Scalable Verification of Designs with Multiple Properties

Rohit Dureja and Kristin Yvonne Rozier

August 24, 2019

Scalable Verification of Designs with Multiple Properties

Rohit Dureja and Kristin Yvonne Rozier
Iowa State University, Ames, USA

***Abstract*—Many industrial verification tasks entail checking a large number of properties on the same design. Formal verification techniques, such as model checking, can verify multiple properties concurrently, or sequentially one-at-a-time. State-of-the-art verification tools do not optimally exploit subproblem sharing between properties, leaving an opportunity to save considerable verification resources. A significant need therefore exists to develop efficient and scalable techniques that intelligently check multiple properties by utilizing implicit inter-property logical dependencies and subproblem sharing, and improve tool orchestration. We report on our investigation of the *multi-property model checking* problem, and discuss research results, and highlight future research directions.**

I. Introduction

The formal verification of a hardware and/or software design often mandates checking a large number of properties. For example, equivalence checking compares pairwise equality of each design output across two designs, and entails a distinct property per output. Functional verification checks designs against a large number of properties ranging from low-level assertions to high-level encompassing properties. Design-space exploration via model checking [1] verifies multiple properties against competing system designs differing in core capabilities or assumptions. However, most research and implementation efforts in formal verification address the problem of single-property verification, and multiple properties are verified concurrently, or one-at-a-time. Possible inter-property relationships and shared subproblems are typically ignored, leaving an opportunity to save considerable verification resources. For example, consider properties φ_1, φ_2, and φ_3. If $\varphi_2 \rightarrow \varphi_1$, then, φ_1 automatically holds if φ_2 holds for a design, i.e., we do not verify φ_1 if φ_2 holds. Moreover, the state-space information learned by verifying φ_2 (e.g., inductive clauses in IC3) can be reused for verifying φ_3 to save resources.

We develop efficient and scalable techniques for automatic verification of multiple properties. Our work focuses on three primary aspects:

1) *Inter-property relationships* [2] – utilizes logical dependencies to minimize the number of model checking runs,
2) *Information reuse* [3] – learned state-space information of the design is reused across different properties, and
3) *Improved orchestration* [4] – properties with nearly-identical cone-of-influence (COI) are verified together.

We establish inter-property relationships by performing fast LTL satisfiability checks on implications between properties, e.g., if $\varphi_2 \rightarrow \varphi_1$ is unsatisfiable, then φ_1 holds whenever φ_2 holds for a design. Model checking algorithms, like IC3, incrementally refine the approximate reachable state-space by blocking and propagating inductive clauses. Most of these clauses can be reused across different properties albeit after careful repair (weakening or strengthening). Improved orchestration helps maximize the reuse of such clauses by concurrently verifying properties with nearly-identical COI.

II. Research Results

A. Inter-Property Relationships

Our algorithm, D^3 [2] preprocesses the set of properties to find pairwise logical dependencies. For two LTL properties φ_1 and φ_2, dependencies can be characterized in four ways: $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \rightarrow \neg\varphi_2)$, $(\neg\varphi_1 \rightarrow \varphi_2)$, and $(\neg\varphi_1 \rightarrow \neg\varphi_2)$. The pairwise dependencies are stored in a property table as shown in Fig. 1. Each row in the table is a *(key, value)* pair. If

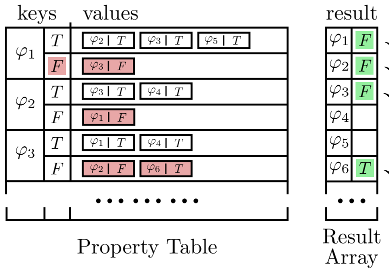

Fig. 1. The D^3 algorithm [2] finds inter-property logical dependencies via LTL satisfiability checking to minimize number of model-checking runs.

$(\varphi_1 \rightarrow \varphi_2)$ is unsatisfiable, then the table contains a row $(\varphi_1 : T, \varphi_2 : T)$. The dependencies help in minimizing the number of model-checking runs required to check all properties in the set. For example, if $M \not\models \varphi_1$, then $M \not\models \varphi_2$, $M \not\models \varphi_3$, and $M \models \varphi_6$; no model-checking run is required for φ_2, φ_3, and φ_6. For Boeing-WBS designs [5], D^3 finds dependencies between 220 properties in a few minutes, and runs the model-checker for $<$10% of the properties for each design.

B. Information Reuse

Given a model M and property φ, the IC3 algorithm incrementally generates an inductive strengthening of φ to prove whether $M \models \varphi$. It maintains a sequence of frames $F_0, \ldots, F_i$ such that F_i is a conjunction of inductive clauses and represents an over-approximation of states reachable in up to i steps. Our algorithm, FuseIC3 [3] sequentially checks each property by reusing information: reachable state approximations, counterexamples, and invariants, learned in earlier runs to reduce total checking time. When the stored information cannot be reused directly, FuseIC3 repairs and patches the information using an efficient algorithm. It adds "just enough" extra information to the saved reachable states, i.e., literals to

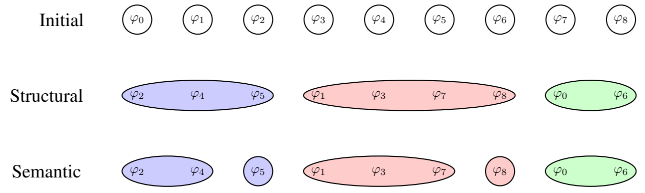

Fig. 2. Orchestration for multi-property verification: structural grouping followed by semantic refinement. Properties in a group are verified concurrently.

inductive clauses, to enable reuse. Our experiments on real-life challenging benchmarks demonstrate that FuseIC3 is on-average 4.39× faster than checking properties without reuse.

C. Improved Orchestration

It is well known [6] that grouping saves substantial resource by concurrent verification of high-affinity properties in a group. However, no scalable online grouping procedure exists. In [4], we present a near-linear runtime, fully-automated algorithm to partition properties into provably high-affinity groups based on structural COI similarity. COI support information is maintained as bitvectors, and grouping is performed in three configurable levels based on: identical COI, strongly-connected components (SCC) in the COI, and Hamming distance. Fig. 2 shows our two-step orchestration: structural grouping followed by semantic partitioning. We advance state-of-the-art in localization by providing an optimal multi-property solution that offers on-average 4.8× end-to-end verification speedup.

III. Related Work

The framework of *local* and *global* proofs [8] has been used to derive a "debugging set" of properties to fix before verifying others, implying a property ordering similar to property dependencies found by [2]. Methods to incrementally reuse information across multiple properties accelerate specific algorithms [9, 10]; these approaches might improve performance of FuseIC3. Algorithms to group properties based on high-level design descriptions extract similarity criteria from high-level information unavailable in low-level designs and benchmark formats such as AIGs [11]. The utility of ideal grouping is experimentally demonstrated in [6] as proof-of-concept using computationally-prohibitive grouping algorithms; we complement their method by providing a fast online grouping procedure, and improved orchestration strategy.

IV. Future Directions

We identify several research directions to further enhance scalability of multi-property verification. Some of these are:

a) Avoiding missed grouping opportunities: Despite the provable threshold of Level-3 grouping in [4], there is some asymmetry in the approach. Two fairly-high-affinity support bitvectors that differ too much in a single segment will not be merged, whereas if the difference was small per-segment with multiple segments differentiated, they may be merged, albeit respecting the quality bound. We plan to investigate methods to mitigate this asymmetry by re-running the analysis on

1) randomized bitvector indices, or
2) different segment-partitioning of the bitvectors.

Clever data structures, such as MA_FSA [7], search for fairly-high-affinity bitvectors that differ in only a few n-bit segments, thereby avoid missed Level-3 grouping opportunities.

b) Structural vs. semantic grouping: The proof or counterexample for properties may only depend on a small subset of their COI. Structural grouping may end up differentiating such properties based on "unimportant" differences in their COI. On the other hand, semantic-based grouping may group such properties based on refinement information (important COI logic) learned during localization abstraction. The question remains: **When to use structural vs. semantic grouping?** It is not trivial to discern what COI subset appears relevant to what property until verification resources are expended. We plan to investigate design preprocessing heuristics and algorithms that help predict the applicability of the two grouping methods without expending significant resources.

c) Enhance grouping scalability: The memory requirement of the support bitvectors remains a bottleneck in end-to-end verification scalability [4]. Future work includes:

1) Compaction of support bitvector bits to improve performance, e.g., support variables present in every property can be projected out of the bitvectors, and
2) Dynamic prefix matching that discounts differences in small SCCs for properties to improve Level-2 grouping.

Lastly, we plan to investigate how semantic information from BMC and IC3 can be used to perform property grouping.

References

[1] M. Gario, A. Cimatti, C. Mattarei, S. Tonetta, and K. Y. Rozier, "Model Checking at Scale: Automated Air Traffic Control Design Space Exploration," in *CAV*, July 2016.

[2] R. Dureja and K. Y. Rozier, "More Scalable LTL Model Checking via Discovering Design-Space Dependencies (D^3)," in *TACAS*, April 2018.

[3] R. Dureja and K. Y. Rozier, "FuseIC3: An algorithm for checking large design spaces," in *FMCAD*, Oct 2017.

[4] R. Dureja, J. Baumgartner, A. Ivrii, R. Kanzelman, and K. Y. Rozier, "Boosting Verification Scalability via Structural Grouping and Semantic Partitioning of Properties," in *FMCAD*, Oct 2019.

[5] M. Bozzano, A. Cimatti, A. Fernandes Pires, D. Jones, G. Kimberly, T. Petri, R. Robinson, and S. Tonetta, "Formal Design and Safety Analysis of AIR6110 Wheel Brake System," in *CAV*, Jul 2015.

[6] G. Cabodi, P. E. Camurati, C. Loiacono, M. Palena, P. Pasini, D. Patti, and S. Quer, "To split or to group: from divide-and-conquer to sub-task sharing for verifying multiple properties in model checking," *STTT*, Jun 2017.

[7] J. Daciuk, S. Mihov, B. W. Watson, and R. E. Watson, "Incremental construction of minimal acyclic finite-state automata," *Computational Linguistics*, vol. 26, no. 1, pp. 3–16, 2000.

[8] E. Goldberg, M. Güdemann, D. Kroening, and R. Mukherjee, "Efficient verification of multi-property designs (The benefit of wrong assumptions)," in *DATE*, March 2018.

[9] Z. Khasidashvili, A. Nadel, A. Palti, and Z. Hanna, "Simultaneous SAT-based model checking of safety properties," in *HVC*, November 2005.

[10] H. Chockler, A. Ivrii, A. Matsliah, S. Moran, and Z. Nevo, "Incremental Formal Verification of Hardware," in *FMCAD*, Oct 2011.

[11] M. Chen and P. Mishra, "Functional test generation using efficient property clustering and learning techniques," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, March 2010.