中央研究院
資訊科學研究所
Institute of Information Science, Academia Sinica • Taipei, Taiwan, ROC

TR-IIS-11-006

# Learning Boolean Functions Incrementally

**Yu-Fang Chen and Bow-Yaw Wang**

**Dec. 31, 2011 || Technical Report No. TR-IIS-11-006**
http://www.iis.sinica.edu.tw/page/library/TechReport/tr2011/tr11.html

# Learning Boolean Functions Incrementally*

Yu-Fang Chen and Bow-Yaw Wang

Academia Sinica, Taiwan

**Abstract.** Classical learning algorithms for Boolean functions assume that unknown targets are Boolean functions over fixed variables. The assumption precludes scenarios where indefinitely many variables are needed. It also induces unnecessary queries when many variables are redundant. Based on a classical learning algorithm for Boolean functions, we develop two learning algorithms to infer Boolean functions over enlarging sets of ordered variables. We evaluate their performance in the learning-based loop invariant generation framework.

## 1 Introduction

Algorithmic learning is a technique for inferring a representation of an unknown target in a specified instance space. When designing a learning algorithm, one formalizes intended scenarios as a learning model. In Boolean function learning, for instance, we are interested in finding a representation (such as a Boolean formula [3]) of an unknown target amongst Boolean functions over fixed variables. The goal of a learning algorithm is to generate a representation of the unknown target under the learning model [1, 13].

Inferring unknown targets over fixed variables however is not realistic in applications such as loop invariant generation [11, 14, 12], or contextual assumption synthesis [5, 4]. In loop invariant generation, one considers a loop annotated with pre- and post-conditions. The instance space hence consists of quantifier-free formulae over a given set of atomic predicates. We are interested in finding a quantifier-free formula which establishes the pre- and post-conditions in the specified instance space [11, 14, 12]. Through predicate abstraction [17, 7], a quantifier-free formula over fixed atomic predicates is associated with a Boolean function over fixed variables. A learning algorithm for Boolean functions can thus be adopted to infer loop invariants over a fixed set of atomic predicates. Note that the given set of atomic predicates may not be able to express any loop invariant. If the current atomic predicates are not sufficiently expressive, more atomic predicates will be added. Hence the set of atomic predicates is not fixed but indefinite. Yet classical learning presumes a fixed set of variables for unknown targets. It does not consider scenarios where new variables can be introduced on the fly. The classical learning model therefore do not really fit the scenario of loop invariant generation.

Another drawback in classical learning algorithms for Boolean functions is their inefficiency in the presence of redundant variables. In contextual assumption generation, one considers the problem of verifying a system composed of two components.

* This work is partially supported by the National Science Council of Taiwan under the grant numbers 99-2218-E-001-002-MY3 and 100-2221-E-002-116-.

We would like to replace one of the components by a contextual assumption so as to verify the system more efficiently. The instance space therefore consists of transition relations over model variables. We are interested in finding the transition relation of a contextual assumption that solves the verification problem [5, 4]. Recall that characteristic functions of transition relations are Boolean functions. A learning algorithm for Boolean functions hence can generate contextual assumptions in automated assume-guarantee reasoning. Observe that a contextual assumption is synthesized for a specific verification problem. If a model variable is not relevant to the problem, contextual assumptions can safely ignore it. Thus we are looking for an unknown transition relation over a subset of model variables. One would naturally expect a learning algorithm to perform really well when many model variables are irrelevant. Yet the complexity of classical learning algorithms depends on the number of given variables, not relevant ones. Classical learning can be unexpectedly inefficient when many given variables are redundant.

Both issues can be addressed by reformulating the learning problem for Boolean functions. In the new formulation, we infer a representation of an unknown target among Boolean functions over *indefinitely many* variables. Note that the new instance space leaves the number of variables unspecified. The new formulation hence fits the scenario of loop invariant generation. Moreover, the number of variables is no longer a parameter to the learning problem. The complexity of learning algorithms under the new formulation can only depend on the number of variables in the unknown target. Such algorithms can be more efficient in contextual assumption generation.

We propose to infer Boolean functions over indefinitely many variables by incremental learning. Instead of Boolean functions over a fixed number of variables, we infer the unknown target by enlarging sets of *ordered* variables incrementally. At iteration $\ell$, we try to infer the unknown target as a Boolean function over the first $\ell$ variables. Our incremental learning algorithm terminates if it infers the target. Otherwise, it proceeds to the next iteration and tries to infer the unknown target as a Boolean function over the first $\ell + 1$ variables. Since the unknown target is over finitely many variables, our incremental learning algorithm will infer the target after finitely many iterations.

A naive approach to incremental learning is to apply the classical CDNF learning algorithm for Boolean functions at each iteration. If the classical algorithm fails to infer the unknown target as a Boolean function over the first $\ell$ variables, the naive incremental algorithm instantiates the classical algorithm again to infer a Boolean function over the first $\ell + 1$ variables at the next iteration. The simple approach however does not work. Note that the complexity of the CDNF algorithm depends on the formula size of the unknown target. When targets are arbitrary, their formula sizes are exponential in the number of variables. Since $\Omega(2^\ell)$ queries are needed to infer an arbitrary target over $\ell$ variables in the worst case, the naive algorithm has to make as many queries before it gives up the iteration $\ell$. Subsequently, the naive algorithm would require an exponential number of queries for *every* unknown target and could not be efficient.

To solve this problem, we develop a criterion to detect failures at each iteration dynamically. At iteration $\ell$, our incremental algorithm checks whether the unknown target is a Boolean function over the first $\ell$ variables during the course of inference. If the incremental algorithm detects that the target needs more than the first $\ell$ variables,

the iteration $\ell$ is going to fail. Hence the incremental learning algorithm should abort and proceed to the next iteration. We propose two incremental learning algorithms with dynamic failure detection. In our simple incremental learning algorithm CDNF+, the classical learning algorithm is initialized at each iteration. Information from previous iterations hence is lost. Our more sophisticated incremental learning algorithm CDNF++ retains such information and attains a better complexity bound. Under a generalized learning model, both of our incremental algorithms require at most a polynomial number of queries in the formula size and the number of ordered variables in the target. Incremental learning on certain Boolean functions is still feasible.

To attest the performance of our incremental learning algorithms for Boolean functions, we compare with the classical algorithm in the learning-based loop invariant generation framework [11, 14, 12]. To evaluate the performance of incremental learning in typical settings, we consider a simple heuristic variable ordering from the application domain. Our incremental learning algorithms achieve up to $59.8\%$ of speedup with the heuristic ordering. To estimate the worst-case performance of incremental learning, we adopt random variable orderings instead of the heuristic ordering. Excluding one extreme case, the incremental learning algorithms perform slightly better than the classical algorithm with random orderings. Since a sensible variable ordering can often be chosen by domain experts in most applications, the artificial worst-case scenario is unlikely to happen. We therefore expect our new algorithms to prevail in practice.

In the classical CDNF learning algorithm for Boolean functions, unknown targets are Boolean functions over fixed variables [3]. It is not applicable to scenarios where unknown targets are over indefinitely many variables. Combining with predicate abstraction and decision procedures, the CDNF algorithm is used to generate invariants for annotated loops [11, 14, 12], and transition invariants for termination analysis [16]. The classical algorithm is also deployed in assume-guarantee reasoning to infer contextual assumptions automatically [5, 4]. In these applications, the CDNF algorithm is used as a black box. We propose a new learning model and develop incremental algorithms under the new model. We do not know of any learning algorithm for Boolean functions over indefinitely many variables. Abstraction techniques in regular language learning are seemingly relevant [8, 2, 10]. Recall that the $L^*$ algorithm does not apply when queries are answered nondeterministically. It is necessary to bring the learning algorithm to consistent states upon nondeterministic answers induced by abstraction. Incremental queries can introduce inconsistencies. We also have to bring the incremental learning algorithms back to consistent states. Since this work is about learning Boolean functions, it is related to [8, 2, 10] only in spirits. Many applications of the $L^*$ algorithm for regular languages have been proposed (see [9], for example).

This paper is organized as follows. After Introduction, preliminaries and notations are given in Section 2. We then review the CDNF algorithm (Section 3). Section 4 presents our technical contribution. It is followed by experimental results in Section 5. Finally, Section 6 concludes our presentation.

## 2 Preliminary

Let $\mathbb{B} = \{\bot, \top\}$ be the *Boolean domain* and $\mathbf{x} = \{x_1, x_2, \ldots, x_n, \ldots\}$ an infinite set of ordered Boolean variables. We write $\mathbf{x}_\ell$ for the subset $\{x_1, x_2, \ldots, x_\ell\}$ of $\mathbf{x}$. A *valuation* over $\mathbf{x}_\ell$ is a function from $\mathbf{x}_\ell$ to $\mathbb{B}$. The set of all valuations over $\mathbf{x}_\ell$ is denoted by $\mathit{Val}_\ell$. For any valuation $u \in \mathit{Val}_\ell$, $x \in \mathbf{x}_{\ell+1}$, and $b \in \mathbb{B}$, define

$$u[x \mapsto b](y) = \begin{cases} u(y) & \text{if } y \neq x \\ b & \text{if } y = x. \end{cases}$$

Note that $u[x_{\ell+1} \mapsto b] \in \mathit{Val}_{\ell+1}$ for every $u \in \mathit{Val}_\ell$. Let $\perp\!\!\!\perp_\ell \in \mathit{Val}_\ell$ be the valuation mapping every $x \in \mathbf{x}_\ell$ to $\bot$, and the valuation $\top\!\!\!\top_\ell \in \mathit{Val}_\ell$ mapping every $x \in \mathbf{x}_\ell$ to $\top$. The *projection* of a valuation $v$ on $\mathbf{x}_\ell$ is the valuation $u \in \mathit{Val}_\ell$ such that $u(x) = v(x)$ for every $x \in \mathbf{x}_\ell$. The symbol $\oplus$ stands for the component-wise exclusive-or operator. Thus $u \oplus \perp\!\!\!\perp_\ell = u$ for every $u \in \mathit{Val}_\ell$. If $R \subseteq \mathit{Val}_\ell$ is a set of valuations and $u \in \mathit{Val}_\ell$, we define $R \oplus u = \{r \oplus u : r \in R\}$. Thus $R \oplus \perp\!\!\!\perp_\ell = R$ for every $R \subseteq \mathit{Val}_\ell$. A *Boolean function* over $\mathbf{x}_\ell$ is a mapping from $\mathit{Val}_\ell$ to $\mathbb{B}$. Let $f$ be a Boolean function. For any valuation $u \in \mathit{Val}_\ell$, the notation $f(u)$ denotes the Boolean function obtained by assigning $x$ to $u(x)$ in $f$. Particularly, $f(u)$ is the Boolean outcome of $f$ on any valuation $u \in \mathit{Val}_\ell$ when $f$ is a Boolean function over $\mathbf{x}_\ell$. Moreover, we say $u$ is a *satisfying* valuation of the Boolean function $f$ if $f(u) = \top$; it is an *unsatisfying* valuation of $f$ if $f(u) = \bot$. When there is a satisfying valuation of a Boolean function $f$, we say $f$ is *satisfiable*. A Boolean formula $F$ over $\mathbf{x}_\ell$ *represents* a Boolean function $[\![F]\!]_\ell$ defined as follows. On any valuation $u \in \mathit{Val}_\ell$, $[\![F]\!]_\ell(u)$ is obtained by evaluating $F$ under the valuation $u$. For example, $[\![x_1 \implies x_2]\!]_2(\perp\!\!\!\perp_2) = \top$.

A *literal* is a Boolean variable or its negation. A *term* is a conjunction of literals. A *clause* is a disjunction of literals. A Boolean formula is in *disjunctive normal form (DNF)* if it is a disjunction of terms. A Boolean formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses. A formula in CNF (DNF) is a *CNF* (*DNF*, respectively) *formula*. A Boolean formula is in *conjunctive disjunctive normal form (CDNF)* if it is a conjunction of DNF formulae. A formula in CDNF is a *CDNF formula*.

## 3 The CDNF Algorithm

The CDNF algorithm is an exact learning algorithm for Boolean functions over $\mathbf{x}_n$ [3]. Suppose $f$ is an unknown *target* Boolean function over $\mathbf{x}_n$. The learning algorithm infers a CDNF formula representing $f$ by interacting with a teacher. The *teacher* is responsible for answering two types of queries.

- *Membership queries* $\mathit{MEM}_n(v)$ with $v \in \mathit{Val}_n$. If $f(v) = \top$, the teacher answers $\mathit{YES}$; otherwise, she answers $\mathit{NO}$.
- *Equivalence queries* $\mathit{EQ}_n(F)$ with a Boolean formula $F$ over $\mathbf{x}_n$ as the *conjecture*. If $[\![F]\!]_n = f$, the teacher answers $\mathit{YES}$. Otherwise, the teacher returns a *counterexample* $v \in \mathit{Val}_n$ such that $[\![F]\!]_n(v) \neq f(v)$.

Let $v \in \mathit{Val}_n$ be a valuation and $F$ a Boolean formula over $\mathbf{x}_n$. We write $\mathit{MEM}_n(v) \to Y$ and $\mathit{EQ}_n(F) \to Z$ to denote that $Y$ and $Z$ are the answers to the membership query on $v$ and equivalence query on $F$, respectively.

1 $t \leftarrow 0$;
2 $EQ_n(\mathsf{true}) \rightarrow v$;
3 **if** $v$ *is YES* **then return** true;
4 $t \leftarrow t + 1$;
5 $H_t, R_t, a_t \leftarrow \mathsf{false}, \emptyset, v$ ; // assert $MEM_n(a_t) \rightarrow NO$
6 $EQ_n(\wedge_{i=1}^{t} H_i) \rightarrow v$;
7 **if** $v$ *is YES* **then return** $\wedge_{i=1}^{t} H_i$;
8 $I \leftarrow \{i : [\![H_i]\!]_n(v) = \bot\}$;
9 **if** $I = \emptyset$ **then goto** 4;
10 **foreach** $i \in I$ **do**
11 $r \leftarrow walkTo(n, a_i, v)$ ; // assert $MEM_n(r) \rightarrow YES$
12 $R_i \leftarrow R_i \cup \{r\}$;
13 **end**
14 **foreach** $i = 1, \ldots, t$ **do** $H_i \leftarrow M_{\text{DNF}}(R_i \oplus a_i)(\mathbf{x}_n \oplus a_i)$;
15 **goto** 6;

**Algorithm 1:** The CDNF Algorithm

We reprint the CDNF algorithm in Algorithm 1. In the algorithm, conjectures in equivalence queries are always CDNF formulae. The variable $t$ maintains the number of DNF formulae in the current conjecture. Initially, the variable $t$ is set to 0. The conjecture is hence degenerated to true (line 2, Algorithm 1).

Three variables keep track of each DNF formula in the conjecture. For the $i$-th DNF formula, the variable $a_i$ is a valuation over $\mathbf{x}_n$, the variable $R_i$ is a set of valuations over $\mathbf{x}_n$, and the variable $H_i$ is a DNF formula over $\mathbf{x}_n$. The $i$-th DNF formula $H_i$ is derived from $a_i$ and $R_i$ by $M_{\text{DNF}}$ (line 14, Algorithm 1):

$$M_{\text{DNF}}(s) = \begin{cases} \bigwedge_{s(x_i)=\top} x_i & \text{if } s \neq \perp\!\!\!\perp_n \\ \mathsf{true} & \text{otherwise} \end{cases} \qquad M_{\text{DNF}}(S) = \begin{cases} \bigvee_{s \in S} M_{\text{DNF}}(s) & \text{if } S \neq \emptyset \\ \mathsf{false} & \text{otherwise} \end{cases}$$

For instance, $M_{\text{DNF}}(\{\perp\!\!\!\perp_2, \top\!\!\!\top_2\}) = M_{\text{DNF}}(\perp\!\!\!\perp_2) \vee M_{\text{DNF}}(\top\!\!\!\top_2) = \mathsf{true} \vee (x_1 \wedge x_2)$.

When a new DNF formula is added to the conjecture, the variable $R_t$ is the empty set and the variable $H_t$ is set to false accordingly (line 5, Algorithm 1). Conjectures in equivalence queries are conjunctions of $H_i$'s.

In order to understand our extensions to the CDNF learning algorithm, we give a new characterization of variables associated with the $i$-th DNF formulae in Algorithm 1. Note that $a_i$ was defined when the $i$-th DNF formula was created and added to the conjecture (line 5, Algorithm 1). It is not hard to see that $a_i$ is a valuation with $MEM_n(a_i) \rightarrow NO$. First, $a_1$ was a counterexample to the equivalence query $EQ_n(\mathsf{true})$. We have $MEM_n(a_1) \rightarrow NO$. For $i > 1$, observe that $a_i$ was the counterexample to the equivalence query $EQ_n(\wedge_{j=1}^{i-1} H_j)$ (line 6, Algorithm 1). Furthermore, $a_i$ was added when the set $\{j < i : [\![H_j]\!]_n(a_i) = \bot\}$ was empty (line 9, Algorithm 1). Since $[\![\wedge_{j=1}^{i-1} H_j]\!]_n(a_i) = \top$ and $EQ_n(\wedge_{j=1}^{i-1} H_j) \rightarrow a_i$, we have $MEM_n(a_i) \rightarrow NO$.

The valuations in $R_i$ can be characterized as easily. When the counterexample $v$ to the equivalence query $EQ_n(\wedge_{i=1}^{t} H_i)$ is returned (line 6, Algorithm 1), the CDNF algorithm checks if the set $\{i : [\![H_i]\!]_n(v) = \bot\}$ is empty (line 9, Algorithm 1). If not,

we have $[\![\wedge_{i=1}^{t} H_i]\!]_n(v) = \bot$. Thus $MEM_n(v) \to YES$ for $v$ is a counterexample to $EQ_n(\wedge_{i=1}^{t} H_i)$. For each $i$ such that $[\![H_i]\!]_n(v) = \bot$, the result of $walkTo(n, a_i, v)$ is added to $R_i$ (line 12, Algorithm 1). Algorithm 2 gives the details of $walkTo(\ell, a, v)$.

**Input**: $\ell \in \mathbb{N} : 1 \leq \ell$; $a \in Val_\ell : MEM_\ell(a) \to NO$; $v \in Val_\ell : MEM_\ell(v) \to YES$
**Output**: $r \in Val_\ell : MEM_\ell(r) \to YES$

```
1  r ← v;
2  k ← 1;
3  while k ≤ ℓ do
4  |   if r(x_k) = a(x_k) then k ← k + 1;
5  |   else
6  |   |   r(x_k) ← a(x_k);
7  |   |   if MEM_ℓ(r) → NO then
8  |   |   |   r(x_k) ← ¬a(x_k);
9  |   |   |   k ← k + 1;
10 |   |   else k ← 0;
11 |
12 end
13 return r;
```

**Algorithm 2:** $walkTo(\ell, a, v)$

The algorithm $walkTo(\ell, a, v)$ finds an $x \in \mathbf{x}_\ell$ with $v(x) \neq a(x)$ and flips the value of $v(x)$. If the new valuation yields $YES$ on a membership query, it continues flipping other values of $v$ different from $a$. Otherwise, the algorithm reverts to the old value of $v(x)$ and flips another value. Roughly, $walkTo(\ell, a, v)$ computes a valuation $r \in Val_n$ closest to $a$ such that $MEM_n(r) \to YES$. Define

$$N_\ell(a, r) = \{w \in Val_\ell : w = r[x \mapsto a(x)] \text{ where } x \in \mathbf{x}_\ell \text{ and } r(x) \neq a(x)\}.$$

Each valuation in $N_\ell(a, r)$ is obtained by flipping the value of exactly one $x \in \mathbf{x}_\ell$ on $r$ with $r(x) \neq a(x)$. Each valuation in $N_\ell(a, r)$ is thus closer to $a$ than $r$. The following lemma summaries Algorithm 2:

**Lemma 1.** *Let $a, v \in Val_\ell$ $(1 \leq \ell)$ be that $MEM_\ell(a) \to NO$ and $MEM_\ell(v) \to YES$. Assume $r = walkTo(\ell, a, v)$ (Algorithm 2). Then $MEM_\ell(r) \to YES$, and $MEM_\ell(w) \to NO$ for every $w \in N_\ell(a, r)$.*

*Proof.* Note that $k$ is set to 0 when flipping gets a $YES$ from the membership query on the valuation (line 10, Algorithm 2). Hence for each $j \in \{0, \ldots, \ell - 1\}$, we have either

- $r(x_j) = a(x_j)$; or
- $r(x_j) \neq a(x_j)$ but flipping $r(x_j)$ would result in $MEM_\ell(r) \to NO$.

Suppose $w \in N_\ell(a, r)$. Assume $w(x_k) = a(x_k) \neq r(x_k)$ for some $k \in \{0, \ldots, \ell - 1\}$. Then $w$ is derived from $r$ by flipping the value of $r(x_k)$. Hence $MEM_\ell(w) \to NO$ by line 7 Algorithm 2. □

Recall that $R_i$ consists of the result of $walkTo(n, a_i, v)$ where $MEM_n(a_i) \to NO$ and $MEM_n(v) \to YES$. Thus $MEM_n(r) \to YES$ for every $r \in R$; $MEM_n(w) \to NO$ for every $r \in R$ and $w \in N_n(a_i, r)$ (Lemma 1). We characterize the pairs $(a, R)$'s maintained in the learning algorithm with the following definition:

**Definition 1.** *For $a \in Val_n$ and $R \subseteq Val_n$, define the property $\Gamma(a, R)$ by*

1. *$MEM_n(a) \to NO$;*
2. *$MEM_n(r) \to YES$ for every $r \in R$; and*
3. *$MEM_n(w) \to NO$ for every $r \in R$ and $w \in N_n(a, r)$.*

Suppose $[\![\neg x_1 \vee \neg x_2]\!]_2$ is the target Boolean function over $\mathbf{x}_2$ as an example. Let $r(x_1) = \bot$ and $r(x_2) = \top$. We have $\Gamma(\top\!\!\top_2, \{r\})$ but not $\Gamma(\top\!\!\top_2, \{\bot\!\!\bot_2\})$.

The following lemma states that $\Gamma(a_i, R_i)$ holds for $1 \leq i \leq t$ in the CDNF algorithm. We call $(a, R)$ a *speculative support* when $\Gamma(a, R)$ holds.

**Lemma 2.** *At line 6 of Algorithm 1, $\Gamma(a_i, R_i)$ holds for every $1 \leq i \leq t$.*

*Proof.* When $a_i$ is added at line 5 Algorithm 1, the set

$$\{H_j : j < i \text{ and } [\![H_j]\!]_(a_i) = \bot\}$$

is empty (line 9 Algorithm 1). Hence $[\![\wedge_{j=1}^{i-1} H_j]\!]_(a_i) = \top$. Recall that $a_i$ is a counterexample to $EQ_n(\wedge_{j=1}^{i-1} H_j)$. Thus $MEM_n(a_i) \to NO$.

By Lemma 1 and the fact that $R_i$ consists of the outputs of Algorithm 2, we have $MEM_n(r) \to YES$ for every $r \in R_i$. Moreover, $MEM_n(w) \to NO$ for every $r \in R_i$ and $w \in N_n(a, r)$. □

The *size* of a DNF formula is the number of terms in the formula; the *size* of a CNF formula is the number of clauses in it. Let $f$ be a Boolean function over $\mathbf{x}_n$. The *DNF size* of $f$ (denoted by $|f|_{\text{DNF}}$) is the minimal size over all DNF formulae representing $f$; the *CNF size* of $f$ (denoted by $|f|_{\text{CNF}}$) is the minimal size of all CNF formulae representing $f$. The number of speculative supports and the size of $R$ in each speculative support $(a, R)$ give the following bounds.

**Theorem 1 ([3]).** *Let $f$ be an unknown target Boolean function over $\mathbf{x}_n$. The CDNF algorithm (Algorithm 1) infers $f$ within $O(n^2|f|_{CNF}|f|_{DNF})$ membership and $O(|f|_{CNF}|f|_{DNF})$ equivalence queries.*

Note that the complexity of the CDNF algorithm is a polynomial in the size of the variable set $\mathbf{x}_n$. If all but one variables in $\mathbf{x}_n$ are redundant, the learning algorithm still requires $O(n^2)$ membership queries to infer the target.

## 4 Incremental Learning

The CDNF algorithm infers an unknown target among Boolean functions over a fixed number of variables. It is not applicable to scenarios where targets are Boolean functions over indefinitely many variables. Moreover, the complexity of the CDNF algorithm is a polynomial in the number of given variables. It can be unexpectedly inefficient when many variables are redundant in the unknown target.

It appears that these issues could be resolved by invoking the CDNF algorithm iteratively. A naive incremental learning algorithm adopts the classical learning algorithm to infer the unknown target as a Boolean function over $\mathbf{x}_\ell$ at iteration $\ell$. If it succeeds, the naive algorithm reports the inferred result. Otherwise, the naive algorithm increments the number of variables and invokes the CDNF algorithm to infer the unknown target as a Boolean function over $\mathbf{x}_{\ell+1}$. The naive approach however has two problems.

The first problem is to answer queries. Recall that the teacher knows a target Boolean function over, say, $\mathbf{x}_m$. At iteration $\ell$, the naive incremental algorithm infers the unknown target as a Boolean function over $\mathbf{x}_\ell$. It thus makes queries on valuations and conjectures over $\mathbf{x}_\ell$. Yet the target Boolean function is over $\mathbf{x}_m$. It is unclear how the teacher answers queries at iteration $\ell$ when $\ell \neq m$. A new learning model where the teacher answers such queries is necessary for learning Boolean functions incrementally.

The other problem of the naive approach is its inefficiency. Recall that the complexity of the CDNF algorithm depends on the CNF and DNF sizes of the unknown target. Since targets are arbitrary, $\Omega(2^\ell)$ queries are needed to decide whether the learning algorithm fails to infer the target at iteration $\ell$. Deciding failures of inference requires an exponential number of queries at each iteration. Naively adopting the CDNF algorithm would be very inefficient compared to the classical learning algorithm. A more sophisticated mechanism to identify failures of inference at each iteration is indispensable.

For the first problem, we generalize the classical learning model to enable the teacher answering queries at all iterations (Section 4.1). To address the second problem, we develop a criterion for determining failures of inference dynamically and use it in our simple incremental learning algorithm (Section 4.2). A sophisticated incremental algorithm with an economical management of information is presented in Section 4.3.

### 4.1 Incremental Teacher

Assume a target Boolean function $f$ over a finite subset of $\mathbf{x}$. In our incremental learning model, an *incremental teacher* should answer the following queries:

- *Incremental membership queries* $MEM_\ell(u)$ with $u \in Val_\ell$. If $f(u)$ is satisfiable, the incremental teacher answers $YES$; otherwise, she answers $NO$.
- *Incremental non-membership queries* $\overline{MEM}_\ell(u)$ with $u \in Val_\ell$. If $\neg f(u)$ is satisfiable, the incremental teacher answers $YES$; otherwise, $NO$.
- *Incremental equivalence queries* $EQ_\ell(G)$ with a Boolean formula $G$ over $\mathbf{x}_\ell$. If $[\![G]\!]_\ell = f$, the incremental teacher answers $YES$. Otherwise, she returns the projection of a valuation $v \in Val_{\mathbf{x}}$ on $\mathbf{x}_\ell$ where $[\![G]\!]_\ell(v) \neq f(v) \in \mathbb{B}$.

*Example*. Let $f = x_1 \oplus x_2$. On incremental queries $MEM_1(\perp\!\!\!\perp_1)$ and $\overline{MEM}_1(\perp\!\!\!\perp_1)$, the incremental teacher answers $YES$. Similarly, the incremental teacher answers $YES$ on incremental queries $MEM_1(\top\!\!\!\top_1)$ and $\overline{MEM}_1(\top\!\!\!\top_1)$. On incremental equivalence queries $EQ_1(\mathsf{true})$ or $EQ_1(\mathsf{false})$, $\top$ is a counterexample.

Incremental queries allow a learning algorithm to acquire (incomplete) information about the unknown target function. Intuitively, the answer to an incremental membership query on a valuation reveals whether a completion of the valuation gives a satisfying valuation; the answer to an incremental non-membership query shows whether

a completion gives an unsatisfying valuation. Incremental equivalence queries check whether the target is equivalent to a Boolean formula over specified variables. If not, a valuation differentiates the conjecture and the target. The projection of such a valuation on specified variables is returned as a counterexample. The following lemma is useful.

**Lemma 3.** *Assume a target Boolean function over $\mathbf{x}_m$ and $1 \leq \ell \leq m$.*

1. *For any valuation $v \in Val_m$, $MEM_m(v) \rightarrow YES$ iff $\overline{MEM}_m(v) \rightarrow NO$.*
2. *For any Boolean formula $G$ and valuation $u$ over $\mathbf{x}_\ell$, $[\![G]\!]_\ell(u) = \bot$ and $EQ_\ell(G) \rightarrow u$ imply $MEM_\ell(u) \rightarrow YES$.*
3. *For any Boolean formula $G$ and valuation $u$ over $\mathbf{x}_\ell$, $[\![G]\!]_\ell(u) = \top$ and $EQ_\ell(G) \rightarrow u$ imply $\overline{MEM}_\ell(u) \rightarrow YES$.*

### 4.2 The CDNF+ Algorithm

Suppose that the CDNF algorithm is inferring an unknown target as a Boolean function over $\mathbf{x}_\ell$ at iteration $\ell$. We check if the classical algorithm will fail at this iteration. If so, we abort and re-instantiate the CDNF algorithm to infer the unknown target as a Boolean function over $\mathbf{x}_{\ell+1}$ at the next iteration. To determine failures of inference, recall that the CDNF algorithm is exact. If the unknown target is indeed a Boolean function over $\mathbf{x}_\ell$, the classical algorithm will infer it. It suffices to check whether the target is a Boolean function over $\mathbf{x}_\ell$ to determine whether the iteration $\ell$ will fail.

In order to detect whether the unknown target is a Boolean function over $\mathbf{x}_\ell$, observe that a function cannot have two different outcomes on one input. When the target is a Boolean function over $\mathbf{x}_\ell$, $MEM_\ell(u) \rightarrow YES$ if and only if $\overline{MEM}_\ell(u) \rightarrow NO$ for every $u \in Val_\ell$ (Lemma 3). Therefore, the unknown target is not a Boolean function over $\mathbf{x}_\ell$ if $MEM_\ell(u) \rightarrow YES$ and $\overline{MEM}_\ell(u) \rightarrow YES$ for some $u \in Val_\ell$. This simple observation motivates the following definition:

**Definition 2.** *A valuation $u \in Val_\ell$ $(1 \leq \ell)$ is* conflicting *if $MEM_\ell(u) \rightarrow YES$ and $\overline{MEM}_\ell(u) \rightarrow YES$.*

The following lemma follows immediately from Lemma 3.

**Lemma 4.** *For any target Boolean function over a finite subset of $\mathbf{x}$, the target Boolean function is not over $\mathbf{x}_\ell$ if there is a conflicting valuation over $\mathbf{x}_\ell$.*

*Example (continued).* Recall that $\bot$ is a counterexample to both $EQ_1(\mathsf{false})$ and $EQ_1(\mathsf{true})$. By Lemma 3, $MEM_1(\bot) \rightarrow YES$ and $\overline{MEM}_1(\bot) \rightarrow YES$. Hence the unknown target is not a Boolean function over $\mathbf{x}_1$.

Our first incremental learning algorithm is now clear. We parameterize the CDNF algorithm by the number of ordered variables. At iteration $\ell$, we apply the parameterized CDNF algorithm and infer the unknown target as a Boolean function over $\mathbf{x}_\ell$. If a conflicting valuation is observed, we increment $\ell$ and move to the next iteration. Algorithm 3 shows the parameterized CDNF algorithm. Note that incremental equivalence queries are invoked in the parameterized algorithm. Similarly, incremental membership queries are used in the algorithm $walkTo(\ell, a, v)$ (Algorithm 2).

We give a parameterized generalization of $\Gamma(a, R)$ in Definition 3.

**Input**: $\ell \in \mathbb{N} : 1 \leq \ell$

1 $t \leftarrow 0$;
2 $EQ_\ell(\mathsf{true}) \rightarrow v$;
3 **if** $v$ *is YES* **then return** true;
4 $t \leftarrow t + 1$;
5 $H_t, R_t, a_t \leftarrow \mathsf{false}, \emptyset, v$ ; // assert $\overline{MEM}_\ell(a_t) \rightarrow YES$
6 $EQ_\ell(\wedge_{i=1}^{t} H_i) \rightarrow v$;
7 **if** $v$ *is YES* **then return** $\wedge_{i=1}^{t} H_i$;
8 $I \leftarrow \{i : [\![H_i]\!]_\ell(v) = \bot\}$;
9 **if** $I = \emptyset$ **then goto** 4;
10 **foreach** $i \in I$ **do**
11 $\quad r \leftarrow walkTo(\ell, a_i, v)$ ; // assert $MEM_\ell(r) \rightarrow YES$
12 $\quad$ **if** $a_i = r$ **then raise** *conflict-found*;
13 $\quad R_i \leftarrow R_i \cup \{r\}$;
14 **end**
15 **foreach** $i = 1, \ldots, t$ **do** $H_i \leftarrow M_{\mathrm{DNF}}(R_i \oplus a_i)(\mathbf{x}_\ell \oplus a_i)$;
16 **goto** 6;

**Algorithm 3:** $\wp$CDNF ($\ell$)

**Definition 3.** *For $a \in Val_\ell$ $(1 \leq \ell)$ and $R \subseteq Val_\ell$, define $\Delta_\ell(a, R)$ by*

1. $\overline{MEM}_\ell(a) \rightarrow YES$*;*
2. $MEM_\ell(r) \rightarrow YES$ *for every* $r \in R$*;*
3. $MEM_\ell(w) \rightarrow NO$ *for every* $r \in R$ *and* $w \in N_\ell(a, r)$.

The following lemma states that $\Delta_\ell(a_i, R_i)$ holds for $1 \leq i \leq t$ in the parameterized CDNF algorithm. Its proof is a generalization of those in Lemma 2. We call $(a, R)$ a *speculative support with parameter* $\ell$ when $\Delta_\ell(a, R)$ holds.

**Lemma 5.** *At line 6 of Algorithm 3, $\Delta_\ell(a_i, R_i)$ holds for every $1 \leq i \leq t$.*

In order to decide conflicting valuations, recall that $(a_i, R_i)$'s are speculative supports with parameter $\ell$. We have $\overline{MEM}_\ell(a_i) \rightarrow YES$ and $MEM_\ell(r) \rightarrow YES$ for every $r \in R_i$ (Lemma 5 and 1). If furthermore $a_i = r$, $a_i$ is conflicting. By Lemma 4, the unknown target is not a Boolean function over $\mathbf{x}_\ell$. We abort the parameterized algorithm by raising an exception (line 12, Algorithm 3).

1 $\ell \leftarrow 1$;
2 **while** $\top$ **do**
3 $\quad$ **try**
4 $\quad\quad G = \wp\mathrm{CDNF}(\ell)$
5 $\quad$ **with** *conflict-found* $\Longrightarrow \ell \leftarrow \ell + 1$;
6 **end**
7 **return** $G$;

**Algorithm 4:** The CDNF+ Algorithm

Algorithm 4 gives our simple incremental learning algorithm. The CDNF+ algorithm starts from the variable $\ell$ equal to one. At iteration $\ell$, it invokes the parameterized algorithm $\wp$CDNF with parameter $\ell$ to infer the unknown target as a Boolean function over $\mathbf{x}_\ell$. If the parameterized algorithm infers the target, our simple algorithm terminates successfully. If the parameterized learning algorithm raises the exception *conflict-found*, the simple algorithm increments the variable $\ell$ and reiterates. The complexity of the CDNF+ algorithm follows from Theorem 1 and the number of iterations.

**Theorem 2.** *Let $f$ be an unknown target Boolean function over a finite subset of $\mathbf{x}$. The CDNF+ algorithm (Algorithm 4) infers $f$ in $O(m^3|f|_{CNF}|f|_{DNF})$ incremental membership and $O(m|f|_{CNF}|f|_{DNF})$ incremental equivalence queries where $m$ is the least number such that $f$ is a Boolean function over $\mathbf{x}_m$.*

*Proof.* A conflict must be observed within $O(n^2|f|_{CNF}|f|_{DNF})$ membership queries and $O(|f|_{CNF}|f|_{DNF})$ equivalence queries at iteration $\ell < n$. Hence the CDNF+ algorithm requires at most $O(n^3|f|_{CNF}|f|_{DNF})$ membership queries and $O(n|f|_{CNF}|f|_{DNF})$ equivalence queries in total.

The CDNF+ algorithm does not presume a fixed set of variables. It is hence applicable to scenarios where unknown targets are over indefinitely many variables. Moreover, the complexity of the CDNF+ algorithm depends on the number of ordered variables in the unknown target. If the target is a Boolean function over $\mathbf{x}_1$, the CDNF+ algorithm will infer the target within $O(|f|_{\text{CNF}}|f|_{\text{DNF}})$ incremental membership queries. The classical learning algorithm in contrast needs $O(n^2|f|_{\text{CNF}}|f|_{\text{DNF}})$ membership queries if it infers the unknown target as a Boolean function over $\mathbf{x}_n$. The performance of the CDNF+ algorithm however depends on variable orderings and how incremental membership queries are resolved in practice. Section 5 evaluates these issues.

### 4.3 The CDNF++ Algorithm

We can actually do better than the CDNF+ algorithm. Observe that the simple incremental learning algorithm restarts the learning process at each iteration. All information from previous iterations known to the incremental algorithm is lost. The parameterized CDNF+ algorithm has to infer the unknown target from scratch. This is apparently not an economical management of information.

To retain the information obtained in previous iterations, we reuse parameterized speculative supports in each iteration. Each speculative support $(a, R)$ with parameter $\ell$ satisfies the property $\Delta_\ell(a, R)$ at iteration $\ell$ (Lemma 5). We compute a speculative support $(a^+, R^+)$ with parameter $\ell+1$ from a speculative support $(a, R)$ with parameter $\ell$. After new parameterized speculative supports are constructed, we initiate the parameterized CDNF algorithm with the extended parameterized speculative supports and the conjecture derived from them. Information from previous iterations is thus retained.

Consider a speculative support $(a, R)$ with parameter $\ell$ and a speculative support $(a^+, R^+)$ with parameter $\ell + 1$. We have $a \in Val_\ell$ and $a^+ \in Val_{\ell+1}$. Similarly, $R \subseteq Val_\ell$ and $R^+ \subseteq Val_{\ell+1}$. Each valuation in a speculative support with parameter $\ell$ is only short of the Boolean assignment to the variable $x_{\ell+1}$. To construct $(a^+, R^+)$

from $(a, R)$, it suffices to extend the valuation $a$ and every valuation over $\mathbf{x}_\ell$ in $R$ by an assignment to $x_{\ell+1}$. To simplify the notation, we use the shorthand $u^{+b}$ for $u[x_{\ell+1} \mapsto b]$ where $u \in \mathit{Val}_\ell$ and $b \in \mathbb{B}$. The following lemma follows from the definition.

**Lemma 6.** *Let $u \in \mathit{Val}_\ell$ $(1 \leq \ell)$ be a valuation over $\mathbf{x}_\ell$.*

1. *If $\mathit{MEM}_\ell(u) \to \mathit{YES}$, $\mathit{MEM}_{\ell+1}(u^{+\bot}) \to \mathit{YES}$ or $\mathit{MEM}_{\ell+1}(u^{+\top}) \to \mathit{YES}$.*
2. *If $\overline{\mathit{MEM}}_\ell(u) \to \mathit{NO}$, $\overline{\mathit{MEM}}_{\ell+1}(u^{+\bot}) \to \mathit{NO}$ and $\overline{\mathit{MEM}}_{\ell+1}(u^{+\top}) \to \mathit{NO}$.*
3. *If $\overline{\mathit{MEM}}_\ell(u) \to \mathit{YES}$, $\overline{\mathit{MEM}}_{\ell+1}(u^{+\bot}) \to \mathit{YES}$ or $\overline{\mathit{MEM}}_{\ell+1}(u^{+\top}) \to \mathit{YES}$.*

Algorithm 5 explicates the construction of $(a^+, R^+)$ from $(a, R)$ where $\Delta_\ell(a, R)$ holds. It starts by extending $a$. Recall that $\overline{\mathit{MEM}}_\ell(a) \to \mathit{YES}$. We can always find an extension $a^+$ with $\overline{\mathit{MEM}}_{\ell+1}(a^+)$ (Lemma 6). For the set $R^+ \subseteq \mathit{Val}_{\ell+1}$, the construction is not more difficult. We simply extend every valuation in $R$ so that the extension yields $\mathit{YES}$ on an incremental membership query.

**Input**: $\ell \in \mathbb{N} : 1 \leq \ell$; $a \in \mathit{Val}_\ell : \overline{\mathit{MEM}}_\ell(a) \to \mathit{YES}$; $R \subseteq \mathit{Val}_\ell : \mathit{MEM}_\ell(r) \to \mathit{YES}$ for every $r \in R$
**Output**: $a^+ \in \mathit{Val}_{\ell+1} : \overline{\mathit{MEM}}_{\ell+1}(a^+) \to \mathit{YES}$; $R^+ \subseteq \mathit{Val}_{\ell+1} : \mathit{MEM}_{\ell+1}(r^+) \to \mathit{YES}$ for every $r^+ \in R^+$
`// assert` $\Delta_\ell(a, R)$
1 $b \leftarrow$ **if** $\overline{\mathit{MEM}}_{\ell+1}(a^{+\bot}) \to \mathit{YES}$ **then** $\bot$ **else** $\top$;
2 $a^+ \leftarrow a^{+b}$;
3 $R^+ \leftarrow \emptyset$;
4 **foreach** $r \in R$ **do**
5 $\quad c \leftarrow$ **if** $\mathit{MEM}_{\ell+1}(r^{+b}) \to \mathit{YES}$ **then** $b$ **else** $\neg b$;
6 $\quad R^+ \leftarrow R^+ \cup \{r^{+c}\}$;
7 **end**
`// assert` $\Delta_{\ell+1}(a^+, R^+)$
8 **return** $(a^+, R^+)$;

**Algorithm 5:** $\mathit{extendSupport}(\ell, a, R)$

The following lemma states that the construction in Algorithm 5 is indeed correct. The only non-trivial part is to show that $N_{\ell+1}(a^+, r^+)$ consists of valuations yielding $\mathit{NO}$ on incremental membership queries for every $r^+ \in R^+$.

**Lemma 7.** *Let $a \in \mathit{Val}_\ell$ $(1 \leq \ell)$, $R \subseteq \mathit{Val}_\ell$, and $(a^+, R^+) = \mathit{extendSupport}(\ell, a, R)$ (Algorithm 5). If $\Delta_\ell(a, R)$, then $\Delta_{\ell+1}(a^+, R^+)$.*

*Proof.* $\overline{\mathit{MEM}}_\ell(a) \to \mathit{YES}$ by assumption. Thus $\overline{\mathit{MEM}}_{\ell+1}(a^{+\bot}) \to \mathit{YES}$ or $\overline{\mathit{MEM}}_{\ell+1}(a^{+\top}) \to \mathit{YES}$ (Lemma 6). By line 1, Algorithm 5, we have $\overline{\mathit{MEM}}_{\ell+1}(a^+) \to \mathit{YES}$ for $a^+ = a^{+b}$.

$\mathit{MEM}_\ell(r) \to \mathit{YES}$ for every $r \in R$ by assumption. Hence $\mathit{MEM}_{\ell+1}(r^{+\bot}) \to \mathit{YES}$ or $\mathit{MEM}_{\ell+1}(r^{+\top}) \to \mathit{YES}$ (Lemma 6). For every $r^+ \in R^+$, we have $\mathit{MEM}_{\ell+1}(r^+) \to \mathit{YES}$ by line 5 Algorithm 5 since $r \in R$ and $r^+ = r^{+c}$.

Finally, assume $a^+ = a^{+b}$ and $r^+ = r^{+c} \in R^+$. Let $w^+ \in N_{\ell+1}(a^+, r^+)$. In the remaining of the proof, we write $w$ for $w^+ \downarrow_{\mathbf{x}_\ell}$. We show $MEM_{\ell+1}(w^+) \to NO$. There are two cases.

- $b = c$. Since $w^+ \in N_{\ell+1}(a^+, r^+)$ and $r^+(x_{\ell+1}) = a^+(x_{\ell+1})$, $w \in N_\ell(a, r)$. We have $MEM_{\ell+1}(w^+) \to NO$ for $MEM_\ell(w) \to NO$ (Lemma 6).
- $b \neq c$. There are two subcases.
  - $w^+(x_{\ell+1}) = r^+(x_{\ell+1})$. Hence $w \in N_\ell(a, r)$ and $MEM_\ell(w) \to NO$. We have $MEM_{\ell+1}(w^+) \to NO$ (Lemma 6);
  - $w^+(x_{\ell+1}) \neq r^+(x_{\ell+1})$. Hence $w = r$ and $w^+ = w^{+b} = r^{+b}$. By line 5 Algorithm 5, we have $MEM_{\ell+1}(r^{+b}) \to NO$ as required.

□

With extended parameterized speculative supports, it is now straightforward to design our incremental learning algorithm (Algorithm 6). Similar to the simple incremental algorithm, the CDNF++ algorithm infers unknown target Boolean functions iteratively. At each iteration, it first proceeds as the parameterized CDNF algorithm. If the parameterized algorithm is able to infer the unknown target at iteration $\ell$, our incremental algorithm terminates successfully and reports the result.

When the CDNF++ algorithm detects a conflicting valuation, it constructs extended parameterized speculative supports with Algorithm 5 (line 14, Algorithm 6). After extended parameterized speculative supports are obtained, the CDNF++ algorithm derives a new conjecture from them and enters the next iteration (line 19, Algorithm 6). The following theorem is proved by bounding the number of parameterized speculative supports and the size of $R$ in each parameterized speculative support $(a, R)$.

**Theorem 3.** *Let $f$ be an unknown target Boolean function over a finite subset of $\mathbf{x}$. The CDNF++ algorithm (Algorithm 6) infers $f$ in $O(m^2|f|_{CNF}|f|_{DNF})$ incremental membership, $O(m|f|_{CNF})$ incremental non-membership, and $O(|f|_{CNF}|f|_{DNF})$ incremental equivalence queries where $m$ is the least number that $f$ is a Boolean function over $\mathbf{x}_m$.*

*Proof.* Let $F = \bigwedge_{i=0}^{t} H_i$ be the CDNF formula inferred by the CDNF algorithm on $f$. $t$ is bounded above by $|f|_{\text{CNF}}$. There are at most $|f|_{\text{DNF}}$ terms in all DNF formulae $H_i$'s. Moreover, $O(1)$ membership queries are needed to extend a speculative support. Hence the CDNF++ algorithm needs additional $O(n(|f|_{\text{CNF}} + |f|_{\text{DNF}}))$ membership queries for there are $n$ iterations.

Compared with the simple incremental learning algorithm, the CDNF++ algorithm improves linearly in the numbers of incremental membership and equivalence queries. In exchange, the sophisticated algorithm makes non-membership queries to extend parameterized speculative supports. Again, the performance of the CDNF++ algorithm depends on the order of variables and the efficiency of incremental query resolution. We give an assessment in the next section.

1 $\ell \leftarrow 1$;
2 $t \leftarrow 0$;
3 $EQ_\ell(\mathsf{true}) \rightarrow v$;
4 **if** $v$ *is YES* **then return** true;
5 $t \leftarrow t + 1$;
6 $H_t, R_t, a_t \leftarrow \mathsf{false}, \emptyset, v$ ; // assert $\overline{MEM}_\ell(a_t) \rightarrow YES$
7 $EQ_\ell(\wedge_{i=1}^{t} H_i) \rightarrow v$;
8 **if** $v$ *is YES* **then return** $\wedge_{i=1}^{t} H_i$;
9 $I \leftarrow \{i : [\![H_i]\!]_\ell(v) = \bot\}$;
10 **if** $I = \emptyset$ **then goto** 5;
11 **foreach** $i \in I$ **do**
12 $\quad r \leftarrow walkTo(\ell, a_i, v)$ ; // assert $MEM_\ell(r) \rightarrow YES$
13 $\quad$ **if** $a_i = r$ **then**
14 $\quad\quad$ **foreach** $i = 1, \ldots, t$ **do** $(a_i, R_i) \leftarrow extendSupport(a_i, R_i)$;
15 $\quad\quad \ell \leftarrow \ell + 1$;
16 $\quad\quad$ **goto** 19
17 $\quad R_i \leftarrow R_i \cup \{r\}$;
18 **end**
19 **foreach** $i = 1, \ldots, t$ **do** $H_i \leftarrow M_{\text{DNF}}(R_i \oplus a_i)(\mathbf{x}_\ell \oplus a_i)$;
20 **goto** 7;

**Algorithm 6:** The CDNF++ Algorithm

## 5 Experiments

We apply our incremental learning algorithms to the learning-based framework for loop invariant generation [11]. Let $\{\ \delta\ \}$ while $\kappa$ do $S$ $\{\ \epsilon\ \}$ be an annotated loop with the *pre-condition* $\delta$, the *post-condition* $\epsilon$, and the *loop guard* $\kappa$. A *loop invariant* $\iota$ verifying the annotated loop is a quantifier-free formula such that $\delta \implies \iota$, $\iota \implies \epsilon \vee \kappa$, and $\iota \wedge \kappa \implies wp(S, \iota)$, where $wp(S, \iota)$ denotes the weakest precondition of $\iota$ for $S$.

The learning-based framework for loop invariant generation applies predicate abstraction [17, 7] and adopts the CDNF algorithm [3] to infer the abstraction of a loop invariant for a given annotated loop. Using an SMT solver [6, 15], a randomized mechanical teacher is devised to answer queries from the learning algorithm. Suppose $n$ atomic predicates are used in the abstraction. Consider a membership query $MEM_n(v)$ with $v \in Val_n$. If the quantifier-free formula corresponding to the valuation $v$ is stronger than $\delta$, it must be stronger than any loop invariant $\iota$ for $\delta \implies \iota$. The mechanical teacher hence answers $YES$ to the membership query $MEM_n(v)$. Similarly, if the the corresponding formula of $v$ is not stronger than $\epsilon \vee \kappa$, it is not included in any loop invariant $\iota$ for $\iota \implies \epsilon \vee \kappa$. The mechanical teacher thus answers $NO$ to the membership query $MEM_n(v)$. In other cases, the mechanical teacher simply gives a random answer. Observe that random answers may yield different loop invariants in different runs. A multitude of loop invariants are exploited by the randomized teacher.

For predicate abstraction, atomic predicates are extracted from program texts heuristically [11]. If many irrelevant atomic predicates are extracted, the performance of clas-

| test case | | vars | cflcts | MEM | $\overline{\text{MEM}}$ | EQ | $\text{MEM}_\$$ | $\overline{\text{MEM}}_\$$ | $\text{EQ}_\$$ | time |
|---|---|---|---|---|---|---|---|---|---|---|
| ide-ide-tape | CDNF | 6.0 | 0.0 | 16.2 | - | 4.8 | 4.0 | - | 0.3 | 0.046s |
| | CDNF+ | 3.0 | 0.0 | 1.0 | - | 3.0 | 0.0 | - | 0.0 | **0.015s** |
| | CDNF++ | 3.0 | 0.0 | 1.0 | 0.0 | 3.0 | 0.0 | 0.0 | 0.0 | **0.015s** |
| ide-wait-ireason | CDNF | 8.0 | 1.6 | 85.5 | - | 32.9 | 14.9 | - | 7.8 | 0.237s |
| | CDNF+ | 4.0 | 0.0 | 8.0 | - | 9.5 | 1.0 | - | 0.0 | **0.044s** |
| | CDNF++ | 4.0 | 0.0 | 19.0 | 0.0 | 29.0 | 0.0 | 0.0 | 0.0 | 0.088s |
| parser | CDNF | 20.0 | 20.5 | 10203.9 | - | 1286.9 | 1306.6 | - | 44.9 | 41.044s |
| | CDNF+ | 9.0 | 0.0 | 97.3 | - | 32.4 | 36.8 | - | 0.0 | **0.501s** |
| | CDNF++ | 9.0 | 0.0 | 304.8 | 0.0 | 91.0 | 8.5 | 0.0 | 0.0 | 1.006s |
| usb-message | CDNF | 10.0 | 0.0 | 21.1 | - | 6.8 | 1.0 | - | 0.0 | 0.097s |
| | CDNF+ | 5.0 | 0.0 | 19.5 | - | 6.6 | 2.2 | - | 0.0 | **0.065s** |
| | CDNF++ | 5.0 | 0.0 | 60.9 | 0.0 | 21.7 | 9.6 | 0.0 | 0.0 | 0.147s |
| vpr | CDNF | 7.0 | 0.9 | 4.6 | - | 6.4 | 20.1 | - | 3.4 | 0.070s |
| | CDNF+ | 5.1 | 0.8 | 4.0 | - | 5.9 | 17.7 | - | 3.0 | **0.057s** |
| | CDNF++ | 5.0 | 0.1 | 5.6 | 3.0 | 9.2 | 21.9 | 0.0 | 2.0 | 0.064s |

**Fig. 1.** Experimental Results – Heuristic Variable Ordering

sical learning will be impeded. We therefore apply incremental learning to improve the efficiency of the learning-based framework.

Two minor modifications derived from the domain knowledge are needed for this application. First, recall that any loop invariant must be stronger than the disjunction of the loop guard and the post-condition. An inferred loop invariant is likely to have atomic predicates from them. We hence start with these atomic predicates and infer loop invariants incrementally. This can be achieved by putting the atomic predicates of the loop guard and the post-condition in front of the variable set, and initializing the variable $\ell$ with the number of such predicates. Second, observe that random answers from the mechanical teacher may induce conflicting valuations. A conflict does not necessarily imply the lack of variables. To give the learning algorithm more chances to infer a loop invariant over the first $\ell$ atomic predicates, the variable $\ell$ is incremented only when the number of conflicts is greater than $\lceil \ell^{1.2} \rceil$. Otherwise, we restart the parameterized CDNF algorithm to infer a loop invariant over the first $\ell$ atomic predicates.

We compare the average performance of 1000 runs in five test cases. Data are collected from an Intel Core2 Quad Processor Q8200 running 64-bit Linux 2.6.32 with 4GB memory. Figure 1 shows our experimental results. Three learning algorithms (CDNF, CDNF+, and CDNF++) are compared in the same test cases from [11]. The number of atomic predicates is reported in the column "vars." For the CDNF algorithm, it indicates the number of atomic predicates extracted from program texts. For the incremental learning algorithms, it indicates the average number of atomic predicates in a loop invariant. The column "cflcts" shows the average number of conflicting valuations induced by random answers or lack of variables. The columns "MEM", "$\overline{\text{MEM}}$", and "EQ" are respectively the average numbers of membership, non-membership, and equivalence queries answered conclusively. The columns "$\text{MEM}_\$$", "$\overline{\text{MEM}}_\$$", and "$\text{EQ}_\$$"

| test case | | vars | cflcts | MEM | $\overline{\text{MEM}}$ | EQ | $\text{MEM}_\$$ | $\overline{\text{MEM}}_\$$ | $\text{EQ}_\$$ | time |
|---|---|---|---|---|---|---|---|---|---|---|
| ide-ide-tape | CDNF | 6.0 | 0.1 | 13.0 | - | 5.0 | 3.6 | - | 0.4 | 0.048s |
| | CDNF+ | 2.7 | 2.5 | 4.1 | - | 10.5 | 0.9 | - | 0.0 | **0.028s** |
| | CDNF++ | 2.8 | 2.7 | 5.2 | 0.0 | 13.2 | 1.6 | 0.0 | 0.1 | 0.037s |
| ide-wait-ireason | CDNF | 8.0 | 1.6 | 87.8 | - | 32.0 | 14.2 | - | 7.6 | 0.247s |
| | CDNF+ | 6.9 | 7.6 | 76.4 | - | 51.7 | 12.6 | - | 5.1 | **0.236s** |
| | CDNF++ | 6.8 | 7.4 | 83.0 | 3.4 | 56.0 | 17.5 | 0.4 | 4.5 | 0.256s |
| parser | CDNF | 20.0 | 5.6 | 2948.4 | - | 405.6 | 563.7 | - | 12.6 | **11.961s** |
| | CDNF+ | 19.0 | 31.1 | 4343.5 | - | 942.0 | 783.0 | - | 8.9 | 18.143s |
| | CDNF++ | 19.1 | 31.5 | 3365.1 | 19.3 | 572.8 | 757.1 | 0.4 | 9.1 | 13.504s |
| usb-message | CDNF | 10.0 | 0.0 | 21.4 | - | 7.3 | 1.0 | - | 0.0 | **0.094s** |
| | CDNF+ | 8.1 | 8.1 | 47.2 | - | 44.1 | 3.1 | - | 0.0 | 0.205s |
| | CDNF++ | 8.4 | 8.4 | 39.8 | 3.5 | 35.1 | 5.0 | 0.0 | 0.0 | 0.181s |
| vpr | CDNF | 7.0 | 1.6 | 9.5 | - | 9.4 | 33.0 | - | 6.3 | 0.112s |
| | CDNF+ | 4.4 | 4.4 | 7.3 | - | 16.4 | 16.2 | - | 6.4 | **0.082s** |
| | CDNF++ | 5.1 | 5.6 | 15.9 | 1.4 | 22.5 | 24.0 | 1.0 | 6.5 | 0.119s |

**Fig. 2.** Experimental Results – Random Variable Orderings

show the average numbers of random membership, non-membership, and equivalence queries respectively. The column "time" indicates average running time.

With our simple heuristic variable ordering, the CDNF+ algorithms performs better than the classical learning algorithm in all test cases. The more sophisticated CDNF++ algorithm is outperformed by the classical algorithm in only one test case (usb-message). Both incremental learning algorithms improve the most complicated case parser significantly. The classical learning algorithm takes about 41 seconds to infer a loop invariant in this test case. The CDNF+ and CDNF++ algorithms use about .5 and 1 second respectively in the same test case. Across the five test cases, the CDNF+ and CDNF++ algorithms have expected speedups of $59.8\%$ and $36.9\%$ respectively.

We now evaluate the worst-case performance of the incremental learning algorithms. To this end, we randomly order the set of atomic predicates extracted from program texts at each run. Starting from the first variable in a random variable ordering, our incremental learning algorithms are invoked to infer loop invariants. Similarly, we invoke the classical CDNF algorithm on all randomly ordered variables at each run. We compare the average of 1000 runs in each test case. Figure 2 gives the results.

With random variable orderings, the incremental learning algorithms perform comparably to the classical learning algorithm in all test cases but usb-message. For this particular case, conflicts are negligible when all atomic predicates are used. Incremental learning, on the other hand, needs to enlarge the set of atomic predicates 8 times. Subsequently, both incremental learning algorithms make lots of useless queries before a loop invariant is inferred. Also note that the CDNF algorithm performs significantly better with random variable orderings in the test case parser. Yet the classical algorithm still requires about 12 seconds to infer a loop invariant. In comparison, our incremental algorithms are an order of magnitude faster with our heuristic variable ordering (cf Figure 1). Using random variable orderings, we observe $19.4\%$ and $18.5\%$ of

slowdowns respectively from the CDNF+ and CDNF++ algorithms across the five test cases. Note that the test case `usb-message` alone registers a slowdown of more than $90\%$. The incremental learning algorithms in fact perform slightly better than the classical algorithm for the other four test cases on average ($5.3\%$ for CDNF+ and $0.1\%$ for CDNF++). Also recall that this is the worst-case scenario for incremental learning. As in loop invariant inference, heuristics for choosing sensible variable orderings are often available for most applications. Our incremental learning algorithms should outperform the classical algorithm with the domain knowledge in practice.

## 6 Conclusion

Classical learning algorithms for Boolean functions assume a fixed number of variables for unknown targets. The assumption precludes applications where indefinitely many variables are needed. It can also be unexpectedly inefficient at the presence of irrelevant variables. We address the problem by inferring unknown targets through enlarging numbers of ordered variables. Our experiments show that incremental learning attains significant improvement with a simple heuristic variable ordering. They also suggest manageable slowdowns in the worst-case scenario with random variable orderings.

Applications of incremental learning in formal verification are under investigation. Particularly, problems in program verification inherently have indefinitely many variables in unknown targets. Applying incremental learning to program verification will be interesting. We are working on applications in automated assume-guarantee reasoning. Domain knowledge about contextual assumptions will be essential in this application.
***Acknowledgement***. We thank the invaluable comments from anonymous referees.

## References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
2. Bobaru, M.G., Păsăreanu, C.S., Giannakopoulou, D.: Automated assume-guarantee reasoning by abstraction refinement. In Gupta, A., Malik, S., eds.: CAV. Volume 5123 of LNCS., Springer (2008) 135–148
3. Bshouty, N.H.: Exact learning Boolean function via the monotone theory. Information and Computation **123**(1) (1995) 146–153
4. Chen, Y.F., Clarke, E.M., Farzan, A., He, F., Tsai, M.H., Tsay, Y.K., Wang, B.Y., Zhu, L.: Comparing learning algorithms in automated assume-guarantee reasoning. In: ISoLA (1). Volume 6415 of LNCS., Springer (2010) 643–657
5. Chen, Y.F., Clarke, E.M., Farzan, A., Tsai, M.H., Tsay, Y.K., Wang, B.Y.: Automated assume-guarantee reasoning through implicit learning. In Touili, T., Cook, B., Jackson, P., eds.: CAV. Volume 6174 of LNCS., Springer (2010) 511–526
6. Dutertre, B., Moura, L.D.: The Yices SMT solver. Technical report, SRI International (2006)
7. Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: POPL, ACM (2002) 191–202
8. Gheorghiu, M., Giannakopoulou, D., Păsăreanu, C.S.: Refining interface alphabets for compositional verification. In Grumberg, O., Huth, M., eds.: TACAS. Volume 4424 of LNCS., Springer (2007) 292–307

9. Giannakopoulou, D., Păsăreanu, C.S.: Special issue on learning techniques for compositional reasoning. Formal Methods in System Design **32**(3) (2008) 173–174
10. Howar, F., Steffen, B., Merten, M.: Automata learning with automated alphabet abstraction refinement. In Jhala, R., Schmidt, D., eds.: VMCAI. Volume 6538 of LNCS. Springer (2011) 263–277
11. Jung, Y., Kong, S., Wang, B.Y., Yi, K.: Deriving invariants in propositional logic by algorithmic learning, decision procedure, and predicate abstraction. In Barthe, G., Hermenegildo, M.V., eds.: VMCAI. LNCS, Springer (2010) 180–196
12. Jung, Y., Lee, W., Wang, B.Y., Yi, K.: Predicate generation for learning-based quantifier-free loop invariant inference. In Abdulla, P.A., Leino, K.R.M., eds.: TACAS. Volume 6605 of LNCS., Springer (2011) 205–219
13. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press (1994)
14. Kong, S., Jung, Y., David, C., Wang, B.Y., Yi, K.: Automatically inferring quantified loop invariants by algorithmic learning from simple templates. In Ueda, K., ed.: APLAS. Volume 6461 of LNCS., Springer (2010) 328–343
15. Kroening, D., Strichman, O.: Decision Procedures - an algorithmic point of view. EATCS. Springer (2008)
16. Lee, W., Wang, B.Y., Yi, K.: Termination analysis with algorithmic learning. In Parthasarathy, M., Seshia, S.A., eds.: CAV. LNCS (this volume), Springer (2012)
17. Saídi, H., Graf, S.: Construction of abstract state graphs with PVS. In Grumberg, ed.: CAV. Volume 1254 of LNCS., Springer (1997) 72–83