The Complexity of Learning Branches and Strategies from Queries

Matthias Ott[1] and Frank Stephan[2]

[1] Institut für Logik, Komplexität und Deduktionssysteme, Universität Karlsruhe, D-76128 Karlsruhe, Germany, Email: m_ott@ira.uka.de *

[2] Mathematisches Institut, Universität Heidelberg, D-69120 Heidelberg, Germany, Email: fstephan@math.uni-heidelberg.de **

Abstract. We study the problems of efficiently learning infinite branches for finite state trees and winninig strategies for closed finite-state games using membership, and branch or strategy queries, respectively. We show that generally no efficient branch learning algorithm exists but we provide such algorithms for several natural cases, in particular for deadend free finite-state trees, the class of trees such that the set of infinite branches has positive measure, and several classes of modulo trees. Furthermore, we find a way to apply Angluin's results about the identification of deterministic finite automata from queries, which yields positive and negative strategy learning results, in particular, we show that the class of deadend free closed finite-state games is efficiently strategy learnable from membership and strategy queries.

1 Introduction

In [6, 10, 13] an inductive inference approach to learning process controllers has been studied. Hereby, the control problem is modeled as a closed, computable game of infinite duration such that the correct controllers correspond exactly to the winning strategies for the first player. It has been shown in [10] that for learners without time and space bounds the resulting strategy learning model is equivalent to learning infinite computable branches of computable trees.

An important open question of this research was to develop a finer model, in which the complexity of strategy and branch learning can be studied, and to provide big natural classes of games/trees, for which strategies/branches can be learned *efficiently*, i.e., in polynomial time.

In the present paper we close this gap, and thus, following [16], we give another example how ideas from inductive inference can be fruitfully brought into computational learning theory. Similar as in inductive inference the computational learning theory literature mainly concentrates on the problems of *identification*, *classification* and *prediction*, and lacks so far an adequate treatment of learning multi-step strategies for reactive systems (i.e., applying learning for

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

** Supported by the Deutsche Forschungsgemeinschaft (DFG) grant Am 60/9-1.

more general problem solving). This kind of learning problems plays a major role in modern machine learning and has, e.g., brought out such important methods as reinforcement learning [8].

We propose a complexity theoretical approach to strategy learning by restricting our domain to games and trees which are represented by *deterministic finite automata* [5,12,15], and by defining a natural and adequate *query learning model* [1–4] for this domain. So we investigate the query and time complexity of branch and strategy learning, where the complexity is measured in the size of the automaton A representing the target tree/game, i.e., the number of states of the minimal automaton equivalent to A. Our work is motivated by that of McNaughton [12] and Lescow [11] who studied the complexity of *synthesizing* strategies for arbitrary *finite-state games* of infinite duration. However, in contrast to [11,12] we only study *closed* finite-state games. In closed games the winning condition says that the first player may never reach a "bad" state. In arbitrary finite-state games the first player wins if the set of nodes visited infinitely often during a play is in the "winning set", which is a family of sets of states.

The branch learning problems for trees are just the one-person specialization of closed, two-person games. With respect to efficient learnability from queries, as considered in this paper, branch learning is no longer equivalent to strategy learning for closed two-person games. However, since many phenomena already occur in these easier "one-player immortality games", we first study the complexity of learning infinite branches for finite-state trees.

2 Learning Branches of Finite-State Trees

With each deterministic finite automata (DFA) A we can naturally associate a tree. This tree consists of all strings u such that A never visits a rejecting state running on input u. If A contains a cycle of accepting states, which is reachable from the starting state via a path of accepting states, then the associated tree contains an infinite branch. In this section we will study whether such an infinite branch can be learned from queries on A.

In the formal definition we can assume w.l.o.g. that A contains exactly one rejecting state:

Definition 1. A finite-state tree (FST) $T = (\Sigma, Q, \tau, q_0, q_\perp)$ over alphabet Σ consists of a set of states Q, a starting state q_0, a rejecting state $q_\perp$ and a transition function $\tau\colon Q \times \Sigma \to Q$, with $\tau(q_\perp, a) = q_\perp$ for all $a \in \Sigma$.

Let $\tau^*: \Sigma^* \to Q$ denote the iterated transition function, i.e., $\tau^*(q, \epsilon) = q$ and $\tau^*(q, a_1 \dots a_n) = \tau(\tau^*(q, a_1 \dots a_{n-1}), a_n)$, where ϵ is the empty word. An infinite string $a_1 a_2 \dots \in \Sigma^\omega$ is an *infinite branch* of T if $(\forall n)[\tau^*(q_0, a_1 \dots a_n) \neq q_\perp]$.

We only call T an FST if T contains an infinite branch. Moreover, we require that each state $q \in Q - \{q_\perp\}$ is reachable from q_0.

For simplicity, we will assume throughout this paper that $\Sigma = \{0, 1\}$. The generalizations to arbitrary finite alphabets are straightforward.

Of course, an FST T can also be considered as a deterministic finite automata, where $q_\perp$ is the only rejecting state, while all other states $q \in Q - \{q_\perp\}$ are accepting states. $\mathcal{L}(T)$ denotes the language accepted by the DFA T. Sometimes, we write $|T|$ instead of $|Q|$ for the the number of states of an FST.

Moreover, an FST T can be interpreted as a one-person game. The player has repeatedly to choose moves $a_1, a_2, \ldots \in \Sigma$ such that T never visits the rejecting state. Thus, the player can only win after infinitely many moves, while every loss happens after finitely many moves. These kind of games have a natural and well known generalization to the two-person case. We will study these two-person games in Section 3. But for simplicity, in this section we will speak of trees and infinite branches instead of one-person games and winning strategies. This allows also to distinguish terminologically between the one- and two-person case.

The following Lemma shows that each FST contains a "normalized" branch, which has the form uv^ω. This is important since such branches have the simple finite representation (u, v). Moreover, this branch can be computed *efficiently* in the number of states of the FST. We call an algorithm *efficient* if it runs in polynomial time.

Lemma 2. *Each FST T contains an infinite branch of the form uv^ω. Moreover, such a branch can be computed efficiently from T.*

2.1 Query Learning Model

In computational learning theory queries are used to model active learning [3]. Hereby, one assumes the existence of a *teacher* (sometimes called *oracle*) which answers the queries of the learner. For motivating our query model we consider the game interpretation of FST's. The task of the learner is to find a winning strategy for an unknown FST T. Thus, the learner may want to test a special strategy f by playing the game according to the strategy f. If f is a winning strategy, the learner succeeds. Otherwise, the player will lose the game after *finitely* many moves and thus, the player gets a counterexample for the strategy f. What we model intuitively in the following definitions is that the player has the possibility to repeatedly play the unknown game and that there is a "referee" who signals when a play is lost:

Definition 3. A *branch query* for an FST T is a query of the form uv^ω, which is answered by "yes" if uv^ω is an infinite branch of T, and by "no" otherwise.

We will also consider the notion of membership query which is widely used in computational learning theory, in particular in the context of learning finite automata, e.g., in [1, 2]. By a membership query the learner can test, whether a finite word is on the tree given by an FST:

Definition 4. A *membership query* $u \in \Sigma^*$ for an FST T is answered by "yes", if $u \in \mathcal{L}(T)$, and by "no" otherwise.

We are looking for algorithms which efficiently learn branches of the form uv^ω for classes of FST's using branch and/or membership queries. The time complexity of such an algorithm is measured in the size n of the target FST T, which is the number of states of the minimal FST equivalent to T, i.e., $n = size(T) = \min\{|T'|: T' \text{ is an FST and } \mathcal{L}(T) = \mathcal{L}(T')\}$. We also consider the membership and branch query complexity of L, which measures the number of queries used by a learning algorithm, when run on an FST of size n. Note that the query complexity is a lower bound for the time complexity of a learning algorithm. If there exists an algorithm, which learns infinite branches for a subclass of FST's w.r.t. a particular complexity measure, we also call this subclass *branch learnable* w.r.t. this complexity measure.

There are several reasonable variants of the query models introduced above. One can show that each of these variants is polynomially equivalent to one of the following three models: learning by branch queries only, learning by membership queries only, and learning by both, branch *and* membership queries. Membership and branch queries are incomparable to each other and both strictly weaker then membership and branch queries together (see Theorem 7 below). This fact is one reason, why we base our study on branch and membership queries as defined in Definitions 3 and 4. Another reason is that there are no (explicit) counterexamples in this query model, and thus the complexity of learning algorithms does not depend on the length of counterexamples.

Rosin and Belew [14] have given an interpretation of their "competitive game learning model" which is very similar to our query model.

2.2 Complexity of Learning Branches for FST's

The following Theorem uses an adversary argument, as e.g., in Angluin's proof showing that deterministic finite automata are not efficiently identifiable from membership queries [1]:

Theorem 5. *There is no algorithm which learns branches for the class of FST's using only polynomially many branch and membership queries.*

Instead of directly learning an infinite branch of an FST, we could also apply an *indirect* approach: first *identify* the FST, i.e., using queries, infer an FST T' which is equivalent to the target FST T, and then use Lemma 2 to compute from T' an infinite branch of T. Thus, Theorem 5 also implies that there exists no identification algorithm for FST's, which uses only polynomially many branch and membership queries.

A further interesting consequence of Theorem 5 is that Angluin's algorithm for learning finite automata from membership and *proper* equivalence queries [2] cannot be restricted to the natural subclass of FST's.

Next, we give an example for a class which is *efficiently branch learnable*. However, this class of FST's is still not efficiently *identifiable*, implying that generally identifying FST's is strictly harder than learning branches for them.

Definition 6. An FST is *deadend free* if for all states $q \in Q - \{q_\perp\}$ there is an $a \in \Sigma$ such that $\tau(q, a) \neq q_\perp$.

Theorem 7. *1. Deadend free FST's are efficiently branch learnable from branch* and *membership queries.*

2. Deadend free FST's are not branch learnable *using either only polynomially many branch* or *polynomially many membership queries.*

3. Deadend free FST's are not identifiable *from polynomially many branch* and *membership queries.*

Hence, Theorem 7 provides a natural example witnessing that learning branches for deadend free FST's from branch *or* membership queries only is strictly harder than learning branches from branch *and* membership queries together.

The following Theorem provides a general sufficient condition for classes which are branch learnable using only polynomially many branch and membership queries. However, in this case we can only state a deterministic learning algorithms in *PSPACE*, while there still exist randomized algorithms which run in polynomial time.

Definition 8. Let μ be the product measure on $\{0,1\}^\omega$ (see, e.g., Halmos [7]), and $B(T)$ be the set of infinite branches of an FST T.

For $0 < \delta < 1$ we define $\mathcal{C}_\delta = \{T\colon T \text{ is an FST and } \mu(B(T)) > \delta\}$, the class of all FST's for which "at least δ of all infinite strings are infinite branches".

Theorem 9. *For each δ, $0 < \delta < 1$, there is an PSPACE-algorithm which learns infinite branches for all $T \in \mathcal{C}_\delta$ using at most polynomially many branch and membership queries. Moreover, there is a randomized algorithm, which learns branches for C_δ with probability at least $1-\epsilon$, running polynomially in size(T) and $\frac{1}{\epsilon}$.*

2.3 Modulo Trees

In this section we investigate a special kind of FST's, which is based on number theoretic ideas. We identify each string $u = b_1 \dots b_n \in \{0,1\}^*$ with the binary number $1\,b_1 \dots b_n = 2^n + \sum_{i=1}^{n} b_i 2^{n-i}$. Thus, the successors of u (as numbers) are $2\cdot u$ and $2\cdot u+1$. Let $p > 0$ and $P \subseteq \{0,\dots,p-1\}$. For strings $u, v \in \{0,1\}^*$, $u \preceq v$ means that u is an initial segment of v. The tree $T_{p,P}$ contains all nodes $x \in \{0,1\}^*$ such that $(y \mod p) \notin P$ for all $y \preceq x$. We call $T_{p,P}$ a modulo tree, if it contains at least one infinite branch.

It is easy to see that each modulo tree is an FST (but not necessarily a deadend free FST). From our complexity theoretic view modulo trees $T_{p,P}$ are especially interesting, since they have succinct representations of size $O((|P|+1) \cdot \log(p))$, while the equivalent FST's generally have size $\Omega(p)$. Knowing p and P, one can compute an infinite branch of a modulo tree $T_{p,P}$ by a simple hull algorithm in time polynomial in $|P| + \log(p)$. The following theorems show that the situation gets more complex, when the complete information about p and P is missing, and one wants to *learn* an infinite branch via branch and membership queries. Hereby, branch and membership queries are defined in the same way as for FST's.

Theorem 10. *A learner who knows the parameter p of a modulo tree can efficiently learn an infinite branch using at most $243 \cdot |P|^2 \cdot (\log(p)+1)$ membership queries. The infinite branch has the form $\sigma\tau^\omega$ where $|\sigma|+|\tau| \leq 241 \cdot |P|^2 \cdot (\log(p)+1)$.*

One might ask whether it is really necessary for the learner to know p. Indeed the above Theorem also holds for a multiplicative p' of p, but, of course, the parameter $\log(p)$ has then to be replaced by $\log(p')$.

Another approach is to provide some nonmembers of $T_{p,P}$ as additional information to the learner. Nonmembers reveal much more information on p and P than members of $T_{p,P}$. A nonmember $y = 2x + a$, $a \in \{0,1\}$, is called *a minimal nonmember of* $T_{p,P}$ if $y \notin T_{p,P}$ while $x \in T_{p,P}$. Indeed, it is possible to efficiently learn an infinite branch via membership queries, if the learner gets a set R of minimal nonmembers as additional information, such that $|R| > |P|$. The learning algorithm runs in time polynomial in the size of the set R, where the size of a set $\{x_1, x_2, \ldots, x_n\}$ is just $n + \log(x_1) + \log(x_2) + \ldots + \log(x_n)$.

Theorem 11. *Given a set R of more than $|P|$ minimal nonmembers of an unknown modulo tree $T_{p,P}$ as additional input, an infinite branch of $T_{p,P}$ can be efficiently learned from branch and membership queries.*

Of course, this result has several variants which restrict R in a similar way. Nevertheless all these approaches have in common that the learner gets some additional information on p. The next theorem shows that the following informations on p are not sufficient to succeed in time polynomial in m using membership queries: "p is a prime such that $2^m < p < 2^{m+1}$ and P contains only one element". This contrasts the situation for deadend free FST's, which are efficiently branch learnable from membership queries and an upper bound on the size of the FST. This difference is due to the fact that modulo trees have a much more compact description than the corresponding automata. Note that for $P = \{x\}$ the modulo tree $T_{p,P}$ is also deadend free.

Theorem 12. *There is no algorithm which learns an infinite branch of a tree $T_{p,\{x\}}$, where p is prime, from a bound m with $2^m < p < 2^{m+1}$ and a polynomial number of polynomial sized membership queries.*

A natural question is whether an infinite branch of a modulo tree can be learned from branch and membership queries, as it is possible for deadend free FST's. But there is one essential difference between FST's and modulo trees: If the automaton has size n and $\sigma\tau^\omega$ is not an infinite branch of the tree then there is a node on $\sigma\tau^\omega$ of size at most $|\sigma| + n \cdot |\tau|$, which is not in $T_{p,P}$. This does not hold for modulo trees with parameter $n = \log(p) + |P|$. The first node on the branch $\sigma\tau^\omega$ outside the tree $T_{p,P}$ may indeed have size exponential in the parameters n, $|\sigma|$ and $|\tau|$. Thus the next theorem gives only a bound on the number of queries but not on the computational complexity.

Theorem 13. *There is an algorithm which learns infinite branches for modulo trees $T_{p,P}$ using polynomially many branch and membership queries (measured in the parameter $n = \log(p) + |P|$). If an upper bound on $\log(p) + |P|$ is given as additional input, it suffices to use polynomially many membership queries.*

3 Two-Person Closed Finite-State Games

As pointed out in Section 2 every finite-state tree which contains an infinite branch can be interpreted as a one-person game: the player has to stay *on* the tree in order to win the game. These one-person games have a natural (and well known) two-person generalization, which is investigated in this section:

Definition 14. A *two-person closed finite-state game* (CFG) $G = (\Sigma, Q, Q_{\mathrm{I}}, Q_{\mathrm{II}}, \tau, q_0, q_\perp)$ over the alphabet Σ consists of a set of states Q, which is partitioned into the three disjoint sets Q_{I}, Q_{II} and $\{q_\perp\}$, a starting state $q_0 \in Q_{\mathrm{I}}$, the losing state $q_\perp$ and a transition function $\tau \colon Q \times \Sigma \to Q$, such that:

- $(\forall q \in Q_{\mathrm{I}})(\forall a \in \Sigma)[\tau(q,a) \in Q_{\mathrm{II}} \cup \{q_\perp\}]$,
- $(\forall q \in Q_{\mathrm{II}})(\forall a \in \Sigma)[\tau(q,a) \in Q_{\mathrm{I}}]$,
- $(\forall a \in \{0,1\})[\tau(q_\perp, a) = q_\perp]$.

Q_Θ is the set of nodes, *from* which player Θ has to move. This partition of Q into Q_{I}, Q_{II} and $\{q_\perp\}$, and the requirements on τ are only due to technical reasons.

A strategy $f : \Sigma^* \to \Sigma$ is a winning strategy for player I in G, if player I wins every play in which he follows f, i.e.,

$$(\forall b_1 \ldots b_n \in \Sigma^*)[\tau^*(q_0, f(\epsilon) b_1 f(b_1) b_2 f(b_1 b_2) \ldots b_n f(b_1 \ldots b_n)) \neq q_\perp].$$

Similar as in the case of FST's we only call that games CFG's in which player I has a winning strategy. Moreover, we require that each state $q \in Q - \{q_\perp\}$ is reachable from q_0, and, for simplicity, we assume $\Sigma = \{0,1\}$.

Similar as FST's closed finite-state games can be considered as a deterministic finite automata, which has only one rejecting state, namely $q_\perp$. Again, we write $\mathcal{L}(G)$ for the language accepted by the DFA G.

Closed finite-state games are a special kind of finite-state games. These more general games have been intensively studied in the literature, e.g., in [5, 9, 12, 15]. The famous Büchi-Landweber Theorem [5] states that in every finite-state game (particularly, in every closed finite-state game) one player has a winning strategy which can be executed by a finite automaton. [12] contains a very nice exposition of this result.

Due to the Büchi-Landweber Theorem we can restrict ourselves to strategies which are executable by finite automata. Note that this is a very natural and — in the context of efficient learning — a very desirable restriction, too.

Definition 15. A strategy automaton is a finite I/O-automaton $A = (\Sigma, Q, \tau, \gamma, q_0)$ over an alphabet Σ, where Q is the set of states, $\tau \colon Q \times \Sigma \to Q$ is the transition function, $\gamma : Q \to \Sigma$ is the output function and q_0 is the starting state. Player I plays according to strategy A iff he initializes $q = q_0$ and then executes an endless loop consisting of the following three steps: (1) Play the move $\gamma(q)$. (2) Read the next move b of player II. (3) Update $q := \tau(q, b)$. Thus, after the moves $\sigma = b_1 \ldots b_n$ of player II, player I answers with the move $A(\sigma) := \gamma(\tau^*(\sigma))$. A strategy automaton A is a *winning automaton* for the CFG G if player I wins every play of G in which he plays according to A.

We now define the notion of strategy queries, which build the analogue to branch queries for two-person games:

Definition 16. A *strategy query* for a CFG $G = (\Sigma, Q, Q_{\mathrm{I}}, Q_{\mathrm{II}}, \tau, q_0, q_\perp)$ is a strategy automaton A, which is answered by "yes", if A is a winning automaton for player I in G. Otherwise the strategy query is answered by a *counterexample*, which is a sequence $b_1 \dots b_n$ of player II moves such that A loses against $b_1 \dots b_n$ but not against $b_1 \dots b_{n-1}$, i.e., $\tau^*(q_0, A(\epsilon)b_1A(b_1)b_2A(b_1b_2) \dots b_{n-1}A(b_1 \dots b_{n-1})) \neq q_\perp$ and $\tau^*(q_0, A(\epsilon)b_1A(b_1)b_2A(b_1b_2) \dots b_nA(b_1 \dots b_n)) = q_\perp$.

Membership queries are defined similar as for FST's:

Definition 17. A *membership query* $u \in \Sigma^*$ for an CFG G is answered by "yes", if $u \in \mathcal{L}(G)$, and by "no" otherwise.

We are looking for algorithms which efficiently learn strategy automata for the class of CFG's from strategy queries, or from strategy and membership queries. The time complexity of such an algorithm is measured in

- the size n of the target CFG G, which is the number of states of the minimal CFG equivalent to T, i.e., $n = \mathit{size}(G) = \min\{|G'| : G' \text{ is a CFG and } \mathcal{L}(G) = \mathcal{L}(G')\}$, and
- the size m of the longest counterexample, received up to a specific point in the run of the algorithm.

A learning algorithm L runs in time $T(m, n)$ if the run time used by L is *at any point π in the run of L* bounded by $T(m, n)$, where m is the size of the longest counterexample received up to the point π. The membership and strategy query complexity of L are defined similar where run time is replaced by the number of membership or strategy queries, respectively.

Each FST yields canonically a CFG by introducing dummy moves for player II. Then, each strategy query A can canonically be transformed into a strategy query uv^ω for the FST by observing the play of A against 0^ω. Membership queries can be handled similar. Using this technique we can apply the lower bounds, which we obtained for FST's in Section 2.2, to the class of CFG's:

Corollary 18.
1. *There is no algorithm which* learns *strategies for the class of CFG's using only polynomially many membership and strategy queries.*
2. *There is no algorithm which* identifies *the class of CFG's using only polynomially many membership and strategy queries.*

Since there is no efficient strategy learning algorithm for the class of all CFG's, using membership and strategy queries, it would be nice to find natural subclasses for which such algorithms exist. One natural possibility is to generalize the concept of "deadend-freeness" to the two-person case, since the deadend free one-person games (namely the finite-state trees) have an efficient strategy learning algorithm:

Definition 19. A CFG is deadend free if for all states $q \in Q_{\mathrm{I}}$ there is an $a \in \Sigma$ such that $\tau(q, a) \neq q_\perp$.

Note that for every node $q \in Q - \{q_\perp\}$ in a deadend free CFG player I has a winning strategy in the game starting from q instead of q_0.

We shortly recall the setting of learning finite automata from queries [2, 4]. The learner gets information about a target DFA A by asking membership and/or equivalence queries. A *membership query* $x \in \Sigma^*$ is answered similar as above, i.e., the response is "yes", if $x \in \mathcal{L}(A)$, and "no" otherwise. In a (proper) *equivalence query* the learner submits a finite automata H as hypothesis, and receives "yes" if H is equivalent to A, i.e., $\mathcal{L}(A) = \mathcal{L}(H)$. If A and H are not equivalent, the answer to the equivalence query is a *counterexample* x in the symmetric difference of $\mathcal{L}(A)$ and $\mathcal{L}(H)$, i.e., $\mathcal{L}(A)\Delta\mathcal{L}(H)$. A learning algorithm *exactly identifies* DFA's, if on any target automata A the learner halts within finitely many steps and outputs a finite automata B which is equivalent to A. The time and query complexity of DFA learning algorithms are defined similar as for strategy learning algorithms.

Theorem 20. *The problems of learning strategies from strategy and membership queries and of identifying DFA's from membership and equivalence queries are reducible to each other:*

1. *If there is an algorithm which learns strategies for the class of deadend free CFG's in time $T(m,n)$ using $M(m,n)$ membership and $S(m,n)$ strategy queries, then there is an algorithm which identifies DFA's in time $O((S(m',n') + M(m',n') + 1) \cdot T(m',n'))$ using $M(m',n')$ membership and $S(m',n')$ equivalence queries, where $m' = m+1$ and $n' \leq 5n+6$.*
2. *If there is an algorithm which identifies DFA's in time $T(m,n)$ using $M(m,n)$ membership and $E(m,n)$ equivalence queries, then there is an algorithm which learns strategies for the class of deadend free CFG's in time $O((M(m,2n)+1) \cdot T(m,2n) + m \cdot E(m,2n))$ using at most $O(M(m,2n) \cdot T(m,2n) + m \cdot E(m,2n))$ membership and $O(E(m,2n))$ strategy queries.*

Note that Theorem 20.1 also includes the reduction of learning strategies for CFG's from strategy queries only, to identifying DFA's from equivalence queries only (namely, when $M(m,n) = 0$). Therefore, Angluin's negative result [4] for the non-learnability of DFA's from equivalence queries only can be carried over to strategy learning for deadend free CFG's. Similarly, her older positive result [2] can also be transferred using Theorem 20.2:

Corollary 21.
1. *There is no algorithm which learns strategies for the class of deadend free CFG's using only polynomially many strategy queries of polynomial size.*
2. *There is an efficient algorithm which learns strategies for the class of deadend free CFG's from membership and strategy queries.*

Theorem 20.2 transforms an algorithm for identifying DFA's from equivalence queries into a strategy learning algorithm for deadend free CFG's using strategy *and* membership queries. But a kind of halving algorithm yields a direct solution for the class of deadend free CFG's using only polynomially many strategy queries. However, this algorithm runs in double-exponential time and may use strategy queries whose size does not have any polynomial bound.

Acknowledgements: We would like to thank Wolfgang Ahrendt and John Case for helpful discussions and comments.

References

1. D. Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51(1):76–87, 1981.
2. D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, Nov. 1987.
3. D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.
4. D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5(2):121–150, June 1990.
5. J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
6. J. Case, M. Ott, A. Sharma, and F. Stephan. Learning to win process-control games watching game-masters. Submitted.
7. P. R. Halmos. *Measure Theory.* Van Nostrand, Princeton, New Jersey, 1950.
8. L. P. Kaelbling. *Learning in Embedded Systems.* The MIT Press: Cambridge, MA, 1993.
9. M. Kummer and M. Ott. Effective strategies for enumeration games. In H. K. Büning, editor, *Proceedings of Computer Science Logic CSL '95*, volume 1092 of *LNCS*, pages 368–387, Berlin, 1996. Springer.
10. 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.
11. H. Lescow. On polynomial-size programs winning finite-state games. In *CAV: International Conference on Computer Aided Verification*, pages 239–252, 1995.
12. R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.
13. 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.
14. C. D. Rosin and R. K. Belew. A competitive approach to game learning. In *Proceedings of the 9th Annual ACM Conference on Computational Learning Theory (COLT'96)*, pages 292–302. ACM Press, New York, NY, 1996.
15. W. Thomas. On the synthesis of strategies in infinite games. In *STACS 95*, volume 900 of *LNCS*, pages 1–13. Springer-Verlag, 1995.
16. R. Wiehagen. From inductive inference to algorithmic learning theory. *New Generation Computing*, 12, 1994.