Fast Cone-Of-Influence Computation and Estimation in Problems with Multiple Properties*

C. Loiacono and M. Palena and P. Pasini and D. Patti and
S. Quer and S. Ricossa and D. Vendraminetto
Dipartimento di Automatica ed Informatica
Politecnico di Torino - Torino, Italy

J. Baumgartner
IBM Corporation
USA

***Abstract*—This paper introduces a new technique for a fast computation of the Cone-Of-Influence (COI) of multiple properties. It specifically addresses frameworks where multiple properties belongs to the same model, and they partially or fully share their COI. In order to avoid multiple repeated visits of the same circuit sub-graph representation, it proposes a new algorithm, which performs a single topological visit of the variable dependency graph. It also studies mutual relationships among different properties, based on the overlapping of their COIs. It finally considers state variable scoring, based on their own COIs and/or their appearance in multiple COIs, as a new statistic for variable sorting and grouping/clustering in various Model Checking algorithms. Preliminary results show the advantages, and potential applications of these ideas.**

I. Introduction

The Cone of Influence (COI) reduction is a fundamental technique to simplify designs in Model Checking [1]. Given a model (represented as a Finite State Machine), specified using some state variables, the COI reduction simplifies the size of the model by eliminating redundant variables, i.e., those ones not relevant to the property under verification. Redundant variables are computed from the dependency graph of state variables (the transition relation or the transition function). The technique is also related to slicing [2], and to localization reduction [3].

In this paper, we address frameworks where repeated COI computations are required for multiple properties and/or graph nodes within a single model, to avoid potentially quadratic slowdown. Moreover, we wish to use COI information as property and/or variable scoring heuristics in various Model Checking algorithms.

Though this paper is obviously related to all past work on COI reduction, ours is more specifically focused upon the efficient verification of models with *multiple properties*. In this context, each property can be originally provided as a user-specified proof obligation, or automatically generated by a model checking algorithm such as property decomposition [4] or speculative lemma generation (e.g., miters in sequential redundancy removal) [5].

On the other hand, we share the same background as papers studying variable and function scoring, sorting, and grouping for various purposes, such as:

- Static variable ordering heuristics, transition relation sorting and clustering, and scheduling for early quantification in BDD-based symbolic Model Checking [6], [7].
- Static initial variable order for SAT-solver heuristics [6].
- Variable scoring and grouping for abstraction [8].

The main contributions of our work are the following:

- A procedure to compute (or estimate) multiple COIs on the same model using a single-pass vs. multiple-pass graph traversal algorithm.
- A study on how to exploit COI-related statistics (exact or approximated) for heuristic sorting and grouping of state variables and properties within various Model Checking algorithms.

II. Background

The sequential synchronous systems we address are usually modeled as Finite State Machines (FSMs). An FSM is a 6–tuple $(S, I, \Sigma, \lambda, T, O)$, where: S is the set of states, $I \subseteq S$ is the set of initial states, Σ is the input alphabet, λ is the output alphabet, $T \subseteq S \times \Sigma \times S$ is the transition relation between the states, and $O \subseteq S \times \Sigma \times \lambda$ is the output relation.

Let V, V' and W be the sets of present state, next state and primary input variables, respectively. A set of current (next) states is expressed by a state predicate $S(V)$ ($S(V')$). We assume that the transition relation $T(V, W, V')$ is given by a *circuit graph*, with state variables mapped to *latches* and relations implemented using logical *gates*. Present and next state variables correspond to latch outputs and inputs, respectively. The transition relation can be described by a set of equations $v'_i = \delta_i(V, W)$, for each $v'_i \in V'$, where δ_i is a Boolean function [1].

We adopt a C-like notation, where || indicates logical OR and | bit-wise OR.

A. COI Computation

Suppose we are given a set of variables $\widehat{V} \subseteq V$ that are of interest with respect to the property P, i.e., $P = P(\widehat{V}, W)$. The COI of $\widehat{V}$ (denoted $COI(\widehat{V})$) is the minimal set of variables such that

- $\widehat{V} \subseteq COI(\widehat{V})$
- If for some $v_i \in COI(\widehat{V})$, its δ_i depends on v_j, then $v_j \in COI(\widehat{V})$

* This work was supported in part by SRC contract 2012-TJ-2328.

A COI-reduced FSM is obtained by removing all elements which reference state variables outside of $COI(\widehat{V})$.

The standard algorithm for computing $COI(\widehat{V})$ works on the variable dependency graph, i.e., a bipartite graph where V and V' variables are the nodes, and all (v_j, v'_i) edges represent a dependency of next state variable v'_i upon present state variable v_j. For each couple of present and next state variable, an edge (v'_i, v_i) is also added. These edges represent the synchronous transfer of data from present states to next states. The algorithm basically implements a backward traversal of the graph, starting from all variables in $\widehat{V}$. The final $COI(\widehat{V})$ is the subset of the reached V nodes.

The process is linear in the number of edges in the graph. When operating on a gate level netlist representation of a circuit, it is linear in the number of interconnections between gates.

III. Computing Multiple COIs

Though computing each COI has a linear-time solution, this process may become computationally expensive when *repeated* COI evaluations are necessary. For example, when multiple properties are to be verified, and each requires an *independent* COI computation, the base algorithm of Section II would need to be applied repeatedly. This entails obvious overhead when multiple properties have overlapping COIs due to sub-graph retraversal. In such cases, the overall COI computation process may degrade to requiring quadratic resources. This may become prohibitive for large models with many thousands of state variables and gates, and a proportional number of properties.

Let us identify as COI *target* a state variable or gate for which we need to compute a COI. We denote the i-th target with t_i, and the set of all targets as $T = \{t_0, ..., t_n\}$. Let $\widehat{V}_i$ be the set of support variables of t_i. We extend the dependency graph by adding node t_i, and an edge from every variable in $\widehat{V}_i$ to t_i. The *multiple COI problem* we are solving can thus be expressed as a mutual reachability problem between each state variable in V and each target T.

The memory-vs-time trade-off is a common decision faced in multiple graph reachability queries arising in numerous application domains, such as office systems, software management, geographical navigation, and ontology queries. Our approach follows the *graph labeling* approach [9], [10], in which graph nodes are assigned labels such that, after labeling, the mutual reachability between nodes can be decided by inspecting labels alone.

Figure 1 shows the pseudo-code, and Figure 2 a circuit example, of our algorithm.

We start by associating with all nodes a *visited flag* and a *bit array encoding*, i.e., a *bitmap*, where the i-th bit correlates to the i-th present state variable v_i. Our bitmaps thus have one bit per state variable. The bitmap associated with node $n_i \in V \cup V'$ in the dependency graph is denoted $BMP(n_i)$. The set of state variables in the COI of a given node correlates to the 1 bits in its bitmap.

Initially, all visited flags are initially set to *false*, and all nodes except present state variables are labeled with a 0 bitmap. V nodes are labeled with a one-hot encoding of their variable index. I.e., $BMP(v_i) = OneHot(i)$ where $OneHot(0) = \ldots 001$, $OneHot(1) = \ldots 010$, etc.

For each target t_i we perform a backward depth-first traversal of unvisited nodes. Notice that "backward" refers to the direction followed for edges in the dependency graph, even though bitmaps are propagated in the forward direction. For each node, we set the visited flag to true, and we recur on all adjacent fanin nodes. Bitmaps are forward propagated, i.e., whenever node n_j is reached by node n_i (thus following the edge (n_j, n_i) in the reverse direction), the label of n_j is bitwise ORed with the label of n_i:

$$BMP(n_j) = BMP(n_j) \mid BMP(n_i)$$

The topological order followed by the DFV guarantees that labels fully represent COI dependencies.

```
BmpFwdPropagate (DG, V)
    clear all visited flags
    clear all node bitmaps
    foreach v_i ∈ V
        BMP(v_i) = OneHot(i)
    foreach t_i ∈ T
        if t_i is not visited
            BMP(t_i) = DFV (t_i)

DFV (n)
    set visited flag of n
    foreach v in the adjacent list of n
        if v is not visited
            BMP (v) = DFV (v)
            BMP (n) = BMP (n) | BMP (v)
```

Fig. 1. Pseudo-code for the Bitmap COI Algorithm

The left-hand side of Figure 2 shows a sequential circuit on which this algorithm is applied starting from all next state variables V'. The upper dotted box represents the combinational single time-frame dependency among variables in V' and variables in V. The lower solid box indicates memory elements (flip-flops). The corresponding dependency graph is reported on the right-hand side of Figure 2. Dotted edges represent present-state vs next-state dependency, whereas solid edges indicate all dependencies through the combinational network. Bitmaps are reported for each node: The initial value before and the final value after the corresponding black arrow. If the v'_1 node is initially selected, a single depth-first visit traverses all nodes following the dotted curve, and it evaluates all labels in a single depth-first traversal.

Notice that it is possible to adopt a second approach, dually proceeding in the forward direction. We can do that by transposing the graph, and starting from V nodes towards the targets. In this case all targets are initialized with a proper one-hot encoding, and the algorithm ends by labeling each state variable with the targets including that variable in their COI. In this case the number of bits of each bitmap is equal to the number of targets t_i.

Notice that the algorithm has a time cost and a memory cost that are linear and quadratic, respectively, in the number

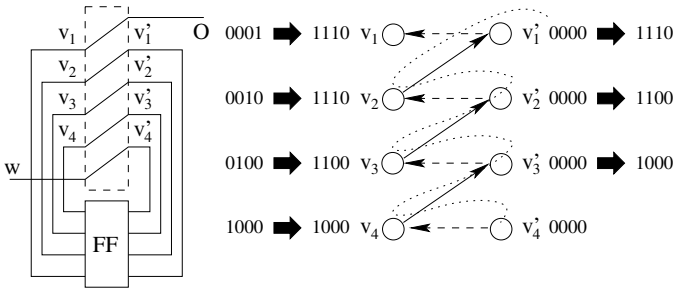

Fig. 2. A graphical representation of the bitmap evaluation

of nodes in the dependency graph.

IV. Approximating and Using COI

The multiple COI computation approach can be applied in all cases where an exact or approximate, COI evaluation is needed. We briefly overview some of most promising optimizations, approximations, and applications of COI computation.

A. Strongly-Connected Component Bundling

Many circuits comprise one or more strongly-connected components (SCCs), within which each node may reach each other node. SCCs may be identified using Tarjan's linear-time algorithm. Each SCC can be collapsed into a single representative node or each state variable within each SCC may share the same bitmap bit, reducing memory requirements.

B. Approximated Computation

If the memory or time costs of computing bitmap labels remain too high, computing COI estimates is an interesting possibility. To compute over-estimated COIs it is possible to adopt compacted bitmaps, i.e., signatures which represent a set of state variables using a smaller number of bits, even for state variables not within the same SCC. To compute under-estimated COIs it is also possible to stop the recursive DFV labeling procedure after a certain number of recursion levels.

C. Sorting and Grouping/Clustering Properties

A first application of multiple COIs is the verification of multiple properties of the same model (see [4], [11]). Whenever the number of properties is high (tens to hundreds), COIs can be exploited for:

- Sorting properties based on COI size, from small to large. The underlying intuition is that smaller COIs often correspond to easier properties, so those properties should be verified first.
- Grouping/clustering two or more properties into a single verification problem as an intermediate option between grouping *all* properties together, and no grouping whatsoever. Due to the algorithmic overhead of verifying properties on larger models, it is often advantageous to solve unrelated properties independently. However, due to potential incremental reuse of verification resources, it is often more effective to solve two highly-correlated properties concurrently vs. independently. Heuristics for grouping/clustering can exploit indicators of *affinity* among properties, represented by how much their COIs overlap each other.

To support the above applications, we provide two COI-related sets of statistics: Sizes of a COI (i.e. the number of bits set in a bitmask), and an affinity matrix α that for property pairs P_i, P_j (with $i \neq j$), measures the overlap between their COIs, normalized w.r.t. the union of the COIs:

$$\alpha_{i,j}(P) = |COI(P_i) \cap COI(p_j)|/|COI(P_i) \cup COI(p_j)|$$

D. Sorting and grouping/clustering variables

Another potential application for multiple COI analysis is to augment existing algorithms that statically and/or dynamically sort/group state variables, in BDD- and SAT-based Model Checking. Most existing BDD-based Model Checkers exploit heuristics for variable sorting, partitioned transition relation management, and partitioned image computation. Topological graph traversals, support evaluation for next-state functions/variables, as well as the variable dependency matrix, are commonly used for such purposes. Static variable scoring is also important for SAT-based techniques. We propose to compute COIs of individual state variables, and to consider COI statistics as a base for the above listed tasks. Specifically, for each state variable, we can consider the size of its COI, as well as the total number of COIs that include the variable. We can finally define a variable affinity matrix, similar to the one previously defined for properties.

V. Experimental Results

In order to analyze the full potential benefit of our methodology we run experiments on the multi-property suite of the HWMCC'11 [12]. It consists of 24 benchmarks a some of them with more than 1000 properties.

Our prototype ran an Intel i7 $860470/2010$ Workstation with 8 MB cache memory, a clock speed of 2.8G Hz (disregarding Turbo Mode), 4 cores, 8 threads, 8 GBytes of main memory DDR III 1333, and hosting a Ubuntu 12.04 LTS Linux distribution.

Figure 3 plots the maximum depth of COIs, i.e. how many next-to-present state variable edges are included in longest *shortest path* connecting a COI variable to a target in the dependency graph. The depth is an indication of how much of sequential information is captured in a COI, with respect to a single time-frame dependency matrix.

Figure 4 shows the COI size for all properties of these benchmarks. A more accurate analysis of those COIs intersections, which we do not report for sake of space, would show that COIs are often diversified, meaning that different properties actually "observe" different parts of a given model. This also implies that COI information can be valuable to sort and/or group properties.

Figure 5 plots a 3-D graph of the correlation among COIs of the same design. Both low and high correlation data can be useful to provide insight into the model under verification, as well as to enable meaningful property clustering.

Figure 6 plots the CPU time (in seconds) necessary to compute the COI with a standard algorithm versus the time

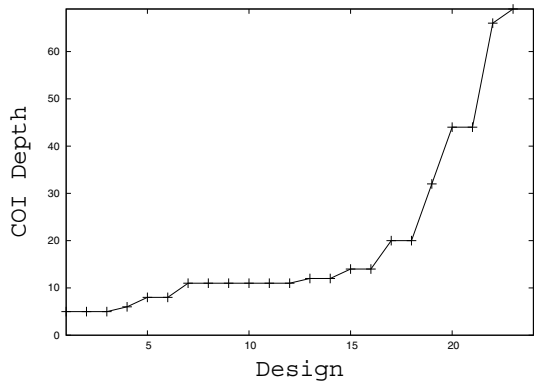

Fig. 3. Maximum depth of COI

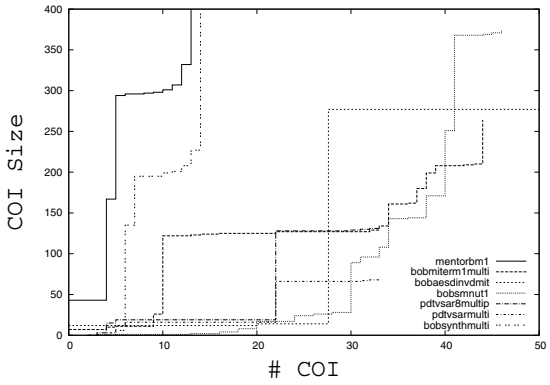

Fig. 4. COI size for all considered benchmarks

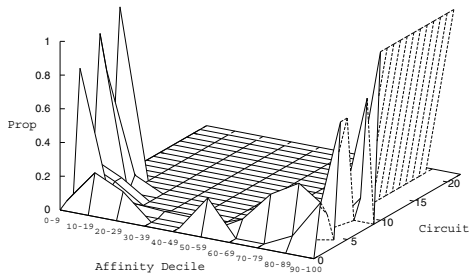

Fig. 5. COIs Correlation.

required by the proposed methodology. Several cases show that the memory-vs-time trade-off of bitmaps and a single graph traversal is, at least, one order of magnitude faster than an approach requiring multiple traversals.

VI. Conclusions

The work introduces new techniques for a fast computation, estimation, and application of the Cone-Of-Influence (COI) of multiple properties. In order to avoid multiple repeated traversals of the same sub-circuit, our algorithm is based on graph node labeling, and it performs a single visit of the variable dependency graph. Its cost is linear in time and

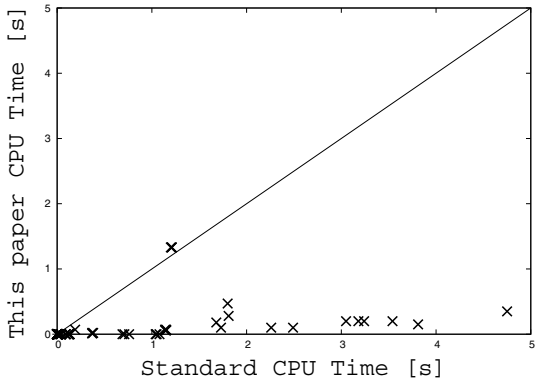

Fig. 6. CPU Time: Standard Method versus Proposed Algorithm

quadratic in memory in terms of the graph size. We also suggest techniques to estimate COI size, to analyze mutual relationships among different properties and among properties and variables. Results are promising on cases where multiple large properties have to be verified or several proof obligations are generated run-time during the verification process.

Among the future works, we envisage the application to the presented ideas to similar computations, such as evaluating the node size of representations sharing sub-graphs, e.g., BDDs or AIGs-based representations.

References

[1] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, 2000.

[2] G. J. Holzmann, *The SPIN Model Checker*. Addison Wesley Professional, 2004.

[3] R. P. Kurshan, "Computer Aided Verification of Coordinating Processes," in *Princeton University Press*, Princeton, NJ, 1995.

[4] G. Cabodi and S. Nocco, "Optimized Model Checking of Multiple Properties," in *Proc. Design Automation & Test in Europe Conf.* Grenoble, France: IEEE Computer Society, Apr. 2011.

[5] J. Baumgartner, H. Mony, A. Mishchenko, and R. K. Brayton, "Speculative reduction-based scalable redundancy identification," in *Proc. Design Automation & Test in Europe Conf.* IEEE Computer Society, Apr. 2009, pp. 1674–1679.

[6] F. A. Aloul, I. L. Markov, and K. A. Sakallah, "MINCE: A Static Global Variable–Ordering for SAT and BDD," in *Proc. Int'l Workshop on Logic Synthesis*, Lake Tahoe, California, May 2001.

[7] I. H. Moon and F. Somenzi, "Border-Block Triangular Form and Conjunction Schedule in Image Conjunction," in *Proc. Formal Methods in Computer-Aided Design*, vol. 1954. Austin, TX, USA: Springer, 2000.

[8] H. Cho, G. D. Hatchel, E. Macii, M. Poncino, K. Ravi, and F. Somenzi, "Approximate Finite State Machine Traversal: Extensions and New Results," in *Proc. Int'l Workshop on Logic Synthesis*, Lake Tahoe, California, May 1995.

[9] H. Wang and H. He and J. Yang and P. S. Yu and J. X. Yu, "Dual Labeling: Answering Graph Reachability Queries in Constant Time," in *Proc. 22nd International Conference on Data Engineering (ICDE'06)*. IEEE Computer Society, 2006, p. 75.

[10] Y. Chen and Y. Chen, "An efficient algorithm for answering graph reachability queries," in *IEEE 24th International Conference on Data Engineering*, ser. ICDE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 893–902. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2008.4497498

[11] A. Biere and K. L. Claessen, "The Hardware Model Checking Competition Web Page, http://fmv.jku.at/hwmcc11," 2011.

[12] A. Biere and T. Jussila, "The Hardware Model Checking Competition Web Page, http://fmv.jku.at/hwmcc."