# Sequential Equivalence Checking Based on $K$-th Invariants and Circuit SAT Solving

Feng Lu and K.-T. Cheng
Department of ECE, University of California at Santa Barbara

## Abstract

*In this paper, we first present the concept of the k-th invariant. In contrast to the traditional invariants that hold for all cycles, k-th invariants guarantee to hold only after the k-th cycle from the initial state. We then present a bounded model checker BMChecker and an invariant prover IProver, both of which are based on circuit SAT techniques. Jointly, BMChecker and IProver are used to compute the k-th invariants, and are further integrated with a sequential SAT solver for checking sequential equivalence. Experimental results demonstrate that the new sequential equivalence checking framework can efficiently verify large industrial designs.*

## I. Introduction

With increasing use of sequential optimization techniques for performance, area, power dissipation, etc., sequential equivalence checking (SEC) is becoming increasingly important. 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 various symbolic techniques [2, 3, 4, 5]. In these approaches, a product machine is constructed over the two finite state machines (FSMs) under verification [2, 3, 4]. 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. A multiplexed machine, which combines the two FSMs under comparison, was introduced in [5]. The equivalence problem of the two FSMs is then transformed into a state equivalence problem of the multiplexed machine. These algorithms are suitable for small- to medium-sized circuits, but may not scale well for larger designs.

A computational model, called *miter* [11], was built from the two sequential circuits under verification in [6]. In this model, starting from the given initial state, if there exists any input sequence 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. Structural similarity is explored and used in [6, 7] for further reduction of the SEC complexity. In both approaches, induction-based proof and fix-point calculation are applied to detect equivalent signals. In addition, sequential Automatic Test Pattern Generation (ATPG) techniques [9, 10] are used in [6] to improve the efficiency for identifying equivalent flip-flop and internal signal pairs.

The method proposed in [7] introduces a two-timeframe, assume-and-prove circuit model for the product machine for efficient calculation of the maximum correspondence relation. In the first timeframe, potential equivalent signals, which are heuristically identified, are modeled. The correctness of these equivalences is then verified in the second timeframe. This procedure iterates until a fix-point is reached. The equivalences between signals remained at the end of this process form the maximum correspondence relation. Under the maximum correspondence relation, if the output of the product machine is constant 1, then the two circuits are equivalent. However, if the output of the product machine is not constant 1, it is not conclusive whether the circuits are equivalent or not. 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 [12], Berkmin [13] and C-SAT [14]. State-of-the-art SAT algorithms implemented in these tools have demonstrated their capability and efficiency for handling large industrial SAT problems.

In [16, 17], SAT solvers are applied in bounded model checking (BMC) for verifying safety properties. BMC systematically searches for counterexamples whose length is bounded by some integer k. The bound $k$ starts at 0 and is increased until a counterexample is found or a given threshold 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 solutions. Therefore, BMC is efficient for problems with short counterexamples. However, BMC is not complete since a property proven true for depths 0 through $k$ may not be true at depths greater than $k$. To achieve completeness, the diameter-based method was introduced

0-7803-9571-9/05/$20.00 ©2005 IEEE.

in [18, 19, 20] to find the true upper-bound $k$ for the property. The interpolation-based method [21] approximates the set of reachable state to extend BMC for unbounded model checking. The $k$-induction-based method [22, 23] attempts to prove the correctness of properties as part of BMC unfolding.

In addition, SAT-based unbounded model checking (UMC) that employs SAT algorithms for pre-image calculation is introduced in [24, 25].

The sequential SAT solver *Seq_SAT* [15] employs backward justification techniques to find an input sequence that can satisfy the objective from the given initial state or prove no such input sequence exists. *Seq_SAT* uses the combinational SAT solver C-SAT [14] as the underlying engine and utilizes circuit structure information for better decision ordering. It can be proved that the *Seq_SAT* algorithm is complete. Intuitively, *Seq_SAT* can be employed to solve the objective of miter output being 1 for checking sequential equivalence. However directly applying *Seq_SAT* for the SEC problem has very limited scalability.

In this paper, we propose an SEC framework that supports incremental verification for large, industrial SEC problems. The framework consists of three major components: a bounded model checker *BMChecker*, an invariant prover *IProver*, and the sequential SAT solver *Seq_SAT*. *BMChecker* employs the explicit learning technique, first introduced in C-SAT [14], to improve the performance for BMC. *IProver* performs fix-point calculation based on the two-timeframe, assume-and-prove circuit model to find $k$-th invariants. $k$-th invariants are a generalization of the traditional invariants. A $k$-th 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 $k-1$. Such $k$-th invariants could be used to simplify the unfolded model in *BMChecker*, and could be taken as constraints to prune the search space in *Seq_SAT*.

*BMChecker*, *IProver*, and *Seq_SAT* have different strengths that complement one another. *BMChecker* is most suitable for satisfiable problems with short solutions. *IProver* is ideal for unsatisfiable problems that can be proven by induction. However, neither of them is complete. *Seq_SAT* is complete and could be used for general problems.

The rest of the paper is organized as the follows. Section II gives the background. In Section III, we present the basic flow of our SEC framework and describe each part of the framework in detail. Section IV shows the experimental results of the SEC framework on some industrial benchmarks. Section V summarizes the contributions of this paper.

## II. Background

We assume that the sequential circuit follows the Huffman circuit model. One timeframe of a sequential circuit is viewed as a combinational circuit where each flip-flop is converted into two corresponding signals: a pseudo primary input (PPI) and a pseudo primary output (PPO). The timeframe where the initial state is applied at the PPIs is numbered as timeframe 0. Timeframe expansion is achieved by connecting the PPIs of timeframe $i+1$ to the corresponding PPOs of the previous timeframe $i$.

The Huffman circuit model, and the circuit models for one timeframe and two timeframes are illustrated in Figure 1.a , Figure 1.b, and Figure 1.c respectively.

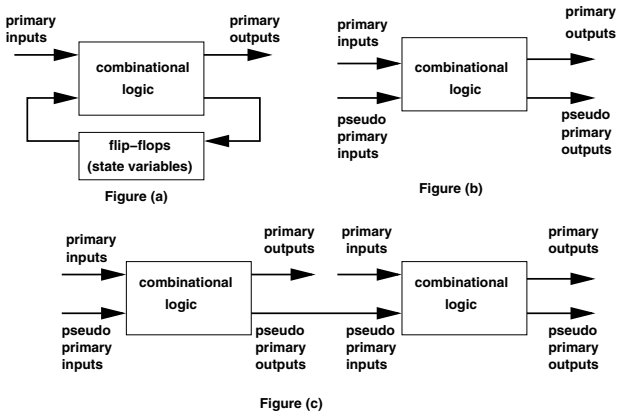


Figure 1: Illustrations of Huffman circuit model, circuit model for one timeframe, and model for two timeframes.

The miter circuit for two circuits under verification is built by joining their primary inputs, connecting corresponding output pairs by XOR gates, and then connecting the XOR outputs to an OR gate. Figure 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 constant 0.

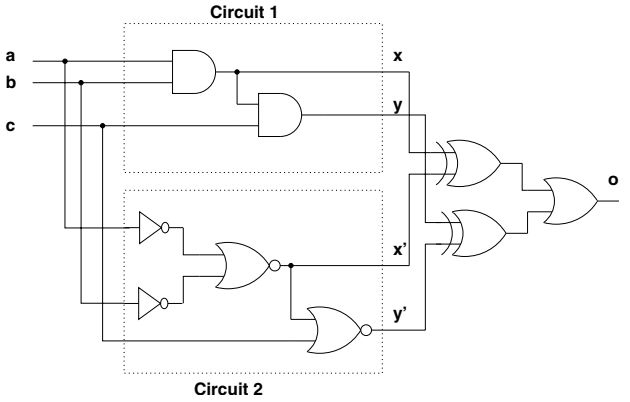


Figure 2: Illustration of a miter circuit.

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

Given a sequential circuit *SC* with one primary output $o$, We denote *PI* 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 $V = PI \bigcup INT \bigcup PPI \bigcup PPO \bigcup \{o\}$ denotes the set of all signals of *SC*. For each flip-flop $ff$, $INIT_{ff}$ denotes its initial value which could be 0, 1 or X. If $INIT_{ff}$ 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, denoted as $s_0$, of the sequential circuit.

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.

The logic function of signal $v$ in the one-timeframe combinational circuit of *SC* can be denoted as $g_v(pi_1, pi_2, ..., pi_{|PI|}, ppi_1, ppi_2, ..., ppi_{|PPI|})$ where $pi_1, pi_2, ..., pi_{|PI|} \in PI$, and $ppi_1, ppi_2, ..., ppi_{|PPI|} \in PPI$. For conciseness, we use $g_v$ instead of $g_v(pi_1, pi_2, ..., pii_{|PI|}, ppi_1, ppi_2, ..., ppi_{|PPI|})$.

We also use $f^i(v)$ to denote the logic function of signal $v^i$, $v^i \in V^i$ at timeframe $i$. $f^i(v)$ can be derived as follows:

1. $f^i(v) = 0$ if $i = 0, v \in PPI$, and $INIT_v = 0$;
2. $f^i(v) = 1$ if $i = 0, v \in PPI$, and $INIT_v = 1$;
3. $f^i(v) = v$ if $i = 0, v \in PPI$, and $INIT_v = X$;
4. $f^i(v) = g_v(pi_1^i, pi_2^i, \ldots, pi_{|PI|}^i, f^i(ppi_1), f^i(ppi_2), \ldots, f^i(ppi_{|PPI|}))$ if $v \notin PPI$ where $pi_1^i, pi_2^i, \ldots, pi_{|PI|}^i \in PI^i$, and $ppi_1, ppi_2, \ldots, ppi_{|PPI|} \in PPI$;
5. $f^i(v) = f^{i-1}(u)$ if $i > 0, v \in PPI, u \in PPO$, 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.

**(a).** A signal $v \in V$ is a strong constant signal iff $g_v = 0$ or $g_v = 1$.

**(b).** Two signals u and v, where $u, v \in V$, are strong equivalent iff $g_u = g_v$.

**(c).** A signal $v \in V$ is a weak constant signal iff $f^i(v) = 0$ for all $i \geq 0$, or $f^i(v) = 1$ for all $i \geq 0$.

**(d).** Two signals u and v, where $u, v \in V$, are weak equivalent iff $f^i(u) = f^i(v)$ for all $i \geq 0$.

**(e).** A k-th invariant $p$ involving $n$ signals $v_1$, $v_2$ ..., $v_n$ is a Boolean function $p(v_1, v_2, \ldots, v_n)$ that $p(f^i(v_1), f^i(v_2), \ldots, f^i(v_n)) = 1$ for all $i \geq k$.

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

Invariants can be used to simplify the sequential circuit and to prune the search space of the SAT problem. For our target output-1-SAT problem, we adopt an incremental solving strategy. Before solving the original problem, invariants are identified to reduce the problem complexity. Among invariants, easy-to-identify invariants are derived first, which helps to reduce the complexity of finding hard-to-identify invariants. For example, strong equivalent signals can be derived easily by checking combinational equivalence in the one-timeframe combinational model. The computation of $k$-th invariants would involve $k$-bounded model checking and fix-point calculation on the two-timeframe, assume-and-prove circuit model.

## III. Sequential Equivalence Checking Framework

Figure 3 shows the basic flow in our sequential equivalence checking framework. The miter circuit of the two circuits under verification is first constructed. It 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 proven 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 proven not equivalent. If no such solution exists, the equivalent or constant signals derived by *BMChecker* at timeframe $k$ are considered as initial candidates of $k$-th invariants. Among these candidates, *IProver* is employed to identify true $k$-th invariants. If "$o = 0$" is identified to be a $k$-th 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 $k$-th invariants derived by *IProver* are inserted as constraints to the miter circuit. *Seq_SAT* is then used to solve the sequential SAT problem "$o = 1$". With simplification as well as constraints included in the miter circuit, the search space is often dramatically reduced and, thus, the problem is easier than the original one.

In the following, we first discuss the simplification of sequential circuits based on strong equivalent and constant signals and flip-flop mapping. We then describe the three key components *BMChecker*, *IProver* and *Seq_SAT* respectively.

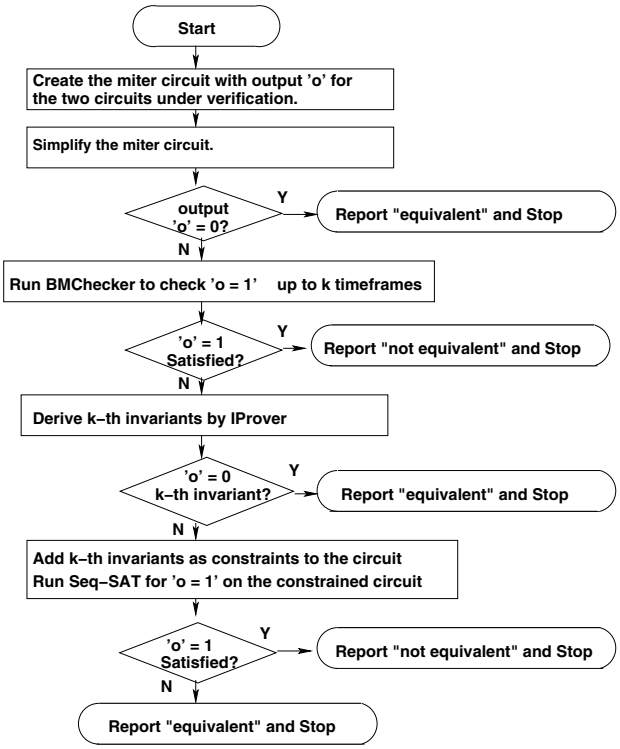


Figure 3: Our sequential equivalence checking framework.

### III-A. Circuit Simplification

Strong equivalent (constant) signals are the 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 [14] and the structure-based method proposed in [26]. In [26] and [14], 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 detection rules are summarized as follows:

1. For a two-input AND gate $g$ with inputs $i_1$, $i_2$ and output $o$, the following simplification rules can be applied:

   $o = i_1 = i_2$ if $i_1 = i_2$.
   $o = 0$ if $i_1 = \overline{i_2}$
   $o = 0$ if $i_1 = 0$ or $i_2 = 0$
   $o = i_2$ if $i_1 = 1$
   $o = i_1$ if $i_2 = 1$

2. For two two-input AND gates $g_1$ and $g_2$, where $i_1$, $i_2$ and $o_1$ are inputs and output of $g_1$, and $i_3$, $i_4$ and $o_2$ are inputs and output of $g_2$, the simplification rule is as follows:

   $o_1 = o_2$ if $i_1 = i_3 \wedge i_2 = i_4$ or $i_1 = i_4 \wedge i_2 = i_3$.

The explicit learning technique used in C-SAT can detect non-trivial 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 ($u = 1, v = 0$). The candidates of strong equivalent or constant signals can be obtained by simulation of random vectors, and refined by the counter examples derived in the explicit learning procedure [17]. 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 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 used for further simplification.

TABLE I: EXPERIMENTAL RESULTS ON CIRCUIT SIMPLIFICATION

| Circuit | Original Information | | | After Simplification | | runtime |
|---|---|---|---|---|---|---|
| | #ff | #X | #g | #ff | #g | (sec) |
| in1 | 294 | 164 | 7643 | 287 | 6785 | 15 |
| in2 | 273 | 173 | 7367 | 273 | 7297 | 3.9 |
| in3 | 108 | 88 | 2516 | 108 | 2516 | 0.5 |
| in4 | 116 | 96 | 2539 | 116 | 2539 | 0.5 |
| in5 | 102 | 51 | 4854 | 102 | 4418 | 1.4 |
| in6 | 457 | 382 | 61589 | 410 | 53528 | 100 |
| in7 | 461 | 386 | 62333 | 414 | 54237 | 103 |

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 miter circuits are created from industrial designs. In each miter model, one circuit is a gate-level netlist synthesized from its behavioral-level description written in System-C, and the other one is a gate-level netlist synthesized from its RTL description written in Verilog. All our experiments in this paper were run on P4 2-GHz Linux machines 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. The column "#X" is the number of flip-flops whose initial value is $X$, and the column "runtime" gives the CPU time in seconds of the simplification process. From the results, it can be seen that some circuits achieve non-trivial simplification while the runtime is relatively small.

### III-B. Bounded Model Checking

*BMChecker* is a bounded model checker developed based on C-SAT. Like 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 checking the objective again on the expanded circuit. The procedure iterates until either a solution is found or a given limit is reached.

The uniqueness of *BMChecker* is that 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 applying the SAT solver to the objective.

We compare *BMChecker* with a state-of-the-art BMC implementation which is based on a plain Conjunctive Normal Form (CNF) translation of the expanded circuit and Berkmin (BerkMin561 linux version) as the underlying SAT solver. The comparison is shown in Table II. The benchmarks used in these experiments are simplified miter circuits listed in Table I. In Table II, Column "#frame" is the number of timeframes expanded, Column "#tg" is the total number of gates in the expanded timeframes, Column "#lg" gives the number of gates left after equivalent or constant signals are merged by *BMChecker*, and Columns "runtime(sec)" show the CPU time (in seconds). The results demonstrate that *BMChecker* not only produces useful information such as equivalent or constant signals for candidates of invariants, but also exhibits superior performance for BMC.

TABLE II: EXPERIMENTAL RESULTS OF BMCHECKER

| Circuit | #frame | #tg | BMChecker | | CNF-based BMC |
|---|---|---|---|---|---|
| | | | #lg | runtime(sec) | runtime(sec) |
| in1 | 50 | 339103 | 126398 | 677 | >36000 |
| in2 | 50 | 364703 | 150577 | 219 | 11171 |
| in3 | 50 | 125653 | 36251 | 33 | >36000 |
| in4 | 50 | 126803 | 37107 | 34 | >36000 |
| in5 | 50 | 220753 | 191078 | 447 | 10799 |
| in6 | 20 | 1070483 | 158787 | 117 | 1642 |
| in7 | 20 | 1084683 | 167189 | 127 | 1746 |

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

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

### III-C. Invariant Prover

Invariant prover *IProver* is also developed based on C-SAT. It works on a two-timeframe, assume-and-prove computational model [7, 8]. In the first timeframe of this model, we add constraints converted from the invariant candidates. These constraints enforce that the invariant candidates hold at the first timeframe. These candidates are then verified in the second timeframe. False candidates are removed from the candidate lists, and fix-point calculations are carried out to compute the real invariants.

The equivalent or constant signals computed at timeframe $k$ by *BMChecker* are the initial invariant candidates taken by *IProver*. After fix-point calculation, the remaining candidates which hold at timeframe $k$ will hold at timeframe $k+1$, and thus at any future timeframes as well. Therefore, these remaining candidates which hold at timeframe $k$ are proven to be $k$-th invariants.

In *IProver*, the constraints are represented by clauses. For example, the equivalence constraint between two signals $u$ and $v$ can be represented by two clauses $(u+\overline{v})$ and $(\overline{u}+v)$. The constraint to force a signal $v$ to be constant 1 is a unique literal clause $(v)$.

For a signal $s$ in a sequential circuit, we use $s_1$ ($s_2$) to denote the signal corresponding to $s$ in the first (second) timeframe of the two-timeframe, assume-and-prove model. Therefore, for the equivalence invariant candidate $(x = y)$, the constraints added to the two-timeframe, assume-and-prove model are $(x_1 + \overline{y_1})$ and $(\overline{x_1} + y_1)$, and the objective to be proved is $(x_2 = y_2)$.

Figure 4 gives an example of this model for equivalence invariant candidate $(x = y)$.

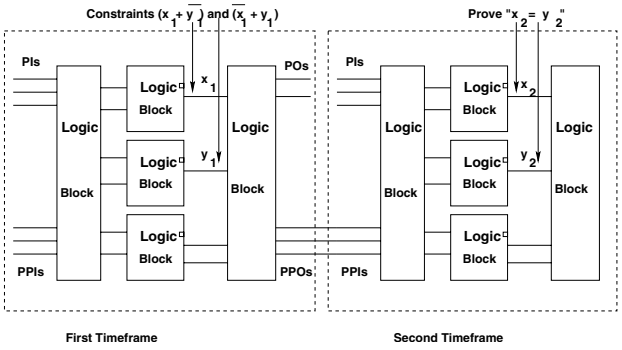

Figure 4: An example of the two-timeframe, assume-and-prove model.

The experimental results of *IProver* are shown in Table III. Columns "#eq" give the number of equivalent signals detected as 0-th invariants, 1-st invariants, and 2-nd invariants respectively. Columns "#const" give the number of detected constant signals. Columns "time" show the CPU time (in seconds). Column "n" gives the smallest value of $k$ if for the miter output $o$, "$o = 0$" is proven to be a $k$-th invariant by *IProver*, where $k$ is 0, 1, or 2. Otherwise, Column "n" shows "NA". These results show that for most of the cases, more signals are identified as 1-st invariants than as 0-th invariants, and more signals are identified as 2-nd invariants than as 1-st invariants. For cases in6 and in7, the miter output is identified as 2-nd invariants but not as 0-th invariants or 1-st invariants by *IProver*, even though *BMChecker* could prove that the miter output $o$ is constant 0 in timeframes 0 and 1 for the two cases.

TABLE III: EXPERIMENTAL RESULTS OF IPROVER

| Circuit | 0-th invariant | | | 1-st invariant | | | 2-nd invariant | | | n |
|---|---|---|---|---|---|---|---|---|---|---|
| | #eq | #const | time | #eq | #const | time | #eq | #const | time | |
| in1 | 1345 | 83 | 1093 | 1671 | 128 | 1151 | 1804 | 129 | 982 | NA |
| in2 | 1087 | 186 | 1477 | 1087 | 186 | 1495 | 1087 | 186 | 1566 | NA |
| in3 | 286 | 35 | 8849 | 422 | 84 | 4145 | 510 | 117 | 2485 | 0 |
| in4 | 292 | 37 | 6635 | 432 | 86 | 3119 | 534 | 119 | 3210 | 0 |
| in5 | 350 | 86 | 3 | 423 | 87 | 4 | 423 | 87 | 4 | 0 |
| in6 | 36429 | 8895 | 2902 | 38667 | 9622 | 3236 | 38796 | 9644 | 822 | 2 |
| in7 | 36749 | 8865 | 2978 | 38989 | 9589 | 4077 | 39118 | 9611 | 832 | 2 |

### III-D. 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 details of the solver can be found in [15]. In this section, we mainly discuss the modifications to the solver when *k*-th invariants are added as constraints to prune the search space.

One of the original stopping criteria of *Seq_SAT* is that when a solution state covers the initial state, the solver stops and reports a solution. However, when constraints of *k*-th invariants are added to the sequential circuit, *Seq_SAT* may not reach the initial state anymore. Thus a satisfiable problem may become unsatisfiable after adding these constraints.

For example, assume a flip-flop in a sequential circuit has an initial value 0, and becomes constant 1 starting from timeframe 1. Therefore, the corresponding pseudo primary input signal *ppi* is a 1-st constant invariant signal. When a unique literal clause $(ppi)$ is added to constrain *ppi* to 1, the initial state can no longer be reached by *Seq_SAT*.

This issue can be solved by changing the stopping criterion to the following one. If the solution state can be reached from the initial state at timeframe *k*, *Seq_SAT* stops and reports the solution. Using this stopping criterion, *Seq_SAT* can employ *BMChecker* to determine whether a solution state can be reached from the initial state at timeframe *k*. The problem of adopting this stopping criterion is that *Seq_SAT* may not 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, provided that *k* is not too large.

## IV. Experimental Results

The experimental results of our sequential equivalence checker are given in Table IV. In these experiments, the limit *k* for *BMChecker* is set to 10. The examples are miter circuits constructed from the industrial examples as those benchmarks in Table I, and cannot be solved by *Seq_SAT* alone in 10 hours, even after simplification. Column "#ff" ("#g") is the number of flip-flops (gates) of the circuit before simplification, Column "runtime(sec)" shows the CPU time (in seconds), and Column "status" shows the part in which the problem is solved. Column "result" gives the verification result where "EQ" means the two circuits are equivalent and "NEQ" indicates they are not equivalent.

The experimental results show that these examples could be solved by our sequential equivalence checker in reasonable time, and demonstrate the effectiveness of our SEC framework.

TABLE IV: EXPERIMENTAL RESULTS OF THE SEC FRAMEWORK

| Circuit | #ff | #g | runtime(sec) | status | result |
|---|---|---|---|---|---|
| in3 | 108 | 2516 | 5405 | IProver | EQ |
| in4 | 116 | 2539 | 6675 | IProver | EQ |
| in5 | 102 | 4854 | 20 | IProver | EQ |
| in6 | 457 | 61589 | 1128 | IProver | EQ |
| in7 | 461 | 62333 | 1093 | IProver | EQ |
| in8 | 183 | 2276 | 1460 | Seq_SAT | NEQ |
| in9 | 159 | 5818 | 35 | BMChecker | NEQ |

*These cases cannot be solved by *Seq_SAT* alone in 10 hours after Flip-Flop mapping.

## V. Conclusion

In this paper, we have introduced the concept of *k*-th invariant, which is a generalization of the traditional invariant. *k*-th invariants can be used in bounded model checking to simplify the expanded circuit model, and can be added as constraints for the sequential SAT solver to prune the search space. We have also presented a bounded model checking tool *BMChecker* and an invariant prover *IProver* that are developed based on circuit SAT solver C-SAT. *BMChecker* not only can efficiently check for solutions of length bounded by a limit *k* for a given objective, but also can identify equivalent or constant signals in the expanded circuit model. The equivalent or constant signals derived by *BMChecker* in timeframe *k* are used as initial *k*-th invariant candidates, from which *IProver* identifies true *k*-th invariants. The framework built upon *BMChecker*, *IProver* and *Seq_SAT* has proven to be powerful for addressing the sequential equivalence checking problem for a wide range of industrial 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. *Proc. Int. Conf. Computer-Aided Design*, pp. 126-129, 1990.

[3] C. Pixley, A Theory and Implementation of Sequential Hardware Equivalence. *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1469-1478, Dec. 1992.

[4] P. Ashar, A. Gupta, and S. Malik, Using Complete-1-Distinguishability for FSM Equivalence Checking. *Proc. Int. Conf. Computer-Aided Design*, pp. 346-353, Dec. 1996.

[5] J.-H. Jiang, R. K. Brayton, On the Verification of Sequential Equivalence. *IEEE Trans. Computer-Aided Design*, 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. Computers*, vol. 49, no. 5, pp. 443-463, May. 2000.

[7] C.A.J. van Eijk, Sequential Equivalence Checking Based on Structural Similarities. *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 814-819, Jul. 2000.

[8] P. Bjesse and K. Claessen, SAT-bsed verification without state space traversal. *FMCAD*, pp. 372-389, 2000.

[9] W.-T. Cheng, The BACK Algorithm for Sequential Test Generation. *Proc. Int. Conf. Computer Design*, pp. 66-69, Oct. 1988.

[10] A. Ghosh, S. Devadas, and A. R. Newton, Test Generation and Verification for Highly Sequential Circuits. *IEEE Trans. Computer-Aided Design*, pp. 652-667, May 1991.

[11] D. Brand, Verify Large Synthesized Designs. *Proc. Int. Conf. Computer-Aided Design*, pp. 534-537, Nov. 1993.

[12] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, Chaff: Engineering an efficient SAT solver. *Proc. of ACM/IEEE Design Automation Conf.*, pp. 530-535, June 2001.

[13] E. Goldberg and Y. Novikov, BerkMin: A fast and robust Sat_Solver. *Proc. European Design and Test Conf.*, pp. 142-149, March 2002.

[14] Feng Lu, Li-C. Wang, Kwang-Ting Cheng, A circuit SAT solver with signal correlation guided learning. *Proc. European Design and Test Conference* 2003.

[15] 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. *Proc. European Design and Test Conference* 2005.

[16] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, Symbolic Model Checking using SAT Procedures instead of BDDs. *Proc. of the 36th ACM/IEEE Design Automation Conf.*, pp. 317-320, June 1999.

[17] Andreas Kuehlmann, Dynamic Transition Relation Simplification for Bounded Property Checking. *Proc. Int. Conf. Computer-Aided Design*, pp. 50-57, Nov. 2004.

[18] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, Symbolic model checking without BDDs. *Proc. Int. Conf. Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, pp. 193-207, Mar. 1999.

[19] J. Baumgarter, A. Kuehlmann, and J. Abraham, Property checking via structure model analysis. *Proc. Computer-Aided Verification (CAV)*, pp.151-165, Jul. 2002.

[20] D. Kroening and O. Strichman, Efficient computation of recurrence diameters. *Proc. Int. Conf. Verification, Model Checking, and Abstract Interpretation*, Jan. 2003.

[21] K. L. McMillan. Interpolation and SAT based Model Checking. Proc. of Int. Conf. Computer-Aided Verification, Vol 2725, LNSC, pp. 1-13, 2003.

[22] M. Sheeran, S. Singh and G. Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. Proc. of Int. Conf. Formal Methods in Computer-Aided Design, pp. 108-125, 2000.

[23] L. D. Moura, H. Rueβ, M. Sorea. Bounded Model Checking and Induction: From Refutation to Verification. Proc. of Int. Conf. Computer-Aided Verification, Vol 2725, LNSC, pp. 14-26, 2003.

[24] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. Proc. of Int. Conf. Computer-Aided Verification, Vol 2404, LNSC, pp. 250-264, 2002.

[25] H.-J. Kang, I.-C. Park. SAT-Based Unbounded Symbolic Model Checking. Proc. of ACM/IEEE Design Automation Conf., pp. 840-843, June 2003.

[26] A. Kuehlmann, M. Ganai, and V. Paruthi, "Circuit-based Boolean Reasoning," in *Proc. ACM/IEEE Design Automation Conference*, June 2001, pp. 232-237.