# Chapter 5

# Exact Learning via Teaching Assistants

## 5.1 Introduction

In this and the next chapter we focus on the issue of classifying the representation classes with respect to the complexity of exactly learning them. For this purpose, we propose a refinement of Angluin's model of learning, called the teaching assistant model of exact learning. Here we give a brief survey on the known results in this area and motivate the need for the new model.

In Angluin's model, a natural method for quantifying the complexity of a representation class is to consider the type and number of queries that a learner has to ask the teacher in order to learn any concept in the class. Hence, characterizing the power of various queries (mainly membership and equivalence) in Angluin's model has been of interest among researchers. Angluin [Ang90] showed that polynomially many equivalence queries are insufficient for learning representation classes which have a combinatorial property called approximate fingerprints. Applying this result she showed that deterministic finite automata (DFA), context free languages (CFL), conjunctive normal form formulas (CNF) and disjunctive normal form formulas (DNF) are not learnable with polynomially many equivalence queries. In [Gav93], Gavaldà proved that the nonexistence of approximate fingerprints is actually a sufficient condition for learnability with polynomially many equivalence queries. In particular, he showed that the representation class of CIRCUITS (boolean functions represented by logical circuits) are polynomial-time equivalence query learnable with the computational aid of an oracle in polynomial-time hierarchy. This shows that CIRCUITS are, in a sense, easier to learn than classes like DFAs, NFAs, CFGs, CNFs or DNFs. An improvement on the complexity of the oracle used for learning CIRCUITS is given in [BCG+96]. More recently, Hellerstein et al. [HPRW96] have shown that a representation class is learnable with polynomially many membership and equivalence queries, if and only if the representation class has *polynomial-size certificates*. A similar characterization has also been given in [Heg95].

Another interesting approach towards analyzing the complexity of exact learning via queries was proposed by Watanabe and Gavaldà [Wat90, WG94]. They used ideas from structural complexity theory for this purpose. In [WG94], a notion of *machine types* is defined in order to capture various types of queries in Angluin's model. The authors also show a close relation between the polynomial-time query learnability of representation classes and the complexity of some related *representation finding* problems.

The results mentioned above help us in classifying various classes according to the difficulty of exactly learning them. However, the classification could be finer. For example, how do we compare two representation classes that are both exactly learnable with polynomially many equivalence queries but not learnable with polynomially membership queries? It could be the case that one of these classes is easier than the other because the full power of equivalence queries may not be needed to learn one although it is required to exactly learn the other. Our main interest in this chapter is to explore this possibility.

In this and the next chapter we are mainly interested in the learnability of some algebraic classes in the teaching assistant model. To motivate this model, we consider the learnability of three representation classes; SYM of permutation groups, LS($p$) of linear spaces over finite fields and the class 3-CNF, in Angluin's model. In the next section, we give some essential notations and definitions from computational learning theory. We also give the precise definitions of the representation classes of interest to us. Then we formally define Angluin's model and prove upper and lower bounds on the learnability of the above-mentioned representation classes in this model. Finally we give the definition of the teaching assistant model in detail and prove upper bounds on learning these algebraic classes in this model. In the next chapter we further study the learnability of various subclasses of the above-mentioned representation classes and prove some absolute lower bounds on the learnability of these classes in this new model.

## 5.2 Learning Theory: Notations and Definitions

A *representation class* $\mathcal{P}$ is a tuple $\langle R, \mu, t\rangle$ where $t$ is a polynomial, $R \subseteq \Sigma^*$ and $\mu$ is a collection $\mu = \{\mu_n\}_{n\in\mathbf{N}}$ such that $\mu_n : R \to 2^{\Sigma^{t(n)}}$ is a many-one mapping. Any element in the range of $\mu$ is called a *concept* of $\mathcal{P}$. The set of all concepts of $\mathcal{P}$ is called the *concept class* represented by the representation class $\mathcal{P}$. When there is no confusion, we denote the concept class represented by $\mathcal{P}$ also by $\mathcal{P}$. For any $n$, let $\mathcal{P}_n$ denote the concepts of $\mathcal{P}$ in $\Sigma^{t(n)}$. For any concept $c \in \mathcal{P}_n$ let $size(c)$ denote $\min\{|r| : \mu_n(r) = c\}$. Let $\mathcal{P}_{n,m}$ denote the concepts of $\mathcal{P}$ in $\Sigma^{t(n)}$ which have representations of size less than or equal to $m$. For two representation classes $\mathcal{P} = \langle R, \mu, t\rangle$ and $\mathcal{P}' = \langle R', \mu', t'\rangle$, $\mathcal{P}'$ is said to be a *subclass* of $\mathcal{P}$, denoted by $\mathcal{P}' \subseteq \mathcal{P}$, if $R' \subseteq R$, $t' = t$, and $\mu' = \mu$ when restricted to $R'$. A representation class is *honest* if the following properties hold: given a string $y \in \Sigma^*$ and $n \in \mathbf{N}$, it can be checked in polynomial time whether $y$ is a valid representation of some concept in $\mathcal{P}_n$. Given input $\langle 0^n, r, x\rangle$, it can be decided in polynomial time whether $x$ is in $\mu_n(r)$. In this thesis, we will be interested only in honest representation classes.

Here, we give the precise definitions of three representation classes of interest to us. In Chapter 6, we also consider some subclasses of these representation classes.

**Representation classes SYM, LS($p$) and 3-CNF**

The *symmetric group* $S_n$ consists of all permutations on {1,...,n} with permutation composition as group operation. Subgroups of $S_n$ are called *permutation groups*. Now let us fix the encoding of permutations. Let the $n$-tuple $\sigma = (i_1, \ldots, i_n)$ represents the permutation $j \to i_j$ in $S_n$. Let $m = \lceil logn \rceil$ and the binary representation of $i_j$ be $x_{j1} \ldots x_{jm}$. Then we represent $\sigma$ with the binary string

$$x_{11}x_{11} \ldots x_{1m}x_{1m}01x_{21}x_{21} \ldots x_{2m}x_{2m}01 \ldots 01x_{n1}x_{n1} \ldots x_{nm}x_{nm}$$

of length $2nm + 2(n-1)$. It is clear that the group operations can be performed in time linear in $n$.

Let SYM $= \langle R, \{\mu_n\}_{n\geq 1}, t\rangle$ denote the representation class of permutation groups where $R = \cup_{n\geq 1} R_n$, and $r \in R_n$ encodes a set of permutations of $S_n$, namely, it is a concatenation of some strings, each encoding a permutation as explained above. For $r \in R_n$, $\mu_n(r)$ is the subgroup of $S_n$ generated by permutations encoded in $r$. Here, $t(n) = 2n\lceil\log n\rceil + 2(n-1)$. In [FHL80], it is shown that checking for membership of an element in a permutation group given by a generator set can be done in polynomial time. Hence it follows that the representation class SYM is honest.

Now we define the class LS($p$). For a fixed prime $p$, let $F_p$ denote the finite field of size $p$. Let the $n$-fold direct sum of $F_p$ be denoted by $F_p^n$. A vector $u \in F_p^n$ is represented as an $n$-tuple $(u[1], \ldots, u[n])$ for $u[i] \in F_p$. We can use the pairing function as in the case of permutation groups to encode this tuple whose length is $t(n) = 2n\lceil\log p\rceil + 2(n-1)$. LS($p$) contains subspaces of $F_p^n$ represented by a set of vectors which spans the subspace. More formally, LS($p$) $= \langle R, \{\mu_n\}_{n\geq 1}, t\rangle$. Here $R = \cup_{n\geq 1} R_n$, where $r \in R_n$ encodes a set of vectors of $F_p^n$. For $r \in R_n$, $\mu_n(r)$ is the vector space spanned by the set of vectors encoded in $r$. Since testing whether a given vector is in the span of a set of vectors can be done in polynomial time, it follows that LS($p$) is an honest representation class.

Finally, 3-CNF= $\langle R, \{\mu_n\}_{n\geq 1}, t\rangle$, where $R = \cup_{n\geq 1} R_n$, and $r \in R_n$ encodes a boolean function from $\{0,1\}^n$ to $\{0,1\}$ in conjunctive normal form where each clause has at most 3 literals. Here $t(n) = n$.

Now we are ready to give a detailed definition of Angluin's model of exact learning and consider the learnability of the above-defined representation classes in this model.

## 5.3 Angluin's Model of Exact Learning

Let $\mathcal{P} = \langle R, \mu, t\rangle$ be a representation class. Angluin's model of learning [Ang88] consists of a teacher and a deterministic learner. The learner $\alpha$ on input $\langle 0^n, 0^l\rangle$, has to output a representation $r \in R$ such that $\mu_n(r) = c$ if $size(c) \leq l$, for the *target* concept $c$ selected by the teacher after some finite number of computation steps. During the course of learning $\alpha$ can make two types of queries to the teacher, namely *equivalence queries* and *membership queries*. To the queries made by the learner, the teacher answers consistent with the target concept he/she has selected. To make an equivalence query, $\alpha$ writes down a string $h \in R$. The teacher responds to this, by answering 'Yes' if $\mu_n(h) = c$, or gives a *counter example* $x \in \mu_n(h)\triangle c$, if $\mu_n(h) \neq c$. These kinds of equivalence queries are called *proper* equivalence queries. If the query $y$ made by the learner is not in $R$, then $y$ is said to be an *improper* equivalence query. In this thesis, by an equivalence query we always mean a proper equivalence query.

To make a membership query, $\alpha$ writes down a string $x \in \Sigma^{t(n)}$. The teacher responds to this, by answering 'Yes' if $x \in c$ and 'No' if $x \notin c$. The learner's output is unspecified if $size(c) > l$. This kind of learning is called *bounded learning* as opposed to unbounded learning where the length parameter $l$ is not given to the learner.

Now we look at some standard complexity measures on resources used by the learner. Let $\mathcal{P}$ be a representation class. Then $\mathcal{P}$ is said to be polynomial-query learnable, if there exists a polynomial $q$ and a learner $\alpha$, such that for all $n$ and $l$, and for all $c \in \mathcal{P}_n$, the learner on input $\langle 0^n, 0^l \rangle$ with $c$ as target concept such that $size(c) \leq l$, outputs a representation of $c$ by making at most $q(n + l)$ queries to the teacher. $\mathcal{P}$ is said to be polynomial-time learnable, denoted by FP-learnable, if there is a polynomial $q$ and a learner $\alpha$, such that for all $n$ and $l$, and for all $c \in \mathcal{P}_n$, the learner on input $\langle 0^n, 0^l \rangle$ with $c$ as target concept such that $size(c) \leq l$, outputs a representation of $c$ within $q(n + l)$ time steps.

Notice that any upper bound on the time complexity is an upper bound on the query complexity. Hence if $\mathcal{P}$ is polynomial-time learnable then it is also polynomial-query learnable.

Before we go into the details of our results, we briefly explain the need for considering bounded learning as opposed to unbounded learning. In unbounded learning the learner is not given the length parameter $l$ as a part of the input. Hence for polynomial-time unbounded learnability the learner is allowed to run for a number of time steps polynomial in $n + size(c)$. Though bounded learnability is weaker than unbounded learnability, we restrict ourselves to bounded learnability because it is difficult to deal with learners whose running time is sensitive to the output, particularly when we consider learners which are nondeterministic oracle Turing machines. (We will be dealing with nondeterministic learners as a technical tool in the next chapter. The exact definition of these learners are given there.) For instance, consider in the unbounded learning setting, a nondeterministic learner $\alpha$ learning a representation class $\mathcal{P}$. Let $\alpha$ take $0^n$ as input and run in time $q(n + size(c))$ where $c \in \mathcal{P}_n$ is the target concept. We can ensure that nondeterministic computation paths of $\alpha$ which outputs a representation of $c$ are of length exactly $q(n + size(c))$. But on those computational paths where $\alpha$ is not going to output a string, cannot be timed to run exactly $q(n + size(c))$ steps, because the machine simply does not know when to stop on that path.

Bounded and unbounded learnability coincide for those representation classes $\mathcal{P}$

which have representations of size bounded by a polynomial in $n$ for all concepts in $\mathcal{P}_n$, (in a complexity-theoretic sense $\mathcal{P}$ can be viewed as a subclass of the nonuniform complexity class P/poly). We note that both the classes SYM and LS($p$) have this property. To see this for the class SYM, we recall Lagrange's theorem which states that if $G$ is a finite group and $H < G$ then $|G|$ is divisible by $|H|$. It can be easily deduced from Lagrange's theorem that every subgroup $G < S_n$ has a generator set of cardinality bounded by $n \log n$. Hence, each concept $c \in \mathrm{SYM}_n$ has a representation of size polynomial in $n$. It holds for LS($p$) also. As a consequence, bounded learnability coincides with unbounded learnability when we consider polynomial-time learnability of subclasses of SYM and LS($p$). So, in this case we can do away with the parameter $0^l$ from the learner's input which denotes the bound on the size of the target concept.

### 5.3.1 Complexity of Learning SYM, LS($p$) and 3-CNF in Angluin's Model

In this subsection we show both upper and lower bounds on the learnability of the classes SYM, LS($p$) and 3-CNF in Angluin's Model. While the upper bound is obtained by a simple application of Lagrange's theorem, for proving the lower bound we need to count the number of cyclic groups generated by $p$-cycles of $S_p$ for prime $p$.

An $m$-cycle is a permutation $\sigma \in S_n$ such that there are indices $i_1, \ldots, i_m$, $1 \leq i_j \leq n$: $\sigma(i_j) = i_{j+1 (\mathrm{mod}\ m)}$, and $\sigma(i) = i$ if $i \neq i_j$ for $1 \leq j \leq m$. It is easy to note that the order of any $m$-cycle is $m$. We have the following proposition on the number of cyclic groups generated by $p$-cycles. See the appendix for a proof.

**Proposition 5.3.1** *Let $p$ be a prime. Then there are $(p-2)!$ cyclic subgroups of $S_p$ generated by $p$-cycles.*

**Theorem 5.3.2** *The representation classes* SYM*,* LS$(p)$ *and* 3-CNF *are*

1. FP-*learnable using only equivalence queries in Angluin's model.*

2. *not* FP-*learnable using only membership queries in Angluin's model.*

*Proof.* (1). It is shown in [Ang88] that the class 3-CNF is FP-learnable using only equivalence queries. We give here a learner for the class SYM which makes at most $n \log n$ queries to output a generator set for a subgroup of $S_n$. A learner for LS$(p)$ which makes at most $n$ queries to output a basis for a subspace of $F_p^n$ can be designed very similarly. We omit the proof of this.

We give the description of the learner EQUIVLEARN for SYM. As mentioned earlier, we omit the length parameter $l$ from the input to the learner.

```
EQUIVLEARN(0^n)
S ← {e};
while  Answer to query S is 'No'
do g ← COUNTER EXAMPLE ;
    S ← S ∪ {g};
return S.
```

Consider the $i^{\text{th}}$ iteration of the **while** loop. Let $S = \{g_1, \ldots, g_{i-1}\}$ be the generator set constructed so far. From the fact that $G$ is a group and the identity $e \in G$, it follows that $\langle S \rangle < G$. So the counterexample $g_i$ given by the teacher at the $i^{\text{th}}$ iteration is in $G$ but not in $\langle S \rangle$ thereby growing the generator set by one. To show that the **while**-loop is executed at most $n \log n$ times, it is enough to show that at most $n \log n$ counter examples are sufficient. Form Lagrange's theorem it follows that if $G$ and $H$ be two groups such that $H < G$ and $g \in G$ but $g \notin H$, then $|\langle H, g \rangle| \geq 2|H|$. Hence addition of an element into the set $S$, grows the size of the group $\langle S \rangle$ by at least twice the previous size. The result follows.

(2). Now we show that none of these classes are FP-learnable using only membership queries. Note that to show this it is enough to show that these are not polynomial query learnable using only membership queries. Again, we prove it for the class SYM. The proof is by an adversary argument. Suppose there is a learner $\alpha$ which on input $0^n$ has to output a generator set for a subgroup $S_n$ by making only membership queries. Let $s$ be the polynomial bounding the number of queries asked by $\alpha$. Choose large enough prime $p$ such that $(p-2)! > s(p)+1$. The adversary keeps all the $(p-2)!$ $p$-cyclic groups with him. Let $\{q_1, q_2 \ldots, q_{s(p)}\}$ be the set of membership queries made by $\alpha$. For each membership query $q_i \neq e$ by $\alpha$, the adversary answers 'No', and the adversary answers 'Yes' if $q_i = e$. Now, from Proposition 5.3.1, and the fact that $(p-2)! > s(p)+1$, it follows that the adversary can consistently answer in this manner and still there will be at least 2 $p$-cyclic groups which are consistent with the answers to the queries. Hence the learner will be unable to decide which one of the cyclic groups to output.

A very similar adversary argument can be given for showing the lower bounds for both LS($p$) and 3-CNF (in the case of LS($p$) the adversary keeps all 1-dimensional liner spaces of $F_p^n$ and in the case of 3-CNF the adversary can keep all the conjunction of n literals). ∎

These upper and lower bound results indicate that the complexity of exactly learning these three representation classes in Angluin's model are very similar. Our next aim is to distinguish the learnability of these classes. The intuition is that the classes SYM and LS($p$) are algebraic and hence by exploiting the structure of these classes, we may be able to get better upper bounds on their learnability. For this purpose, in the next section we introduce a new exact learning model called the teaching assistant model of exact learning. Then in the last section of this chapter we prove new upper bounds on the learnability of these classes in this new model.

## 5.4 The Teaching Assistant Model of Exact Learning

Before we give formal definitions of the main ideas in the new model, we give some intuitive explanations. As mentioned earlier, apart from the learner and the teacher the new ingredient of this model is an intermediate agent between the learner and the teacher, called a teaching assistant. We define a teaching assistant as a set of strings whose membership in the set depends on the concept of interest. Then we define the notion of oracle Turing machines accepting these sets. These oracle Turing machines are allowed to make membership queries to the concept of interest (the concept fixed by the teacher). By varying the acceptance criteria of these Turing machines, we get different teaching assistant classes. We will be interested in acceptance criteria which are analogous to known complexity classes. A learner learning a representation class can ask membership queries to a pre-defined teaching assistant. Now the complexity of a representation class can be quantified by the complexity of the teaching assistant using which a learner (usually a deterministic oracle Turing machine) can learn the concept (output a representation for the concept). We move onto the formal definitions of the teaching assistant model.

**Definition 5.4.1** Let $\mathcal{P}$ be a representation class. Let $Q : \mathbf{N}\times\mathbf{N}\times\Sigma^*\times 2^{\Sigma^*} \rightarrow \{0,1\}$ be a predicate. A tuple $\langle n, l, x, c\rangle \in \mathbf{N}\times\mathbf{N}\times\Sigma^*\times 2^{\Sigma^*}$, is said to be $\mathcal{P}$-valid if $c \in \mathcal{P}_n$ and $size(c) \leq l$. The *teaching assistant* defined by $Q$ w. r. t. the representation class $\mathcal{P}$ is the set
$L(\mathcal{P}) = \{\langle n, l, x, c\rangle | \langle n, l, x, c\rangle$ is P-valid and $Q(n, l, x, c)\}$.

In the above definition notice that, consistent with the notion of bounded learnability, we have the parameter $l$ as part of $\mathcal{P}$-valid tuples.

Now we formalize the notion of learning via teaching assistants. Let $\mathcal{P} = \langle R, \mu, t\rangle$ be a representation class and $L(\mathcal{P})$ be a teaching assistant for $\mathcal{P}$. As in Angluin's model, learners are deterministic oracle Turing machine transducers. The learning algorithm comprises of a learner and teaching assistant pair. The teacher selects a target concept $c \in \mathcal{P}_n$. The learner $\alpha$, on input $\langle 0^n, 0^l \rangle$, has to output a representation $r \in R$ such that $\mu_n(r) = c$ and $size(c) \leq l$, after some finite steps of computation. During the course of the computation $\alpha$ can query its teaching assistant, say $L(\mathcal{P})$. Notice that when $\alpha$ wants to query the teaching assistant about $\langle n, l, x, c \rangle$, it is enough that $\alpha$ communicates $x$ to the teaching assistant, since the other parameters are implicit. For such a query $x$ by the learner, the learner receives the answer 'Yes' if $\langle n, l, x, c \rangle \in L(\mathcal{P})$ and 'No' if $\langle n, l, x, c \rangle \notin L(\mathcal{P})$. $\mathcal{P}$ is said to be learnable with assistant $L(\mathcal{P})$ if there is a learner $\alpha$ for $\mathcal{P}$ with queries to $L(\mathcal{P})$ as teaching assistant and we say $\alpha$ learns $\mathcal{P}$ using teaching assistant $L(\mathcal{P})$.

Let $\mathcal{P}$ be a representation class. Let $\alpha$ be a learner which learns $\mathcal{P}$ using a teaching assistant $L(\mathcal{P})$. For an input $\langle 0^n, 0^l \rangle$ and $c$ as the target concept, the time complexity of the learner is the number of steps it takes before it writes down a representation for $c$. $\mathcal{P}$ is said to be deterministic polynomial-time learnable (in short, FP-learnable) with a teaching assistant $L(\mathcal{P})$, if there is a polynomial $q$ and a deterministic learner $\alpha$ learning $\mathcal{P}$ using $L(\mathcal{P})$ such that for all $n$, and for all $c \in \mathcal{P}_n$, on input $\langle 0^n, 0^l \rangle$, the time complexity of $\alpha$ is bounded by $q(n + l)$.

In order to quantify the complexity of teaching assistants we define certain teaching assistant classes with respect to a given representation class. The teaching assistant classes we define here are the ones analogous $\mathrm{P}, \mathrm{NP}, \Sigma_2^p, \mathrm{SPP}$ and LWPP.

**Definition 5.4.2** Let $\mathcal{P}$ be a representation class and $L(\mathcal{P})$ be a teaching assistant.

1. $L(\mathcal{P})$ is said to be in the class $\Sigma_2^p(\mathcal{P})$ if there exists a polynomial-time $\Sigma_2$ oracle machine which for any $\mathcal{P}$-valid tuple $\langle n, l, x, c \rangle$, on input $\langle 0^n, 0^l, x \rangle$ uses $c$ as oracle and accepts $\langle 0^n, 0^l, x \rangle$ if and only if $\langle n, l, x, c \rangle \in L(\mathcal{P})$

2. $L(\mathcal{P})$ is said to be in the class $\mathrm{P}(\mathcal{P})$ if there exists a polynomial-time deterministic oracle Turing machine $M$ which for any $\mathcal{P}$-valid tuple $\langle n, l, x, c \rangle$, on

input $\langle 0^n, 0^l, x\rangle$ uses $c \in \mathcal{P}_n$ as oracle and accepts $\langle 0^n, 0^l, x\rangle$ if and only if $\langle n, l, x, c\rangle \in L(\mathcal{P})$.

3. $L(\mathcal{P})$ is said to be in the class NP($\mathcal{P}$) if there exists a polynomial-time nondeterministic oracle Turing machine $M$ which for any $\mathcal{P}$-valid tuple $\langle n, l, x, c\rangle$, on input $\langle 0^n, 0^l, x\rangle$ uses $c \in \mathcal{P}_n$ as oracle and accepts $\langle 0^n, 0^l, x\rangle$ if and only if $\langle n, l, x, c\rangle \in L(\mathcal{P})$.

4. $L(\mathcal{P})$ is said to be in the class SPP($\mathcal{P}$) if there exists a polynomial-time nondeterministic oracle Turing machine $M$ which for any $\mathcal{P}$-valid tuple $\langle n, l, x, c\rangle$, on input $\langle 0^n, 0^l, x\rangle$ uses $c$ as oracle and produces a *gap*=1 if $\langle n, l, x, c\rangle \in L(\mathcal{P})$ and *gap*=0 if $\langle n, l, x, c\rangle \notin L(\mathcal{P})$.

5. $L(\mathcal{P})$ is said to be in the class LWPP($\mathcal{P}$) if there exists a polynomial-time computable function $f$ and a polynomial time nondeterministic oracle Turing machine $M$ which for any $\mathcal{P}$-valid tuple $\langle n, l, x, c\rangle$, on input $\langle 0^n, 0^l, x\rangle$ uses $c$ as oracle and produces a *gap*=$f(n)$ if $\langle n, l, x, c\rangle \in L(\mathcal{P})$ and *gap*=0 if $\langle n, l, x, c\rangle \notin L(\mathcal{P})$.

In all the above definitions, the behavior of machines accepting the teaching assistant is not specified on inputs which are not $\mathcal{P}$-valid.

The following containments among teaching assistant classes follow directly from the above definitions.

**Proposition 5.4.3** *For any representation class* $\mathcal{P}$

- $\mathrm{P}(\mathcal{P}) \subseteq \mathrm{SPP}(\mathcal{P}) \subseteq \mathrm{LWPP}(\mathcal{P})$.
- $\mathrm{P}(\mathcal{P}) \subseteq \mathrm{NP}(\mathcal{P}) \subseteq \Sigma_2^p(\mathcal{P})$

Finally, we have the following definition of learning with an assistant from a particular teaching assistant class.

**Definition 5.4.4** Let $\mathcal{P}$ be a representation class and $\mathcal{C}$ be any assistant class defined with respect to $\mathcal{P}$. We say a representation class $\mathcal{P}$ is FP-learnable with a $\mathcal{C}$-assistant if there exists a teaching assistant $L(\mathcal{P}) \in \mathcal{C}(\mathcal{P})$ and a learner $\alpha$, such that $\mathcal{P}$ is FP-learnable with $L(\mathcal{P})$.

**Remark.** The notion of learning via teaching assistants is somewhat similar to the notion of self-producible circuits studied in [GW93]. (This is a generalization of self-p-producible circuits introduced by Ko [Ko85] and further studied by Balcázar and Book [BB86].) For comparing the two notions, we give some definitions. For any reltivizable function complexity class $\mathcal{F}$, a language $L \in$ P/poly is said to have $\mathcal{F}$-self-producible circuits, if an advice for $L$ can be computed in $\mathcal{F}(L)$. The motivation for this definition was to study the relative complexity of producing circuits for languages in P/poly. The main focus in [GW93] was on proving lower bounds. For example, it is shown that there exists a language $A \in$ P/poly which is not FP-self-producible. We are interested in producing representations (learning) for a class of concepts (as oppose to a language in [GW93]). We need learners which work for *all* the concepts in the class. Hence intuitively, it may be difficult (easier) to prove upper (respectively, lower) bound results in our framework compare to the one in [GW93]. For example, for proving the above-mentioned lower bound in [GW93], the authors have to diagonalize against all polynomial time reductions. On the other hand, an easy adversary argument is enough for showing the existence of a representation class that is not FP-learnable with membership queries.

For the sake of comparing the teaching assistant model with Angluin's model, in the next subsection we give a brief sketch of how to simulate equivalence and membership queries in the new model.

### 5.4.1 Comparison with Angluin's model

In this section we first prove a universal upper bound on the learnability of any representation class. Then we show the relationship between Angluin's model and the teaching assistant model for specific assistant classes.

**Theorem 5.4.5** *For any representation class $\mathcal{P}$, $\mathcal{P}$ is* FP*-learnable with a $\Sigma_2^p$-assistant.*

*Proof* Let $\mathcal{P} = \langle R, \mu, t\rangle$ where $\mu = \{\mu_n\}_{n\geq 1}$ be any representation class. Define a teaching assistant as follows: $L(\mathcal{P}) = \{\langle n, l, x, c\rangle$ is $\mathcal{P}$-valid$|\ \exists y$ such that $|xy| \leq l$ and $\mu_n(xy) = c\}$. We define a $\Sigma_2^p$ machine $M$ accepting $L(\mathcal{P})$. $M$ on input $\langle 0^n, 0^l, x\rangle$ existentially guesses $y$ of length bounded by $l - |x|$ and then universally verifies whether $\mu_n(xy) = c$ by universally guessing a string $z \in \Sigma^{t(n)}$ and verifying whether $z \in \mu_n(xy)$ if and only if $z \in c$. Since $\mathcal{P}$ is honest, checking membership in $\mu_n(xy)$ can be done in polynomial time, given $xy$. Membership of $z$ in $c$ can be done by querying the oracle $c$.

Now we give the description of the learner $\alpha$ for $\mathcal{P}$ which prefix searches for a representation of the target concept using teaching assistant $L(\mathcal{P})$. Let $c \in \mathcal{P}_n$ be the target concept and $l$ be such that $size(c) \leq l$.

$\Sigma_2^p$-LEARNER$(0^n, 0^l)$
$i \leftarrow l$;
**while** $\langle n, i-1, \lambda, c\rangle \in L(\mathcal{P})$
**do** $i \leftarrow i - 1$;
$x \leftarrow \lambda$;
**for** $j \leftarrow 1$ **to** $i$
**do if** $\langle n, i, x1, c\rangle \in L(\mathcal{P})$
  **then** $x \leftarrow x1$;
  **else** $x \leftarrow x0$;
**Output** $x$ .

We now compare Angluin's model with the teaching-assistant model. We show that membership queries are characterized by P-assistants. On the other hand, the combination of membership and equivalence queries can be simulated with NP-assistants. However, we show that NP-assistants are strictly more powerful than the combination of membership and equivalence queries. In fact we prove in this section that it is not possible to capture the power of proper equivalence queries in the teaching-assistant model.

**Theorem 5.4.6** *For any representation class $\mathcal{P}$, it is* FP*-learnable with membership queries if and only if it is* FP*-learnable with a* P*-assistant.*

*Proof* Let $\alpha$ be an FP-learner for $\mathcal{P} = \langle R, \mu, t\rangle$ where $\mu = \{\mu_n\}_{n\geq 1}$ which uses membership queries to the teacher. Define a teaching assistant $L(\mathcal{P})$ as follows. $L(\mathcal{P}) = \{\langle n, l, x, c\rangle | x \in c\}$. It is obvious that $L(\mathcal{P}) \in \mathrm{P}(\mathcal{P})$. Let $c \in \mathcal{P}_n$ be the target concept and let $l$ be such that $size(c) \leq l$. Consider a learner $\alpha'$ which behaves as follows. $\alpha'$ on input $\langle 0^n, 0^l\rangle$ simulates $\alpha$ on input$\langle 0^n, 0^l\rangle$. Whenever $\alpha$ makes a membership query $x$, $\alpha'$ makes a query $\langle n, l, x, c\rangle$ to $L(\mathcal{P})$ and treats the answer as the answer to the membership query for $\alpha$ and continues the simulation of $\alpha$. It is easily verified that $\alpha'$ outputs a representation of $c$ in time polynomial in $n + l$, if and only if $\alpha$ outputs a representation of $c$ in time polynomial in $n + l$.

To show the other containment, let $\beta$ be a FP-learner which learns $\mathcal{P}$ using teaching assistant $L(\mathcal{P}) \in \mathrm{P}(\mathcal{P})$. Let $M$ be the polynomial-time oracle Turing machine witnessing $L(\mathcal{P}) \in \mathrm{P}(\mathcal{P})$. Let $c \in \mathcal{P}_n$ be the target concept and let $l$ be such that $size(c) \leq l$. Consider a learner $\beta'$ which on input $\langle 0^n, 0^l\rangle$ and $c$ as target concept, simulates $\beta$. Whenever $\beta$ makes a query $\langle n, l, x, c\rangle$ to $L(\mathcal{P})$, $\beta'$ writes down $\langle 0^n, 0^l, x\rangle$ on its work tape and start simulating $M$ on input $\langle 0^n, 0^l, x\rangle$. During this simulation of $M$, whenever $M$ makes a membership query $y$ to $c$, $\beta'$ makes a membership query $y$ to the teacher and treats the answer to this as the answer to the query $y$ of $M$. Now, $\beta'$ outputs a representation of $c$ in time polynomial in $n+l$, if and only if $\beta$ outputs the same representation of $c$ in time polynomial in $n + l$.

**Theorem 5.4.7** *For any representation class $\mathcal{P}$, If $\mathcal{P}$ is* FP*-learnable with equivalence and membership queries, then $\mathcal{P}$ is* FP*-learnable with an* NP*-assistant.*

*Proof.* Let $\alpha$ be an FP-learner for $\mathcal{P} = \langle R, \mu, t\rangle$ where $\mu = \{\mu_n\}_{n\geq 1}$ which uses membership and equivalence queries to the teacher. Define a teaching assistant $L(\mathcal{P})$ as follows.

$L(\mathcal{P}) = \{\langle n, l, x, c\rangle \mid$ if $x = 1y$ implies $y \in c$, and if $x = 0y\#z$ implies there is a $w \in \Sigma^{p(n)-|y|}$ such that $yw \in c\triangle\mu_n(z)\}$.

It is easy to see that $L(\mathcal{P}) \in \mathrm{NP}(\mathcal{P})$, since $\mathcal{P}$ is honest. Let $c \in \mathcal{P}_n$ be the target concept and $l$ be such that $\mathit{size}(c) \leq l$. Consider a learner $\alpha'$ which behaves as follows. $\alpha'$ on input $\langle 0^n, 0^l\rangle$ simulates $\alpha$ on input $\langle 0^n, 0^l\rangle$. Whenever $\alpha$ makes a membership query $x$, $\alpha'$ makes a query $\langle n, l, 1x, c\rangle$ to $L(\mathcal{P})$ and takes the answer as answer to the membership query of $\alpha$ and proceeds. Whenever $\alpha$ makes an equivalence query $x$, $\alpha'$ first makes $\langle n, l, 0\#x, c\rangle$ to $L(\mathcal{P})$. If the answer to this query is 'No', then $\alpha'$ outputs $x$. Otherwise $\alpha'$ uses $L(\mathcal{P})$ to prefix search for a string $u \in c\triangle\mu_n(x)$ and uses $u$ as counterexample and proceeds. It is easy to see that $\alpha'$ outputs a representation of $c$ in time polynomial in $n + l$, if and only if $\alpha$ outputs a representation of $c$ in time polynomial in $n + l$. ∎

Let $\mathcal{P}$ and $\mathcal{P}'$ be concept classes such that $\mathcal{P}'$ is a subclass of $\mathcal{P}$. Although it intuitively appears reasonable that $\mathcal{P}'$ should be as easy to learn as $\mathcal{P}$, such is not the case in Angluin's model [Ang88]. While 3-CNF is FP-learnable with only equivalence queries, the subclass of 3-CNF consisting of concepts of cardinality exactly 1 (SINGLETONS) is not FP-learnable with membership and equivalence queries. It is known that this anomaly is caused by equivalence queries: more precisely, the anomaly arises from the fact that in Angluin's model equivalence queries must be representations of concepts that *belong* to the concept class. On the other hand, we observe that in the teaching-assistant model this anomaly does not arise by virtue of the fact that the teaching assistant makes only membership queries to the teacher. We state this as a proposition. The proof is by a very straightforward adversary argument.

**Proposition 5.4.8** [Ang88] *There exist honest representation classes $\mathcal{P}$ and $\mathcal{P}'$ such that $\mathcal{P}'$ is a subclass of $\mathcal{P}$, and $\mathcal{P}$ is* FP*-learnable with equivalence queries but $\mathcal{P}'$ is not* FP*-learnable with equivalence and membership queries in Angluin's model.*

**Remark.** We observe here that the teaching assistant model does not have the above anomaly. It follows from the definitions of a subclass and learning with teaching assistants that if $\mathcal{P}$ is FP-learnable with a $\mathcal{C}$-assistant ($\mathcal{C}$ is any of the above-defined teaching assistant classes) then $\mathcal{P}'$ is also FP-learnable with a $\mathcal{C}$-assistant. Hence, any upper bound on learning $\mathcal{P}$ is an upper bound on learning $\mathcal{P}'$ and any lower bound on learning $\mathcal{P}'$ is a lower bound on learning $\mathcal{P}$. We summarize this as a proposition.

**Proposition 5.4.9** *Let $\mathcal{P}$ and $\mathcal{P}'$ be representation classes such that $\mathcal{P}'$ is a subclass of $\mathcal{P}$. Let $\mathcal{C}$ be any of the above-defined teaching assistant classes. Then if $\mathcal{P}$ is* FP*-learnable with a $\mathcal{C}$-assistant, $\mathcal{P}'$ is also* FP*-learnable with a $\mathcal{C}$-assistant.*

From Propositions 5.4.8 and 5.4.9, we have the following interesting corollary.

**Corollary 5.4.10** *There is no teaching assistant class $\mathcal{C}$ such that the teaching assistant model captures equivalence query learnability in Angluin's model.*

In the next section we prove upper bounds on the learnability of the two algebraic concepts SYM and LS($p$) in the teaching assistant model.

## 5.5 Upper Bounds on Learning SYM and LS($p$)

In this section, we show some of the main results of this part of the thesis. First we show that the representation class SYM is FP-learnable with an LWPP-assistant.

Then we show that the class LS($p$) is FP-learnable with an SPP-assistant. These results indicate that LS($p$) may be easier to learn that the class SYM. Finally we show a lower bound on the learnability of the class 3-CNF. More precisely we show that 3-CNF is not FP-learnable with an LWPP-assistant (SPP-assistant) unless NP $\subseteq$ LWPP (respectively NP $\subseteq$ SPP).

**Remark.** We would like to observe here that as in the case of Angluin's model, without loosing any generality, the length-bound parameter $l$ in various definitions of the teaching assistant model also can be omitted when we are considering representation classes which have short representations (polynomially bounded) for all the concepts. Since all the algebraic classes we are interested in have this property, we will not mention this parameter for various learning algorithm that we design here, either as part of the input to a learner or as part of tuples defining a teaching assistant with respect to these classes.

**Learning SYM with LWPP-assistant**

We prove the following theorem.

**Theorem 5.5.1** *The representation class* SYM *is* FP-*learnable with an* LWPP-*assistant.*

Before we go into the proof, we give some definitions from the theory of permutation groups. Please recall the basic group-theoretic definitions given in Chapter 2.

Let $G < S_n$. For any $i; 0 \leq i \leq n$, the set $\{g \in G \mid \forall j \leq i : g(j) = j\}$ forms a subgroup of $G$. Denote this subgroup by $G^{(i)}$. This definition gives rise to the following chain of subgroups of $G$: $\{e\} = G^{(n)} < G^{(n-1)} < \cdots < G^{(0)} = G$. Now consider the left cosets of $G^{(i)}$ in $G^{(i-1)}$. It is easy to verify that two elements $g_1, g_2 \in G^{(i-1)}$ are in the same left coset of $G^{(i)}$ iff $g_1(i) = g_2(i)$. A set $T_i = \{g_{i1}, g_{i2}, \ldots, g_{id_i}\}$ of distinct (left) coset representatives of $G^{(i)}$ in $G^{(i-1)}$ is called a *transversal.* Notice that $G^{(i-1)} = \bigcup_{j=1}^{d_i} g_{ij}G^{(i)}$, where $d_i = |T_i|$. This means that any element $g \in G^{(i-1)}$ can be uniquely expressed as $g_{ij}h$ for some $h \in G^{(i)}$ and $1 \le j \le d_i$. It follows that the set $\bigcup_{i=1}^{n} T_i$ generates $G$. We call such a generator set a *strong generator set* for $G$.

For any set $X \subseteq \{1, 2, \ldots, n\}$, let $G_{[X]}$ denote the subgroup $\{g \in G \mid g(i) = i \ \ \forall i \in X\}$. For any permutation $\sigma \in S_n$ let $G_{[\sigma,X]}$ denote the set $\{g \in G \mid g(i) = \sigma(i) \ \ \forall i \in X\}$. Note that $G_{[\sigma,X]} = \sigma G_{[X]}$.

Now we state some useful properties of permutation groups in the following lemma. A sketch of the proof is given in the appendix.

**Lemma 5.5.2** *Let $G < S_n$.*

1. $|G^{(i)}| = \prod_{j>i} d_j$ *for each $i$, $1 \le i \le n$.*
2. *The number of strong generator sets for $G^{(i)}$ is $\prod_{j=i+1}^{n} |G^{(j)}|^{d_j}$ for each $i$, $1 \le i \le n$.*
3. *Let $X \subseteq \{1, 2, \ldots, n\}$ and $\sigma \in S_n$. Then $|G_{[\sigma,X]}| = |G_{[X]}|$*

Now we formally give the proof of the Theorem 5.5.1

*Proof.* (*of Theorem 5.5.1*) Let $G$ be the target group. Define the assistant $L(\mathrm{SYM})$ as:

$L(\mathrm{SYM}) = \{\langle 0^n, k, i, j_1, \ldots, j_k, G\rangle | \exists g \in G$ such that $g$ fixes 1 to $i$; $g(i+l) = j_l$ for $1 \le l \le k\}$. We first show that this set defines an LWPP-assistant by constructing a nondeterministic machine which produces zero *gap* if the input string is not in the set and produces a $gap = (n!)^{2n^2+1}$ if it is in the set. Then we give a FP-learner which uses $L(\mathrm{SYM})$ as assistant for prefix searching for a strong generator set of any target group.

**Claim 5.5.2.1** *There is a nondeterministic oracle machine $M$ such that given a group $G < S_n$ as oracle it has the following behavior:*

- *If* $\langle 0^n, k, i, j_1, \ldots, j_k, G\rangle \in L(\mathrm{SYM})$ *then* $M$ *on input* $\langle 0^n, k, i, j_1, \ldots, j_k\rangle$ *produces a gap* $= (n!)^{2n^2+1}$.
- *If* $\langle 0^n, k, i, j_1, \ldots, j_k, G\rangle \notin L(\mathrm{SYM})$ *then* $M$ *on input* $\langle 0^n, k, i, j_1, \ldots, j_k\rangle$ *produces zero gap.*

*Proof* For ease of exposition, we will give a structured description of machine $M$. Intuitively speaking, $M$ simulates another machine $N$ on all computation paths on which a "correct" guess is made. We need to introduce the new machine $N$ (invoked as subroutine by $M$) in order to ensure that the gap produced by $M$ is the easily computable quantity stated in the claim. We will also further structure the description of $N$ by designing another machine $N'$ which is a subroutine to $N$. All these machines access $G$ as oracle. The main work involved is in the design of $N$ which has the behavior that on input $\langle 0^n, i\rangle$ it produces a *gap* $= (n!)^{2n^2+1}/|G^{(i)}|$. The following is the description of machine $M$.

DESCRIPTION OF $M(0^n, k, i, j_1, \ldots, j_k)$

1 Guess $g \in S_n$;
2 **if** $g \in \mu_n(r)$ and $g(j) = j;\ \forall j; 1 \le j \le i$ and $g(i+l) = j_l;\ \forall l; 1 \le l \le k$
3    **then** Simulate Machine $N$ on input $\langle 0^n, i+k\rangle$ ;
4    **else** Branch into two paths ;
5         **accept** on one path and **reject** on the other path ;
6 **end-if**

Now we argue that $M$ has the claimed properties (assuming the behavior of $N$ which we later establish). Suppose $\langle 0^n, k, i, j_1, \ldots, j_k, G\rangle \notin L(\mathrm{SYM})$. Then there exists no $g \in G^{(i)}$ such that $g(i+l) = j_l$ for $1 \le l \le k$. In this case the **if** test in *line* 2 of $M$ fails for all guesses and $M$ produces zero *gap*. Now, suppose $\langle 0^n, k, i, j_1, \ldots, j_k, G\rangle \in L(\mathrm{SYM})$. It follows from part 3 of Lemma 5.5.2 that the number of guessed permutations $g$ that fulfill the **if** test is $|G^{(i+k)}|$. Now, suppose $N$ on input $\langle 0^n, i+k\rangle$ produces *gap* $= (n!)^{2n^2+1}/|G^{(i+k)}|$. Since $N$ is being simulated on $|G^{(i+k)}|$ paths, the total gap produced by $M$ is $((n!)^{2n^2+1}/|G^{(i+k)}|) \times |G^{(i+k)}| = (n!)^{2n^2+1}$.

Now the task is to design $N$ so that it has the desired behavior. Before we describe $N$ we describe its subroutine $N'$.

Description of $N'(0^n, i, j)$

Guess $g \in S_n$;<br>
**if** $g \in G^{(i-1)}$ such that $g(i) = j$<br>
  **then if** $g \in G$<br>
      **then accept.**<br>
  **else reject.**<br>
**end-if**

Description of $N(0^n, i)$

**Guess** an encoding $X_i = \langle T_{i+1}, \dots, T_n \rangle$ of a strong generator set for $G^{(i)}$<br>
**if** $X_i$ does not correctly encodes a strong generator set produce a *gap* $= 0$<br>
**for** $g \in X_i$; **if** $g \notin G^{(i)}$ produce a *gap* $= 0$<br>
 Using $X_i$ compute $\alpha = \prod_{n \geq k > i} |T_k|$<br>
 Using $X_i$ compute $\beta = \prod_{n \geq j > i} (\prod_{n \geq k > j} |T_k|)^{|T_j|}$<br>
 Branch into $(n!)^{n^2+1}/\alpha\beta$ and continue<br>
 Using $X_i$ construct the set $S_i$<br>
 Compute $\gamma = \prod_{n \geq j > i} (\prod_{n \geq k > j} |T_k|)^{n-j-|T_j|+1}$<br>
 Branch into $(n!)^{n^2}/\gamma$ paths<br>
 Produce a gap of $\prod_{\langle j,k \rangle \in S_i} ((\prod_{n \geq l > j} |T_l|) - acc_{N'} \langle 0^n, j, k \rangle)$

Observe that $N'$ has the following behavior: If there is $g \in G^{(i-1)}$ such that $g(i) = j$ then there are $|G^{(i)}|$ accepting paths, otherwise $N$ has no accepting paths.

The machine $N$ produces the required *gap* as follows. On input $\langle 0^n, i \rangle$, $N$ tries to compute $|G^{(i)}|$ nondeterministically with $G$ as oracle. For that it first guesses an encoding of a strong generator set $X_i = \langle T_{i+1}, \dots, T_n \rangle$ of $G^{(i)}$ and verifies that it actually encodes a strong generator set of some subgroup of $S_n$. By making queries to $G$ it verifies that the elements encoded in $X_i$ are all in $G^{(i)}$. The bad case that still remains to be handled is that the guessed set $X_i$ does not generate all of $G^{(i)}$ but only a *proper* subgroup of it. On computation paths where a generator set for a proper subgroup of $G^{(i)}$ is guessed, it must produce zero *gap*. We now give details.

Let $X_i = \langle T_{i+1}, \ldots, T_n \rangle$ where $T_j$, for $(i+1) \leq j \leq n$, encodes in lexicographic order a transversal of $G^{(i)}$ in $G^{(i-1)}$. (We use $T_i$ to denote both a transversal of $G^{(i)}$ in $G^{(i-1)}$ and its lexicographic encoding.) Let $S_i = \{\langle j, k\rangle \mid (i+1) \leq j \leq k \leq n$ and $\nexists\ g \in X_i \cap G^{(j-1)}$ such that $g(j) = k\}$. (By '$g \in X_i$' we mean $g$ is encoded in $X_i$.) Let $K_i$ denotes the number of strong generator sets for $G^{(i)}$.

Let us analyze the behavior of $N$. It is clear from the description of $N$ that after *line* 3 only those computation paths matter which correctly encode a subgroup of $G^{(i)}$. All other paths contribute zero *gap* to the overall *gap* produced by $N$. The remaining steps of $N$ ensure that those paths on which the guessed $X_i$ generates a proper subgroup of $G^{(i)}$ contribute zero *gap*.

First, let $\rho$ be a computation path on which the guessed $X_i$ encodes a strong generator set for $G^{(i)}$. We now show that $N$ produces $gap = (n!)^{2n^2+1}/|G^{(i)}|$. By part 1 of Lemma 5.5.2, $\alpha$ computed in *line* 4 is $|G^{(i)}|$. Also, from part 2 of Lemma 5.5.2, $\beta$ computed in *line* 5 is $K_i$, which is the number of strong generator sets for $G^{(i)}$. So, in *line* 6 $\rho$ contributes $gap = (n!)^{n^2+1}/(|G^{(i)}|K_i)$. (Notice that $(n!)^{n^2+1}$ is chosen to be divisible by the denominator.) In *line* 9, each such path again branches into $(n!)^{n^2}/\gamma$ paths. So at this point the path $\rho$ has contributed $(n!)^{2n^2+1}/(|G^{(i)}|K_i\gamma)$ paths. In *line* 10, since $X_i$ generates $G^{(i)}$ it follows that $acc_{N'}(\langle 0^n, j, k\rangle) = 0$, for each $\langle j, k\rangle \in S_i$. Hence the *gap* produced in *line* 10 is $\prod_{\langle j,k\rangle \in S_i}(\prod_{n\geq l>j}|T_l|)$ which, by rearranging terms, is easily seen to be $\prod_{n\geq j>i}(\prod_{n\geq k>j}|T_k|)^{(n-j-|T_j|+1)} = \gamma$. Therefore, at end of *line* 10 $\rho$ contributes a $gap = \gamma((n!)^{2n^2+1}/|G^{(i)}|K_i\gamma)$. Since there are $K_i$ different paths $\rho$, one for each strong generator set for $G^{(i)}$, the total contribution to the *gap* from the $K_i$ generator sets adds up to $(n!)^{2n^2+1}/|G^{(i)}|$.

Now, let $\rho$ be a computation path on which the guessed set $X_i$ generates a proper subgroup of $G^{(i)}$. We show that this path contributes a $gap = 0$. Notice that in order to prove that $\rho$ contributes $gap = 0$, it suffices to show that the *gap* produced in *line* 8 is zero. Observe that corresponding to $X_i$ there is a least $j < i$, say $j_0$, such that $T_{j_0}$ is not a transversal of $G^{(j_0)}$ in $G^{(j_0-1)}$. Therefore, there exists $k \leq n$ and $g \in G$ such that $\langle j_0, k\rangle \in S_i$, $g \in G^{(j_0-1)}$, and $g(j_0) = k$. On the other hand, since $j_0$ is the largest such index $\bigcup_{n\geq l>j_0} T_l$ is a strong generating set for $G^{(j_0)}$. Putting the preceding two statements together, we can see that machine $N'$ on input $\langle 0^n, j_0, k\rangle$ has $|G^{(j_0)}|$ accepting paths. Furthermore, since $\prod_{n\geq l>j_0} |T_l| = |G^{(j_0)}| = acc_{N'}(\langle 0^n, j_0, k\rangle)$, it holds that the product $\prod_{\langle j,k\rangle\in S_i}(acc_{N'}(\langle 0^n, j, k\rangle) - (\prod_{n\geq l>j} |T_l|)) = 0$. Hence the gap produced on path $\rho$ is zero. Finally, notice that the *gap* prescribed in *line* 10 is indeed producible from the closure properties of GapP. This proves Claim 5.5.2.1.

**Claim 5.5.2.2** *There is a polynomial time deterministic learning algorithm* LearnSYM *with* $L$(SYM) *as assistant which, given a target concept* $G$ *from the class* SYM *outputs a strong generator set for* $G$.

*Proof* The objective of the learner LearnSYM is to compute a strong generator set as a union $\bigcup_{i=1}^{n} T_i$, where $T_i$ is a transversal of $G^{(i)}$ in $G^{(i-1)}$, for $1 \leq i \leq n$. To this end, the learner first computes, with the help of $L$(SYM), the set of pairs $\{\langle i,j\rangle \mid \exists \ g \in G$ such that $g \in G^{(i-1)}$ and $g(i) = j\}$. For each such pair $\langle i,j\rangle$ it invokes a subroutine Construct on input $\langle 0^n, i, j\rangle$ in order to compute the lexicographically first $g \in G^{(i-1)}$ such that $g(i) = j$. This is done by a prefix search with the help of $L$(SYM). It is easy to see that the set of all these computed elements $g$ is a strong generator set. We give a formal description of this simple learning algorithm below.

LearnSYM($0^n$)
**for** pairs $\langle i,j\rangle; 1 \le i \le j \le n$ **do**
  **if** $\langle 0^n, 1, i-1, j, G\rangle \in L(\mathrm{SYM})$
    **then** Construct($0^n, i, j$);
  **end-if**
**end-for**

Construct($0^n, i, j$)
$k \leftarrow 1$
**repeat**
    Find first $l_k; i < l_k \le n$ such that
    $\langle 0^n, k+1, i-1, j, x_1, \ldots, x_{k-1}, l_k, G\rangle \in$L(SYM);
    $x_k \leftarrow l_k$;
    $k \leftarrow k+1$;
  **until** $k = n - i$
**return** $(1, \ldots, i-1, j, x_1, \ldots, x_{n-i})$.

It is easy to see that LearnSYM has time complexity $O(n^4)$. ∎

**Learning LS($p$) with SPP-assistant**

Now we show the following upper bound on learning the class LS($p$).

**Theorem 5.5.3** *For any prime $p$, the representation class* LS($p$) *is* FP-*learnable with an* SPP-*assistant.*

For the proof we need to give some notations and definitions from linear algebra. We give only definitions that are essential for our proof. For more basic definitions and results please refer to any standard linear algebra book (see for example [HK71]).

Let $U$ be a vector space over a field $F$. We denote the zero vector of $U$ by $\mathbf{0}$. $U$ is called nontrivial if $U \neq \{\mathbf{0}\}$. Let $V < U$ denote that $V$ is a subspace of $U$. For $X \subseteq U$ let $\langle X\rangle$ denote the subspace of $U$ spanned by $X$. The direct sum of $U$ and $V$ is denoted by $U \oplus V$, and the $n$-fold direct sum of $F$ is denoted by $F^n$. For the proof of the Theorem 5.5.3, we need to recall some results form linear algebra. The following result easily follow from basic definitions.

**Proposition 5.5.4** *Let $W < V < U$ be finite dimensional vector spaces over a field $F$. Let $A$ be a basis for $U$ and $B$ a basis for $W$. Let $A'$ be a maximal subset of $A$ such that $A' \cup B$ is linearly independent. Then $A' \cup B$ is a basis for $U$. Moreover, $V = W \oplus (\langle A' \rangle \cap V)$.*

Let $F$ be a field, and $U$ be a nontrivial subspace of $F^n$. Let $U^{(i)}$ denote the subspace $\{u \in U \mid \text{for } 1 \le j \le i: \ u[j] = 0\}$ of $U$, for $1 \le i \le n$, and let $U^{(0)}$ denote $U$. The *0-index* of $U$ is the maximum $k$ such that $U^{(k)}$ is nontrivial. We have the following proposition. See appendix for a proof.

**Proposition 5.5.5** *Let $F$ be a field and $U$ be a nontrivial subspace of $F^n$. If $k$ is the 0-index of $U$ then $U^{(k)}$ is a 1-dimensional subspace.*

*Proof.* (*Of Theorem 5.5.3*) We first define an assistant $L(\mathrm{LS}(p))$ and show that $\mathrm{LS}(p)$ is FP-learnable with assistant $L(\mathrm{LS}(p))$. Then we prove that $L(\mathrm{LS}(p)) \in \mathrm{SPP}(\mathrm{LS}(p))$.

On input $0^n$, let $V < F_p^n$ be the target subspace. Let vectors $F_p^n$ be uniformly encoded in strings of length $t(n)$ for polynomial $t$ (recall the definition of $\mathrm{LS}(p)$). Define $L(\mathrm{LS}(p))$ as: $L(\mathrm{LS}(p)) \triangleq \{\langle 0^n, A, i, y, V\rangle | i \text{ is the 0-index of } (\langle A \rangle \cap V); \exists z \in \Sigma^{t(n)} \text{ with } yz \in \langle A \rangle \cap V; yz \neq \mathbf{0}\}$.

We first give an intuitive idea of how the learner works. The aim of the learner is to construct a basis for the target subspace $V$. Suppose we have constructed an independent set of vectors $B \subseteq V$. We use $L(\mathrm{LS}(p))$ to prefix search for a vector $v \in V - \langle B \rangle$ as follows: Let $A$ be a fixed basis for $F_p^n$ (for example, the standard basis of unit vectors). Construct $A' \subseteq A$ such that $A' \cup B$ is a basis for $F_p^n$. (This can be done with some rank computations, which can be carried out in polynomial time.) Next, compute the 0-index of the space $(\langle A' \rangle \cap V)$ with $L(\mathrm{LS}(p))$ as oracle, by making queries of the form $\langle 0^n, A', i, \epsilon, V\rangle$ for $1 \le i \le n$. We now construct $v$ using prefix search with queries to $L(\mathrm{LS}(p))$. Notice that if $\langle B \rangle \neq V$

then by Proposition 5.5.4 $\langle A' \rangle \cap V$ is nontrivial and every nonzero element in it is independent of vectors in $\langle B \rangle$. Therefore, including $v$ in $B$ increases the dimension of $\langle B \rangle$. This process of including new vectors in $\langle B \rangle$ will therefore terminate within $n$ steps. We now formally describe the learning algorithm.

LEARNLINEAR($0^n$)
$B \leftarrow \{\mathbf{0}\}$;
$A \leftarrow \{e_1, \ldots, e_n\}$ (* Standard basis for $F_p^n$ *) ;
Construct $A' \subseteq A$ such that $B \cup A'$ is maximally independent ;
$i \leftarrow 1$;
**while** $\langle 0^n, A', i, \lambda, V \rangle \notin L(\mathrm{LS}(p))$
**do** $i \leftarrow i + 1$;
**end-while**
**if** $i = n + 1$
  **then** **output** B and stop .
**end-if**
$y \leftarrow \lambda$;
**for** $j \leftarrow 1$ **to** $t(n)$
**do if** $\langle 0^n, A', i, y1, r \rangle \in L(\mathrm{LS}(p))$
    **then** $y \leftarrow y1$;
    **else** $y \leftarrow y0$;
  **end-if**
**end-for**
$B \leftarrow B \cup \{y\}$;
**goto** 3;

We now show that $L(\mathrm{LS}(p)) \in \mathrm{SPP}(\mathrm{LS}(p))$. We construct a GapP machine $M$ which on input $\langle 0^n, A', i, y \rangle$, uses $V$ as oracle, and has the property that $M(0^n, A', i, y)$ produces $gap = 1$ if $\langle 0^n, A', i, y, V \rangle \in L(\mathrm{LS}(p))$ and produces $gap = 0$ otherwise.

To this end we define another NP machine $N$ (which uses $V$ as oracle). On input $\langle 0^n, A', i, y, k \rangle$ the machine $N$ guesses $p - 1$ distinct nonzero vectors $v_1, \ldots, v_{p-1} \in \langle A' \rangle^{(k)}$ in lexicographically increasing order, such that $y$ is a prefix of $v_1$ and $N$ accepts along this path iff each $v_i \in V$. It follows from the definition of 0-index that if $k$ is more than the 0-index of $A' \cap V$ then $N$ has no accepting paths. Furthermore, if $k$ is the 0-index then, from Proposition 5.5.5, $N$ has a unique accepting path.

Let $acc_N(\langle 0^n, A', i, y, k\rangle)$ be the number of accepting paths of $N$ on input $\langle 0^n, A', i, y, k\rangle$. We now design the desired machine $M$ which uses $N$.

DESCRIPTION OF $M(0^n, A, i, y)$

1 Produce a *gap* of $gap_N(\langle 0^n, A', i, y, i\rangle) \cdot \prod_{k=i+1}^{n}(1 - acc_N(\langle 0^n, A', i, y, k\rangle)$.

Notice that on input $\langle 0^n, A', i, y\rangle$, if $i$ is not the 0-index then $M$ has a $gap = 0$. Furthermore, if $i$ is the 0-index and $y$ is the prefix of the guessed vector $v_1$, then $M$ has $gap = 1$ otherwise $M$ has $gap = 0$. Hence $L(\mathrm{LS}(p))$ is in $\mathrm{SPP}(\mathrm{LS}(p))$. $M$ can produce the *gap* prescribed in *line* 1 above by appropriate simulations of machine $N$. ∎

**Hardness of learning 3-CNF**

Now we show a complexity-theoretic evidence that it is unlikely that 3-CNFs are learnable with an SPP-assistant or an LWPP-assistant. We prove the following theorem.

**Theorem 5.5.6** *If* 3-CNF *is* FP-*learnable with an* SPP-*assistant, then* NP $\subseteq$ SPP. *If* 3-CNF *is* FP-*learnable with an* LWPP-*assistant, then* NP $\subseteq$ LWPP.

*Proof.* We prove that if 3-CNF is FP-learnable with an SPP-assistant, then NP $\subseteq$ SPP. The proof of the second statement is identical and is omitted. For proving this, we show that if 3-CNF has a learning algorithm with an SPP-assistant, then the co-NP-complete language TAUT= $\{\langle 0^n, 0^l, f\rangle | f$ encodes 3-CNF on $n$ variables of size $\leq l$ and $f$ is a tautology$\}$ is in $\mathrm{P}^{\mathrm{SPP}}$ which is SPP [FFK94]. Since SPP is closed under complement, the result follows.

Let $\alpha$ be an FP-learner, which on input $\langle 0^n, 0^l\rangle$ learns an $n$-variable 3-CNF of size less than or equal to $l$ with an SPP-assistant $L($ 3-CNF$) = \{\langle n, l, y, c\rangle | y \in$

$\Sigma^*; Q(n, l, y, c)\}$ accepted by a machine $N$, where $Q$ is the predicate defining the SPP-assistant. Define a language $A = \{\langle 0^n, 0^l, y, f\rangle | y \in \Sigma^*; f$ encodes a 3-CNF formula and $Q'(n, l, y, f)\}$ where the predicate $Q'$ is such that if $Q'(n, l, y, f)$ is true then $N$ on input $\langle 0^n, 0^l, y\rangle$ with $f$ as target formula has $gap = 1$ and if $Q'(n, l, y, f)$ is false then $N$ will have a $gap = 0$. It is easy to see that $A \in$ SPP. Consider the deterministic machine $M$, for accepting TAUT, which on input $\langle 0^n, 0^l, f\rangle$ does the following. $M$ simulates $\alpha$ on input $n$ two times, sequentially. In the first simulation whenever $\alpha$ makes a query $\langle n, l, y, c\rangle$ to $L($ 3-CNF$)$, $M$ makes a query $\langle 0^n, 0^l, y, f\rangle$ to $A$. In the next simulation whenever $\alpha$ makes a query $\langle n, l, y, c\rangle$ to $L($ 3-CNF$)$, $M$ makes a query $\langle 0^n, 0^l, y, T\rangle$ to $A$ ($T$ is the trivial formula *true*). $M$ accepts $\langle 0^n, 0^l, f\rangle$ if the output of $M$ in both the cases are the same. Now we crucially observe that since $\alpha$ is a deterministic machine it outputs the same string in both simulations iff $f$ is a tautology. Theorem follows. ∎

## 5.6 Summary

In this chapter we investigated the complexity of exact learning some algebraic representation classes. Our major interest was in learning the class of permutation groups and linear spaces over finite fields. First we observed that Angluin's model is insufficient for a fine classification of the complexity of these representation classes. Hence, we proposed a refinement of Angluin's model called the teaching assistant model of exact learning. To classify the complexity of learning various representation classes in the teaching assistant model, we introduced the notion of teaching assistant classes and defined some teaching assistant classes analogous to well-known complexity classes. Among these, the teaching assistant classes analogous to the complexity classes SPP and LWPP were of importance. As our main results, we showed that permutation groups are polynomial-time learnable with LWPP-assistants and linear spaces are polynomial-time learnable with SPP-assistants. We also showed that it is unlikely that the representation class 3-CNF is polynomial time learnable with any of these assistant classes. These results formalize the intuition the complexity of learning 3-CNF is quite different from that of learning either permutation groups or linear spaces. Notice that in Angluin's model, it is not possible to bring out this difference.

In the next chapter we continue our investigations on the complexity of exact learning. We define more assistant classes and prove some absolute separations among these classes. We use subclasses of the representation class SYM for showing these separations.