SEChecker: A Sequential Equivalence Checking Framework Based on Kth Invariants

Feng Lu and Kwang-Ting Cheng, *Fellow, IEEE*

***Abstract*—In recent years, considerable research efforts have been devoted to utilizing circuit structural information to improve the efficiency of Boolean satisfiability (SAT) solving, resulting in several efficient circuit-based SAT solvers. In this paper, we present a sequential equivalence checking framework based on a number of circuit-based SAT solving techniques as well as a novel invariant checker. We first introduce the notion of kth invariants. In contrast to the traditional invariants that hold for all cycles, kth invariants are guaranteed to hold only after the kth cycle from the initial state. We then present a bounded model checker (*BMChecker*) and an invariant checker (*IChecker*), both of which are based on circuit SAT techniques. Jointly, *BMChecker* and *IChecker* are used to compute the kth invariants, and are further integrated in a sequential circuit SAT solver for checking sequential equivalence. Experimental results demonstrate that the new sequential equivalence checking framework can efficiently verify large industrial designs that cannot be verified by existing solutions.**

***Index Terms*—Formal verification, invariant, sequential equivalence checking.**

I. Introduction

With the increasing use of sequential optimization techniques for improving performance, area utilization, power dissipation, etc., sequential equivalence checking (SEC) has became an important task for functional verification. In the past few years, several methods and techniques have been proposed to solve the SEC problem. Binary decision diagrams (BDDs) [1] have been widely used for implementing various symbolic techniques [2]–[5]. In these approaches, a product machine is constructed over the two finite state machines (FSMs) under verification [2]–[4]. The output of the product machine is of the value 1 if all corresponding outputs of the two FSMs are equivalent; otherwise, the output of the product machine is of the value 0. For a given initial state, the fix-point computation is carried out on the product machine to compute the set of reachable states. The equivalence of the two FSMs can be proved by checking if the output of the product machine is constant 1 under all reachable states. These algorithms are suitable for small- to medium-sized circuits, and may not scale well for larger designs.

Structural similarity is explored and used in [6], [7] for reduction of SEC complexity. In both approaches, induction-based proof and fix-point calculation are applied to detect sequential equivalent signals.

Manuscript received September 17, 2006; revised July 06, 2007. First published April 17, 2009; current version published May 20, 2009.

F. Lu is with Cadence Design Systems, Fremont, CA 94536 USA (e-mail: lufeng1204@gmail.com).

K.-T. Cheng is with University of California, Santa Barbara, CA 93106 USA.

Digital Object Identifier 10.1109/TVLSI.2008.2005311

In [6], a computational model, called *miter*[10], was built from the two sequential circuits under verification. In this model, starting from the given initial state, if any input sequence exists that would produce a constant 1 at the miter's output, then the two circuits are not equivalent. Otherwise, they are equivalent. Through this miter model, the SEC problem is transformed into a sequential backward justification problem, and sequential Automatic Test Pattern Generation (ATPG) techniques [8], [9] can then be used address this problem.

The method proposed in [7] introduces a two-timeframe, assume-then-prove circuit model (TTAPCM) of the miter for efficient calculation of the maximum correspondence relationship. In the first timeframe, a candidate list of equivalent signals, which are heuristically identified, are assumed to be equivalent and, thus, constraints corresponding to all these equivalences are added to the circuit model in this timeframe. The correctness of these equivalences is then verified in the second timeframe. False candidates are removed from the candidate list. This procedure iterates until a fix-point is reached. The equivalences between signals that remain at the end of this process form the maximum correspondence relation. Under the maximum correspondence relationship, if the output of the miter is constant 0, then the two circuits are indeed equivalent. Otherwise, it is not conclusive whether or not the circuits are equivalent. Therefore, this method is not complete.

In recent years, Boolean satisfiability (SAT) has attracted tremendous research efforts, resulting in several efficient SAT solver packages such as zChaff [11], Berkmin [12], SATO [35], GRASP [36], MiniSAT [37], and C-SAT [13]. State-of-the-art SAT algorithms implemented in these tools have demonstrated their capability and efficiency for handling large industrial SAT problems. Furthermore, circuit structural information has proved very important for efficiently solving circuit-based SAT problems. The utilization of circuit structural information to improve the efficiency of SAT solving has been investigated in [13], [25]–[30].

SAT-based techniques have been extensively used in various approaches to address SEC problems. The main approaches include bounded model checking (BMC) [15], [16], unbounded model checking (UMC) [14], [23], [24], [39], the interpolation-based method [20], and k-induction-based invariant computation [21], [22].

BMC systematically searches for those counter-examples with a length bounded by some integer k. The bound k starts from 0 and is increased until a counter-example is found or a given limit is reached. BMC can be viewed as a forward search method for finding a path from the given initial state to the objective. If such paths exist, BMC can find the shortest one among all possible solutions. Therefore, BMC is an efficient checking approach for those problems with a relatively short counter-example. However, BMC is not complete since a

property that has been proven true at depths 0 through k may not be true at depths greater than k. To achieve completeness, a diameter-based method was introduced in [18], [19] to find the true upper bound k for the property to be checked.

The sequential SAT solver *Seq-SAT*[14] employs backward search techniques to find an input sequence that can either satisfy the objective from the given initial state or else can prove that no such input sequence exists. *Seq-SAT* uses the combinational SAT solver C-SAT [13] as the underlying engine and utilizes circuit structural information for better decision ordering. It has been proven that the *Seq-SAT* algorithm is complete. Intuitively, *Seq-SAT* can be employed to solve the objective of the miter output being a constant 1 for checking sequential equivalence; however, directly applying *Seq-SAT* to the SEC problem has limited scalability.

The interpolation-based method [20] approximates the set of reachable states, allowing BMC to be extended for use in unbounded model checking.

A SAT-based, fixed-point calculation method was proposed in [32] to compute equivalence invariants on TTAPCM. The authors adopt the Stålmarck method in their SAT engine and incorporate k-induction [21], [22] and unique state induction [21], [22] to achieve complete induction. K-induction assumes that a property holds at the first k timeframes (without applying the initial state), and proves whether the property still holds at the $(k+1)$th timeframe. The unique state induction enforces the rule that no equivalent states will appear within the first k timeframes.

In [38], Amla *et al.* analyzed and compared the above approaches in an industrial environment where extensive industrial test-cases are used in their experiments. The version of *SATORI* utilized in their experiments is *Seq-SAT* described in [14]. Their experimental results show that BMC is good for those test-cases with short counter-examples. While *Seq-SAT* is comparable with state-of-the-art techniques for unsatisfiable test-cases, its performance for satisfiable test-cases is worse than forward search algorithms. One reason for its poor performance for satisfiable test-cases is likely that the backward search algorithm used in *Seq-SAT* does not consider much initial state information.

In this paper, we propose an SEC framework, named SEChecker, that supports incremental verification. The framework, designed to tackle large, industrial SEC problems, consists of three major components: a bounded model checker, *BMChecker*, an inductive invariant checker, *IChecker*, and the sequential SAT solver, *Seq-SAT*. *BMChecker* employs the explicit learning technique first introduced in C-SAT [13] to improve BMC performance. *IChecker* performs fix-point calculation based on the TTAPCM to find kth invariants. The kth invariants are generalizations of the traditional invariants. A kth invariant is a property that holds at cycles k and after, assuming the initial state starts at cycle 0. It may or may not hold from cycle 0 to cycle $k-1$. The reachable states after k cycles from the initial state should satisfy kth invariants. Thus, kth invariants could be taken as constraints to prune the search space in *Seq-SAT*. The details of this approach will be described later.

BMChecker, *IChecker*, and *Seq-SAT* have different strengths and complement one another. *BMChecker* is most suitable for satisfiable problems with short solutions. *IChecker* is ideal for unsatisfiable problems that can be proved by induction. However, neither of these checkers is complete. *Seq-SAT* is complete and could be used for general problems. The invariants found by *IChecker* and the learned information derived by *BMChecker* can also be used by *Seq-SAT* to effectively bound the search space.

This paper is an extension of our previous conference paper [31]. The major improvements to the earlier conference publication include the following.

1) An incremental simulation technique is now employed in *BMChecker* to derive equivalent or constant signal candidates to avoid the simulation of the whole expanded model when expanding a new timeframe.
2) The inductive invariant checker, *IChecker*, is utilized to derive kth invariants instead of *IProver*[31]. *IChecker* uses a new method to simplify the TTAPCM and to avoid adding unnecessary constraints for those signals assumed to be equivalent or constant. When checking invariant candidates, sub-problems are generated from the original, complex problem based on the invariant candidates, and these sub-problems are solved following their topological order. Implementing these techniques allows *IChecker* to achieve significant performance improvement over *IProver*.
3) More experiments have been conducted for industrial test-cases as well as for publicly available benchmarks.

The rest of this paper is organized as follows. Section II gives the background. In Section III, we present the basic flow of our SEC framework and describe some circuit simplification techniques with low computational costs. Sections IV–VI describe in detail the three major components of our SEC framework: *BMChecker*, *IChecker*, and *Seq-SAT*. Section VII shows the experimental results of the SEC framework on some benchmarks from both industry and the public domain. Section VIII concludes this paper.

II. Background

We could view a sequential circuit using the Huffman circuit model. One timeframe of a sequential circuit is viewed as a combinational circuit in which each flip-flop is converted into two corresponding signals: a pseudo primary input (PPI) and a pseudo primary output (PPO). Timeframe expansion is achieved by connecting the PPIs of the timeframe $i+1$ to the corresponding PPOs of the previous timeframe i. The timeframes are numbered starting from 0.

The Huffman circuit model and the circuit models for one timeframe and two timeframes are illustrated in Fig. 1(a)–(c).

The miter circuit for two circuits under verification is built by joining their primary inputs, connecting the corresponding output pairs by XOR gates, and then connecting the XOR outputs to an OR gate. Fig. 2 shows an example of a miter circuit. The two circuits under verification are equivalent if and only if, for all states reachable from the initial state, the output of the miter circuit is the constant 0.

We focus on the sequential SAT problem of either finding an input sequence from the given initial state to set the miter output to 1, or else proving no such sequence exists. We call this the output-1-SAT problem. Without loss of generality, we consider sequential circuits with only one primary output for the rest of the discussion.

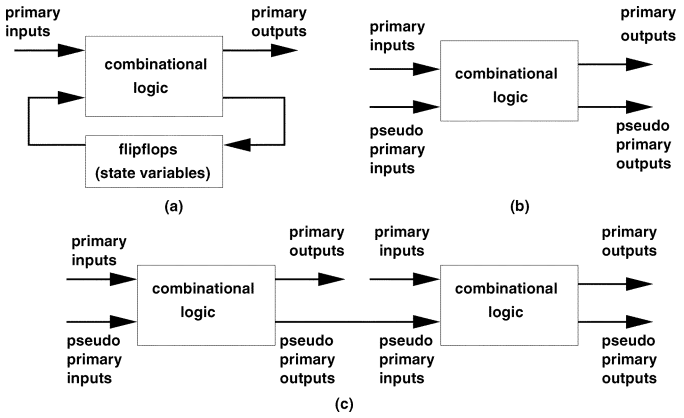


Fig. 1. Illustration of the Huffman circuit model, the circuit model for one timeframe, and the model for two timeframes.

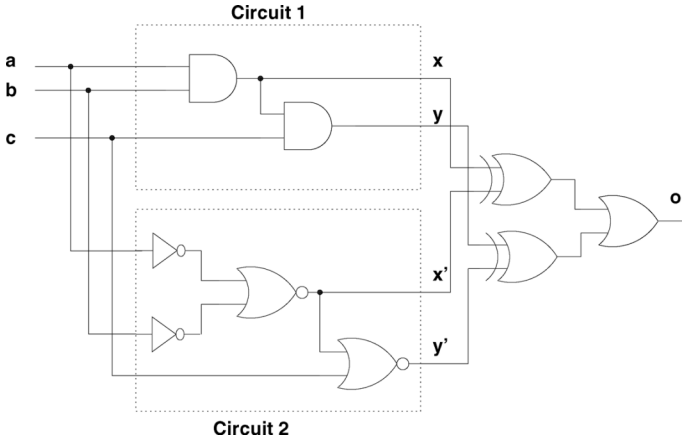


Fig. 2. Illustration of a miter circuit.

Definitions

Given a sequential circuit SC with one primary output o, we denote PI as the set of its primary inputs I as its input space $\{0,1\}^{|PI|}$, FF as the set of its flip-flops, and S as its state space $\{0,1\}^{|FF|}$. The output (input) of a flip-flop corresponds to a pseudo primary input (pseudo primary output). PPI denotes the set of pseudo primary inputs and PPO is the set of pseudo primary outputs. In addition, INT denotes the set of internal signals, and $V = \mathrm{PI} \bigcup \mathrm{INT} \bigcup \mathrm{PPI} \bigcup \mathrm{PPO} \bigcup \{o\}$ denotes the set of all signals of SC. For each flip-flop ff, $\mathrm{INIT}_{\mathrm{ff}}$ denotes its initial value. This could be 0, 1, or X. If $\mathrm{INIT}_{\mathrm{ff}}$ is X, then the initial value in ff could be either 0 or 1. The initial values of all flip-flops form the initial state, which is denoted as s_0.

Given a signal $v \in V$, we use $FI(v)$ to denote the set of its fan-ins, and $FO(v)$ to denote the set of its fan-outs. We levelize the signals in the sequential circuit based on its one-timeframe combinational circuit model. The level of a signal v, denoted as $L(v)$, is computed as follows:

1) $L(v) = 0$ if $FI(v) = \emptyset$;
2) $L(v) = 1 + \max((L(i_1), L(i_2), \ldots, L(i_{|FI(v)|}))$ where $i_1, i_2, \ldots, i_{|FI(v)|} \in FI(v)$.

For a signal $v \in V$, v^i denotes the corresponding signal of v at timeframe i. Similarly, PI^i represents the set of primary inputs at timeframe i, V^i the set of all signals at timeframe i, etc.

We refer to either a signal v or its negation $\overline{v}$ as a literal, v as a positive literal, and $\overline{v}$ as a negative literal. A clause is a logical **OR** of one or more literals.

The logic function of signal v in the one-time-frame combinational circuit SC can be denoted as $g_v(pi_1, pi_2, \ldots, pi_{|\mathrm{PI}|}, ppi_1, ppi_2, \ldots, ppi_{|\mathrm{PPI}|})$ where $pi_1, pi_2, \ldots, pi_{|\mathrm{PI}|} \in \mathrm{PI}$, and $ppi_1, ppi_2, \ldots, ppi_{|\mathrm{PPI}|} \in \mathrm{PPI}$. To be concise, we denote this function as g_v, instead of $g_v(pi_1, pi_2, \ldots, pi_{|PI|}, ppi_1, ppi_2, \ldots, ppi_{|\mathrm{PPI}|})$. The sequential logic function of signal v in SC is denoted as $f(v)$.

We also use $f^i(v)$ to denote the logic function of signal v^i, $v^i \in V^i$, and $f^i(v)|s_0$ to denote the logic function of signal v^i with respect to the initial state s_0. $f^i(v)|s_0$ can be derived as follows:

1) $f^i(v)|s_0 = 0$ if $i = 0, v \in \mathrm{PPI}$, and $\mathrm{INIT}_v = 0$;
2) $f^i(v)|s_0 = 1$ if $i = 0, v \in \mathrm{PPI}$, and $\mathrm{INIT}_v = 1$;
3) $f^i(v)|s_0 = v$ if $i = 0, v \in \mathrm{PPI}$, and $INIT_v = X$;
4)

$$f^i(v)|s_0 = g_v(pi_1^i, pi_2^i, \ldots, pi_{|\mathrm{PI}|}^i, f^i(ppi_1)|s_0, f^i(ppi_2)\,|s_0, \ldots,\, f^i(ppi_{|PPI|})|s_0)$$

if $v \notin \mathrm{PPI}$ where $pi_1^i, pi_2^i, \ldots, pi_{|\mathrm{PI}|}^i \in \mathrm{PI}^i$, and $ppi_1, ppi_2, \ldots, ppi_{|\mathrm{PPI}|} \in PPI$;

5) $f^i(v)|s_0 = f^{i-1}(u)|s_0$ if $i > 0, v \in \mathrm{PPI}, u \in \mathrm{PPO}$, and u is the pseudo primary output signal of the same flip-flop as v.

According to the above notations, we give the following definitions.

1) A signal $v \in V$ is a strong constant signal if and only if $g_v = 0$ or $g_v = 1$.
2) Two signals u and v, where $u, v \in V$, are strong equivalent if and only if $g_u = g_v$.
3) A signal $v \in V$ is a weak constant signal if and only if $f^i(v)|s_0 = 0$ for all $i \geq 0$, or $f^i(v)|s_0 = 1$, for all $i \geq 0$.
4) Two signals u and v, where $u, v \in V$, are weak equivalent if and only if $f^i(u)|s_0 = f^i(v)|s_0$, for all $i \geq 0$.
5) A property p involving n signals $v_1, v_2 \ldots, v_n$ is a Boolean function $p(f(v_1), f(v_2), \ldots, f(v_n))$, where $v_k \in V$ for $1 \leq k \leq n$.
6) A property p in timeframe i, denoted as p^i, is the Boolean function $p(f^i(v_1), f^i(v_2), \ldots, f^i(v_n))$, where $v_k \in V$ for $1 \leq k \leq n$.
7) A property p in timeframe i with respect to the initial state s_0, denoted as $p^i|s_0$, is the Boolean function $p(f^i(v_1)|s_0,\ f^i|s_0(v_2), \ldots, f^i|s_0(v_n))$, where $v_k \in V$ for $1 \leq k \leq n$.
8) A property p is a kth invariant if and only if $p^i|s_0 = 1$, for all $i \geq k$.
9) A set S of properties—$p_1, p_2, \ldots, p_{|S|}$—is an inductive property set (IPS) if and only if

$$(p_1^0 = 1 \wedge p_2^0 = 1 \cdots \wedge p_{|S|}^0 = 1) \Rightarrow (p_1^1 = 1 \wedge p_2^1 = 1 \cdots \wedge p_{|S|}^1 = 1).$$

In particular, $\emptyset$ is an IPS.

10) If a set S is an IPS, then any property p of S is called an *guarded invariant* of S. We will use the term *invariant* for brevity.

Lemma 1: If both A and B are IPSs, then $A \bigcup B$ is an IPS.

Proof: The proof can be derived directly from the definition. □

Lemma 2: For any finite set S of properties, there is one and only one maximal subset TS, such that TS is an IPS.

Proof: Since $\emptyset$ is an IPS and S has a finite number of elements, there exists at least one maximal subset that is an IPS. Assume there are two different maximal subsets T and U, which are IPSs according to Lemma 1; $T \bigcup U$ is an IPS and is greater than T and U, which violates the assumption that T and U are maximal. □

Note that strong equivalent (constant) signals are equivalent (constant) in the one-timeframe combinational circuit in which PPIs are treated the same as PIs. Weak equivalent (constant) signals are equivalent (constant) under all reachable states of the initial state. So, by definition, strong equivalent (constant) signals are also weak equivalent (constant) signals, but not vice versa. Weak equivalent or constant signals are special cases of 0th invariants in which the function p represents the equivalence or constancy relationship. Therefore, these equivalence or constancy relations can be viewed as some kind of invariants. Furthermore, if a property is a kth invariant, it is also an ith invariant for any $i > k$.

The invariant checker (*IChecker*) addresses the following problem. Given a set S of properties (called invariant candidates), find the maximal subset that is an IPS. It can be seen that if all the invariant candidates in S hold at timeframe k, the properties in the maximal IPS subset derived from S are kth invariants.

In our SEC framework to solve output-1-SAT problem, we adopt an incremental solving strategy. Before solving the original problem, invariants are identified to reduce the problem complexity. Easy-to-identify invariants are derived first to help reduce the computing complexity of hard-to-identify invariants. For example, strong equivalent signals can be derived easily by checking combinational equivalences in the one-timeframe combinational model. The computation of kth invariants is more difficult and would involve k-bounded model checking and fix-point calculation on the TTAPCM.

A. Fix-Point Calculation for Guarded Invariants

Given a set S of invariant candidates, the maximum inductive property set can be computed on a TTAPCM by induction. The first timeframe of the TTAPCM is used for adding constraints derived from the invariant candidates. These constraints enforce the invariant candidates to be held in the first timeframe. The candidates are then checked in the second timeframe and the false candidates are removed from S. The above procedure iterates until a fix-point is reached. The invariant candidates left in S form the maximum inductive property set. Fig. 3 gives an example of a TTAPCM for the equivalence invariant candidate $(v_1 = v_2)$.

Algorithm II.1 summarizes the process, where SC denotes the sequential circuit and S the set of invariant candidates. Function $\mathrm{AssumeInvariants}(\mathrm{TTC}, S)$ enforces the invariant candidates of S to be held in the first timeframe of TTC, and $\mathrm{ProveInvariants}(\mathrm{TTC}, S)$ checks the validity of the invariant candidates of S in the second timeframe of TTC, and removes the false candidates from S. If any false candidate is found in $\mathrm{ProveInvariants}(\mathrm{TTC}, S)$, it returns false. Otherwise it returns true. This algorithm is the basis of our invariant checker

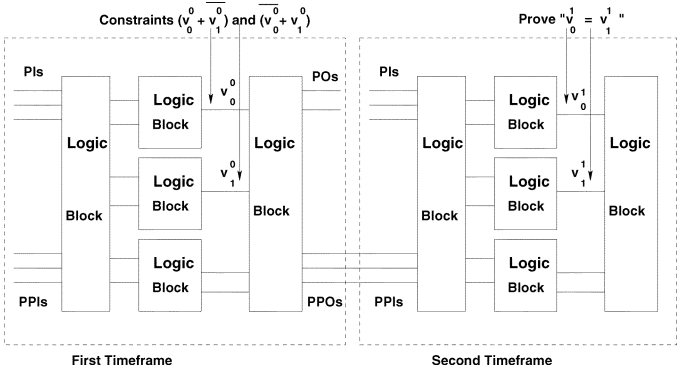


Fig. 3. Example of the TTAPCM.

IChecker. The details of the algorithm will be discussed in Section V.

Algorithm II.1: COMPUTEINVARIANTS(SC, S)

fixpoint $\leftarrow$ false
while (!fixpoint)
do
{ Create a two-timeframe model TTC of SC
Assume Invariants(TTC, S)
fixpoint $\leftarrow$ Prove Invariants(TTC, S)
return (S)

III. SEQUENTIAL EQUIVALENCE CHECKING

A. Proposed Framework

Fig. 4 shows the basic flow in our sequential equivalence checking framework SEChecker. The miter circuit of the two circuits to be verified is first constructed. The miter circuit is then simplified by detecting and merging strong equivalent or constant signals, and also by flip-flop mapping. After simplification, if the miter output o becomes a constant-0 signal, then the two circuits under verification are proved equivalent. Otherwise, *BMChecker* is applied to check whether there is a solution of length between 0 and a certain limit k, to set o to 1. If such a solution exists, the two circuits under verification are proved not equivalent. If no such solution exists, the equivalent or constant signals derived by *BMChecker* at timeframe k are considered as initial candidates of kth invariants. Among these candidates, *IChecker* is employed to identify true kth invariants. If "$o = 0$" is identified as a kth invariant, we can conclude that the two circuits are equivalent because *BMChecker* has proved that "$o = 0$" is true from timeframes 0 to k. If such a conclusion cannot be made, all kth invariants derived by *IChecker* are inserted as constraints in the miter circuit. *Seq-SAT* is then used to solve the sequential SAT problem "$o = 1$". By using both simplification and constraints in the miter circuit, the search space is often dramatically reduced and, thus, the initial problem is much more solvable by using *Seq-SAT*.

In the following, we discuss the simplification of sequential circuits based on strong equivalent signals, strong constant signals, and flip-flop mapping. The relevant details of the three key components *BMChecker*, *IChecker* and *Seq-SAT* are discussed in Sections IV–VI.

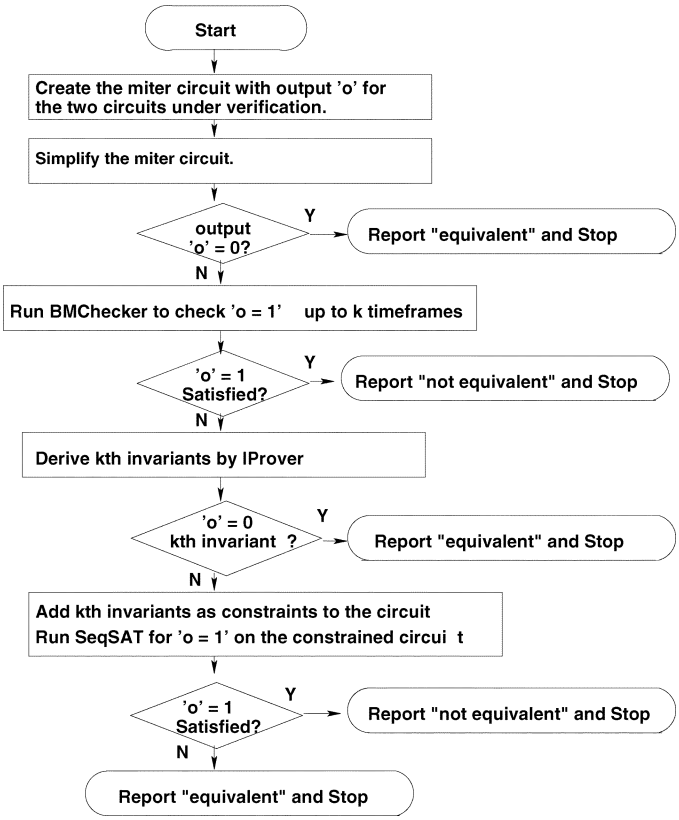


Fig. 4. Proposed flow of sequential equivalence checking.

B. Circuit Simplification

Strong equivalent (constant) signals are combinational equivalent (constant) signals in the one-timeframe combinational circuit. They can be efficiently derived by combining the explicit learning technique proposed in C-SAT [13] and the structure-based method proposed in [25]. In [13] and [25], the circuit is converted into a netlist consisting of only two-input AND gates and INVERTERs, where the INVERTERs are represented by edge attributes. The structure-based method can efficiently detect trivial equivalent or constant signals.

The explicit learning technique used in C-SAT can detect nontrivial strong equivalent or constant signals. For example, to check the strong equivalence of two signals u and v, C-SAT solves two SAT sub-problems $(u = 0, v = 1)$ and $(u = 1, v = 0)$. If both of the sub-problems are unsatisfiable, then u and v are found to be strong equivalent. The candidates of strong equivalent or constant signals can be obtained by simulating random vectors, and then refined by the counter-examples derived in the explicit learning procedure [16]. Identified strong equivalent or constant signals are merged to simplify the sequential circuits.

The flip-flop mapping step [6] attempts to detect sequential equivalent or constant flip-flops, which are special cases of the weak equivalent or constant signals. The equivalent or constant flip-flops can be derived by using fix-point calculation on the one-timeframe, combinational circuit. Merging equivalent or constant flip-flops might result in more strong equivalent or constant signals, which can be used for further simplification.

Table I shows the experimental results of circuit simplification based on both detection of strong equivalent or constant signals and flip-flop mapping. In these experiments, the "case" benchmarks are the miter circuits created from industrial examples. In each of the miter circuits, one sub-circuit is a gate-level implementation synthesized from its System-C behavioral-level description, and the other is a gate-level circuit synthesized from its Verilog register-transfer-level (RTL) description. For some of these industrial cases, there are bugs in the Verilog RTL description.

TABLE I
EXPERIMENTAL RESULTS ON CIRCUIT SIMPLIFICATION

Circuit	Original Information			After Simplification		run time
	#ff	#X	#g	#ff	#g	(sec)
case1	159	95	5818	128	5032	2
case2	273	173	7367	273	7297	4
case3	183	98	2276	179	2216	4
case4	116	96	2539	116	2539	1
case5	396	0	2672	362	2596	2
case6	457	382	61589	410	53528	100
case7	461	386	62333	414	54237	103
s5378	545	0	4146	301	3150	2
s13207	1715	0	9842	553	3597	6
s35932	4330	0	32306	3777	29536	16
s38417	4070	0	27538	2892	23031	14
s38584	4935	0	36486	2179	17880	18

The "s" benchmarks are miter circuits created from ISCAS-89 sequential benchmark circuits. For each of these circuits, one sub-circuit is the original gate-level benchmark circuit; the other is its retimed and optimized version produced by the ABC package [33] from Berkeley.

All experiments were run on a P4 2 GHz Linux machine with 2 GB memory.

The two "#ff" columns show the number of flip-flops in the circuits before and after simplification. The two "#g" columns give the number of gates before and after simplification. The column "#X" is the number of flip-flops whose initial value is X, and the column "runtime" gives the CPU time, expressed in seconds, for the simplification process. From the results, it can be seen that some circuits achieve nontrivial simplification while the run time is relatively small. These experiments indicate the amount of reduction in circuit size that can be achieved for these testcases by traditional combinational equivalence checking techniques. These simplified miter circuits are then used as the starting points for further experiments in later sections for evaluating the techniques proposed in this paper.

IV. BOUNDED MODEL CHECKING

BMChecker is a bounded model checker developed based on C-SAT. Similar to other BMC tools, *BMChecker* starts from timeframe 0 where the initial state is applied and checks whether the given objective is satisfiable or not on the one-timeframe combinational circuit. If the objective is unsatisfiable, it expands the circuit by adding a new timeframe and then checks the objective again on the expanded circuit. The procedure iterates until either a solution is found or else a given limit is reached.

In *BMChecker*, whenever a new timeframe is added, it employs the explicit learning technique and the structure-based method to detect equivalent or constant signals, and, in turn, to simplify the expanded circuit before solving the objective. The approach in [16] also simplifies the expanded circuit model before solving the objectives. The difference is that in *BMChecker*, only one unfolding of timeframes is performed and the initial

state is applied to simplify the unfolded timeframes. While in [16], BMC is based on a dual unfolding scheme. The first unfolding is performed without applying the initial state, and the unfolded timeframes are simplified. The simplified timeframes are then applied in the second unfolding, in which the initial state is enforced and properties are verified. The advantage of the dual unfolding scheme is that it can be used to prove the unsatisfiability of some cases. In *BMChecker*, the unfolded timeframes can be more simplified than those in [16], since more signals can become equivalent or constant when the initial state is considered.

The main concepts used in *BMChecker* can be summarized as follows.

1) Whenever a new timeframe is added to the circuit model for BMC, the 32-bit incremental parallel simulation technique is employed to simulate only the newly added timeframe (NAT) instead of the whole expanded circuit model. For each signal of the NAT, its signature is computed from the simulation results of different simulation runs by the bit-wise XOR operation. The signatures are recorded and used to derive equivalent and constant signal candidates.
2) Based on the candidates derived above, *BMChecker* employs the explicit learning technique and the structure-based method to detect equivalent or constant signals, and, in turn, to simplify the expanded circuit before solving the objective.

In the incremental simulation of the NAT, random values are assigned to the PIs, and the values assigned to the PPIs originate from the recorded simulation results of the corresponding PPOs in the previous timeframe. When the simulation is complete, the simulation results of the PPOs in the NAT are recorded for the simulation of those timeframes that will be expanded in the future. A constant signal candidate is checked for constancy in the explicit learning stage of *BMChecker*. It is checked for constancy only if the signal is in the NAT. The two equivalent signal candidates are checked for equivalence only if at least one of the two signals is in the NAT. The counter-examples derived during the explicit learning process are used to further refine the candidate list as in [16].

In order to compare the performance of *BMChecker* and the SAT-Sweeping algorithm proposed in [16], we implemented the SAT-Sweeping algorithm [16] on our framework. When implementing this algorithm, we also referred to the implementation of SAT-Sweeping in the OpenAccess Gear toolkit [41]. The comparison is shown in Table II. The benchmarks used in these experiments are those same simplified miter circuits listed in Table I. In Table II, "$\#\mathrm{frame}$" is the number of timeframes expanded, "$\#\mathrm{tg}$" is the total number of gates in the expanded timeframes, "$\#\mathrm{lg}$" gives the number of gates left after equivalent or constant signals are merged by *BMChecker*, "$\mathrm{runtime(sec)}$" shows the CPU time elapsed in seconds. The results demonstrate that *BMChecker* not only produces useful information such as equivalent or constant signals as candidates of invariants, but also has superior performance for BMC.

In addition, Figs. 5 and 6 show the CPU time comparison with respect to a range of expanded-timeframe counts for testcases "case2" and "case3" respectively. In these two figures, we record the CPU time as 36000 seconds if it exceeds 36 000 s. While both approaches suffer from performance degradation when the number of expanded timeframes increases, the figures clearly indicate that our *BMChecker* scales better than the SAT-Sweeping-based approach.

TABLE II
EXPERIMENTAL RESULTS OF *BMCHECKER*

Circuit	#frame	#tg	BMChecker		SAT-SWEEP
			#lg	run time (sec)	run time (sec)
case1	10	50293	39684	5	9
case2	50	364703	150577	74	300
case3	50	110453	24171	10	162
case4	50	126803	37107	13	118
case5	50	127953	59146	138	280
case6	20	1070483	158787	89	101
case7	20	1084683	167189	123	152
s5378	50	157353	58855	148	367
s13207	50	179703	40216	31	169
s35932	20	590663	219974	94	347
s38417	20	460563	80144	36	48
s38584	20	357543	87922	193	821

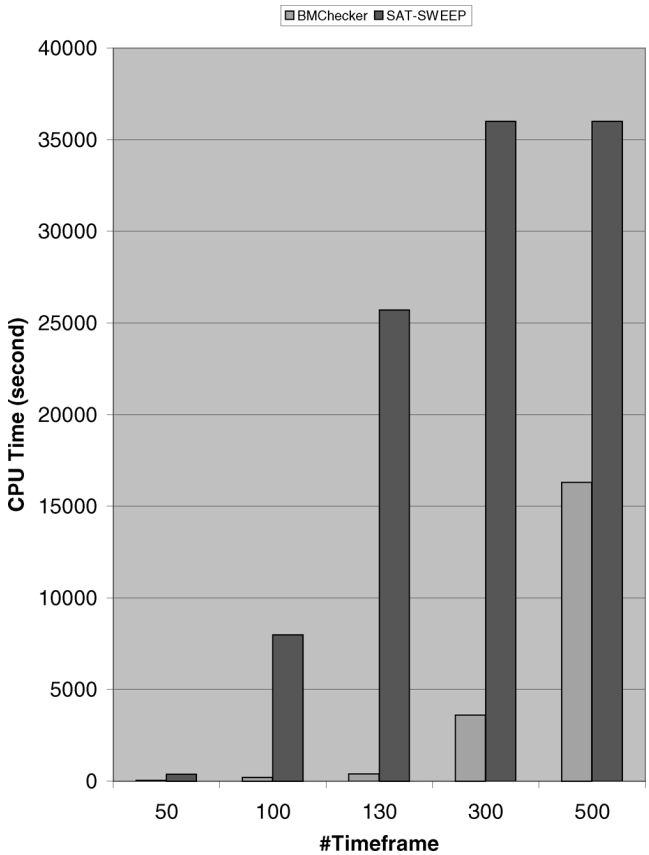


Fig. 5. CPU time comparison with respect to the bound (i.e., # of timeframes) for case2.

The equivalent or constant signals identified by *BMChecker* in timeframe k can be considered as the candidates of kth equivalence or constant invariants. Based on these candidates, *IChecker* identifies true kth invariants.

Note that when a new timeframe i is added to the circuit, kth equivalence or constant invariants for $k < i$ can be used to simplify timeframe i. Therefore, once a kth invariant is identified, it can be applied to all timeframes whose index is greater than k, thus avoiding unnecessary repeated computations in these timeframes.

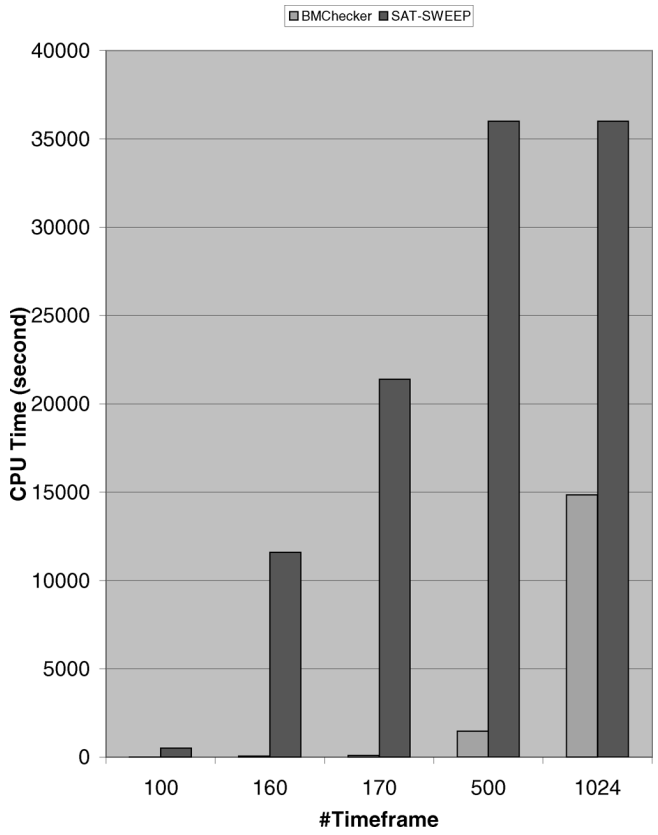


Fig. 6. CPU time comparison with respect to the bound (i.e., # of timeframes) for case3.

V. Invariant Checking

Invariant checker *IChecker* is based on C-SAT. It performs fix-point calculation on a TTAPCM.

The equivalent or constant signals computed at timeframe k by *BMChecker* can be used by *IChecker* as the initial invariant candidates. After fix-point calculation, the remaining candidates that hold at timeframe k will hold at timeframe $k+1$, thus hold at any future timeframes. Therefore, these remaining candidates are proved to be kth invariants. We first describe some details of *IChecker* before applying it to compute kth invariants.

In our implementation, *IChecker* accepts three types of invariant candidates.

1) *Constant Invariant Candidates (CstICs):* Each CstIC is an equivalence function of a signal and a binary value, and is represented by a literal. A positive literal v represents $(v = 1)$, while a negative literal $\overline{v}$ represents $(v = 0)$. All such literals are put in one *constant candidate group*.
2) *Equivalence Invariant Candidates (EICs):* each EIC is an equivalence function or inverted equivalence function of two signals. e.g., $(v_1 = v_2)$ and $(v_3 = \overline{v_4})$. Instead of explicitly representing EICs by signal pairs, we represent them in *equivalence candidate groups*. Signals involved in an EIC are placed in the same equivalence candidate group. For example, three EICs $(v_1 = v_2)$, $(v_1 = \overline{v_3})$ and $(v_4 = v_5)$ are represented by two equivalence candidate groups $\{v_1, v_2, \overline{v_3}\}$, and $\{v_4, v_5\}$.
3) *Clause Invariant Candidates (ClsICs):* Each ClsIC is a clause of one or more literals and is represented by a *clause candidate group* whose member(s) is (are) the literal(s) in the clause.

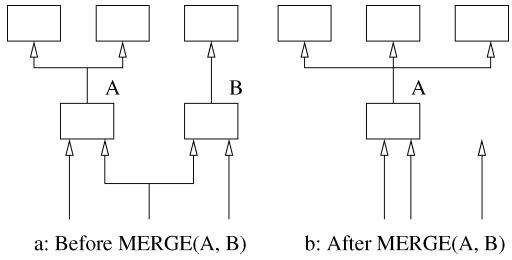


Fig. 7. Illustration of $\mathrm{MERGE}(A, B)$.

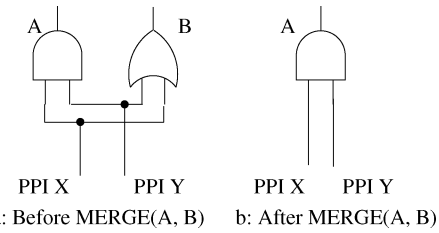


Fig. 8. Loss of constraints due to $\mathrm{MERGE}(A, B)$.

A. "Assuming" Step of Invariant Candidates

Given a TTAPCM of a sequential circuit, the previous methods directly add constraints to make the invariant candidates hold in the first timeframe [7], [31]. For example, for a constant invariant candidate $(v_1 = 1)$, these methods add a unique literal clause (v_1^0), and for an equivalence invariant candidate $(v_1 = v_2)$, they add two clauses $(v_1^0 + \overline{v_2^0})$ and $(\overline{v_1^0} + v_2^0)$. However, directly adding such constraints as shown above does not simplify the computational model.

The general signal-merge operation $\mathrm{MERGE}(A, B)$ replaces each wire connected to signal B by a wire connected to A, and removes the gate producing signal B [25]. Fig. 7 illustrates this merge operation. In [16], the merge operations are applied to equivalent and constant signals in order to simplify the verification problem, which can achieve significant performance enhancement. Applying $\mathrm{MERGE}()$ to signals that are assumed equivalent or constant in the assume-then-prove circuit model, however, could result in the loss of the assumed constraints. Fig. 8 shows one such case.

In this case, $(A = B)$ is an equivalence invariant candidate. If A and B are constrained to be equivalent (Fig. 8(a)), then the state space of (X, Y) is constrained to be (0, 0) and (1, 1). Simply merging A and B by $\mathrm{MERGE}(A, B)$ as shown in Fig. 8(b) will result in the loss of those constraints imposed to the state space of (X, Y). This problem has been pointed out by Mony *et al.* in [34]. It is mentioned in [34] that when signals A and B are constrained to be equivalent, each wire connected to signal B is replaced by a wire connected to A, and constraints are added between A and B to enforce equivalence. However, no further details were given. Based on the same idea, we address the problem of the loss of constraints by implementing a new operation $\mathrm{CMERGE}(A, B)$ This operation not only performs the $\mathrm{MERGE}(A, B)$ operation, but also adds constraints among A and the fan-ins of B to ensure that A is equal to B under these constraints. Fig. 9 illustrates the concept of $\mathrm{CMERGE}()$.

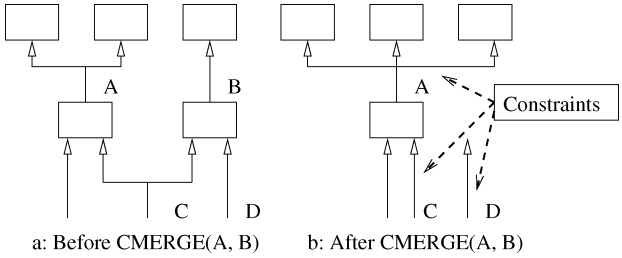


Fig. 9. Illustration of the $\mathrm{CMERGE}()$ operation.

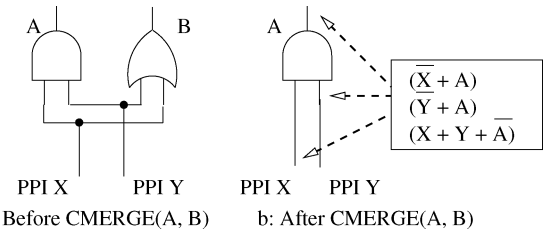


Fig. 10. Example of $\mathrm{CMERGE}(A, B)$.

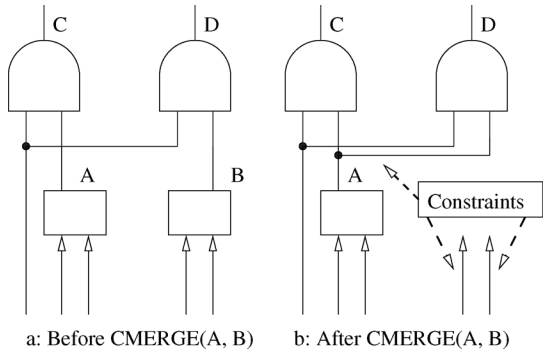


Fig. 11. Example: $\mathrm{CMERGE}(A, B)$ enables $\mathrm{MERGE}(C, D)$.

The specific constraints added by $\mathrm{CMERGE}()$ for the example shown in Fig. 8 are given in Fig. 10.

The advantage of applying $\mathrm{CMERGE}()$ is more than just removing a signal from the computational model. More importantly, similar to the approaches of [25], [34], after applying such operations, a structure-based method [25] can be employed to further simplify the computational model and to avoid adding unnecessary constraints. Fig. 11 illustrates this point. In this example, $(A = B)$ and $(C = D)$ are equivalence invariant candidates. After performing the $\mathrm{CMERGE}(A, B)$ operation to enforce the equivalence of A and B, as shown in Fig. 11(b), we can use a structure-based method to detect that C and D are equivalent. This means that the constraints added to enforce the equivalence of A and B can also enforce the equivalence of C and D. So C and D can be merged without adding additional constraints. From this example, we can also see the advantages of processing the invariant candidates following the topological order from the inputs toward the outputs. If the invariant candidate $(C = D)$ is processed before $(A = B)$, the constraints among C and the inputs of D need to be added, which become redundant after $(A = B)$ is processed.

Note that, in our implementation, the inverters are represented as a wire (edge) property [25]. So, the merging of inverted equivalent signals can be conducted in a similar fashion by modifying the wire property. The $\mathrm{CMERGE}()$ operation is implemented to accept **literals** as its parameters, and it can merge signals of inverted equivalence.

To describe the algorithm for enforcing the candidates to hold in the first timeframe of a TTAPCM, we use LA to denote the array of literals of the sequential circuit SC. LA is sorted in ascending order of the signal level, and thus follows the topological order from inputs to outputs in the one-timeframe combinational model of SC. We also assume the literals in the candidate groups are sorted in ascending order by their indexes in LA. TTC is the two time frame circuit model of SC, and S is the set of invariant candidates. Algorithm V.1 summarizes the $\mathrm{AssumeInvariants}()$ process.

This algorithm first handles the constant and equivalence invariant candidates following the **topological order** from inputs to outputs. For a literal v, its corresponding literal in the first time frame of TTC, v^0, might have been merged in either an earlier $\mathrm{CMERGE}()$ operation or else in a structure-based simplification. So we must obtain the literal, u, equivalent to v^0 in TTC, before $\mathrm{CMERGE}()$ can be applied. This literal can be obtained by the function $\mathrm{GetUnmergedLiteral}(v, n, \mathrm{TTC})$ shown in Algorithm V.2 (Note that in TTC, if a signal s is merged with a signal t, and l is a literal of s, function $\mathrm{GetMerged(l)}$ returns the literal of t equivalent to l; otherwise $\mathrm{GetMerged(l)}$ returns NULL). In this function, n can be either 0 or 1. This indicates the first timeframe or the second timeframe of the TTC, since in each timeframe of the TTC, there is a signal corresponding to v.

Whenever $\mathrm{CMERGE}()$ is called, structure-based simplifications are performed on TTC.

Algorithm V.1: ASSUMEINVARIANTS(TTC, S)

$num \leftarrow$ the number of elements of LA
for $i \leftarrow 1$ **to** num
do
$v \leftarrow \mathrm{LA}[i]$
$u \leftarrow \mathrm{GetUnmergedLiteral}(v, 0, TTC)$
if ($v \in$ a constant candidate group)
then
$\mathrm{CMERGE}(1, u)$
Simplify TTC by structure based methods
else if ($v \in$ an equivalence candidate group EG)
then
$x \leftarrow$ the first member of EG
$y \leftarrow \mathrm{GetUnmergedLiteral}(x, 0, \mathrm{TTC})$
$\mathrm{CMERGE}(y, u)$
Simplify TTC by structure based methods
Process clause candidate groups

Note that, in this algorithm, to avoid generating combinational loops, when performing the merge operation between two signals, we require the signal with the larger index in LA to be merged to the signal with the smaller index. Thus, the function $\mathrm{GetUnmergedLiteral}$ returns the literal of the smallest index in LA among those literals merged. Also, LA will not be affected by the merge operations. Please note that while the circuit simplifications affect the topological order of the signals, but we do not perform any reordering.

The process for dealing with clause candidate groups is much simpler. For each candidate group, we replace its literals with

TABLE III
EXPERIMENTAL RESULTS OF THE ENHANCED SIMPLIFICATION TECHNIQUES

Circuit	k	#const	#eq	original method		our method	
				#g	#c	#g	#c
case1	0	248	490	10061	888	8668	86
case2	1	1790	1691	14591	3900	7408	66
case3	2	953	392	4421	1467	1501	50
case4	3	134	566	5075	828	3229	50
case5	4	677	986	5121	1895	2570	237
case6	5	9656	38826	107051	77190	20620	360
case7	6	9623	39148	108471	77805	21638	408
s5378	0	894	1645	6297	3080	2682	242
s13207	1	1438	1504	7191	3660	2208	353
s35932	2	1736	23300	59069	41722	15658	2209
s38417	3	8325	11673	46059	28007	10129	441
s38584	4	2319	12647	35757	18901	15565	1879

the corresponding unmerged literals of TTC in the first timeframe, and remove the redundant literals. If the resulting clause does not contain a positive literal and a negative literal of the same signal, the resulting clause is then added to TTC.

Algorithm V.2: GETUNMERGEDLITERAL(v, n, TTC)

$u \leftarrow v^n$ of TTC
while (GetMerged(u)! = NULL)
  do $u \leftarrow$ GetMerged(u)
return (u)

In our algorithm, since enforcing ClsIC constraints will not trigger the merge of the signals, we process the ClsICs after processing the CstICs and the EICs. If ClsICs are enforced earlier in the process, some signals in the ClsICs may be merged with other signals later, and, thus, this might introduce unnecessary overhead needed to update the enforced constraints. The CstICs and the EICs are processed following the topological order of the signals involved in them. Through the CMERGE operation and the structure-based simplification, the TTPACM can be greatly simplified.

The experimental results comparing our simplification techniques with those of the original techniques [7], [31] are shown in Table III.

The benchmarks used in these experiments are the same miter circuits as listed in Table II. The initial invariant candidates are generated based on the constant or equivalent signals in the kth timeframe of the circuit with respect to the given initial state, and are refined by random simulation. Note that, in random simulation, only the simulation results after k cycles are used to refine the candidates. Column "k" indicates that kth initial invariant candidates are derived. Column "#const" ("#eq") gives the number of signals in the initial constant (equivalence) invariant candidates. The two columns "#g" and "#c" show the numbers of gates and added constraints in the TTAPCMs of the original method and our method. The results clearly indicate that the TTAPCM can be greatly simplified utilizing the new method.

B. The "Proving" Step of Invariant Candidates

In the previous subsection, we described the assumption part of the inductive method to derive the invariants. In this section, we discuss and compare two methods for proving the invariant candidates under the imposed assumptions. As described previously, the invariant candidates are represented as candidate groups. The main issues to address are: 1) how to formulate checking problems from the candidate groups and 2) in what order to solve these problems.

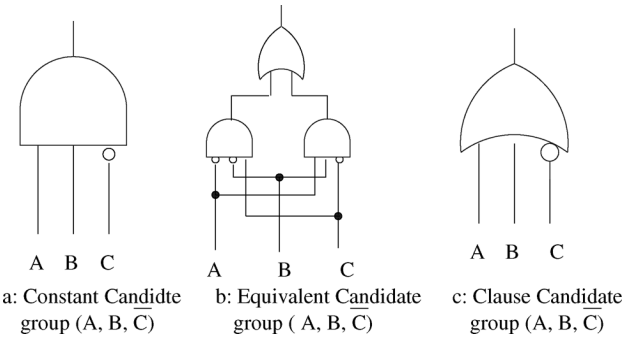


Fig. 12. Illustration of problem generation for different candidate groups.

In the first method, for each candidate group, a Boolean SAT problem [11] is formulated by adding some extra gates to a TTAPCM. For a constant candidate group, we connect all literals in the group to a logic AND gate g, and check whether the output of g can be set to 0. For an equivalence candidate group, we connect all its literals to a logic AND gate $g1$, connect all its inverted literals to logic AND gate $g2$, and, finally, connect the outputs of $g1$ and $g2$ to an OR gate $g3$. The problem is to check whether the output of $g3$ can be set to 0. For a clause candidate group, we simply connect its literals to an OR gate g. The problem is to check whether the output of g can be set to 0.

Fig. 12 illustrates the corresponding Boolean SAT problems for the three types of candidate groups.

In description of formulating Boolean SAT problems, to be concise, we have referred to the literals in the candidate groups without explicitly mentioning the timeframes. These literals actually correspond to the signals in the second timeframe of the TTAPCM.

Formulating the SAT problems in these ways has significant advantages. If the SAT problem generated for proving invariant candidates in a candidate group is proved unsatisfiable, all the invariant candidates represented by the group would hold. This applies to both constant candidate and equivalence candidate groups. This formulation is particularly powerful for equivalence candidate groups since an equivalence candidate group with x elements can represent $x * (x-1)/2$ equivalence invariant candidate pairs.

If the SAT problem generated for a clause candidate group is satisfiable, the candidate group can be removed. If the SAT problem of a constant or equivalence candidate group is satisfiable, based on the value assignments in its solution, the candidate group can be partitioned into two smaller candidate groups for which new problems can be formulated. For example, given an equivalence candidate group $\{A, \overline{B}, C\}$, if its corresponding SAT problem is satisfiable, and one solution is $(A = 1, B = 1, C = 0)$, then the candidate group $\{A, \overline{B}, C\}$ is partitioned into two equivalence candidate groups: $\{A\}$ and $\{\overline{B}, C\}$. Since $\{A\}$ contains only one literal, it is removed from further consideration.

Note that, a constant candidate group can be partitioned into a smaller constant candidate group and an equivalence candidate

group of the same or a smaller size. For example, in the solution of the SAT problem for a constant candidate group, if all literals in the group are assigned to 0, then none of the signals can be constant invariant. They could still be equivalent.

The algorithm based on this method is summarized in Algorithm V.3. In this algorithm, TTC denotes the two timeframe circuit model, S denotes the set of invariant candidates, and GLIST is the list of candidate groups. If the total number of literals in the initial constant candidate group and in the equivalence candidate groups is N, and the number of clause candidate groups is M, the number of generated SAT problems will not be greater than $N+M$. Note that according to the result of solving a SAT problem of an equivalence candidate group, either one literal can be merged to reduce one literal from the equivalence candidate group (for an unsatisfiable case) or else the equivalence candidate group is partitioned into two smaller groups (for a satisfiable case). Equivalence candidate groups of only one remaining literal can then be removed from further consideration. So for an equivalence candidate group with n literals, at most $n-1$ SAT problems are needed. The constant candidate group can be addressed similarly if we treat the constant candidate group as an equivalent candidate group by adding a literal with constant 1 into the group. The order of the problems to be solved is determined by the order of their corresponding candidate groups in GLIST. The initial candidate groups provided to IChecker are placed in the beginning of GLIST and the new candidate groups obtained by partitioning of the initial groups are appended at the end of GLIST when the groups are generated.

This method may still encounter some limitations. For very large circuits, a constant or equivalence candidate group could have hundreds (even thousands) of literals, for which the generated SAT problems could be difficult to solve. In addition, since the SAT problems for such candidate groups usually involve many signals, it is difficult to determine an optimal order in which to solve these SAT problems so that the information learned in solving earlier problems in the problem-solving processes could help make subsequent problems easier to solve.

Algorithm V.3: PROVEINVARIANTS_IMP1(TTC, S)

isfixed $\leftarrow$ true
while (GLIST is not empty)
do
$g \leftarrow$ the first of element of GLIST
Generate SAT problem from g and solve it.
if (the problem is satisfiable)
then
isfixed $\leftarrow$ false
if (g is not a clause candidate group)
then
Partition g into $g1$ and $g2$
Add $g1$ and $g2$ to GLIST
return (isfixed)

Based on the previous observations, we propose an improved method to generate SAT problems in a finer granularity. In this improved method, each time a SAT problem is generated for an invariant candidate, instead of an invariant candidate group. The order for the SAT problems to be solved is based on the topological order (from inputs to outputs) of the literals in the corresponding invariant candidates. Invariant candidates can be proved in any order. However, following the topological order could improve the reuse of learned clauses as described in [13].

The improved algorithm is shown in Algorithm V.4. We use LA to denote the array of literals of the sequential circuit, and it is sorted by the signal level as in Algorithm V.1. We also assume the literals in the candidate groups are sorted in ascending order by their indexes in LA. LC(l) is the list of clause candidate groups in which literal l is the member with the largest index in LA. TTC is the two timeframe circuit model, and S is the set of invariant candidates.

Algorithm V.4: PROVEINVARIANTS_IMP2(TTC, S)

$isfixed \leftarrow true$
$num \leftarrow$ the number of elements of LA
for $i \leftarrow 1$ **to** num
do
$v \leftarrow$ LA[i]
if ($v \in$ a constant candidate group)
then
Solving SAT problem ($v = 0$) with all the constraints
if (the problem is satisfiable)
then
isfixed $\leftarrow$ false
Perform solution-based simulation on TTC
Refine candidate groups by simulation results
else Simplify TTC based on the constant signal
else if ($v \in$ an equivalence candidate group EG)
then
$u \leftarrow$ the first member of EG
Solving SAT problem ($v \neq u$) with all the constraints
if (the problem is satisfiable)
then
isfixed $\leftarrow$ false
Perform solution based simulation on TTC
Refine candidate groups by simulation results
else
Simplify TTC based on the equivalent signals
for each Clause candidate group cg of LC(v)
do
Generate the clause c of cg
Solving SAT problem ($c = 0$) with all the constraints
if (the problem is satisfiable)
then
isfixed $\leftarrow$ false
Perform solution based simulation on TTC
Refine candidate groups by simulation results
return (isfixed)

In this algorithm, when a SAT problem for an invariant candidate is found satisfiable, we perform solution-based random simulation [16] on TTC. That is, for inputs (including primary inputs and pseudo primary inputs) assigned a binary value in a solution, the assigned values are used, and random values are assigned to any unassigned input in order to form a vector for simulation.

Based on the simulation results, invariant candidate groups are refined. If the corresponding clause is not satisfied by the

simulation results, a clause candidate group will be removed. A constant candidate group or an equivalence candidate group will be partitioned into subgroups based on the simulation results. In this way, the number of generated SAT problems will be no more than $N+M$, where N is the total number of literals in the initial constant candidate group and the equivalence candidate groups, and M is the number of initial clause candidate groups before the algorithm is applied. Note that without the simulation-based refinement described above, for an equivalence candidate group with x members, the number of generated SAT problems for verifying all equivalence invariant candidate pairs in the group could reach $x*(x-1)/2$ in the worst case. Also in this algorithm, TTC is simplified whenever signals are proved to be constant, equivalent, or inverted equivalent.

Note that this algorithm is based on the explicit learning heuristic used in C-SAT [13], and is similar to the SAT-sweeping algorithm in [16]. The difference is that in SAT-sweeping, the checking of equivalent candidate pairs may not follow the topological order as in our algorithm. Given two equivalent invariant candidates $(s_1 = t_1)$ and $(s_2 = t_2)$, assume $\mathrm{IND}(s_1) > \mathrm{IND}(t_1)$ and $\mathrm{IND}(s_2) > \mathrm{IND}(t_2)$, where $\mathrm{IND}(v)$ denotes the index of literal v in LA. It can be shown from our algorithm that $(s_1 = t_1)$ will be checked prior to $(s_2 = t_2)$ if

$$(\mathrm{IND}(s_1) < \mathrm{IND}(s_2)) \vee ((\mathrm{IND}(s_1) = \mathrm{IND}(s_2)) \wedge (\mathrm{IND}(t_1) < \mathrm{IND}(t_2))).$$

Please note again that, in this algorithm, when the merge operation between two signals incurs, the signal of the larger index in LA is merged into the signal of the smaller index in LA. Also, LA will not be affected by the merge operations.

We like to point out that while the circuit simplifications might change the signals' topological orders, but we do not perform any reordering. The checking of candidates follows the original topological order.

The experimental results of comparing the two methods described above are given in Table IV. In Table IV, we also give the results of checking invariants based on SAT-Sweeping algorithm. The benchmarks and the initial invariant candidates used in these experiments are the same as those listed in Table III. Column "#const" ("#eq") gives the number of signals in the final constant (equivalence) invariants. Column "#ite" shows the number of iterations needed to reach the fix-point. The CPU times (in seconds) of the two methods descried above are shown in Column "imp1" and Column "imp2". Column "SAT-S" shows the CPU time (in seconds) of the SAT-sweeping based method. Column "status" shows whether the output of the miter circuit is a constant invariant. In the last column "#p", we give the average number of SAT problems generated in each iteration in our second method "imp2".

The results demonstrate that IChecker ("imp2") can efficiently compute invariants for larger circuits. In the SAT-sweeping based method, each sub-problem generated is also based on an invariant candidate as in "imp2", but may not be solved following the topological order.

C. Computing kth Invariants and Comparison With k-Induction

In this subsection, we combine *BMChecker* and *IChecker* to derive kth invariants. We also give an example that can be solved more effectively by applying kth invariants than by using k-induction. Kth invariant candidates are derived from equivalent or constant signals computed by *BMChecker* at timeframe k of a sequential circuit. These candidates are further refined by random simulation, and *IChecker* takes the refined candidates as its initial invariant candidates. Note that, in the random simulation, only the simulation results after k cycles are used to refine the candidates. In all the following experiments, the improved proving algorithm, Algorithm V.4, is adopted in *IChecker*. As a kth invariant is also a $(k+1)$th invariant, it is expected that more invariants can be derived by increasing the value of k. The experimental results of computing kth invariants, where k is 0, 1 or 10, are shown in Table V.

TABLE IV
EXPERIMENTAL RESULTS TO COMPARE THE TWO METHODS

Circuit	#const	#eq	#ite	imp1 (s)	imp2 (s)	SAT-S (s)	status	#p
case1	74	434	19	56	2	4	NO	14
case2	255	1182	144	674	27	32	NO	43
case3	119	726	147	50	6	10	NO	33
case4	134	566	1	1835	7	140	YES	14
case5	103	248	19	8	1	1	NO	63
case6	9655	38826	2	1203	4	55	YES	108
case7	9622	39148	2	1323	4	67	YES	124
s5378	1977	17	48	2	2	2	YES	116
s13207	331	2519	48	90	4	4	YES	78
s35932	1279	20760	22	2245	28	87	YES	84
s38417	211	19337	30	2740	27	32	YES	175
s38584	572	13902	19	1077	20	34	YES	337

TABLE V
EXPERIMENTAL RESULTS OF COMPUTING k-TH INVARIANTS

Circuit	0-th invariant			1-st invariant			10-th invariant			n	total
	#eq	#const	time	#eq	#const	time	#eq	#const	time		
case1	434	74	2	434	74	2	434	74	2	NA	21
case2	1182	145	17	1052	255	27	1651	353	24	NA	262
case3	368	50	5	575	90	7	766	137	7	NA	72
case4	286	35	8	422	84	3	574	135	2	0	38
case5	205	64	2	215	100	2	248	103	2	NA	20
case6	36429	8895	20	38667	9622	22	38826	9655	42	2	345
case7	36749	8865	24	38989	9589	27	39148	9622	53	2	459
s5378	1977	470	2	1961	487	2	1924	533	2	0	19
s13207	2587	249	5	2519	331	5	2512	369	5	0	49
s35932	20760	1279	30	20760	1279	37	20760	1279	57	0	493
s38417	19325	211	28	19337	211	33	19337	211	34	0	361
s38584	13902	572	19	13902	572	21	13902	572	42	0	354

Columns "#eq" give the number of equivalent signals detected as 0th invariants, 1-st invariants, and 10th invariants respectively. Columns "#const" give the number of detected constant signals. Columns "time" show the CPU time (in seconds) spent in both *BMChecker* and *IChecker*. Column "n" gives the smallest value of k if the miter output $o = 0$ is proved to be a kth invariant by *IChecker* where k could be $0, 1, \ldots, 10$. Otherwise, Column "n" shows "NA". Column "total" gives the total CPU time (in seconds) needed to compute from 0th invariants to 10th invariants. These results show that for most cases, the count of identified equivalent and constant invariants (i.e., $\#eq + \#const$) increase as k increases. For cases case6 and case7, the miter output is identified by *IChecker* as a second invariant but not as a 0th invariant, nor a first invariant, even though *BMChecker* could prove that the miter output o is constant 0 in timeframes 0 and 1 for these two cases.

Note that in our experiments, the invariants candidates not only contain the constancy of the miter output, but also the individual original output equivalences and many other invariants

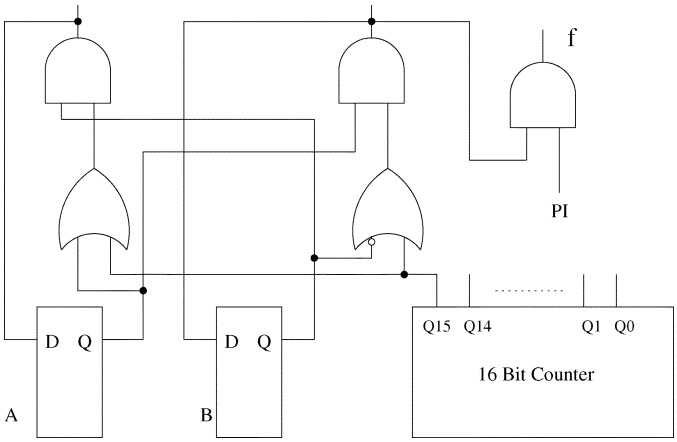


Fig. 13. Sequential circuit.

among internal signals. Thus, the induction in our experiments can be much stronger than that considering only the invariants among the outputs.

The k induction and the unique state induction [21], [22] are techniques proposed to strengthen 0th invariant computation. These can be extended to compute kth invariants as well. Another extension is to consider the n-timeframe expansion of the original circuit as a new circuit for which the invariants are computed. This extension requires $2 * n$ timeframes of the original circuit, but can compute invariants related to those signals in different timeframes. Detailed discussion of these extensions is beyond the scope of this paper. Instead, we provide an example to demonstrate the value of kth invariants. Considering the sequential circuit given in Fig. 13, the initial states of A, B and the 16-bit counter are 1, 0, and 0. f is the output signal. If we take Q_{15} as a variable, the next state function of A is $N(A) = (Q_{15}$ OR $A)$ AND B , and the next state function of B is $N(B) = (\overline{B}$ OR $Q_{15})$ AND A. It can be seen that the value of A is 1 in timeframe 0, and is 0 in any other timeframe; the value of B is 1 in timeframe 1 and is 0 in any other timeframe. Using the kth invariant concept, if we take $(A = 0)$, $(B = 0)$ and $(f = 0)$ as second invariant candidates, these can be proven to be true second invariants using the TTAPCM model. We can then conclude that $(f = 0)$ is a 0th invariant by checking that $f = 0$ holds in the first two timeframes. However, if we directly apply k induction and unique state induction to prove whether $(f = 0)$ is a 0th invariant, the state space of the 16-bit counter needs to be explored, which might require the value of k to be exponential with respect to the number of bits in the counter.

In Table VI, we show some experimental results comparing IChecker with k-induction. To conduct this experiment, we implemented a k-induction tool on our framework, and the 0th induction invariants are utilized to strengthen k-induction. In this experiment, we tried six testcases, including four property checking cases, "b04.prop", "b09.prop", "b12.prop", and "b13.prop" derived from the ITC'99 [40] benchmarks. The sub-column "k" under "IChecker" shows the minimum value of k that the output of the circuit is proven to be a kth invariant. The sub-column "time" under "IChecker" lists the total CPU runtime (in seconds) needed to prove, by *IChecker* and *BMChecker* jointly, that the output of the circuit is an invariant. Similarly, the sub-columns "k" and "time" under "K-induction" show the minimum k and the CPU runtime (in seconds) for the k-induction method to prove that the circuit's output is an invariant. For this experiment, the limit of k is set to 100. For testcases "case6", "case7", and "b13.prop", IChecker significantly outperforms k-induction, while for testcase "b09.prop", k-induction performs much better than IChecker. These results indicate that k-induction and IChecker have their own unique advantages.

TABLE VI
COMPARISON OF ICHECKER AND K-INDUCTION

Circuit	IChecker		K-induction	
	k	time	k	time
case6	2	69	> 100	2086
case7	2	81	> 100	4797
b04.prop	1	0.1	4	0.1
b09.prop	> 100	23	3	0.1
b12.prop	> 100	282	> 100	992
b13.prop	3	2	> 100	29

VI. SEQUENTIAL SAT SOLVER

The sequential SAT solver *Seq-SAT* implements a backward search strategy that tries to find a path from a given objective to the initial state. The process of solving a sequential SAT problem of a sequential circuit can be seen as solving a sequence of combinational SAT problems in one timeframe of the sequential circuit. The objectives of these combinational SAT problems are called frame objectives. The overall algorithm of the sequential SAT solver *Seq-SAT* is described in Algorithm VI.1. The details of the solver can be found in [14]. In this algorithm, frame objectives are stored in the *objective list*, which contains only the initial objective at the beginning. Function select_a_frame_objective() selects a frame objective from the *objective list*. The selected frame objective is solved in function combinational_solve_a_frame_objective(). This returns a solution if the selected frame objective is satisfiable, and returns NULL if it is unsatisfiable. Function state_reduction() tries to derive a minimized solution by removing unnecessary assignments to the state variables (PPIs) from the solution returned by the combinational_solve_a_frame_objective(). When a frame objective is proved unsatisfiable, PPO_state_conflict_analysis() tries to determine which part of the objective makes the problem unsatisfiable. Both state_reduction() and PPO_state_conflict_analysis() help to find a smaller state clause that can be added to the circuit to prune a larger search space. A frame objective is removed from the *objective list* if it is proved unsatisfiable.

Algorithm VI.1: SEQUENTIALSOLVER(C, obj, s_0)

comment: C is the circuit with PPIs and PPOs expanded
comment: obj is the initial objective
comment: s_0 is the initial state
comment: FO is the objective list
FO $\leftarrow \{$obj$\}$;
while (FO $\neq \emptyset$)
do
$f_{\text{obj}} \leftarrow$ select_a_frame_objective(FO);
$f_{\text{sol}} \leftarrow$ combinational_solve_a_frame_objective(C, f_{obj});
if ($f_{\text{sol}} =$ NULL)
clause $\leftarrow$ PPO_state_conflict_analysis(C, f_{obj});
add_state_clause(C, clause);

FO $\leftarrow$ FO $-$ $\{f_{\rm obj}\}$;
else
state_cube $\leftarrow$ state_reduction(C, $f_{\rm obj}$, $f_{\rm sol}$);
if ($s_0 \in$ state_cube)
return (SAT);
else
clause $\leftarrow$ convert_to_clause(state_cube);
add_state_clause(C, clause);
FO $\leftarrow$ FO $+$ {state_cube};
return (UNSAT)

Note that there are two stopping criteria in this algorithm: 1) when a solution state covers the initial state, the algorithm stops and reports a solution and 2) when the *objective list* becomes empty during the sequential search, which means all the back-reachable state space has been exhausted without reaching the initial state, the algorithm stops and reports that the sequential SAT problem is unsatisfiable.

The second stopping criterion can still be used when the constraints of kth invariants are added to the sequential circuit. However, with these constraints, the initial state may not be back-reachable even if the sequential SAT problem is a satisfiable problem. For example, if a flip-flop in a sequential circuit has an initial value of 0, and becomes constant 1 starting from timeframe 1, the corresponding pseudo primary input signal ppi is a first constant invariant signal. When a unique literal clause (ppi) is added to constrain ppi to 1, the initial state could no longer be reached using this algorithm.

This issue can be solved by changing the first stopping criteria to the following: If the solution state *state_cube* can be reached from the initial state at timeframe k, the algorithm stops and reports the solution.

In SEChecker, *Seq-SAT* has been modified to employ *BMChecker* to check if a solution state can be reached from the initial state at timeframe k once the solution state is derived. The problem in adopting this modified stopping criteria is that *Seq-SAT* may not be able to find a solution whose length is less than k. However, this problem is not a serious one because it is quite easy for *BMChecker* to determine whether such solutions exist if k is not too large.

VII. Experimental Results

Table VII shows the experimental results of our sequential equivalence checker. In these experiments, the limit k for *BMChecker* is set to 10. The benchmarks are the same as those in Table I, and none of them can be solved by *Seq-SAT* alone within 10 hours of CPU runtime even after simplification and flip-flop mapping are applied.

Column "#ff" ("#g") is the number of flip-flops (gates) of the circuit before simplification, and Column "runtime(sec)" shows the CPU time (in seconds). Column "status" shows the stage in which the problem is solved, Column "#state" shows the number of state solutions derived by *Seq-SAT*, if available, Column "#len" shows the length of computed counter-example, if available, and Column "result" gives the verification result. "EQ" means the two circuits are equivalent and "NEQ" indicates they are not equivalent.

All these examples could be solved by our sequential equivalence checker in a reasonable amount of time, and demonstrate the effectiveness of our SEC framework.

TABLE VII
Experimental Results of the SEC Framework

Circuit	#ff	#g	runtime(sec)	status	#state	#len	result
case1	159	5818	7	BMChecker	-	10	NEQ
case2	273	7367	32	Seq-SAT	1385	1026	NEQ
case3	183	2276	134	Seq-SAT	46649	1026	NEQ
case4	116	2539	3	IChecker	-	-	EQ
case5	396	2672	697	Seq-SAT	121710	-	EQ
case6	457	61589	132	IChecker	-	-	EQ
case7	461	62333	141	IChecker	-	-	EQ
s5378	545	4146	4	IChecker	-	-	EQ
s13207	1715	9842	10	IChecker	-	-	EQ
s35932	4330	32306	66	IChecker	-	-	EQ
s38417	4070	27538	43	IChecker	-	-	EQ
s38584	4935	36486	54	IChecker	-	-	EQ

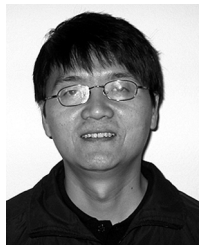
VIII. Conclusion

In this paper, we propose an SEC framework, named SEChecker. The framework consists of three complementary components: a bounded model checker, *BMChecker*, an inductive invariant checker, *IChecker*, and a sequential SAT solver, *Seq-SAT*. *BMChecker* employs the explicit learning technique to improve the performance for BMC and provides kth invariant candidates. *IChecker* performs fix-point calculation based on the TTAPCM model to find true kth invariants from the kth invariant candidates. The kth invariants derived by *IChecker* could be taken as constraints to prune the search space for *Seq-SAT*. The framework proves to be powerful for addressing sequential equivalence checking problems for larger and more complex designs.

References

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, Aug. 1986.
[2] O. Coudert and J. C. Madre, "A unified framework for the formal verification of sequential circuits," in *Proc. Int. Conf. Comput.-Aided Des.*, 1990, pp. 126–129.
[3] C. Pixley, "A theory and implementation of sequential hardware equivalence," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 11, no. 12, pp. 1469–1478, Dec. 1992.
[4] P. Ashar, A. Gupta, and S. Malik, "Using complete-1-distinguishability for FSM equivalence checking," in *Proc. Int. Conf. Comput.-Aided Des.*, Dec. 1996, pp. 346–353.
[5] J.-H. Jiang and R. K. Brayton, "On the verification of sequential equivalence," *IEEE Trans. Comput.-Aided Des.*, vol. 22, no. 6, pp. 686–697, Jun. 2003.
[6] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen, "AQUILA: An equivalence checking system for large sequential designs," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 443–463, May 2000.
[7] C. A. J. van Eijk, "Sequential equivalence checking based on structural similarities," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 7, pp. 814–819, Jul. 2000.
[8] W.-T. Cheng, "The BACK algorithm for sequential test generation," in *Proc. Int. Conf. Comput. Des.*, Oct. 1988, pp. 66–69.
[9] A. Ghosh, S. Devadas, and A. R. Newton, "Test generation and verification for highly sequential circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 10, no. 5, pp. 652–667, May 1991.
[10] D. Brand, "Verify large synthesized designs," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 1993, pp. 534–537.
[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2001, pp. 530–535.
[12] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust Sat_Solver," in *Proc. Eur. Des. Test Conf.*, Mar. 2002, pp. 142–149.
[13] F. Lu, L.-C. Wang, and K.-T. Cheng, "A circuit SAT solver with signal correlation guided learning," in *Proc. Eur. Des. Test Conf.*, 2003, pp. 892–897.
[14] F. Lu, M. K. Iyer, G. Parthasarathy, L. C. Wang, K. T. Cheng, and K. C. Chen, "An efficient sequential SAT solver with improved search strategies," in *Proc. European Design and Test Conference*, 2005.

[15] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. 36th ACM/IEEE Des. Autom. Conf.*, Jun. 1999, pp. 317–320.
[16] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2004, pp. 50–57.
[17] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Proc. Int. Conf. Tools Algorithms for Construction Analysis Syst. (TACAS)*, Mar. 1999, pp. 193–207.
[18] J. Baumgarter, A. Kuehlmann, and J. Abraham, "Property checking via structure model analysis," in *Proc. Comput.-Aided Verification (CAV)*, Jul. 2002, pp. 151–165.
[19] D. Kroening and O. Strichman, "Efficient computation of recurrence diameters," in *Proc. Int. Conf. Verification, Model Checking, Abstract Interpretation*, Jan. 2003, pp. 298–309.
[20] K. L. McMillan, "Interpolation and SAT based model checking," in *Proc. Int. Conf. Comput.-Aided Verification*, 2003, vol. 2725, LNSC, pp. 1–13.
[21] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a SAT-solver," in *Proc. Int. Conf. Formal Methods Comput.-Aided Des.*, 2000, pp. 108–125.
[22] L. D. Moura, H. Rueβ, and M. Sorea, "Bounded model checking and induction: From refutation to verification," in *Proc. Int. Conf. Comput.-Aided Verification*, 2003, vol. 2725, LNSC, pp. 14–26.
[23] K. L. McMillan, "Applying SAT methods in unbounded symbolic model checking," in *Proc. Int. Conf. Comput.-Aided Verification*, 2002, vol. 2404, LNSC, pp. 250–264.
[24] H.-J. Kang and I.-C. Park, "SAT-based unbounded symbolic model checking," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2003, pp. 840–843.
[25] A. Kuehlmann, M. Ganai, and V. Paruthi, "Circuit-based Boolean reasoning," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2001, pp. 232–237.
[26] P. Tafertshofer, A. Ganz, and M. Henftling, "A SAT-based implication engine for efficient ATPG, equivalence checking, and optimization of netlists," in *Proc. Int. Conf. Comput.-Aided Des.*, 1997, pp. 648–657.
[27] L. Silva, L. Silveira, and J. M. Silva, "Algorithms for solving Boolean satisfiability in combinational circuits," in *Proc. Des., Autom. Test Eur.*, 1999, pp. 526–530.
[28] L. Silva and J. M. Silva, "Solving satisfiability in combinational circuits," in *Proc. IEEE Des. Test Comput.*, Jul./Aug. 2003, pp. 16–21.
[29] A. Gupta, Z. Yang, and P. Ashar, "Dynamic detection and removal of inactive clauses in SAT with application in image computation," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2001, pp. 536–541.
[30] M. K. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik, "Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2002, pp. 747–750.
[31] F. Lu and K.-T. Cheng, "Sequential equivalence checking based on K-th invariants and circuit SAT solving," in *Proc. IEEE HLDVT Workshop*, Nov. 2005, pp. 45–52.
[32] P. Bjesse and K. Claessen, "SAT-based verification without state space traversal," in *Proc. FMCAD*, 2000, vol. 1954, LNCS, pp. 372–389.
[33] Berkeley Logic Synthesis and Verification Group, Berkeley, CA, "ABC: A system for sequential synthesis and verification," 2006. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/
[34] H. Mony, J. Baumgartner, and A. Aziz, "Exploiting constraints in transformation-based verification," *Proc. CHARME*, pp. 269–284, 2005.
[35] H. Zhang, "SATO: An efficient propositional prover," in *Proc. Int. Conf. Autom. Deduction*, 1997, vol. 1249, LNAI, pp. 272–275.
[36] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 506–521, May 1999.
[37] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT*. : Springer, 2003, vol. 2919, LNCS, pp. 502–518.
[38] N. Amla, X. Q. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan, "An analysis of SAT-based model checking techniques in an industrial environment," in *CHARME*. New York: Springer, 2005, vol. 3725, LNCS, pp. 254–268.
[39] M. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring," in *Proc. Int. Conf. Comput.-Aided*, 2004, pp. 129–140.
[40] CAD Group at Politecnico di Torino, Torino, Italy, "ITC'99 Benchmarks," 1999. [Online]. Available: http://www.cerc.utexas.edu/itc99-benchmarks/bench.html
[41] Z. Xiu, D. A. Papa, P. Chong, C. Albrecht, A. Kuehlmann, R. A. Rutenbar, and I. L. Markov, "Early research experience with OpenAccess Gear: An open source development environment for physical design," in *Proc. ACM Int. Symp. Phys. Des.*, 2005, pp. 94–100.



Feng Lu received the B.S. degree in computer science from the Civil Aviation Institute of China, Tianjin, China, in 1993, the M.S. degree in computer science from Tsinghua University, Beijing, China, in 1996, and the Ph.D. degree in electrical and computer engineering from the University of California, Santa Barbara, in 2006.

He has worked with Cadence Design Systems since 2006. His research interests include equivalence checking and formal verification.

Kwang-Ting Cheng (F'00) received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1983 and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1988.

He worked with Bell Laboratories, Murray Hill, NJ, from 1988 to 1993 and joined the faculty at the University of California, Santa Barbara, in 1993, where he is currently a Professor and Chair of the Electrical and Computer Engineering Department. His current research interests include VLSI testing, design verification, and multimedia computing. He has published more than 250 technical papers, coauthored three books, and holds 10 U.S. Patents in these areas.

Dr. Cheng was a recipient of Best Paper Awards at the 1994 and 1999 Design Automation Conferences, a 2001 Annual Best Paper Award in the *Journal of Information Science and Engineering*, a Best Paper Award in the 2003 Conference of Design Automation and Test in Europe (DATE 2003), and the Best Paper Award at the 1987 AT&T Conference on Electronic Testing. He currently serves as Editor-in-Chief for IEEE DESIGN AND TEST OF COMPUTERS, Associate Editor for *ACM Transactions on Design Automation of Electronic Systems*, Editor for *Journal of Electronic Testing: Theory and Applications*, and Editor for *Foundations and Trends in Electronic Design Automation*. He had been General Chair and Program Chair of the IEEE International Test Synthesis Workshop, Program Cochair of International Mixed-Signal Test Workshop, and has also served on the technical program committees for a number of international conferences on design, design automation, and test.