Learning a Circuit by Injecting Values

Dana Angluin [a] James Aspnes [a,1] Jiang Chen [b] Yinghua Wu [a,3]

[a] *Department of Computer Science, Yale University.*
[b] *Center for Computational Learning Systems, Columbia University.*

Preprint submitted to Elsevier *7 March 2008*

Abstract

We propose a new model for exact learning of acyclic circuits using experiments in which chosen values may be assigned to an arbitrary subset of wires internal to the circuit, but only the value of the circuit's single output wire may be observed. We give polynomial time algorithms to learn (1) arbitrary circuits with logarithmic depth and constant fan-in and (2) Boolean circuits of constant depth and unbounded fan-in over AND, OR, and NOT gates. Thus, both **AC0** and **NC1** circuits are learnable in polynomial time in this model. Negative results show that some restrictions on depth, fan-in and gate types are necessary: exponentially many experiments are required to learn AND/OR circuits of unbounded depth and fan-in; it is NP-hard to learn AND/OR circuits of unbounded depth and fan-in 2; and it is NP-hard to learn circuits of constant depth and unbounded fan-in over AND, OR, and threshold gates, even when the target circuit is known to contain at most one threshold gate and that threshold gate has threshold 2. We also consider the effect of adding an oracle for behavioral equivalence. In this case there are polynomial-time algorithms to learn arbitrary circuits of constant fan-in and unbounded depth and to learn Boolean circuits with arbitrary fan-in and unbounded depth over AND, OR, and NOT gates. A corollary is that these two classes are PAC-learnable if experiments are available. Finally, we consider an extension of the model called the synchronous model. We show that an even more general class of circuits are learnable in this model. In particular, we are able to learn circuits with cycles.

Key words: Circuit, Learning, Gene regulatory network

1 Introduction

We introduce a new model of active learning for acyclic circuits in which we may inject chosen values on an arbitrary subset of wires but can observe only the value of the circuit's output wire. Our results illuminate the relative importance of manipulation and observation in discovering the structure of networks modeled as circuits.

Gene regulatory networks are an important area in which Boolean network models have been used. In one basic model in this domain, each node in a finite

Email addresses: angluin@cs.yale.edu (Dana Angluin), aspnes@cs.yale.edu (James Aspnes), criver@gmail.com (Jiang Chen), y.wu@yale.edu (Yinghua Wu).

[1] Supported in part by NSF grants CNS-0305258 and CNS-0435201.
[2] Supported in part by a research contract from Consolidated Edison.
[3] Supported by NSF grant CNS-0305258.

network represents a gene, which has a current state of active or inactive. The states of all nodes in the network are updated synchronously; for each node there is a Boolean function giving its new state in terms of the current states of some subset of the other nodes. A key point is that the node states are *fully observable*: it is assumed that gene expression data gives the state of every node in the network at every time step. The discovery problem is to learn the updating functions of all the nodes, in particular, one needs to learn both the set of inputs and the functionality of each node. Of course this is difficult if the updating function may be an arbitrary Boolean function; further assumptions generally restrict the fan-in or types of the possible updating functions. One of the main difficulties in this problem is to discover the topology of the network, that is, which nodes are inputs to which nodes.

Akutsu et al. [1] describe an approach to this discovery problem that models the experimental capability of multiple gene disruption and overexpression. At each time step several selected genes may be disrupted (put in the inactive state), several other selected genes may be overexpressed (put in the active state), while unaffected genes are updated as usual. In this model the states of the nodes are *fully controllable* as well as fully observable. For networks of N nodes and fan-in bounded by k, Akutsu et al. give an $O(N^{2k})$ algorithm for the discovery task. Ideker, Thorsson, and Karp [10] also consider this model and give more practical discovery methods for acyclic networks, using information theoretic criteria to select genes to disrupt or overexpress. These results show that if the class of updating functions is sufficiently restricted, the problem of learning the structure of a network in this model is tractable.

By contrast, there is ample evidence that learning Boolean circuits or formulas from their input-output behaviors may be computationally intractable. Positive learnability results include those for fairly limited classes, including propositional Horn formulas [5] general read once Boolean formulas [6], and decision trees [8], and those for specific distributions, including **AC0** circuits [15], DNF formulas [11] and **AC0** circuits with a limited number of majority gates [12]. (Note that the algorithms in papers [12,15] for learning **AC0** circuits and their variants run only in quasi-polynomial time.) Valiant gives cryptographic evidence for the difficulty of PAC learning general Boolean circuits [19]. Kearns and Valiant [13] show that specific cryptographic assumptions imply that **NC1** circuits and **TC0** circuits are not PAC learnable in polynomial time. These negative results have been strengthened to the setting of PAC learning with membership queries [7], even with respect to the uniform distribution [14].

For these results on learning circuits and formulas, observation and control are both restricted: values on internal wires cannot be observed or manipulated. A natural question is: *What are the relative contributions of full observation and full control to the tractability of learning Boolean networks?*

Our new model addresses this question: we postulate full control and restricted observation. Our results show that the ability to inject values into the circuit gives the learner considerable power, but not as much as would be the case with full observation. In particular, with value injection queries, **NC1** circuits and **AC0** circuits are exactly learnable in polynomial time, but our negative results show that the depth limitations are necessary. However, if behavioral equivalence queries are also available, the depth limitations can be removed, which also implies the polynomial time PAC-learnability of these classes with value injection queries.

In the other direction, Rivest and Sloan [17] propose an interesting model of hierarchical learning of Boolean formulas and acyclic circuits, in which a teacher teaches the circuit one gate at a time to the learner in an order consistent with the graph of the circuit; examples at each stage are drawn from a fixed initial distribution. This increases the observability of the values on internal wires, but not does not provide for their control. Rivest and Sloan give a polynomial time algorithm that successfully learns arbitrary acyclic Boolean circuits in their model.

We also consider an extension of our model called *the synchronous model*, in which we assume that the circuit runs in discrete time and gates are synchronized. The circuits are allowed to have cycles in this extension. In this model, we show that an even larger class of circuits are learnable with experiments only.

2 The Model

2.1 Circuits

We define a variant of the usual circuit model that has no distinguished inputs. This is the convention in gene regulatory network models; it also allows for a more uniform theoretical treatment. In this model, gates with no inputs play the role of input wires: each defaults to a specific constant value, but its output may be overridden by an experiment to any chosen value.

Also, instead of just the Boolean values 0 and 1, we permit a wire to take any value from a finite set Σ, where we assume that $|\Sigma| \geq 2$. This means that the results in this paper apply to models in which gene activations take on a small number of discrete values. Results for models with larger wire alphabets may be found in [3].

A *circuit* C consists of N *wires*, $W = \{w_1, w_2, \ldots, w_N\}$, and for each wire w_i

a *gate* g_i that determines the value on this wire. The *size* of the circuit is N. The wire w_N is assumed to provide the output of the circuit as a whole. A *gate* consists of a function mapping Σ^k to Σ, and a vector of k integers from $[1, N]$ specifying the *input wires* of the gate. The value k is the *fan-in* of the gate. Gates of fan-in zero compute constant functions. The maximum fan-in taken over all the gates in the circuit is the *fan-in* of the circuit.

We define the *circuit graph* to have a node for each wire and its corresponding gate and a directed edge from node i to node j if w_i is one of the input wires to gate j. Until Section 7, we assume that the graph of the circuit is acyclic.

We define the *depth* of the circuit to be the number of edges in the longest simple path to the output in the circuit graph. It should be noted that in this model, a wire is the output of a gate, and a wire may be represented by several edges in the circuit graph.

As an example, we consider a circuit C_0 of 6 wires, as follows.

$$
\begin{aligned}
w_6 &= \mathrm{AND}(w_1, w_3) \\
w_5 &= 0 \\
w_4 &= 1 \\
w_3 &= \mathrm{AND}(w_2, w_4) \\
w_2 &= 1 \\
w_1 &= \mathrm{OR}(w_5, w_2)
\end{aligned}
$$

C_0 is also depicted in Figure 1; note that the single wire w_2 corresponds to the two directed edges (w_2, w_1) and (w_2, w_3) in the circuit graph of C_0.

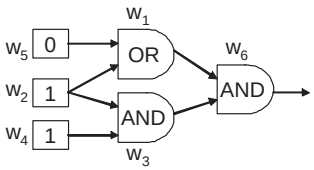

Fig. 1. The circuit C_0; w_6 is the output wire.

2.2 Behavior

We focus on the behavior of a circuit in response to experiments in which we fix the values of certain wires and observe the final output of the circuit. Define an *experiment* to be a vector s in $(\Sigma \cup \{*\})^N$, where s_i specifies the value of w_i (if it is in Σ) or leaves the value of w_i as whatever gate g_i computes (if it is $*$). If $s_i \in \Sigma$, we say w_i is *fixed* in s; otherwise, it is *free* in s. The value

of w_i given s, written $w_i(s)$, is defined as

$$w_i(s) = \begin{cases} g_i(w_{i_1}(s), w_{i_2}(s), \ldots, w_{i_{k_i}}(s)) & \text{if } s_i = *, \\ s_i & \text{if } s_i \neq *. \end{cases} \quad (1)$$

where gate i has function g_i and inputs $(i_1, i_2, \ldots, i_{k_i})$. Gates of fan-in zero, which compute constant functions, give the base cases for the above recursive definition. The output of the circuit given an experiment s is the output of wire w_N, that is, $w_N(s)$; this is also denoted $C(s)$.

Two examples will help clarify this definition; consider again the circuit C_0 defined above. Define the experiment s_0 to leave every wire in C_0 free, that is, $s_0(i) = *$ for $1 \leq i \leq 6$. We compute the values $w_i(s_0)$ as follows: w_2, w_4 and w_5 have no inputs, and take their default values: $w_2(s_0) = 1$, $w_4(s_0) = 1$ and $w_5(s_0) = 0$. Having determined the values of the input wires of w_1 and w_3, we find their values: $w_1(s_0) = \mathrm{OR}(0, 1) = 1$ and $w_3(s_0) = \mathrm{AND}(1, 1) = 1$. Finally, because the values of the inputs to w_6 are determined, we have $C_0(s_0) = w_6(s_0) = \mathrm{AND}(1, 1) = 1$. Thus, when all wires are left free, the circuit C_0 outputs 1. As another example, define the experiment s_1 to fix w_2 to 0 and w_3 to 1 and leave all other wires free, that is, $s_1(2) = 0$, $s_1(3) = 1$ and $s_1(i) = *$ for $i = 1, 4, 5, 6$. Then w_4 and w_5 take their default values, that is, $w_4(s_1) = 1$ and $w_5(s_1) = 0$, but w_2 takes the fixed value 0, so $w_2(s_1) = 0$. The values of the inputs to w_1 are now determined and w_1 is left free, so $w_1(s_1) = \mathrm{OR}(0, 0) = 0$. However, wire w_3 is fixed to 1, so $w_3(s_1) = 1$, regardless of the values of its input wires. Finally, the values of the inputs to w_6 are determined and w_6 is left free, therefore $C_0(s_1) = w_6(s_1) = \mathrm{AND}(0, 1) = 0$. Thus, the output of C_0 for experiment s_1 is 0.

The *behavior* of a circuit is the function mapping experiments s to $C(s)$. Two circuits C and C' are *behaviorally equivalent*, if they have the same behavior, that is, if $\forall s \in (\Sigma \cup \{*\})^N, C(s) = C'(s)$. To compare our work with previous work on learning circuits, we treat the gates of fan-in zero as the *input gates* and denote the number of input gates by n. An experiment is *input-only* if it fixes every input gate and leaves every other gate free. The *input-output behavior* of a circuit C is the restriction of its behavior function to input-only experiments. Clearly behavioral equivalence implies equality of input-output behaviors but not conversely. Behavioral equivalence is more constrained by the internal structure of the circuits, though different structures may have the same behavior (see Figure 2.)

An important special case is *Boolean circuits*, for which $\Sigma = \{0, 1\}$. The input-output behavior of a Boolean circuit is just the usual concept of a circuit computing a Boolean function of n inputs. **NC1** circuits are Boolean circuits with constant fan-in and depth $O(\log n)$. **AC0** circuits are Boolean circuits of constant depth and polynomial size whose gates are unbounded fan-in AND,

OR, and NOT. The *threshold function* Θ_t is the Boolean function that is 1 if and only if at least t of its inputs are 1. **TC0** circuits are Boolean circuits of constant depth and polynomial size whose gates are unbounded fan-in AND, OR, NOT and threshold gates.

2.3 Queries

We assume that the learner can get information about the circuit by specifying an experiment s and observing $C(s)$, the output of the circuit. Such an action is termed a *value injection query*, abbreviated VIQ. We also define a *behavioral equivalence query*, abbreviated BEQ: the learner proposes a circuit C', and, if it is not behaviorally equivalent to the target circuit C, receives in response a *counterexample*, that is, an arbitrarily chosen experiment s such that $C'(s) \neq C(s)$. To be consistent with previous usage, we use the term *membership query*, for a VIQ restricted to an input-only experiment s, although $C(s)$ may be non-binary, and the term *equivalence query*, for a query that tests whether the proposed circuit C' has the same input/output behavior as the target circuit C and returns an arbitrary input-only experiment s witnessing $C'(s) \neq C(s)$ if not. Thus, membership queries and equivalence queries necessarily refer to the input/output behavior of the circuit. An algorithm that learns the (full) behavior of any circuit from a given class using VIQ's and BEQ's also learns the input/output behavior of any circuit from the class using VIQ's and equivalence queries. Then a standard polynomial time transformation yields a PAC learning algorithm using VIQ's for input/output behavior of circuits in the class [2], which implies the following.

Proposition 2.1 *If a class of circuits is learnable in polynomial time with VIQ's and BEQ's, then the input/output behaviors of circuits in the class are PAC-learnable in polynomial time using VIQ's.*

2.4 The Problem

The learning problems we address are: by making VIQ's (respectively, VIQ's and BEQ's) to a target circuit C, find a behaviorally equivalent circuit C'. Both C and C' use gates from a specified class $\mathcal{F}$.

It is not possible to discover the exact structure of the target circuit, even when all wires are relevant. The following example shows that the same behavior may be exhibited by structurally distinct circuits. The three circuits C_1, C_2, and C_3 shown in Figure 2 are behaviorally equivalent, where G_1 and G_2 are arbitrary gate functions. Only when G_2 and V both have value 1 (respectively, 0) can the value of G_1 propagate through the depth 1 AND gate (respectively,

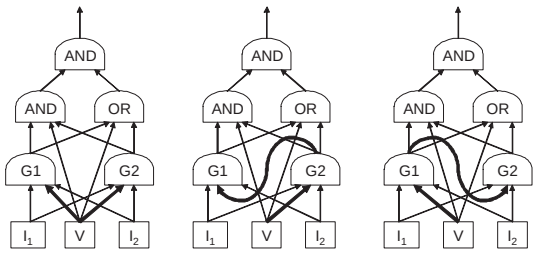

Fig. 2. Three behaviorally equivalent circuits

OR gate). Therefore we cannot decide which one of G_2 and V is an input of G_1, and similarly in the other case.

3 Result summary

We investigate the computational tractability of these problems for classes of circuits defined by restrictions on depth, fan-in, and the class of gate functions $\mathcal{F}$. Our results are summarized in Table 1.

Table 1
Summary of our results for acyclic circuits

Depth	**Fan-in**	**Gates**	**Query types**	**Learnability**	**Reference**
Unbounded	Unbounded	AND/OR	VIQ	$2^{\Omega(N)}$ queries	Theorem 4.1
Unbounded	2	AND/OR	VIQ	NP-hard	Theorem 4.2
Constant	Unbounded	AND/OR/Θ_2	VIQ/BEQ	NP-hard	Theorem 4.5
Logarithmic	Constant	Arbitrary	VIQ	Poly-time	Theorem 5.13
Constant	Unbounded	AND/OR/NOT	VIQ	Poly-time	Theorem 5.15
Unbounded	Constant	Arbitrary	VIQ/BEQ	Poly-time	Theorem 6.1
Unbounded	Unbounded	AND/OR/NOT	VIQ/BEQ	Poly-time	Theorem 6.4

Section 4 contains negative results for exact learning with VIQ's and BEQ's. In Section 5 we give an algorithm, CircuitBuilder, that takes a class of gates $\mathcal{F}$ and a set U of experiments and constructs a circuit C' by making VIQ's on experiments in U and their one-symbol perturbations, and then finding a gate for each wire consistent with the results. If U contains for every wire and every gate that is wrong for that wire a witness experiment that excludes the incorrect gate, then the resulting circuit is behaviorally equivalent to the target circuit. We then show how to construct appropriate sets of experiments U for the class of log-depth constant fan-in circuits and for the class of **AC0** circuits. In Section 6, we extend these methods to use BEQ's as well as VIQ's, and show that the limitations on circuit depth can be removed for both classes.

Finally, in Section 7, we study the synchronous model. We show that any class of circuits with gates that are learnable with membership queries in polynomial time and closed under two natural operations, *projection* and *blurring*, (defined in Section 7) is learnable in this model. This includes any circuits with constant fan-in gates and AND/OR gates with unbounded fan-in. For these results, there is no limitation on circuit depth and we do not restrict circuits to be acyclic.

3.1 Learnability of the gates

What is the relationship between the learnability of circuits in our model and learnability of the class $\mathcal{F}$ of permitted gates? A depth 1 circuit consists of n input gates and one gate g depending on some subset of the inputs; if any nontrivial circuits are to be learnable, then depth 1 circuits must be learnable in the same sense.

For depth 1 circuits, a VIQ reduces to a membership query. Classes $\mathcal{F}$ of gates for which depth 1 circuits are learnable in polynomial time with membership queries include (1) the class of gates with fan-in at most some constant k over an arbitrary finite value set Σ and (2) the class of all symmetric Boolean gates (which includes unbounded fan-in AND, OR, NAND, NOR, threshold and parity gates.)

Another aspect of the learnability of depth 1 circuits is the *consistency problem*. The consistency problem arises when the results of queries rule out certain possible combinations of inputs and outputs for gates; it is defined as follows. The input is a set E of *prohibited pairs* (s, σ) where s is an input-only experiment (recall that an input-only experiment fixes all and only the input gates), $\sigma \in \Sigma$ is a value, and the desired output is a depth 1 circuit C' over $\mathcal{F}$ such that $C'(s) \neq \sigma$ for every pair $(s, \sigma) \in E$. For Boolean circuits, because $C'(s) \neq 0$ implies $C'(s) = 1$ and similarly $C'(s) \neq 1$ implies $C'(s) = 0$, the consistency problem can be stated as finding C' that agrees with given values of experiments in E. The consistency problem is a computational problem and therefore each s should fix all input gates for the problem to be well defined. One of the major differences between the consistency problem and the problem of learning with membership queries is that in the consistency problem no information is provided beyond the set E and hence one may not be able to query Hamming neighbors of an experiment.

For the class of arbitrary gates with fan-in at most k, the consistency problem can be solved in time $O(|E| \cdot n^k)$. For every possible k-set of inputs, we can restrict experiments in E to the set of k inputs by simply ignoring assignments to other inputs. Thus, the restricted version of E contains prohibited values

for settings to the k inputs. The set of k inputs is not consistent with E, if there exists one setting to the k inputs such that the set of prohibited values in E is as large as Σ. (In other words, one cannot find a value for this setting so as to be consistent with E.) Otherwise, for every setting of the k inputs, we set the function value to be any value in Σ that is not prohibited.

There is also a polynomial time algorithm to solve the consistency problem over the class of unbounded fan-in AND, OR, NAND, NOR, NOT and parities. For these binary functions, each prohibited pair determine the function value for the corresponding experiment. To learn AND, take all experiments in E whose function value is 1, and remove all inputs that are ever set 0 in these experiments. The AND of remaining inputs is a function that is consistent with E if it is consistent with *experiments* in E whose function value is 0. Otherwise, there is no AND function consistent with E. AND, OR, NAND, NOR can be learned similarly. NOT is easy. The consistency problem for parities corresponds to a linear system $Ax = b$ in the finite field $GF(2)$, where E corresponds to the matrix A, the function values correspond to b, and the set 1's of in the solution, the 0-1 vector x, corresponds to the set of inputs of the parity function.

However, Lemma 4.4 shows that the consistency problem is NP-hard over the class of unbounded fan-in AND, OR, and thresholds. CircuitBuilder makes use of algorithms for the consistency problem. Lemma 4.3 shows that polynomial time learnability with VIQ's and BEQ's implies a polynomial time algorithm for consistency in certain cases.

3.2 Relation to circuit testing

A central challenge for learning algorithms in our model is to propagate the effects of a changed value on some internal wire to the observable output of the circuit. Our methods are similar in some respects to the idea of *path sensitization* in circuit testing, used to detect whether the output of some gate is "stuck" at a constant value instead of computing the correct value of its inputs [9].

In path sensitization, a path in the correct circuit from a gate g to the output is constructed, and an input x is sought such that the output of g would be the complement of the "stuck" value, and the values of the other inputs to gates along the chosen path are set so as to propagate the output of g (or its complement) to the output of the circuit.

Path sensitization is not a complete method: there are examples of single stuck-at faults in acyclic circuits that cannot be detected by path sensitization, though they are detectable by other tests. In our model, the ability to inject

values on internal wires gives the approach greater power. However, this power does not trivialize the problem; the negative results in Section 4 illustrate the subtle "shadowing" or "filtering" effects that limit the power of VIQ's.

Fujiwara [9] considers the computational problem of deciding, for a given circuit, gate, and stuck-at value, whether there is any test to detect the fault. He shows that the problem is NP-complete, even when restricted to AND/OR circuits of depth three. By contrast, since this class is contained in **AC0**, we show that it is polynomial time learnable using VIQ's.

4 What Cannot be Learned Efficiently?

Figure 3 presents a *gadget*, which is an AND/OR circuit of fan-in 2 that computes $Y = \Theta_2(X, V, W)$. We can view V and W as controlling a switch: only when their values are different will the value of X be passed on to Y. Gadgets can be concatenated to get a gadget chain (see Figure 4), in which the value of X is passed on to Y only when every pair V_i and W_i have different values. The chain can be used to "hide" part of the circuit unless the values of V_i and W_i are complements of each other. In Figure 4, exactly one of each pair V_i and W_i is an input to the big AND gate. The learner has to guess which combination of them are the inputs to the AND gate, which yields the following negative result.

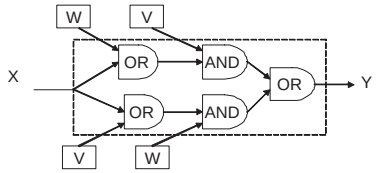

Fig. 3. The gadget

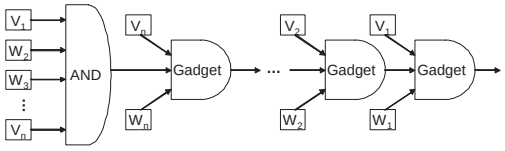

Fig. 4. A hidden AND gate and the gadget chain

Theorem 4.1 *Learning the class of acyclic Boolean AND/OR circuits requires* $2^{\Omega(N)}$ *VIQ's.*

PROOF. Suppose an adversary reveals the gadget chain and the fact that the last gate is an AND gate with exactly one of each pair V_i and W_i as an input, but hides the exact combination. When there exists a pair V_i and W_i that are both 0 or both 1, the output of the circuit is determined by the gadget

chain. In particular, it is determined by the pair with smallest index that are both set 0 and 1 (the pairs are ordered according to their distances to the output gate as in Figure 4). The adversary answers 0 if the pair are set 0 and 1 if the pair are set 1.

Only when for every pair, V_i and W_i are set differently, will the value of the big AND gate affect the output of the circuit. There are 2^n such settings of V's and W's. The adversary answers 0 until only one setting remains. The theorem then follows because $N = O(n)$. $\square$

Theorem 4.1 gives an exponential information-theoretic lower bound, using a deep circuit and a gate of large fan-in. The following construction uses the gadget chain to give a computational hardness result for deep circuits with fan-in 2.

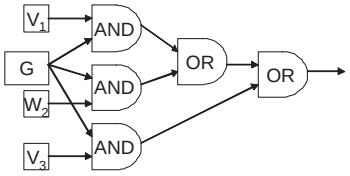

Fig. 5. $G \wedge (x_1 \vee \overline{x_2} \vee x_3)$

Theorem 4.2 *Learning the class of fan-in 2 AND/OR circuits using VIQ's is NP-hard.*

PROOF. We use the implicit negation enforced by the gadget chain to construct a circuit representing a CNF formula, for which a satisfying instance must be found in order to expose a hidden part of the circuit. (Using a NOT gate would not achieve the same effect because we can override its output in an experiment.) We associate a Boolean variable x_i with each pair V_i and W_i, and let $x_i = 1$ if $V_i = 1, W_i = 0$ and 0 if $V_i = 0, W_i = 1$ (we only deal with the case that V_i and W_i are set differently thanks to the gadget chain). Then the circuit in Figure 5 computes $G \wedge (x_1 \vee \overline{x_2} \vee x_3)$. We can chain such clauses by replacing G with the output of the succeeding clause and finally connect the output of the first clause to the gadget chain shown in Figure 4 (note that the big AND gate is not part of the gadget chain). Let g be an AND gate of fan-in 2 with inputs I_1 and I_2. We connect the output of g to the last clause of the clause chain (by replacing G of the last clause by the output of g). In order to learn the circuit, we have to be able to observe the output of g because we are not able to distinguish between this circuit and the circuit with g being replaced by an OR gate if we can not observe g's output. In this construction, the functionality of g matters if and only if there exists an assignment satisfying all the clauses.

As in the proof of Theorem 4.1, when there exists a pair V_i and W_i that are both set to 0 and 1, the output of the circuit is determined by the gadget chain regardless of other parts of the circuit. Suppose every pair of V_i and W_i are set differently. In order to observe g's output, we must compute an assignment to V's and W's that satisfies all clauses in the clause chain, since otherwise the circuit output will simply be 0. In other words, we have to solve the 3-SAT problem. Because the gadget chain consists of AND/OR gates of fan-in 2, the theorem follows. □

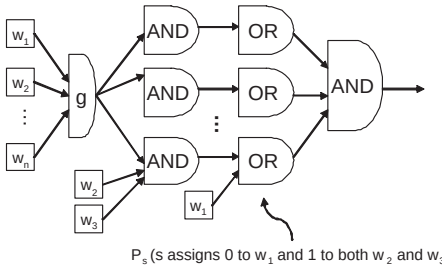

Fig. 6. The "filtering" circuit and an illustration of a filtering path P_s with three input wires.

The gadget chain is a deep circuit, but in the following construction we use AND and OR gates to achieve a constant depth filter that forces a learning algorithm to solve the consistency problem for $\mathcal{F}$ (defined in Section 3.1). Given (1) any depth 1 circuit with input wires $w_1, w_2, \ldots, w_n$ and gate g from a class $\mathcal{F}$ of Boolean gates and (2) a set E of input-only experiments, we add the following structure to construct another circuit C as follows (see Figure 6). For each $s \in E$, which assigns each w_i to 0 or 1, we add a distinct directed path P_s of length 3 consisting of g, a new AND gate, a new OR gate, and the output gate. Let each wire w_i that is set 0 in s be an input of the OR gate of P_s. Let each wire w_i that is set 1 in s be an input of the AND gate of P_s. The construction has the property that if the assignment to $w_1, w_2, \ldots, w_n$ is not s, either the output of the AND gate or the output of the OR gate in P_s is determined, and hence the output of g can not be passed through the path P_s. Therefore, P_s "filters" out all assignments in E other than s. Finally, we take the output gate of the whole circuit to be an AND gate. (Note that we cannot use the same method to replace the gadget chain because it would require $|E|$ to be exponential.) We will call the circuit a *filtering* circuit.

Lemma 4.3 *Any algorithm that learns the class of filtering circuits with input-only experiment set E and function class $\mathcal{F}$ (both E and $\mathcal{F}$ are known to the learner) using VIQ's and BEQ's solves the consistency problem for E and $\mathcal{F}$.*

PROOF. We will show how to use the learning algorithm to solve the consistency problem. Given E and $\mathcal{F}$, we construct the filtering circuit as above

and run the learning algorithm. We answer the algorithm's VIQ's by evaluating the circuit. We need to answer BEQ's only when the learning algorithm proposes a circuit C' with a gate function g' (where the true gate is g) that does not solve the consistency problem. We are done otherwise. When g' does not solve the consistency problem, we will find a counterexample to C' with which to answer the BEQ.

Note that the proposed gate g' and the true gate g correspond to the same wire, denoted by w. Since g' does not solve the consistency problem, there must exist an $s \in E$ such that $g'(s) \neq g(s)$. Suppose w.l.o.g. that $g(s) = 1$ and $g'(s) = 0$. If $C(s) \neq C'(s)$, s is itself a counterexample. Otherwise, $C(s) = C'(s)$. Construct s_0 (s_1) by fixing w to 0 (1) in s. Under experiment s, w takes value 1 in circuit C but takes value 0 in C'. Thus, $C(s) = C(s_1)$ and $C'(s) = C'(s_0)$. Now, we have

$$C'(s_0) = C'(s) = C(s) = C(s_1) \neq C(s_0)$$

The last inequality holds because, by the structure of the filtering circuit C, $C(s_1) = 1 \neq C(s_0) = 0$. $\square$

Lemma 4.4 *The consistency problem is NP-hard for the class of unbounded fan-in AND, OR, and Θ_2 gates.*

PROOF. We reduce a 3-SAT instance ϕ over the variables x_i for $i \in [1, n]$ to the consistency problem for this class. The input wires are

$$\{I_1, I_2, I_3, V_1, W_1, V_2, W_2, \ldots, V_n, W_n\}.$$

Let W denote the output wire of the unknown gate. We define a correspondence between literals of ϕ and wires: literal x_i corresponds to wire V_i, and literal $\overline{x_i}$ corresponds to wire W_i. We design the set of experiments and their outputs as follows, so as to constrain the unknown gate to be a Θ_2 gate with its inputs corresponding to a satisfying assignment of ϕ.

- For each of the eight experiments assigning 0 and 1 to I_1, I_2, and I_3, with all $V_i = W_i = 0$, the output value is $\Theta_2(I_1, I_2, I_3)$. This guarantees that the gate function for W cannot be AND or OR, and must therefore be a Θ_2 gate whose inputs include I_1, I_2, and I_3.
- For each i, on the experiment with $I_1 = V_i = W_i = 1$ and all other input wires assigned 0, the output value is 1. This implies at least one of V_i and W_i is an input of wire W.
- For each i, on the experiment with $V_i = W_i = 1$ and all other input wires assigned 0, the output value is 0. This implies not both V_i and W_i are inputs of wire W.

- For each clause of ϕ, on the experiment that sets I_1 and the three wires corresponding to the three literals in the clause to 1 and all other wires to 0, the output is 1. This ensures that at least one wire corresponding to a literal in the clause is an input of wire W.

It is easily verified that ϕ is satisfiable if and only if there is a gate g for wire W from the specified class of gate functions consistent with these experiment/value pairs. $\square$

Lemma 4.3 and Lemma 4.4 establish the following theorem.

Theorem 4.5 *Learning constant depth AND/OR/Θ_2 circuits with VIQ's and BEQ's is NP-hard.*

5 Learning with experiments

In this section, we give algorithms for arbitrary circuits with logarithmic depth and constant fan-in and Boolean circuits of constant depth and unbounded fan-in over AND, OR and NOT gates. Therefore, we show that both **AC0** and **NC1** circuits are learnable in polynomial time with VIQ's. One of the main issues is to learn a viable set of inputs for each gate. The gates in the circuit that the algorithms output may not have the same set of inputs as the target circuit (see Figure 2).

First we develop some definitions and basic results. A *partial experiment* is a partial function from $[1, N]$ to $\Sigma \cup \{*\}$. Wires in the domain of a partial experiment are specified as fixed (a value in Σ) or free ($*$); wires not in the domain of the experiment are unspecified. Let s be an experiment and τ be a partial experiment. Define $s|_\tau$ to be the experiment obtained by replacing in s the settings of all wires that are specified in τ by the corresponding settings in τ. Let s and t be two experiments. We say that $t \preceq s$ if the set of free wires in t is a subset of the set of free wires in s, and every wire with a fixed value in s has the same fixed value in t. $\preceq$ defines a partial order among experiments. We say $t \prec s$ if $t \preceq s$ and there is at least one free wire in s that is fixed in t. Let s be an experiment with wire w set free. We call $s|_{w=\sigma}$, where $\sigma \in \Sigma$, the (w, σ)-*perturbation* of s. If $C(s) \neq C(s|_{w=\sigma})$, we say s is (w, σ)-*exposing*.

Consider any gate g with inputs $(i_1, i_2, \ldots, i_l)$. We overload g to take an experiment s as an argument. That is, let $g(s) = g(w_{i_1}(s), w_{i_2}(s), \ldots, w_{i_l}(s))$, where $w_i(s)$ is the value of wire w_i on s in the target circuit C. The following useful facts are easily verified.

Proposition 5.1 *Let s and t be two experiments with the output wire set free. If s and t assign the same value to every wire that is either free in s or is an input to a wire that is free in s then $C(s) = C(t)$.*

Proposition 5.2 $C(s) = C(s|_{w=w(s)})$.

This is meaningful only when w is set free in s. In this case, $w(s)$ is the value the corresponding gate computes. The proposition thus says that if we fix w to the value it takes on an experiment s, the circuit output stays the same.

Proposition 5.3 *Let w and u be two wires and suppose there is no path from w to u in the graph of the circuit. Then $u(s) = u(s|_{w=\sigma})$ for any experiment s and $\sigma \in \Sigma$.*

This says that changing the value on wire w cannot affect the value on wire u if there is no path from w to u in the graph of the circuit.

The main task of our learning algorithms is to find a "correct" gate function for each wire. Formally, a gate g is *wrong* for a wire w, if there exists an experiment s that fixes all of g's inputs and is $(w, g(s))$-exposing. We call such an s a *witness* experiment for g and w. Otherwise, we say that g is *correct* for w.

Lemma 5.4 *Let C' be a circuit with the same set of wires as C. If C' is acyclic and every gate of C' is correct for the corresponding wire in C, C' is behaviorally equivalent to C.*

PROOF. Suppose to the contrary that C' is not behaviorally equivalent to C. Let s be a minimal (with respect to the partial order $\preceq$) experiment such that $C'(s) \neq C(s)$. Let w be a free wire in experiment s and g be its corresponding gate in C', chosen so that all g's inputs are fixed in s (such a wire exists because C' is acyclic and the input gates of C' are considered to have fixed inputs because they have no inputs). By Proposition 5.2, we have $C'(s) = C'(s|_{w=g(s)})$. By the minimality of s, we have $C'(s|_{w=g(s)}) = C(s|_{w=g(s)})$, which then implies that $C(s|_{w=g(s)}) \neq C(s)$. This contradicts the fact that g is correct for w. □

5.1 Constructing a circuit

Let $\mathcal{F}$ be a class of gates containing all of the gates in the target circuit C. We describe an important subroutine, *CircuitBuilder* (Algorithm 1), that takes a set of experiments U and constructs an acyclic circuit C' using gates from $\mathcal{F}$. CircuitBuilder builds C' from the bottom up, starting with gates of fan-in

Algorithm 1 CircuitBuilder

INPUT: U and $\mathcal{F}$.
OUTPUT: C'.

Let U_w denote the set of experiments in U with w set free.
$\forall w, \forall s \in U_w, \forall \sigma \in \Sigma$, let V contain the (w, σ)-perturbation of s. Also, let V_w denote the set of experiments in V with w set free.
Make a VIQ on every experiment $s \in U \cup V$.
$C' \leftarrow \emptyset$. $Z \leftarrow W$.
while Z is not empty **do**
  for $w \in Z$ **do**
    if there exists a function $g \in \mathcal{F}$ that depends only on wires in C', such that $\forall s \in U_w, C(s) = C(s|_{w=g(s)})$ **then**
      Add w and g to C' and remove w from Z.
      $\forall s \in U_w \cup V_w$, replace s by $s|_{w=g(s)}$.
      break

zero. At each iteration, CircuitBuilder attempts to add another wire to C' by choosing a gate in $\mathcal{F}$ among those that depend only on wires that are already in C'. This method has the advantage of building an acyclic circuit, which is crucial because the dependence between gates is not always clear, as in Figure 2.

Define U to be a *sufficient set of tests* for C and $\mathcal{F}$ if for every wire w_i in C and every gate $g \in \mathcal{F}$ that is wrong for w_i, U contains at least one witness for g and w_i. In the remainder of this section we prove the following.

Theorem 5.5 *If U is a sufficient set of tests for C and $\mathcal{F}$ then C' is behaviorally equivalent to C, where C' is the circuit constructed by CircuitBuilder. Moreover, CircuitBuilder is non-adaptive.*

In CircuitBuilder, since before we replace s by $s|_{w=g(s)}$, we check whether $C(s) = C(s|_{w=g(s)})$ for s in U_w, we do not need to make queries on the replacing experiments in U. However, we may have to make queries on the perturbations of replacing experiments. This would require the algorithm to make queries adaptively. Instead, in CircuitBuilder, we maintain another set of experiments V which contains all possible perturbations of experiments in U at the beginning of the algorithm. In Lemma 5.7, we will show that after replacement, V still contains all necessary perturbations of experiments in U. In Lemma 5.8, we show that even in V, a replacing experiment will have the same circuit output as the original one. Therefore, we only need to make queries on $U \cup V$ at the beginning of the algorithm. Thus, the algorithm is non-adaptive.

At each iteration of the algorithm, experiments in $U \cup V$ may be replaced. We make the following claims about the replacements.

Lemma 5.6 *At any iteration, for any $s \in U \cup V$, no wire in C' is set free in s.*

PROOF. CircuitBuilder fixes each wire it adds to C'. □

The following lemma says that if $s \in U$ and $t \in V$ are an experiment and perturbation pair, and w is the corresponding wire, they will continue to be such a pair until w is added to C'.

Lemma 5.7 *Consider any iteration, any $w \in Z$ and any $s \in U_w$, and let s_0 be the version of s at the start of the algorithm. For any $\sigma \in \Sigma$, let t_0 be the (w, σ)-perturbation of s_0 at the start of the algorithm and t be the replacement of t_0 at the iteration considered. Then t is a (w, σ)-perturbation of s.*

PROOF. The statement is clearly true at the start of the algorithm. At each subsequent iteration, at most one setting of s and t will be changed. We only need to show that each replacement will replace the same value for s and t. Note that the replaced value is the output of a function that depends on wires that do not include w (w has not been added to C' yet). Since s and t differ only at their settings of w, the function has the same inputs and hence outputs the same values. □

Together with Lemma 5.7, the following lemma shows that although we need to compare the circuit outputs of replacement experiments and their perturbations, we do not need to make any further VIQ's.

Lemma 5.8 *Suppose U is a sufficient set of tests for C and $\mathcal{F}$. At any iteration consider any $s \in U \cup V$ and let s_0 be the version of s at the start of the algorithm. Then we have $C(s) = C(s_0)$.*

If $s \in U$, there is nothing to prove since the algorithm checks the equality before making the replacement. The case that $s \in V$ is a little bit trickier. We prove an even more general lemma, from which the case $s \in V$ follows.

Lemma 5.9 *Suppose U is a sufficient set of tests for C and $\mathcal{F}$. Let g be the function that CircuitBuilder chooses for gate w. Then g is correct for w. That is, for all s with g's input wires fixed, $C(s) = C(s|_{w=g(s)})$.*

PROOF. W.l.o.g., let w be the first wire added to C' for which the statement in the lemma does not hold. That is, g is wrong for w. By the assumption that U is sufficient for C and $\mathcal{F}$, there exists an experiment $s_0 \in U$ at the start of

the algorithm such that s_0 fixes all g's inputs, and $C(s_0) \neq C(s_0|_{w=g(s_0)})$. Let $s \in U$ be the replacement of s_0 at the iteration w is added to C'. We have that $C(s) = C(s_0) \neq C(s_0|_{w=g(s_0)}) = C(s|_{w=g(s_0)})$, by the assumption that w is the first wire violating the condition. Moreover, $g(s) = g(s_0)$ because s_0 and s both fixes all g's input wires and therefore should agree on them. Therefore, we have

$$C(s) \neq C(s|_{w=g(s)})$$

which contradicts the choice of g by CircuitBuilder. □

Lemma 5.9 together with Lemma 5.4 show that if U is a sufficient set, C' is behaviorally equivalent to C. Lemma 5.7 and 5.8 show that all the queries can be made at the beginning of the algorithm, which establishes Theorem 5.5. Lemma 5.6 validates the operation of picking a function g, which amounts to solving the following consistency problem (defined in Section 3.1). Let E be the projection of U_w to C' (note that all wires in C' are fixed) and let the prohibited pairs (t, σ) be those $t \in E$ and $\sigma \in \Sigma$ such that there is an experiment in U_w that agrees with t and is (w, σ)-exposing.

The next lemma shows that the algorithm terminates in N iterations.

Lemma 5.10 *At each iteration, the algorithm adds one wire to C'.*

PROOF. First we observe that there is at least one wire w in Z whose input wires are all contained in C', because the circuit graph of C is acyclic. The true gate of w in C will survive every *if-test* in the algorithm. □

5.2 Test paths

One of the key ideas of our algorithms is to use test paths. A *test path* is an experiment whose free wires are a directed path from some wire w to w_N, through which w is exposed. Let a *side wire* of a test path be a fixed wire that is an input to a gate whose corresponding wire is set free. The meaning of test paths is made clear in the following lemma, which says that using test paths is sufficient.

Lemma 5.11 *Let s^* be a minimal (w, σ)-exposing experiment, where $\sigma \in \Sigma$. Then the free wires in s^* are a directed path in the graph of C, which starts with w and ends with the output wire w_N. (s^* is a test path.)*

PROOF. When $w = w_N$, the directed path is just w_N itself. Suppose the claim is true for any free wire whose corresponding gate has w as an input.

First we claim that only those wires that w can reach (in the underlying digraph) can be free in s^*. This is because wires that w cannot reach take the same values in s^* and the perturbation $s^*|_{w=\sigma}$ (see Proposition 5.3). Thus, we can set them to the corresponding values and the resulting experiment is still (w, σ)-exposing, which contradicts the minimality of s^*.

Let u be a free wire in s^* whose only free input wire is w. u must exist, because the underlying digraph is acyclic. Let $\sigma_0 = w(s^*)$ and $\beta_0 = u(s^*)$ and $\beta = u(s^*|_{w=\sigma})$. We claim that $s^*|_{w=\sigma_0}$ is a minimal (u, β)-exposing experiment. Let us view the circuit as a function of the values of w and u. That is, let $F(x, y) = C(s^*|_{w=x,u=y})$. By the assumption, we have $F(\sigma_0, \beta_0) \neq F(\sigma, \beta)$. By the minimality of s^*, we have $F(\sigma_0, \beta) = F(\sigma, \beta)$. Thus, we have

$$F(\sigma_0, \beta) \neq F(\sigma_0, \beta_0)$$

which implies that $s^*|_{w=\sigma_0}$ is (u, β)-exposing.

Now, we need to show that $s^*|_{w=\sigma_0}$ is a minimal (u, β)-exposing experiment. Suppose on the contrary that there exists $s' \prec s^*|_{w=\sigma_0}$ that is (u, β)-exposing. We set w free in s' and let the resulting experiment be s''. Thus, $s'' \prec s^*$. Let $F''(x, y) = C(s''|_{w=x,u=y})$. Again, by the assumption and the minimality of s^*, we have

$$F''(\sigma_0, \beta_0) \neq F''(\sigma_0, \beta) = F''(\sigma, \beta)$$

Therefore, we conclude s'' is (w, σ)-exposing by observing that $w(s'') = \sigma_0$, $u(s'') = \beta_0$ and $u(s''|_{w=\sigma}) = \beta$, which contradicts the minimality of s^*.

Therefore, we conclude that $s^*|_{w=\sigma_0}$ is a minimal (u, β)-exposing experiment. By induction, its free wires consist of a directed path starting with u and ending with w_N. We append w to this path to obtain the directed path in s^*. □

5.3 Learning log depth circuits with constant fan-in

We use CircuitBuilder to give an algorithm that learns an arbitrary log depth, constant bounded fan-in circuit. The algorithm does not perform any additional queries and hence is non-adaptive. The main idea is based on the observation that in an acyclic circuit of depth d and fan-in k, a test path has at most d free wires and at most kd side wires. There are at most $|\Sigma|^{O(kd)}$ settings of these wires. If we randomly assign one of the symbols of $\Sigma \cup \{*\}$ to each wire with equal probability, the probability that we generate one of the settings is $1/|\Sigma|^{O(kd)}$. We can generate all settings using $|\Sigma|^{O(kd)} \log \frac{1}{\delta}$ random experiments, which succeeds with probability at least $1 - \delta$. We can also generate them deterministically using a universal set construction. The following definition of universal set is adapted from Seroussi and Bshouty [18].

An experiment set U is called (N, l)-*universal* if for every set of indices $R = \{r_1, r_2, \ldots, r_l\} \subseteq [N]$, the projection of U to R contains all $(|\Sigma| + 1)^l$ l-tuples. It is shown in [16] that a (N, l)-universal set of size $2^{O(l \log |\Sigma|)} \log N$ can be constructed in polynomial time. [4]

Lemma 5.12 *Let C be a circuit of depth d and fan-in k, $\mathcal{F}$ be the class of all gates of fan-in at most k, and U be an $(N, (d+1)(k+1))$-universal set. U is a sufficient set for C and $\mathcal{F}$.*

PROOF. Let s be a witness experiment that g is wrong for w; all g's inputs are fixed in s. Let $s^* \preceq s$ be a minimal $(w, g(s))$-exposing experiment. According to Lemma 5.11, there are at most d free wires and dk side wires in s. Since U is a universal set, there exists an experiment $s_0 \in U$ at the beginning of CircuitBuilder, such that s_0 agrees with s^* in all s^*'s free wires and side wires and also all g's inputs (there are at most $(d+1)(k+1)$ wires). Proposition 5.1 shows that s_0 is a witness experiment that g is wrong for w. □

Whenever $kd = O(\log N)$, the size of U is polynomial in N and so is that of V. We reach the following theorem.

Theorem 5.13 *Log depth, constant bounded fan-in circuits can be learned non-adaptively in polynomial time using VIQ's.*

PROOF. Combining Theorem 5.5 and Lemma 5.12, we can learn a circuit of depth d and fan-in k, by applying CircuitBuilder with U being an $(N, (d+1)(k+1))$-universal set. When k is $O(1)$ and d is $O(\log N)$, the query complexity $2^{O(kd \log |\Sigma|)}$ is polynomial. The time complexity depends mainly on the complexity of the consistency problem, which is polynomial for constant fan-in circuits (see Section 3.1). The algorithm is non-adaptive since CircuitBuilder is non-adaptive. □

*5.4 Learning **AC0** circuits*

Theorem 4.5 precludes polynomial time algorithms for learning constant depth unbounded fan-in circuits with fairly simple gates. In this section, we show that if we allow only AND and OR gates (it is easy to extend it to NAND, NOR

[4] The paper [16] only deals with binary vectors. But it can be easily extended to the non-binary case by viewing each non-binary literal in $\Sigma \cup \{*\}$ as a binary vector of size $\lceil \log(|\Sigma| + 1) \rceil$.

and NOT), constant depth unbounded fan-in Boolean circuits are learnable. Thus we show that **AC0** circuits are learnable with VIQ's.

We are not able to use a universal set, since k can be as large as $\Omega(N)$. Instead, we use Algorithm 2 to gather the necessary test paths adaptively. Algorithm 2 begins with learning the output gate, which can be easily done for AND and OR gates. It then sets one of its input wires free and fixes the other input wires so that the free input wire is still relevant. In particular, it sets the other input wires to 1 if the output gate is an AND gate, or 0 if the output gate is an OR gate. This partial experiment is then used to find (some of) the inputs of the corresponding gate. The algorithm goes on exploring the whole circuit. Algorithm 2 alone is not sufficient, because some input wires may be fixed as side wires and therefore hidden to the learner.

Algorithm 2 Gathering test paths for a constant depth AND/OR circuit

Let Γ contain the partial experiment that sets the output wire free, $\{w_N = *\}$. $\{\Gamma$ *is the agenda.*$\}$
Let $\mathbf{1}$ *(*$\mathbf{0}$*) be an experiment that sets all wires to* 1 *(*0*),*
while Γ *is not empty* ***do***
  Pick $\tau \in \Gamma$ *and remove it from* Γ.
  if $C(\mathbf{1}|_\tau) \neq C(\mathbf{0}|_\tau)$ ***then***
    Let $Z = \{w \mid$ *w is unspecified in* τ*, and* $C(\mathbf{1}|_{\tau,w=0}) \neq C(\mathbf{1}|_\tau)$ *or* $C(\mathbf{0}|_{\tau,w=1}) \neq C(\mathbf{0}|_\tau)$ $\}$.
    for $w \in Z$ ***do***
      if $C(\mathbf{1}|_{\tau,w=0}) \neq C(\mathbf{1}|_\tau)$ ***then***
        Add $\tau|_{w=*,\forall w' \in Z\setminus\{w\},w'=1}$ *to* Γ. $\{$AND gate.$\}$
      else if $C(\mathbf{0}|_{\tau,w=1}) \neq C(\mathbf{0}|_\tau)$ ***then***
        Add $\tau|_{w=*,\forall w' \in Z\setminus\{w\},w'=0}$ *to* Γ. $\{$OR gate.$\}$

The following lemma shows the correctness of the learning algorithm.

Lemma 5.14 *Let* U *contain all tests that are made in Algorithm 2 with target circuit* C *and* $\mathcal{F}$ *be all AND and OR gates.* U *is sufficient for* C *and* $\mathcal{F}$.

PROOF. Suppose s is a witness experiment that g is wrong for w and $s^* \preceq s$ is a minimal $(w, g(s))$-exposing experiment. Let u be the successor of w in the directed path from w to w_N (Lemma 5.11). We define two partial experiments τ_u and τ_w. τ_u sets all free wires in the directed path before u and their side wires as in s^* and sets u free. τ_w is similarly defined. We claim that τ_w is added to Γ in Algorithm 2. We assume inductively τ_u has been added to Γ.

Compare τ_u and τ_w. Those wires unspecified by τ_u but specified by τ_w are side wires that are inputs only to u. They are set to 1 in τ_w if the corresponding gate of u is an AND gate and 0 if the corresponding gate of u is an OR gate so as to keep w relevant. Furthermore, we observe that

(1) If the corresponding gate of u is an AND gate, $C(\mathbf{1}|_{\tau_u}) \neq C(\mathbf{0}|_{\tau_u})$ and $C(\mathbf{1}|_{\tau_u,w=0}) \neq C(\mathbf{1}|_{\tau_u})$;
(2) If the corresponding gate of u is an OR gate, $C(\mathbf{1}|_{\tau_u}) \neq C(\mathbf{0}|_{\tau_u})$ and $C(\mathbf{0}|_{\tau_u,w=1}) \neq C(\mathbf{0}|_{\tau_u})$.

Therefore, τ_w must be added. Thus U must contain the following experiments $\mathbf{0}|_{\tau_w}, \mathbf{1}|_{\tau_w}$, and for all w' unspecified in τ_w, $\mathbf{1}|_{\tau_w,w'=0}$, and $\mathbf{0}|_{\tau_w,w'=1}$.

Let g^* be the gate for w in the target circuit C. The two projected functions $g|_{\tau_w}$ and $g^*|_{\tau_w}$ (fixing some inputs of the functions) must be different, because otherwise it contradicts the fact that s^* is a witness experiment. Since g and g^* are AND, OR or constant gates, their projections $g|_{\tau_w}$ and $g^*|_{\tau_w}$ can be AND, OR and constant gates.

We show, by the following case analysis, that we can find an experiment s_0 in aforementioned experiments such that $g|_{\tau_w}(s_0) \neq g^*|_{\tau_w}(s_0)$. By the way that τ_w is constructed, we know that $C(s_0|_{w=0}) \neq C(s_0|_{w=1})$, that is, the difference in w is reflected in the circuit output, and thereby s_0 is a witness experiment that g is wrong for w.

(1) $g|_{\tau_w}$ and $g^*|_{\tau_w}$ are both constant gates, their constant outputs must be different. Any of the experiments is good for us.
(2) One of them is a constant gate and the other is not. The non-constant gate must have different outputs for inputs $\mathbf{0}|_{\tau_w}$ and $\mathbf{1}|_{\tau_w}$. Therefore, one of $\mathbf{0}|_{\tau_w}$ and $\mathbf{1}|_{\tau_w}$ serves the purpose of s_0.
(3) Both of them are AND gates. The two AND gates must have different relevant inputs. Let w' be a relevant input to one of the two gates but not to the other. The two gates must have different outputs on $\mathbf{1}|_{\tau_w,w'=0}$.
(4) Both of them are OR gates. This case is similar to the previous one. Only this time $\mathbf{0}|_{\tau_w,w'=1}$ serves the purpose of s_0.
(5) One of the two gates is an AND gate and the other an OR gate. If their relevant input sets are the same, both $\mathbf{1}|_{\tau_w,w'=0}$, and $\mathbf{0}|_{\tau_w,w'=1}$, where w' is a relevant input to both gates, show their difference. If their relevant inputs sets are different, either we can find a w' relevant to the AND gate but not to the OR gate, or we can find a w' relevant to the OR gate but not to the AND gate. In the former case, $\mathbf{1}|_{\tau_w,w'=0}$ is what we look for; in the latter case, $\mathbf{0}|_{\tau_w,w'=1}$ is what we look for. □

It is clear that each partial experiment collected by Algorithm 2 corresponds to a directed path in the circuit C. Thus the number of partial experiments is bounded by $O(N^d) = poly(N)$ when the depth d is a constant. The size of U and the number of tests are hence polynomially bounded. The theorem then follows from the fact that the consistency problem for AND/OR gates can be solved in polynomial time.

Theorem 5.15 *Constant depth, unbounded fan-in AND/OR circuits are learnable in polynomial time using VIQ's.*

6 Learning with experiments and counterexamples

BEQ's overcome the obstacles of Theorem 4.1 and Theorem 4.2, because the counterexample has to give away the combination when an appropriate hypothesis circuit is presented. However, the result in Theorem 4.5 still applies. Assuming both VIQ's and BEQ's are available, we give polynomial time algorithms to learn both arbitrary constant fan-in circuits and AND/OR circuits with unbounded depth.

Both algorithms repeatedly make a BEQ on a candidate circuit C' until C' is behaviorally equivalent to the target circuit. Each counterexample s is processed to give a minimal counterexample $s^* \preceq s$ such that $C'(s^*) \neq C(s^*)$. This process, *Minimize*, can easily be done with $O(N)$ VIQ's. The minimal counterexample is then used in rebuilding the candidate circuit C'. As in the proof of Lemma 5.4, a minimal counterexample is a witness experiment that a candidate gate g is wrong for a wire w. Therefore, each counterexample eliminates at least one candidate gate for at least one wire. For constant fan-in circuits, this immediately leads to a polynomial-time learning algorithm, since there are at most $|\Sigma|^{|\Sigma|^k}\binom{N}{k}$ candidate gates to eliminate (for each of the $\binom{N}{k}$ combinations of k inputs, there are $|\Sigma|^{|\Sigma|^k}$ candidate functions). In fact, we will receive at most $|\Sigma| \cdot |\Sigma|^k\binom{N}{k}$ counterexamples for each wire as shown in the following theorem.

Theorem 6.1 *Bounded fan-in circuits are learnable in polynomial time using VIQ's and BEQ's.*

PROOF. The algorithm is described in the previous paragraph, except that we haven't specified how to build the candidate circuit C'. We will use CircuitBuilder, but instead of checking each gate with respect to U, we just pick a gate that is not eliminated for the corresponding wire. For each of the $\binom{N}{k}$ combinations of k inputs, any for each of the $|\Sigma|^k$ settings to the k inputs, there are at most $|\Sigma|$ many possible counterexamples, each of which rules out one possible output. Therefore, there are at most $|\Sigma| \cdot |\Sigma|^k N^k$ counterexamples for each wire. Thus, the algorithm will receive at most $(|\Sigma|N)^{k+1}$ counterexamples in total, which is polynomial when k is a constant. Because we use CircuitBuilder, it is not hard to see that the number of $VIQ's$ and the total running time are also polynomial. □

However, the same method does not work for AND/OR circuits, because $|\mathcal{F}|$ is exponential. But each minimal counterexample will still help us learn the gate function of an individual wire. Let us denote each counterexample by a pair indicating the outputs of the true gate and the proposed gate. For example, in a $(1, 0)$ counterexample for wire w, the true gate of w outputs 1 but the proposed gate outputs 0. A $(0, 1)$ counterexample is the opposite. It is not hard to see that a $(1, 0)$ counterexample eliminates the constant 0 gate for the wire and similarly a $(0, 1)$ counterexample eliminates the constant 1 gate.

Now we will see how counterexamples help to learn AND/OR gates. There are 3 cases in terms of the proposed gate and the true gate: both are AND; both are OR; one is AND and the other is OR. In all three cases, counterexamples can be divided into two types, namely, *input removing* and *input demanding* counterexamples. The first two cases are similar and we will start with them.

Assume that the proposed gate and the true gate are both AND gates. In the following, let w be the wire that receives the counterexamples. A $(1, 0)$ counterexample says that the *0-inputs* (inputs that are set 0) of the proposed gate are not inputs of the true gate of w, and thus should be removed from the set of potential inputs. Therefore, in this case, a $(1, 0)$ counterexample is an *input removing* counterexample. Let $R_w^\wedge$ contain all inputs removed by input removing counterexamples for wire w.

On the other hand, a $(0, 1)$ counterexample implies that the inputs of the proposed gate do not include all inputs of the true gate of w. Thus, a $(0, 1)$ counterexample demands that the learner include more inputs for wire w, and hence is *input demanding* in this case. (As in CircuitBuilder, to avoid building a cyclic circuit, we cannot learn each wire/gate as a function of all other $N - 1$ wires at the same time. Therefore, we use only some of the other wires as inputs for each wire/gate.) Let the set of inputs of the proposed gate be T. An input demanding counterexample says that any AND gate whose inputs are completely contained in T cannot be the true gate of w. Let $T_w^\wedge$ be a collection of sets like T. That is, whenever an input demanding counterexample is received, we will add the set of inputs of the proposed gate to $T_w^\wedge$. $T_w^\wedge$ will serve as constraints on candidate gates for wire w.

Similar arguments can be made when the proposed gate and the true gate are both OR. However, in this case, a $(0, 1)$ counterexample is *input removing* while a $(1, 0)$ counterexample is *input demanding*. Let $R_w^\vee$ be analogous to $R_w^\wedge$ and $T_w^\vee$ be analogous to $T_w^\wedge$.

When the true gate and the proposed gate are of different types, we will reduce them to the first two cases. If the proposed gate is AND, but the true gate of w is OR, we process the counterexamples as if the true gate is AND (there is no way to tell anyway). Thus, we will add 0-inputs of the proposed gate to $R_w^\wedge$

upon receiving a $(1,0)$ counterexample, and add the whole set of inputs of the proposed gate to $T_w^\wedge$ upon receiving a $(0,1)$ counterexample. The important fact is that although we attempt to learn w as AND, *the true gate, the OR gate will never be eliminated.* The constraints imposed by $R_w^\wedge$ and $T_w^\wedge$ are valid constraints only on candidate AND gates. Since we learn w as an OR at the same time, we might either figure out that it cannot be an AND or find an OR gate that is correct. Similar arguments can be made when the proposed gate is OR, but the true gate of w is AND.

Now we are ready to give our algorithm for learning AND/OR circuits. (NAND gates can be dealt with in a way similar to AND gates. NOR gates can be dealt in a way similar to OR gates.) The overall algorithm runs in the same cycle of proposing a circuit, receiving and processing a counterexample, and then proposing a new circuit. Each counterexample is processed by *Minimize* first and then used either to eliminate a constant gate from $\mathcal{K}_w$, which contains a constant 1 gate and a constant 0 gate at the beginning of the algorithm, or to update $R_w^\wedge$ and $T_w^\wedge$ or $R_w^\vee$ and $T_w^\vee$, which are all empty at the beginning of the algorithm, depending on the type of proposed gate function for wire w. To build the proposed circuit, we use Algorithm 3.

Algorithm 3 Building the proposed circuit

INPUT: $\forall w \in W$, $R_w^\wedge, T_w^\wedge, R_w^\vee, T_w^\vee$ and the set of constant functions $\mathcal{K}_w$ that have not yet been eliminated.

OUTPUT: C'.

```
1: C' ← ∅. Z ← (w_1, w_2, ..., w_N).
2: while Z is not empty do
3:    Pop the first wire w in Z.
4:    if K_w ≠ ∅ then
5:       Add w to C' with any function in K_w.
6:    else if ∀T ∈ T_w^∧, C'\R_w^∧ ⊈ T then
7:       Add w to C' with AND of wires in C'\R_w^∧.
8:    else if ∀T ∈ T_w^∨, C'\R_w^∨ ⊈ T then
9:       Add w to C' with OR of wires in C'\R_w^∨.
10:   else
11:      Put w at the end of Z.
```

At each iteration, we try to add a wire w in Z to C'. As in CircuitBuilder, we try to learn w as a function of wires that have already been added to C'. We organize Z as a queue and let the initial order be $w_1, w_2, \ldots, w_N$. At each iteration, the first wire in Z will be considered. If it is not added to C', it will be put at the end. We add the wire to C' if and only if one of the following is true. One of the two constant functions is not eliminated, or in other words, $\mathcal{K}_w$ is not empty; $C' \backslash R_w^\wedge$ is not contained in any set in $T_w^\wedge$, and hence AND of $C' \backslash R_w^\wedge$ does not violate any counterexample; $C' \backslash R_w^\vee$ is not contained in any set in $T_w^\vee$, and hence OR of $C' \backslash R_w^\vee$ does not violate any counterexample. We add the wire to C' with a constant function, AND of wires in $C' \backslash R_w^\wedge$, or OR

of wires in $C' \backslash R_w^\vee$, respectively.

Now we bound the number of counterexamples each wire can receive. There are at most 2 counterexamples that eliminate constant functions. There are at most $O(N)$ input removing counterexamples. The most subtle case is input demanding counterexamples. We identify the phase number of the learning algorithm with the number of counterexamples it has received. Algorithm 3 is called to rebuild the circuit at each phase. In the process of Algorithm 3, let *the round number of an iteration* be the number of times the wire being considered has been popped from Z. Let $I_w(t)$ be the round number of the iteration that w is finally added to C' at phase t. We will show that $I_w(t)$ will never decrease in the following. The intuition is that we add more constraints on learning w as we receive more counterexamples.

Let $C'_{(w,i)}(t)$ be the set of wires in C' when w is considered at round i in phase t. If we order pairs in $W \times [1, N]$ first by their round number and then by the order of wires in W, we have that $C'_{(w,i)}(t) = \{w' | (w', I_{w'}(t)) \leq (w, i)\}$. Together with R_w's and T_w's, $C'_{(w,i)}(t)$ decides whether w can be added to the circuit at round i in phase t. In the following, we show that $C'_{(w,i)}(t)$ never gets bigger as t grows. The key observations are that if a set C' cannot pass the test

$$\forall T \in T_w^\wedge, C' \backslash R_w^\wedge \nsubseteq T,$$

(1) no subset of C' can pass the test;
(2) C' cannot pass the test if we add more wires to $R_w^\wedge$ or more sets to $T_w^\wedge$.

The same statements can be said about C', $R_w^\vee$ and $T_w^\vee$.

Lemma 6.2 *$C'_{(w,i)}(t+1) \subseteq C'_{(w,i)}(t)$. (To avoid triviality, we define $C'_{(w,i)}(t) = \emptyset$, if w is added to C' before round i at phase t, or in other words, there is no round i for w at phase t)*

PROOF. We do induction on the pairs (w, i). The lemma clearly holds for $(w_1, 1)$ because $C'_{(w_1,1)}(t)$ is always empty; w_1 is the first wire considered each time Algorithm 3 runs.

Suppose it holds for all pairs that precede (w, i). Suppose there exists a wire w' in $C'_{(w,i)}(t+1) \backslash C'_{(w,i)}(t)$. We have that $(w', j = I_{w'}(t+1)) \leq (w, i)$. Therefore, w' is added at the j^{th} round at phase $t+1$ but after the j^{th} round at phase t. In other words, $C'_{(w',j)}(t)$ fails the test with $R_{w'}^\wedge$ and $T_{w'}^\wedge$ (or $R_{w'}^\vee$ and $T_{w'}^\vee$) at round j, while $C'_{(w',j)}(t+1)$ succeeds. By our inductive assumption, $C'_{(w',j)}(t+1) \subseteq C'_{(w',j)}(t)$. This contradicts observations (1) and (2). $\square$

It follows again by observations (1) and (2), and the same observations on C',

$R_w^\vee$ and $T_w^\vee$ that

Corollary 6.3 *$I_w(t)$ is non-decreasing as t grows.*

Now let us bound $I_w(t)$. Recall that the true gate will never be eliminated. Therefore, whenever C' contains all inputs of the true gate, w will be added. Thus, if the true gate of w is a constant gate, $I_w(t) = 1$. If the true gate of w depends only on constant gates, $I_w(t) \leq 2$. In general, we have that $I_w(t) \leq \max(I_{w_{i_1}}(t), I_{w_{i_2}}(t), \ldots, I_{w_{i_k}}(t)) + 1$, where $w_{i_j} (j \in [1, k])$ is an input of a true gate of w. Therefore, $I_w(t) \leq d$.

Each input demanding counterexample for wire w at phase $t+1$ will eliminate the gate that Algorithm 3 picked for w at round $I_w(t)$ at phase t, by adding all inputs of the gate to $T_w^\wedge$ or $T_w^\vee$ as constraints. By Lemma 6.2 and observations (1) and (2), this means that at phase $t + 1$, we cannot pick gates of the same type again for w at or before round $I_w(t)$. Thus $I_w(t)$ will increase when w receives at most 2 input demanding counterexamples, one for AND gates and the other for OR gates. Therefore, we can bound the number of input demanding counterexamples by $O(d)$ per wire. Thus we can bound the total number of $BEQ's$ by a polynomial in N and it follows that the total number of $VIQ's$ and the total running time are polynomial as well. We reach the main theorem.

Theorem 6.4 *AND/OR circuits with unbounded fan-in and unbounded depth are learnable in polynomial time using VIQ's and BEQ's.*

7 Learning synchronous circuits with cycles

In this section, we study a new learning model called *the synchronous model*, where time is quantized and we can inject values as well as observe the output of the circuit at each time step. The value of each wire at time $t + 1$ is determined either by the injected value at time $t+1$ or by its inputs at time t if it is set free. An *experiment sequence* is a series of experiments. We require the first experiment in an experiment sequence to fix all wires for the sake of completeness. Two circuits are *equivalent* in the synchronous model, if and only if they have the same set of wires, and produce the same output sequence given the same experiment sequence.

In this section, circuits are allowed to have cycles. Although cycles incur conflicts in the previous circuit model, they do not cause the same cyclic dependence between gates in the synchronous model, because the computation of gates is timed and each gate depends only on the status of its input gates in the previous time step. In fact, for a finite experiment sequence, the circuit

in the synchronous model works like an acyclic circuit in the previous model. For an experiment sequence of length m, we can build an acyclic circuit of m layers, each corresponding to a time step, by making one copy of each wire at each layer. An experiment sequence of length m then corresponds to an experiment for the new acyclic circuit. In this circuit, a copy of a wire at layer $t+1$ depends on the copies of its input wires at layer t and has the same gate function as its original, except for the wires at the first layer, which are always fixed by an experiment sequence. The following diagram illustrates a cyclic circuit in the synchronous model and its corresponding acyclic circuit of 3 layers.

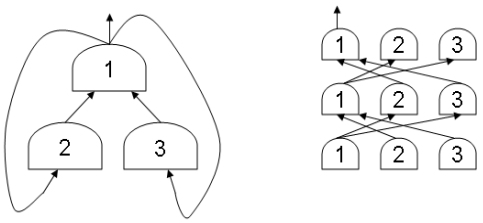

Fig. 7. A cyclic circuit and its corresponding 3-layer acyclic circuit.

The ability to inject values at each time step in the synchronous model gives us more power to learn. To some extent, it allows us to isolate and examine each individual gate as we will see later in this section. We will show that any circuit over a large class of gates, including constant fan-in gates and AND/OR gates, are learnable in the synchronous model.

Let us first define two operations on gate functions. Let $g(u_1, u_2, \ldots, u_k)$ be a function from Σ^k to Σ. For all $\sigma \in \Sigma, i \in [1,k]$, let $g|_{u_i=\sigma} = g(u_1, \ldots, u_i = \sigma, \ldots, u_k)$ be a *projection* of g. Let $P \subseteq 2^\Sigma$ be a partition of Σ. Let p from Σ to P be the corresponding partition function, i.e., p maps a symbol to the set that contains it in P. Let $p(g(u_1, u_2, \ldots, u_k))$ be a *blurring* of g. A blurring of g is only interesting when $|\Sigma| > 2$, since when $|\Sigma| = 2$, the only possible blurring of g is a constant function.

The idea of blurring is natural in our setting, because blurred versions of gates are all that the learner observes, that is, the learner is able to observe the difference between two values α and β for a wire w only when they make a difference to the circuit output. On the other hand, the difference between α and β is only relevant when it makes a difference to the circuit outputs. Therefore, for a gate and a setting of its inputs, the learner is only able to and only needs to identify a set of values, any of which can be used as the function value.

Let $\mathcal{G}$ be the collection of all classes $\mathcal{F}$ of gates that are closed under projection and blurring and learnable with membership queries in polynomial time. The main theorem of this section is the following.

Theorem 7.1 *Circuits (which may have cycles) with gates from a class $\mathcal{F}$ in $\mathcal{G}$ are learnable in polynomial time in the synchronous model.*

The classes of constant fan-in gates and AND/OR gates each belong to $\mathcal{G}$. Therefore, the theorem applies to these classes of gates. In the following, we assume all gates are from a class $\mathcal{F}$ in $\mathcal{G}$ and hence there is a polynomial-time learning algorithm that learns projections and blurrings of gates from $\mathcal{F}$ using membership queries.

The main idea is natural. First we observe that an experiment s^0 that fixes all wires followed by an experiment s^1 that sets w_N free can be used to simulate a membership query on the output gate, as we can observe the value $g_N(s^0)$ after s^1 is applied. In general, given an experiment sequence $\xi = (s^1, s^2, \ldots, s^m)$, we can simulate a membership query s^0 on a gate g by inserting s^0 at the beginning of the experiment sequence, and setting the corresponding wire w free in s^1, i.e., we test the experiment sequence $(s^0, s^1|_{w=*}, s^2, \ldots, s^m)$. However, we can not observe $g(s^0)$ if the gate is not the output gate. But as usual, we can compare the circuit outputs of an experiment sequence and its perturbations. Let τ be a partial experiment. Let $\xi|_\tau$ be an experiment sequence obtained by modifying the first experiment of ξ by τ. More specifically, when $\xi = (s^1, s^2, \ldots, s^m)$, we have $\xi|_\tau = (s^1|_\tau, s^2, \ldots, s^m)$. We compare the circuit outputs of $(s^0, \xi|_{w=*})$ and $(s^0, \xi|_{w=\beta})$ for some $\beta \in \Sigma$ to determine whether $g(s^0) = \beta$, or more precisely, determine whether $g(s^0) \neq \beta$. More formally, we say an experiment sequence ξ is (w, α, β)-*exposing* if the circuit output sequences are different given two experiment sequences, $\xi|_{w=\alpha}$ and $\xi|_{w=\beta}$. If we partition Σ such that α and β are put in the same partition if and only if ξ is not (w, α, β)-exposing, and let g^ξ be the blurring of g under this partition, we can simulate a membership query on g^ξ by comparing the circuit outputs of $(s^0, \xi|_{w=*})$ with $(s^0, \xi|_{w=\beta})$, $\forall \beta \in \Sigma$.

By our assumption, we can learn g^ξ in polynomial time with membership queries. Given a collection of experiment sequences $\xi_1, \xi_2, \xi_3, \ldots$, we can learn $g^{\xi_1}, g^{\xi_2}, g^{\xi_3} \ldots$ For an s^0 that fixes all wires, each g gives us a suggested set of possible outputs, the intersection of which gives a more accurate guess on what $g(s^0)$ should be. Or in other words, the intersection of the $g^{\xi_i}(s^0)$'s gives a set of outputs for $g(s^0)$ that are consistent with $\xi_1, \xi_2, \xi_3, \ldots$ Another way to view the blurrings is that each $g^{\xi_i}(s^0)$ distinguishes $g(s^0)$ from the symbols in the complement $\Sigma \backslash g^{\xi_i}(s^0)$ (i.e., ξ_i is $(w, g(s^0), \beta)$-exposing $\forall \beta \in \Sigma \backslash g^{\xi_i}(s^0)$). The union of all the complements is the set of symbols $\xi_1, \xi_2, \xi_3, \ldots$ can distinguish $g(s^0)$ from.

Therefore, in Algorithm 4, our goal is to collect a sufficient set of (w, α, β)-exposing experiment sequences. It starts with an arbitrary length 1 experiment sequence s^1, which is surely an exposing experiment sequence for the output wire w_N for any pair of distinct symbols in Σ. We will show in the following

Algorithm 4 Learning a synchronous circuit

OUTPUT: $C' = \{\forall i \in [1, N], (w_i, g'_i)\}$.

Let ξ_0 be an arbitrary length 1 experiment sequence. Let V be a database of experiment sequences, with the keys being tuples in $W \times \Sigma \times \Sigma$. Initialize V to be empty. Add ξ_0 to V with the key (w_N, α, β) for any $\alpha, \beta \in \Sigma$ and $\alpha \neq \beta$; all entries are initially unmarked.
while there exists an unmarked experiment sequence ξ with key (u, α', β') in V **do**
  Mark ξ.
  Simulate the membership query algorithm to learn g^ξ, where g is the true gate of wire u.
  for $w \in W$, $\alpha, \beta \in \Sigma$ **do**
    if the key (w, α, β) is not in V **then**
      Compute s^0 such that $(s^0, \xi|_{u=*})$ is (w, α, β)-exposing or decide that s^0 does not exist (as described above).
      if s^0 exists **then**
        Add the unmarked entry (s^0, ξ) to V with the key (w, α, β).
For all $i \in [1, N]$ and for all s^0 that fixes all wires, let $g'_i(s^0) = \sigma$, where $\sigma \in \bigcap_{\xi \in V} g_i^\xi(s^0)$.

how to extend a known experiment sequence to experiment sequences for other wires using the membership query algorithm for $\mathcal{F}$.

The idea is simple. If there exists s^0 such that $g_N(s^0|_{w=\alpha}) \neq g_N(s^0|_{w=\beta})$, we can extend the experiment sequence to $(s^0, s^1|_{w_N=*})$, which is a (w, α, β)-exposing experiment sequence. In general, given an experiment sequence $\xi = (s^1, s^2, \ldots, s^m)$ and a wire u, let the gate function of u be g and let w be an input of g^ξ. We want to compute an s^0 such that $(s^0, s^1|_{u=*}, s^2, \ldots, s^m)$ is a (w, α, β)-exposing experiment sequence.

First we learn g^ξ using the simulated membership query learning algorithm. Consider the two projections $g^\xi|_{w=\alpha}$ and $g^\xi|_{w=\beta}$. s^0 does not exist if the two projections are equivalent (i.e., they output the same value for every input). Otherwise, we can simulate the learning algorithm to learn $g^\xi|_{w=\alpha}$. One query must be made to distinguish the two projections as they both belong to the class of gates by our assumption. That is, the learning algorithm must make a query on an s^0 such that $g^\xi|_{w=\alpha}(s^0) \neq g^\xi|_{w=\beta}(s^0)$. This s^0 serves our purpose. After we have learned g^ξ, we can answer the membership query of $g^\xi|_{w=\alpha}$ by computation. Thus, no additional value-injecting experiment is needed to compute s^0.

We claim that Algorithm 4 collects all exposing experiment sequences.

Lemma 7.2 *For any wire w, and $\alpha, \beta \in \Sigma$, if there exists a (w, α, β)-exposing experiment sequence, there exists one in V.*

PROOF. Suppose $\xi = (s^0, s^1, \ldots, s^m)$ is a (w, α, β)-exposing experiment sequence, but there is no corresponding (w, α, β)-exposing experiment sequence in V. Assume further that ξ has the minimum length among all experiment sequences that are exposing but do not have counterparts in V.

Consider the two experiment sequences $\xi|_{w=\alpha}$ and $\xi|_{w=\beta}$. Consider the second time step (after we apply s^1). Suppose wire w_i takes values α_i in $\xi|_{w=\alpha}$ and β_i in $\xi|_{w=\beta}$. We claim that there must be no (w_i, α_i, β_i)-exposing experiment sequence in V.

We do not need to prove anything if $\alpha_i = \beta_i$. When $\alpha_i \neq \beta_i$, w_i must set free in s^1 (if w_i is fixed, we will have that $\alpha_i = \beta_i$) and $g_i(s^0|_{w=\alpha}) = \alpha_i$ and $g_i(s^0|_{w=\beta}) = \beta_i$. Therefore, given a (w_i, α_i, β_i)-exposing experiment sequence ξ' in V, we will have that $g_i^{\xi'}(s^0|_{w=\alpha}) \neq g_i^{\xi'}(s^0|_{w=\beta})$, in which case, Algorithm 4 will extend it to a (w, α, β)-exposing experiment sequence.

By the minimality assumption on ξ, there does not exist any (w_i, α_i, β_i)-exposing sequence of length less than $m + 1$. Let $s^1_\alpha = s^1|_{\forall w_i, w_i = \alpha_i}$ and $s^1_\beta = s^1|_{\forall w_i, w_i = \beta_i}$. We can conclude that $(s^1_\alpha, s_2, \ldots, s_m)$ has the same circuit outputs as $(s^1_\alpha|_{w_1=\beta_1}, s_2, \ldots, s_m)$, as well as $(s^1_\alpha|_{w_1=\beta_1, w_2=\beta_2}, s_2, \ldots, s_m)$, as well as $(s^1_\alpha|_{w_1=\beta_1, w_2=\beta_2, w_3=\beta_3}, s_2, \ldots, s_m)$, etc., and finally that it has the same circuit outputs as $(s^1_\beta, s_2, \ldots, s_m)$, which leads to a contradiction. □

Since the number of keys in V is bounded by $|\Sigma|^2 N$, the running time of Algorithm 4 is polynomial in N. We now show that the constructed circuit C' is equivalent to C in the synchronous model.

PROOF. (of Theorem 7.1) As we have seen in the beginning of this section, for any experiment sequence ξ of length m, we can build the corresponding m-layer acyclic circuit $\tilde{C}$ and $\tilde{C}'$, and the outputs of $\tilde{C}$ and $\tilde{C}'$ are the last circuit outputs of C and C' over ξ. We will show the equivalence of $\tilde{C}$ and $\tilde{C}'$ for any m, which establishes the theorem.

First, we show that, if there exists a (w', α, β)-exposing experiment s in $\tilde{C}$, where w' is a layer t copy of wire w, there exists a (w, α, β)-exposing experiment sequence for C. All free wires in layer t must have the same values in $s|_{w'=\alpha}$ and $s|_{w'=\beta}$, because they are not reachable from w'. Therefore, if we fix them to these values, the resulting experiment s' will still be (w', α, β)-exposing. If we take first t layers of s', the corresponding experiment sequence will be (w, α, β)-exposing for C.

Since V contains a representative of every possible (w, α, β)-exposing experiment sequence and gate functions are picked to be consistent with them, we conclude that every gate function of $\tilde{C}'$ is correct (note that in $\tilde{C}$ and $\tilde{C}'$ every

copy of the same wire has the same gate function). The equivalence of $\tilde{C}$ and $\tilde{C}'$ is then established by Lemma 5.4. $\square$

8 Discussion

The learning algorithms and lower bounds in this paper outline the possibilities for tractable learning of circuits with small alphabets of wire values using value injection queries, with and without behavioral equivalence queries. Further results concerning the cases of large alphabets and analog values may be found in [3]. Another important topic is the learnability of probabilistic circuits or Bayesian networks using value injection queries; gate functions in potential applications are likely to be probabilistic. For circuits containing cycles, our synchronous model gives encouraging positive results, but its assumptions may be too strong in practice. In particular, the transient behavior of a circuit in response to an experiment may be hard to observe or asynchronous. Understanding the effects of relaxing these assumptions is important. An interesting open problem for acyclic circuits is whether there is an efficient *non-adaptive* algorithm to learn constant-depth, unbounded fan-in circuits over AND, OR, and NOT using value injection queries.

9 Acknowledgments

The work reported here was done while the third author was a graduate student in the Department of Computer Science at Yale University. A preliminary version of this paper appeared as [4]. The authors would like to thank the anonymous referees, Lev Reyzin, and David Eisenstat for their thoughtful comments on this paper.

References

[1] T. Akutsu, S. Kuhara, O. Maruyama, and S. Miyano. Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In *SODA '98: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 695–702, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[2] D. Angluin. Queries and concept learning. *Machine Learning*, pages 2(4) 319–342, 1988.

[3] D. Angluin, J. Aspnes, J. Chen, and L. Reyzin. Learning large-alphabet and analog circuits with value injection queries. In *the 20th Annual Conference on Learning Theory*, pages 51–65, 2007.

[4] D. Angluin, J. Aspnes, J. Chen, and Y. Wu. Learning a circuit by injecting values. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 584–593, New York, NY, USA, 2006. ACM Press.

[5] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

[6] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40:185–210, 1993.

[7] D. Angluin and M. Kharitonov. When won't membership queries help? *J. Comput. Syst. Sci.*, 50(2):336–355, 1995.

[8] N. H. Bshouty. Exact learning boolean functions via the monotone theory. *Inf. Comput.*, 123(1):146–153, 1995.

[9] H. Fujiwara. *Logic Testing and Design for Testability.* MIT Press, 1986.

[10] T. Ideker, V. Thorsson, and R. Karp. Discovery of regulatory interactions through perturbation: Inference and experimental design. In *Pacific Symposium on Biocomputing 5*, pages 302–313, 2000.

[11] J. C. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *J. Comput. Syst. Sci.*, 55(3):414–440, 1997.

[12] J. C. Jackson, A. R. Klivans, and R. A. Servedio. Learnability beyond AC0. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 776–784, New York, NY, USA, 2002. ACM Press.

[13] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994.

[14] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 372–381, New York, NY, USA, 1993. ACM Press.

[15] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[16] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. In *ACM Symposium on Theory of Computing*, pages 213–223, 1990.

[17] R. L. Rivest and R. Sloan. A formal model of hierarchical concept learning. *Inf. Comput.*, 114(1):88–114, 1994.

[18] G. Seroussi and N. H. Bshouty. Vector sets for exhaustive testings of logic circuits. In *IEEE Transactions on Information Theory*, volume 34, pages 513–522, May 1988.

[19] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27:1134–1142, 1984.