IC3-Guided Abstraction

Jason Baumgartner, Alexander Ivrii, Arie Matsliah, Hari Mony
IBM Corporation

***Abstract*—*Localization* is a powerful automated abstraction-refinement technique to reduce the complexity of property checking. This process is often guided by SAT-based bounded model checking, using counterexamples obtained on the abstract model, proofs obtained on the original model, or a combination of both to select irrelevant logic. In this paper, we propose the use of bounded invariants obtained during an incomplete IC3 run to derive higher-quality abstractions for complex problems. Experiments confirm that this approach yields significantly smaller abstractions in many cases, and that the resulting abstract models are often easier to verify.**

I. Introduction

Automated property checking techniques hold considerable promise to mitigate what has become one of the most important problems facing the semiconductor industry today: the *verification crisis*. Through the advent of numerous advanced proof, falsification, abstraction and reduction techniques, formal property checking has scaled to the necessary level to address many practical industrial applications, and has become an essential CAD technology. However, many problems remain beyond the capacity of current property checking algorithms, thus continued advances are of critical importance.

Localization is a powerful abstraction technique which reduces the size of a verification problem by replacing gates by *cutpoints*, which act as unconstrained nondeterministic variables. Because the cutpoints may simulate the behavior of the original gates, this approach over-approximates the behavior of a design hence is sound yet incomplete. *Refinement* is used to eliminate spurious failures on the abstract design by eliminating cutpoints which are deemed responsible for the failure. Ultimately, the abstract design is passed to a proof engine. It is desirable that the abstract design be as small as possible to enable more efficient proofs, while being immune to spurious counterexamples.

Various techniques have been proposed to guide the abstraction-refinement process of localization. Most state-of-the-art localization implementations use SAT-based bounded model checking to select the abstract netlist upon which an unbounded proof is attempted, given relative scalability of bounded model checking vs. proof techniques. This abstraction process is either based upon counterexamples obtained on the abstract design [1], based upon proofs obtained on the original design [2], or a hybrid of both [3]. It has been noted that the latter approach tends to yield the smallest abstractions, albeit at the cost of additional runtime. Specifically, the abstraction-refinement process relies upon heuristics and thus may include unnecessary logic, in turn entailing unnecessary proof complexity. In hybrid approaches, one approach is often used to re-process the abstraction computed by the other, in an attempt to eliminate this unnecessary logic. The additional runtime spent on the abstraction-refinement process is often a worthwhile strategy for overall minimal resources.

Various SAT-based unbounded proof techniques have been developed which in cases are very scalable, such as IC3 [4]. One powerful characteristic of IC3 is that when successful, it may yield a proof or counterexample while analyzing only a small approximation of the overall behavior of the design under verification. In the case of a proof, an often-compact inductive invariant is derived. In the case of a counterexample, a directed search from the initial states toward the property found a path, in cases requiring less effort than bounded model checking.

While often efficient, the intrinsic complexity of property checking explains why IC3 often fails to solve a complex problem given practical resources. Furthermore, while IC3 intrinsically attempts to analyze the design in an abstract manner, it is well-known that abstraction may synergistically boost the scalability of various verification algorithms, even approximate ones. In this paper, we seek to exploit this synergy in two ways. **(1)** We extract design insight from an incomplete IC3 run via its (bounded) invariants. **(2)** We use the extracted information to improve the quality of localization, and thereby boost the scalability of subsequent verification algorithms including IC3.

II. Preliminaries

We focus on verification of safety properties on finite state machines (FSMs). An FSM M is a tuple $\langle X, I, T\rangle$, where X is a set of Boolean state variables, such that each assignment $s \in \{0,1\}^X$ corresponds to a state of M, and the predicates $I \subseteq \{0,1\}^X$ and $T \subseteq \{0,1\}^X \times \{0,1\}^X$ define its initial states and the transition relation,

978-0-9835678-2-0/12/$31.00 ©2012 FMCAD Inc.

respectively. A predicate $P \subseteq \{0,1\}^X$ defines a property to be verified on M.

State variables and their negations are called *literals*, and disjunctions (conjunctions) of literals are called *clauses* (*cubes*). A CNF formula is a conjunction of clauses. We follow the standard notation of X' representing next-state variables, and derive CNF formulas from FSMs in a straight-forward way.

A sequence π of states $t_0, \ldots, t_n$ is a *path* if for each $0 \leq i < n$, $\langle t_i, t_{i+1} \rangle \in T$. If t_0 is an initial state, the path is called *initialized*. A state t is *reachable* if there is an initialized path that ends in t. Let R denote the set of all reachable states, and for $k \geq 0$, let R_k denote the set of states reachable by initialized paths of length at most k. The verification objective is to prove $R \subseteq P$.

A CNF formula φ is an *invariant* if $s \in R \implies s \models \varphi$. Furthermore, φ is a *k-step invariant* if $s \in R_k \implies s \models \varphi$. A CNF formula φ is an *inductive invariant* if $I \implies \varphi$ and $\big((s \models \varphi) \wedge (\langle s, s' \rangle \in T)\big) \implies s' \models \varphi$.

If φ is an inductive invariant and $\varphi \implies P$, then $R \subseteq P$ and φ is called an *inductive proof of P*.

III. Generating Hints with IC3

In this section we briefly describe IC3 [4], [5]. This algorithm proceeds by incrementally refining and extending a sequence $\mathcal{F}_1, \ldots, \mathcal{F}_k$ of sets of clauses (termed *bounded invariants*) referring solely to state variables. For simplicity we define $\mathcal{F}_0 = I$. Throughout the run of IC3 the following invariants are preserved:

- $\mathcal{F}_i \Rightarrow \mathcal{F}_{i+1}$, for $0 \leq i \leq k-1$,
- $\mathcal{F}_i \supseteq \mathcal{F}_{i+1}$ as sets of clauses, for $1 \leq i \leq k-1$,
- $\mathcal{F}_i \wedge T \Rightarrow \mathcal{F}'_{i+1}$, for $0 \leq i \leq k-1$,
- $\mathcal{F}_k \Rightarrow P$.

Initially $k = 1$, and is increased whenever $\mathcal{F}_k \wedge T \Rightarrow P'$ holds. Note that each $\mathcal{F}_i$ constitutes an overapproximation of R_i, and $\mathcal{F}_1 (= \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_k)$ implies that P holds in the first k timesteps. While processing bound k, the condition $\mathcal{F}_k \wedge T \Rightarrow P'$ might fail to hold, and then IC3 attempts to propagate additional information to sets $\mathcal{F}_{i \leq k}$ to address this potential failure. If it cannot, this signifies a true counterexample. Otherwise, each added clause is *pushed* to the highest-possible $\mathcal{F}_i$. Some bounded invariants may be determined to be unbounded invariants, and when that set implies P an inductive proof has been completed.

A. Hint Generation

Recall that the conjunction of clauses in $\mathcal{F}_1 \cup \ldots \cup \mathcal{F}_k$ implies that P holds for the first k timesteps. This leads to the idea of creating a localization including only the state variables appearing in these clauses. However, as with traditional localization, some of these state variables may be unnecessary to complete a proof of correctness.

Note that there is additional information one may draw on the relevance of various state variables from the IC3 invariants which may be used to improve the abstraction quality. In particular, for each IC3 invariant c one can consider whether it is an unbounded invariant, the first bound k for which c was introduced, and the maximal frame i that it can be pushed to relative to k. For each state variable, one can analyze the number or the proportion of bounded invariants or clause sets it belongs to. In this section we address how to use the sets $\mathcal{F}_i$ to provide hints to localization on the relative *priority* of various state variables, represented by integers with the convention that a lower number reflects a higher priority.

We experimented with numerous heuristics for assigning priorities, and our best method (**PM1**) is as follows. The priority of each state variable is initially ∞, and may be decreased during the IC3 run. Whenever a new bounded invariant clause c is produced, all state variables referenced by c that have priority ∞ have their priorities updated to the current bound k being processed. I.e., after the run the priority of each state variable represents the earliest bound requiring a corresponding clause for a proof of validity of P, and the clauses for proofs of smaller bounds have higher priorities. Also note that an abstraction including the state variables with priorities $\leq i$ will satisfy P for the first i timesteps.

(PM2) In this variant, the priority of each state variable is initially ∞, and when a new bounded invariant clause c is produced that is "pushable" to frame $i \leq k$, we update the priority of each latch participating in c to $k-i$, unless a smaller number was assigned to it earlier. If c is an unbounded invariant, then the priorities of those latches in c are updated to 0.

(PM3) The priority of each state variable is initially ∞, and whenever a new invariant clause c is produced (bounded or unbounded), we update the priority of each latch participating in c to 0.

(PM2) and **(PM3)** performed generally worse than **(PM1)**, hence we restrict our experimental focus to **(PM1)**. Continued experiments with alternate heuristics is subject of ongoing research.

IV. Localization with IC3 Hints

In this section, we detail how the priorities assigned on the state variables may be used to guide localization. The localization abstraction is done by selection of state variables: if included, its entire next-state function will also be included, else the state variable will be replaced by a cutpoint. The localization is performed by interleaving counterexample based abstraction (CBA) and proof-

based abstraction (PBA) in a bottom-up manner [6]. We start with no state variables included, and perform SAT-based bounded model checking for one timestep on the abstract design. Spurious counterexamples are analyzed and the abstraction is refined by adding state variables which are deemed adequate to rule out the spurious behavior. Once there are no more counterexamples of a given depth, we perform proof-based abstraction for that depth to attempt to eliminate unnecessary logic. We then increment the depth and repeat the process for a configurable resource limit.

The key challenge in localization abstraction is the refinement used to to eliminate the spurious counterexamples. Several approaches have been proposed to minimize the amount of logic added to refute a spurious counterexample, e.g., [7], [1], [8]. Our approach relies upon counterexample trace minimization to minimize the number of cutpoints assigned in a spurious counterexample, using a combination of SAT and ternary analysis [7], [6]. Once the counterexample has been minimized, all cutpoints assigned in that trace are refined.

Even with aggressive trace minimization, this minimization is heuristic and greedy, which may entail a suboptimal abstraction. Furthermore, minimality is not unique, and across multiple refinements it is likely that cumulative refinement choices result in unnecessary logic being included. This is the motivation for using PBA to attempt to further reduce the CBA. We propose to improve this process to a greater degree by using the IC3 priorities to guide the abstraction via one of the following refinement methods.

(RM1) Start CBA with empty abstraction. When refining the abstract model by adding state variables that are assigned in the spurious counterexample during CBA, we only add the subset of those with the highest IC3 priority.

(RM2) In addition to using **(RM1)**, we begin with the abstract model including all state variables of highest IC3 priority vs. starting with an empty abstraction.

The aim of **(RM1)** is to skew CBA to avoid including state variables with lower IC3-assigned priorities in the abstract model. Even though this may result in an increased number of refinements during CBA, our experiments demonstrate that this guidance results in smaller abstractions in practice. **(RM2)** was developed based on the observation that the abstract model often ultimately includes all state variables with the highest IC3-assigned priority regardless; beginning with this set reduces abstraction-refinement runtime due to fewer refinements, and fewer refinements entail fewer heuristic mistakes which bloat abstraction size. We have experimentally found that **(RM2)** yields smaller abstractions, hence we restrict our experimental focus to **(RM2)**.

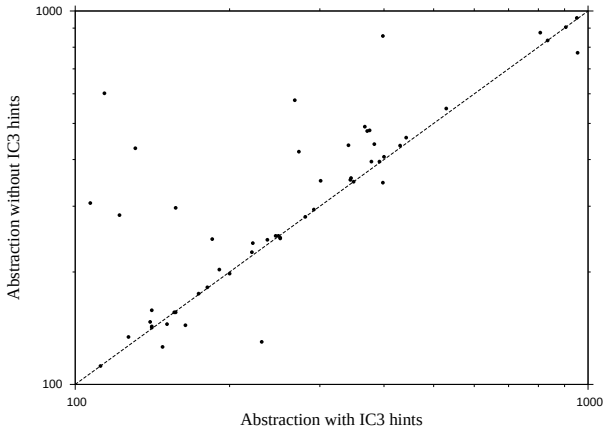

Fig. 1. Number of localized state variables with vs. without IC3 hints

V. Experiments

In this section we present our experiments. All experiments were performed on 2.0Ghz Linux-based machines with 4Gb of RAM, using the techniques presented in this paper as implemented in the IBM verification tool *SixthSense* [9]. We focused upon the single-target benchmarks from HWMCC 2011 [10].

A. Effect on Abstraction Size

This first set of experiments compares abstraction sizes generated with and without IC3 hints. Both include an interleaved CBA/PBA localization for a time limit of 300 seconds. For the IC3 hint-guided abstraction, we run IC3 for 120 seconds to generate the priorities, then proceed with the abstraction-refinement loop **(RM2)**. 294 instances that were solved by IC3 within the 120 seconds limit, or during localization, are excluded from the analysis. In the remaining instances, the total number of state variables in the abstracted models reduces from 47994 to 41036 when using IC3 hints: a cumulative reduction of 14.5%, with median reduction of 6.8%. Figure 1 depicts the reduction on instances with 100 to 1000 state variables after localization.

These results clearly show improved abstraction size in most cases using IC3 hints. There are some which yield worse abstractions, though we note that this is likely inevitable in rare cases given the heuristic nature of abstraction.

B. Effect on IC3 Resources

To demonstrate that the reduced abstract model size improves verification resources, we used IC3 with a 900 second timeout on the localized design. Of the 171 HWMCC 2011 benchmarks which are unsolved by the first 120-second IC3 or during localization, 15

Design	Traditional Abstraction	IC3 Hint-Guided Abstraction
6s9	TO	357s
6s19	TO	519s
6s43	TO	236s
6s50	TO	484s
6s51	TO	198s
bc57sensorsp0	TO	838s
pdtvsarmultip29	TO	473s
pdtswvtma6x6p2	TO	665s
pdtswvtma6x6p1	TO	333s

TABLE I
IMPACT OF HINT-GUIDED LOCALIZATION ON SUBSEQUENT IC3 PROOF RUNTIME

were solved by the heavier-weight IC3 using traditional abstraction-refinement without IC3 hints. In contrast, 24 were solved using abstraction with IC3 hints; a proper superset. While this is only a modest improvement in terms of conclusive solutions, we note that **(1)** 900 second IC3 runtime is not very substantial in terms of a state-of-the-art industrial solver (see the following section), though was motivated by the HWMCC timeout period; **(2)** increasing the number of solved instances by 60% is a nontrivial improvement nonetheless. Table I details the results of these additional solutions.

C. Effect on a State-of-the-Art Verification Tool

Due to space limits, our experiments do not detail many algorithms which are commonplace in a state-of-the-art verification tool. It is well-known that higher-quality abstractions may boost the effectiveness of a variety of these algorithms, such as the reduction capability of retiming [11]. The fact that our limited experiments demonstrate verification benefits for IC3 alone is practically encouraging. When using a larger set of algorithms and a larger runtime, the benefits of higher-quality abstractions becomes even more pronounced. We additionally note that an industrial-strength multiple-algorithm tool likely would use a strategy of running IC3 for a small time-bound early in its strategy, to rule out simpler problems. Extracting localization hints from those runs has virtually no overhead, yet may immediately yield higher-quality abstractions. We also note that we have tuned our bounded model checking-based abstraction over years of industrial application, whereas the use of IC3 hints is much less mature and we are hopeful to discover improved invariants with continued experience.

D. Justification for Effectiveness

A natural question is why IC3 hint-guided localization yields better abstractions than bounded model checking alone. We provide two insights: **(1)** just as CBA and PBA complement each other to yield smaller abstractions, IC3 offers yet another qualitatively-distinct heuristic guidance to complement these techniques. **(2)** There are commonalities in how bounded model checking and IC3 attempt to justify a property failure via backward analysis of the design. Some of these justification attempts identify necessary logic; some spuriously involve unnecessary logic. IC3 in a sense performs additional filtering of the impact of the spurious justification attempts, by requiring forward reachability analysis to generate those invariants only where a potential failure justification may be eliminated. In contrast, justifications that cannot be eliminated by reachability analysis will not yield invariants and thus be less likely to bloat the abstraction.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we demonstrate how the invariants generated by an incomplete IC3 run may be used to generate higher-quality localization abstractions. We furthermore demonstrate that the improved abstractions enhance verification using IC3 itself, and have noted even greater benefits from the higher-quality abstractions using heavier-weight verification flows. Areas of ongoing work include improving heuristics for prioritizing state variables given IC3 information, and to explore methods to prune irrelevant invariants that IC3 is tuned to aggressively propagate. Another direction is to additionally explore the use of IC3 hints on proof-based abstraction.

REFERENCES

[1] P. Chauhan et al., "Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis," in *FMCAD*, 2002.
[2] K. L. McMillan and N. Amla, "Automatic abstraction without counterexamples," in *TACAS*, April 2004.
[3] N. Amla and K. McMillan, "A hybrid of counterexample-based and proof-based abstraction," in *FMCAD*, Nov. 2004.
[4] A. Bradley, "SAT-based model checking without unrolling," in *VMCAI*, Jan. 2011.
[5] N. Eén, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *FMCAD*, Nov. 2011.
[6] N. Eén, A. Mishchenko, and N. Amla, "A single-instance incremental sat formulation of proof- and counterexample-based abstraction," in *FMCAD*, 2010.
[7] D. Wang, P.-H. Ho, J. Long, J. H. Kukula, Y. Zhu, H.-K. T. Ma, and R. F. Damiano, "Formal property verification by abstraction refinement with formal, simulation and hybrid engines," in *DAC*, June 2001.
[8] B. Li and F. Somenzi, "Efficient computation of small abstraction refinements," in *ICCD*, Nov. 2004.
[9] H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman, and A. Kuehlmann, "Scalable automated verification via expert-system guided transformations," in *FMCAD*, Nov. 2004.
[10] Hardware Model Checking Competition 2011. http://fmv.jku.at/hwmcc11.
[11] J. Baumgartner and H. Mony, "Maximal input reduction of sequential netlists via synergistic reparameterization and localization strategies," in *CHARME*, Oct. 2005.