Optimized Model Checking of Multiple Properties*

G. Cabodi and S. Nocco
Dipartimento di Automatica e Informatica
Politecnico di Torino - Torino, Italy
Email: {gianpiero.cabodi, sergio.nocco}@polito.it

***Abstract*—This paper addresses the problem of model checking multiple properties on the same circuit/system. Although this is a typical scenario in several industrial verification frameworks, most model checkers currently handle single properties, verifying multiple properties one at a time. Possible correlations and shared sub-problems, that could be considered while checking different properties, are typically ignored, either for the sake of simplicity or for Cone-Of-Influence minimization. In this paper we describe a preliminary effort oriented to exploit possible synergies among distinct verification tasks of several properties on the same circuit. Besides considering given sets of properties, we also show that multiple properties can be automatically extracted from individual properties, thus simplifying difficult model checking tasks. Preliminary experimental results indicate that our approach can lead to significant performance improvements.**

I. Introduction

Typical industrial verification frameworks are characterized by the need to prove a number of properties on the same model. As far as formal verification is concerned, most of the research and implementation efforts in Bounded Model Checking (BMC) and Unbounded Model Checking (UMC) have addressed the problem of a single property verification.

Although this is by far the most common case considered in all state-of-the-art Model Checking approaches, one could find an attractive perspective in the idea to verify several properties together. For instance, several properties can (fully or partially) share their "cone of influence" (COI) in the system. Hence, proving all of them in a single model checking session could yield significant speed-ups, coming from shared sub-problems and proper learning techniques. This intuition has recently inspired a few works [1], [2], addressing BMC and inductive UMC, with optimizations ranging from the simple idea to conjoin (by Boolean "and") several properties together to the use of incremental SAT across proofs of different properties.

We extend the mentioned approaches in two directions:

- managing and organizing sets of properties to be verified. We discuss how to group and sort given sets of properties on the same model. We also address the issue of partitioning single properties into sets of properties, i.e., uncovering multiple (simpler) properties, whenever they are embedded into individual properties.
- exploring new techniques to exploit mutual synergies among proofs of different properties. We extend the idea of clause-based learning mechanism, as exploited in [1], [2], to reachable state sets, invariants and model transformations/simplifications, generated as a sub-product of individual property verification tasks.

Henceforth, we show how to automatically extract from the design circuit a set of *assumptions*, implied by the properties that have already been proved, to be exploited (e.g., as care sets or as external constraints) when verifying subsequent properties. In particular, we discuss how the model checking procedures can be optimized by exploiting such a set of constraints/assumptions.

As this work is at its preliminary steps, our main target is to describe its potential improvements, and show its feasibility on selected experimental cases. To the best of our knowledge, the number of scientific works specifically addressing this issue is very limited, although the problem we target is very important at an industrial setting.

The paper is organized as follows. Section II introduces our notation and a few preliminaries. Section III discusses the possible strategies for model checking a set of properties, highlighting their advantages and disadvantages. In particular, it also motivates the approach we propose. Section IV describes how we organize multiple properties for the verification, and Section V shows how useful information can be generated from a model checking task and exploited during the next ones. Finally, Section VI presents the experiments we performed, and Section VII concludes the paper.

II. Preliminaries

We address systems modeled by labeled state transition structures and represented implicitly by Boolean formulas. From our standpoint, a system M is a triplet $M = (S, S_0, T)$, where S is a finite set of states, $S_0 \subseteq S$ is the set of initial states, and $T \subseteq S \times S$ is a total transition relation. The system state space is encoded through an indexed set of Boolean variables $V = \{v_1, \ldots, v_n\}$, so that a state $s \in S$ corresponds to a valuation of the variables in V and a set of states can be implicitly represented through a Boolean formula over V.

Given a system M, a state path of length k is a sequence of states $\pi = (s_0, \ldots, s_k)$ such that $T(s_i, s_{i+1})$ is true for all $0 \leq i < k$. A state set R is said to be reachable if there exists a path of any length connecting a state in S_0 to another state in R. An over-approximation R^+ of a set of states R is any state set including R: $R \subseteq R^+$.

Given a system M, we assume that $P = \{p_1, ..., p_m\}$ is a set of m invariant properties to be verified over M. For each invariant $p_i \in P$, the model checking problem can be

*This work was supported in part by SRC contract 2009-TJ-1968.

described as exploring the set of states reachable by M, while verifying whether p_i holds. If p_i is true for all such states, the invariant holds in M. On the other hand, if there exists a reachable state such that p_i is false, then the invariant does not hold for M. To this respect, several algorithms have been proposed over the years, including BDD-based traversals [3], simple induction [4] and inductive invariants [5], interpolant-based verification [6], etc. In the sequel, when talking about a single property verification, we implicitly refer to a model checking task performed through any of these approaches.

III. Multiple Property Verification: Divide-and-Conquer or All Together?

A. Single Property Verifications

The main motivations for considering and model checking properties only one at a time are the classical ones for divide-and-conquer approaches: the data structures and the computational effort for verifying one property can be smaller (in some cases much smaller) than those required by the simultaneous verification of multiple properties.

More in details, and more concretely, here are some considerations in favour of single property verification.

- The cone of influence (COI) of a single property is a subset of the COI of a set of properties. Most Model Checking techniques (both BDD- and SAT-based), are computationally very sensitive to the COI size. As properties often consider localized behaviors and/or sub-circuits, COI reductions can be relevant, and the computational cost of verifying a single property can be much lower than verifying all properties together.
- Although simultaneous verification of multiple properties could be attractive, due to the potential sharing of common sub-problems, the cumulative cost of individual checks (with repeated sub-problems) could still be advantageous, due to the lower complexity obtained.
- In case of failures, an analysis of single properties is necessary, in order to discriminate the failed ones and to generate individual counterexamples.

B. Simultaneous Multiple Property Verification

In order to model check a set P of properties at the same time, a very straightforward possibility [7] is to generate and verify a "grouped-property" p, given by the Boolean conjunction of all properties in P: $p = \bigwedge_i p_i$.

The most interesting aspect of simultaneous verification is the opportunity to share sub-problem solutions and learning. As the underlying circuit model M is common to all properties, this is an attractive chance, especially for properties that actually share most of their COI. The works presented in [1], [2] all reap most of their benefits from the above observation.

A second (often disregarded) advantage is inherently tied with simultaneous property verification. Each property p_i can be considered as a specification of the correct behavior for the model under verification. Thus, the grouped property p, that encompasses all single properties, is the most complete specification available, that can help some UMC techniques in tightening and/or constraining their search space when looking for property failures. This is particularly true when properties share a good portion of their COI.

Overall, we can conclude that simultaneous property verification can be particularly beneficial in case of COI sharing.

C. Our Approach: Single Property Verification with Learning

As the main drawback of the simultaneous multiple checks is the cumulative growth of the COI, a possible alternative is to individually verify single properties, and to inherit/propagate some learning across individual verifications. Although this idea was present in part in [1], [2], due to the shared clause database (and related learning), we aim at exploring fully new techniques, specifically oriented to UMC, working at a higher level of abstraction than the SAT engine:

- Given a set of properties, we classify properties according to their mutual affinity, or potential ability to share sub-problems and learned data.
- We group properties in subsets. Properties in a group are verified simultaneously. The groups are sorted by increasing expected verification effort.
- We modify UMC algorithms in order to:
  - Extract learned data from successful verifications (e.g. reachable states, inductive invariants, circuit transformations/simplifications).
  - Exploit learned data coming from the verification of other properties. We specifically deal with the problem of how to project constraints and reachable state sets from a given COI onto another one, with potentially different sets of support variables.

The overall verification of multiple properties is thus organized through a compositional-like scheme, where we sequentially prove individual properties or groups of properties, and exploit mutual propagation of learned information.

Finally, we extend our approach beyond the case of a "given" set of properties over the same system M, by addressing the potential decomposition of a single "monolithic" property into a set of sub-properties. Starting from a circuit representation of an invariant property (this is actually a common situation, as invariants are often specified as additional outputs of M), it turns out that the property may usually be expressed as the conjunction of several terms. This is true even if the property corresponds to the output of a Boolean "or" gate, because it is possible to rewrite it as a conjunction by applying simple logic rules (e.g., if $p = z \vee (x \wedge y)$, then $p = (z \vee x) \wedge (z \vee y)$ by distribution). Thus, reversing the reasoning of Section III-B, the property can be viewed as a "grouped" specification, with each term of the conjunction providing a separate invariant.

IV. Organizing Multiple Properties

As previously mentioned, the properties in P to be verified over model M are initially classified according to their mutual affinity. Several criteria could be used for affinity measures (e.g., the amount of circuit sharing). In our current implementation, we limited ourselves to considering the sets of support variables V_i in the cone of influence of each p_i.

More specifically, given two invariants p_j and p_k, the affinity α between them is computed as: $\alpha = |V_j \cap V_k|/|V_j \cup V_k|$. If α is larger than a given threshold, than properties p_j and p_k are grouped together and verified simultaneously, as described in Section III-B.

It should be finally observed that, before starting the model checking tasks, the grouped properties are sorted by increasing number of variables in their cone of influence. The underlying motivation is an attempt to attack property checks by increasing complexity, that is often related to the COI size (and set of support variables). Although different solutions could be explored, this is a good starting point to study the simplification impact of learned data coming from previous verifications.

V. Exchanging Learning among Related Property Verifications

The overall organization of our approach follows the "compositional verification" paradigm [8]. Compositionality is traditionally used whenever a large model under verification is decomposed in sub-models, to be verified under an assume-guarantee reasoning. In our case a single model is shared, whereas the properties, and their COIs, can be considered as overlapping specifications to be compositionally verified.

Although a circular assume-guarantee scheme could be used, where, while verifying property p_i, all the other not-yet verified ones are assumed true, for practical reasons (and due to the preliminary nature of this work) we prefer here the non-circular approach. Given an order in the set of verified properties, we use it to drive the chronology of proofs. Let p_i be the property currently being verified and $P_{<i}$ the set of all properties proved before p_i. Then, all properties in $P_{<i}$ can be safely assumed as true when verifying p_i.

Our further contribution is to generate an additional set of assumptions $A_{<i}$, tighter than $P_{<i}$ and more powerful in terms of problem simplification, generated as a by-product when proving the properties in $P_{<i}$, and exploited as additional constraints while verifying p_i.

We propose here two kinds of such assumptions: over-approximations of reachable states and previously performed circuit transformations.

A. Reachable States

Most UMC methods are potentially able to generate over-approximated reachable states sets, as a by-product of proving properties. BDD-based traversals (either exact or over-approximate) are the most obvious techniques, but SAT and circuit-based approaches as well can produce valuable approximations:

- Enlarged targets or backward circuit unrollings (starting from the negation of the property), can produce such over-approximations, by simply taking their complement.
- Inductive invariants [5] can be considered as an over-approximate reachable state set R^+ [9].
- Interpolant-based methods [6] incrementally compute an over-approximated states set, typically represented as a disjunction (Boolean "or") of approximate images. Whenever such a method converges proving the property, this set of states is a valid R^+.

All of the above mentioned cases, and potentially other ones, can produce over-approximations of states sets. The big issue here is how to effectively exploit such representations for improved performance.

Whenever verifying property p_i, with given COI$_i$ and state variables V_i, an over-approximation $R^+_{<i}(V_{<i})$ of reachable states coming from previous verifications is expressed in terms of a not fully overlapping set of variables $V_{<i}$. Should we extend COI$_i$ in order to include all $V_{<i}$ variables? In our experience, this is generally not a good choice, as it tends to rapidly (and often unnecessarily) increase COI$_i$. Two alternative solutions are:

1) projection of $R^+_{<i}$ over the V_i variables, by existential quantification of the variables in $V_{<i} \setminus V_i$
2) localization abstraction, which means considering the out-of-COI$_i$ variables as free.

We propose the former solution for BDD-based states sets (where existential quantification is typically available), and the latter one for all other cases.

Once $R^+_{<i}$ is available, it can be used as an external constraint (an assumption) for the verification of p_i. To this respect, see for instance [10], [11].

B. Circuit Transformations

Besides considering a constraint, to be propagated for next verifications, one could consider forwarding equivalence preserving simplifications and/or circuit transformations performed on shared parts of the model under verification.

Many UMC algorithms operate equivalence preserving transformations, aimed at producing a simpler model. For instance, let us mention merging equivalent circuit nodes (under either speculative or non-speculative schemes), and redundancy removal. All such transformations can be immediately transferred across different property verifications, even in the case of not fully overlapping COIs.

VI. Experimental Results

We present some preliminary experimental data, oriented to provide an initial evaluation of our strategy w.r.t. standard model checking approaches. Our prototype implementation ran on a Quad-core i7 workstation, with CPU frequency at 2.8 GHz and equipped with 8 GB of main memory. Since this is still an on-going work, we are not yet able to provide detailed data on all the features we described, nor we could experiment the proposed technique on a wide set of benchmarks.

Here we present a single representative case study, which well witnesses the potential advantages of our approach. The benchmark for this case study is represented by circuit bobsmnut1, submitted by R. K. Brayton at the last Hardware Model Checking Competition [12].

This is presumably an equivalence checking instance, for a circuit characterized by a total of 76 primary inputs and 644 latches, with a single invariant to be verified (specified as the only system output). At the competition, this benchmark was solved by both ABC [13] and ic3 [14], whereas all the other 19 submitted solvers were unable to prove the property within the adopted time and memory limits. This means that the verification of this instance can be actually achieved though some advanced techniques, but certainly this benchmark cannot be classified as "easy".

However, by applying the decomposition described in Section III-C, we discovered that this invariant can be viewed as the conjunction of 46 partitions. The number of state variables in the cone of influence for each partition is detailed in Fig. 1. Interestingly, more than half of the partitions depend on a very small number of variables (less than 30). Furthermore, even the largest partitions include only a part of the total state variables defined in the circuit.

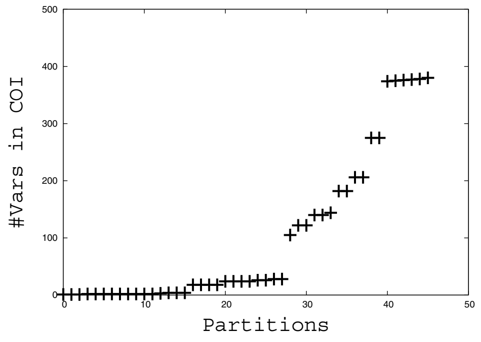

Fig. 1. Number of variables in the cone of influence for every partition in which the single invariant for benchmark bobsmnut1 can be decomposed.

Then, we tried to verify this property according to the "classical" approach, by means of several possible algorithms, and with the proposed technique, with time and memory limits of 2 hours and 2 GB, respectively. The results are summarized in Table I, where each row provides the algorithm we applied and the results we obtained. In particular, inductive invariants is the only technique able to prove the property in the traditional approach, in more than one hour. On the other hand, following our decomposed verification strategy, we were able to obtain the same result in about one minute. This clearly shows the potential of our approach.

TABLE I
PERFORMANCE COMPARISON AMONG DIFFERENT VERIFICATION METHODS FOR THE bobsmnut1 BENCHMARK. ovf MEANS OVERFLOW.

Method	Time [s]	Memory [MB]
Forward BDDs	ovf	179.9
Backward BDDs	ovf	317.0
Simple induction	ovf	432.7
Inductive invariants	4316.8	762.5
Craig interpolants	3225.3	ovf
Our decomposed approach	61.2	120.3

Finally, we also give a few raw data confirming that the decomposition of a single property into a set of properties is not just an unusual possibility: among all the 818 benchmarks considered at the last HWMCC edition, as many as 154 instances could be conjunctively decomposed (directly, without applying two level rewriting rules). Although the set includes many equivalence checking problems (with a single property resembling all the equivalences to be checked), we also found several decompositions for other kinds of properties. In more details, the maximum (minimum) number of partitions we found for a single property is 788 (2), whereas the average number is 37.6.

VII. CONCLUSIONS

This work addresses the problem of checking multiple properties on the same circuit. Although this is a very important issue for industrial settings, most research in the formal verification field only targets single properties. We describe a preliminary effort oriented to exploit possible synergies among different verification tasks of several properties. Furthermore, we show that multiple properties can be automatically extracted from individual properties, thus simplifying difficult model checking tasks.

REFERENCES

[1] Z. Khasidashvili, A. Nadel, A. Palti, and Z. Hanna, "Simultaneous SAT-based Model Checking of Safety Properties," in *Proc. Haifa Verification Conf.*, Nov. 2005, pp. 56–75.

[2] X. Qin, M. Chen, and P. Mishra, "Synchronized Generation of Directed Tests using Satisfiability Solving," in *Proc. Int'l Conf. on VLSI Design*, Jan. 2010, pp. 351–356.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," in *Proc. Symposium on Logic in Computer Science*, Jun. 1990, pp. 428–439.

[4] M. Sheeran, S. Singh, and G. Stålmarck, "Checking Safety Properties Using Induction and a SAT Solver," in *Proc. Formal Methods in Computer-Aided Design*, ser. LNCS, vol. 1954, 2000, pp. 108–125.

[5] P. Bjesse and K. Claessen, "SAT-based Verification without State Space Traversal," in *Proc. Formal Methods in Computer-Aided Design*, ser. LNCS, vol. 1954, 2000.

[6] K. L. McMillan, "Interpolation and SAT-based Model Checking," in *Proc. Computer Aided Verification*, ser. LNCS, vol. 2725, Jul. 2003, pp. 1–13.

[7] R. Fraer, S. Ikram, G. Kamhi, T. Leonard, and A. Mokkedem, "Accelerated Verification of RTL Assertions based on Satisfiability Solvers," in *Proc. High-Level Design Validation and Test Workshop*, Oct. 2002, pp. 107–110.

[8] S. Berezin, S. Campos, and E. M. Clarke, "Compositional Reasoning in Model Checking," in *Proc. COMPOS*, 1998, pp. 81–102.

[9] M. L. Case, A. Mishchenko, and R. K. Brayton, "Inductively Finding a Reachable State Space Over-Approximation," in *Proc. Int'l Workshop on Logic Synthesis*, May 2006.

[10] G. Cabodi, P. Camurati, L. Garcia, M. Murciano, S. Nocco, and S. Quer, "Speeding up Model Checking by Exploiting Explicit and Hidden Verification Constraints," in *Proc. Design Automation & Test in Europe Conf.*, Apr. 2009, pp. 1686–1691.

[11] G. Cabodi, L. A. Garcia, M. Murciano, S. Nocco, and S. Quer, "Partitioning Interpolant-based Verification for Effective Unbounded Model Checking," *IEEE Trans. on Computer-Aided Design*, vol. 29, no. 3, pp. 382–395, 2010.

[12] A. Biere and T. Jussila, "The Model Checking Competition Web Page, http://fmv.jku.at/hwmcc10," 2010.

[13] A. Mishchenko, "ABC: A System for Sequential Synthesis and Verification, http://www.eecs.berkeley.edu/∼alanmi/abc/," 2005.

[14] A. Bradley, "ic3: SAT-Based Model Checking without Unrolling http://ecee.colorado.edu/∼bradleya/ic3/," 2010.