IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 8 January 2023, accepted 30 January 2023, date of publication 3 February 2023, date of current version 9 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3242124

RESEARCH ARTICLE

A Reinforcement Learning Based Grammatical Inference Algorithm Using Block-Based Delta Inverse Strategy

FARAH HANEEF AND MUDDASSAR AZAM SINDHU
Department of Computer Science, Quaid-i-Azam University, Islamabad 45320, Pakistan
Corresponding author: Farah Haneef (farah@cs.qau.edu.pk)

This work was supported by the Higher Education Commission of Pakistan (HEC) under Grant 9223/Federal/NRPU/RD/HEC/2017.

ABSTRACT A resurgent interest for grammatical inference aka automaton learning has emerged in several intriguing areas of computer sciences such as machine learning, software engineering, robotics and internet of things. An automaton learning algorithm commonly uses queries to learn the regular grammar of a *Deterministic Finite Automaton* (DFA). These queries are posed to a *Minimum Adequate Teacher* (MAT) by the *learner* (Learning Algorithm). The membership and equivalence queries which the learning algorithm may pose, are often capable of having their answers provided by the MAT. The three main categories of learning algorithms are incremental, sequential, and complete learning algorithms. In the presence of a MAT, the time complexity of existing DFA learning algorithms is polynomial. Therefore, in some applications these algorithms may fail to learn the system. In this study, we have reduced the time complexity of DFA learning from polynomial to logarithmic form. For this, we propose an efficient complete DFA learning algorithm; the *Block based DFA Learning through Inverse Query* (BDLIQ) using block based delta inverse strategy, which is based on the idea of inverse queries that John Hopcroft introduced for state minimization of a DFA. The BDLIQ algorithm possess $O(|\Sigma|N.logN)$ complexity when a MAT is available. The MAT is also made capable of responding to inverse queries. We provide theoretical and empirical analysis of the proposed algorithm. Results show that our suggested approach for complete learning; BDLIQ algorithm, is more efficient than the ID algorithm in terms of time complexity.

INDEX TERMS Automaton learning algorithm, complete learning, inverse query, machine learning, reinforcement learning.

I. INTRODUCTION

Automaton learning, also known as grammatical inference, is a field in which a system is inferred in the form of an automaton by providing a sequence of inputs $(i_1, i_2, \ldots, i_n)$, and then synthesising the corresponding output sequence $(o_1, o_2, \ldots, o_n)$.There are two main concepts used in the automated learning; the *Learner* and the *Minimal Adequate Teacher* (MAT) [4]. Depending on the setting that the learning algorithm provides, the learner learns the regular set through questions and counterexamples. The learner asks questions to the MAT, and the MAT provides answers regarding the unidentified regular set. It responds to two different questions:A membership query is the first type and consists of the string $t \in \Sigma^*$. Depending on whether or not string t is a part of the unidentified regular set, the teacher responds as ''yes'' or ''no''. The second sort of query is a conjecture, which is made up of a description of the regular set S. If S is behaviorally equivalent to the unknown language, the response is ''yes,'' otherwise it is a string t in the symmetric difference between S and the unknown language. In the second instance, the string t is referred to as a witness or a counterexample because it serves to refute the conjectured set S.

Automaton learning algorithms are created in such a way that they *learn in the limit* to produce the isomorphic representation of the desired *Deterministic Finite Automaton*

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Da Lin.

(DFA). E.M. Gold [19] developed the idea of ''learning in the limit'' for the first time in 1967. In his paper, he demonstrated how a regular language corresponding to some unidentified target DFA may be inferred by a finite number of inquiries or guesses using a grammatical inference or an automaton learning algorithm.

There are three main categories of algorithms proposed in literature for automaton learning [35], [36], including incremental learning algorithms, sequential learning algorithms and complete learning algorithms. In incremental learning, the system under learning (SUL) is learnt in a series of increments i $= 1, 2, \ldots, n$. At the conclusion of each increment, the learner formulates the hypothesis DFA M_i and asks the teacher an equivalence question. The teacher may or may not give a counterexample in the event that the equivalency question receives a negative response. If the teacher gives a counterexample, the learner expands its learning process based on the counterexample it received and incorporates the learning material from the prior increment(s) into the new increments. Similarly, learning is also done in a number of increments in sequential learning, but the learner begins learning from scratch and does not draw on the knowledge from earlier increments. A complete learning involves learning the entire system what it needs to know in order to produce a hypothesis. The hypothesis DFA, M, is generated only when the learner has mastered the entire system (SUL).

The ability of grammatical inference [32], [39] to address a variety of real-world applications of machine learning [18], [20], [45], software engineering [1], [2], [37], robotics, artificial intelligence and big data [27] has led to its use in recent years by the computer science research community. In general, these applications employ the idea of inferring an automaton by creating a model of a system under learn (SUL) and examining it to determine whether its behaviour adheres to a specification.

In the presence of MAT, the current automaton learning algorithms have polynomial time complexity (at least in cubic form). The current automaton learning algorithms require a lot of time when inferring the model of the System Under Learn (SUL). These algorithms can occasionally fail to learn extremely complicated systems. In the paper [40] authors specifically give some real-world examples of 6 transition systems of CSS processes [13], [14], [15] like buffers, schedulers, vending machines, and mutual exclusion protocols [34], [44] where they fail to learn them due to inefficient learning algorithms (in terms of time) and a lack of storage space. In there study, authors also emphasize the need for a good automaton learning algorithm that is sufficiently efficient in terms of execution time and memory. According to the existing literature [7], even though significant progress has been made in the development of DFA learning algorithms, many researchers agree that more effective automaton learning algorithms must still be developed in order to address real-world learning issues and situations [42].

Due to the fact that the notion of the minimal adequate teacher (MAT) was first time introduced in the ID (Identification of Regular Languages) algorithm and without it the learning of a DFA is an NP-hard problem therefore in this study we propose a new efficient DFA learning algorithm called BDLIQ based on the ID algorithm. In contrast to the DLIQ algorithm described in our earlier work [21], the BDLIQ algorithm does not employ the live complete set (as an input) for learning purposes, which poses a limitation to learning the *System Under Learn* (SUL) in many practical applications. The BDLIQ algorithm substitutes it with a set of input alphabet Σ for block-based learning. This learning approach also reduces its complexity from polynomial to logarithmic form.

By partitioning the states of *System Under Learn* (SUL) into blocks of final and non-final states, the BDLIQ algorithm identify behaviorally equivalent and non-equivalent states by traversing back to the initial state using inverse queries, which were first put forth by John Hopcroft for state minimization of DFA.

The main contributions of this research work are as follows:

1) We develop a novel and an efficient DFA learning algorithm that reduces the worst-case time complexity of complete learning process to logarithmic form.
2) We introduce the concept of *delta inverse* (δ^{-1}) and *Inverse Queries*(IQ) in *DFA learning*.
3) We improve the capabilities of the current MAT to make it capable of answering inverse queries.

The remainder of the paper is structured as follows: in Section II, we provide background information to help readers comprehend the proposed algorithm and in Section III, we discuss relevant research in the field. The proposed BDLIQ algorithm is presented in Section IV. In Section V we study and describe the BDLIQ algorithm's time complexity, the BDLIQ algorithm's termination and correctness are proven in Section VI, and a functioning example of the algorithm is demonstrated in Section VII. We contrast the effectiveness of our suggested BDLIQ algorithm and the ID algorithm in Section VIII. At the end, in Section IX, we conclude our findings.

II. PRELIMINARIES AND NOTATIONS

A *Deterministic Finite Automaton* (DFA) $\mathcal{A}$ consists of a five tuples $\langle Q, \Sigma, \delta, q_0, F \rangle$ where:
Q denotes the finite set of states.
A finite set of input symbols is represented as Σ.
The transition function δ determines the subsequent state when an input symbol is read from a certain state $\delta : Q \times \Sigma \rightarrow Q$.
The start state is represented by the state $q_0 \in Q$.
The collection of final states is $F \subseteq Q$.

Definition 1: Let for a DFA A, having transition function $\delta : Q \times \Sigma \rightarrow Q$ which can also be written as $\delta(q_i, \sigma) = q_j$ and the iterated transition function $\delta^* : Q \times \Sigma^* \rightarrow Q$

inductively defined by $\delta(q, \lambda) = q$ where λ is an empty string and $\delta^*(q_j, b_1, b_2, \ldots, b_n) = \delta(\delta^*(q, b_1, b_2, \ldots, b_{n-1}), b_n)$

Likewise, we inductively define δ^{-1*} using the inverse transition relation δ^{-1}. Where $\delta^{-1} : Q \times \Sigma \subseteq Q$ and can also be written as $\delta^{-1}(q_j, \sigma) \subseteq Q$ where Σ^{-1} denotes an inverse transition by reading an element of Σ from a state Q to give its predecessor states. The inductive definition of δ^{-1*} is now simple to follow as $\delta^{-1}(q, \lambda) = q$, if and only if q is a starting state otherwise it returns $\emptyset$ and $\delta^{-1^*}(q_j, b_1, b_2, \ldots, b_n) = \delta^{-1}(\delta^{-1^*}(\{q\}, b_2, \ldots, b_n), b_1)$. □

The *Inverse Query* (IQ) is a question which is asked by the learner from the MAT about the predecessor state(s) of a state q_j, by reading some string $\alpha \in \Sigma^*$ from it, i.e., $\delta^{-1*}(q_j, \alpha) =?$ The teacher's response is either an empty set or a set of one or more states.

A *block* is a collection of states indicated by the symbol $B(num)$, where *num* is the block number.

Let $B(k)$ stand for the k^{th} block in the collection of blocks. The number of states in a block is indicated by the symbol $|B(k)|$, which also serves as the block's size.

A *LearningBlock* is a collection of all the blocks that have already been en-queued for learning purposes in *BlockQueue*, where *BlockQueue* is a queue and each element is represented as B_{new}.

A collection of blocks called B_{num} is used to keep predecessor states in the corresponding block number, such as $B(num'')$.

The *BlockSet* is a collection of blocks that correspond to the states of the hypothesis DFA.

III. RELATED WORK

Many studies have been done on automaton learning during the previous few decades. Some researchers have concentrated in particular on previously developed learning algorithms and have attempted to determine the effects of the size of the alphabet and the number of states on the required number of membership queries. Some researchers have contributed by outlining the restrictions placed on equivalence queries, using [5]. Numerous of them have concentrated primarily on the kinds of queries and their importance in automaton learning [6], [8], [9], [10], [11], [22] and in learning regular expressions [26].

Some researchers have studied how to use grammatical inference in various contexts like robotic planning [12] vehicle platooning [31], [25] and automated verification of distributed control programs [24]. Some researchers have proposed the algorithms to learn the random DFA [7], nondeterministic input enabled labelled transition systems [43] and Buchi Automata based on families of DFA's [28].

Grammatical inference algorithms have been proposed by numerous scholars such as L* [4], ID [3], IID [33], IDS [38], IKL [30], kerans [23], RPNI [16], RPNII [17] and DKL [29] for learning and testing purposes. Some of them are incremental in nature whereas some work has been done on the basis of complete learning.

According to the best of our knowledge, in literature the existing complete learning algorithms other than DLIQ are L^* [4], ID [3] and RPNI [16].

The learning algorithm L^* is complete. Two different types of queries; membership and equivalence queries are used to infer a regular language. It asks questions about membership and stores the answers in a table known as a *observationtable* (OT). Before making a hypothesis and requesting equivalence questions, the OT must satisfy two fundamental criteria. According to [4], these characteristics include *closure* and *consistency*. A hypothesis can be created when the OT is *closed* and *consistent*. Up until OT becomes consistent and closed, the L^* continues to learn.

According to [3], the ID algorithm is a complete learning algorithm. To learn the regular set, it poses membership questions from the teacher (MAT). This algorithm was the one that initially introduced the idea of MAT. The terms live states, live complete set, distinguishing strings and dead state d_0 are used in this algorithm. The ID algorithm creates a table and searches it for the blocks of accepting and non-accepting states. When the first iteration is finished, the ID algorithm selects two strings from the live complete set that behave similarly, but when some $\sigma \in \Sigma$ is concatenated with one of the strings, the other behaves differently, moving to the rejecting block. This provides a possible distinguishing string. The hypothesis DFA, $\mathcal{H}$, which is isomorphic to the target DFA is constructed by the ID algorithm if it fails to find such a pair of strings.

According to Oncina Garcia in 1992, the RPNI algorithm is a passive learning algorithm. The information regarding the hypothesis is stored in a tree data structure rather than a table, and consistency is not maintained. It doesn't employ membership queries for learning purposes. It requires two input sets, S_+ and S_-, which stand for positive and negative examples, respectively. It creates the prefix tree $PT(S_+)$ from the set of positive examples and their prefixes after first writing the items of S_+ and its prefixes in lexical order. Then, it divides the tree's branches into blocks in a recursive manner. Each element of $PT(S_+)$ is initially a part of the block that it self-contains. These blocks are subjected to joint operation in a recursive manner by the RPNI algorithm in order to be combined into two final blocks. The accepting state block is the first, while the non-accepting state block is the second.

If we examine the complexity of these complete learning algorithms, we can find that, when a adequate teacher is present, the complexity of these learning algorithms is polynomial as shown in table 1. The John Hopcroft established the idea of state minimization, and in his algorithm he had employed the tactic of creating blocks of final and non-final states. He found the similar states to create the minimal target automaton using the δ^{-1} transition approach. He stated that his technique generates a minimal target automaton with *nlogn* complexity.

In this paper, we present BDLIQ, a new effective DFA learning algorithm based on the concepts of the ID algorithm

along with inverse transition strategy of John Hopcroft algorithm for state minimization of DFAs, which is based on the existing literature for automaton learning and state minimization of an automaton. The current study aims to decrease bounds for complete DFA learning by reducing the complexity of DFA learning from polynomial(cubic) to logarithmic form. The comparison of our proposed approach with existing DFA (Complete) learning algorithms is provided in table 1.

TABLE 1. Summary of existing DFA complete learning algorithms.

Algorithm	Data Structure	MQ	IQ	Time Complexities
L^*	Table	Yes	No	$O(\|\Sigma\|.N^2M)$
ID	Table	Yes	No	$O(\|\Sigma\|.\|P\|.N)$
RPNI	Tree	No	No	$O((\|S_p\|+\|S_n\|).\|S_p\|^2)$

MQ: Membership Query, IQ: Inverse Query
$|\Sigma|$ represents the size of input alphabet, N represents the number of states in target DFA, M represents the number of membership queries, P represents the set of live complete set, $|S_p|$ and $|S_n|$ represent the size of positive and negative labeled examples respectively.

IV. THE PROPOSED BDLIQ ALGORITHM

The proposed BDLIQ algorithm is presented in Algorithms 1 and 2.

Algorithm 1 BDLIQ Algorithm

Input: A set of positive labeled examples S_+ and a DFA A to act as a teacher to answer queries.
Output: A DFA M equivalent to the target DFA A.
Step 1:
Initiate *BlockSet* = $\emptyset$, *BlockQueue* = *empty*, *LearningBlock* = $\emptyset$
Identify the final states block B(1) with the help of S_+ by asking MQ for all $s_i \in S_+$ as for all $\delta(s_i, \lambda) = q_i$ where $q_i \in F$
B(1) = F //Final states block
Step 2:
Make table δ^{-1} by using final state set F and all their predecessor states, for all input elements σ along with empty string λ, where $\sigma \in \Sigma$
To identify the non-final states block, create B(temp) which consists of all the predecessor states of final state block B(1), (given in δ^{-1} Table)
B(2) = $(Q - F)$ where Q = B(1) $\cup$ B(temp) // B(2) is non-final states block
Update *BlockSet*
Step 3:
k = 3
Step 4:
enQueue *(BlockQueue*(B(1))
Update *LearningBlock* //put B(1) in *LearningBlock*
while *BlockQueue* != *empty* **do**
  deQueue(B_{new})
  for all $\sigma_i \in \Sigma$ ask inverse query for B_{new} where $B_{new} \in$ *BlockQueue* **do**
    // finding predecessor state(s) of set B_{new} via reading element σ.
    there are three possibilities against the response of inverse query; no predecessor state, one predecessor state or multiple predecessor states.
    if $\delta^{-1*}(B_{new}, \sigma_i) = \emptyset$ **then**
      goto Line 15;
    else if $\delta^{-1*}(B_{new}, \sigma_i) = \{q_j\}$ **then**
      BQuery(q_j, *BlockSet*) = *num* // Block Membership call using Algorithm 2
      $B(k') = \{q_j\}$
      if $|B(k')| < |B(num)|$ **then**
        $B(k) = B(k')$
        $B(num) = B(num) - B(k)$
        enQueue (*BlockQueue*($B(k)$))
        Update *BlockSet*
        Update*LearningBlock* // put B(k) in *LearningBlock*
      end if
      if $|B(k')| == |B(num)|$ **then**
        if $B(num) \notin$ *LearningBlock* **then**
          enQueue (*BlockQueue*($B(num)$))
          Update *LearningBlock* // put $B(num)$ in *LearningBlock*
        end if
      end if
    else if $\delta^{-1*}(B_{new}, \sigma_i) = \{q_1, \ldots, q_m\}$ **then**
      for $j = 1$ to $j = m$ **do**
        BQuery(q_j, *BlockSet*) = *num* // Block Membership call using Algorithm2.
        //All predecessor states those belong to the same block are placed in a single block named as $B(num'')$ such as:
        $B(num'')$=$B(num'')$ $\cup$ $\{q_j\}$ where $B(num'')$ belongs to the list B_{num}
      end for
      $B(k') = B(num'')$
      if $|B(k')| < |B(num)|$ **then**
        $B(k) = B(k')$
        $B(num) = B(num) - B(k)$
        enQueue (*BlockQueue*($B(k)$))
        Update *BlockSet*
        Update *LearningBlock* // put B(k) in *LearningBlock*
      end if
      if $|B(k')| == |B(num)|$ **then**
        if $B(num) \notin$ *LearningBlock* **then**
          enQueue (*BlockQueue*($B(num)$))
          Update *LearningBlock* // put $B(num)$ in *LearningBlock*
        end if
      end if
    end if
    Step 5
    if($B(k) \neq \emptyset$) **then** k++
    if($B_{num} \neq \emptyset$) **then** goto Line 39
  end for
end while

59: Generate hypothesis automaton M by reading inverse transitions from δ^{-1} table.

Algorithm 2 Block Membership Algorithm

1: Input: A state q_j where $q_j \in Q$ and a set of existing blocks *BlockSet*.
2: Output: Block number named as *num*, from where q_j belongs to.
3: **Function** BQuery(q_j, *BlockSet*)
4: {
5: **return** *num*
6: }

V. COMPLEXITY ANALYSIS

The minimum adequate teacher (MAT) has polynomial time complexity, but for the query complexity analysis, [4] its complexity is taken as constant. For a membership and δ^{-1} query, the MAT has $O(1)$ time complexity, however for an equivalence query, it has polynomial time complexity.

Table construction takes $O(|\Sigma|N + 1)$ time complexity. In each iteration of inner for loop, the learner asks δ^{-1} query against an element of set Σ (used as distinguishing string in the proposed algorithm) therefore, its time complexity is $O(|\Sigma|)$ whereas, (as for a newly created block) the number of states visited at each step are maximum of $logN$ whereas outer while loop executes maximum N times (N is number of states) therefore, the nested loop takes $O(|\Sigma|.NlogN)$ time complexity.

The block splitting process takes $O(1)$ time complexity. Automaton reconstruction is done by reading the δ^{-1} table therefore, it also takes $O(|\Sigma|N)$ time complexity. In view of above asymptotic analysis, the worst case time complexity of the proposed algorithm is $O(|\Sigma|NlogN)$.

VI. CORRECTNESS AND TERMINATION

The algorithm's accuracy is proven by the fact that it successfully creates a learned DFA that is consistent with the desired DFA A.

Definition 2: The splitting of a block $B(num)$ into $B(1)$ and $B(2)$ is done on the basis of the $B(1) \cap B(2) = \emptyset$ and $B(1) \cup B(2) = B(num)$ properties, which are mutually exclusive and totally exhaustive, respectively.

*Theorem 1 (**Termination Theorem**):* Let *LearningBlock* be a set consisting of all the blocks (possessing by *BlockQueue*) which have already been used for learning new blocks. The execution of BDLIQ on *BlockQueue* terminates when the $|LearningBlock| = N$ and N is number of distinct blocks at the end of the execution.

Proof: As the *LearningBlock* consists of all blocks those are enqueued in the *BlockQueue*. So it is obvious that the while loop of Algorithm 2 terminates when the *BlockQueue* is empty (completely exhausted) and all the distinct blocks reside in the *LearningBlock* such as $|LearningBlock| = N$. □

*Theorem 2 (**Correctness Theorem**):* The BDLIQ algorithm terminates on *BlockQueue* and the hypothesis automaton M is a canonical representation of A.

Proof: In order to learn more about the grammar of the target DFA A, the BDLIQ algorithm poses inverse and equivalence queries. The fact that BDLIQ explicitly learns through inverse queries must be noted. To establish this theorem, we rely on the following two premises: (a) The Termination Theorem 1 establishes that learning terminates when all elements of the set *LearningBlock* those are enqueued in *BlockQueue* are exhausted. (b) The BDLIQ algorithm combines all the states of target automaton A that have behavior-equivalent values into a single state block, and the hypothesis automaton M is then built utilising all the blocks formed as states of the learnt hypothesis. As a result, the hypothesis automaton M is a unique minimal representation of A. Combining (a) and (b) prove the theorem. □

VII. AN EXAMPLE

We now provide an example to demonstrate how the BDLIQ algorithm functions. For this, we consider an example automaton taken from [41] given in Fig. 1.

The inputs of the algorithm consist of the Target Automaton (A) (given in Figure 1 and a set of positive labeled examples $S_+ = \{01\}$.
Initially, BlockSet = ∅, BlockQueue = empty and LearningBlock = ∅
According to the Line 4 of Algorithm 2, B(1) = {C}. According to the Step 2 the delta inverse table is created as described in table 2.

TABLE 2. δ^{-1} **Table.**

States/Inputs	λ	0	1
C	∅	F, D	B, C, H
F	∅	∅	A, E
D	∅	∅	∅
B	∅	A	∅
H	∅	E	∅
A	A	C	∅
E	∅	∅	G
G	∅	B, H, G	F, D

According to the Line 7 of Algorithm 2, the predecessor states of final state set; B(1) are A, B, D, E, F, G, H are used to find the non-final states block as B(2) = {A, B, D, E, F, G, H}. According to the Line 9, now BlockSet = {B(1), B(2)}. According to the step 3 of Algorithm 2, now k = 3. According to the Line 12, enqueue the final state block B(1) in BlockQueue so now BlockQueue = [B(1)].
According to the Line 13, LearningBlock = {B(1)}. Now according to the Line 14, as BlockQueue is not empty therefore dequeue B(1) from BlockQueue as B_{new} = B(1). According to the Line 15, $\sigma_i = 0$ therefore with the help of IQ, find predecessor state(s) of block B(1) for input 0. According to the Line 35, predecessor state of B(1) via reading 0 is F and D where F and D both belong to the block B(2) so B(2'')={F, D}. According to the Line 39, B(3') = B(2'') so B(3') = {F, D}. According to the Line 40 as $|B(3')| < |B(2)|$

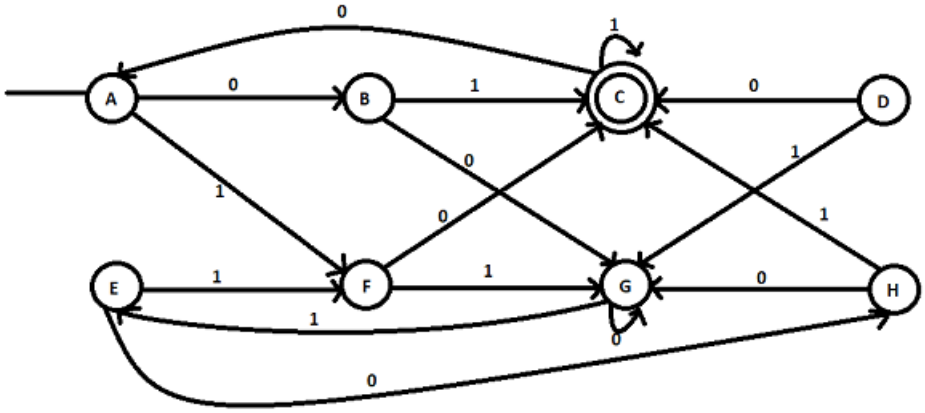

FIGURE 1. Target automaton *A* (SUL).

so B(3) = {F, D} and B(2) = {A, B, E, G, H}. According to the Line 43, enqueue B(3) in BlockQueue therefore according to the Line 44 and 45, BlockSet = {B(1), B(2), B(3)} and LearningBlock = {B(1), B(3)}. According to the Line 47, as if condition is wrong so goto the Line 55. As B(3) $\neq \emptyset$ so k = 4. According to the Line 56 as $B_{num} = \emptyset$ therefore go to the next iteration of for loop.

According to the Line 15 of Algorithm 2, $\sigma_i = 1$. Predecessor states of B(1) via reading 1 is B, C and H. According to the Line 35, B and H belong to the B(2) and C belongs to the B(1) therefore, B(2'') = {B, H}, B(1'') = {C} and B_{num} = {B(2''), B(1'')}. According to the Line 39, B(4') = B(2'') = {B, H}. According to the Line 40, as $|B(4')| \leq |B(2)|$ therefore, B(4) = {B, H} and B(2) = { A, G, E}. According to the Line 43, enqueue B(4) in BlockQueue so BlockSet = {B(1), B(2), B(3), B(4)} and LearningBlock = {B(1), B(3), B(4)}. As on Line 47, if condition false so goto Line 55. As B(4) $\neq \emptyset$ therefore, k = 5. According to the Line 56, as $B_{num} \neq \emptyset$ so goto the Line 39. Now B(5') = B(1'') = {C}. According to the Line 40, as $|B(5')|$ is not less than $|B(1)|$ therefore, goto the Line 48. According to the Line 48 as B(1) belongs to the LearningBlock set so if condition does not execute. Again goto the Line 55. B(5) = $\emptyset$ so k remains same as k = 5. According to the Line 56 now $B_{num} = \emptyset$ so inner for loop ends. Now goto the Line 13 as BlockQueue is not empty so B_{new} = B(3) and $\sigma_i = 0$.

According to the Line 16, predecessor states of B(3) via reading 0 is $\emptyset$ so according to the Line 18, goto the Line 15. Now $\sigma_i = 1$. According to the Line 16, predecessor states of B(3) via reading 1 are A, E. According to the Line 35, A and E both belong to the B(2) so B(2'') = {A, E}. According to the Line 39, B(5') = {A, E} as $|B(5')| < |B(2)|$ therefore B(5) = {A, E} and B(2) = {G}. According to the Lines 43, 4, 45 BlockQueue = {B(4), B(5)}, BlockSet = {B(1), B(2), B(3), B(4), B(5)} and LearningBlock = {B(1), B(3), B(4), B(5)}. As Line 47, condition is false therefore goto the Line 55. As B(5) $\neq \emptyset$ so k = 6. As $B_{num} = \emptyset$ so inner for loop ends. Now goto the outer while loop at Line 14. As BlockQueue is not empty so enqueue next element as B_{new} = B(4) and $\sigma_i = 0$.

Predecessor states of B(4) via reading 0 is A and E. A, E both belong to the B(5). So according to the Line 37, B(5'') = {A, E} and B_{num} = {B(5'')}. According to the Line 40 if condition is false therefore goto the Line 47. As $|B(6')| = |B(5)|$ therefore goto the Line 48. As B(5) belongs to the LearningBlock set so if condition does not execute and control goto the Line 55. As B(6) = $\emptyset$ so k remains same as k = 6. According to the Line 56 as $B_{num} = \emptyset$ therefore goto the Line 15. Now $\sigma_i = 1$. Predecessor states of B(4) via reading 1 is $\emptyset$ so inner loop ends. Now goto the Line 14 as BlockQueue is not empty so dequeue next element as B_{new} = B(5) and now $\sigma_i = 0$.

Predecessor states of B(5) via reading 0 is C. According to the Line 20, as C belongs to B(1) so B(6') = {C}. As if condition false at Line 22 so goto the Line 29. As $|B(6')| = |B(1)|$ and according to the Line 30, B(1) belongs to the set LearningBlock so any condition of step 5 also does not meet. Now again goto the Line 15. Now $\sigma_i = 1$. Predecessor state of B(5) via reading 1 is G. G belongs to B(2). According to the Line 21, B(6') = {G}. As according to the Line 29, $|B(6')| = |B(2)|$ and according to the Line 30, B(2) does not belong to the set of LearningBlock therefore, enqueue B(2) in BlockQueue so, now BlockQueue = [B(2)] and LearningBlock = {B(1), B(3), B(4), B(5), B(2)}. Now goto the Line 55 as B(6) = $\emptyset$ so k remains the same as k = 6 and B_{num} = *emptyset* therefore, for loop ends. Again goto the out while loop at the Line 14. Now B_{new} = B(2). Execute the nested loop. We can see that now no new block is created and ultimately BlockQueue remains empty therefore algorithm terminates. After termination, the final blocks will be: {{C}, {B, H}, {A, E}, {D, F} and {G}}.

Automaton Construction: The final number of blocks in BlockSet shows the number of states of hypothesis DFA. Each block represents a single state. By reading the δ^{-1} table input transitions, algorithm will construct the hypothesis automaton *M* (described in Fig. 2).

VIII. COMPARISON OF BDLIQ AND ID ALGORITHM

To compare the performance of BDLIQ with other relevant algorithms, we looked at the algorithms listed in Table 1, which lists all the complete learning algorithms that are currently available in the literature. Only three other learning

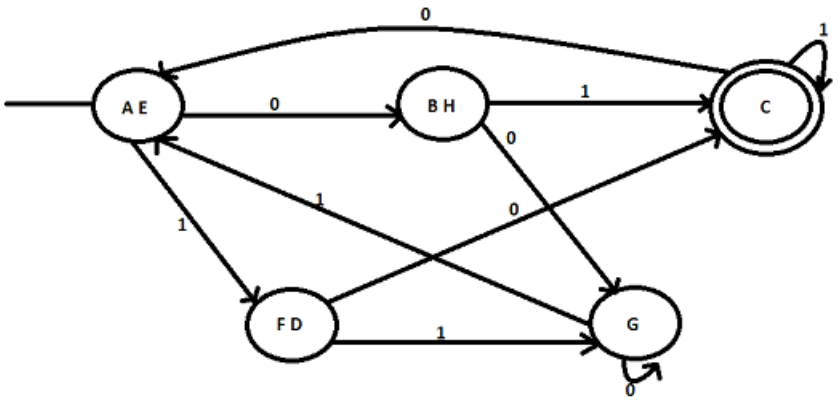

FIGURE 2. Hypothesis automaton *M*.

algorithms, the L*, ID, and RPNI, are complete learning algorithms, similar to the BDLIQ algorithm proposed in this study. The RPNI cannot be compared to BDLIQ since it is based on passive learning while the L*, ID, and BDLIQ are based on active learning. Therefore, only the L* and/or the ID can be utilised for BDLIQ comparison. However, the L^*'s obligation to utilise a counterexample whenever the hypothesis it builds is not equal to the target automaton is a key distinction from other. This characteristic is lacking in both the ID and the BDLIQ algorithms and makes learning more directed when compared to them. Therefore, We were left with only the ID algorithm to perform comparison with the BDLIQ algorithm as both share almost all comparison parameters with the exception of the inverse query.

For the comparison of BDLIQ and ID algorithms we have setup an evaluation framework (given in Fig. 3) consisting of following modules:

1) A target DFA (A)
2) A random DFA generator
3) BDLIQ and ID algorithms
4) A DFA equivalence checker

A. EXPERIMENTAL SETUP

We have used Java language to implement the BDLIQ and ID algorithms. We used a computer with Windows 8.1 pro, 16GB of RAM, and an Intel Core i5-3470 processor to carry out the research. We have run numerous tests on both of these algorithms. Due to two key parameters of the target DFA A, the tests varied. These parameters include:

1) The state size $|Q|$
2) The input alphabet size $|\Sigma|$

We set up the tests to be run with state size $|Q|$ ranging between $10, 20, 30, \ldots, 100$ and alphabet size varied between $2, 4, \ldots, 10$ in order to analyse the performance of both of these algorithms. Randomly generated target DFAs included every possible pairing of the two parameters such as for ($|Q|$=10, $|\Sigma|$=2), ($|Q|$=10, $|\Sigma|$=4), ($|Q|$=10, $|\Sigma|$=6), ($|Q|$=10, $|\Sigma|$=8), ($|Q|$=10, $|\Sigma|$=10). Additionally, we

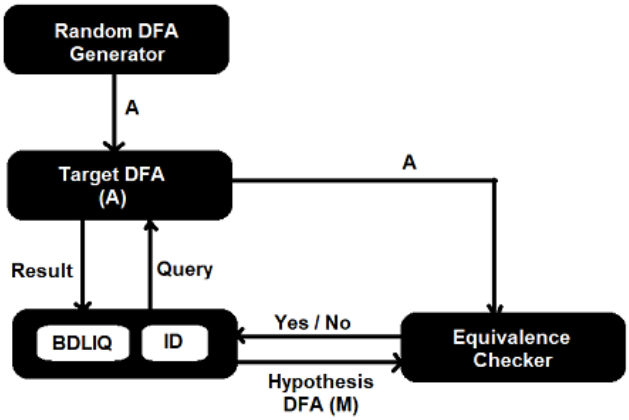

FIGURE 3. Evaluation framework.

conducted the studies ten times for a given parameter configuration before compiling the data to determine the mean of the ten trials.

B. EMPIRICAL EVALUATION

For comparison of both these algorithms (BDLIQ and ID), we have considered the following two parameters:

1) Number of queries (posed by the learner to the MAT for learning purpose)
2) The learning time(ms) (time taken by equivalence checker not included)

C. RESULTS AND ANALYSIS

The calculated results are provided in tables 3, 4, 5 and 6. Results given in table 4 show that due to learning through inverse queries instead of membership queries the overall number of queries posed by the BDLIQ algorithm for learning purpose is very small than the ID algorithm (given in table 3). Even the calculated values of the BDLIQ algorithm against the parameter set $|\Sigma| = 10$ and number of states $|Q| = 100$ is around 40 times less than the calculated values of ID algorithm against same parameter set. Even these values

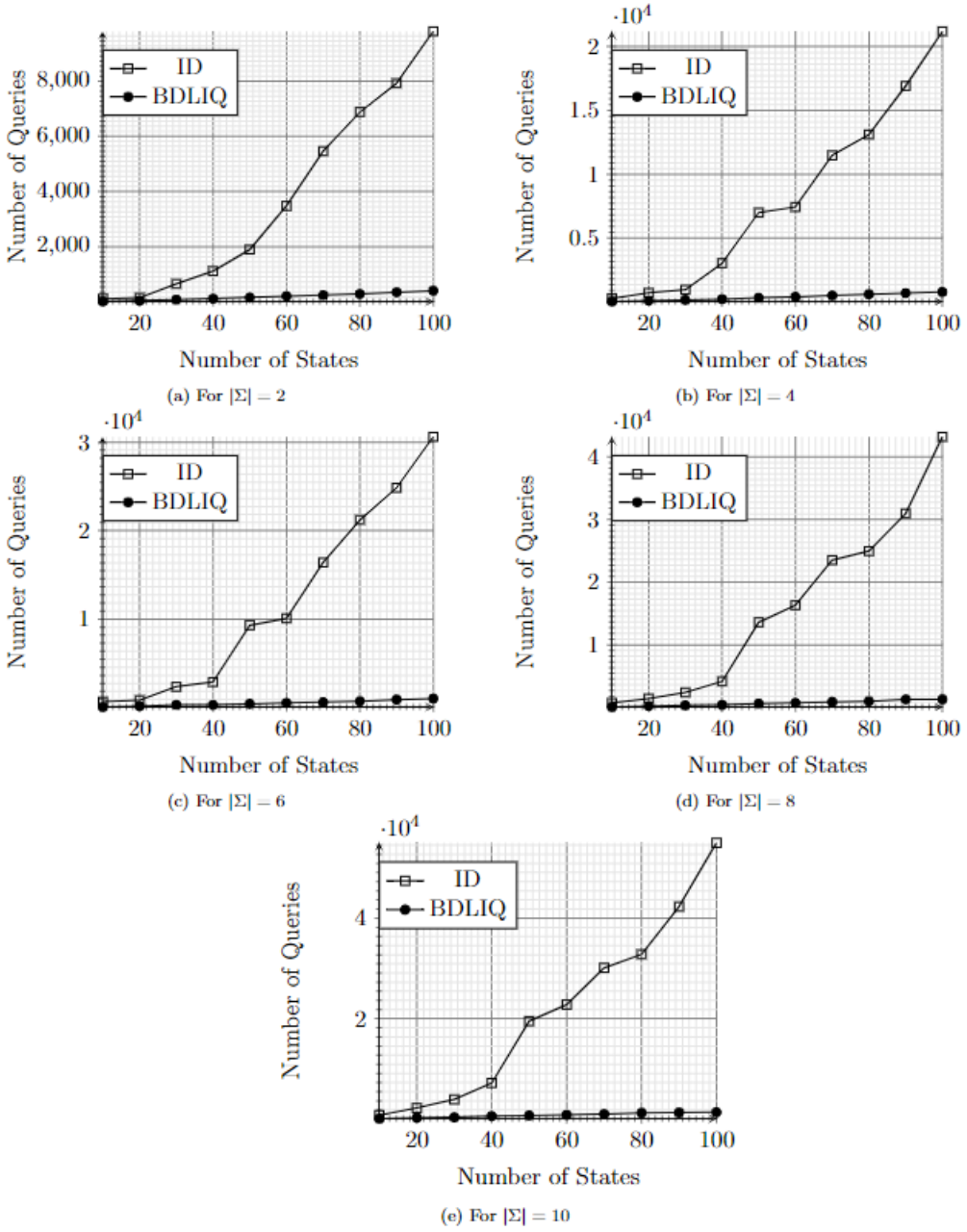

FIGURE 4. Comparison of BDLIQ and ID algorithm with respect to queries.

are less than the parameter set $|\Sigma| = 2$ and number of states $|Q| = 100$ of ID algorithm. Similarly, we can see that results given in table 5 and table 6 describe the same picture regarding learning times. Using delta inverse strategy in BDLIQ algorithm, the learning time of BDLIQ algorithm is very less than the ID algorithm which uses conventional delta strategy for learning purpose. We can see that the maximum value against the parameter set $|\Sigma| = 10$ and number of states $|Q| = 100$ is more than 20 times less than the value computed against the learning time of ID algorithm which clearly shows the efficiency of BDLIQ algorithm over ID algorithm with respect to learning time.

According to the comparative analysis presented in Fig.4 and Fig.5, we can see that due to the large number of queries posed by the ID algorithm for learning purpose, it experiences a significant increase in the number of queries and learning time as the number of states or input alphabet size increases, whereas due to the delta inverse strategy and inverse queries

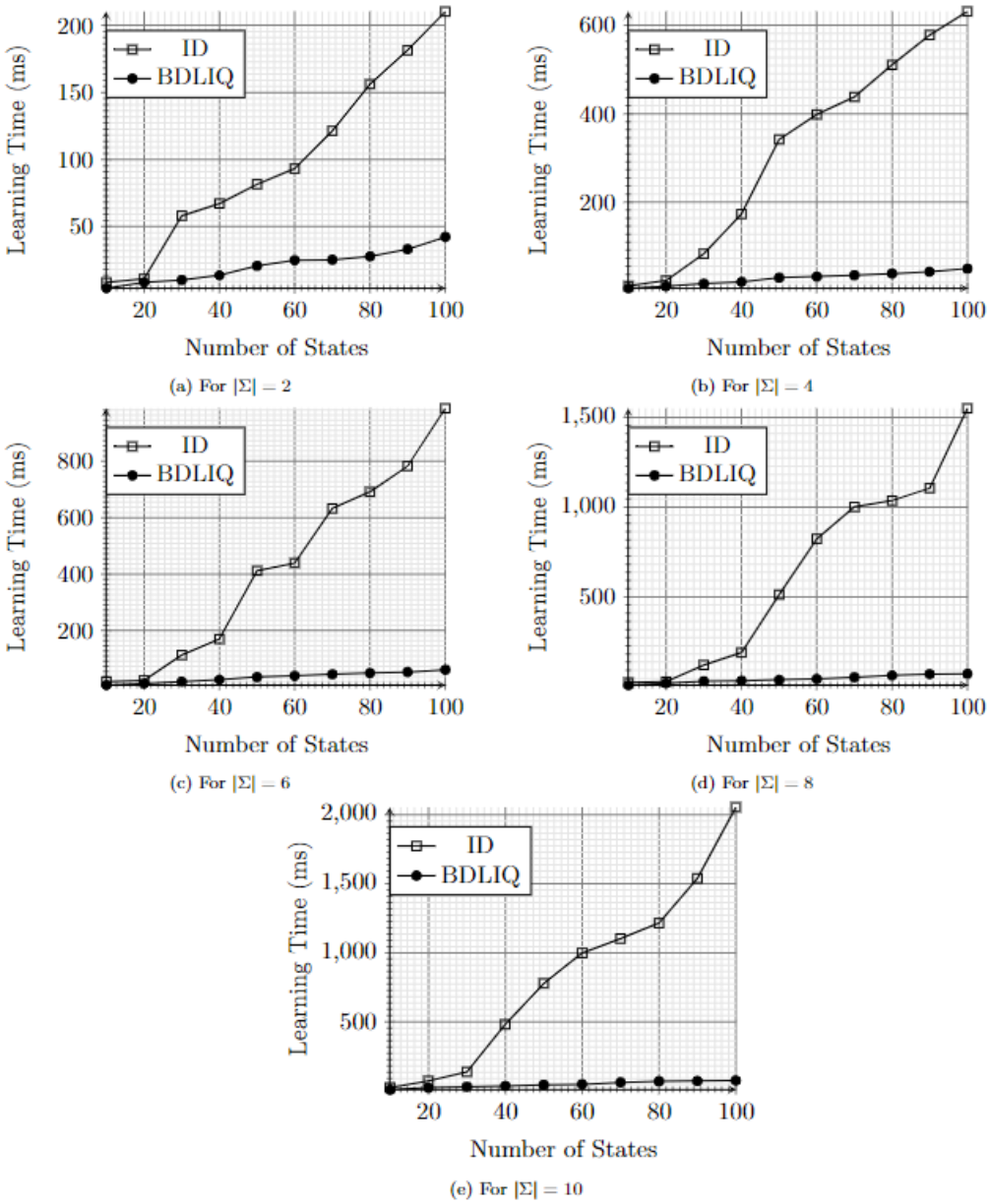

FIGURE 5. Comparison of BDLIQ and ID algorithm with respect to learning time(ms).

used in BDLIQ algorithm, the graphs of the BDLIQ algorithm grow relatively slowly and show logarithmic behavior with increase in number of states or input alphabet size. It follows that in terms of time and the number of queries pose to the MAT, the BDLIQ algorithm is more efficient than the ID algorithm.

1) DISCUSSION

All the existing complete learning algorithms described in Table 1 learn target DFA using δ strategy and possess polynomial time complexity. Our proposed BDLIQ algorithm uses δ^{-1} strategy due to which it learns model from final to start state unlike other learning algorithms. The L^* and ID algorithms are active in nature and uses membership queries for learning purposes whereas RPNI algorithm is passive in nature and does not pose any kind of query. In other hand, the BDLIQ algorithm uses IQs and successfully infer model by posing smaller number of queries than other complete learning algorithms. Therefore, the learning time of the BDLIQ algorithm becomes small and ultimately it possesses worst case time complexity in *nlogn* form.

TABLE 3. Number of queries posed by ID algorithm.

No. of	ID				
States	$\|\Sigma\|$=2	$\|\Sigma\|$=4	$\|\Sigma\|$=6	$\|\Sigma\|$=8	$\|\Sigma\|$=10
10	120.3	280.1	661.7	720.6	800.0
20	160.4	720.7	840.3	1440.3	2200.2
30	660.8	960.4	2340.8	2400.8	3900.5
40	1120.0	3040.1	2880.3	4160.3	7200.1
50	1900.3	7000.0	9300.1	13600.2	19500.1
60	3480.2	7440.2	10080.0	16320.2	22800.1
70	5460.2	11480.2	16389.3	23520.0	30100.2
80	6880.2	13120.6	21221.4	24960.1	32800.1
90	7920.3	16920.4	24840.3	30960.7	42300.1
100	9800.2	21200.3	30600.2	43200.1	55000.3

TABLE 4. Number of queries posed by BDLIQ algorithm.

No. of	BDLIQ				
States	$\|\Sigma\|$=2	$\|\Sigma\|$=4	$\|\Sigma\|$=6	$\|\Sigma\|$=8	$\|\Sigma\|$=10
10	20.4	39.8	58.7	78.6	101.3
20	51.9	100.3	158.3	218.4	271.2
30	87.4	163.2	298.4	351.9	351.4
40	121.3	221.2	291.8	408.3	583.2
50	163.7	331.8	418.2	619.4	721.7
60	211.8	406.3	511.9	751.7	832.3
70	249.3	509.8	603.2	898.1	1007.2
80	291.7	608.9	702.9	1001.9	1241.3
90	354.2	690.8	897.0	1254.3	1299.2
100	411.3	788.2	1009.3	1308.2	1350.0

TABLE 5. Learning time(ms) of ID algorithm.

No. of	ID				
States	$\|\Sigma\|$=2	$\|\Sigma\|$=4	$\|\Sigma\|$=6	$\|\Sigma\|$=8	$\|\Sigma\|$=10
10	8.3	10.1	19.1	22.3	24.8
20	11.0	22.3	23.3	27.6	72.1
30	58.1	83.0	113.0	120.0	138.2
40	67.2	173.0	169.7	189.2	481.5
50	81.6	341.7	411.8	512.3	778.4
60	93.2	398.2	439.1	823.4	998.3
70	121.4	437.9	632.4	1001.0	1102.5
80	156.2	510.2	691.2	1035.6	1214.3
90	181.4	578.0	783.0	1105.5	1538.4
100	210.3	631.2	987.4	1550.0	2053.6

TABLE 6. Learning time(ms) of BDLIQ algorithm.

No. of	BDLIQ				
States	$\|\Sigma\|$=2	$\|\Sigma\|$=4	$\|\Sigma\|$=6	$\|\Sigma\|$=8	$\|\Sigma\|$=10
10	4.1	4.9	6.2	7.9	9.1
20	8.3	9.4	11.9	18.4	23.2
30	10.2	15.3	18.4	29.3	29.8
40	13.7	19.2	25.7	32.1	35.2
50	20.8	28.9	34.9	37.8	43.0
60	24.9	31.2	39.1	42.0	48.7
70	25.3	34.4	44.8	51.9	61.3
80	27.8	38.1	49.3	62.4	70.2
90	33.2	42.1	52.7	69.8	72.0
100	42.3	49.2	60.9	71.3	75.0

IX. CONCLUSION

The existing grammatical inference algorithms have at-least polynomial time complexity due to which these algorithms take a lot of time to infer the required model. In some situations, these algorithms even become fail to learn extremely complicated systems. In this paper, we have developed a revolutionary complete learning algorithm called BDLIQ for learning deterministic finite automata (DFAs), which reduces the complexity of DFA learning from polynomial to *nlogn* form such as $O(|\Sigma|N.logN)$. For this, we have used the delta inverse strategy along with inverse queries. Additionally, we have performed empirical analysis of our proposed algorithm with existing complete learning algorithms possessing similar characteristics like BDLIQ algorithm such as; ID algorithm. We have used two parameters; number of queries posed by the learning algorithm and learning time (ms). Results demonstrated that the BDLIQ outperforms the ID algorithm in terms of number of queries and learning time. By utilising inverse queries, this technique enhances the entire learning process of DFAs. In future, we intend to apply this learning approach to design incremental learning algorithm as well.

REFERENCES

[1] B. K. Aichernig and M. Tappler, "Efficient active automata learning via mutation testing," *J. Automated Reasoning*, vol. 63, pp. 1103–1134, Oct. 2018.
[2] B. K. Aichernig and M. Tappler, "Learning from faults: Mutation testing in active automata learning," in *Proc. NASA Formal Methods Symp.*, 2017, pp. 19–34.
[3] D. Angluin, "A note on the number of queries needed to identify regular languages," *Inf. Control*, vol. 51, no. 1, pp. 76–87, Oct. 1981.
[4] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Aug. 1987.
[5] D. Angluin, "Negative results for equivalence queries," *Mach. Learn.*, vol. 5, no. 2, pp. 121–150, Jun. 1990.
[6] D. Angluin, "Queries revisited," *Theor. Comput. Sci.*, vol. 313, Feb. 2004, Art. no. 175194.
[7] D. Angluin and D. Chen, "Learning a random DFA from uniform strings and state information," in *Proc. Int. Conf. Algorithmic Learn. Theory*, 2015, pp. 119–133.
[8] D. Angulin, "Queries and concept learning," *Mach. Learn.*, vol. 2, pp. 319–342, Apr. 1988.
[9] J. L. Balcázar, J. Díaz, R. Gavaldà, and O. Watanabe, "The query complexity of learning DFA," *New Gener. Comput.*, vol. 12, no. 4, pp. 337–358, Sep. 1994.
[10] L. Becerra-Bonache, A. H. Dediu, and C. Tîrnăucă, "Learning DFA from correction and equivalence queries," in *Proc. Int. Colloq. Grammatical Inference*, 2006, pp. 281–292.
[11] F. Bergadano and S. Varricchio, "Learning behaviors of automata from multiplicity and equivalence queries," *SIAM J. Comput.*, vol. 25, no. 6, pp. 1268–1280, Dec. 1996.
[12] J. Chandlee, J. Fu, K. Karydis, C. Koirala, J. Heinz, and H. Tanner, "Integrating grammatical inference into robotic planning," in *Proc. 7th Int. Conf. Grammatical Inference*, vol. 21, 2012, pp. 69–83.
[13] R. Cleaveland, J. Parrow, and B. Steffen, "A semantics based verification tool for finite state systems," in *Proc. 9th IFIP Symp. Protocol Specification, Test. Verification*. Amsterdam, The Netherlands: North Holland, 1989, pp. 1–26.
[14] R. Cleaveland, J. Parrow, and B. Steffen, "The concurrency workbench," in *Proc. Workshop Autom. Verification Methods Finite State Syst.* (Lecture Notes in Computer Science), 1989, pp. 24–37.
[15] J. P. R. Cleaveland and B. Steffen, "The concurrency workbench: Operating instructions," Lab. Found. Comput. Sci., Univ. Edinburgh, Edinburgh, U.K., Tech. Note 10, Sep. 1988.
[16] J. N. Departarnento and P. Garcia, "Identifying regular languages in polynomial," in *Advances in Structural and Syntactic Pattern Recognition* (Series in Machine Perception and Artificial Intelligence), vol. 5. Singapore: World Scientific, 1992, pp. 99–108.
[17] P. Dupont, "Incremental regular inference," in *Proc. 3rd Int. Colloq. Grammatical Interference Learn. Syntax Sentences (ICGI)*, Montpellier, France. Berlin, Germany: Springer, Sep. 1996 pp. 222–237.
[18] L. Feng, S. Lundmark, K. Meinke, F. Niu, M. A. Sindhu, and P. Y. Wong, "Case studies in learning-based testing," in *Proc. 25th IFIP WG 6.1 Int. Conf. Test. Softw. Syst. (ICTSS)*, Istanbul, Turkey. Berlin, Germany: Springer, Nov. 2013, pp. 164–179.

[19] E. M. Gold "Language identification in the limit," *Inf. Control*, vol. 10, no. 5, pp. 447–474, May 1967. [Online]. Available: http://groups.lis.illinois.edu/amag/langev/paper/gold67limit.html
[20] A. Hagerer, H. Hungar, O. Niese, and B. Steffen, "Model generation by moderated regular extrapolation," in *Proc. 5th Int. Conf. Fundam. Approaches Softw. Eng., Joint Eur. Conf. Theor Pract. Softw. (FASE/ETAPS)*, Grenoble, France. Berlin, Germany: Springer, Apr. 2002, pp. 80–95.
[21] F. Haneef and M. A. Sindhu, "DLIQ: A deterministic finite automaton learning algorithm through inverse queries," *Inf. Technol. Control*, vol. 51, no. 4, pp. 611–624, Dec. 2022.
[22] C. de la Higuera, "Learning grammars and automata with queries," in *Topics in Grammatical Inference*, J. Heinz and J. Sempere, Eds. Berlin, Germany: Springer, 2016, doi: 10.1007/978-3-662-48395-4_3.
[23] M. J. Kearns, U. V. Vazirani, and U. Vazirani, *An Introduction to Computational Learning Theory*. Cambridge, MA, USA: MIT Press, 1994.
[24] A. Khalili, M. Narizzano, L. Natale, and A. Tacchella, "Learning middleware models for verification of distributed control programs," *Robot. Auton. Syst.*, vol. 92, pp. 139–151, Jun. 2017.
[25] H. Khosrowjerdi and K. Meinke, "Learning-based testing for autonomous systems using spatial and temporal requirements," in *Proc. 1st Int. Workshop Mach. Learn. Softw. Eng. Symbiosis*, 2018, pp. 6–15.
[26] E. Kinber, "On learning regular expressions and patterns via membership and correction queries," in *Proc. Int. Colloq. Grammatical Inference*, 2008, pp. 125–138.
[27] M. Krichen, "Improving formal verification and testing techniques for Internet of Things and smart cities," *Mobile Netw. Appl.*, 2019, doi: 10.1007/s11036-019-01369-6.
[28] Y. Li, Y.-F. Chen, L. Zhang, and D. Liu, "A novel learning algorithm for Büchi automata based on family of DFAs and classification trees," in *Int. Conf. Tools Algorithms Construct. Anal. Syst.*, 2017, pp. 208–226.
[29] R. Mazhar and M. A. Sindhu, "DKL: An efficient algorithm for learning deterministic Kripke structures," *Acta Inform.*, vol. 58, no. 6, pp. 611–651, Dec. 2021.
[30] K. Meinke and M. A. Sindhu, "Incremental learning-based testing for reactive systems," in *Proc. 5th Int. Conf. Tests Proofs (TAP)*, Zurich, Switzerland. Berlin, Germany: Springer, Jun. 2011, pp. 134–151.
[31] K. Meinke, "Learning-based testing of cyber-physical systems-of-systems: A platooning study," in *Computer Performance Engineering (EPEW)* (Lecture Notes in Computer Science), vol. 10497. Cham, Switzerland: Springer, 2017, pp. 135–151.
[32] J. Michaliszyn and J. Otop, "Learning infinite-word automata with loop-index queries," *Artif. Intell.*, vol. 307, Jun. 2022, Art. no. 103710.
[33] R. Parekh, C. Nichitiu, and V. Honavar, "A polynomial time incremental algorithm for regular grammar inference," in *Proc. 4th Int. Colloq. Grammatical Inference (ICGI)* (LNAI). Springer, 1998, pp. 1–15.
[34] J. Parrow, "Verifying a CSMA/CD-protocol with CCS," in *Proc. 7th IFIP Symp. Protocol Specification, Test. Verification*, 1987, pp. 1–12.
[35] A. Petrenko, F. Avellaneda, R. Groz, and C. Oriat, "FSM inference and checking sequence construction are two sides of the same coin," *Softw. Quality J.*, vol. 27, pp. 651–674, Dec. 2018.
[36] P. Lamela Seijas, S. Thompson, and M. Á. Francisco, "Model extraction and test generation from JUnit test suites," *Softw. Quality J.*, vol. 26, no. 4, pp. 1519–1552, Dec. 2018.
[37] M. Shahbaz and R. Groz, "Analysis and testing of black-box component-based systems by inferring partial models," *Softw. Test., Verification Rel.*, vol. 24, no. 4, pp. 253–288, Jun. 2014.
[38] M. A. Sindhu and K. Meinke, "IDS: An incremental learning algorithm for finite automata," 2012, *arXiv:1206.2691*.
[39] R. Smetsers, M. Volpato, F. Vaandrager, and S. Verwer, "Bigger is not always better: On the quality of hypotheses in active automata learning," in *Proc. 12th Int. Conf. Grammatical Inference*, 2014, pp. 167–181.
[40] T. Berg, B. Jonsson, M. Leucker, and M. Saksena, "Insights to Angluin's learning," *Electron. Notes Theor. Comput. Sci.*, vol. 118, pp. 3–18, Feb. 2005.
[41] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Reading, MA, USA: Addison-Wesley, 1979.
[42] F. Vaandrager, "Model learning," *Commun. ACM*, vol. 60, no. 2, pp. 86–95, Jan. 2017.
[43] M. Volpato and J. Tretmans, "Active learning of nondeterministic systems from an ioco perspective," in *Proc. Int. Symp. Leveraging Appl. Formal Methods, Verification Validation*, 2014, pp. 220–235.
[44] D. Walker, "Analysing mutual exclusion algorithms using CCS," Univ. Edinburgh, Edinburgh, U.K., Tech. Rep. ECS-LFCS-88-45, 1988.
[45] S. Windmüller, J. Neubauer, B. Steffen, F. Howar, and O. Bauer, "Active continuous quality control," in *Proc. 16th Int. ACM Sigsoft Symp. Component-Based Softw. Eng.*, Jun. 2013, pp. 111–120.

FARAH HANEEF received the B.S. degree in computer science from Arid Agriculture University, Rawalpindi, Pakistan, in 2014, and the M.Phil. degree in computer science from Quaid-i-Azam University, Islamabad, Pakistan, in 2016, where she is currently pursuing the Ph.D. degree in computer science. She has worked as a Research Assistant, from 2020 to 2022. Her research interests include the design and development of algorithms, machine learning, and social network analysis. Up until now, she has authored five research articles in well reputed international journals. She has been awarded Gold Medal from the Arid Agriculture University, Pakistan, in 2014.

MUDDASSAR AZAM SINDHU received the M.Sc. degree in computer science from the University of the Punjab, Lahore, Pakistan, in 2004, and the Licentiate of Engineering (Lic.Eng.) and Ph.D. degrees in computer science from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2011 and 2013, respectively. He is currently an Associate Professor in computer science with Quaid-i-Azam University, Islamabad, Pakistan. He has also designed algorithms for learning deterministic finite automata and Kripke structures. He is also the author of LBTest tool which tests reactive systems in a fully automated manner on the basis of formal linear temporal logic requirements. He has authored around 35 research papers in reputed international journals and conferences. His research interests include automating software testing using formal and informal requirements. He has been a Professional Member of the ACM, since 2015.