# Speculative Reduction-Based Scalable Redundancy Identification

Hari Mony[1] Jason Baumgartner[1] Alan Mishchenko[2] Robert Brayton[2]
[1]IBM Systems & Technology Group, Austin, TX
[2]Department of EECS, University of California, Berkeley

## Abstract

*The process of sequential redundancy identification is the cornerstone of sequential synthesis and equivalence checking frameworks. The scalability of the proof obligations inherent in redundancy identification hinges not only upon the ability to cross-assume those redundancies, but also upon the way in which these assumptions are leveraged. In this paper, we study the technique of* speculative reduction *for efficiently modeling redundancy assumptions. We provide theoretical and experimental evidence to demonstrate that speculative reduction is fundamental to the scalability of the redundancy identification process under various proof techniques. We also propose several techniques to speed up induction-based redundancy identification. Experiments demonstrate the effectiveness of our techniques in enabling substantially faster redundancy identification, up to six orders of magnitude on large designs.*

## 1 Introduction

Sequential redundancy identification is the process of demonstrating that two gates in a design always evaluate to the same or opposite values in all reachable states. Once a pair of redundant gates are identified, the design may be simplified by merging one of the gates onto the other. Efficient redundancy identification algorithms are at the core of scalable sequential equivalence checking [1] as well as sequential synthesis [2] frameworks.

For scalability, many sequential redundancy identification frameworks adopt an *assume-then-prove* [3, 4, 5, 6, 7] paradigm, where using assume-guarantee reasoning, certain suspected redundancies may be assumed to simplify the proof that others hold. A variety of proof techniques including simple induction [4], $k$-step induction [8, 6, 7], and synergistic transformation and verification algorithms [6] have been proposed to identify sequential redundancy. Regardless of the proof technique, the scalability of redundancy identification hinges on the way redundancy assumptions are modeled in the assume-then-prove framework.

A technique called *speculative reduction* was proposed in [6] to efficiently model redundancy assumptions through structural logic simplifications. Speculative reduction is key to the scalability of induction-based techniques; the gain in runtime due to speculative reduction can be several orders of magnitude as noted in [2]. Furthermore, speculative reduction is critical to enable the efficient use of arbitrary transformation and verification algorithms to prove redundancy assumptions [6].

In this paper, we provide a detailed study of the technique of speculative reduction. We first provide a theoretical justification of how speculative reduction is critical to the efficiency of sequential redundancy identification. We next provide detailed experimental evidence to illustrate the benefit of speculative reduction to induction-based proofs. We additionally present several new techniques to speed up this process, which collectively enable up to six orders of magnitude speedup. We also provide an empirical study on the necessity of speculative reduction to enable the application of algorithms such as interpolation [9] and localization [10] to identify non-inductive redundancy. While several prior works have noted the benefit of speculative reduction (e.g., [6, 2]), this paper is the first to study this benefit in detail, both theoretically and experimentally.

## 2 Preliminaries

We represent our design as a *netlist*, which is a tuple $\langle\langle V, E\rangle, G\rangle$ comprising a directed graph with vertices $V$ and edges $E \subseteq V \times V$. Function $G : V \mapsto types$ represents a mapping from vertices to $gate\ types$, including constants, primary inputs, registers, and combinational gates with various functions. A *state* is a valuation to the registers of the netlist. Registers have designated *initial states* which define the time-0 state of the netlist, as well as *next-state functions* which define the time $i + 1$ state of the netlist. A *trace* is a temporal sequence of Boolean valuations to netlist vertices which is consistent with $G$.

Given a netlist, redundancy identification frameworks based on the assume-then-prove paradigm such as those of [3, 4, 5, 6, 7] operate as per the algorithm of Figure 1. Step 1 of this algorithm guesses the redundancy candidates, using a variety of techniques which includes both *semantic* and *syntactic* approaches [6].

Once the redundancy candidates are guessed, the next step is to create a netlist which includes the verification goals necessary to prove those candidates correct. For efficiency, it is desirable to be able to *assume* candidate re-

1. Guess the *redundancy candidates* – sets of equivalence classes of gates, where each gate $g$ in equivalence class $Q(g)$ is suspected to be equivalent to every other gate in the same class, along every trace.
2. Select a *representative gate* $R\big(Q(g)\big)$ from each equiv. class $Q(g)$.
3. Construct the *speculatively-reduced netlist* by replacing the source gate $g$ of every edge $(g, h) \in E$ by $R\big(Q(g)\big)$. Additionally, for each gate $g$, add a miter representing function $g \not\equiv R\big(Q(g)\big)$.
4. Attempt to prove that each of the miters is unassertable.
5. If any miters cannot be proven unassertable, refine the equivalence classes to separate the corresponding gates; go to Step 2.
6. All miters have been proven unassertable; return the accurate equivalence classes which reflect true redundancy.

Figure 1: Sequential redundancy identification framework

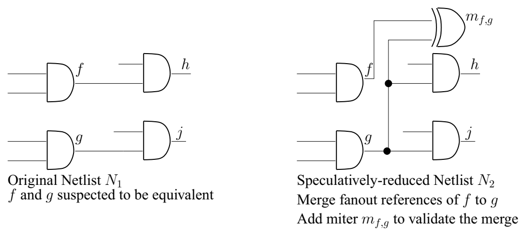
$f$
$h$
$g$
$j$
Original Netlist $N_1$
$f$ and $g$ suspected to be equivalent
$m_{f,g}$
$f$
$h$
$g$
$j$
Speculatively-reduced Netlist $N_2$
Merge fanout references of $f$ to $g$
Add miter $m_{f,g}$ to validate the merge

Figure 2: Illustration of speculative reduction

dundancies to simplify these proof obligations. Speculative reduction was proposed in [6] to efficiently model these assumptions at the netlist level. The netlist is first speculatively reduced by merging the fanout references of each candidate gate onto a representative from its equivalence class. Next, to prove the correctness of the assumptions, a *miter* is added to check whether the candidate and its representative can be differentiated as illustrated in Figure 2. Finally, proof analysis is performed on the speculatively-reduced netlist to attempt to validate the correctness of the redundancy candidates, represented by the un-assertability of the miters. Failed proofs, whether falsified or inconclusive (e.g., due to resource limitations), cause a refinement of the candidates and another proof iteration.

## 3 Speculative Reduction

In this section, we detail the ways in which the scalability of sequential redundancy identification hinges on speculative reduction. These advantages will be experimentally justified in Section 5.

First, speculative reduction simplifies proof obligations by minimizing the logic in the fanin of each miter. For example, in Figure 2, gate $f$ no longer will appear in the logic driving gate $h$. Cumulatively, the set of all speculative reductions often enables a dramatic reduction in the size of the speculatively-reduced netlist, and even in the number of distinct miters to be solved. For example, if the dangling inputs of gates $h$ and $j$ in Figure 2 are redundancy candidates, the miter for $h \not\equiv j$ would trivially be 0 since $h$ and $j$ would hash to the same gate in the speculatively-reduced netlist. Speculative reduction plays a similar role to cut-pointing in combinational equivalence checking; if pairs of corresponding registers of designs being sequentially equivalence checked are speculatively reduced, the complexity of the equivalence checking problem [11] may be reduced from PSPACE to NP, precluding any need for sequential reasoning. Though unlike cut-pointing which replaces registers by inputs and thus loses reachability information, speculative reduction preserves netlist behavior (provided that the redundancy candidates are correct), hence does not yield false failures e.g. if a miter checks the equivalence of two logic cones which were optimized using unreachability conditions.

Speculative reduction is also key to enabling arbitrary transformation and verification algorithms to efficiently discharge complex non-inductive miters [6]. Experiments confirm that transformation and verification algorithms are much more effective *after* the speculative merging. For example, speculative merging helps share logic across the netlist comprising two designs being equivalence checked, in turn enabling sharing-aware logic rewriting algorithms [12] and min-area retiming [13] to further reduce the speculatively-reduced netlist in ways that would not be possible otherwise. Speculative merging can also increase the capability of structural isomorphism detection [14] to suppress isomorphic miters. In equivalence checking frameworks, abstraction algorithms such as localization [10] – which seeks to eliminate logic irrelevant to the current proof obligation through cutpointing – will be ineffective without speculative merging, often requiring the entire logic cones driving the two redundancy candidates. With speculative merging, only the subset of logic required to enforce adequate satisfiability don't cares over the locally redesigned logic being equivalence checked will be necessary as noted in [6]. Verification algorithms also face bottlenecks without speculative reduction. Interpolation [9] is a verification algorithm that we have found to be very useful to solve non-inductive miters. Our experiments demonstrate that in the absence of speculative reduction, interpolation rarely converges and has run-times and memory consumption that are several orders of magnitude higher.

The last and possibly biggest advantage of speculative reduction is that it enables the efficient elimination of simpler miters, allowing heavier-weight proof algorithms to focus solely on the inherently most complex miters. To illustrate this advantage, consider the two designs being equivalence checked in Figure 3(a), each having two components $C_1$ and $C_2$. Assume that the changes to $C_1$ enable induction-based techniques to prove $C_1 \equiv C_1'$. However induction cannot prove $C_2 \equiv C_2'$. For the design in Figure 3(a), we can apply induction to first merge the outputs of $C_1$ and $C_1'$ and then use heavier-weight algorithms to address $C_2 \equiv C_2'$. Speculative reduction is not needed to solve this problem given the appropriate miter scheduling,

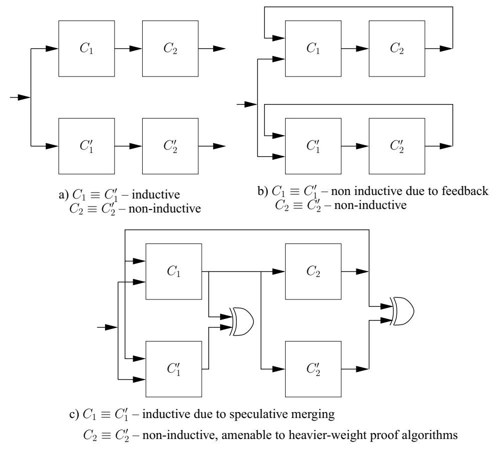
$C_1$
$C_2$
$C'_1$
$C'_2$
a) $C_1 \equiv C'_1$ – inductive
$C_2 \equiv C'_2$ – non-inductive
b) $C_1 \equiv C'_1$ – non inductive due to feedback
$C_2 \equiv C'_2$ – non-inductive
c) $C_1 \equiv C'_1$ – inductive due to speculative merging
$C_2 \equiv C'_2$ – non-inductive, amenable to heavier-weight proof algorithms

Figure 3: Advantages of speculative reduction

e.g., using a levelized scheduling as in combinational equivalence checks [15]. Now consider Figure 3(b), where there is feedback from $C_2$ to $C_1$. In this case, the equivalence check between $C_1$ and $C'_1$ will no longer be inductive. The inability to inductively prove $C_1 \equiv C'_1$ will make the proof of $C_2 \equiv C'_2$ even more difficult since there is no merge of the outputs of $C_1$ with $C'_1$ without speculation. However, if we use speculative reduction as illustrated in Figure 3(c), the feedback from $C_2$ is no longer a bottleneck for induction to prove $C_1 \equiv C'_1$, hence these miters may be easily discharged allowing heavier-weight algorithms to focus solely on the simplified proof of $C_2 \equiv C'_2$.

## 4 Refining the Redundancy Candidates

It is easy to understand why the efficiency of redundancy identification is dependent on efficiently solving the miters. However, the efficiency of the refinement process also has a big role in overall performance, since there may be as many refinements (and hence proof attempts) as there are gates in the netlist. Techniques for guessing redundancy candidates are practically inexact, hence many initial candidate guesses may be incorrect requiring refinement. Moreover, the inconclusive nature of algorithms such as resource-bounded induction also entails refinements due to the inability to prove even accurate candidates.

$k$-induction frameworks used to prove miters operate as follows.

- **Base Case**: Validate that the miters are unassertable in the first $k$ time-steps starting from the initial states.
- **Inductive Case**: Assuming that the miters are unassertable for the first $k$ time-steps from *any* state, check whether they can be asserted at time-step $k + 1$.

The key observation is that when induction fails to prove a miter unassertable, it will provide a trace starting from either an initial state or an inductive state showing how the miter can be asserted. Due to resource limitations, traces from the base case will only exhibit candidate inequivalences which may be demonstrated in earlier time-steps. The inductive step may yield traces both for candidates which may become inequivalent only at a deeper time-step than the base case could reach, or spurious traces from unreachable states due to the incompleteness of resource-bounded $k$-induction. While the induction counterexample traces may be spurious, for efficiency it is nonetheless desirable to identify *all* candidates which may be differentiated by a given induction trace (similar to the reuse of SAT sweeping traces for combinational analysis [15, 16]) so that it is not necessary to compute a redundant counterexample asserting such miters. The set of differentiated candidates may be computed by simulating the behavior of the original netlist with the inductive state and input sequence from the induction trace. Two enhancements that we have discovered for this process are as follows.

**(1)** During simulation of the induction trace, in addition to merely checking whether other miters for the current refinement iteration can also be asserted by that trace, we can check for mismatches between candidates of the equivalence classes being refined for the next iteration which do not yet have miters constructed. In other words, this simulation can look for mismatches between candidates and their representatives against the evolving window of equivalence classes being refined from the current to the next refinement iteration. This enhancement does not alter the final identified redundancy, since every refinement effectively weakens the induction hypothesis. If the gates in the new equivalence classes can be differentiated by the induction trace with an inductive starting state that corresponds to the pre-refinement induction hypothesis, they can be safely refined since a failed proof attempt under the previous stronger induction hypothesis implies that proof attempts will fail under the weaker post-refinement induction hypothesis.

**(2)** We can sequentially extend the induction trace by a number of time-steps (randomly generating input valuations for the additional time-steps) and check whether equivalence classes can be further refined during simulation of the extended trace. This extension does not alter the final identified redundancy because the inductive starting state already adheres to the current "stronger" induction hypothesis, and each extended time-step will by construction adhere to the "weaker" induction hypothesis which would hold in a subsequent iteration after equivalence class partitioning.

It is noteworthy that both of these optimizations enable maximal incrementality for efficiency, vs. rebuilding the speculatively-reduced netlist upon every refinement. The induction counterexamples can be used for refining as noted above irrespective of the nature of the SAT-solver used for induction, i.e., whether it is CNF-based or circuit-based. In practice, we have observed that a circuit SAT solver [17] is more powerful than a CNF-based SAT solver [18] due to its native ability to produce minimal assignments in counterexamples. Any unassigned inputs may be randomly assigned, which further randomizes the inductive starting state. This enables a huge reduction in the number of counterexamples generated by the SAT-solver and thereby in overall resources. Without this capability, resimulation is often less effective, motivating the use of approximating tricks such as *distance-1* simulation which resimulates all traces resulting from flipping a single input valuation in the original complete counterexample [16]. Post-processing CNF traces to minimize assignments [19] may be equally effective, though at an additional cost.

## 5 Experimental Results

In this section we provide experiments to illustrate the power of speculative reduction and our accelerating techniques. All experiments were run on a 2.1GHz processor, using the IBM internal verification tool *SixthSense*.

We first provide experimental results for induction-based redundancy identification, to illustrate the advantages of using speculative reduction and the techniques for accelerated refinement proposed in Section 4. The benchmarks include a mixture of difficult industrial sequential equivalence checking and synthesis examples. **IBUF**, **SSC**, **MIS1**, **MIS2**, **MIS3**, **SDQ** and **SBIU** are sequential equivalence checking examples, while **DAA**, **SMM**, **FIER**, **CIU** and **L2** are synthesis examples. Five different flavors of SAT-based simple induction were run on these benchmarks. These include: **(1)** speculative reduction disabled, **(2)** speculative reduction enabled, **(3)** speculative reduction with induction traces resimulated to check for additional miter assertions, **(4)** speculative reduction with induction traces resimulated to differentiate newly-generated equivalence classes (item **(1)** from Section 4), and **(5)** speculative reduction with induction traces sequentially extended by 2 time-steps (item **(2)** from Section 4).

The first two columns in Table 1 indicate the benchmark name and size of the original netlist. Columns 3-11 indicate the runtime for each flavor of induction along with the improvement in runtime compared to the previous inferior flavor. Column 12 indicates the cumulative improvement. Column 5 illustrates the importance of speculative reduction for improving the scalability of induction-based redundancy identification. Speculative reduction results in an average speedup of $21.94\times$. Simulation of induction counterexamples is a powerful extension to this technique, enabling an additional $11.65\times$ speedup on average by transferring some of the burden to assert miters from SAT to simulation. The early refinement techniques described in Section 4 enable a speedup of $104.56\times$ on average over the use of counterexample simulation alone.

A particularly interesting testcase is **L2**. Using our techniques, we were able to identify all inductively-provable redundant gates in 165 seconds. If speculative reduction were disabled, we are only able to complete 4 iterations of the "assume-then-prove" algorithm before we hit a timeout of 48 hours. Extrapolating, it would have taken almost 10 years to reach the fixed-point if we had continued to run all 6860 iterations necessary as per the run with speculative reduction with simulation (induction flavor **(3)**).

To identify the precise source of these runtime improvements, we provide additional experimental results to compute the total number of miters solved by SAT (Table 2) and the number of refinement iterations and induction counterexamples (Table 3). As illustrated in Table 2, speculative reduction results in a $11.31\times$ reduction on average in the total number of miters solved by SAT. This is due to the fact that speculative merging reduces the number of distinct nontrivial miters. Our accelerated refinement techniques enable an additional average reduction by $815.8\times$ in the number of miters solved by SAT. There are two reasons for this reduction: **(1)** early refinement shields SAT from receiving certain miters that are destined to fail through induction, and **(2)** since the number of refinement iterations is reduced by this technique, induction need not re-solve as many identical unassertable miters across iterations.

Table 3 illustrates the advantage of using simulation to assert miters instead of relying on SAT-based analysis. Here we list the total number of induction traces and the number of refinements to reach a fixed-point for various flavors of induction. Trace simulation enables $27.04\times$ reduction on average in the number of traces generated during induction. This is due to its ability to preclude SAT from needing to solve the corresponding miters. Our accelerated refinement techniques further reduces the number of induction traces, by $124.02\times$ on average. This is due to the ability to drastically reduce the number of refinement iterations, by an average of $113.45\times$.

Speculative reduction is also indispensable for speeding up sequential equivalence checking in the presence of recorded synthesis history [20]. In this work, a History And-Inverter Graph (HAIG) is constructed during sequential synthesis to represent the original, intermediate, and final circuit structures. Recorded along with these structures are proof obligations which validate each synthesis step. Miters may be constructed to represent the equivalence of each pre- and post-synthesis component; if all miters are successfully proven as un-assertable, the synthesis process has been proven as correct often with dramatically lesser resources than would be required by an overall input-output equivalence check across the sum of the synthesis steps. Because these miters each correlate to suspected equivalences, speculative reduction can be used to simplify the overall verification problem. Though space does not allow us to provide detailed experiments for this application, we have measured a speedup of $20.5\times$ across a testsuite of 12 HAIG

| **Design Info** | | **Spec. Red. Disabled (1)** | **Spec. Red. Enabled (2)** | | **Spec. Red. & Cex Sim (3)** | | **Spec. Red. & Cex Analysis (4)** | | **Spec. Red. & Seq Cex Extension (5)** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Name** | **Gates** | **Time (s)** | **Time (s)** | **Improve. w.r.t. Col 3** | **Time (s)** | **Improve. w.r.t. Col 4** | **Time (s)** | **Improve. w.r.t. Col 6** | **Time (s)** | **Improve. w.r.t. Col 8** | **Improve. w.r.t. Col 3** |
| **IBUF** | 11424 | 9963.54 | 7525.14 | 1.32× | 1670.70 | 4.50× | 27.03 | 61.88× | 11.74 | 2.31× | 849.06× |
| **SSC** | 16385 | 6492.82 | 5235.01 | 1.24× | 1120.95 | 4.67× | 74.46 | 15.05× | 28.32 | 2.63× | 229.42× |
| **MIS1** | 21044 | 22928.14 | 1304.81 | 17.57× | 288.31 | 4.53× | 18.53 | 15.58× | 7.36 | 2.52× | 3100.10× |
| **MIS2** | 22772 | 67861.09 | 1040.27 | 65.23× | 222.53 | 4.67× | 20.20 | 11.01× | 8.84 | 2.29× | 7698.75× |
| **MIS3** | 18094 | 39752.06 | 6869.12 | 5.79× | 1156.71 | 5.94× | 88.88 | 13.01× | 26.69 | 3.33× | 1489.82× |
| **SDQ** | 26666 | 16736.32 | 7263.21 | 2.30× | 2082.55 | 3.49× | 84.80 | 24.56× | 40.05 | 2.12× | 417.94× |
| **SBIU** | 30442 | 8408.64 | 4828.65 | 1.74× | 820.21 | 5.89× | 88.53 | 9.27× | 31.91 | 2.77× | 263.55× |
| **DAA** | 31424 | 41027.82 | 32449.83 | 1.26× | 5926.20 | 5.48× | 199.36 | 29.71× | 90.43 | 2.20× | 452.49× |
| **SMM** | 85574 | 846.11 | 83.46 | 10.13× | 81.56 | 1.02× | 8.55 | 9.6× | 5.78 | 1.46× | 145.38× |
| **FIER** | 89943 | 158938.12 | 15028.18 | 10.57× | 9357.58 | 1.61× | 20.30 | 461.00× | 15.01 | 1.35× | 10617.11× |
| **CIU** | 92232 | 6744.56 | 166.07 | 40.63× | 138.58 | 1.2× | 12.73 | 10.91× | 10.61 | 1.19× | 637.30× |
| **L2** | 374977 | 172800 (4 / 6860) | 172800 (420 / 6860) | 105× | 28404.73 | 96.75× | 214.24 | 132.54× | 165.02 | 1.30× | 1748739.80× |
| **AVG.** | 68415 | | | 21.94× | | 11.65× | | 66.18× | | 2.12× | 147886.66× |

Table 1: Induction-based redundancy identification results

| **Design Info** | **Spec. Red. Disabled (1)** | **Spec. Red. Enabled (2)** | | **Spec. Red. & Cex Analysis (4)** | | **Spec. Red. & Seq Cex Extension (5)** | | |
|---|---|---|---|---|---|---|---|---|
| **Name** | **Miters Solved** | **Miters Solved** | **Improvement w.r.t. Col 2** | **Miters Solved** | **Improvement w.r.t. Col 3** | **Miters Solved** | **Improvement w.r.t. Col 5** | **Improvement w.r.t. Col 2** |
| **IBUF** | 1970075 | 1588746 | 1.24× | 11978 | 132.64× | 7318 | 1.64× | 269.20× |
| **SSC** | 1182197 | 794239 | 1.49× | 35554 | 22.34× | 18028 | 1.97× | 65.65× |
| **MIS1** | 2850774 | 642594 | 4.44× | 13466 | 47.72× | 6785 | 1.98× | 420.51× |
| **MIS2** | 3180878 | 351192 | 9.06× | 13784 | 25.49× | 6870 | 2.00× | 463.36× |
| **MIS3** | 3351149 | 1374897 | 2.44× | 46505 | 29.56× | 26145 | 1.78× | 128.30× |
| **SDQ** | 719262 | 567757 | 1.27× | 18456 | 30.76× | 9538 | 1.93× | 75.39× |
| **SBIU** | 1705755 | 1054606 | 1.62× | 24062 | 43.83× | 13792 | 1.74× | 123.54× |
| **DAA** | 4836934 | 4147228 | 1.67× | 44949 | 92.27× | 28321 | 1.58× | 244.56× |
| **SMM** | 172579 | 161487 | 1.07× | 2743 | 58.87× | 2221 | 1.23× | 77.79× |
| **FIER** | 40101056 | 5161148 | 7.77× | 9137 | 564.86× | 8265 | 1.10× | 4852.02× |
| **CIU** | 334274 | 92709 | 3.61× | 7577 | 12.23× | 5376 | 1.41× | 62.17× |
| **L2** | 6207648900 | 61660373 | 100.07× | 62334 | 989.19× | 59169 | 1.05× | 103937.65× |
| **AVG.** | | | 11.31× | | 170.81× | | 1.62× | 9226.67× |

Table 2: Miters solved by SAT during induction-based redundancy identification

| **Design Info** | **Spec. Red. Enabled (2)** | **Spec. Red. & Cex Sim (3)** | | **Spec. Red. & Cex Analysis (4)** | | **Spec. Red. & Seq Cex Extension (5)** | | |
|---|---|---|---|---|---|---|---|---|
| **Name** | **Induction Cexs; Refinements** | **Induction Cexs; Refinements** | **Improvement w.r.t. Col 2** | **Induction Cexs; Refinements** | **Improvement w.r.t. Col 3** | **Induction Cexs Refinements** | **Improvement w.r.t. Col 5** | **Improvement w.r.t. Col 2** |
| **IBUF** | 552623; 988 | 13712; 988 | 40.30× | 1561;22 | 8.78×; 44.90× | 701; 11 | 2.22×; 2.00× | 787.91×; 89.80× |
| **SSC** | 202892; 537 | 20974; 537 | 9.67× | 1869; 38 | 11.22×; 14.13× | 794; 19 | 2.35×; 2.00× | 255.40×; 28.26× |
| **MIS1** | 230457; 709 | 9310; 709 | 24.75× | 1511; 29 | 6.16×; 24.45× | 497; 18 | 3.04×; 1.61× | 463.51×; 39.39× |
| **MIS2** | 136217; 408 | 8275; 408 | 16.46× | 1391; 30 | 5.95×; 13.60× | 468; 16 | 2.97×; 1.88× | 291.10×; 25.50× |
| **MIS3** | 368629; 564 | 13060; 564 | 28.23× | 2145; 32 | 6.09×; 17.63× | 649; 19 | 3.30×; 1.68× | 568.21×; 29.68× |
| **SDQ** | 177672; 407 | 10827; 407 | 16.41× | 1974; 10 | 5.48×; 40.70× | 440; 7 | 4.48×; 1.43× | 403.46×; 58.14× |
| **SBIU** | 372830; 767 | 24403; 767 | 15.28× | 4711; 22 | 5.18×; 34.86× | 1436; 17 | 3.28×; 1.29× | 259.66×; 45.12× |
| **DAA** | 969845; 1633 | 49374; 1633 | 19.64× | 6118; 25 | 8.07×; 65.32× | 2484; 16 | 2.46×; 1.56× | 390.35×; 102.06× |
| **SMM** | 157002; 428 | 6097; 428 | 25.75× | 392; 14 | 15.55×; 30.57× | 327; 10 | 1.19×; 1.40× | 476.49×; 42.80× |
| **FIER** | 5157805; 3199 | 638443; 3199 | 8.08× | 704; 6 | 906.88×; 533.16× | 579; 5 | 1.22×; 1.20× | 8939.66×; 639.80× |
| **CIU** | 89946; 340 | 8263; 340 | 10.89× | 692; 22 | 11.94×; 15.45× | 431; 14 | 1.60×; 1.57× | 208.76×; 24.29× |
| **L2** | 29625219; 6860 | 271791; 6860 | 109.00× | 1701; 30 | 159.78×; 228.66× | 1502; 29 | 1.13×; 1.03× | 19680.10×; 236.55× |
| **AVG.** | | | 27.04× | | 95.92×; 88.62× | | 2.44×; 1.55× | 2727.05×; 113.45× |

Table 3: Induction traces and refinement iterations during redundancy identification

examples using speculative reduction. This benefit is not as pronounced as for redundancy identification since it does not require refinement, though is nonetheless substantial.

To illustrate the advantage of applying transformation and verification algorithms on the speculatively-reduced netlist, we performed a set of experiments which are detailed in Table 4. For SDQ, we ran interpolation (**ITP**) on both the speculatively-reduced netlist as well as the original netlist checking overall input-output equivalence. **ITP** was able to solve all the miters (including the original equivalence obligations) in the speculatively-reduced netlist in 51 seconds, consuming only 54 MB of memory. Without speculative reduction, **ITP** could not solve any original equivalence checks within 48 hours, consuming 10.5 GB. This clearly illustrates the advantage of speculative reduction for interpolation. For IFU, we did three sets of experiments: **(1)** localization (**LOC**) followed by interpolation on the original netlist without speculative reduction, **(2) LOC** followed by interpolation on the speculatively-reduced netlist, (**3**) interpolation directly on the speculatively-reduced netlist. **LOC** yields several orders of magnitude reduction in the size of each miter (a single representative miter is illus-

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **SDQ** | **Spec. Red. Disabled** | **ITP** | | | | | | | | **Spec. Red. Enabled** | **ITP** |
| Registers | 3655 | 178200s | | | | | | | | 1833 | 51s |
| Gates | 27429 | 10575.5 MB | | | | | | | | 11220 | 54 MB |
| Miters | 2141 | 2141 | | | | | | | | 61 | 2141 |
| **IFU** | **Spec. Red. Disabled** | **LOC** | **ITP** | | **Spec. Red. Enabled** | **LOC** | **ITP** | | | **Spec. Red. Enabled** | **ITP** |
| Registers | 66421 | 15441 | 178200 s | | 33231 | 4 | 646 s | | | 33231 | 178200s |
| Gates | 686790 | 241461 | 1678 MB | | 303620 | 22 | 832 MB | | | 303620 | 7184 MB |
| Miters | 33879 | 1 | 1 | | 1035 | 1 | 0 | | | 1035 | 435 |
| **SYNTH3** | **Spec. Red. Disabled** | **COM** | **MOD** | **RET** | **COM** | | **Spec. Red. Enabled** | **COM** | **MOD** | **RET** | **COM** |
| Registers | 5180 | 5180 | 5167 | 1343 | 810 | | 5180 | 2439 | 2437 | 7 | 35.81s |
| Gates | 68389 | 68377 | 54750 | 70449 | 48359 | | 57477 | 32399 | 25176 | 39378 | 103 MB |
| Miters | 2293 | 2293 | 2293 | 2293 | 2000 | | 1631 | 294 | 294 | 294 | 0 |
| **IOC** | **Spec. Red. Disabled** | **COM** | **MOD** | **COM** | **ISO** | | **Spec. Red. Enabled** | **COM** | **MOD** | **COM** | **ISO** |
| Registers | 5382 | 5319 | 5319 | 3268 | 2194 | | 5324 | 5125 | 5124 | 896 | 669 |
| Gates | 35219 | 33747 | 76773 | 30636 | 20160 | | 29235 | 23080 | 31541 | 4417 | 3698 |
| Miters | 4119 | 4077 | 4034 | 2557 | 1678 | | 3967 | 3658 | 3550 | 1169 | 380 |

Table 4: Multi-algorithmic benefit of speculative reduction

trated) and this enables **ITP** to easily solve these miters. This example illustrates that speculative reduction alone is insufficient to enable certain **ITP**-based proofs, which times out at 48 hours leaving 435 miters unsolved. The advantage of speculative reduction for localization is also illustrated by this example. Applying **LOC** on the original netlist results in a localized netlist with 15441 registers as opposed to 4 when applying **LOC** on the speculatively-reduced netlist. **ITP** times out on this much larger netlist, though efficiently solves the reduced netlist.

SYNTH3 and IOC illustrate the advantage of speculative reduction across a larger set of algorithms. For SYNTH3, the application of phase abstraction **MOD** [21] followed by min-area retiming **RET** [13] and combinational optimization **COM** [12] on the speculatively-reduced netlist is able to efficiently solve all miters. For IOC, **MOD** and **COM** followed by structural isomorphism detection **ISO** dramatically simplifies the speculatively-reduced netlist.

## 6 Conclusion

In this paper, we have demonstrated theoretically and empirically that speculative reduction is fundamental to the scalability of redundancy identification, irrespective of the proof technique used therein. We also introduced new techniques which enable several orders of magnitude speedup in induction-based redundancy identification.

## References

[1] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen, “Scalable sequential equivalence checking across arbitrary design transformations,” in *ICCD*, 2006.

[2] A. Mishchenko, M. Case, R. Brayton, and S. Jang, “Scalable and scalably-verifiable sequential synthesis,” in *ICCAD*, Nov. 2008.

[3] D. Stoffel and W. Kunz, “Record & play: A structural fixed point iteration for sequential circuit verification,” in *ICCAD*, Nov. 1997.

[4] C. A. J. van Eijk, “Sequential equivalence checking without state space traversal,” in *DATE*, Feb. 1998.

[5] Shi-Yu Huang et al., “AQUILA: An equivalence checking system for large sequential designs,” *TCAD*, 2000.

[6] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman, “Exploiting suspected redundancy without proving it,” in *DAC*, June 2005.

[7] F. Lu and K.-T. Cheng, “IChecker: An efficient checker for inductive invariants,” in *HLDVT*, Nov. 2006.

[8] P. Bjesse and K. Claessen, “SAT-based verification without state space traversal,” in *FMCAD*, Nov. 2000.

[9] K. L. McMillan, “Interpolation and SAT-based model checking,” in *CAV*, 2003.

[10] Dong Wang et al., “Formal property verification by abstraction refinement with formal, simulation and hybrid engines,” in *DAC*, June 2001.

[11] J.-H. Jiang and R. Brayton, “Retiming and resynthesis: A complexity perspective,” in *TCAD*, vol. 25, Dec. 2006.

[12] A. Mishchenko, S. Chatterjee, and R. Brayton, “DAG-aware AIG rewriting: A fresh look at combinational logic synthesis,” in *DAC*, July 2006.

[13] A. Kuehlmann and J. Baumgartner, “Transformation-based verification using generalized retiming,” in *CAV*, July 2001.

[14] G. S. Manku, R. Hojati, and R. K. Brayton, “Structural symmetry and model checking,” in *CAV*, July 1998.

[15] A. Kuehlmann, “Dynamic transition relation simplification for bounded property checking,” in *ICCAD*, Nov. 2004.

[16] A. Mishchenko et al., “Improvements to combinational equivalence checking,” in *ICCAD*, 2006.

[17] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, “Robust Boolean reasoning for equivalence checking and functional property verification,” *TCAD*, vol. 21, Dec. 2002.

[18] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an efficient SAT solver,” in *DAC*, June 2001.

[19] K. Ravi and F. Somenenzi, “Minimal satisfying assignments for bounded model checking,” in *TACAS*, 2004.

[20] A. Mishchenko and R. K. Brayton, “Recording synthesis history for sequential verification,” in *FMCAD*, 2008.

[21] P. Bjesse and J. Kukula, “Automatic generalized phase abstraction for formal verification,” in *ICCAD*, Nov. 2005.