Complete-k-Distinguishability for Retiming and Resynthesis Equivalence Checking Without Restricting Synthesis[*]

Nikolaos Liveris
Northwestern University
Evanston, IL 60208

Hai Zhou
Fudan University, China
Northwestern University, USA

Prithviraj Banerjee
HP Labs
Palo Alto, CA 94301

Abstract

Iterative retiming and resynthesis is a powerful way to optimize sequential circuits but its massive adoption has been hampered by the hardness of verification. This paper tackles the problem of retiming and resynthesis equivalence checking on a pair of circuits. For this purpose we define the Complete-k-Distinguishability (C-k-D) property for any natural number k based on C-1-D. We show how the equivalence checking problem can be simplified if the circuits satisfy this property and prove that the method is complete for any number of retiming and resynthesis steps. We also provide a way to enforce C-k-D on the circuits without restricting the optimization power of retiming and resynthesis or increasing their complexity. Experimental results demonstrate that enforcing C-k-D property can speed up the verification process.

1 Introduction

A powerful optimization technique for sequential circuits is a sequence of retiming and resynthesis operations [11] (*RnR sequence*). Resynthesis is a combinational transformation that can be applied to blocks of logic between registers. Retiming is a sequential transformation that moves registers across gates to expose new logic blocks, giving resynthesis new opportunities for optimization. The optimization power of the RnR sequence has been discussed in many works [15]. Despite its optimization power, the RnR sequence is not widely used due to the complexity of checking sequential equivalence [7] between the initial and final design. There is a need, therefore, for efficient verification methods that preserve the optimization power of the retiming and resynthesis sequence.

Van Eijk developed an efficient method for checking sequential equivalence based on finding equivalent signals in the two circuits [14]. Jiang et. al. showed that the method is complete for sequences of retiming and resynthesis transformations with at most one resynthesis step [8]. If more than one resynthesis step has been applied and the verification procedure shows that the outputs are not equivalent, no conclusion can be drawn.

Ashar *et al.* [1] demonstrated that circuits with the *Complete-1-Distinguishability (C-1-D)* property can be verified with an efficient and complete method. In a C-1-D circuit, each pair of distinct states produces different output values for some input and, therefore, each state is distinguishable from any other in a single cycle. If one of the two circuits satisfies the C-1-D property, sequential equivalence checking can be reduced to combinational equivalence checking. Not all circuits satisfy C-1-D, thus the authors developed a method to enforce this property by modifying the structure of the circuit. However, a side effect of the modifications is that the optimization power of retiming and resynthesis is reduced.

A complete method to check sequential equivalence of two circuits without restrictions on the synthesis part is reachability analysis [5]. Starting with the initial state of the circuits a forward traversal of the state space can be performed to check whether a "bad state", i.e., a state where the two circuits are not equivalent, can be reached. During each iteration, the method uses the next state relation to increase the set of reachable states. In backward reachability analysis, the process starts from the set of bad states and checks whether an initial state is reachable using the inverse of the next state relation. The number of iterations required to produce a useful answer is generally large and hard to bound. When the set of reachable states does not converge after some number of iterations, no conclusion can be drawn for the correctness of the transformations.

To improve the efficiency of reachability analysis and reduce the number of iterations without destroying completeness, a number of structural optimizations have been proposed [9, 6]. For example, retiming can be used to reduce the number of registers before the traversal starts. These techniques can be used in conjunction with the ideas proposed in this paper.

The approach described in this paper targets the equivalence checking of a pair of circuits, one of which has been obtained from the other by a sequence of retiming and resynthesis transformations. We extend the C-1-D property to C-k-D, where $k \in \mathbb{N}$. A circuit fulfills the C-k-D property if every two non-equivalent states can be distinguished within k clock cycles. The contributions of this paper are the following:

1. We show how to check sequential equivalence for circuits with C-k-D property by unrolling the product circuit a bounded number of times. This verification technique is complete; if it fails then the circuits are not RnR equivalent.
2. We provide an approach to modifying a circuit to be C-k-D.
3. We design a method for applying retiming and resynthesis transformations to the modified circuit, so that the optimization power is not restricted. Moreover, the complexity of retiming and resynthesis is not increased by the modification.

The modification of the circuit to enforce the C-k-D property is similar to target enlargement [2]. However, the method we propose is applicable even when BDD construction cannot be completed. Moreover, it is a structural transformation applied before synthesis, and it improves the verification running-time without affecting the synthesis result.

2 Problem Formulation

Two formalisms, namely netlists and finite state machines (FSMs), are used for representing circuits. Netlists are structural and consist of an interconnection of gates and registers. Finite state machines are behavioral and specify how the system changes its states and produces outputs responding to inputs.

A *netlist* is a directed graph, where the nodes correspond to elementary circuit elements, and the edges correspond to wires connecting these elements. The three basic circuit elements are *primary inputs, registers*, and *gates*. Primary input nodes have no fanins; registers have a single input. Associated with a gate g on n-inputs $w_1, w_2, \ldots, w_n$ is a function from B^n to B, where $B = \{0, 1\}$. Some nodes are designated as *primary outputs*.

A *Finite State Machine (FSM)* is a quintuple $(\Sigma, I, O, \lambda, \delta)$ where Σ is a finite set of *states*, I and O are finite sets of *inputs* and *outputs* respectively, $\delta : \Sigma \times I \rightarrow \Sigma$ gives the *next-state function*, and $\lambda : \Sigma \times I \rightarrow O$ the *output function.*

The output and next-state functions can be extended transitively to the domains $\Sigma \times I^+ \rightarrow O^+$ and $\Sigma \times I^+ \rightarrow \Sigma$, respectively, where

[*]Supported by NSF under CNS-0613967.

A^+ is the set of finite non-empty sequences of elements in set A. We continue to use λ and δ to denote these extended functions.

Two circuits are called *compatible* if they have the same set of primary inputs and primary outputs. For two compatible circuits C_a and C_b the product circuit $C_x = C_a \times C_b$ is defined. The netlist of the product circuit is built by joining the corresponding primary inputs and connecting the corresponding outputs to xor-gates forming the outputs of the product circuit. These outputs are all zero if and only if the outputs of the two circuits are equal.

A *State Transition Graph* (STG) of the circuit can be built for its FSM. The STG has one node for each state and its edges represent the transitions between states in one clock cycle. The longest path between any two nodes of the STG is called the *diameter* of the circuit.

For any circuit element o, we denote as $o^{(x)}$ the value of the element after x clock cycles. For a predicate ϕ over circuit elements (e.g., registers, inputs), $\phi^{(x)}$ is obtained by replacing each circuit element by its value after x clock cycles.

Retiming and resynthesis are structural operations on netlists. Retiming consists of moving a given number of registers between the inputs and outputs of each combinational node [10]. A retiming can be described mathematically by a lag function, which gives for each combinational node, the number of registers that are moved from fanout to fanin. Resynthesis restructures the netlist within register boundaries without changing its functionality. It leaves the FSM of the design unchanged.

Retiming becomes very powerful when interspersed with resynthesis of the netlist within the changing register boundaries. This is the basis for the retiming and resynthesis (RnR) paradigm proposed in [11].

A state s of a circuit is *dangling*, if and only if either it has no predecessor state or all its predecessor states are dangling [7]. Let D represent the set of dangling states. The states in $\Sigma - D$ are called *non-dangling*. Two states s_a and s_b are *equivalent*, denoted as $s_a \approx s_b$, if and only if

$$\forall i \in I^+ : \lambda_a(s_a, i) = \lambda_b(s_b, i)$$

Two compatible circuits with a designated initial state are *sequentially equivalent*, if and only if their initial states are equivalent. Two compatible circuits are *RnR equivalent* , if and only if one can be obtained from the other by a sequence of retiming and resynthesis transformations.

Problem 1 (Complete RnR Equivalence Checking) *Given two circuits, check whether they are sequentially equivalent or one cannot be obtained from the other by a sequence of retiming and resynthesis operations.*

For the rest of this paper we will refer to the Complete RnR Equivalence Checking Problem as RnR checking for simplicity. Obviously, this problem can be solved by model checking. However, we are interested in forcing a bound on the number of iterations to give a solution to the RnR checking problem. This bound should be forced without restricting the optimization power of the retiming and resynthesis sequence.

3 State Characterization via Output Equivalence

Given two compatible circuits C_a and C_b, our first step for proving their equivalence is to build the product circuit C_x. In this section we show how to explore the structure of C_x starting from the outputs to derive a relation between the registers of C_a and C_b (Section 3.1). Then we define C-k-D and show how it is related to the characteristic predicate of the relation between the registers (Section 3.2). This predicate is used to derive an invariant for C_x. By modifying the structure of the circuit, we can enforce C-k-D (Section 3.3).

3.1 Output Equivalence

Every output of the product circuit C_x has the value 0 in all reachable states if and only if C_a and C_b are sequentially equivalent. By construction of C_x, this is equivalent to corresponding outputs o_a and o_b being equal in all those states. Outputs o_a and o_b can be expressed as functions of the registers and inputs that drive them. The function can be extracted by traversing the netlist backwards from the outputs until a register or a primary input is met. Since the inputs cannot be restricted, this is a relation over the registers of the two circuits that must hold for all input values. More precisely, let λ_a, λ_b be the combinational functions that give the values of o_a and o_b, and $R_a = \{p_1, p_2, ..., p_m\}$, $R_b = \{q_1, q_2, ..., q_n\}$, I_a, I_b be the set of registers and inputs that are connected to o_a and o_b by a combinational path, then

$$o_a = o_b \quad \Leftrightarrow \quad \forall i \in I : \lambda_a(p_1, p_2, ..., p_m, i) = \lambda_b(q_1, q_2, ..., q_n, i) \qquad (1)$$

where $I = I_a \cup I_b$.

We define as χ_0 the characteristic predicate over the registers in $R_a \cup R_b$ that are connected to the outputs by a path of 0 registers, i.e.,

$$\chi_0 \overset{\Delta}{=} \forall i \in I : \lambda_a(p_1, p_2, ..., p_m, i) = \lambda_b(q_1, q_2, ..., q_n, i)$$

Predicate χ_0 characterizes states of the product circuit that have an output value of 0 for any input.

The next state function δ is composed of a number of δ_i functions, each giving the value of a register in the next state as a function of the registers and the input. Function δ_{r_i} of register r_i can be extracted from the circuit by traversing the signal that drives r_i backwards until a register or an input is met. By replacing the registers of (1) with the functions giving their value in the next state, predicate χ_1 is generated.

Predicate χ_1 defines a relation between the register values of the two circuits that are connected by a path of exactly one register to the outputs o_a and o_b. The predicate uses the inputs connected to those outputs by a path of 0 or 1 register to determine which states of the product circuit cause $o_a = o_b$ in the next cycle for any input, i.e.,

$$\chi_1 \Rightarrow o_a^{(1)} = o_b^{(1)} \Leftrightarrow \chi_0^{(1)}$$

As an example, consider the two circuits shown in Figure 1. For those circuits we have

$$\begin{aligned} \chi_0 &= (p1 \vee p2) \leftrightarrow (q1 \wedge q2) \\ \chi_1 &= \forall i \in \{0,1\} : ((i \wedge p4) \vee p3) \leftrightarrow \left((i \wedge q_4) \vee \overline{q3}\right) \end{aligned}$$

States $s_a =< p1 = 0, p2 = 0, p3 = 0, p4 = 1 >$ and $s_b =< q1 = 1, q2 = 0, q3 = 1, q4 = 0 >$ satisfy χ_0. Therefore, the output of the product circuit is zero in state (s_a, s_b). However, (s_a, s_b) does not satisfy χ_1. For input $i = 1$ the next state of the product circuit violates χ_0 and the output is 1.

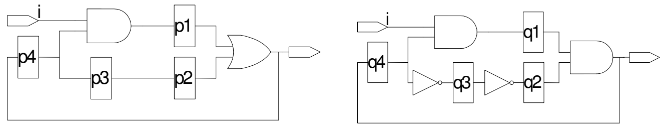

Figure 1: Two example circuits.

Similarly, by processing the circuit structure we extract χ_k from χ_{k-1} for any $k \in \mathbb{N}$. Moreover, we build the predicates in the same way for all outputs and for each $k \geq 0$ we take their conjunction. Therefore, the states that satisfy χ_k guarantee that for all xor-ed output pairs (o_a, o_b) and for all input values, $o_a = o_b$ after k cycles. By construction, it holds

$$\chi_k \Rightarrow \chi_{k-1}^{(1)} \tag{2}$$

for all $k \geq 1$. By the definition of χ_k, for a pair of states satisfying $\neg\chi_k$ there exists an input sequence of length k, such that a pair of corresponding outputs are different after k cycles.

States of the product circuit satisfying the conjunction

$$\psi_k \overset{\Delta}{=} \bigwedge_{i \in 0..k} \chi_i$$

have the same output values for the first k cycles. States of the two circuits that are equivalent must satisfy ψ_k for any k, i.e.,

$$\forall (s_a, s_b) \in \Sigma_a \times \Sigma_b \; : \; s_a \approx s_b \Rightarrow (s_a, s_b) \models \psi_k \tag{3}$$

In the worst case the evaluation of ψ_k for a state requires I^{k+1} input values. However, by the method described above only the inputs relevant to producing the output values are processed in each of the $k+1$ cycles.

3.2 Complete-k-Distinguishability

In this section we define the C-k-D property and show how we can find whether a circuit satisfies it. First, we define a property for states.

Definition 1 *A pair of states s_1 and s_2 of circuit C are k-Distinguishable if and only if there exists some input sequence i of length $m \leq k$, such that $\lambda(s_1, i) \neq \lambda(s_2, i)$.*

Definition 2 *A circuit C satisfies the C-k-D property, if and only if every pair of non-equivalent states s_1 and s_2 is k-Distinguishable.*

If we take the product C_x of C with itself, then states s_1 and s_2 are k-Distinguishable, if and only if $(s_1, s_2) \not\models \psi_{k-1}$. Therefore, circuit C has the C-k-D property, if and only if

$$\forall (s_1, s_2) \in \Sigma_x \; : \; s_1 \approx s_2 \Leftrightarrow (s_1, s_2) \models \psi_{k-1} \tag{4}$$

which follows from the definition of C-k-D and (3).

Note that when $k = 1$ our definition of C-1-D property is more general than previous approaches [1], since we require that distinguishable be non-equivalent states, not necessarily distinct states as required by [1].

By the definition of C-k-D, if any circuit satisfies the C-k-D property, it also satisfies the C-m-D property for all $m \geq k$. Therefore, C-1-D is the most restricted property of this class. More specifically, the circuits satisfying C-1-D are a subset of the circuits satisfying C-k-D for any $k \geq 1$.

Lemma 1 *For every circuit C there exists $k \in \mathbb{N}$ such that C has the C-k-D property, where k is bounded from above by the diameter of $C_x = C \times C$.*

3.3 Convergence

For verification purposes it is useful to have an invariant of the product circuit that implies correctness. Property ψ_{k-1} implies output equivalence in the current state by construction. In this section we show that for a circuit C that satisfies C-k-D, ψ_{k-1} is also an invariant of the product of C with itself (Lemma 2). Moreover, we show how we can transform the product circuit $C_x = C \times C$, so that formula (4) holds for C_x. Then in Section 4 we show how we can use ψ_{k-1} to check equivalence between the original and the transformed circuit.

Lemma 2 *If C fulfills the C-k-D property, then ψ_{k-1} is an invariant of C_x.*

Proof: From (4) we know that $(s_1, s_2) \models \psi_{k-1}$ if and only if s_1 and s_2 are equivalent. The equivalence of these states implies that for any input $i \in I$ their next states are also equivalent, and, therefore, satisfy ψ_{k-1} (Formula 3). □

As we can see from the proof of Lemma 2, it is sufficient that C_x satisfies formula (4) for ψ_{k-1} to be an invariant of C_x.

In Figure 2 a Venn diagram of the state space of the product circuit can be seen. The sets of states satisfying ψ_{k-1}, .., ψ_0 are displayed. The set of states satisfying ψ_m is a subset of the states satisfying ψ_{m-1}, as $\psi_m = \psi_{m-1} \wedge \chi_m$. In order for ψ_{k-1} not to be an invariant, there must be a state s satisfying ψ_{k-1} and having a next state s_1, such that $s_1 \not\models \psi_{k-1}$. That means that $s_1 \models \neg\chi_{k-1}$, since $s \models \bigwedge_{i \in 0..k-1} \chi_i$ implies that all states that we can reach from s in one cycle, including s_1, satisfy $\bigwedge_{i \in 0..k-2} \chi_i$, which is equivalent to ψ_{k-2}. This follows from (2). Moreover, from the fact that $s_1 \models \neg\chi_{k-1} \wedge \psi_{k-2}$ we know that the path from s to a state that violates output equivalence is exactly of length k. Each state on that path satisfies a different ψ predicate and, therefore, each state on that path is distinct.

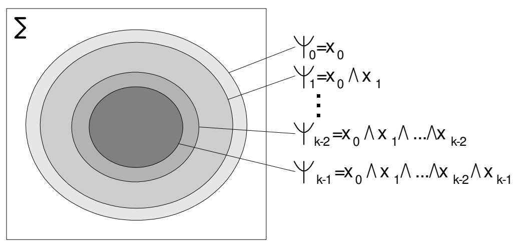

Figure 2: A Venn diagram of the states with output equivalence for the next $0 \leq m \leq k-1$ cycles.

It is possible to enforce any circuit to satisfy the C-k-D predicate, e.g., by the method described in [1]. However, this is not the only method. Methods to achieve this goal can be applied not only on C, but also on the product $C_x = C \times C$ of the circuit with itself. In Section 5 we provide a way to apply retiming and resynthesis without restrictions, which is independent of the method with which the C-k-D property was enforced.

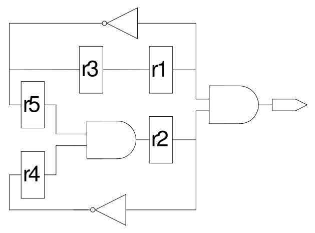

Figure 3: An example circuit.

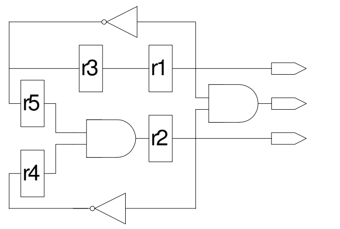

Figure 4: The circuit after the transformation.

As an example, consider the circuit of Figure 3. We take the product C_x of that circuit with itself and denote the registers of the first and second copy as $r1_a, ..., r5_a$ and $r1_b, ..., r5_b$ respectively. For C_x we have

$$\begin{aligned}
\chi_0 &= (r1_a \wedge r2_a) \leftrightarrow (r1_b \wedge r2_b) \\
\chi_1 &= (r3_a \wedge r4_a \wedge r5_a) \leftrightarrow (r3_b \wedge r4_b \wedge r5_b) \\
\chi_2 &= (\overline{r1_a} \wedge \overline{r2_a}) \leftrightarrow (\overline{r1_b} \wedge \overline{r2_b})
\end{aligned}$$

States $s_a = < r1_a = 0, r2_a = 0, r3_a = 0, r4_a = 1, r5_a = 0 >$ and $s_b = < r1_b = 1, r2_b = 0, r3_b = 0, r4_b = 1, r5_b = 0 >$ satisfy $\psi_1 = \chi_0 \wedge \chi_1$, but not χ_2. In order to enforce ψ_1 as an invariant, we create pseudo-outputs and connect to them the registers with a path of 2 registers to an existing output. The set of registers satisfying this requirement is $R_2 = \{r1, r2\}$. The new circuit can be seen in Figure 4. After simplifications, on the new product circuit we have

$$\begin{aligned}\chi_0' &= (r1_a \leftrightarrow r1_b) \wedge (r2_a \leftrightarrow r2_b)\\ \chi_1' &= (r3_a \leftrightarrow r3_b) \wedge ((r4_a \wedge r5_a) \leftrightarrow (r4_b \wedge r5_b))\end{aligned}$$

States s_a and s_b do not satisfy χ_0' and as a result they cannot satisfy $\psi_1' = \chi_0' \wedge \chi_1'$. It can be shown that ψ_1' is an invariant of the modified product circuit and, therefore, the new circuit has the C-2-D property. Next, we formally describe the transformation.

Given a $k \in \mathbb{N}$ and a circuit C we modify the product $C_x = C \times C$ of the circuit with itself, so that formula (4) holds for C_x. This has the same effect as enforcing the C-k-D property on C. Assume that C_x is the product of the circuit with itself. We build ψ_{k-1} and check whether it is an invariant. This check can be formulated as

$$\forall (s_a, s_b) \in \Sigma_x \quad : \quad (s_a, s_b) \models \psi_{k-1} \Rightarrow (s_a, s_b) \models \chi_k$$

The reason we do not need to check $\bigwedge_{j \in 0..k-1} \chi_j$ is that we know that it is satisfied by formula (2) and the definition of ψ_{k-1}. We assume that k is chosen in such a way that the above check is tractable. The above formula cannot be false unless the circuit C is not C-k-D. Then we chose $1 \le m \le k$, so that the number $|R_m|$ of registers connected by a path of m registers to an output is minimum ($m = 2$ in Figures 3, 4). We create for each register $r \in R_m$ a pseudo-output with the name o_{r_m} and connect the register to the output.

Let us denote as χ_0', ψ_{k-1}' the predicates of the circuit after the first modification. For each state (s_a, s_b) with $(s_a, s_b) \not\models \chi_m$ we know that there must be at least one register in R_m that has a different value in s_a than in s_b. By connecting those registers to outputs, when we consider χ_0', we have that $(s_a, s_b) \not\models \chi_0'$. Therefore, if we start again the process of deriving ψ_{k-1}', this time ψ_{k-1}' implies ψ_{k+m-1}. If ψ_{k-1}' fails the invariant test, we repeat the same process after identifying another number m'. Then the next predicate ψ_{k-1}'' tried implies $\psi_{k+m+m'-1}$. This process continues until ψ_{k-1} becomes an invariant. During this process the number of input variables for the computation of ψ_{k-1} remain bounded by $k \cdot |I|$ and the number of state variables are bounded by the number $|R|$ of registers in the product circuit. Since for the number m chosen in each iteration we have $m \ge 1$ and the diameter is an upper bound for k, eventually this process terminates. The advantage of this approach is that it is easy to apply as it does not require the BDD construction for ψ_{n-1} for some n. For example, if for some $n \in \mathbb{N}$ the BDD construction of ψ_{n-1} cannot be completed, by applying the transformation we can enforce ψ_{l-1} as an invariant with $l < n$. Then ψ_{l-1} could be easier to compute. The choice of k for the target invariant ψ_{k-1} should be made in such a way that ψ_{k-1} can be computed.

The modification we described is applied to the product circuit. However, it does not prevent us from obtaining the transformed circuit by applying a sequence of retiming and resynthesis operations. In Section 5 we show how we can apply these operations without restricting their optimization power or increasing their complexity.

4 RnR Equivalence Checking Under C-k-D

In this section we show how we can check the equivalence between the original and the transformed circuit when we know that the original circuit fulfills the C-k-D property. The check can be done by unrolling the product circuit a bounded number of times. First, we prove some properties for the non-dangling states of the original C_a and the transformed circuit C_b. Using these properties we show that ψ_{k-1} is an invariant of those circuits in the non-dangling state space, if C_a satisfies the C-k-D property. Based on this result we prove the completeness of our method, i.e., if the checks fail, C_b cannot be obtained from C_a using retiming and resynthesis operations. Finally, we show how to check sequential equivalence between C_a and C_b.

Lemma 3 *If the original circuit C_a is RnR equivalent to the transformed circuit C_b, then for every non-dangling state of C_b there exists an equivalent non-dangling state in the STG of C_a.*

Proof: We prove the theorem by induction. The base case, before any step of the RnR sequence, is trivial as the transformed circuit is identical to the original circuit and the STGs are isomorphic. Assume that the lemma holds after m steps of the RnR sequence, we prove that it holds after the $m+1$ step has been applied.

The first case is that the $m+1$ step is a resynthesis step. Then the STG of the transformed circuit is preserved. Therefore, for every non-dangling state of C_b, there exists an equivalent non-dangling state in the STG of C_a.

The second case is that the $m+1$ is a retiming step. Let C_b^m, C_b^{m+1} be the transformed circuits before and after the m step, respectively. Retiming does not create new non-dangling states, but merges equivalent non-dangling states or splits non-dangling states to equivalent non-dangling states [7]. Consequently, for every non-dangling state in the STG of C_b^{m+1}, there exists an equivalent non-dangling state in the STG of C_b^m. □

Lemma 4 *If the original circuit C_a has the C-k-D property and the transformed circuit C_b is RnR equivalent to C_a, then ψ_{k-1} is an invariant of the product circuit $C_{a\times b} = C_a \times C_b$ in the non-dangling state space.*

Proof: Let us assume C_a has the C-k-D property and C_b is RnR equivalent, but ψ_{k-1} is not an invariant of $C_{a\times b}$. Then there exist k distinct states (s_1, t_1), ..., (s_k, t_k) in $C_{a\times b}$ such that

$$\begin{aligned}\forall m \in 1..k-1 \quad &: \quad \exists i \in I : (\delta_a(s_m, i), \delta_b(t_m, i)) = (s_{m+1}, t_{m+1})\\ \forall m \in 1..k-1 \quad &: \quad (s_m, t_m) \models \psi_{k-m}\\ (s_k, t_k) \quad &\models \quad \neg\chi_0\end{aligned}$$

Since C_a has the C-k-D property, there is no state x such that $(s_1, x) \models \psi_{k-1}$ and $s_1 \not\approx x$. This implies that t_1 is either non-equivalent to any state of C_a or t_1 is equivalent to some state y of C_a for which $(s_1, y) \not\models \psi_{k-1}$. In the first case we have that t_1 is a dangling state or C_b is not RnR equivalent to C_a (Lemma 3), which is a contradiction. Therefore, t_1 is equivalent to y with $(s_1, y) \not\models \psi_{k-1}$. This implies that there exists $m < k-1$, such that $(s_1, y) \not\models \chi_m$. However, then there exists an input $i \in I^m$ for which $\lambda_b(t_1, i) = \lambda_a(y, i) \ne \lambda_a(s_1, i)$. Consequently, $(s_1, t_1) \not\models \psi_{k-1}$, which is a contradiction. □

Based on the lemmas above, if we are given an initial state (s_{ia}, s_{ib}) for the product circuit, we can use the following method

$$\exists s_a \in \Sigma_a, \exists i \in I^{n_d} \quad : \quad \delta_a(s_a, i) = s_{ia} \tag{5}$$

$$\exists s_b \in \Sigma_b, \exists i \in I^{n_d} \quad : \quad \delta_b(s_b, i) = s_{ib} \tag{6}$$

$$(s_{ia}, s_{ib}) \models \psi_{k-1} \tag{7}$$

$$\forall (s_a, s_b) \in \Sigma_{a\times b}, \forall i \in I^{n_d} \quad : \quad \delta(s_a, s_b, i) \models \psi_{k-1} \Rightarrow \delta(s_a, s_b, i) \models \chi_k \tag{8}$$

where n_d is the register depth of the initial circuit. The first two formulas check that the initial states of the two circuits are non-dangling. They can be posed as SAT problems. The third formula checks the initial state of the product circuit satisfies ψ_{k-1}. Finally, the last formula checks that ψ_{k-1} is an invariant in the non-dangling state space. By considering only states reachable after n_d cycles, the check is restricted to non-dangling states.

If $C_{a \times b}$ satisfies formulas (5)–(8), then C_a and C_b are sequentially equivalent. This is because the product circuit starts from non-dangling states. Then dangling states are not reachable. Moreover, ψ_{k-1} is an invariant and it implies output equivalence by construction. The following theorem shows that if C_a is a C-k-D circuit and one of the formulas does not hold, then either C_a and C_b are not RnR equivalent or the initial states include dangling states.

Theorem 1 *If C_a satisfies the C-k-D property, the initial state of the product circuit is non-dangling and either formula (7) or (8) does not hold, then C_a and C_b are not RnR equivalent.*

Proof: Since the initial state of the product circuit is non-dangling, formulas (5) and (6) must hold. Then from Lemma 4, it follows that the circuits are RnR equivalent only if (8) holds. Moreover, if (7) does not hold, either dangling states are included in the initial state set (contradiction), or the circuits are not RnR equivalent. $\square$

We believe that the assumption for (s_{ia}, s_{ib}) is reasonable. If s_{ia} is a dangling state, then the problem of finding a corresponding state after retiming may be unsolvable.

From Theorem 1 and the discussion above, we know that if (5)–(8) hold, then C_a and C_b are sequentially equivalent. If one of them does not hold, then C_b cannot have been obtained from C_a by a sequence of retiming and resynthesis transformations, i.e., C_b is not RnR equivalent to C_a. There may be a case that C_b is not RnR equivalent to C_a, but the two circuits are sequentially equivalent. In that case the checks for (7) and (8) may succeed or fail. In such a case a failure of the checks can point to an error of our RnR transformation implementation. A success is also a good result, because even though there may be a problem in the way the RnR transformation was implemented, the two circuits are sequentially equivalent. Ideally, in this case we would like to get both results. Our method gives only one of the results.

5 Retiming and Resynthesis Without Restrictions

We are given a modified product circuit C_x that has been augmented with additional logic and outputs by a method enforcing the C-k-D property such as the method of Section 3.3. Assume that predicate ψ_{k-1} is an invariant of C_x, and in C_x we can distinguish the two copies of C, namely, C_1 and C_2, and the additional logic and outputs C_3. More specifically, each node and edge of the product circuit is colored by $\{c_1, c_2, c_3\}$. A c_3 edge can be driven by a gate of any color. However, a c_1 or c_2 edge can only be driven by a node of the same color. Our purpose is to apply a sequence of retiming and resynthesis transformations on C_2, the second copy of C, without being restricted by the additional logic. Moreover, after the transformation we want the product circuit to have ψ_{m-1} as an invariant for a known m.

Before a resynthesis step we extract C_2 from C_x by considering all circuit nodes and edges marked with c_2. All c_3 edges that are driven by a c_2 node are left hanging, i.e., not driven by any node, by this transformation. We express each of these edges as a function of c_2 registers and primary inputs. Then we add logic to C_3, so that the only edges hanging, i.e., not driven by a node, are the edges that would be driven by a C_2 register or a primary input. It is easy to extract the additional logic by traversing backwards a c_2 node that drives a c_3 edge until a register or a primary input is met in each path. Then this logic is replicated and added to C_3. Then we apply resynthesis on C_2. The resynthesis optimization is unrestricted as only nodes and edges of C_2 are considered. Resynthesis does not remove registers or inputs and after the step we can bring the modified version of C_2 back in C_x by connecting c_2 nodes (registers, inputs) to the corresponding hanging c_3 edges.

Before a retiming step we extract C_2 from C_x again. We maintain a mapping between the hanging c_3 edges and the c_2 nodes that drive those edges. We retime C_2 as an independent circuit. Retiming does not change the circuit structure, so we reconnect the modified C_2 to obtain C_x by preserving the mapping. Nodes belonging to C_2 that drive c_3 edges may have a lag value r that is different than 0. In such a case, the number of registers on the c_3 edges need to be adjusted, so that the weights of the edges are consistent with the lag (r-) values.

If for a c_2 node v that drives a c_3 edge we have $r(v) < 0$, then we add to the c_3 edge $-r(v)$ number of registers. These registers have been moved across v from its fan-in, which are c_2 edges. Therefore, this move is based on pre-existing registers. Based on the results proved on Section 4, we know that after such a retiming move ψ_{k-1} should still be an invariant of the product circuit C_x in its non-dangling state space.

In the case that $r(v) > 0$ for a c_2 node v driving a c_3 edge, then we remove $r(v)$ registers from the c_3 edge. If the weight of the edge becomes negative, we try to adjust the $r(u)$ value of the head u of the edge, which is a c_3 node. This may cause other edges to have negative values. The paths from all these edges terminate at a single pseudo output o.The end result may be that we have to adjust the value of the pseudo output o. In that case the effect of the additional logic is moved in the past. Therefore, instead of ψ_{k-1} the predicate that must be an invariant of the product circuit is $\psi_{k-1+r(o)}$. Every time we retime a pseudo output, we adjust the number m, for which ψ_{m-1} must be an invariant of the modified product circuit. Using this number, at the end of the retiming and resynthesis sequence, we will derive ψ_{m-1} and require that the checks described in Section 4 succeed, in order for the two circuits to be sequentially equivalent.

The method described in this section resembles recording the transformation history of retiming and resynthesis. In [13] a similar method is presented. There are important differences between that approach and our method. First, in [13] verification relies on synthesis to record the candidate problems to be solved. If the problems are solved successfully, then the circuits are assumed equivalent. However, it is unclear whether a bug in recording synthesis history can result in a false positive. Moreover, that technique is described for a tool that uses a specific data structure to represent a logic network, namely And-Inverter-Graphs. Our technique is general in that the data passed from synthesis to verification are in the form of a circuit. Therefore, synthesis and verification can use different data structures. In terms of efficiency, our approach creates new nodes only when required for the fan-in cones of the pseudo-outputs, while the approach in [13] stores every node created during synthesis. Moreover, it stores one node for each move of a register over a gate during a retiming operation. Because of that, restrictions on the synthesis part may become necessary for the approach in [13] to be practical.

The product circuit obtained by the method of the previous section can be used for verification purposes. However, after verification we want to extract only the optimized copy C_2 of the product circuit. It is easy to see that by taking only the nodes and edges colored by c_2, we have the optimized version. The extracted C_2 circuit is the same as the circuit obtained by applying retiming and resynthesis to C_1. The reason is that during the optimization the additional edges and nodes were not considered.

6 Experimental Results

We used the ABC framework [3] and VIS [4] to test our ideas. Checking formula (8) is the most difficult step of our approach, so we focus on the implementation and results for that part. With ABC we implemented the check as a SAT problem on a miter. More specifically, the predicate ψ_{k-1} is built as a BDD after unrolling the product circuit and applying universal quantification on the inputs. The result is appended to the circuit specifying χ_k using the utility of ABC that implements a BDD as a circuit of muxes. Then we check whether for any input sequence and state $\psi_{k-1} \wedge \neg\chi_k$ is satisfiable. The approach in [2] for building the BDD for target enlargement or the SAT approach for quantification [12] could be more efficient for check-

Bench	SAT on miter with ABC				VIS: backward traversal		VIS: forward traversal	
	Without Addit. Logic		**With Addit. Logic**		**Without Addit. Logic**	**With Addit. Logic**	**Without Addit. Logic**	**With Addit. Logic**
	Exec Time (sec)	k	Exec Time (sec)	k	Exec Time (sec)	Exec Time (sec)	Exec Time (sec)	Exec Time (sec)
$s635$	4	>4	4	2	> 2000	1.2	> 2000	> 2000
$s838$	9.8	>4	6.5	2	2.9	1.0	1.2	1.0
$s938$	26.4	>4	8.6	2	>2000	1.9	> 2000	> 2000
$s953$	40.2	3	3.6	2	5.1	2.9	3.3	2.9
$s967$	9.6	3	5.6	2	3.1	5.5	1.7	2.4
$s1196$	4.0	2	3.8	1	1.2	1.1	1.0	0.9
$s1512$	>2000	>4	27.25	2	> 2000	14	> 2000	> 2000
$s3271$	>2000	>4	>2000	2	> 2000	> 2000	> 2000	> 2000

Table 1: Running-time of different verification methods with and without additional logic. In the SAT on the miter case, k is the value for which ψ_k becomes an invariant. The computation is terminated if the running-time is greater than 2000 secs, or k is greater than 4 (SAT-on-the-miter case).

ing formula (8). However, we do not expect that another method will significantly change the improvement shown in the results or the conclusions drawn by them.

Verified are a number of ISCAS benchmarks, with and without additional logic to enforce the C-k-D property. Each pair of circuits to verify are retiming and resynthesis equivalent. The results are reported in Table 1. The first two columns display the running time and the k value of the verification process without the additional logic and output. The value k is such a number that ψ_k first becomes an invariant. The next two columns give the results with additional logic and output. Except for the additional outputs and the logic driving them the pairs of circuits are in both cases the same. The circuits with and without the additional logic were then checked with VIS. We tried both backward (columns 5 and 6) and forward traversal (columns 7 and 8) traversal with VIS. The VIS-command we used on the circuits is “seq_verify” with option “-r” for variable reordering.

In most cases the running time has been substantially reduced when additional logic and output are introduced. Examples are the cases of s635, s938, and s1512. VIS does not terminate in 2000 secs without the additional logic. However, the computation of ψ_{k-1} with additional logic takes a few seconds in those cases. Enforcing ψ_{k-1} can speed up the verification in VIS, as well, when it operates in backward traversal. The additional logic does not have any significant effect for the forward traversal. For the check of ψ_{k-1} as an invariant, the speed up obtained by enforcing a small k is significant in most cases.

From the results we conclude that by using the additional logic we can significantly speedup the checking procedure in some cases. The described procedure is based on BDDs and, therefore, does not scale well compared to SAT based procedures, e.g., [8]. However, those procedures are not proved complete for retiming and resynthesis sequences with more than one resynthesis step. We conclude that enforcing the C-k-D property can simplify the verification task without restricting the synthesis part.

7 Conclusions

In this paper we extended the property of C-1-D to C-k-D and we showed how we can check circuits for equivalence if one of them satisfies the C-k-D property. We also presented a technique to enforce the C-k-D property on a circuit and then apply a sequence of retiming and resynthesis transformations without restricting their optimization power or increasing their complexity. Our method provides a bound to the number of timeframes that need to processed during verification and is complete in the sense that any result provides useful information. Our experimental results show that enforcing the C-k-D property can speed up the verification process.

References

[1] Ashar, P., Gupta, A., and Malik, S. Using complete-1-distinguishability for FSM equivalence checking. *ACM Trans. Des. Autom. Electron. Syst. 6*, 4 (2001), 569–590.

[2] Baumgartner, J., Kuehlmann, A., and Abraham, J. A. Property checking via structural analysis. In *CAV ’02: Proceedings of the 14th International Conference on Computer Aided Verification* (London, UK, 2002), Springer-Verlag, pp. 151–165.

[3] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, Release, 70930,http://www.eecs.berkeley.edu/alanmi/abc/.

[4] Brayton, R. K., Hachtel, G. D., Sangiovanni-Vincentelli, A. L., Somenzi, F., and et. al., A. A. Vis: A system for verification and synthesis. In *CAV ’96: Proceedings of the 8th International Conference on Computer Aided Verification* (London, UK, 1996), Springer-Verlag, pp. 428–432.

[5] Clarke, E. M., Grumberg, O., and Peled, D. A. *Model Checking*. The MIT Press, 1999.

[6] Hasteer, G., Mathur, A., and Banerjee, P. Efficient equivalence checking of multi-phase designs using retiming. *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on* (8-12 Nov 1998), 557–562.

[7] Jiang, J.-H. R., and Brayton, R. K. Retiming and resynthesis: A complexity perspective. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 25*, 12 (Dec. 2006), 2674–2686.

[8] Jiang, J.-H. R., and Hung, W.-L. Inductive equivalence checking under retiming and resynthesis. In *ICCAD ’07: Proceedings of the International Conference on Computer-Aided Design (ICCAD’07)* (2007).

[9] Kuehlmann, A., and Baumgartner, J. Transformation-based verification using generalized retiming. In *CAV ’01: Proceedings of the International Conference on Computer Aided Verification* (London, UK, 2001), Springer-Verlag, pp. 104–117.

[10] Leiserson, C. E., and Saxe, J. B. Retiming synchronous circuitry. *Algorithmica 6*, 1 (1991), 5–35.

[11] Malik, S., Sentovich, E., Brayton, R., and Sangiovanni-Vincentelli, A. Retiming and resynthesis: optimizing sequential networks with combinational techniques. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 10*, 1 (Jan 1991), 74–84.

[12] McMillan, K. L. Applying sat methods in unbounded symbolic model checking. In *CAV ’02: Proceedings of the 14th International Conference on Computer Aided Verification* (London, UK, 2002), Springer-Verlag, pp. 250–264.

[13] Mishchenko, A., and Brayton, R. K. Recording synthesis history for sequential verification. In *IWLS* (2008).

[14] van Eijk, C. A. J. Sequential equivalence checking based on structural similarities. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 19*, 7 (2000), 814–819.

[15] Zhou, H., Singhal, V., and Aziz, A. How powerful is retiming? In *IWLS* (1998).