数理解析研究所講究録
第754巻 1991年 25-34

On the role of equivalence queries in learning via queries

Seiichi Tani* (谷 聖一)

概要: 本論では, 質問を用いた概念の同定問題における同値質問の役割ついて考える. 同値質問は入力として仮説を受けとり, (1) 同定が成功したかどうかを教えること, および, (2) 同定が成功していない場合には反例を与えることの2つの役割を持っている. 同値質問と所属質問のどちらか一方のみからでは多項式時間で同定できないが, その両方の質問を用いると多項式時間で同定できる概念のクラスの存在が知られている. 本論では, そのようなクラスのなかに同値質問の (1) の役割を用いなくても多項式時間で同定できるものが存在することを示す. その証明は, 非冗長な k- 項単調 DNF 論理式 (ktMDNF) がこのような性質を持つことを示すことにより行なう.

次に, R を概念の表現のクラスとし, c_1, c_2 を R の元で記述される概念とする. このとき, c_1 の表現 $r \in R$ と c_2 に対する所属質問に答える神託が与えられたとき、c_1 と c_2 とが同じ概念かどうかを判定する問題を検定問題と呼ぶことにする. 本論では, 決定性有限オートマトン (dfa) の検定問題が決定不能である強い状況証拠が示される. 一方, ktMDNF および最小状態数が n の dfa に対する検定問題は多項式時間で解けることが示される.

1 Introduction

In this paper, we consider the problem of exactly identifying a representation of an unknown set (i.e. "*concept*") from a representation class using various types of queries. This problem is one of central issues in computational learning theory and there has been extensive research into the problem. Recently, polynomial time learning algorithms have been found for several representaion classes. For example, Angluin showed a learning algorithm that identifies deterministic finite automata (dfa) using membership and equivalence queries[2]. Furthermore, she showed a learning algorithm that identifies desjunctive normal form formulas with at most k-terms (k-term DNF formula) using membership and equivalence queries[1]. See also [5, 7, 10]. On the other hand, some negative results have been showed. For example, Angluin showed that there is no polynomial time algorithm that exactly identifies dfa using only equivalence queries [4]. Valiant introduced stochastic setting learning model, so called PAC (*Probably Approximately Correct*) learning model[8]. In this setting, Pitt and Valiant showed for all $k \geq 2$, k-term DNF formulas cannot be probably approximately identified in polynomial time from randam sampling using onlu equivalence queries unless RP = NP[1][6]. As a corollary of this result, in [3] Angluin showed that there is no polynomial time algorithm that exactly identifies k-term DNF formulas unless RP = NP.

*Department of Mathematics, Waseda University. E-mail address : tani@ds5000.math.waseda.ac.jp

[1]RP is the class of sets recognizable in randam polynomial time. RP is a subclass of NP, but it is unknown whether the containment is strict. Many reserchers suspect that RP and NP are unequal.

In this paper, we consider the power of equivalence queries in learning unknown concepts from what is called "minimally adequate teacher (MAT for short)". MAT is a model of a teacher introduced by Angluin[2], which can answer membership queries and equivalence queries. An equivalence query proposes a representation r, and MAT answers *yes* if the set c specified by r is equivalent to the unknown set u, while selects a string t in $u \oplus c$ otherwise. (For any set A and B, $A \oplus B$ denotes $(A \cup B) - (A \cap B)$.) In the latter case, the string t is called a *counterexample*. Equivalence queries have the following two roles: (1) indicating whether learning algorithms have succeeded and (2) giveing counterexamples. Several reserchers have delt with restricted equivalence queries, which is equivalence queries playing the former role only. But no results concerning the equivalence queries playing the latter role only have shown in the literature as far as the author knows. In this paper, we will show that there exists a representation class that can be identified by using MAT without requiring the role (1) of equivalence queries . We introduce concept of "learning from *successful couterexample queries* (SCEX)", and characterize a subcollection of representation classes with this property. We show that there is no polynomial time algorithm to identify nonredundant monotone DNF formulas with k-term (ktMDNF) using only equivalence queries unless NP = RP. We also show that there is a polynomial time algorithm to identify ktMDNF from MAT without the role (1) of equivalence queries.

In order to give some insights into the identifiable representation classess from MAT without role (1) of equivalence queries, we introduce the notion of "detectable"representation classes. Suppose R is a representaion language for some concepts. We consider the problem, given a representaion $r \in R$ and membership queries consistent with c which is a concept represented by a string in R, to decide whether c is equivalent to the concept described by r using the membership queries. If this problem is solvable, we call the class R is *detectable*. We show that ktMDNF is detectable in polynomial time. On the other hand, we show that if dfa is detectable then there exists a deterministic algorithm to identify dfa using only membership queries.

2 Preliminaries

2.1 Representation Classes

Here we specify "concepts ", objectives for learners, by using a "representation class"introduced by Warmuth [9].

A *representaion class* is a four-tuple $(R, \Gamma_1, \Phi, \Gamma_2)$, where

- Γ_1 and Γ_2 : finite sets, called an *alphabet for representations* and an *alphabet for concepts* respectively,
- R : a subset of Γ_1^*, called a *representaion language*, and
- Φ : a mapping from R to concepts over Γ_2^*.

For each represetation $r \in R$, the set $\Phi(r)$ is the *concept* represented by r. Since we use a fixed alphabet Σ $(= \{0, 1\})$ throughout this paper, we omit specifying alphabets and write, e.g., (R, Φ).

We assume a natural way to encode Turing machines and Boolean formulas. For any representation r of Turing machine, let $L(r)$ denote the language accepted by the Turing machine specified by r; $L(\cdot)$ is used similarly for Boolean formulas. In this paper the following semantic function is used common: a function Φ_{TM} that maps every representation r of Turing machine to $L(r)$ and a function Φ_{BF} such that for every representaion r of Boolean formula, $\Phi_{\mathrm{BF}}(r)$ consists of all assignments $\{0,1\}^n$ that satisfy the formula.

Below we define some representation languages.

- $R_{\mathrm{dfa}} = \{r : r \text{ is a description of dfa}\}$
- $R_{\mathrm{DNF}} = \{r : r \text{ encodes a Boolean formula in disjunctive normal form}\}$
- $R_{k\text{-tDNF}} = \{r : r \text{ is a DNF formula with at most } k\text{-terms}\}$
- $R_{\mathrm{MDNF}} = \{r : r \text{ encodes a monotone Boolean formula in disjunctive normal form}\}$
- $R_{k\mathrm{tMDNF}} = \{r : r \text{ is a nonredundunt monotone DNF formula with } k \text{ terms}\}$

2.2 Learnability

Let $R = (R, \Phi)$ be a representation class. For the exact learning problem there is an unknown concept $c \in \Phi(R)$, and information may be gathered about c by asking some types of queries. In this paper, we consider following two types of queries:

1. *Membership query.* The input is a string x and the output is *yes* if $x \in c$ and *no* if $x \notin c$.
2. *Equivalence query.* The input is a representation $r \in R$ and output is *yes* if $\Phi(r) = c$ and a *no* otherwise. In addition of the answer *no*, a *counterexample* is supplied, that is, a string $t \in \Phi(r) \oplus c$. The choice of counterexamples is assumed to be arbitary.

We shall also consider *restricted* version of equivalence queries, for which the responses are just *yes* and *no*, with no counterexample provided.

Let Q be a set of some types of queries. For example, Q is membership and equivalence queries. A deterministic algorithm A exactly learns C using Q if and only if for every $c \in \Phi(R)$, when A is given $0^{|r|}$ as input and run with an oracle for Q for c, A eventually halts and outputs some $r \in R$ such that $\Phi(r) = c$. Such an algorithm runs in polynomial time if and if only if there is a polynomial $p(l, m)$ such that for every $c \in \Phi(R)$, if l is the length of input then when A is run with an orcle for Q for c at any point in the run the time used by A is bounded by $p(l, m)$, where m is the maximum length of answer from the oracle for Q so far in the run, or $m = 0$ if no equivalence queries have been used.

R is (polynomial time) learnable using Q if and only if there exists a deterministic (polynomial time) algorithm to learn using Q.

3 Learning from membership and successful counterexample queries

In this section, we will introduce a concept of "successful counterexample queries (SCEX)".

3.1 Successful counterexample queries SCEX

Equivalence queries have two roles: equivalence queries (1) indicate whether learning algorithms have succeeded and (2) give an counterexample. Equivalence queries can play the former role only (*restricted equivalence queries*), but they cannot play the latter role only. Let $C = (R, \Phi)$ be a representation class. Consider an algorithm tries to identify an unknown concept $c \in \Phi(R)$. Suppose equivalence queries try to answer to give counterexamples only and no information about whether the algorithm has succeeded or not. If a equivalence query is taken a representation r such that $\Phi(r) = c$, then the query has no counterexample returned to the algorithm and indicates that the algorithm has identified since $\Phi(r) \oplus c = \emptyset$

Here we introduce a concept of "learning from membership queries and *successful counterxample queries* (SCEX)"for characterizing a subcollection of representation classes such that is learnable using MAT but don't require the role(1) of equivalence queries.

We consider SCEX for C. When SCEX takes a representation $r \in R$, SCEX outputs a counterexample if there exists a counterexample and a special string *fail* othrewise.

Definition. Let $C = (R, \Phi)$ be a representation. A deterministic algorithm A learns C using membership queries and SCEX if and only if for every $c \in \Phi(R)$, when A is given $0^{|r|}$ as input and run with an oracle for membership queries and SCEX for c, A eventually halts and outputs some $r \in R$ such that $\Phi(r) = c$ and A is not given "fail "by the oracle in the run time. Such an algorithm runs in polynomial time if and if only if there is a polynomial $p(l, m)$ such that for every $c \in \Phi(R)$, if l is the length of input then when A is run with an orcle for membership queries and SCEX for $\Phi(r)$ at any point in the run the time used by A is bounded by $p(l, m)$, where m is the maximum length of answer from the oracle for membership queries and SCEX so far in the run, or $m = 0$ if no equivalence queries have been used.

R is (polynomial time) learnable using membership queries and SCEX if and only if there exists a deterministic (polynomial time) algorithm to learn using membership queries and SCEX.

Suppose a learning algorithm tries to identify an unknoun concept $c \in \Phi(R)$. By this definition, the learning algorithms use counterexamples but when the algorithms constructs a hypothesis r r such that $\Phi(r) = c$ they must halt before asking an SCEX query.

3.2 Nonredundant k-term MDNF is learnable from SCEX

Let ktDNF $= (R_{k\text{-tDNF}}, \Phi_{\text{BF}})$, MDNF $= (R_{\text{MDNF}}, \Phi_{\text{BF}})$, and ktMDNF $= (R_{k\text{tMDNF}}, \Phi_{\text{BF}})$. In this subsection, we consider the problem to learn ktMDNF using membership and SCEX queries.

Pitt and Valiant showed for all $k \geq 2$, ktDNF cannot be probably approximately identified in polynomial time from randam sampling unless NP = RP[6]. We can certify Pitt and Valiant's result holds for ktMDNF. By relationships between types queries and Valiant's stochastic setting the followin proposition is proved. (See [3])

Proposition 3.1 *Let $k \geq 2$. ktMDNF is not polynomial time learnable using only equivalence queries unless* NP = RP.

On the other hand, ktMDNF is polymomial time learnble using MAT since Angluin showed there exists a polynomial time algorithm to learn MDNF using MAT[1].

The main result to be proved in this section is the following.

Theorem 3.2 *Let $k \geq 1$. ktMDNF is polynomial time learnable using membership queries and SCEX.*

Proof. This theorem is porved by showing a polynomial time algorithm to learn ktMDNF. The algorithm is a modification of Angluin's algorithm to learn MDNF using membership and equivalence queries in polynomial time.

A *prime implicant* of a Boolean formula ψ is a satisfiable product t of literals such t implies ψ, but no proper subterm of t implies ψ.

The algorithm keeps a current hypothesis ψ', initially the empty formula. The hypothesis ψ' always consists of a sum of prime implicants of ψ, and therefore implies ψ.

The algorithm takes an assignment a that satifies ψ but not ψ' as a counterexample by asking SCEX. Initially there is a counterexample since the algorithm keeps the empty formulas and $k \geq 1$. From a the algorithm searches for a new prime implicant of ψ. Let t be the product of all those variables x_i such that $a(x_i)$ is true. Clearly $a(t)$ is true. The following procedure is used to reduce t to prime implicant.

For each t' obtained by deleting one literal from t, determine whether t' implies ψ as follows. Let a' be the assignment that assigns true those variables in t' and false to the others. By making a membership query with a', the algorithm decides whether t' implies ψ or not. ($a'(\psi)$ is true if and only if t' implies ψ.)

If t' implies ψ, then t is replaced by t' and the reduction process is continued. Eventually the algorithm arrives at a term t such that t implies ψ, but no term obtained from t by deleting one literal implies ψ.

$a(t)$ is true and so t is not already in ψ'. The algorithm replace the hypothesis ψ' by $\psi' + t$. The algorithm iterates from getting a counterexample k-th times. After k-th times iterations, the algorithm has a hypothesis ψ' such that $\Phi_{\mathrm{BF}}(\psi') = \Phi_{\mathrm{BF}}(\psi)$.

At each iteration the number of terms of ψ' increments just one. Therefore there exists a counterexample until the algorithm iterates k-th times, so the algorithm cannot get "fail".

Each prime implicant added to the formula requires one SCEX and at most n membership queries, so the running time of the algorithm is clearly bounded by polynomial n. This concludes the proof of Theorem 3.2. □

Let n-DFA denote $(R_{n\text{-dfa}}, \Phi_{\mathrm{TM}})$, where
$R_{n\text{-dfa}} = \{r : R \in R_{\mathrm{dfa}}$ and the minimum dfa accepting $\Phi(d)$ has n states$\}$. Since $R_{n\text{-dfa}} \in R_{\mathrm{dfa}}$, n-DFA is learnable using MAT. The following proposition is also shown.

Proposition 3.3 *n-DFA is polynomial time learnable from membership queries and SCEX.*

4 Detectability

4.1 Detectability

Let $C = (R, \Phi)$ be a representation class. For any representation $r \in R$, and any concept $c \in \Phi(R)$, we say that c is *correct with respect to* r if and only if c is equivalent to $\Phi(r)$.

In this section we consider the problem, given a representaion $r \in R$ and an access to membership query for $c \in \Phi(R)$, to decide whether c is correct with respect to r. We will introduce the concept of the *detecter*. Consider a detecter that is designed for a representation class $C = (R, \Phi)$. The goal of the detecter D is that if D is given $r \in R$ and an access to membership query for $c \in \Phi(R)$ then D decides whether the concept c is correct with respect to r using membership queries for c.

Definition. Let $C = (R, \Phi)$ be a representaion class. A deterministic algorithm D *detects* C if and only if for every $r \in R$ and every $c \in \Phi(R)$, when, given r as input, D is run with an orcle for membership queries for c, eventually D halts and outputs special string "correct ", if c is correct with respect to r and "incorrect"otherwise. A deterministic algorithm D *polynomial time detects* C if and only if there exist a polynomial p such that for every $r \in R$ and every $c \in \Phi(r)$, given r as input D is run with an oracle for membership query for c at any point in the run the time used by D is bounded by $p(|r|)$ and detects C.

For any representation class C, we say C is *detectable* (*polynomial time detectable*) if there exists a deterministic algorithm D that detects (polynomial time detects) C.

4.2 Detectable representation classes

In this subsection we show some nontrivial polynomially detectable representation classes.

Theorem 4.1 *n-DFA is polynomilal time detectable.*

In this section we follow standard definitions and notations in computational complexity theory .

By a *string* we mean an element of Σ^*, and by a *language* we mean a subset of Σ^*. A symbol $\perp \notin \Sigma$ is used to denote "undefined ". The length of a string x is denoted by $|x|$; a function $\lambda x.|x|$ is denoted by $|\cdot|$.

For $x, y \in \Sigma^*$, $x \cdot y$ denotes the concatinaton of x and y. For $X, Y \subseteq \Sigma^*$, $X \cdot Y$ denotes $\{x \cdot y | x \in X, y \in Y\}$.

Proof of Theorem 4.1 is done by showing an algorithm that polynomially detects n-DFA DETECT_nDFA. DETECT_nDFA is shown in figure 1.

Let r be a representation in $R_{n\text{-dfa}}$ and c be a concept in $\Phi_{\text{TM}}(R_{n\text{-dfa}})$. Suppose DETECT_$n$DNF takes r as input and tries to decide whether r is correct with respect to c using an oracle for menmbership queries of c. DETECT_nDFA simulates Angluin's learning algorithm for $\Phi_{\text{TM}}(r)$. DETECT_nDFA checks whether the examples used by the learning algorithm are consistent with c. DETECT_nDFA halts and outputs 'incorrect'if an answer from membership query for $\Phi_{\text{TM}}(r)$ and an answer from membership query for $\Phi_{\text{TM}}(r')$ are different. If DETECT_nDFA doesn't output 'incorrect'and gets a representation r' such that $\Phi_{\text{TM}}(r) = \Phi_{\text{TM}}(r')$ by simulateing Angluin's algorithm, DETECT_nDFA halts and output 'correct'

The DETECT_nDNF use the following three subprocedures:

1. **MEM_BB** : the input is an instance a, and the output is *yes* if $a \in c$ and *no* otherwise.

$S := \{\lambda\}$; $E := \{\lambda\}$;
For each $t \in S \cup A$ **do**
    ans_1 :=MEM_MAT(t); ans_2 :=MEM_BB(t);
    If $ans_1 \neq ans_2$ **then** halt and output 'incorrect '
end
construct the initial observation table (S, E, T);
Repeat % Main Loop%
    While (S, E, T) is not closed or not consistent **do**
        If (S, E, T) is not closed **then do**
            find $s_1 \in S$ and $a \in \Sigma$ such that
                $f[s_1 \cdot a] \neq f[s]$ for all $s \in S$;
            add $s_1 \cdot a$ to S;
            For each $t \in (S \cup S \cdot \Sigma - (S - s_1 \cdot a) \cup (S - s_1 \cdot a) \cdot \Sigma) \cdot E$ **do**
                ans_1 :=MEM_MAT(t); ans_2 :=MEM_BB(t);
                If $ans_1 \neq ans_2$ **then** halt and output 'incorrect'
            end
            extend T to $(S \cup S \cdot \Sigma) \cdot E$
        end
        If (S, E, T) is not consistent **then do**
            find $s_1, s_2 \in S$, $a \in \Sigma$ and $e \in E$ such that
                $f[s_1] = f[s_2]$ and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$;
            add $a \cdot e$ to E;
            For each $t \in (S \cup S \cdot \Sigma) \cdot a \cdot e$ **do**
                ans_1 :=MEM_MAT(t); ans_2 :=MEM_BB(t);
                If $ans_1 \neq ans_2$ **then** halt and output 'incorrect '
            end
            extend T to $(S \cup S \cdot \Sigma) \cdot E$
        end
    end
    Make the conjecture $M_C := M(S, E, T)$;
    If the number of the states of M_c is less than n **then do**
        c :=CE_MAT(M_C); $P := \{c$ and all its prefixes $\}$;
        add all elements of P to S;
        For each $t \in (S \cup S \cdot \Sigma - (S - P) \cup (S - P) \cdot \Sigma) \cdot E$ **do**
            ans_1 :=MEM_MAT(t); ans_2 :=MEM_BB(t);
            If $ans_1 \neq ans_2$ **then** halt and output 'incorrect '
        end
        extend T to $(S \cup S \cdot \Sigma) \cdot E$
    end
Until the number of states of M_C is n;
halt and output 'correct'

Figure 1: The Algorithm for Detecting n-DFA, DETECT_nDFA

2. **MEM_MAT** : the input is an instance a, and the output is *yes* if $a \in \Phi_{\mathrm{TM}}(d)$ and *no* otherwise.

3. **CE_MAT** : the input is a representation $r' \in R_{n\text{-dfa}}$, and the output $\perp$ if $\Phi_{\mathrm{TM}}(r) \oplus \Phi_{\mathrm{TM}}(r') = \emptyset$ and a counterexample $t \in \Phi_{\mathrm{TM}}(r) \oplus \Phi_{\mathrm{TM}}(r')$ otherwise.

MEM_BB is realized by using the oracle for membership queres of c. MEM_MAT takes a as inputs and simulates a dfa represented by r. Given r' as input, CE_MAT constructs a dfa exactly accepting $\Phi_{\mathrm{TM}}(r) \oplus \Phi_{\mathrm{TM}}(r')$ and checks whether the accepting set of the dfa is empty or not. If the dfa accepts any string, CE_MAT outputs a couterexample in $\Phi_{\mathrm{TM}}(r) \oplus \Phi_{\mathrm{TM}}(r')$. These three procedures run in polynomial time.

Angluin introduced the concept the CCOT and proved the Lemma 4.2[2]. An *observation table* is a triple (S, E, T) consists of 1-3 as follows:

1. S is a nonempty finite prefix-closed set,

2. E is a nonempty finite suffix-closed set,

3. T is a finite function mapping $((S \cup S \cdot \Sigma) \cdot E)$ to $\{0, 1\}$.

(A set is *prefix-closed* if and only if every prefix of every member of the set is also a member of the set. Suffix-closed is defined analogously.)

An observation table can be visualised as 2-demensional matrix with rows labelled by elements of $S \cup S \cdot \Sigma$, columuns labelled by elements of E, and entry for row s and column e equal to $T(s \cdot e)$. For $s \in S \cup S \cdot \Sigma$, let $f[s]$ denote a function such that

$$f[s] : E \to \{0, 1\}, \quad f[s](e) = T(s \cdot e)$$

An observation table is *closed* if for each $t \in S \cdot \Sigma$ there exist an $s \in S$ such that $f[t] = f[s]$. An observation table is *consistent* if whenever $s_1, s_2 \in S$ are elements of S such that $f[s_1] = f[s_2]$, for all $a \in \Sigma$, $f[s_1 \cdot a] = f[s_2 \cdot a]$.

Let (S, E, T) is a closed consistent observation table (CCOT for short). The *corresponding dfa* $M(S, E, T)$ over Σ constructed from (S, E, T) is defined with the state set Q, the set of final states F, and the state transition function δ as follows:

$$\begin{aligned} Q &= \{f[s] \mid s \in S\}, \\ q_0 &= f[\lambda], \\ F &= \{f[s] \mid s \in S \text{ and } T(s) = 1\}, \\ \delta(f[s], a) &= f[s \cdot a]. \end{aligned}$$

Lemma 4.2 *Let (S, E, T) be a CCOT. The dfa $M(S, E, T)$ defined above satisfied the following three conditions.*

1. *M is well-defined.*

2. *M is consistent with the finite function T. That is, for every $x \in (S \cup S \cdot \Sigma) \cdot E$, $\delta(q_0, x) \in F$ if and only if $T(x) = 1$.*

3. *Suppose that M has n states. If M is any dfa consistent with T that has n or fewer states, then M' is isomorphic to M.*

The following lemma is proved by modifying a Lemma in [2].

Lemma 4.3 *In running of DETECT_nDFA, the number of states of the dfa $M(S,E,T)$ increases monotonic.*

Lemma 4.4 *For all $r \in R_{\text{dfa}}$ and all $c \in \Phi_{\text{TM}}(R_{n-\text{dfa}}$,* DETECT_$n$DFA *halts in time polynomial in n and answers whether c is correct with respect to r.*

Proof of Lemma 4.4. Consider DETECT_nDFA takes $r \in R_{n\text{-dfa}}$ and an access for membership queries of $c \in \Phi_{\text{TM}}(R_{n\text{-dfa}})$ and tries c is correct with respect r. Suppose DETECT_nDFA has an ovsevation table (S,E,T). Let $M(S,E,T)$ be the corresponing dfa to (S,E,T). Note r is always consistent with T.
1) In the case c is correct with respect to r: Since $c = \Phi_{\text{TM}}(r)$, $ans_1 \neq ans_2$ never happens. Therefore A can't output 'incorrect'.

By Lemma 4.3 and Lemma 4.2, after at most n-th iterations, the number of states of $M(S,E,T)$ is n and halts and output 'correct'. By the difinition of T, the ${\Phi_{\text{TM}}}^{-1}(c)$ and r are consistent with T. Therefore ${\Phi_{\text{TM}}}^{-1}(c)$, r, and $M(S,E,T)$ are descriptions of dfa consistent with T with n states. By Propositon 4.2, ${\Phi_{\text{TM}}}^{-1}(c)$, r, and $M(S,E,T)$ are isomorphic.
2) In the case c is correct with respect to r: $\Phi_{\text{TM}}(r) \oplus \neq \emptyset$. We call an element of $\Phi_{\text{TM}}(r) \oplus c$ is *witness*. When A constructs an observation table, for each $a \in (S \cup S \cdot A) \cdot E$, membership queries for $\Phi_{\text{TM}}(r)$ and c are done. If a witness a contained in the observation table, $ans_1 \neq ans_2$. Then A halts and outputs 'incorrect'. Assume any witness a is not contained in the observation table and then $M(S,E,T)$ with n states is constructed. By $|{\Phi_{\text{TM}}}^{-1}(c)| = n$ and Proposition 4.2, ${\Phi_{\text{TM}}}^{-1}(c)$ and $M(S,E,T)$ are isomorphic. This contradicts to the assumption. Therefore A must find witness a and halt and output"incorrect".

Angluin's learning algorithm for DFA, L^*[2] halts in time polynomial in n. DETECT_DFA halts also in time polymomial in n □

Proposition 4.5 *k*tMDNF *is polynomial time detecterbale.*

Proof of proposition 4.5 is omitted.

4.3 DFA is not detectable

Here we consider the problem to detect DFA. Angluin showed there exist a deterministic algorithm to learn DFA in polynomial time using MAT [2]. But we show followg theorem.

Theorem 4.6 DFA *is not detectable unless there exists a deterministic algorithm to learn* DFA *using membership queries.*

Proof. Assume DFA is detectable. Let $d_1, d_2, \ldots, d_i, \ldots$ be a natural enumeration of all elements of R_{dfa}. Suppose we try to identify an unknown concept $c \in \Phi_{\text{TM}}(R_{\text{dfa}})$. Let i be a nutural number such that $c = \Phi_{\text{TM}}(d_i)$. Since DFA is detectable by assumption, there exists a deterministic algorithm D to detect DFA. For every $d_j \in R_{\text{dfa}}$, given an access to membership queries for c, D can answer whether $\Phi(d_k) = c$. If D is given d_j in enumeration order D answers $c = \Phi_{\text{TM}}(d_i)$ when $i = j$. We can identify the unknown concept $\Phi(d_k)$ in finite time. This is inconsistent to that there exist no deterministic algorithms to learn DFA using membership queries. □

5 Concluding remarks

We characterize a subcollection of the representation classes that can be identified by using MAT without requiring the role (1) of equivalence queries . It is open whether there exists a characterization all representaion classes that can be identified by using MAT without requiring the role (1). It is unknown whether the collection of polynomial time detectable representaion classes is equal to the collection of representation classes that is polynomial time learnable using membership queries and SCEX.

Acknowledgement The author would like to express his sincere thanks to Professor Hiroshi Noguchi of Waseda University for his kind advice. He also thanks to Dr. Tetsuro Nishino of Tokyo Denki University, for his useful suggestions.

References

[1] Angluin, D. : *Learning k-term DNF formulas using queries and counterexamples.* Techinical Report, YALEU/DCS/RR-559, Department of Computer Science, Yale University (1987).

[2] Angluin, D. : Learning regular sets from queries and counterexamples, *Information and Computation, 75*(1987), pp.87-106.

[3] Angluin, D. : Queries and concept learning, *Machine Learning, 2* (1987), pp.319-342.

[4] Angluin, D. : Negative Results for Equivalence Queries, Machine Learning, 5(1990), pp.121-150.

[5] Nishino, T. : *Learninig Logic Formulas and Programs Based in Their Models.* Reserch Report, TDU-IS-19, Tokyo Denki University (1990).

[6] Pitt, L. and L. G. Valiant : Computational limitation on learning from examples, *Journal of the ACM, 35* (1988), pp.965-984.

[7] Sakakibara, Y. : Learning context-free grammars from structual data in polynomial time, *Proceedings of the 1988 Workshop on Computational Lerning Theory*, Morgan Kaufmann (1988).

[8] Valiant, L. G. : A theory of the learnable, *Communications of the ACM, 27* (1984), pp.1124-1142.

[9] Warmuth, M. : Towards representation independence in PAC learning, Lecture Notes in AI 397, Springer-Verlag (1989), 78-103.

[10] Watanabe, O. : A formal study of learnability via queries, *Automata, Languages and Programing, 1990. Proccedings.* Lecture Notes in Computer Science 443, Springer-Varlag (1990).