Query-driven PAC-Learning for Reasoning[*]

Omitted for blind review

Abstract

We consider the problem of learning rules from a data set that support a proof of a given query, under Valiant's PAC-Semantics. We show how any backward proof search algorithm that is sufficiently oblivious to the contents of its knowledge base can be modified to learn such rules while it searches for a proof using those rules. We note that this gives such algorithms for standard logics such as chaining and resolution.

1 Introduction

Machine learning, coupled with plentiful data, promises an approach to the problem of constructing the large knowledge bases needed for AI. Whereas traditional knowledge engineering by hand, as exemplified by the CYC project [Lenat, 1995], proved difficult to scale, machine learning holds the promise of producing a large and consistently interpreted knowledge base. Of course, any kind of inductive learning faces the danger of incorrect generalization, and thus such knowledge must use a semantics that is weaker than classical logic. Valiant [2000] proposed *PAC-Semantics* as a semantics for classical logics that is liberal enough to tolerate the imperfect rules produced by models of machine learning [Valiant, 1984]. Subsequently, Valiant [2006] also demonstrated that a large knowledge base can be soundly learned from a reasonable size data set. Michael and Valiant [2008] demonstrated the use of such knowledge bases on a sentence completion task, and a system using this approach [Isaak and Michael, 2016] was tied for top performance in the first Winograd Schema Challenge competition [Davis *et al.*, 2017].

Although Valiant [2006] showed that there is no *statistical* barrier to learning a large knowledge base, *computational* issues from representing and accessing such a large knowledge base may still arise. One way of avoiding these issues was proposed by Juba [2013], building on Khardon and Roth's *learning to reason* approach [Khardon and Roth, 1997]: instead of representing a knowledge base explicitly, Juba decides a query using the data directly, and guarantees that the result is sufficient to distinguish queries that have low validity from queries with small proofs using knowledge that *could* have been learned from the data. Thus, the query is answered using a knowledge base that is only *implicitly* learned. Crucially, this approach applies to settings where some attributes are not observed in the examples used for learning, and therefore some reasoning may be required. These attributes may still be mentioned in the background knowledge and query. For example, we may observe medical test results, but not whether a given patient actually has a given disease.

A drawback of Juba's approach, however, is that it provides no explicit representation of *what* knowledge could have been learned to support the query. It only provides a set of proofs for specific examples from the data set. This may not be adequately interpretable for human oversight of the system; if possible, we would like to inspect the knowledge that is being used to provide answers to our queries. Moreover, there are applications for which we are more interested in what knowledge could have been used to derive the conclusion than we are in the conclusion itself. For example, such algorithms might be applied to learn a screening rule for fraud detection as follows: Given a definition of behavior that is legitimate and a set of example transaction histories that are known to be legitimate, we could seek to learn what properties (if any) of the observed portions of these transactions can be used to guarantee that they are legitimate by using the definition of legitimate behavior as our query. We can then check whether or not these learned properties are observed to hold on future transactions to decide whether or not they warrant further inspection. Note that in this case the query is presumed to hold, and the interesting part is what properties we can discover to justify the query. We will discuss a similar but more easily formalized application to learning input filters later.

In this work, we show how for a large class of *oblivious backward search* algorithms for reasoning, we *can* explicitly identify rules that suffice to answer queries. Thus, we explicitly identify a sufficient set of relevant rules from this "implicit" knowledge base in a goal-driven fashion. We observe that this suffices to learn such rules for logics such as chaining and treelike resolution (e.g. DPLL-like algorithms [Davis and Putnam, 1960; Davis *et al.*, 1962]).

Our algorithms resemble algorithms arising in inductive logic programming (ILP) [Muggleton and De Raedt, 1994], in particular work by Muggleton and Buntine [1988] that con-

[*]Proofs and additional examples available in supplemental material: https://tinyurl.com/y2oqp2e4

structed rules for resolution; although Muggleton [1991] anticipated that a connection to PAC-Learning should exist, ILP learning theory is quite different. The main distinction here is conceptual: ILP treats the input examples as *defining* a domain for which we seek to synthesize a description. By contrast, in PAC-Semantics, we are simply seeking to bound the probability our formulas are true with respect to some probability distribution D over valuations, and the examples are simply drawn from D, and thus only statistically representative of it. We thus do not have complete knowledge of D nor do we even have access to the valuation of every formula in any given example. Although this bounding of probability is similar to a probability logic (e.g., as discussed by Nilsson [1986] for propositional logic or Halpern [1990] for first-order), we stress that the logical languages we use are simple Boolean logics such as chaining and resolution, and the probability bounds appear only in our semantics.

2 Problem formulation

We now describe the framework we use for learning and reasoning from partial examples (interpretations). First, we describe learning from partial examples, and give the key definition of concealment that captures when a credulous strategy for learning from partial examples will succeed. (Skeptical learning is considered later.) We then define a family of backward search algorithms, the family of algorithms for which we will be able to introduce query-driven learning. Since our guarantees will have the form of ensuring that these reasoning algorithms are as successful as if they had started their search with the learned knowledge given up-front, we will need a technical condition that the search algorithms are not too sensitive to the contents of the knowledge base they are given up-front—that is, "oblivious" to the knowledge base. Indeed, this definition will guarantee that we can introduce learned knowledge as the search proceeds without harming its performance, which is our main strategy.

2.1 Learning and reasoning in PAC-Semantics

Following Valiant [2000], we will describe our logic in terms of the linear threshold connective (a common generalization of the usual AND and OR connectives). The linear threshold connective has the advantage that it can capture softened versions of AND and OR.

Definition 1 (Threshold connective) *A* threshold connective *for a list of k formulas $\varphi_1, \dots, \varphi_k$ is given by a list of $k+1$ real numbers, $c_1, \dots, c_k, b$. The formula $[\sum_{i=1}^{k} c_i\varphi_i \geq b]$ is interpreted as follows: given a Boolean interpretation for the k formulas, the connective is true if $\sum_{i:\varphi_i=1} c_i \geq b$.*

A threshold connective expresses a k-ary AND connective by taking the $c_i = 1$, and $b = k$, and expresses a k-ary OR by taking $c_1, \dots, c_k, b = 1$. Negation corresponds to $c_i < 0$.

The main feature of PAC-Semantics is a probability distribution D on interpretations of the relation symbols, i.e., assignments of truth values to their groundings. Equivalently, we take each ground atomic formula as a Boolean-valued random variable. We stress that we do not assume independence (or any other relationship) between these random variables. Given an interpretation drawn from D, the semantics of a formula are then defined classically. $KB \models \varphi$ denotes that φ is a *(classically) valid formula given the knowledge base (set of formulas) KB*. We denote the truth value of a formula φ under an interpretation x as $\varphi|_x$. We may view an interpretation as a Boolean vector indexed by the set of ground atomic formulas. We refer to these as *scenes*.

Definition 2 ($(1-\epsilon)$-valid) *A sentence (i.e., formula with no free variables) φ is said to be* $(1-\epsilon)$-valid under D *if the probability that φ evaluates to true under an interpretation drawn from D is at least $1-\epsilon$. If $\epsilon = 0$, we say that φ is* perfectly *valid.*

Now, an *obscured scene* is a *partial* interpretation of the ground atomic formulas of the logic:

Definition 3 (Obscured scene) *A* obscured scene *ρ is a mapping taking ground atomic formulas to $\{0, 1, *\}$ where * denotes an "unknown" value. We say that an obscured scene ρ is* consistent *with an interpretation if whenever ρ assigns an ground atomic formula a value other than $*$, the interpretation agrees with ρ.*

We need obscured scenes because frequently our knowledge base will refer to atomic formulas that are not observed in the data we use for learning. Sometimes these unobserved atomic formulas take the form of properties we wish to reason about or predict with learned rules. These may also, for example, refer to the contents of variables during an execution in program analysis, where the "obscured scene" itself might simply be an example input to the program.

Following Juba [2013], we will consider the following operation that uses obscured scenes to partially evaluate quantifier-free formulas defined using linear threshold connectives. These could have been obtained by propositionalization, and a suitable family of formulas for these purposes is described by Valiant [2000]. Note that the recursive definition corresponds to a linear-time algorithm for computing these partially evaluated formulas:

Definition 4 (Partial evaluation and witnessing) *Given an obscured scene ρ and a quantifier-free formula φ, the* partial evaluation of φ under ρ, *denoted $\varphi|_\rho$, is recursively defined as follows; when the partial evaluation produces a Boolean constant, we say that the formula is* witnessed*: A ground atomic formula φ is replaced by its value under ρ (i.e., it is witnessed) unless this value is *, in which case it remains φ. For $\varphi = [\sum_{i=1}^{k} c_i\psi_i \geq b]$, 1. φ is witnessed true if $\sum_{i:\psi_i \text{ witnessed true}} c_i + \sum_{i:\psi_i \text{ not witnessed}} \min\{0, c_i\} \geq b$, 2. φ is witnessed false if $\sum_{i:\psi_i \text{ witnessed true}} c_i + \sum_{i:\psi_i \text{ not witnessed}} \max\{0, c_i\} < b$, 3. and otherwise, supposing that $\psi_1, \dots, \psi_\ell$ are witnessed in ρ (and $\psi_{\ell+1}, \dots, \psi_k$ are not witnessed), $\varphi|_\rho$ is $[\sum_{i=\ell+1}^{k} c_i(\psi_i|_\rho) \geq d]$ where $d = b - \sum_{i:\psi_i|_\rho=1} c_i$.*

Following Rubin [1976] and Michael [2010], we suppose that a "masking process" takes interpretations drawn from the distribution D and hides some ground atomic formulas.

Definition 5 (Masking process) *A* mask *is a function mapping interpretations to obscured scenes that are consistent* *with the respective interpretations. A* masking process M *is a mask-valued random variable (i.e., a random function). We denote the probability distribution over obscured scenes obtained by applying a masking process M to a distribution D over interpretations by $M(D)$.*

For a given ground atomic formula α, a PAC-Learning algorithm could be used to learn a formula φ that predicts α, in which case the formula $[\varphi \equiv \alpha]$ is $(1-\epsilon)$-valid with probability $1-\delta$ over the random example scenes (for δ given to the algorithm). Following Michael [2010], we consider learning from example obscured scenes, provided that the value of α is not obscured in too many of the examples. Essentially we distinguish formulas that are *perfectly* valid under the unknown distribution from those that are not even $(1-\epsilon)$-valid for some given $\epsilon > 0$. The main finding is that the learnability of such rules is controlled by the probability of observing counterexamples to flawed rules under the masking process.

Definition 6 (Concealment) *We say that a masking process M is (at most)* $(1-\eta)$-concealing *with respect to a set of formulas $\mathcal{C}$ and a distribution over interpretations D if*

$$\forall \varphi \in \mathcal{C} \Pr_{x \in D, m \in M}[\, \varphi \text{ witnessed on } m(x) \mid \varphi|_x = 0 \,] \geq \eta.$$

Bounded concealment justifies a "credulous" learning strategy: rules are satisfactory as long as we do not observe counterexamples to them.

Proposition 7 (Theorem 2.2 of [Michael, 2010]) *For any distribution D over interpretations, masking process M, and class $\mathcal{C}$ of formulas, if M is $(1-\eta)$-concealing with respect to $\mathcal{C}$ and D and $\varphi \in \mathcal{C}$ is not $(1-\epsilon)$-valid, then $\Pr_{\rho \in M(D)}[\varphi|_\rho = 0] \geq \eta\epsilon$. Conversely, if M is not $(1-\eta)$-concealing with respect to $\mathcal{C}$ and D then there exists $\varphi \in \mathcal{C}$ such that $\Pr_{\rho \in M(D)}[\varphi|_\rho = 0] \leq \eta \Pr_{x \in D}[\varphi|_x = 0]$ (where, note, φ is $(1 - \Pr_{x \in D}[\varphi|_x = 0])$-valid).*

That is, the degree of concealment controls the discounting of the probability of observing counterexamples to formulas from $\mathcal{C}$. We have actually modified the definitions slightly from the original version by including the distribution in the definition of concealment, and moreover, by using a notion of witnessed evaluation (as opposed to the value merely being *determined* by the obscured scene) but the proof is similar.

In the above formulation of PAC-Learning using equivalence rules, this means that for any incorrect hypothesis in our representation class, the value of α should be witnessed (by the masking process) on an example where the hypothesis is incorrect with probability at least η. It turns out that for many classes of interest, learning is still possible as long as the masking process features an η bounded away from zero.

In this work, we are interested in reasoning problems, in which we only consider proofs of bounded complexity. Formally, we consider the following family of problems:

Definition 8 (Search problem) *Fix a proof system, and let $\mathcal{P}$ be a set of proofs in the proof system (e.g., of bounded complexity). The* search problem for $\mathcal{P}$ *is then the following promise problem: given as input a formula φ with no free variables and a set of formulas KB such that either there is a proof of φ in $\mathcal{P}$ from KB or else $KB \not\models \varphi$, return such a proof in the former case, and return "Fail" in the latter.*

Our first example of a proof system is a mild generalization of the usual forward-chaining system that was presented by Valiant [2000]. It is designed to utilize rules of the form $[\varphi \equiv \alpha]$ in which α is an atomic formula, i.e., of the sort obtained from PAC-Learning algorithms. The main inference rule is *chaining*: given a formula of the form $[\varphi \equiv \alpha]$ in which α is a ground atomic formula, and a consistent set of literals (atomic formulas or their negations) $\{\ell_1, \ldots, \ell_k\}$ such that for the obscured scene ρ that satisfies $\ell_1, \ldots, \ell_k$ (and leaves every ground atomic formula not appearing in this list unassigned) $\varphi|_\rho \in \{0,1\}$, if $\varphi|_\rho = 0$, infer $\neg\alpha$, and otherwise (i.e., if $\varphi|_\rho = 1$) infer α.

2.2 A model of backward search algorithms

Informally, such "backward" (goal-directed) algorithms start with a goal query and repeatedly generate sets of subgoal queries such that if all of the subgoal queries succeed – i.e., if proofs can be found for all of these subgoal queries – then the algorithm can construct a proof of the goal query. Although this informal description strongly suggests a recursive algorithm, in general it is highly desirable to cache the results of subgoal queries when they are answered—beyond the obvious time-efficiency improvements, it is often possible for a naïve recursive algorithm to get stuck following a circular sequence of subgoals. Thus, our model of such algorithms will be stated in terms of a graph indicating the dependency structure among the (sub)goals considered by the algorithm.

This graph would conventionally be an "AND-OR" graph: a query would be associated with an OR node, with edges to various alternative lists of subgoal queries, represented by AND nodes with edges to the OR nodes corresponding to the queries in the list. The success of the algorithm corresponds to the goal query node evaluating to 'true' in the natural interpretation of such a graph when one more generally associates success at finding a proof of a (subgoal) query with a node evaluating to 'true.' Given our interest in using rules expressed using linear threshold connectives, though, we will find it natural and convenient to replace the AND nodes with more general "linear threshold" nodes, expressing that success at the node is achieved if an appropriate subset of the subgoals are successful.

Definition 9 (Subgoal dependency graph) *A* subgoal dependency graph *is a (possibly infinite) directed graph G in which the vertices are either* query nodes *labeled with a formula or are labeled with an integer* threshold *and have outgoing edges labeled by integer* weights. *A* partial subgoal dependency graph *also contains, for each threshold vertex, weights w_+ and w_-. Given sets of* successful *nodes S and* unsuccessful *nodes U, each node v is considered* successful *using S, U if there is an acyclic subgraph of G featuring v as the (unique) source with sinks from $S \cup U$ such that v has a path to every sink and at every non-query node, the sum of the weights on the outgoing edges to vertices in S plus w_- and the negative weights on outgoingedges to vertices outside S or U is at least the threshold. Similarly, v is* unsuccessful *if the sum of the weights on outgoing edges to vertices in S plus w_+ and the positive weights of edges to vertices outside S and U is less than its threshold.*

```
input  : Query formula $\varphi$, set of formulas $KB$
begin
    if $\varphi \in KB$ then
        return Trivial proof of $\varphi$
    end
    $G \leftarrow$ vertex labeled by $\varphi$, marked "unexplored"
    while TEST$(G, \varphi, KB) =$ FAIL do
        if $v \leftarrow$EXPLORE$(G, \varphi, KB)$ is not FAIL then
            if $G' \leftarrow$GENERATE$(G, v)$ is not "fully explored" then
                if $G'$ contains a vertex not in G, not labeled with $\psi \in KB$ then
                    Mark the new vertex "unexplored."
                end
                $G \leftarrow G'$
            end
            else Remove "unexplored" mark from $v$
        end
        else return Fail
    end
    return TEST$(G, \varphi, KB)$
end
```

Algorithm 1: Backward search meta-algorithm

Our rule for determining when a vertex is successful or unsuccessful match our rules for witnessing connectives true and false, respectively. The weights w_+ and w_- will allow us to determine witnessing with (unwitnessed) unrepresented vertices. Our sucessful vertices will intuitively represent either a provable query, or an applicable inference.

The backward search algorithm is now a meta-algorithm (Algorithm 1) parameterized by three sub-algorithms. One algorithm, `GENERATE`, generates the subgoal dependency graph, and another, `EXPLORE`, chooses edges in the dependency graph to explore (as long as the algorithm is not done). The third algorithm, `TEST`, generates a proof of the query if enough of the subgoal dependency graph has been revealed so that the original goal vertex was successful (given the axioms and a knowledge base as successful vertices), or else indicates that the search is not successful yet. Thus, the algorithm explores the subgoal dependency graph (starting from the original query) until an appropriate collection of successful subgoals is discovered or the search algorithm gives up.

Definition 10 (Backward search algorithm) *A backward search algorithm is given by an instantiation of Algorithm 1 with three algorithms: An algorithm* `EXPLORE` *that, given a finite partial subgoal dependency graph G with a source labeled by φ and a subset of vertices marked "unexplored," and set of formulas KB, chooses an unexplored vertex $v \in G$ or outputs "FAIL." An algorithm* `GENERATE` *that, given a partial subgoal dependency graph G and a vertex $v \in G$ either returns "fully explored" or returns a subgoal dependency graph G' that extends G by adding one new edge starting from v, possibly to a new vertex, and reducing w_+ or w_- by its weight if it is positive or negative, respectively. An algorithm* `TEST` *that, given a partial subgoal dependency graph G, a query formula φ labeling some vertex of G, and a set of formulas KB, either returns a proof of φ from KB, or if the vertex labeled by φ is not successful using KB and the axioms of the logic, returns "FAIL."*

A key property possessed by instantiations of the backward-search paradigm is that the graph generation algorithm is often *oblivious* to which queries are successful, the algorithm exploring the graph only "terminates early" when it encounters a vertex labeled by a successful query, and the algorithm recovering the proof depends only on the portion of the subgoal dependency graph revealed thus far (and the formulas appearing on query vertices appearing in it). We will restrict our attention to such *oblivious* algorithms.

Definition 11 (Oblivious backward search algorithm) *We say that a backward search algorithm is* oblivious *if:*

1. *For any partial subgoal dependency graph G, query φ (contained as a label in G), and set of formulas Φ not appearing as labels of query nodes in G,* TEST$(G, \varphi, KB) =$ TEST$(G, \varphi, KB \cup \Phi)$.
2. *For any query φ and sets of formulas KB and Φ, the execution of the algorithm on input φ and $KB \cup \Phi$ differs from the execution on input φ and KB only in that the sequence of vertices proposed by* `EXPLORE` *on input φ and $KB \cup \Phi$ is the subsequence of vertices proposed on input φ and KB that omits (skips) exploring vertices in the sequence that are successful from $KB \cup \Phi$ in the partial subgoal dependency graph used by the algorithm in the corresponding step.*

An example: oblivious backward chaining

We briefly note that standard backward-chaining algorithms (for Horn KBs on ground atomic formulas) are oblivious backward search algorithms in the sense of Definition 11: recall that a *Horn clause* is a formula of the form $[\alpha_1 \wedge \cdots \wedge \alpha_k] \Rightarrow \alpha_{k+1}$ where each α_i is a ground atomic formula. α_{k+1} is the *head* whereas $\alpha_1 \wedge \cdots \wedge \alpha_k$ is the *body*. The knowledge base then consists of such clauses and a set of ground atomic formulas. The query is a conjunction of ground atomic formulas, represented as a threshold vertex with a threshold equal to the number of atomic formulas in the conjunction.

The oblivious backward chaining algorithm works as follows: `EXPLORE` performs a depth-first search of the subgoal dependency graph, terminating a search early only when it encounters an atomic formula in KB. `GENERATE`, on the other hand, when given a vertex corresponding to a ground atomic formula, returns an edge to a threshold vertex corresponding to the next clause in KB with the given atomic formula appearing in the head (in some fixed ordering), with a threshold equal to the number of atomic formulas in the body; when given a vertex corresponding to one of the clauses of KB (or the goal conjunction), it returns an edge to the vertex labeled with the next atomic formula in the body (also in some fixed ordering) of weight 1. Finally, `TEST` uses a dynamic programming algorithm to check if the query is successful from KB and return a chaining proof if it is. The running time is bounded by a polynomial in the size of KB: it runs for a linear number of iterations, and `TEST` may take quadratic time on each iteration.

```
input : Query formula $\varphi$, set of formulas $KB$, list of
        obscured scenes $\rho_1 \ldots, \rho_m$
begin
    $G \leftarrow$ vertex labeled by $\varphi$, marked "unexplored"
    while TEST$(G, \varphi, KB) = FAIL$ do
        if $v \leftarrow$EXPLORE$(G, \varphi, KB)$ is not FAIL then
            if $G' \leftarrow$GENERATE$(G, v)$ is not "fully
            explored" then
                if $G'$ contains a query vertex labeled by
                a formula $h$ not in $G$ and for no $\rho_i$ is
                $h|_{\rho_i} = 0$ then $KB \leftarrow KB \cup \{h\}$
                if $G'$ contains a vertex not in $G$, not
                labeled with $\psi \in KB$ then
                    Mark the new vertex "unexplored."
                end
                $G \leftarrow G'$
            end
            else Remove "unexplored" mark from $v$
        end
        else return Fail
    end
    return TEST$(G, \varphi, KB)$
end
```

Algorithm 2: Backward search with query-driven learning

3 Query-driven learning in backward search

The relative blindness of oblivious algorithms to the contents of the knowledge base allows us to add new members as the search proceeds, and obtain the same result as if we had started with them. As some algorithms may consider families of formulas that scale with the size of the query, in our theorem we will parameterize our bounds on the proof size B and degree of concealment η by the size of the query ℓ (in bits). For example, adding clauses to a query for resolution generally increases the variety of clauses that may be derived. For larger families, we expect that the bounds grow weaker.

Theorem 12 *Let $\mathcal{P}$ be a set of proofs such that proofs of queries of length ℓ only have proofs with $B(\ell)$-bit encodings in $\mathcal{P}$ (in some fixed encoding scheme). Suppose there is an oblivious backward search algorithm for the search problem for $\mathcal{P}$ that on input φ and KB over N ground atomic formulas, runs in time $T(N, |\varphi|, |KB|)$ (for a function T that is monotone increasing in $|KB|$). Let D be a distribution over scenes and M be a masking process that is at most $(1-\eta(\ell))$-concealing for the set of formulas Φ that may be hypotheses in proofs of formulas of length ℓ in $\mathcal{P}$. Then for any $\delta, \epsilon \in (0,1)$, on input φ and KB and $\Theta(\frac{1}{\epsilon\eta(|\varphi|)}(B(|\varphi|) + \log 1/\delta))$ example obscured scenes, Algorithm 2 using the same* EXPLORE, GENERATE, *and* TEST *as the given algorithm runs in time $O(\frac{B(|\varphi|)}{\epsilon\eta(|\varphi|)}(B(|\varphi|) + \log 1/\delta)T(N, |\varphi|, C))$ (for $C = |KB| + T(N, |\varphi|, |KB|)$), and with probability $1-\delta$ returns "Fail" if $[KB \Rightarrow \varphi]$ is not $(1-\epsilon)$-valid with respect to D, or returns a proof of φ from $KB \cup H'$ for a set of formulas $H' = \{h'_1, \ldots, h'_k\}$ such that $h'_1 \wedge \cdots \wedge h'_k$ is $(1-\epsilon)$-valid if there exists a set of perfectly valid formulas H such that there is a proof of φ from $KB \cup H$ in $\mathcal{P}$.*

Example: backward chaining

If we represent chaining proofs by the sequence of inferred ground atomic formulas (using $\log N$ bits for each), then since any proof needs only write down an atomic formula at most once, we can use the bound $B = N \log N$ in the statement of Theorem 12. (The size of the query ℓ is always the length of a single ground atomic formula, $\log N$ bits.) So, after applying the transformation depicted in Algorithm 2 to the standard backward-chaining algorithm, Theorem 12 establishes that this modified backward chaining algorithm finds chaining proofs of queries using not only ground atomic formulas from the explicitly given KB, but also additional formulas that are almost always true. The modified algorithm automatically supplements a given KB with any such additional ground atomic formulas that suffice to complete some chaining proof of a query if one exists, provided further that the masking process has bounded concealment with respect to ground atomic formulas—meaning here, the masking process leaves the value of each ground atomic formula present with some bounded probability when it is false.

3.1 Skeptical query-driven learning

Theorem 12 relies on an assumption of bounded concealment, and uses a credulous learning strategy of searching for hypotheses for which we do not possess counterexamples. A more conservative strategy would replace the condition "for no ρ_i is $h|_{\rho_i} = 0$" in Algorithm 2 with the condition "for all ρ_i, $h|_{\rho_i} = 1$." An h that is witnessed true with probability 1 will pass this condition. Moreover, if $h|_{\rho_i} = 1$ for $\rho_i = m_i(x_i)$ (where m_i is drawn from M and x_i is drawn from D), then $h|_{x_i} = 1$ as well. Thus, the probability that an h that is *not* $(1-\epsilon)$-valid with respect to D passes this test for m examples is less than $(1-\epsilon)^m$. We can use this observation in place of Proposition 7 to finish an analogous proof, given that we are searching for an H that is always *witnessed* rather than one that is merely perfectly *valid*. We leave further details to an interested reader.

Example application: learning input filters

We note that our transformation for skeptical learning of rules could be applied to static program analysis algorithms to automate the generation of sound and approximately complete input filters. For example, the SIFT system [Long *et al.*, 2014] is based on a set of sound transformation rules for analyzing integer overflow errors in a given program. Its associated static analysis algorithm uses the knowledge base given by these transformation rules and the program code to generate symbolic expressions that are propagated backwards from integer operations that might produce overflows, until they refer only to program inputs. The condition expressed by these expressions may then be used to filter out inputs that do not satisfy the condition. The soundness of SIFT's transformation rules ensures that no input that passes this condition generates an integer overflow error.

We can interpret SIFT as taking the safety property of "no integer overflows occur at the given point in the program" as a query, and seeking a proof of this query using a property of the input, together with the transformation rules and program code. Note that once SIFT generates expressions that refer only to input values, the conditions they express are witnessed against example inputs, if they are $(1-\epsilon)$-valid for inputs in practice. Thus, we can apply the transformation of the skeptical variant of Algorithm 2 to the static analysis algorithm used by SIFT, and we would obtain an algorithm with a similar termination condition, with the added requirement that the condition found must be witnessed on a set of given examples.

Thus, the distinction between the approach taken by Long et al. and our transformation is that SIFT has no guarantee that it produces a rule that is testable on real inputs. SIFT does not use any training data, and simply terminates once it finds a rule that refers only to the input; thus, while this rule is sound – inputs that satisfy it provably do not generate integer overflow errors – it has no guarantee of completeness, approximate or otherwise. Indeed, SIFT conservatively considers all possible execution paths and relatedly uses some hard-coded limits on, for example the number of loop iterations it considers in search of loop invariants, to ensure termination. If this limit is exceeded because the loop potentially relies on an unbounded number of values (for example), then SIFT simply fails to find a condition. Thus, there is scope for a backtracking variant of SIFT's static analysis algorithm to obtain greater completeness by iteratively refining a symbolic condition. Under our transformation, we would then obtain an algorithm that searches for a condition on the inputs that empirically satisfies witnessing for a large fraction of a training set of benign inputs.

3.2 Query-driven learning in treelike resolution

Recall that *resolution* is a logic that operates on *clauses* (ORs of literals), using two kinds of inference rules: *cut* and (optionally) *weakening*. Cut takes two clauses $C = C' \vee \alpha$ and $D = D' \vee \neg\alpha$ and produces a clause of the form $C' \vee D'$. (More general variants use substitutions to unify distinct atomic formuals α and $\neg\alpha'$.) Weakening, by contrast, simply adds new literals to the clause. Resolution is typically used to prove a DNF (an OR of ANDs) by deriving an (unsatisfiable) empty clause from its negation, a CNF.

DPLL [Davis and Putnam, 1960; Davis *et al.*, 1962] is another example of such a goal-directed search algorithm. In particular, bounded variants of DPLL that (efficiently) solve the proof search problem for space-bounded treelike resolution are known [Kullmann, 1999; Esteban and Torán, 2001]. This is the fragment of resolution refutations that can be derived while *(i)* storing at most s clauses in memory simultaneously and *(ii)* "forgetting" a clause as soon as it is used in a proof step (so that it must be derived again if it is needed again). This is a second example of an algorithm into which we can introduce query-driven explicit learning along the lines of Theorem 12. The algorithm will be more interesting because it will discover a CNF that suffices to complete the proof out of an exponentially large (in terms of the number of ground atomic formulas) set of possible such formulas.

Unfortunately, we cannot apply Theorem 12 directly, as the standard algorithm is not actually *oblivious* in our strict sense. The difficulty is that the base case of the recursive algorithm involves a search for a hypothesis clause for which we can derive the current clause via weakening. This "looking ahead" into the hypothesis set means that the algorithm may not engage in exactly the same search pattern if we modify the hypothesis set during the search. Nevertheless, the use is innocuous enough that essentially the same technique can be used to give a variant of the algorithm that learns an explicit set of clauses from the data.

```
input : CNF $\varphi$, integer space bound $s \geq 1$, current
        clause $C$, list of obscured scenes $\rho^{(1)}, \ldots, \rho^{(m)}$.
Learn+SearchSpace($\varphi, s, C, (\rho^{(1)}, \ldots, \rho^{(m)})$)
begin
    if No $\rho^{(i)}$ has $C|_{\rho^{(i)}} = 0$ then
        return A proof asserting C (from H).
    end
    else if C is a superset of some clause $C'$ of $\varphi$ then
        return The weakening derivation of C from $C'$.
    end
    else if $s > 1$ then
        foreach Literal $\ell$ such that neither $\ell$ nor $\bar{\ell}$ is in C
        do
            if $\Pi_1 \leftarrow$Learn+SearchSpace
            ($\varphi, s-1, C \vee \ell, (\rho^{(i)} : \ell|_{\rho^{(i)}} \neq 1)$) does not
            return none then
                if $\Pi_2 \leftarrow$Learn+SearchSpace
                ($\varphi, s, C \vee \bar{\ell}, (\rho^{(i)} : \ell|_{\rho^{(i)}} \neq 0)$) does not
                return none then
                    return Derivation of C from $\Pi_1$ and
                    $\Pi_2$
                end
                else return none
            end
        end
    end
    return none
end
```
Algorithm 3: Space-bounded resolution with learning

The running time of Algorithm 3 (and its basic correctness for finding space-s treelike resolution proofs) essentially follows the standard analysis. Using a bound on the lengths of proofs in terms of the space used and number of ground atomic formulas, we obtain:

Theorem 13 *Let a clause C and a (KB) CNF φ be given. Suppose the examples are drawn from a masking process that is $(1-\eta)$-concealing with respect to CNFs of size $(eN/s-1)^{s-1}$ for the distribution D; suppose further that φ is perfectly valid with respect to D and there exists some other perfectly valid CNF H for which there is a space-s treelike resolution proof of C from $\varphi \wedge H$. Then, Algorithm 3 run on φ and C with parameter s on a sample of size $\Theta((N^{s-1}\log N + \log\frac{1}{\delta})\frac{1}{\eta\epsilon})$ runs in time $O(\frac{N^{2(s-1)}|\varphi|}{\eta\epsilon}(N^{s-1}\log N + \log\frac{1}{\delta}))$ and returns a proof of C from $\varphi \wedge H'$ for some CNF H' of size $O(N^{s-1})$ that is $(1-\epsilon)$-valid with respect to D with probability $1-\delta$. Similarly, if $[\varphi \Rightarrow C]$ is not $(1-\epsilon)$-valid, Algorithm 3 rejects in the same time bound using the same number of examples.*

References

[Davis and Putnam, 1960] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *JACM*, 7(3):201–215, 1960.

[Davis *et al.*, 1962] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[Davis *et al.*, 2017] Ernest Davis, Leora Morgenstern, and Charles L Ortiz. The first Winograd Schema Challenge at IJCAI-16. *AI Magazine*, 38(3):97–98, 2017.

[Esteban and Torán, 2001] Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Inf. Comp.*, 171(1):84–97, 2001.

[Halpern, 1990] Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.

[Isaak and Michael, 2016] Nicos Isaak and Loizos Michael. Tackling the Winograd Schema Challenge through machine logical inferences. In David Pearce and Helena Sofia Pinto, editors, *STAIRS*, volume 284 of *Frontiers in Artificial Intelligence and Applications*, pages 75–86. IOS Press, 2016.

[Juba, 2013] Brendan Juba. Implicit learning of common sense for reasoning. In *Proc. 23rd IJCAI*, pages 939–946, 2013.

[Khardon and Roth, 1997] Roni Khardon and Dan Roth. Learning to reason. *J. ACM*, 44(5):697–725, 1997.

[Kullmann, 1999] Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. Technical Report TR99-041, ECCC, 1999.

[Lenat, 1995] Douglas B. Lenat. CYC: a large-scale investment in knowledge infrastructure. *CACM*, 38(11):33–38, 1995.

[Long *et al.*, 2014] Fan Long, Stelios Sidiroglou, Deokhwan Kim, and Martin Rinard. Sound input filter generation for integer overflow errors. In *Proc. 41st POPL*, 2014.

[Michael and Valiant, 2008] Loizos Michael and Leslie G. Valiant. A first experimental demonstration of massive knowledge infusion. In *Proc. 11th KR*, pages 378–389, 2008.

[Michael, 2010] Loizos Michael. Partial observability and learnability. *Artificial Intelligence*, 174(11):639–669, 2010.

[Muggleton and Buntine, 1988] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proc. 5th ICML*, pages 339–352, 1988.

[Muggleton and De Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Logic Programming*, 19:629–679, 1994.

[Muggleton, 1991] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

[Nilsson, 1986] Nils J Nilsson. Probabilistic logic. *Artificial intelligence*, 28(1):71–87, 1986.

[Rubin, 1976] Donald B. Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.

[Valiant, 1984] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 18(11):1134–1142, 1984.

[Valiant, 2000] Leslie G. Valiant. Robust logics. *Artificial Intelligence*, 117:231–253, 2000.

[Valiant, 2006] Leslie G. Valiant. Knowledge infusion. In *Proc. AAAI-06*, pages 1546–1551, 2006.