Inductive Equivalence Checking under Retiming and Resynthesis

Jie-Hong R. Jiang and Wei-Lun Hung

Department of Electrical Engineering/Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan

ABSTRACT

Retiming and resynthesis are among the most important techniques for practical sequential circuit optimization. However, their applicability is much limited due to verification concerns. Overcoming the verification bottleneck is a supreme task. This paper studies both the theoretical and practical aspects of inductive verification on the equivalence between circuits under retiming and resynthesis transformation. We study the completeness condition of the inductive approach to equivalence checking and show that prior work is only complete for circuits transformed under retiming or resynthesis, but not both. We overcome prior limitation and make complete the equivalence checking for circuits transformed up to retiming+resynthesis+retiming. The theoretical insights lead to a robust satisfiability formulation of verification under various retiming and resynthesis scenarios. Experimental results demonstrate the scalability of the approach. Several previously unverifiable circuits and unverifiable transformation scenarios can now be verified effectively.

1. INTRODUCTION

Retiming [13] is an elementary yet effective technique in optimizing synchronous hardware systems. By simply repositioning registers, it is capable of rescheduling computation tasks in an optimal way subject to various design criteria. However, since retiming preserves circuit structures expect for register locations, its optimization power is limited. Resynthesis [4, 17] was proposed, in combination with retiming, to allow structural modification. Despite the effectiveness of retiming and resynthesis in circuit optimization, they are not widely used in hardware synthesis flows due to the verification hindrance. Overcoming the bottleneck is an important task to enhance the applicability of retiming and resynthesis.

A recent result [11] showed that equivalence checking of circuits under unbounded retiming and resynthesis transformation is PSPACE complete, i.e., as hard as the general sequential equivalence checking. Unless synthesis history is kept or the transformation of retiming and resynthesis is bounded, it is unlikely to have a satisfactory solution to the intractable problem. This paper is concerned with equivalence checking under bounded transformation of retiming and resynthesis.

Prior work on the subject focuses mostly on retiming verification, e.g., [6, 9, 5, 16], where the key issue is how to exploit structural similarities to simplify verification tasks. While exploiting structural similarities has been a standard technique in practical combinational equivalence checking, it is not as straightforward in sequential verification because register correspondence does not always exist and thus structural similarities are harder to identify. In [5], an inductive approach was proposed to automate the detection of signal correspondence. Although the technique is very useful, there are still some limitations on its practicality. Firstly, the so-found structural similarities are only complete for retiming or resynthesis verification, but not both. Secondly, an excessive amount of signal pairs are checked iteratively. It substantially slows down verification. Thirdly, BDD-based computation is inherently nonrobust. This paper overcomes the above limitations.

Through the study of the fundamental condition that makes inductive signal correspondence complete for retiming verification, we are able to extend the verification capability and devise reduction techniques. The main results include: 1). Complete verification procedures are obtained for equivalence checking under several retiming and resynthesis scenarios. 2). Reduction techniques are proposed to simplify verification tasks. 3). Robust SAT-based formulation is addressed to extend verification capacity. Thereby, several previously unverifiable instances can now be verified. We hope that through this study the practicality of retiming and resynthesis can be enhanced.

This paper is organized as follows. Section 2 gives the preliminaries. Our main theoretical results are presented in Section 3 and the practical implementation issues are discussed in Section 4. The proposed methods are then evaluated with experiments in Section 5. Our results are contrasted with prior work in Section 6. Section 7 gives the concluding remarks.

2. PRELIMINARIES

2.1 Circuits and Finite State Machines

A circuit $C = (G, N)$ consists of gates G and directed weighted two-terminal nets N. Without loss of generality, we consider a fanout stem as a dummy gate with an identity function, and thus a multi-terminal net can be treated as a set of two-terminal nets. Every gate $g \in G$ represents either a logic/dummy gate or a primary input/output. Every net $e = (g_1, g_2) \in N \subseteq G \times G$ represents a connection from g_1 to g_2, and is weighted by the number of registers (state-holding elements) on the net. Often a register is of some specific initial value, either 0 or 1. In the sequel, a *signal* of a circuit is meant to be the input or output of a gate or register. Hence, several nets of a circuit may be of the same signal (due to dummy gate fanouts); a net may correspond to several signals (due to net weights).

Definition 1. *The* register boundary *of a circuit C is the set of register input and output signals. In particular, the register input (respectively output) signals is called the* register input *(respectively* output*)* boundary.

Definition 2. *A signal set S of a circuit C forms a* feedback edge set *if cutting C at the signals in S makes C acyclic.*

We assume that a circuit contains no combinational loops. Hence, the register input (output) boundary of a circuit must form a feedback edge set.

DEFINITION 3. *The* register depth *of a circuit C with respect to a feedback edge set S is the largest number of registers along any path in the induced acyclic circuit after cutting C at the signals of S.*

(Cutting C at a signal s results in a new circuit C', where a pseudo-primary output and input are added to the input- and output-side of the cut, respectively.)

In the behavior viewpoint, a circuit implements some deterministic and completely specified finite state machine (FSM).

DEFINITION 4. *A* finite state machine $\mathcal{M}$ *is a six-tuple $(Q, I, \Sigma, \Omega, \delta, \lambda)$, where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, Σ and Ω are the sets of input and output alphabets, respectively, and $\delta : \Sigma \times Q \to Q$ (respectively $\lambda : \Sigma \times Q \to \Omega$) is the transition function (respectively output function).*

2.2 Retiming and Resynthesis

2.2.1 Retiming

Retiming is made up from atomic moves of registers in forward and backward directions across a gate [13]. Any retiming of a circuit $C = (G, N)$ can be specified by a retime function $\rho : G \to \mathbb{Z}$, where $\rho(g)$ denotes the *lag* of gate $g \in G$, namely, the number of unit delays (in clock cycles) added to the output signal of gate g. Thus, a lag is positive (negative) for backward (forward) retime. Moreover, we assert that $\rho(g) = 0$ if g is a primary input or output. That is, no retime moves are made over primary inputs and outputs. (We are concerned with *valid* retime functions. That is, any net after retiming is of nonnegative weight.) Note that, if circuit C' is retimed from C through ρ, then C is retimed from C' through $-\rho$.

As retiming repositions registers, it changes the register boundary of a circuit. When a circuit C and its retimed version C' are compared, any register can be moved forward, backward, or stay unmoved. We may accordingly divide the set of signals in C into several subsets as follows.

DEFINITION 5. *Given a circuit $C = (G, N)$ and retime function ρ, the* retime region $\langle C \rangle_\rho$ *of C with respect to ρ is $|C\rangle_\rho \cup \langle C|_\rho$, where*

$$|C\rangle_\rho = \{\text{the input/output signals of } g \in G \mid \rho(g) < 0\} \text{ and}$$
$$\langle C|_\rho = \{\text{the input/output signals of } g \in G \mid \rho(g) > 0\}$$

are the forward *and* backward retime regions*, respectively.*

We extend the above definition to the *largest retime region* of a circuit $C = (G, N)$ (under all possible retimings) as $\langle C \rangle = |C\rangle_{\rho_\triangleright} \cup \langle C|_{\rho_\triangleleft}$, where $\rho_\triangleright$ and $\rho_\triangleleft$ are the retime functions of the most forward and most backward retimings, respectively. In particular, we call $|C\rangle_{\rho_\triangleright}$ (also denoted as $|C\rangle$) and $\langle C|_{\rho_\triangleleft}$ (also denoted as $\langle C|$) the *largest forward* and *backward retime regions* of C, respectively.[1]

[1]In retiming a circuit most forwardly, the lag of a gate cannot be bounded from above when some feedback loop is not in the transitive fanout cone of any primary input. However, it does not affect our discussion of the forward retime region. In contrast, there is no such problem in most backward retiming since, if a feedback loop is not in the transitive fanin cone of any primary output, it is redundant and can be removed from the circuit.

2.2.2 Resynthesis

Resynthesis, on the other hand, is a structural rewriting of a combinational function. Unlike retiming, resynthesis changes the circuit structure but not the global functionality.

2.2.3 Retiming and Resynthesis

Retiming and resynthesis can be applied alternatively and iteratively for circuit optimization [4, 17]. An (ordered) sequence of retiming and resynthesis operations is denoted as $\widehat{\mathbf{op}} = \mathrm{op}_1 + \mathrm{op}_2 + \cdots + \mathrm{op}_n$, where $\mathrm{op}_i \in \{\text{retiming, resynthesis}\}$. Without loss of generality, we assume that $\mathrm{op}_i \neq \mathrm{op}_{i+1}$ since two consecutive retiming (similarly, resynthesis) operations can be seen as a single transformation. In the sequel, we say two circuits are $\widehat{\mathbf{op}}$ *equivalent*, or *equivalent under* $\widehat{\mathbf{op}}$, if one circuit can be derived from the other with $\widehat{\mathbf{op}}$. In particular, $\widehat{\mathbf{op}}$ in the sequel can be retiming, resynthesis, retiming+resynthesis, resynthesis+retiming, or retiming+resynthesis+retiming. In fact, the more iterations are performed, the harder the circuits can be verified [11].

2.3 Signal Correspondence and Retiming Invariants

Safety property checking (which includes sequential equivalence checking) over an FSM $(Q, I, \Sigma, \Omega, \delta, \lambda)$ can be considered as identifying some state set R satisfying

$$I(s) \Rightarrow R(s), \tag{1}$$
$$R(s) \Rightarrow R(\delta(x, s)), \text{ and} \tag{2}$$
$$R(s) \Rightarrow P(s), \tag{3}$$

for all $x \in \Sigma, s \in Q$. (Notationally we shall not distinguish a set and its characteristic function.) That is, R contains the initial states I by Eq. (1) and is closed under state transition δ by Eq. (2). Moreover, all states in R must satisfy the desired property P by Eq. (3). In general, exact reachability analysis may be needed to compute the smallest or largest such R, which is costly, in fact, PSPACE-complete in the number of state variables.

Given two FSMs $\mathcal{M}_1 = (Q_1, I_1, \Sigma, \Omega, \delta_1, \lambda_1)$ and $\mathcal{M}_2 = (Q_2, I_2, \Sigma, \Omega, \delta_2, \lambda_2)$, their *product FSM* is $(Q_\times, I_\times, \Sigma, \mathbb{B}, \delta_\times, \lambda_\times)$, where $Q_\times = Q_1 \times Q_2$, $I_\times = I_1 \times I_2$, $\delta_\times = (\delta_1, \delta_2)$, and $\lambda_\times = (\lambda_1 \equiv \lambda_2)$. That is, the corresponding *product circuit* $C_\times$ is built from the circuits C_1 of $\mathcal{M}_1$ and C_2 of $\mathcal{M}_2$ by sharing the inputs and adding an equivalence comparator at the outputs. In the sequel, we are concerned with checking the equivalence between C_1 and C_2.

Given two circuits, where one is retimed from the other, there exists a *retiming invariant* [16] that specifies a relation among state variables between these circuits and satisfies the above three conditions with property P asserting the equivalence of their outputs. Retiming invariant is easy to obtain if signal correspondences between two circuits are known. Van Eijk [5] proposed an inductive approach to automate the identification of such correspondences. It can be seen as over-approximating reachability analysis and is more robust than exact analysis.

Any signal f of a circuit can be expressed as a function over the primary input variable x and current-state variable s. In fact, the equivalence between two signals, in a sequential sense, can be more general than the equivalence of their combinational functions. We may speak of the observational equivalence of two signals under any execution of an FSM. To avoid the costly exact reachability analysis, we exploit a subset of the equivalence relation by induction as follows.

DEFINITION 6 ([5]). *An* inductive signal correspondence *(or briefly* signal correspondence*), denoted as $\doteqdot$, of a circuit C implementing an FSM $(Q, I, \Sigma, \Omega, \delta, \lambda)$ is an equiva-*

InductiveSignalCorrespondence
input: a sequential circuit (with signals S) implementing $\mathcal{M} = (Q, I, \Sigma, \Omega, \delta, \lambda)$
output: the inductive register correspondence of $\mathcal{M}$
begin
01 $i := 0$
02 $\fallingdotseq^{(i)} := \bigwedge(f_p \equiv f_q)$ for $\{(f_p, f_q) \in S \times S \mid \forall x \in \Sigma, s \in Q. [I(s) \Rightarrow (f_p(x,s) \equiv f_q(x,s))]\}$
03 **repeat**
04 $i := i + 1$
05 $\fallingdotseq^{(i)} := \bigwedge(f_p \equiv f_q)$ for $\{(f_p, f_q) \mid \forall x, x' \in \Sigma, s \in Q. [\fallingdotseq^{(i-1)} \Rightarrow ((f_p \equiv f_q) \wedge (f_p(x', \delta(x,s)) \equiv f_q(x', \delta(x,s))))]\}$
06 **until** $\fallingdotseq^{(i)} = \fallingdotseq^{(i-1)}$
07 **return** $\fallingdotseq^{(i)}$
end

Figure 1: Computation of signal correspondence $\fallingdotseq$.

lence relation over a set S of signals in C such that signals $f_i, f_j \in S$ with $\fallingdotseq \Rightarrow (f_i \equiv f_j)$, i.e. f_i and f_j are corresponding signals, satisfy

$$I(s) \Rightarrow (f_i(x,s) \equiv f_j(x,s)), \text{ and} \quad (4)$$

$$\fallingdotseq(x,s) \Rightarrow (f_i(x', \delta(x,s)) \equiv f_j(x', \delta(x,s))) \quad (5)$$

for all $x, x', \in \Sigma$ and $s \in Q$.

(For brevity, in the sequel we abuse the notation $(f_i, f_j) \in \fallingdotseq$ to mean $\fallingdotseq \Rightarrow (f_i \equiv f_j)$.) Note that the above definition can be slightly generalized by considering signal pairs complement to each other as corresponding signals. For brevity, our discussion focuses on true identical signals. Definition 6 suggests a fixed-point computation as shown in Figure 1.

The connection between signal correspondence and equivalence checking is as follows.

PROPOSITION 1 ([5]). *Two circuits C_1 and C_2 are equivalent if the signal correspondence $\fallingdotseq$ of their product circuit $C_\times$ satisfies*

$$\fallingdotseq(x,s) \Rightarrow \lambda_\times(x,s) \quad (6)$$

for all $x \in \Sigma, s \in Q_\times$.

Note that Eq. (6) can be checked alternatively by embedding $(\lambda_1 \equiv \lambda_2)$ into $\fallingdotseq^{(0)}$ of the computation of Figure 1 with $C_\times$ as the input circuit. Thereby, C_1 and C_2 are equivalent if $(\lambda_1 \equiv \lambda_2)$ remains in $\fallingdotseq^{(i)}$ upon return.

In the context of equivalence checking between C_1 and C_2, in the sequel, for $(f_i, f_j) \in \fallingdotseq$ of $C_\times$, we shall assume for simplicity that f_i and f_j are signals of C_1 and C_2, respectively. Although f_i and f_j can be signals of the same circuit, their appearance in $\fallingdotseq$ does not affect the conclusion of (in)equivalence between the two circuits.

3. THEORETICAL INVESTIGATION

We prove below an unjustified statement in [5] that signal correspondence is complete for retiming verification. The proof helps us understand what conditions make the approach complete and how to simplify verification tasks.

LEMMA 1. *Let $C_\times$ be the product of two circuits C_1 and C_2. If C_2 is retimed from C_1, then*

$$\forall x \in \Sigma, s \in Q_\times.\ \fallingdotseq(x,s) \Rightarrow \lambda_\times(x,s), \quad (7)$$

where $\fallingdotseq$ is the signal correspondence of virtually forward retimed[2] *$C_\times$.*

[2]A circuit C is *virtually forward retimed* if the signals that can be produced through any forward retiming are added to C [5].

PROOF. Since $C_\times$ is virtually forward retimed, any signal that can be produced through forward retiming is added to the constituent circuit C_1 or C_2 of $C_\times$. Hence, in $C_\times$, the input and output signals of every forwardly retimed register of one constituent circuit (forward with respect to the other constituent circuit) have corresponding signals in the other. Let P be the set of such corresponding pairs.

We show that signal pairs P remain in $\fallingdotseq$ throughout the refinement procedure of Figure 1. By induction, for the base case, it can be checked that the signal pairs of P must present in $\fallingdotseq^{(0)}$ for correct retiming. For the induction step, assume signal pairs P are all present in $\fallingdotseq^{(k)}$. Observe that, for any signal pair (f_i, f_j) in P, f_i and f_j are isomorphic functions over equivalent input signals (each equivalent input pair is either the same primary input or a signal pair in P). Therefore, the valuation update of f_i and f_j for the next timeframe can always be expressed in terms of corresponding primary inputs and signal pairs in P. That is, adding one more timeframe does not make f_i and f_j inequivalent. Hence all of the signal pairs in P are present in $\fallingdotseq^{(k+1)}$ as well. By induction, we conclude signal pairs P remain in $\fallingdotseq$.

Finally, since C_1 and C_2 are retiming equivalent, their any corresponding primary output pair again consists of two isomorphic functions over equivalent input signals. Thus, $\fallingdotseq \Rightarrow \lambda_\times$. ■

We learn from the above proof that the signal pairs P, instead of all possible corresponding signal pairs, suffices to demonstrate the implication of Eq. (7). In fact, P forms a feedback edge set in $C_\times$. It suggests a way of deriving weak $\fallingdotseq$, yet complete for retiming equivalence checking. That is, we do not need to compute *maximal* signal correspondence in order to show retiming equivalence. The reduction in corresponding signal pairs turns out to be helpful in SAT-based verification as to be discussed in Section 4.

The following theorem summarizes the connection between signal correspondence and retiming equivalence checking.

THEOREM 1. *Signal correspondence $\fallingdotseq$ is both sound and complete (i.e. no false negative) in checking retiming equivalence provided that virtual forward retiming is performed.*

PROOF. The soundness follows from Proposition 1; the completeness follows from Lemma 1. ■

Essentially, checking retiming equivalence is combinational equivalence checking if register correspondence is known.

On the other hand, a result similar to Lemma 1 holds for resynthesis transformation.

LEMMA 2. *Let $C_\times$ be the product of two circuits C_1 and C_2. If C_2 is resynthesized from C_1, then*

$$\forall x \in \Sigma, s \in Q_\times.\ \fallingdotseq(x,s) \Rightarrow \lambda_\times(x,s),$$

where $\fallingdotseq$ is the signal correspondence of $C_\times$.

PROOF. Observe that resynthesis does not change transition and output functions. Thus any signal in the register boundary of C_1 has a corresponding signal in C_2, and vice versa. The signal correspondence induces the equivalence between λ_1 and λ_2 for correct resynthesis transformation. ■

3.1 Timeframe Expansion

Although virtual forward retiming makes signal correspondence complete for retiming equivalence checking, it has two main limitations. Firstly, it is insufficient for checking the equivalence of circuits transformed with both retiming and resynthesis because signal pairs essential to equivalence checking can be missing. In this case, virtual backward retiming

InductiveSignalCorrespondence_k-Timeframe
input: k and a sequential circuit (with signals S) implementing $\mathcal{M} = (Q, I, \Sigma, \Omega, \delta, \lambda)$
output: the inductive register correspondence of $\mathcal{M}$
begin
01 $i := 0$
02 $\fallingdotseq_k^{(i)} := \bigwedge(f_p \equiv f_q)$ for $\{(f_p, f_q) \in S \times S \mid \forall x^1, \ldots, x^k \in \Sigma, s \in Q.[I \Rightarrow \bigwedge_{t=1}^{k}(f_p^t \equiv f_q^t)]\}$
03 **repeat**
04 $i := i + 1$
05 $\fallingdotseq_k^{(i)} := \bigwedge(f_p \equiv f_q)$ for $\{(f_p, f_q) \mid \forall x^1, \ldots, x^{k+1} \in \Sigma, s \in Q.[(\fallingdotseq_k^{(i-1)} \Rightarrow (f_p \equiv f_q)) \wedge ((\bigwedge_{t=1}^{k} \fallingdotseq_k^{(i-1)\,t}) \Rightarrow (f_p^{k+1} \equiv f_q^{k+1}))]\}$
06 **until** $\fallingdotseq_k^{(i)} = \fallingdotseq_k^{(i-1)}$
07 **return** $\fallingdotseq_k^{(i)}$
end

Figure 2: Computation of signal correspondence $\fallingdotseq_k$.

is needed, which is, however, impractical due to the possible nonexistence and nonuniqueness of valid initial states. Secondly, the extra signals added by virtual forward retiming induce more signal pairs to be checked in $\fallingdotseq^{(i)}$ of Figure 1. This side effect is critical especially for SAT-based formulation of equivalence checking as to be discussed in Section 4.

We abandon virtual retiming and overcome the above limitations through timeframe expansion.

Definition 7. *Given a circuit C with transition function $\delta(x, s)$, the* k-timeframe expansion *of signal $f(x, s)$ in C, denoted as f^k, is*

$$f^k(x^k, \ldots, x^1, s) = f(x^k, \delta(x^{k-1}, \delta(\ldots, \delta(x^1, s)))), \qquad (8)$$

where input variable x is instantiated with time indices given in superscript.

That is, f^k is a timed Boolean formula.

The signal correspondence of Definition 6 can be generalized under the k-timeframe expansion as follows.

Definition 8. *A* signal correspondence under the k-timeframe expansion*, denoted as $\fallingdotseq_k$, of a circuit C implementing an FSM $(Q, I, \Sigma, \Omega, \delta, \lambda)$ is an equivalence relation over a set S of signals such that signals $f_i(x, s), f_j(x, s) \in S$ with $(f_i, f_j) \in \fallingdotseq_k$ satisfy*

$$I \Rightarrow \bigwedge_{t=1}^{k}(f_i^t \equiv f_j^t), \text{ and} \qquad (9)$$

$$(\bigwedge_{t=1}^{k} \fallingdotseq_k^t) \Rightarrow (f_i^{k+1} \equiv f_j^{k+1}) \qquad (10)$$

for all $x^1, \ldots, x^{k+1} \in \Sigma, s \in Q$.

Similar to Figure 1, Figure 2 shows the computation for $\fallingdotseq_k$. Moreover, the proposition below is analogous to Proposition 1.

Proposition 2. *Two circuits C_1 and C_2 are equivalent if the signal correspondence $\fallingdotseq_k$ of their product circuit $C_\times$ satisfies*

$$\fallingdotseq_k(x, s) \Rightarrow \lambda_\times(x, s) \qquad (11)$$

for all $x \in \Sigma, s \in Q_\times$.

Proof. Observe that $(f_i, f_j) \in \fallingdotseq$ implies $(f_i, f_j) \in \fallingdotseq_k$. By the transitivity of implication, we have $\fallingdotseq_k(x, s) \Rightarrow \fallingdotseq(x, s) \Rightarrow \lambda_\times(x, s)$. The proposition follows. ■

Checking Eq. (11) is similar to checking Eq. (6), and can be alternatively embedded into the $\fallingdotseq_k$ computation.

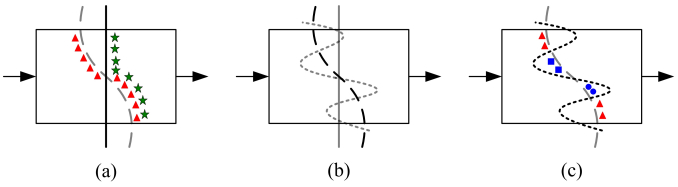

Figure 3: (a) Original circuit C with register boundary in solid line. (b) Retimed (and then resynthesized) circuit $C^\dagger$ with register boundary in dashed line. (c) Further retimed circuit $C^\ddagger$ with register in dotted line.

The intuition behind timeframe expansion is not to create intermediate signals as in virtual retiming, but rather to bridge the missing temporal links among signals between C_1 and C_2 through timed Boolean formulas. Thus, unlike in virtual retiming, the signal pairs to be checked is not increased due to timeframe expansion, even though we do need to check the same signal pair in different timeframes.

A useful property connecting the feedback edge set and timeframe expansion of a circuit is as follows.

Proposition 3. *Given a feedback edge set S of a circuit C, there exists some constant k such that any signal of C can be represented as a timed Boolean formula over the variables of the signals in S with time indices no greater than k. In particular, k is bounded from above by the register depth of C with respect to S.*

Hence as long as signals $S_1 = \{f_1 \mid (f_1, f_2) \in \fallingdotseq_k$ with f_1 of C_1, f_2 of $C_2\}$ and $S_2 = \{f_2 \mid (f_1, f_2) \in \fallingdotseq_k$ with f_1 of C_1, f_2 of $C_2\}$ form feedback edge sets in C_1 and C_2, respectively, any signal of $C_\times$ can be expressed as a timed Boolean formula over primary input variables and the corresponding variables of signals in S_1 and S_2. In this case, corresponding signal pairs in $\fallingdotseq_k$ can be used to check the equivalence between C_1 and C_2; not all signal pairs need to be considered in $\fallingdotseq_k$.

From practical experience, k is small in timeframe expansion. In our experimental study, $k = 1$ (i.e. no extra timeframe expansion) suffices for most equivalence checking instances. Also, for equivalence checking under several retiming and resynthesis scenarios, only a modest part of the signal pairs suffices to show the (in)equivalence as long as it forms a feedback edge set. The reduction in signal pairs is important in our SAT-based verification.

3.2 Equivalence Checking

Below we study the equivalence checking under various retiming and resynthesis scenarios. We assume that the transformation history is unknown. The original and final circuits are to be compared for their equivalence.

3.2.1 *Retiming Equivalence*

Let circuit C_2 be derived from C_1 by retiming. We are concerned with checking their equivalence. Signal correspondence $\fallingdotseq_k$ is complete for retiming equivalence checking assuming the checked signal pairs in the procedure of Figure 2 are comprehensive enough. However, to avoid checking unnecessary signal pairs, we characterize a minimal set of signal pairs to be checked sufficient for maintaining the completeness property. We propose the following two constructions:
Construction T1: Every signal in the register output boundary of C_1 is paired up with every signal in the forward retime region $|C_2\rangle$, and vice versa.
The validity of this construction is asserted below.

Theorem 2. *Signal pairs created by Construction T1 are*

sufficient for complete retiming equivalence checking. In particular, Proposition 2 gives the equivalence condition for C_1 and C_2.

PROOF. Consider Figure 3 (a) and (b). Let $C_1 = C$ and $C_2 = C^\dagger$. The signals in the register output boundary of C_1 are paired up with the signals in $|C_2\rangle$; the signals in the register output boundary of C_2 are paired up with the signals in $|C_1\rangle$. As a result, register output signals signified in "$\star$" are common signals exist in both C_1 and C_2. Thus, they form corresponding signals between C_1 and C_2. Every such corresponding signal pair is included in Construction T1. Moreover, the $\star$ signals form a feedback vertex set in both C_1 and C_2. (Otherwise a combinational cycle exists in the original circuit and that violates our assumption.) With timeframe expansion, it is guaranteed that the outputs of C_1 and C_2 can be expressed as timed Boolean formulas over the $\star$ signals. The signal correspondence between C_1 and C_2 induces the condition that the primary output signals must be equivalent if the retiming is performed correctly. ■

Construction T2: Every signal in the register (both input and output) boundary of one circuit, say C_1, is paired up with every signal in the retime region of the other circuit, $\langle C_2\rangle$.

THEOREM 3. *Signal pairs created by Construction T2 are sufficient for complete retiming equivalence checking. In particular, C_1 and C_2 are equivalent if $I_\times(s) \Rightarrow \lambda_\times(x,s)$ and $\leftrightarrows_k(x,s) \Rightarrow \lambda_\times(x', \delta(x,s))$ for all $x, x' \in \Sigma, s \in Q_\times$.*

PROOF. Consider Figure 3 (a) and (b). Let $C_1 = C$ and $C_2 = C^\dagger$. The register (input and output) boundary signals of C_2 are paired up with the signals in $|C_1\rangle$. As a result, the signals signified in "▲" are common signals exist in both C_1 and C_2. Thus, they form corresponding signals between C_1 and C_2. Every such corresponding signal pair is included in Construction T2. It is important to note the ▲ signals are on the left-hand and right-hand sides of the register boundary of C_2 due to the backward and forward retiming, respectively, performed on C_1. As a result, the ▲ signals does not form a cut in a *single* timeframe separating the primary-input and current-state signals from the primary-output and next-state signals. However, in a two-timeframe expansion, the ▲ signals in two consecutive timeframes form a cut that separates the primary-input and current-state signals of the first timeframe from the primary-output and next-state signals of the second timeframe. Therefore, the correspondence of the signals in the cut induces the equivalence of the second-timeframe primary outputs of C_1 and C_2 for correct retiming, i.e., $\forall x, x' \in \Sigma, s \in Q_\times . \leftrightarrows_k(x,s) \Rightarrow \lambda_\times(x', \delta(x,s))$. In addition, since $\forall x \in \Sigma, s \in Q_\times . I_\times(s) \Rightarrow \leftrightarrows_k(x,s)$, it implies $\forall x \in \Sigma . \lambda_\times(x,q) \equiv 1$ for any state q reachable from $I_\times$. Therefore, as long as $\forall x \in \Sigma, s \in Q_\times . I_\times(s) \Rightarrow \lambda_\times(x,s)$, the output $\lambda_\times$ of $C_\times$ always produces constant 1. That is, C_1 and C_2 are equivalent. ■

3.2.2 Retiming+Resynthesis Equivalence

Let circuit C_2 be derived from C_1 by retiming followed with resynthesis. To check their equivalence, Construction T2 (but not Construction T1) can be slightly modified to reduce the signal pairs to be checked in computing $\leftrightarrows_k$. In particular, the roles of C_1 and C_2 should be distinguished.

Construction TS: Every signal in the register boundary of C_2 is paired up with every signal in $\langle C_1\rangle$.

The validity of the construction is asserted below.

THEOREM 4. *Signal pairs created by Construction TS are sufficient for complete retiming+resynthesis equivalence checking. In particular, C_1 and C_2 are equivalent if $I_\times(s) \Rightarrow \lambda_\times(x,s)$ and $\leftrightarrows_k(x,s) \Rightarrow \lambda_\times(x', \delta(x,s))$ for all $x, x' \in \Sigma, s \in Q_\times$.*

PROOF. Consider Figure 3 (a) and (b). Let $C_1 = C$ and $C_2 = C^\dagger$. The ▲ signals are those with signal correspondence between the original circuit C_1 and the retimed circuit C_2. Suppose C_2 is further resynthesized. Then the signals in C_1 and C_2 can be completely different except for those in the register boundary of C_2 because resynthesis preserves the transition and output functions. (In contrast, resynthesis on C_2 makes Construction T1 not able to generate all necessary signal pairs for complete verification.) Therefore essential signal correspondence still exists, and the proof is similar to that of Theorem 3. ■

Hence unlike virtual retiming, the timeframe expansion method makes equivalence checking complete for retiming+resynthesis.

3.2.3 Resynthesis+Retiming Equivalence

Checking resynthesis+retiming equivalence is essentially the same as checking retiming+resynthesis equivalence except for switching the roles between C_1 and C_2.

3.2.4 Retiming+Resynthesis+Retiming Equivalence

Let circuit C_2 be derived from C_1 by retiming, resynthesis, and then retiming. To check their equivalence, the following construction can be used to reduce the signal pairs to be checked in computing $\leftrightarrows_k$. Essentially for the verification to be complete, in addition to the signal correspondence within a single timeframe, we need to explore signal correspondence across two adjacent timeframes.

Construction TST: Consider C_1 and C_2 expanded in two timeframes. There are four types of signal pairs to take care. Let $\langle C\rangle^i$ denote the largest retime region of circuit C at the i^{th} timeframe. Every signal in $\langle C_1\rangle^i$ is paired up with every signal in $\langle C_2\rangle^i$, for $i = 1, 2$. In addition, every signal in $\langle C_1\rangle^2$ (respectively $\langle C_1\rangle^1$) is paired up with every signal in $\langle C_2\rangle^1$ (respectively $\langle C_2\rangle^2$). (Therefore $\leftrightarrows_k$ depends on variables x, x', s, where x and x' are the primary inputs of two adjacent timeframes.)

THEOREM 5. *Signal pairs created by Construction TST are sufficient for complete retiming+resynthesis+retiming equivalence checking. In particular, C_1 and C_2 are equivalent if $I_\times \Rightarrow (\lambda^1_\times \wedge \lambda^2_\times)$ and $\leftrightarrows_k \Rightarrow \lambda^3_\times$ for all valuations on input and state variables.*

PROOF. Consider Figure 3 (a) and (c). Let $C_1 = C$ and $C_2 = C^\ddagger$. In Figure 3 (c), the symbol "■" denotes where signal correspondence exists between the signals of C_1 and those of C_2 delayed by one clock cycle; the symbol "•" denotes where signal correspondence exists between the signals of C_2 and those of C_1 delayed by one clock cycle. Construction TST recovers the above signal correspondence across two adjacent timeframes as well as the correspondence of ▲ signals within the same timeframe. Observe that, if the circuits are expanded into three timeframes, the signals marked with ▲, ■, • in (c) in the three timeframes form a cut for every constituent circuit that separates the primary-input and current-state signals of the first timeframe from the primary-output and next-state signals of the third timeframe. Moreover, by Construction TST, every signal f_i on the cut of a circuit has a corresponding signal f_j on the cut of the other circuit such that $(f_i, f_j) \in \leftrightarrows_k$. Therefore, the signal pairs created by Construction TST induce the equivalence of the third-timeframe primary outputs of C_1 and C_2 for correct retiming+resynthesis+retiming, i.e., $\leftrightarrows_k \Rightarrow \lambda^3_\times$. Since $I_\times \Rightarrow \leftrightarrows_k$, it implies $\lambda_\times(x', \delta(x,q)) \equiv 1$ for any state q reachable from $I_\times$ in two or more transitions. Therefore, as long

as $I_\times \Rightarrow (\lambda_\times^1 \wedge \lambda_\times^2)$, the output $\lambda_\times$ of $C_\times$ always produces constant 1. That is, C_1 and C_2 are equivalent. ■

It should be emphasized that the purpose of k-timeframe expansion differs from that of the above discussion. The former is concerned with expressing timed Boolean functions in terms of the feedback edge set signals, whereas the latter is concerned with the correspondence of signals across two adjacent timeframes. They are orthogonal and necessary techniques to make complete the equivalence checking under retiming+resynthesis+retiming.

3.2.5 Resynthesis+Retiming+Resynthesis Equivalence and Beyond

Inductive signal correspondence is incomplete in proving the equivalence under resynthesis+retiming+resynthesis and beyond. The reason is that, once circuit transformation involves two resynthesis steps (interleaved with retiming), signal correspondence may not even exist between the original and transformed circuits. In this case, verification techniques for general sequential equivalence checking may be needed.

4. PRACTICAL IMPLEMENTATION

We formulate the computation of signal correspondence $\eqcirc_k$ as SAT solving. Formally speaking, in the initial step, signal pair (f_p, f_q) is in $\eqcirc_k^{(0)}$ if $I \Rightarrow \bigwedge_{t=1}^k (f_p^t \equiv f_q^t)$, that is, $I \wedge \bigvee_{t=1}^k (f_p^t \not\equiv f_q^t)$ is unsatisfiable. In every refining iteration, (f_p, f_q) is in $\eqcirc_k^{(i+1)}$ if (f_p, f_q) is in $\eqcirc_k^{(i)}$ and $\bigwedge_{t=1}^k \eqcirc_k^{(i)^t} \Rightarrow (f_p^{k+1} \equiv f_q^{k+1})$, that is, $\bigwedge_{t=1}^k \eqcirc_k^{(i)^t} \wedge (f_p^{k+1} \not\equiv f_q^{k+1})$ is unsatisfiable. Thus, checking whether a signal pair remains in $\eqcirc_k$ requires a call to the SAT solver. For the SAT-formulation to be practical, it is of great importance to reduce the number of signal pairs to be checked.

The following techniques are useful making SAT-based checking practical: 1). Constructions T1, T2, TS, TST provide ways of reducing signal pairs to be checked under different retiming and resynthesis scenarios. 2). Multi-timeframe simulation is effective in pruning inequivalent signal pairs. The idea is that, for signal pair (f_i, f_j) to be in $\eqcirc_k$, they must be equivalent in all timeframes. In other words, if two signals appear to be inequivalent in some timeframe, then they can be removed from consideration. Thus with a multi-timeframe expansion we are able to prune more inequivalent signals. Removing inequivalent pairs as early as possible reduces the number of calls to expensive SAT solving. 3). Exploring the transitivity of equivalence relation $\eqcirc_k$ substantially reduces signal pairs to be checked. For instance, $(f_1, f_2), (f_3, f_4), (f_3, f_2) \in \eqcirc_k$ directly implies $(f_1, f_4) \in \eqcirc_k$ without SAT solving. Hence considering complementary signals as corresponding signals (as mentioned right after Definition 6) helps reduce the number of signal pairs to be checked. 4). Incremental SAT solving can be applied in every refining iteration. However, incremental solving across different iterations may not be memory efficient when the number of signal pairs to be checked is large. 5). A satisfying assignment for one inequivalent pair may be useful in proving the inequivalence of other pairs. See also [14, 12] for more useful techniques.

5. EXPERIMENTAL RESULTS

We implemented our algorithm in C++ within the ABC [2] framework and adopted miniSAT [7] as the underlying solver. All experiments were conducted on a Linux machine with Xeon 3.4GHz CPU and 6Gb RAM. The ISCAS benchmark circuits were transformed under retiming and resynthesis to test our equivalence checker.

Table 1 shows the equivalence checking results for retiming by Construction T1, retiming+resynthesis by Construction TS, and retiming+resynthesis+retiming by Construction TST. Column 1 lists the circuits. Column 2 (respectively 3) shows the numbers of AIG nodes (respectively registers) of the original/transformed circuits. CPU time and memory usage are shown in Columns 4 and 5, respectively. Column 6 shows the numbers of iterations taken in pruning inequivalent pairs and the sizes (in parentheses) of timeframe expansion for the circuits to be verifiable. Comparing the three verification problems, we see that the verification gets harder as more retiming and resynthesis steps are involved. The run time is in general proportional to the numbers of iterations and signal pairs to be checked. We were able to check retiming and retiming+resynthesis equivalences up to circuit s38584.1, and retiming+resynthesis+retiming equivalence up to s9234, with CPU time limit of 60,000 secs. Compared with the BDD-based approach [5], our memory usage is small. In contrast, the largest circuit reported in [5] is s5378 with CPU time 801 secs (corresponding to our 70 secs in the retiming+resynthesis case). Another special circuit s3330 took 1316 secs in contrast to our 5 secs. Note that the comparisons are unfair for different experimental circuits and machines, and only provides a rough reference. Moreover, Table 1 reveals that most circuits can be verified within a few pruning iterations and without extra timeframe expansion. Circuit s838.1 is one of the few exceptions, where some inequivalent signal pairs are hard to drop. It takes many iterations to terminate. In such instances, *intelligent simulation* [15] may be helpful in pruning inequivalent pairs more quickly.

Figure 4 compares the reduction power of Constructions T1, T2 and T3, the construction of pairing up all signals between $\langle C_1 \rangle$ and $\langle C_2 \rangle$ for circuits C_1 and C_2 under verification. The column chart has y-axis log-scaled indices on the left-hand side. It shows the numbers of initial signal pairs left after simulation on an 80-timeframe expansion. The line chart with y-axis indices on the right-hand side shows the reduction ratios of Constructions T1, T2, and T3 compared with T4, the construction of forming all signal pairs between C_1 and C_2. The average reduction ratios of T1, T2, and T3 are 0.84, 0.69, and 0.17, respectively. (All the experiments are in terms of retiming equivalence checking.) These ratios indirectly reveal the typical relative sizes among Constructions T1, TS, and TST.

In addition to Figure 4, we study the relation between the number of initial signal pairs (after a 80-timeframe simulation) and the total verification time in Figure 5. The x-axis indicates the run time; the y-axis indicates the number of initial signal pairs in log scale. (Here the reported circuits are chosen for best fit into the log-scaled plot.) For a circuit, four points are plotted with respect to the numbers of initial pairs from Constructions T1, T2, T3, and T4. As expected, the more initial pairs are present, the longer a verification task takes. However, s3384 and s6669 are the exceptions and can be explained as follows. Observe that, since the y-axis is log-scaled, Constructions T1 and T2 do not reduce signal pairs much in the two circuits. It may also due to the fact that adding more constraints makes unsatisfiable a CNF easier to solve. It is also interesting to observe that T2 is the worst construction for s3384 and s6669. It can be explained that the numbers of signals in signal correspondence $\eqcirc$ from C_1 and C_2 under verification are unbalanced. It may lead to inefficient learning during SAT solving. On the other hand, for circuits with many initial signal pairs (typical in large circuits) or involving many pruning iterations (e.g., s635, s838.1), the reduction is very effective.

Figure 6 compares our approach with temporal induction

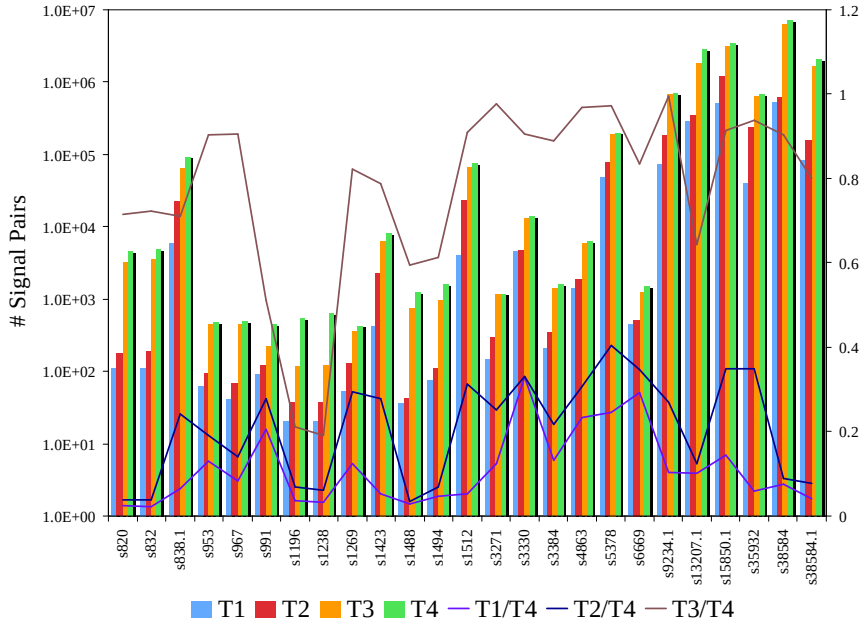

Figure 4: Comparison of reduction techniques.

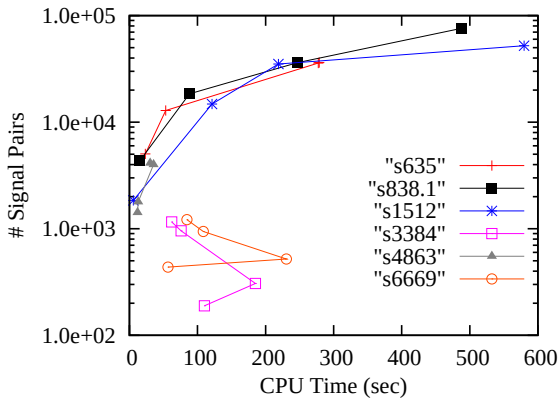

Figure 5: Run time vs. number of initial pairs.

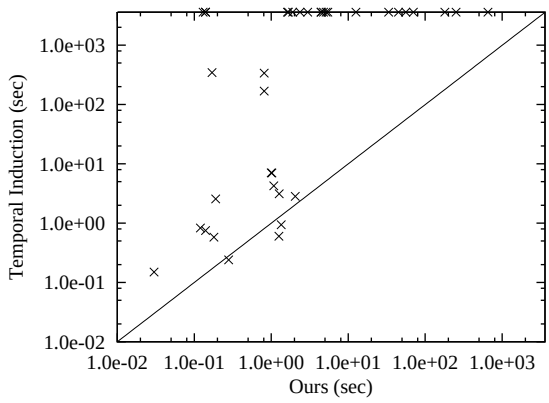

Figure 6: Comparison with temporal induction.

[8] under retiming+resynthesis equivalence checking. A time limit of 3600 secs is imposed. Since temporal induction is a general safety property checking, only primary output pairs are formed to avoid refining an excessive amount of inequivalent pairs. As can be seen, exploiting synthesis structure is very important in circuit equivalence checking.

6. COMPARISONS WITH PRIOR WORK

Retiming Verification. Shenoy *et al.* [18] exploited the structure preserving property of retiming for retiming verification. Firstly, graph isomorphism checking is performed over the two circuits under comparison (ignoring the weights on nets). If two circuits are retiming equivalent, then their structures must be isomorphic. Secondly, loop invariants are asserted. That is, the number of registers on every cycle remains the same before and after retiming. The limited applicability of this method may be due to the lack of effective approaches to the graph isomorphism checking.

Assuming gate-to-gate matching between a circuit and its retimed version is known, Mneimneh and Sakallah [16] showed that the relation among state variables derived from a retime function forms a retiming invariant. In contrast, we do not assume the existence of such matching.

Van Eijk's signal correspondence [5] automates the computation of retiming invariant without knowing gate matching *a priori*. However, it is only complete for retiming verification helped by virtual retiming.

SAT-Based Induction. Bjesse and Claessen [1] strengthened van Eijk's method with *generalized temporal induction* [19]. A SAT-based formulation was proposed to solve the safety property given as the equivalence (or implication) relation among signal pairs. As was not the focus, how to reduce signal pairs in proving retiming equivalence was not addressed. Although the general induction is complete for checking safety properties, it does not exploit the speciality of retiming and resynthesis. It is interesting to note that, unlike [1, 19], no unique-state constraint is needed in our verification.

Verification-Aware Synthesis. A recent approach [3] to the verifiability of circuit transformation, orthogonal to ours, is to maintain synthesis history. Keeping all synthesis history with structural hashing makes inductive invariants

directly derivable from the synthesis database without fixed-point computation. The approach is promising yet inapplicable if the synthesis history is unavailable.

7. CONCLUSIONS

Prior work mostly focused on either retiming verification or general property checking. This paper studied practical equivalence checking for circuits transformed under both retiming and resynthesis. We showed that van Eijk's signal correspondence is only complete for retiming verification even with virtual retime. Through the study of its completeness condition, we extended the verification capability and capacity to the complete checking of equivalence under retiming+resynthesis+retiming. As retiming and resynthesis are among the most important techniques in sequential circuit optimization, our results overcame part of the verification bottlenecks in logic synthesis. We hope that the applicability of retiming and resynthesis can thus be enhanced through our study.

Acknowledgments

The authors would like to thank Bing-Yi Weng for helpful discussions. This work was supported in part by NSC grants 95-2221-E-002-432 and 95-2218-E-002-064-MY3.

REFERENCES

[1] P. Bjesse and K. Claessen. SAT-based verification without state space traversal. In *Proc. FMCAD*, 2000.

[2] Berkeley Logic Synthesis and Verification Group. *ABC: A system for sequential synthesis and verification.* (release 51205) `http://www.eecs.berkeley.edu/ alanmi/abc/`

[3] R. Brayton and A. Mishchenko. Scalable sequential verification. *ERL Technical Report*, EECS Dept., UC Berkeley, 2007.

[4] G. De Micheli. Synchronous logic synthesis: algorithms for cycle-time minimization. *IEEE Trans. on Computer-Aided Design*, vol. 10, pp.63–73, Jan. 1991.

[5] C. A. J. van Eijk. Sequential equivalence checking based on structural similarities. *IEEE Trans. Computer-Aided Design*, pp. 814–819, July 2000.

[6] C. A. J. van Eijk and J. A. G. Jess. Detection of equivalent state variables in finite state machine verification. In *Proc. IWLS*, 1995.

[7] N. Eén and N. Söensson. An extensible SAT-solver. In *Proc. SAT*, 2003.

[8] N. Eén and N. Söensson. Temporal induction by incremental SAT solving. In *Proc. BMC*, 2003.

[9] S.-Y. Huang, K.-T. Cheng and K.-C. Chen. On verifying the correctness of retimed circuits. In *Proc. GLSVLSI*, pp. 277–281, 1996.

[10] J.-H. R. Jiang and R. Brayton. Functional dependency for verification reduction. In *Proc. CAV*, pp. 268–280, 2004.

[11] J.-H. R. Jiang and R. Brayton. Retiming and resynthesis: A complexity perspective. *IEEE Trans. on Computer-Aided Design* , vol. 25, no. 12, pp. 2674–2686, Dec. 2006.

[12] F. Lu and K.-T. Cheng. IChecker: An efficient checker for inductive invariants. In *Proc. HLDVT*, 2007.

[13] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, vol. 6, pp. 5–35, 1991.

[14] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman. Exploiting suspected redundancy without proving it. In *Proc. DAC*, 2005.

[15] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Eén. Improvements to combinational equivalence checking. In *Proc. ICCAD*, pp. 836-843, 2006.

[16] M. Mneimneh and K. Sakallah. Reverse: Efficient sequential verification of retiming. In *Proc. IWLS*, 2003.

[17] S. Malik, E. M. Sentovich, R. K. Brayton, A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimization of sequential networks with combinational techniques. *IEEE Trans. Computer-Aided Design*, vol. 10, pp.74–84, Jan. 1991.

[18] N. Shenoy, K. Singh, R. Brayton, and A. Sangiovanni-Vincentelli. On the temporal equivalence of sequential circuits. In *Proc. DAC*, pp. 405–409, 1992.

[19] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *Proc. FMCAD*, 2000.

Table 1: Equivalence Checking

circuit	#node old/new	#reg old/new	time (sec)	mem (Mb)	#ite (#tf)
Retiming					
s820	388/426	5/21	1.42	14.7	5(1)
s832	399/437	5/20	1.64	14.6	5(1)
s838.1	404/492	32/117	15.48	14.6	43(1)
s953	416/432	29/45	0.91	14.1	1(1)
s967	437/447	29/39	0.86	13.6	1(1)
s991	399/470	19/90	1.03	14.3	1(1)
s1196	524/526	18/20	1.47	13.9	1(1)
s1238	579/581	18/20	2.28	14.7	1(1)
s1269	541/612	37/108	1.79	14.6	1(2)
s1423	559/598	74/113	2.22	15.0	4(1)
s1488	697/723	6/27	1.62	15.2	2(1)
s1494	707/749	6/40	4.29	15.2	4(1)
s1512	578/714	57/190	5.61	14.4	7(1)
s3271	1259/1379	116/235	3.55	15.7	1(2)
s3330	1153/1157	132/306	4.83	15.8	1(2)
s3384	1323/1296	183/156	110.18	17.3	6(2)
s4863	1917/2177	104/363	11.8	16.7	1(1)
s5378	1654/1814	179/398	88.18	16.9	13(2)
s6669	2641/3133	239/731	56.89	22.4	1(3)
s9234.1	2245/2526	211/459	722.87	19.8	23(2)
s13207.1	3572/3491	638/627	10189.86	26.1	45(1)
s15850.1	-	-	-	-	-
s35932	14032/14330	1728/2026	1707.34	39.7	20(1)
s38417	-	-	-	-	-
s38584	14143/14148	1452/1457	2807.39	49.9	17(1)
s38584.1	14169/14170	1426/1427	6692.99	43.9	13(1)
Prolog	1214/1194	136/306	3.46	15.5	1(1)
Retiming+Resynthesis					
s820	388/363	5/21	1.02	13.7	6(1)
s832	399/372	5/20	1.00	13.8	6(1)
s838.1	404/477	32/117	56.1	14.1	82(1)
s953	416/424	29/45	0.81	13.5	1(1)
s967	437/439	29/39	0.81	13.6	1(1)
s991	399/470	19/90	1.08	14.2	1(1)
s1196	524/505	18/20	1.26	13.8	1(1)
s1238	579/551	18/20	1.35	13.8	1(1)
s1269	541/589	37/108	1.62	14.2	1(2)
s1423	559/582	74/113	5.44	14.3	8(1)
s1488	697/671	6/27	1.27	14.4	2(1)
s1494	707/707	6/40	2.05	14.5	4(1)
s1512	578/677	57/190	33.45	14.7	12(1)
s3271	1259/1311	116/235	4.47	15.7	2(1)
s3330	1153/1104	132/306	5.15	15.7	1(2)
s3384	1323/1201	183/156	180.15	17.1	11(2)
s4863	1917/1898	104/363	12.58	16.2	1(1)
s5378	1654/1533	179/398	70.17	17.0	15(2)
s6669	2641/2974	239/731	251.86	23.2	1(3)
s9234.1	2245/2169	211/459	653.37	25.3	32(2)
s13207.1	3572/3013	638/627	16853.72	27.4	45(1)
s15850.1	-	-	-	-	-
s35932	14033/11060	1728/2026	17392.83	41.6	23(1)
s38417	-	-	-	-	-
s38584	14143/12184	1452/1457	5339.16	49.3	17(1)
s38584.1	14169/12211	1426/1427	7919.02	40.5	14(1)
Prolog	1214/1126	136/306	4.21	15.4	1(1)
Retiming+Resynthesis+Retiming					
s820	388/339	5/16	4.43	13.8	8(1)
s832	399/356	5/29	5.17	13.8	8(1)
s838.1	404/381	32/68	179.95	14.9	154(1)
s953	416/426	29/52	1.34	13.7	1(1)
s967	437/440	29/40	1.43	13.7	1(1)
s991	399/401	19/21	1.18	14.5	1(1)
s1196	524/498	18/19	1.34	13.8	1(1)
s1238	579/540	18/19	1.38	13.8	1(1)
s1269	541/563	37/85	1.42	14.6	1(1)
s1423	559/559	74/90	19.04	14.6	12(1)
s1488	697/664	6/30	5.19	14.6	8(1)
s1494	707/697	6/36	5.31	14.6	8(1)
s1512	578/535	57/63	261.89	16.1	23(1)
s3271	1259/1306	116/230	11.82	16.2	3(2)
s3330	1153/893	132/122	5.22	15.3	1(2)
s3384	1323/1201	183/156	50.89	16.4	12(2)
s4863	1917/1705	104/221	25.02	17.4	1(2)
s5380	1654/1368	179/276	6672.95	28.8	16(6)
s6669	2645/2727	239/489	124.39	20.8	1(2)
s9234.1	2245/2169	211/459	3482.47	34.3	28(2)
s13207.1	-	-	-	-	-
s15850.1	-	-	-	-	-
s35932	-	-	-	-	-
s38417	-	-	-	-	-
s38584	-	-	-	-	-
s38584.1	-	-	-	-	-
Prolog	1214/908	136/148	4.73	15.0	1(1)