Combinational Techniques for Sequential Equivalence Checking

Hamid Savoj[1] **David Berthelot**[1] **Alan Mishchenko**[2] **Robert Brayton**[2]

Envis Corporation[1] and Department of EECS, University of California, Berkeley[2]

{hamid, david}@envis.com and {alanmi, brayton}@eecs.berkeley.edu

Abstract

Often sequential logic synthesis can lead to substantially easier verification problems, compared to the general-case for sequential equivalence checking (SEC). We prove some general theorems about when SEC can be reduced to combinational equivalence checking (CEC). These can be applied to many sequential clock gating transforms, where correctness is argued intuitively using a finite unrolling of a sequential design. A method based on these theorems was applied to six large industrial examples. It completed on all examples and was about 30x faster on the three examples where the conventional engine was able to finish.

1 Introduction

To motivate this work, consider a sequential circuit, *A*, which is to be optimized by a *k*-step unrolling process; then combinational synthesis is applied to the first frame while the other *k*-1 frames are left untouched. This synthesis is done so that no difference between the two circuits is observed i.e. neither at the POs of each of the *k* frames nor at the flip-flop (FF) inputs of the final frame (see Figure 1). The last *k*-1 copies of *A* are used only to produce "ODCs" for transforming the combinational part of *A* into the combinational part of a new sequential circuit *B*.

Several questions arise in similar types of synthesis:

1. Is the derived circuit *B* *sequentially* equivalent to *A*? This is not obvious because it is the *k*-1 copies of *A* that provide the observability don't cares (ODCs), for *B*, and not *B* producing those ODCs as would be the case during the sequential operation of machine *B*. Although there is a known 1-1 correspondence between FFs in *A* and *B*, their state-transition functions are not necessarily the same.
2. Suppose *A* is unrolled *n* times and the *last* copy of *A* is synthesized using satisfiability don't cares (SDCs) provided by the first *n*-1 copies of *A*. Are *A* and *B* sequentially equivalent? As in Question 1, this is not obvious.
3. More generally, suppose *A* is unrolled *n* + *k* times and the *n*th copy of *A* is synthesized, using both ODCs and SDCs, to produce machine *B*. Is *B* sequentially equivalent to *A*? This is not only not obvious, but generally incorrect.

In Section 2, we answer these questions, affirmatively for the first two with Theorems 1 and 2, and give a counterexample for the last one. The theorems are stated for general SEC and give sufficient conditions when it can be solved by a CEC method. Theorem 1 might be expected to apply when a synthesis transform can be argued from non-observability principles and Theorem 2 when non-controllability is used. In Section 3, we discuss relevant literature and related parallels to the results obtained in this paper, and in Section 4, we give some experimental results illustrating how these methods can make sequential equivalence checking (SEC) much more effective and practical on certain types of problems. Section 5 summarizes and poses some open questions for future research.

2 Sequential Equivalence

Let *A* be a sequential circuit and A^1 denote the combinational part of *A*. Let A^n denote the *combinational* circuit obtained by connecting *n* copies of A^1 at the FF inputs and outputs. The outputs of A^n are the set of *n* POs of *A* one for each time frame plus the final FF input signals after the *n*th frame. The inputs of A^n are the set of *n* PIs of each frame, plus the initial FF output signals at the start of the first frame.

Let *A* and *B* be two sequential circuits with the same PIs and POs, and the same number of FFs. (B^n, A^k) denotes the combinational circuit where the outputs corresponding to the final FF inputs of B^n are connected to the inputs corresponding to the initial FF outputs of A^n . The connection is done using some 1-1 mapping between the FF of *A* and *B*. We overload notation by dropping the superscript in A^1 when the context is clear, as in (B, A^k) .

In this paper, it is always assumed that a single initial state is given for a sequential machine. We are not concerned with initializing sequences etc., but follow the philosophy articulated in [1]. Thus two machines are considered sequentially equivalent if starting at their respective initial states they produce the same sequence of POs for any sequence of PIs. This is usually equivalence checked by forming a miter (which creates a single output formed by ORing XORs of corresponding POs) of the two circuits to obtain one machine with a single output. Then it is to be proved that the output is always 0 for all time if the miter machine is started in the initial state given by the initial states of the two machines.

For two sequential circuits, *A* = *B* denotes that the circuits are sequentially equivalent starting from the two given initial states. If *C* and *D* are combinational circuits, the *C* = *D* means that they are combinationally equivalent, i.e for any input, their outputs match.

The first question in Section 1 concerns equivalence of two related *combinational* circuits, i.e. does $A^k = (B, A^{k-1})$ imply *A* = *B*? This is depicted in Figure 1 where *k* = 3. We emphasize that to create the related combinational circuit (B, A^{k-1}) from A^1 and B^1, it is necessary that there is a 1-1 correspondence between the FFs of *A* and *B*. In some applications, this can be relaxed by inserting dummy FFs in one of the circuits.

Theorem 1: Suppose two sequential circuits *A* and *B* have the same PIs and POs. Using some 1-1 mapping between the FF of *A* and *B* to form (B^n, A^k)**,** suppose that $(B^n, A^k) = A^{n+k}$**.** Then *A* = *B*, for *any* common initial state.

Note that *A* and *B* are initialized with the same initial state.

Proof:[1] Assume that $(B^n, A^k) = A^{n+k}$. Consider the following infinite sequence of lines.

$$\begin{array}{l}
\qquad\quad \to\to \text{ time frame number } \to\to \\
\xrightarrow{S*} A^n \xrightarrow{S_0(n)} A^n \xrightarrow{S_0(2n)} A^n \cdots \\
\xrightarrow{S*} B^n \xrightarrow{S_1(n)} A^n \xrightarrow{S_1(2n)} A^n \xrightarrow{S_1(3n)} \cdots \\
\xrightarrow{S*} B^n \xrightarrow{S_2(n)} B^n \xrightarrow{S_2(2n)} A^n \xrightarrow{S_2(3n)} \cdots \\
\qquad\qquad\qquad \vdots \\
\xrightarrow{S*} B^n \xrightarrow{S_j(n)} B^n \xrightarrow{S_j(2n)} B^n \xrightarrow{S_j(3n)} B^n \cdots
\end{array}$$

It is assumed that all lines at each time-frame receive the same sequence of common PI inputs. Denote the POs of line j by $PO_j(t)$, where $t \in \{1,2,3,\cdots\}$ and $j \in \{0,1,2,3,\cdots\}$. Similarly for the states; $S_j(t)$ denotes the state of line j at time t, $t \in \{0,1,2,3,\cdots\}$ where $S_j(0) = S*$, the set of all states. Since $(B^n, A^k) = A^{n+k}$, then $PO_0(t) = PO_1(t)$ for $t \in \{1,2,3,\cdots,n+k\}$. Note that for all $t > n+k$, this is also true because $S_0(n+k) = S_1(n+k)$ and the circuit copies in both lines are A from then on. Now compare lines 1 and 2. Clearly $PO_1(t) = PO_2(t), \quad t = 1,...,n$ since in both lines, the inputs are to n copies of B, and by using the template, $(B^n, A^k) = A^{n+k}$, but applying it starting at the end of frame n, we have $PO_1(t) = PO_2(t)$ for all t, by the same argument that established that $PO_0(t) = PO_1(t)$ for all t. Thus by transitivity, $PO_0(t) = PO_2(t)$. Continuing, we get $PO_0(t) = PO_j(t)$ for all lines $j \in \{1,2,3,\cdots\}$. Thus line 0 and line ∞ always produce the same sequence of POs for all time no matter what is the initial state. Since the miter for $A \oplus B$ is proven to be UNSAT, we have $A = B$. **QED**.

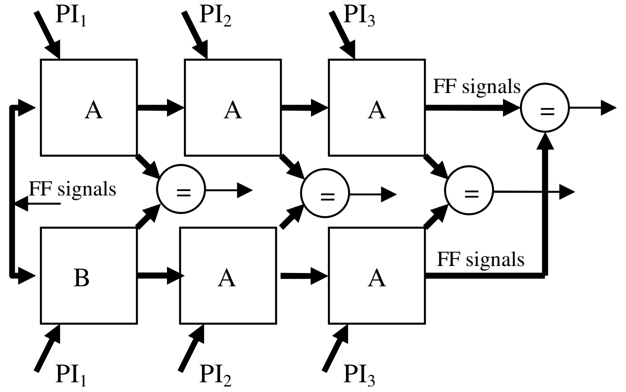

Figure 1. SEC by unrolling and CEC. POs are compared at each time frame as well as FF inputs *after* the last frame.

Note that nothing is assumed about how B derived. Also, if $(B^n, A^k) \neq A^{n+k}$, one can still try to prove $A = B$ by increasing k or n, and a false negative may go away.

[1] It has been suggested by several people (including one reviewer) that the theorems of this paper can be proved more elegantly by induction. However, we prefer the more graphical proofs (which are basically induction).

Note also that no initial state information was used in proving this theorem, i.e. $S_j(0) = S*$ is the set of all states. However, we could use a subset $\hat{S} \subset S*$ as long it is guaranteed that $S_0(n), S_1(2n), S_2(3n), \cdots$ are all **subsets** of $\hat{S}$. Thus a corollary of the theorem would be that $[(B^n, A^k) = A^{n+k}]_{\hat{S}} \Rightarrow A = B$ for any initial state $s \in \hat{S}$, where $[(B^n, A^k) = A^{n+k}]_{\hat{S}}$ denotes that combinational equivalence need only hold on state inputs in $\hat{S}$.

A variation of Theorem 1 states that SEC holds after n cycles of A.

Theorem 2: $[(A^n, B^k) = A^{n+k}] \Rightarrow A = B$ *on any initial state chosen from the subset of states that can be reached by A after n cycles, denoted* S_n^A.

Proof: We use the fact that the state space is finite, and therefore its diameter, D, is bounded. Thus after D time frames, every possible state has been seen under all possible inputs. The proof is similar to that of Theorem 1, except we proceed backwards from time-frame $T = D + (k\lceil D \div k \rceil)$. We first apply the template, $(A^n, B^k) = A^{n+k}$, to insert k B's just before $t = T$. This is iterated $\lceil D \div k \rceil$ times until we arrive at a line with n A's followed by all B's up to $t = T$. At each iteration j, we are assured that $PO_1(t) = PO_j(t)$ for all time. At this point we know that all states and all PIs for these states have been seen and for all of these the PO's agree. Thus starting at any state in S_n^A, $A = B$. **QED.**

To illustrate the need to start only on the states reachable by an initial sequence of A's, consider the example of Figure 2 (a bubble at an input to a gate denotes inversion).

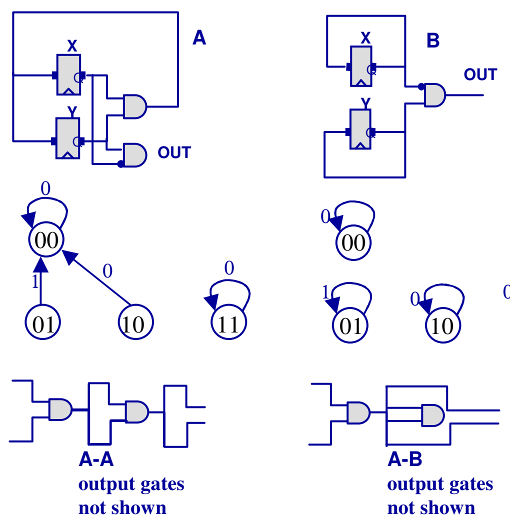

Figure 2. $A, B = A^2$, but sequential equivalence occurs only after one cycle of A.

It is easy to check that $(A,A) = (A,B)$ from the STGs shown, but $A \neq B$. The counterexample is that if A and B start in State 01, the PO sequences for A and B are not the same. However, note that starting from any state that can be reached after one clock cycle of A (i.e. States 00 and 11), then $A = B$.

The first theorem is essentially an observability theorem and the second a controllability theorem. One might conjecture that analgous combined controllability and observability theorems hold. Indeed we have the following.

Theorem 3: $[(B,A,A)=(B,B,A)] \Rightarrow A=B$ *on any initial state chosen from the subset of states that can be reached by B after one cycle, denoted* S_1^B.

Proof: Consider the sequence of transformations shown below.

$$\begin{array}{l} B,A,A,A,A,A,\cdots \\ B,B,A,A,A,A,\cdots \\ B,B,B,A,A,A,\cdots \\ B,B,B,B,A,A,\cdots \\ \cdots \end{array}$$

Each new line is obtained by using $(B,A^2)=(B^2,A)$. At each line note that $PO_j(t)=PO_0(t), \forall t$. Thus after the first time frame, the set of states that can exist is S_1^B and after that, we have

$$A,A,A,A,A,A,\cdots = B,B,B,B,B,B,\cdots$$

Thus, $A=B$ on S_1^B. **QED**

Note that in Figure 2, $(B^2,A) \neq (B,A^2)$, which can be seen by starting at State (01), so it is not a counterexample to Theorem 3.

One could consider this as a *B*-controllable, *A*-observable theorem and the first two theorems as *A*-observable and *A*-controllable theorems respectively. What about an *A*-controllable, *A*-observable theorem, where we consider $(A,B,A)=(A,A,A)$?

Such a result does not hold. Consider the STG example shown in Figure 3, which has no inputs; the label on the edges denotes the output value. Although $(A,A,A)=(A,B,A)$, one can check that $A \neq B$, even on the states that *A* can reach after one cycle, e.g. starting at State 01 *A* 110… and *B* outputs 111….

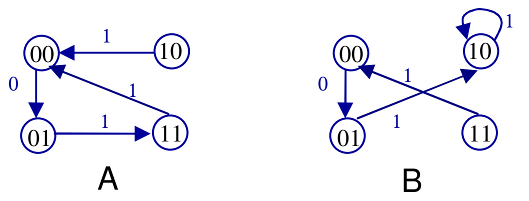

Figure 3. Although $(A,A,A)=(A,B,A)$ combinationally, *A* and *B* are not sequentially equivalent.

Although such a theorem does not hold, it still might be useful to synthesize (A,A,A) into (A,B,A) to derive a new sequential machine *B*. This is easier to do than obtaining a new machine *B*, for example, by synthesizing (A,A,A) into (B,B,A) or (A,B,B). These twi cases can guarantee equivalence using Theeorems 1 and 2 respectively. However, it is possible in the synthesis into (A,B,A), that the SDC or ODC don't cares actually used would be produced also by *B*. We can try to check $A^3=(B,A,A)$ or $A^3=(A,B,B)$ using Theorems 1 or 2, or $(B,A,A)=(B,B,A)$ or $(A,A,B)=(A,B,B)$ using Theorem 3. If any of these cases hold, then $A=B$. For the last three checks, it needs to be checked also that the initial state is in the appropriate subspace.

3 Relations to Previous Work

One of the pragmatic aspects of sequential synthesis is that it is insufficient to provide synthesis software, which may even use formally-proved[2] transforms because the software that embodies these may have bugs. Even if the software has withstood the test of time having been applied to many examples, most companies insist on formally verifying the result against the original design. Equivalence checking of combinational netlists (CEC) is practical for most industrial designs and, partly because of this, combinational synthesis is readily accepted. Also, resolution proofs [9] can be used for CEC.[3]

However, the PSPACE-complete complexity of SEC often discourages the use of sequential synthesis. In special cases, the complexity of SEC is simpler, e.g. if synthesis is restricted to one set of combinational transformations followed by one retiming (a sequential synthesis step) or vice versa, the problem is provably simpler - *only* NP-complete. If retiming and resynthesis are iterated, the problem is PSPACE complete [3]. Like CEC, SEC becomes simpler in practice if there are many structural or functional similarities (cut-points) between the two circuits being compared.

There are instances where SEC can be transformed into a CEC problem on which today's commercial CEC engines usually can be successful, even on very large problems. One is where *sequential signal* equivalences (signals that are equivalent on the reachable state set) are derived using induction [5] and used in the synthesis process. If equivalence checking is done immediately after this *without* other transformations intervening, SEC can be proved by CEC methods.

Another example is where a history of synthesis is recorded as a redundant sequential circuit [6]. In most cases, this history circuit provides a set of intermediate equivalences, which can be proved inductively, and these are enough prove SEC . Also, the concept of speculative reduction [7] can be used to make the equivalence checking problem even easier in this case.

Several papers have used an (explicitly or implicitly) unrolled version of the circuit to derive redundancies for synthesizing an improved sequential circuit. These papers do not address the formal SEC of the synthesized result. All deal with the case where the redundancies derived are independent of any initial state, similar to the theorems in the present paper. These types of results come mostly from the testing community, where a signal is redundant if the good and faulty (with a *stuck-at* fault inserted) machines can not be distinguished for any initial state.

There is a subtle distinction between *untestable* faults and *redundant* faults. If s_f and *s* are the initial states of the faulty and good machines respectively, then a fault is *untestable* if $\forall(I)\exists(s,s_f)[Z(I,s)=Z^f(I,s_f)]$ and it is *redundant* if $\forall(I,s_f)\exists(s)[Z(I,s)=Z^f(I,s_f)]$. $Z(I,s)$ is the trace, starting at state *s*, of POs under the sequence *I* of PI inputs. Using redundancy in synthesis means that when the good machine is replaced with the "faulty" (redundancy removed) machine, no difference can be observed externally because no matter what state s_f the faulty machine starts in, there is an equivalent state in which the good machine could have started in. Such a replacement is *safe*[4] [10] and *compositional*. In contrast, if the fault is merely untestable, then there could exist a *pair* of states in which the two machines could start, such that the difference between the two machines could not observed. However, there

[2] There are cases where "proved" methods in the literature have been shown to have counterexamples.

[3] We know of no similar capability for SEC.

[4] A safe replacement is one for which there is no possibility of externally detecting any difference from the original.

could be a state in either machine which has no equivalent in the other, and if one of the machines happened to start in such a state, the two machines would have different observable behaviors. Such a (untestable) replacement is not safe and is not compositional, and its use in synthesis is problematic. A good discussion on the difference between undetectable faults and redundant faults is in [3].

From Theorem 1, if $(B^n, A^k) = A^{n+k}$, then the synthesized circuit is a safe replacement for the original one. Safe replacements are useful because safety implies that every synchronization sequence for the original design also synchronizes the replacement. This is often desired because it is not necessary to re-derive a new synchronizing sequence for initializing the synthesized machine.

A useful notion is *c*-cycle redundancy [2] where the two circuits' outputs need not match for the first *c* cycles after power-up. This allows more flexibility in synthesizing a circuit because the behavior of the machine need only be preserved on states that can be reached after *c* cycles as long as initialization is preserved. Several papers make use of this and determine a bound *k* and a new circuit with the redundancies removed (called a *k*-delayed replacement) [4]. In [2] such redundancies are identified, one is then removed, and new ones identified. This is repeated until no more can be found. In [4], a set of "compatible" redundancies is found and removed simultaneously.

The method of [4] derives a constant *n* which is the difference between the time frame of an identified redundancy and the least time frame needed to infer this redundancy. Their theorem states that if the redundancy is used to create the new circuit, then it is an *n*-delayed replacement of the original. Note that in *n*-delayed replacement, it is *B* that is delayed for *n* cycles before equivalence can be guaranteed, but in Theorem 2 it is *A* that is delayed *n* cycles.

A sequential ATPG engine can be used to determine if a test vector sequence can be found which justifies a state that activates the fault in *n* cycles and then propagates the fault effect to a PO in *k* cycles. If none can be found, the fault might be redundant, but three things can go wrong; (i) undetectable faults are not necessarily redundant, (ii) the justification and propagation conditions are usually done on the good machine, and (iii) finite values for *n* and *k* were used. Such a fault is a good candidate for redundancy removal, but the result must be sequentially verified, possibly by applying Theorems 1-3, which may work if *A* or *B* are supplying a sufficient set of SDCs or ODCs. An interesting discussion of some incorrect "proofs" in the literature related to the use of ATPG for redundancy removal can be found in [3], as well as limitations of some other methods.

4 Experimental Results

A few experimental results are shown in Table 1. They were designed to compare the efficiency of applying the new SEC approach of this paper with the general SEC method of the ABC system. Six large industrial benchmarks were synthesized using sequential clock-gating transforms, based intuitively on sequential ODC arguments, but not formally proved. The synthesized versions are denoted by *B* and the originals by *A*. Columns 1-5 give the sizes of the circuits. The entries in column 6-11 give the times in minutes taken to verify equivalence. The columns labeled *New* denote the use of Theorem 1 and Berkeley's ABC system CEC algorithm to prove SEC. The column *ABC general* denotes that the ABC command *dsec* was used. Columns *seq-j* denote experiments where (B, A^j) was compared combinationally with A^{j+1} to illustrate how CEC run-times might scale as *j* increases. The items marked with * or **, denote that the corresponding equivalence checking problem timed out.

Observations.

1. In general, *New* is significantly faster than *General,* as expected (about 30 times faster when *General* could complete. The fact that *General* could actually complete on three out of six large problems was surprising to us).

2. Except for Design 4, CEC times scale approximately linearly with the size of the CEC problem.

5 Conclusions and Future Work

Some sequential synthesis transforms do not use the initial state information but preserve a circuit's behavior starting from any initial state. Such transforms may use sequential observability [2] [4] and can be practical because they do not use state space search or can be argued using structural information as in the case of many clock-gating methods. These contrast with transforms that extract ODCs using reachability analysis such as BDD reachability, interpolation or SAT-based induction [5].

In the sequential observability case, it may be possible that sequential equivalence can be proved by combinational equivalence checking methods, making SEC much easier. This can have a significant impact in applications where parts of the circuit are changed based on a local view of the circuit.

We have given a method for SEC, which can be effective in certain special cases, leading to considerable reduction in computation effort. The method is conservative; it fails no information is obtained. Some conditions under which it can be expected to succeed include sequential clock-gating methods and methods that alter pipeline behavior. Experimental results were given on a six large industrial SEC problems, comparing the sequentially synthesized design against the original design. It was demonstrated that the new SEC method was about 30 times faster than in the general case. In addition, it was able to check three examples where general SEC could not complete.

Our theorems are stated in terms of having a one-to-one correspondence between the FFs of *A* and *B*. This was necessary for combinational circuits (*A,B*) or (*B,A*) to be formed where signals in the first circuit are wired to their corresponding signals in the second circuit. However, some clock-gating transforms require that a signal be delayed one or more time-frames. In such cases, FFs must be introduced in *B* that have no correspondence in *A*. This can be handled by introducing dummy FFs in *A* with no fanout.

We conjecture, more generally, that it is sufficient to find two cuts of the same size, one in *A* and the other in *B*. The signals in the cuts can be a mixture of internal wires and FFs. It may be that the only requirement is that the cuts are feedback arc sets, i.e. cutting them makes each circuit acyclic. This would allow applications of the theorems to retimed circuits.

Also, it would be desirable to have a practical method to check general *k*-delayed equivalence, such as for designs produced by the methods of [2][4]. These situations are cases of local sequential synthesis being done. Note that if $S_B^k \subseteq S_A^k$, then Theorem 2 applies and can be used to prove *k*-delayed equivalence. It is possible that Theorem 3 can be used in such cases, although at the moment, we have no experimental results on this.

Theorem 1 legitimizes sequential synthesis based on unrolling a sequential machine *A*, *k* times, and combinationally synthesizing the first copy of *A* to obtain a new equivalent sequential machine

B. However, we have not done experiments on how effective this might be in terms of improved quality of the synthesis result.

Acknowledgements

This work was supported by SRC contract 1444.001, NSF grant CCF-0702668, and the (now-defunct) California Micro program.

References

[1] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman and G. Janssen, "Scalable Sequential Equivalence Checking across Arbitrary Design Transformations." International Conference on Computer Design, San Jose, CA. October 2006.

[2] M. A. Iyer, D. E. Long, and M. Abramovici, “Identifying sequential redundancies without search,” *DAC’96*, pp. 457-462.

[3] M. A. Iyer, D. E. Long, and M. Abramovici, ‘‘Surprises in Sequential Redundancy Identification,’’ *EDTC’96*.

[4] A. Mehrotra, S. Qadeer, V. Singhal, R. K. Brayton, A. Aziz, and A. L. Sangiovanni-Vincentelli. “Sequential optimisation without state space exploration”. *ICCAD’97*, pp. 208-215.

[5] A. Mishchenko, M. L. Case, R. K. Brayton, and S. Jang, "Scalable and scalably-verifiable sequential synthesis", *ICCAD'08*. http://www.eecs.berkeley.edu/~alanmi/publications/2008/iccad08_seq.pdf

[6] A. Mishchenko and R. K. Brayton, "Recording synthesis history for sequential verification", *FMCAD'08*, pp. 27-34. http://www.eecs.berkeley.edu/~alanmi/publications/2008/fmcad08_haig.pdf

[7] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman, “Exploiting suspected redundancy without proving it”. *DAC’05*.

[8] "Scalable Sequential Equivalence Checking across Arbitrary Design Transformations."

[9] S. Chatterjee, A. Mishchenko, R. Brayton, and A. Kuehlmann. “On Resolution proofs for combinational equivalence”, *DAC’07*.

[10] V Singhal and C. Pixley. “The verification problem for safe replaceability,” *CAV’94*, LNCS, Vol. 818, pp. 311-323.

Table 1. Experimental results.

Design	Statistics				Seq-1		Seq-2	Seq-3	Seq-4	Seq-5
	Ands	Flops	PI	PO	New	General	New	New	New	New
1	39282	6506	51	83	0.68	15.16	0.98	1.34	1.55	1.78
2	18932	10544	96	115	0.51	18.88	0.7	0.88	1.06	1.25
3	31103	7276	105	79	0.78	*60.55	1.29	1.51	1.69	1.63
4	81782	13822	394	703	1.61	*152.21	2.34	17.22	72.93	267.83
5	45241	11595	1741	301	0.94	25.63	1.26	1.63	2.37	**6.18
6	114824	15284	857	804	2.05	*112.83	3.26	4.09	4.79	6.22

Notes: * General sequence equivalence in ABC timed out. Although time-out was set to 1 hour, we were curious to see if the problem could complete if more time was given. Hence the irregular time-out times reported.
** Unresolved by ABC combinational equivalence checking
Entries in columns 6-11 denote run times in minutes.
Seq-*j* denotes the CEC problem where *j* copies of *A* are used, i.e. (B,A^j) is compared to A^{j+1}.