CEGAR-based EF Synthesis of Boolean Functions with an Application to Circuit Rectification

Heinz Riener*
*Institute of Space Systems,
German Aerospace Center, Germany
heinz.riener@dlr.de

Rüdiger Ehlers†‡
‡DFKI GmbH,
Bremen, Germany
rehlers@uni-bremen.de

Goerschwin Fey*†
†Faculty of Mathematics and Computer Science,
University of Bremen, Germany
goerschwin.fey@dlr.de

***Abstract*—The *Exists-Forall* (EF) synthesis problem deals with finding parameters such that for all input assignments a correctness specification is met. Many standard problems from computer-aided design and verification can be formulated as an instance of EF synthesis when a function template with holes — parameters to be synthesized — is provided. In this paper, we generalize the idea of EF synthesis in the context of Boolean logic by allowing existential quantification over the domain of Boolean functions (rather than Boolean variables) and present a bounded synthesis approach guided by counterexamples to generate them using techniques from Boolean learning. As an application, we present circuit rectification as an EF synthesis problem and apply the presented approach to incrementally synthesize patches for digital circuits with multiple seeded faults.**

I. Introduction

Motivation. *Satisfiability* (SAT) solving has many applications in computer science and artificial intelligence. The classical SAT problem asks, given a set of constraints encoded into a logic formula over Boolean variables, whether a satisfying assignment to the variables exists such that all constraints are satisfied. SAT is the canonical NP-complete problem and was, due to its practical importance, intensively studied in the last fifty years. As a result, effective reasoning tools, called *SAT oracles* or *SAT solvers*, are nowadays available. In practice, a SAT oracle decides whether a Boolean formula is satisfiable and, if so, provides a satisfying assignment as witness. Beyond SAT, recently the problem of solving *Quantified Boolean Formulæ* (QBF) with exactly one quantifier alternation — the so-called *Exists-Forall* (EF) or *2QBF problem* — is of major interest. Instances of 2QBF arise, e.g., in the context of parameter synthesis (or EF synthesis), where the *existence* of parameters is checked such that *for all* possible input assignments a correctness specification has to be met. Algorithms for deciding 2QBF problems using two incremental SAT oracles were proposed which outperform traditional approaches for evaluating general quantified Boolean formulæ.

Problem. In the context of Boolean logic, the EF synthesis problem asks, given a Boolean predicate $Q(x, y)$ over Boolean variables x and y, whether an assignment $\hat{x}$ exists such that for all possible assignments $\hat{y}$, the predicate $Q(\hat{x}, \hat{y})$ holds. In many practical applications, e.g., invariant generation or repair, functions over parameters have to be found. A straight-forward approach to synthesize functions is to enumerate abstract function templates with holes — parameters to be synthesized — and apply EF synthesis to "concretize" them. Blindly enumerating function templates by size and testing them against the specification, however, requires to solve many EF synthesis problems and is typically too costly to find a solution for practical problems in reasonable time. A smart enumeration strategy that avoids the enumeration of functionally equivalent expressions and learns from unsuccessful synthesis attempts is desirable.

Solution. In this paper, we generalize the EF synthesis problem by allowing existential quantification over the domain of Boolean functions (rather than Boolean variables) and present a bounded synthesis approach guided by counterexamples to generate them using techniques from Boolean learning. In an iterative guess-and-check scheme, the approach determines a concrete Boolean function in *Disjunctive Normal Form* (DNF) bounded by the number of terms or concludes that no such function exists. The approach is functional and extends ideas from parameter synthesis, where some parameters are iteratively determined in a *CounterExample-Guided Abstraction Refinement* (CEGAR) loop to guarantee that a given correctness specification is met. Instead of parameters, in our approach, a Boolean function is synthesized assuming that the function has a simple DNF representation with only a few terms. As an application, we use the CEGAR-based bounded synthesis approach to functionally rectify digital circuits with multiple seeded faults when a single location at which the circuit can be rectified is known and discuss how this approach can be generalized by synthesizing Boolean functions with multiple outputs.

Structure. The remainder of the paper is structured as follows: in Section II, we discuss related work. Section III is dedicated to the synthesis problem. We describe the EF synthesis problem for Boolean functions and present an algorithm for solving it using techniques from Boolean learning. In Section IV, we describe how circuit rectification problems can be solved by the proposed EF synthesis approach and present in Section V experimental results for rectifying digital circuits on the gate level with seeded faults. Section VI concludes.

II. Related Work

Exists-forall synthesis and QBF. *CounterExample-Guided Inductive Synthesis* (CEGIS) [1] was introduced to effectively reduce $\exists\forall$-queries to SAT in the context of program sketching.

This is the author-archived version of the paper. The original publication is available on IEEE XPlore under https://doi.org/10.1109/ASPDAC.2017.7858328.

The idea was generalized, e.g., in GhostQS [2] or RaReQS [3], to (recursively) solve quantified Boolean formulæ with an arbitrary number of quantifier alternations. EFSMT [4] solves $\exists\forall$-queries modulo theories with applications in control theory and parameter synthesis for cyber-physical-systems [5]. More recently, incremental approaches for solving $\exists\forall\exists$-queries with two quantifier alternations have been developed, e.g., for exact diagnosis of digital circuits [6] or for synthesizing Byzantine-resilient distributed systems [7].

Syntax-guided synthesis. *Syntax-Guided Synthesis* (SyGuS) [8] arose in the context of program synthesis and program optimization. The SyGuS problem asks for finding an expression (modulo theories) that meets a semantic correctness specification and an additional syntactic requirement that constrains the space of allowed implementations. SyGuS, e.g., the refutation-based approach [9], allows to synthesize functions modulo background theories. However, SyGuS is not tweaked for the Boolean case and often either timeouts or produces huge and suboptimal solutions.

Boolean learning. Boolean learning is a technique to synthesize a Boolean function with a hidden, unknown structure from a set of input-output samples. The problem can be reduced to SAT and was, e.g., in [10], solved using mathematical programming. The description of Boolean learning in this paper mainly follows the notation of Knuth [11]. However, in contrast to Knuth, an incremental version of the problem is considered, where input-output samples are systematically added to refine the Boolean function.

Engineering change order and circuit rectification. Logic rectification approaches can be broadly classified into three categories: 1.) *Dictionary-based approaches* use an error model for rectification [12]. They are typically fast, but limited in their applicability to certain fault types. 2.) *Structural approaches* [13], [14], [15], [16], match the differences of the circuit and its specification to reduce the possible number of rectifications and work only well if the circuit and the specification share common substructures. 3.) *Functional approaches*, e.g., post-rectification [17], Boolean unification [18], BDD-based synthesis [19], [20], counterexample-guided re-synthesis [21], [22], apply synthesis techniques to rectify the circuits. The functional approaches are not restricted to any fault model or assumptions about the structure of the circuit, but typically do not scale well with the circuit size. We describe functional circuit rectification as an EF synthesis problem. The overall idea is similar to counterexample-guided re-synthesis, but the synthesis approach is orthogonal to the problem-specific search techniques, e.g., presented in [21].

III. Synthesis of Boolean functions

Let $\mathbb{B} := \{0, 1\}$. Suppose that $f : \mathbb{B}^n \to \mathbb{B}$ is a Boolean function over Boolean variables $x := x_1, \ldots, x_n$ with multiple inputs and a single output and Q is a Boolean predicate (a specification) that imposes constraints on f that have to be met. The problem of synthesizing f is to find a concrete expression $\hat{f}$ in Boolean logic such that for all assignments

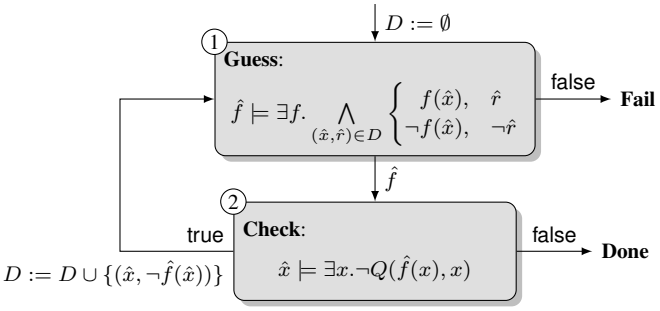


Fig. 1. CEGAR-based synthesis of Boolean functions.

to x the predicate $Q(\hat{f}(x), x)$ holds. This problem can be formalized as a query in second-order logic[1] of the form

$$\hat{f} \models \exists f. \forall x. Q(f(x), x), \tag{1}$$

where a model $\hat{f}$ for the Boolean function f has to be constructed. Queries of such a form are beyond the capabilities of classical SAT and QBF solvers. SAT solvers cannot deal with quantifier alternations, whereas neither SAT nor QBF solvers allow to quantify over Boolean functions.

Exists-forall synthesis for Boolean functions. Suppose that the specification Q is provided, a CEGAR-based exist-forall synthesis approach approximates the domain of the $\forall$-quantified variables in Eq. 1 with a subset of selected sample points. The sample points are iteratively generated using the guess-and-check scheme shown in Fig. 1. In each iteration, a Boolean function $\hat{f}$ is guessed that is consistent with all sample points in an initially empty database D. The function is then checked against Q. If the check fails, a counterexample is generated and added to D. The process terminates if either no new Boolean function $\hat{f}$ consistent with the sample points in D can be guessed, which proves that no such function exists (**Fail**), or if no counterexample exists that refutes $\hat{f}$, which proves that the guessed function is a correct solution (**Done**).

Learning Boolean functions. We use techniques from Boolean learning [10], [11] to guess a concrete candidate function $\hat{f}(x)$ over Boolean variables $x := x_1, \ldots, x_n$ consistent with all sample points in the database D. The Boolean function is represented as a DNF with a bounded number of terms. Suppose that the number m of terms is fixed, we construct a satisfiability problem over $2nm$ Boolean variables $p_{j,l}$ and $q_{j,l}$ for $1 \le j \le m$ and $1 \le l \le n$, to determine a concrete DNF representation of the Boolean function, where

$$p_{j,l} := \begin{cases} 1, & x_l \text{ in term } j \\ 0, & \text{otherwise} \end{cases} \text{ and } q_{j,l} := \begin{cases} 1, & \neg x_l \text{ in term } j \\ 0, & \text{otherwise.} \end{cases}$$

The general form of all Boolean functions in m-DNF, i.e., bounded by the number m of terms, can be described as

$$F(x_1, \ldots, x_n; p_{1,1} \ldots, p_{m,n}; q_{1,1} \ldots, q_{m,n}) \equiv \bigvee_{j=1}^{m} \bigwedge_{l=1}^{n} \Big(\mathsf{ITE}(p_{j,l}, x_l, \mathsf{true}) \wedge \mathsf{ITE}(q_{j,l}, \neg x_l, \mathsf{true}) \Big), \tag{2}$$

[1] f is a second-order variable

where the expressions of form $\mathsf{ITE}(c, g, h)$ evaluate to g if $c = 1$ and to h if $c = 0$, respectively. If concrete values $\hat{p}_{j,l}$ and $\hat{q}_{j,l}$ are assigned to all $p_{j,l}$ and $q_{j,l}$, then $F(x_1, \ldots, x_n; \hat{p}_{1,1} \ldots, \hat{p}_{m,n}; \hat{q}_{1,1} \ldots, \hat{q}_{m,n})$ is a Boolean function in m-DNF. Hence the conditions in the ITE-expressions are evaluated such that the expressions degenerate to Boolean variables. Note, also, that if $\hat{p}_{j,l}$ and $\hat{q}_{j,l}$ are both true for some j, the corresponding j-th term cancels out.

Non-incremental concretization. Given a database D of pairs $(\hat{x}, \hat{r})$ of concrete sample points $\hat{x} := \hat{x}_1, \ldots, \hat{x}_n$ and the correct evaluation $\hat{r} := f(\hat{x})$ of the function to be synthesized, a satisfying assignment $\hat{p}$ and $\hat{q}$ to the Boolean variables $p := p_{1,1}, \ldots, p_{m,n}$ and $q := q_{1,1}, \ldots, q_{m,n}$ can be computed as a solution of the query

$$\hat{p}, \hat{q} \models \bigwedge_{(\hat{x},\hat{r}) \in D} \begin{cases} \mathcal{N}(\hat{x}, p, q), & \hat{r} = 0 \\ \mathcal{P}(\hat{x}, p, q), & \hat{r} = 1 \end{cases}, \tag{3}$$

with

$$\mathcal{N}(x_1, \ldots, x_n; p_{1,1}, \ldots, p_{m,n}; q_{1,1}, \ldots, q_{m,n}) := \Big(\bigwedge_{j=1}^{m} \bigvee_{l=1}^{n} (\mathsf{ITE}(x_l, q_{j,l}, p_{j,l}))\Big)$$

and

$$\mathcal{P}(x_1, \ldots, x_n; p_{1,1}, \ldots, p_{m,n}; q_{1,1}, \ldots, q_{m,n}) := \exists z_1, \ldots, z_m . \Big(\bigvee_{j=1}^{m} z_j\Big) \wedge \Big(\bigwedge_{j=1}^{m} \bigwedge_{l=1}^{n} (\neg z_j \vee \neg \mathsf{ITE}(x_l, q_{j,l}, p_{j,l}))\Big).$$

using a SAT oracle.

When used in Eq. 2, the assignments $\hat{p}$, $\hat{q}$ concretize the general m-DNF, such that $\hat{f}$ is a m-DNF consistent with all sample points in D. If Eq. 3 becomes unsatisfiable, then no such m-DNF exists, which either means that m is too restricted or no Boolean function at all exists.

Incremental concretization. Eq. 3 can be used to learn $\hat{p}$ and $\hat{q}$ from a given database D. In many scenarios, however (e.g., CEGAR-based approaches), not a complete database is initially provided, but the database iteratively improves through refinements over time. In these cases, incremental SAT oracles typically outperform non-incremental approaches because they preserve and re-use learned conflict clauses from previous SAT calls. If m is fixed, the previously discussed approach to Boolean learning can be re-formulated in an incremental fashion. Suppose that $(\hat{x}, \hat{r})$ is a newly provided pair and $\mathcal{I}(p, q)$ is the current SAT instance with all constraints from the previously learned sample points (and initially true), then

$$\hat{p}, \hat{q} \models \mathcal{I}(p, q) \wedge \begin{cases} \mathcal{N}(\hat{x}, p, q), & \hat{r} = 0 \\ \mathcal{P}(\hat{x}, p, q), & \hat{r} = 1 \end{cases}, \tag{4}$$

is an incremental formulation of Eq. 3. If m is increased, however, the SAT instance has to be re-constructed from D and the learned clauses are lost.

CEGAR-based bounded functional synthesis. The CEGAR-based exists-forall synthesis approach for Boolean functions combined with the Boolean learning scheme enables a bounded synthesis approach for Boolean functions as shown in Fig. 2. Given the Boolean predicate Q, a lower bound l and an upper bound u for the number of terms, the synthesis approach constructs a Boolean function in m-DNF representation, $l \leq m \leq u$, guided by counterexamples (or reports that no such function exists). The synthesis approach iteratively refines a candidate function. In each iteration, 1.) a Boolean function in m-DNF is guessed (starting with $m = l$) and 2.) checked against Q. If the check succeeds, **Done** is reported and the guessed m-DNF is returned to the user. If the check fails, the generated counterexamples shed light on why the DNF does not meet the specification and 3.) is used to incrementally refine the guess. If no new Boolean function can be guessed and the upper bound u on the number of terms is reached, i.e., if $m > u$ evaluates to true, the process terminates with **Fail**, or otherwise the number m of terms is increased. This step is non-incremental, i.e., when the number of terms is increased, in 4.) the satisfiability problem is re-constructed from the whole database of sample points. Consequently, all learned clauses of the SAT oracle are lost in this step. The knowledge collected in the database D, however, is re-used for the next guessed Boolean function.

Generalization to multi-output functions. Suppose that $f : \mathbb{B}^n \to \mathbb{B}^s$ is a Boolean function to be synthesized over Boolean variables $x := x_1, \ldots, x_n$ with s output bits. The presented approach is still applicable with the major difference that concrete sample points obtained as counterexamples cannot be easily classified as positives or negatives. Instead, each counterexample can be seen as a lemma, which states that a certain combination of the output bits is not admissible for a concrete input assignment. One of the output bits has to differ for the same input assignment in the next iteration. Consequently, the approach explores and mines the relationship between the individual output bits driven by counterexamples until a Boolean function that meets the correctness specification is found or the SAT solver is able to prove that no such function exists within the given restrictions.

IV. Circuit Rectification as EF Synthesis

Incremental synthesis of circuit patches. Suppose that C is a faulty combinational circuit with correctness specification S, e.g., provided as a reference circuit or a set of logic constraints. Moreover, suppose that also a fixed single location l in C is known at which the circuit can be rectified. The location may be manually chosen by a designer or automatically computed using techniques from automated fault localization. We introduce a Boolean predicate $\mathsf{Rectify}(f, x)$ that evaluates to true if C produces the same outputs as S on input x when the circuitry described by the single-output Boolean function $f : \mathbb{B}^k \to \mathbb{B}$ is plugged into the circuit as a replacement for the gate at location l.

In this setting, the inputs of f could be any subset of the primary inputs and the circuit gates not in the output cone of

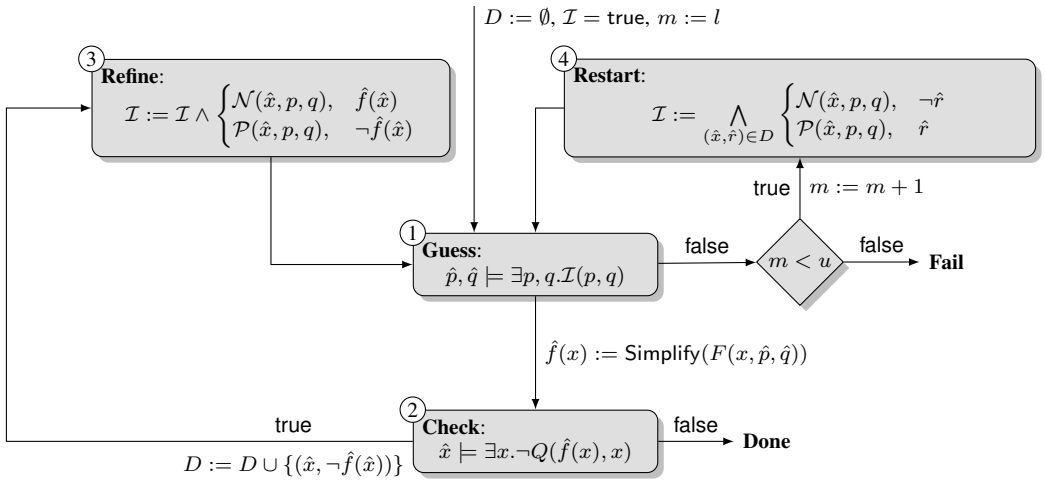
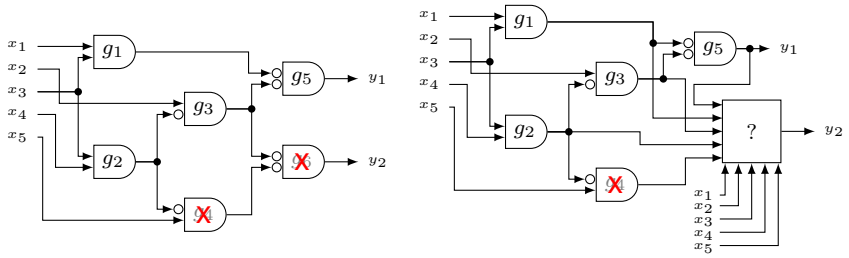


Fig. 2. CEGAR-based bounded synthesis of a Boolean function in DNF.



i	$x_1x_2x_3x_4x_5g_1g_2g_3g_4g_5g_6$	?
1.	0 0 0 0 0 0 0 0 0 1 0 **1**	true
2.	1 1 0 0 1 0 0 1 0 0 **0**	$(\overline{x}_1)$
3.	1 1 1 1 1 1 1 1 0 0 1 **1**	$(\overline{g}_3)$
4.	0 0 0 1 1 0 0 0 1 0 **0**	$(\overline{x}_4 \cdot \overline{g}_3) \vee (g_2)$
5.	0 0 0 1 0 0 0 0 1 0 **1**	$(\overline{x}_5 \cdot \overline{g}_3) \vee (g_2)$

Fig. 3. Incremental synthesis of a circuit patch by CEGAR-based EF synthesis.

gate l. Selecting the primary inputs only suffices to guarantee completeness of the approach with respect to the term bound. The more signals are additionally selected, the better are the odds to find a *small* rectification by re-using existing logic. The functional synthesis problem can then be formalized as the second-order logic query

$$\exists f.\forall x.\exists y.\text{Rectify}(f(x, y), x), \tag{5}$$

where the additional variables y are introduced to describe internal gates of C. The Boolean predicate encodes the circuit C and ensures that those y variables can only take value compatible with the circuit structure and the input assignment to x. Once the x values are chosen, the values for y are totally determined such that the innermost quantifier can be handled using the bounded EF synthesis approach for Boolean functions as previously described.

Example 1. As an example, consider the combinational circuit shown in Fig. 3, e.g., obtained after optimization. The circuit has two faults marked for the reader with x symbols (in red) at the respective gates. The fault locations, however, are actually not known to the designer. A specification in terms of an unoptimized, reference circuit is available and can be used for functional verification and debugging, but does not shed light on the exact problem for the behavioral mismatch of the two circuits, e.g., because they are structurally too different. An automated fault localization tool reveals that the circuit can be fixed at the output of gate g_6. The designer is now left with the problem of determining a fix that rectifies the input-output behavior of the circuit at g_6. The problem of finding a possible replacement can be formalized as an instance of Eq. 5, as shown in Fig. 3, where a new combinational block **?** has to be synthesized using primary inputs or internal signals. The table on the right shows the progress of the bounded synthesis algorithm for synthesizing a patch at location g_6, where the counterexample that disproved functional equivalence of the optimized and unoptimized circuits is provided to the algorithm as an initial sample point. Each line shows one mined input-output sample from the truth table of the Boolean function to be synthesized and the corresponding learned circuit patch in DNF (with at most 3 terms) that is plugged into **?** in the next iteration. After 5 iterations, a correct patch is found and the algorithm terminates.

Don't care optimization. The presented Boolean learning scheme in Section III uses concrete input-output samples to mine a Boolean function but does not make use of don't cares. Consider the rectification circuit shown in Fig. 4. The objective is to synthesize the combinational block **?** using the primary inputs x_1, x_2 and x_3 and the internal output signals of the gates g_1 and g_2. Suppose that combinational equivalence checking produces a sample -10-0 with correct output $g_4 = 0$, where x_1 and g_2 are don't cares (denoted by -). The three-valued assignment covers (among others) the concrete assignment 11000 which is uncontrollable in the circuit, i.e., $g_1 = 1$ if $x_1 = 1$ and $x_2 = 1$. Thus, the don't cares impose an additional but unncessary constraint on the rectification that

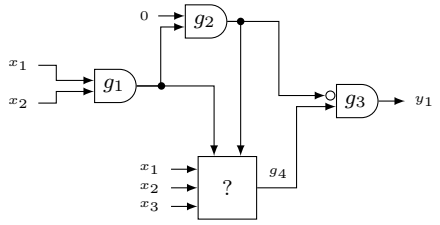


Fig. 4. Overconstrainted learning by considering don't care.

fixes g_4 to 0 for 11000. In general, these additional constraints may lead to the exclusion of possible fixes as well as cause a significant performance overhead, e.g., when a simple solution is missed and a much more complicated solution is found later in the rectification process. However, since the additional constraints stem from uncontrollable gate outputs, soundness of the overall approach is not affected. In the worst-case, the rectification can be implemented using only primary inputs, which are assumed to be always controllable.

V. Experimental Evaluation

The counterexample-guided EF synthesis approach was implemented and evaluated in the context of circuit rectification on the gate level. We used the approach to synthesize patches for digital circuits with multiple seeded faults, where exactly one location for rectifying the circuits is known.

Benchmarks. No standard benchmark set for circuit rectification (with known faults) is available. Thus, we use the ISCAS'85 benchmarks as well as the EPFL combinational benchmark suite[2] with seeded random faults. The rectification approach does not rely on any assumptions about the type of the seeded faults. However, for the experiments, we seeded stuck-at faults as well as single negations on multiple locations. To keep our implementation simple and flexible, all circuits were first translated to *And-Inverter Graphs* (AIGs) utilizing ABC [23]. From each circuit considered, then 10 faulty versions are generated. Each contains at least three faults at random locations; however, a single location for rectifying the circuit is known. These locations were computed using the automated fault localization approach presented in [6]. Benchmarks that cannot be rectified at a single location were not considered. For instance, in case of the multiplier `c6288`, multiple seeded faults always propagate to multiple outputs on independent paths such that a single rectification is not possible. Since multiple faults are seeded at random locations which are potentially far distanced from each other, in several cases large logic cones have to be synthesized in order to correctly rectify the circuits.

Experiments. All experiments are conducted on a quad-core Intel® Core™ i5-2520M CPU with 2.50GHz and 8GB RAM running Arch Linux kernel 4.5.4-1. The check-step is implemented by leveraging the combinational equivalence checker of ABC via its API interface. Each counterexample is re-simulated on the circuit graph to obtain the output values for all internal gates. For incrementally learning the candidate functions, we used the SAT solver MiniSAT 2.2.0[3] via its API interface. Table I and Table II present experimental results for circuit rectification with the term bounds $m \leq 2$ and $m \leq 3$, respectively. In total, we attempt to rectify 170 circuits. For each rectification problem, time was limited to T/O := $100s$. Both tables are built as follows: the first column names the benchmark, followed by three columns that list the number of successfully rectified circuits (**R**), the number of identified unrectifiable circuits (**U**) within the term bound, and the number of timeouts (**T**). The next two columns list the mean number of variables (**n**) as an indication of the complexity of the Boolean learning problem and the mean number of iterations (**i**) needed for rectification. The last three columns list the mean runtimes in seconds for successful rectification ($\mathbf{t_R}$), identified unrectifiable circuits ($\mathbf{t_U}$), and mean total runtime ($\mathbf{t_\Sigma}$), respectively. The last line summarizes the results for all benchmarks.

TABLE I
Bounded patch synthesis up to 2-DNF

Name	T/O := 100s / [l,u] := [1,2]							
	R	**U**	**T**	**n**	**i**	$\mathbf{t_R}$ [s]	$\mathbf{t_U}$ [s]	$\mathbf{t_\Sigma}$ [s]
c17	9	1	0	7.00	7.30	0.01	0.01	0.01
c432	6	4	0	175.90	50.50	0.14	0.15	0.14
c499	0	10	0	285.30	53.50	0.00	0.99	0.99
c880	0	10	0	386.00	70.20	0.00	1.06	1.06
c1355	1	9	0	350.60	61.40	1.27	2.73	2.58
c1908	0	10	0	411.30	75.80	0.00	0.79	0.79
c2670	1	9	0	948.80	157.10	4.06	18.52	17.07
c3540	0	10	0	1030.10	152.50	0.00	21.62	21.62
c5315	0	0	10	1922.10	120.60	0.00	0.00	99.91
c7552	1	0	9	2275.40	169.60	0.79	0.00	90.05
cavlc	0	10	0	702.00	103.60	0.00	2.07	2.07
ctrl	10	0	0	174.00	13.50	0.03	0.00	0.03
int2float	0	9	1	270.00	60.00	0.00	0.27	0.25
max	0	5	5	2419.60	184.60	0.00	65.93	82.96
priority	0	10	0	1105.00	120.80	0.00	5.53	5.53
router	1	9	0	293.90	45.70	0.50	0.58	0.57
sin	0	0	10	5245.30	236.30	0.00	0.00	100.00
Total	29	106	35	1059.04	99.00	0.27	8.02	25.04

Discussion. The CEGAR-based EF synthesis algorithm was able to rectify 29 benchmarks for $m \leq 2$ terms and 40 for $m \leq 3$ terms and, moreover, proved 106 benchmarks for $m \leq 2$ and 17 for $m \leq 3$ as "unrectifiable", respectively. The minimal number of terms sufficient for rectification for the circuit benchmarks are not known. We are interested in small rectification and, thus, restrict the number of terms to at most 2 or 3 terms. Notice that the rectification is fast when possible — less than $1s$ in all cases. Proving that a rectification is not possible, however, is typically costly because all possibilities over all variables have to be taken into account. Increasing the time bound, however, does not help. Only for two benchmarks, the results are affected when the time bound is increased to $200s$. On the other hand, the successful rectifications are still generated if the time bound is decreased, e.g., to 5 seconds. This is also true if a lower bound of $m \geq 5$ (or more) is

[2]The EPFL Combinational Benchmark Suite, http://lsi.epfl.ch/benchmarks

[3]MiniSAT [24], http://minisat.se/MiniSat.html

TABLE II
BOUNDED PATCH SYNTHESIS UP TO 3-DNF

Name	T/O := 100s / [l,u] := [1,3]							
	R	U	T	n	i	t_R [s]	t_U [s]	t_Σ [s]
c17	9	1	0	7.00	7.80	0.02	0.01	0.01
c432	7	3	0	175.90	72.80	0.31	38.90	11.89
c499	0	0	10	285.30	97.70	0.00	0.00	99.99
c880	0	0	10	386.00	124.90	0.00	0.00	99.89
c1355	1	0	9	350.60	97.30	1.28	0.00	90.01
c1908	0	0	10	411.30	144.40	0.00	0.00	99.84
c2670	1	0	9	948.80	239.40	4.53	0.00	90.28
c3540	0	0	10	1030.10	278.50	0.00	0.00	99.76
c5315	0	0	10	1922.10	120.30	0.00	0.00	99.88
c7552	1	0	9	2275.40	168.70	0.89	0.00	89.87
cavlc	3	1	6	702.00	232.20	3.14	96.89	70.59
ctrl	10	0	0	174.00	13.50	0.03	0.00	0.03
int2float	5	4	1	270.00	141.60	1.12	20.11	8.61
max	0	0	10	2419.60	245.40	0.00	0.00	99.72
priority	1	1	8	1105.00	237.70	5.58	19.40	82.48
router	2	7	1	293.90	84.90	0.62	3.33	12.45
sin	0	0	10	5245.30	236.20	0.00	0.00	99.97
Total	40	17	113	1058.96	149.61	0.78	19.81	67.96

considered for the number of terms.

Boolean learning does not use any information from the circuit structure except for the input-output samples constructed during equivalence checking. Hence, the internal complexity of a benchmark circuit does not impact the performance, if equivalence checking is fast enough to produce input-output samples, but is mainly determined by the number of variables to consider. For all benchmarks, we consider all primary inputs and the outputs of all internal gates as variables. Existing techniques are mostly orthogonal to our work and can be combined with the proposed rectification approach. For instance, entropy-guided search [21] provides a necessary (but not sufficient) criterion for selecting a subset of variables that are sufficient for synthesis, which allows to reduce the size of the search space without excluding possible solutions.

VI. CONCLUSION

We presented a generalized EF synthesis approach in the context of Boolean logic that allows to existentially quantify over the domain of Boolean functions (rather than Boolean variables) and described a counterexample-guided algorithm for generating them using Boolean learning. The algorithm incrementally refines a candidate function in DNF bounded by the number of terms until either a provided correctness specification is met or the algorithm proves that no function within the given term bound exists. As an application, we described circuit rectification as an EF synthesis problem and applied the present approach to incrementally synthesize patches for combinational circuits with multiple faults.

ACKNOWLEDGMENT

This work was supported by the European Union (grant no. 644905) and the Institutional Strategy of the University of Bremen, funded by the German Excellence Initiative.

REFERENCES

[1] A. Solar-Lezama, "Program sketching," *Software Tools for Technology Transfer*, vol. 15, no. 5-6, pp. 475–495, 2013.

[2] M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke, "Solving QBF with counterexample guided refinement," in *Theory and Applications of Satisfiability Testing*, 2012, pp. 114–128.

[3] M. Janota, W. Klieber, J. Marques-Silva, and E. M. Clarke, "Solving QBF with counterexample guided refinement," *Artificial Intelligence*, pp. 1–25, 2016.

[4] C. Cheng, N. Shankar, H. Ruess, and S. Bensalem, "EFSMT: A logical framework for cyber-physical systems," *CoRR*, vol. abs/1306.3456, 2013. [Online]. Available: http://arxiv.org/abs/1306.3456

[5] H. Riener, R. Könighofer, G. Fey, and R. Bloem, "SMT-based CPS parameter synthesis (tool presentation)," in *ARCH Workshop*, 2016.

[6] H. Riener and G. Fey, "Exact diagnosis using Boolean satisfiability," in *International Conference on Computer Aided Design*, 2016, to Appear.

[7] R. Bloem, N. Braud-Santoni, and S. Jacobs, "Synthesis of self-stabilizing and byzantine-resilient distributed systems," in *Computer Aided Verification*, 2016, pp. 157–176.

[8] R. Alur, R. Bodík, E. Dallal, D. Fisman, P. Garg, G. Juniwal, H. Kress-Gazit, P. Madhusudan, M. M. K. Martin, M. Raghothaman, S. Saha, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa, *Dependable Software Systems Engineering*. IOS Press, 2015, ch. Syntax-Guided Synthesis, pp. 1–25.

[9] A. Reynolds, M. Deters, V. Kuncak, C. Tinelli, and C. W. Barrett, "Counterexample-guided quantifier instantiation for synthesis in SMT," in *Computer Aided Verification*, 2015, pp. 198–216.

[10] A. P. Kamath, N. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende, "A continuous approach to inductive inference," *Mathematical Programming*, vol. 57, pp. 215–238, 1992.

[11] D. E. Knuth, *The Art of Computer Programming, Volume 4, Pre-Fascicle 6a: A (partial draft) of Section 7.2.2.2: Satisfiability*. Addison-Wesley, 2015.

[12] A. G. Veneris and I. N. Hajj, "A fast algorithm for locating and correcting simple design errors in VLSI digital circuits," 1997, pp. 45–50.

[13] D. Brand, A. D. Drumm, S. Kundu, and P. Narain, "Incremental synthesis," in *International Conference on Computer Aided Design*, 1994, pp. 14–18.

[14] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri, "DeltaSyn: An efficient logic difference optimizer for ECO synthesis," in *International Conference on Computer Aided Design*, 2009, pp. 789–796.

[15] S.-Y. Huang, K.-C. Chen, and K.-T. Cheng, "AutoFix: A hybrid tool for automatic logic rectification," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 9, pp. 1376–1384, 1999.

[16] S.-L. Huang, W.-H. Lin, P.-K. Huang, and C.-Y. R. Huang, "Match and replace: A functional ECO engine for multierror circuit rectification," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 3, pp. 467–478, 2013.

[17] Y. Watanabe and R. K. Brayton, "Incremental synthesis for engineering changes," in *International Conference on Computer Design*, 1991, pp. 40–43.

[18] M. Fujita, Y. Tamiya, Y. Kukimoto, and K.-C. Chen, "Application of Boolean unification to combinational logic synthesis," in *International Conference on Computer Aided Design*, 1991, pp. 510–513.

[19] C.-C. Lin, K.-C. Chen, S.-C. Chang, M. Marek-Sadowska, and K.-T. Cheng, "Logic synthesis for engineering change," in *Design Automation Conference*, 1995, pp. 647–652.

[20] C.-C. Lin, K.-C. Chen, and M. Marek-Sadowska, "Logic synthesis for engineering change," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 3, pp. 282–292, 1999.

[21] K.-H. Chang, I. L. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 184–188, 2008.

[22] ——, "Automating postsilicon debugging and repair," *IEEE Computer*, vol. 41, no. 7, pp. 47–54, 2008.

[23] R. K. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification*, 2010, pp. 24–40.

[24] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Theory and Applications of Satisfiability Testing*, 2003, pp. 502–518.