T A$. At all it follows $K'' \equiv_T \{e : W_e \text{ cofinite}\} \leq_T A$. □

In summary, *BranchBC* and *BranchWBC* have the same omniscient degrees and behave similar when compared to *BranchOnl*. But in the comparisons with *Uni* the two *BC*-style branch learning notions behave very different. The two notions differ also with respect to their trivial degrees:

Theorem 7.10. *1. $BranchWBC[A] = BranchWBC \iff A$ is recursive.*

2. For all $A \leq_T K$: $BranchWBC[A] \subseteq BranchWBC[B] \iff A \leq_T B$ or B high.

Proof of (2). ($\Leftarrow$): Follows from 3.4 and Theorem 7.2.

($\Rightarrow$): Since $A \leq_T K$, by the Limit Lemma there is a total computable function $u : \omega^2 \to \omega$ with $A = \lambda x.\lim_{s\to\infty} u(x,s)$. Let $f \in REC$ be given. We combine the constructions of the trees R_f in 4.2 and the trees T_x in Theorem 4.5 to define recursive trees Q_f:

$$\begin{aligned} Q_f := \{ef(e)a_0\dots a_e b_0 \dots b_n\colon\ &(\forall m \leq n) \\ &[b_m = \mu t[t > b_{m-1} \wedge \varphi_{e,t}(m)\downarrow = f(m) \wedge (\forall x \leq e)[u(x,t) = a_x]]]\}. \end{aligned}$$

Hereby, we set $b_{-1} = -1$. Q_f has an infinite recursive branch β with $e \preceq \beta$ iff $\varphi_e = f$, in which case the infinite recursive branch $\beta \succeq e$ is unique and $eA(0)\dots A(e) \preceq \beta$.

Assume that B is not high. Then there is a family of recursive functions $S \in BC - EX[A]$ ([14]). We consider the class $\mathcal{C} := \{Q_f\colon\ f \in S\}$. Analogously to the trees R_f (Lemma 4.3) one can effectively translate an enumeration of Q_f into an enumeration of f. Moreover, from f one can compute uniformly in A an infinite recursive branch of Q_f:

> *Given the index e of f output a program for $e\varphi_e(e)A(0)\dots A(e)b_0b_1\dots$ where $b_m := (\mu\, t > b_{m-1})[\varphi_{e,t}(m)\downarrow \wedge (\forall x \leq e)[u(x,t) = A(x)]]$.*

Note, that in contrast to the trees R_f, here we need the oracle A to compute from f an infinite recursive branch of Q_f. Now, one shows $\mathcal{C} \in BranchWBC[A]$ similar as in the proof of $\mathcal{B}(S) \in BranchWBC$ (Theorem 7.2): Translate the enumeration for Q_f into one for f. From this enumeration BC-learn an index for f. On the resulting sequence of guesses for f apply the above A-recursive procedure for uniform computation of infinite branches.

Since $BranchWBC[A] \subseteq BranchWBC[B]$ the class $\mathcal{C}$ is also in $BranchWBC[B]$ via some machine M^B.

Assume that M^B converges on all $T \in \mathcal{C}$ to a finite set of infinite recursive branches, i.e. $\{\varphi_{M^B(T\restriction n)}\colon\ n \in \omega\}$ is finite for all $T \in \mathcal{C}$. Then the machine

$$N^B(T\restriction n) := M^B(T\restriction \max\{m \leq n\colon\ \varphi_{M^B(T\restriction m),n}(0)\downarrow\}),$$

where $\max\emptyset = 0$ by convention, converges for all Q_f to a finite set of programs for f, i.e. there is an n_0 such that $\{N^B(Q_f\restriction n) : n \geq n_0\}$ is a finite set of programs, which compute infinite recursive branches of Q_f. Since enumerations for f can be effectively translated into enumerations for Q_f, and $\varphi_{\beta(0)} = f$ for every infinite recursive branch of Q_f, it follows that $S \in FEx[B] = EX[B]$ (see [6]), which is a contradiction.

Hence, there is a tree $T \in \mathcal{C}$ such that M^B converges on T to an infinite set of infinite recursive branches of T, i.e. $\{\varphi_{M^B(T\restriction n)}\colon\ n \in \omega\}$ is infinite. We choose an n_0 such that $M^B(T\restriction n)$ is an infinite recursive branch for all $n \geq n_0$. By definition, the tree T has for all e at most one infinite recursive branch β

with $e \preceq \beta$. Thus, for each $x \in \omega$ there is an $n \geq n_0$ such that $\varphi_{M^B(T\restriction n)}(0) \geq x$. Setting $s(x) := \mu n \geq n_0[\varphi_{M^B(T\restriction n)}(0) \geq x]$ the procedure $\lambda x.\varphi_{M^B(T\restriction s(x))}(x+2)$ decides A relative in B, i.e. $A \leq_T B$. $\square$

Proof of (1). $BranchWBC[A] = BranchWBC$ implies $BC[A] = BC$ via the trees T_f from Definition 3.7:

$$\begin{aligned} &S \in BC[A] \\ &\qquad \Longrightarrow \{T_f\colon f \in S\} \in BranchBC[A] \subseteq BranchWBC[A] = BranchWBC \\ &\qquad \Longrightarrow S \in BC. \end{aligned}$$

Thus, $\mathcal{G}(A)$ and in particular $A \leq_T K$ holds. By (2) we get $A \leq_T \emptyset$. $\square$

8 Equivalence between binary and arbitrary trees

When we consider the concept of branch learning as a model for process learning, it seems more natural to define the trees over a finite alphabet. This is because the states of the process and the actions of the controller actually are bounded. It is clear that trees over a finite alphabet can always be coded as trees over $\{0,1\}$. But in theoretical considerations proofs often get more simple if we work with arbitrary trees, i.e. trees over the infinite alphabet ω. In this section we show that it does not matter whether we base the definition on arbitrary trees or on binary trees. $TREE_{0,1} := \{T \in TREE\colon T \text{ is a binary tree}\}$ denotes the set of all binary trees in $TREE$. We will use the following well known fact (see e.g. [24, Proposition V.5.25]):

Fact 8.1. *There is an infinite recursive binary tree $\widetilde{T}$ without infinite recursive branches. $\widetilde{T}$ has infinitely many leafs (i.e. $\sigma \in \widetilde{T}$ with $\sigma 0, \sigma 1 \notin \widetilde{T}$). The set $L := \{\sigma\colon \sigma \text{ is a leaf of } \widetilde{T}\}$ is decidable, thus there is an effective enumeration of L without repetitions, which we denote by $\sigma_0, \sigma_1, \ldots$.*

Theorem 8.2. *For all $\mathcal{C} \subseteq TREE$, there is a class $\mathcal{B} \subseteq TREE_{0,1}$, such that for all criteria Crit, which we consider in this paper, and all oracles A:*

$$\mathcal{B} \in Crit[A] \iff \mathcal{C} \in Crit[A].$$

Proof. For an arbitrary $T \in TREE$ let

$$S_T := \{\sigma_{a_0} \ldots \sigma_{a_n}\tau\colon a_0 \ldots a_n \in T \wedge \tau \in \widetilde{T} \text{ but } \tau \text{ is not a leaf of } \widetilde{T}\}.$$

S_T is recursive: To decide $\sigma \in S_T$ first compute the decomposition $\sigma = \sigma_{a_0} \ldots \sigma_{a_n}\tau$ such that no prefix of τ is in L. This decomposition is computable and unique, since L is decidable and there are no $\sigma', \sigma'' \in L$ with $\sigma' \prec \sigma''$. If $\sigma' \in L$ then $\sigma' = \sigma_a$ for $a = \mu k[\sigma_k = \sigma']$. Now, $\sigma \in L$ iff $a_0 \ldots a_n \in T$ and $\tau \in \widetilde{T}$.

If $a_0 a_1 \ldots$ is an infinite recursive branch of T, then clearly $\sigma_{a_0}\sigma_{a_1} \ldots$ is an infinite recursive branch of S_T. And if β is an infinite branch of S_T then either

- $\beta = \sigma_{a_0}\sigma_{a_1} \ldots$ for an infinite branch $a_0 a_1 \ldots$ of T. β is recursive iff $a_0 a_1 \ldots$ is recursive.

$$
\begin{tikzcd}
T \arrow[d, "\Psi_1"'] & \beta_T \\
S_T \arrow[r, "M^A", "Crit[A]"'] & \beta_{S_T} \arrow[u, "\Theta_1"']
\end{tikzcd}
\qquad
\begin{tikzcd}
T \arrow[r, "M^A", "Crit[A]"'] & \beta_T \arrow[d, "\Theta_2"] \\
S_T \arrow[u, "\Psi_2"] & \beta_{S_T}
\end{tikzcd}
$$

Figure 1: Reductions between T and S_T.

- or $\beta = \sigma_{a_0} \dots \sigma_{a_n}\gamma$ for a *non*recursive infinite branch γ of $\widetilde{T}$, in which case β is also not recursive.

It follows that S_T is in $TREE_{0,1}$. We set $\mathcal{B} := \{S_T \colon T \in \mathcal{C}\}$.

The idea of the proof is to reduce the $Crit[A]$-procedures for $\mathcal{B}$ and $\mathcal{C}$ to each other as illustrated in Figure 1. E.g., assume $\mathcal{B} \in Crit[A]$ via M^A. From M^A we can build an $Crit[A]$-procedure for $\mathcal{C}$ by translating an input tree $T \in \mathcal{C}$ into the tree S_T (transformation Ψ_1). On S_T we apply the machine M^A which yields an infinite recursive branch β_{S_T} of S_T. From the branch β_{S_T} we then compute an infinite recursive branch β_T of T (transformation Θ_1). The reduction for the other direction (reducing $\mathcal{B}$ to $\mathcal{C}$) works analogously.

Therefore, we have to show that the transformations ψ_i and Θ_i can be done effectively according the the requirements of the different criteria. In general we say that Y_X can be *computed uniformly* from $X \in \mathcal{X}$, where $X \in \mathcal{X}$ and Y_X are decidable sets, if there is a partial recursive function g such that

$$(\forall e)(\forall X \in \mathcal{X})[\varphi_e = X \implies g(e)\downarrow \wedge\, \varphi_{g(e)} = Y_X].$$

Enumerations of (the characteristic functions of) sets $X \in \mathcal{X}$ can be *translated effectively* into enumerations of (the characteristic functions of) Y_X if there are computable functions h_1, h_2 such that for all $X \in \mathcal{X}$ and all $n \in \omega$:

$$Y_X(n) = h_1(X{\restriction} h_2(n)).$$

All learning criteria require that the transformations ψ_i translate enumerations of T and S_T effectively into each other. *Uni* requires that T and S_T can be computed uniformly from each other. For the output transformations Θ_i only *BranchOnl* requires translation of enumerations. All other criteria need uniform computations between β_T and β_{S_T}.

If enumerations for $X \in \mathcal{X}$ can be translated effectively into enumerations for Y_X, then Y_X can obviously also be computed uniformly from X. Thus, it remains to show the statements about effective translations of enumerations.

It follows from the decision procedure for S_T described above that given the value of $T(a_0 \dots a_n)$ one can compute $S_T(\sigma)$ — where $\sigma = \sigma_{a_0} \dots \sigma_{a_n}\tau$ is decomposed as above. Thus, an enumeration of T can effectively be translated into an enumeration for S_T.

The converse — that an enumeration of S_T can be be translated effectively into an enumeration of T — holds since

$$a_0 \dots a_n \in T \iff \sigma_{a_0} \dots \sigma_{a_n} \in S_T.$$

It directly follows from the above argumentation about the infinite branches of T and S_T that the enumerations of infinite recursive branches of T and S_T can be translated effectively into each other. $\square$

Theorem 2.1 from Section 2 directly follows from the above theorem.

Acknowledgements: We would like to thank John Case, Martin Kummer and Martin Riedmiller for helpful discussions and comments.

References

[1] L. Adleman and M. Blum. Inductive inference and unsolvability. *Journal of Symbolic Logic*, 56(3):891–900, 1991.

[2] O. Arnold and K. P. Jantke. Therapy plan generation as program synthesis. In *Proc. 5th Int. Workshop on Algorithmic Learning Theory*, pages 40–55. Springer-Verlag, 1994.

[3] L. Blum and M. Blum. Towards a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[4] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[5] J. Case, S. Kaufmann, E. Kinber, and M. Kummer. Learning recursive functions from approximations. In *EuroCOLT'95*, volume 904 of *LNCS*, pages 140–153. Springer-Verlag, 1995.

[6] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoret. Comput. Sci.*, 25:193–220, 1983.

[7] D. Cenzer and J. Remmel. Recursively presented games and strategies. *Mathematical Social Sciences*, 24:117–139, 1992.

[8] O. Föllinger. *Regelungstechnik*. Hüthig, Heidelberg, 8th edition, 1994.

[9] L. Fortnow, W. Gasarch, S. Jain, E. Kinber, M. Kummer, S. Kurtz, M. Pleszkoch, T. Slaman, R. Solovay, and F. Stephan. Extremes in the degrees of inferability. *Annals of Pure and Applied Logic*, 66:21–276, 1994.

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

[11] S. Kaufmann and M. Kummer. On a quantitative notion of uniformity. In *Mathematical Foundations of Computer Science*, volume 969 of *LNCS*, pages 169–178. Springer-Verlag, 1995.

[12] M. Kummer and M. Ott. Effective strategies for enumeration games. In H. K. Büning, editor, *Proceedings of Computer Science Logic CSL '95*, pages 368–387, Berlin, 1996. Springer.

[13] 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.

[14] M. Kummer and F. Stephan. On the structure of degrees of inferability. *Journal of Computer and System Sciences*, 52(2):214–238, Apr. 1996.

[15] A. H. Lachlan. On some games which are relevant to the theory of recursively enumerable sets. *Annals of Mathematics*, 91(2):291–310, 1970.

[16] D. Luzeaux. Machine learning applied to the control of complex systems. In *8th International Conference on Artificial Intelligence and expert systems applications*, Paris, France, 1996.

[17] D. Luzeaux and E. Martin. Steps or stages for incremental control? In *Symposium on training issues in incremental learning*, Stanford University, CA, USA, 1993. AAAI-93 Spring Symposium Series.

[18] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 95*, volume 900 of *LNCS*, pages 229–242. Springer-Verlag, 1995.

[19] E. Martin. Oracles for learning programs. In *IEEE International Conference on Systems Man Cybernetics*, Le Touquet, France, 1993.

[20] E. Martin, D. Luzeaux, and B. Zavidovique. Learning and control from a recursive viewpoint. In *IEEE International Symposium on Intelligent Control*, Glasgow, Ecosse, 1992.

[21] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[22] W. T. Miller, R. S. Sutton, and P. J. Werbos, editors. *Neural networks for control.* MIT Press, Cambridge, Massachusetts, 1990.

[23] K. S. Narendra and S. Mukhopadhyay. Intelligent control using neural networks. *IEEE Control Systems Magazine*, 12(5):11–18, April 1992.

[24] P. Odifreddi. *Classical Recursion Theory.* North-Holland, Amsterdam, 1989.

[25] D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn.* MIT Press, Cambridge, Massachusetts, 1986.

[26] M. Riedmiller. Learning to control dynamic systems. In R. Trappl, editor, *Proceedings of the 13th. European Meeting on Cybernetics and Systems Research - 1996 (EMCSR '96)*, Vienna, 1996.

[27] J. G. Thistle and W. M. Wonham. Control of infinite behavior of finite automata. *SIAM Journal on Control and Optimization*, 32(4):1075–1097, 1994.

[28] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 133–191. Elsevier Science Publishers B. V., 1990.

[29] W. Thomas. On the synthesis of strategies in infinite games. In *STACS 95*, volume 900 of *LNCS*, pages 1–13. Springer-Verlag, 1995.

[30] W. Thomas and H. Lescow. Logical specification of infinite computations. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency: Reflections and Perspectives*, volume 803 of *LNCS*, pages 583–621. Springer-Verlag, 1993.

[31] D. A. White and D. A. Sofge, editors. *Handbook of Intelligent Control*. Van Nostrand Reinhold, New York, 1992.