Partitioning Interpolant-Based Verificationfor effective Unbounded Model Checking

Partitioning Interpolant-Based Verification for Effective Unbounded Model Checking

Gianpiero Cabodi, Luz Amanda Garcia, Marco Murciano, Sergio Nocco, and Stefano Quer

***Abstract*—Interpolant-based model checking has been shown to be effective on large verification instances, as it efficiently combines automated abstraction and reachability fixed-point checks. On the other hand, methods based on variable quantification have proved their ability to remove free inputs, thus projecting the search space over state variables. In this paper, we propose an integrated approach which combines the abstraction power of interpolation with techniques that rely on and-inverter graph (AIG) and/or binary decision diagram (BDD) representations of states, directly supporting variable quantification and fixed-point checks. The underlying idea of this combination is to adopt AIG or BDD-based quantifications to limit and restrict the search space and the complexity of the interpolant-based approach. The exploited strategies, most of which are individually well known, are integrated with a new flavor, specifically designed to improve their effectiveness on difficult verification instances. Experimental results, specifically oriented to hard-to-solve verification problems, show the robustness of our approach.**

***Index Terms*—Binary decision diagrams, formal methods, formal verification, model checking, satisfiability, symbolic techniques.**

I. Introduction

ABSTRACTION techniques have been shown to be very effective in hardware formal verification, due to their ability to remove those parts of a system that are not relevant for the verification task. Craig interpolants [1], [2], recently introduced by McMillan [3] in the field of formal verification for unbounded model checking (UMC), represent a fully automated abstraction/refinement method. This approach relies on the ability of modern Boolean satisfiability (SAT) solvers to generate proofs of unsatisfiability. Abstractions are computed as over-approximations of the reachable states, generated from refutation proofs of unsatisfied bounded model checking (BMC) instances. Refinements are achieved through increments of the bounds of the BMC runs, thus iteratively tightening the over-approximations. The process iterates until it converges on a proof, or the property is falsified by a BMC check on the concrete model.

Manuscript received April 8, 2009; revised August 12, 2009. Current version published February 24, 2010. This paper was recommended by Associate Editor V. Bertacco.

G. Cabodi, M. Murciano, S. Nocco, and S. Quer are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin I-10129, Italy (e-mail: gianpiero.cabodi@polito.it; marco.murciano@polito.it; sergio.nocco@polito.it; stefano.quer@polito.it).

L. A. Garcia is with AleniaSIA, Turin I-10146, Italy (e-mail: luz.garcia@polito.it).

Digital Object Identifier 10.1109/TCAD.2010.2041847

Unfortunately, the main strength of interpolants, i.e., the automated exploitation of SAT refutation proofs, is also their weakness, as interpolants can become very large with complex SAT checks.

A careful analysis of the unsolved problems leads to the following observations.

1) Craig interpolants tend to produce highly redundant circuit representations of state sets. Combinational logic optimizers are often not powerful and scalable enough to compact those representations.
2) Abstraction usually reduces the sequential depth of state transition graphs. Nonetheless, over-approximation can trigger state space explorations within unreachable state areas, with direct consequences in terms of visited states and their representations.
3) Alternative circuit and SAT-based state set representations often imply complex quantifier elimination. As a consequence, those techniques often perform well on sub-sets of problems/variables, whereas memory blow-up occurs when working on the complete model.
4) Other verification techniques, e.g., based on binary decision diagrams (BDDs), can outperform interpolants and complete some of the unsolved problems. Nevertheless, the exploitation of possible interactions among alternative methods, and the use of interpolants in cooperation with other verification approaches, are mostly unexplored fields.

A. Proposed Method

Starting from the above listed problems, we propose to complement interpolant-based verification with a set of cooperating techniques, properly selected among state-of-the-art approaches to UMC.

More specifically, we adopt a divide-and-conquer approach, in which different verification methods are used to solve a part of the problem and cooperate to reach the final goal. We incrementally run three main techniques: Craig interpolation, circuit-based quantification, and SAT enumeration. Although a possible methodology is to put those strategies in competition by running them in parallel, we chose a sequential approach with increasing time bounds and preemption. In this way, each strategy potentially contributes to part of the solution. We exploit interpolants for abstraction purposes, as interpolation approaches basically use forward over-approximate (i.e., abstract) reachability, controlled by backward (BMC-like) circuit unrollings. We complete this analysis with partial reacha-

bility and state set manipulation in the backward direction. Backward reachable states either directly support verification checks, or provide search space restrictions for interpolant-based verification. We adopt BDDs whenever the size of the problem is within their typical range of applicability. We resort to circuit-based representations and SAT solvers, for quantifier elimination, in the other cases. Notice that other techniques are active within our framework, such as combinational circuit rewritings, inductive invariants, inductive checks, and so on. Anyhow, we consider those techniques as incremental optimizations, rather than other competing/cooperating tools.

We extensively adopt data partitioning schemes, at the level of both circuit-based and state set representations, to reduce the complexity of expensive operations. In particular, we resort to partitioned interpolations to enhance scalability and robustness.

Overall, our integrated method strongly relies on the general idea of tentatively transforming circuit-based representations into state sets, whenever this is feasible and convenient. When the transformation is not completely possible, because either time or memory resources are exceeded, we use the partial state sets to simplify further verification steps, performed by the same or a different engine. Notice that our methodology is more than just a mix of several competing techniques, such that, for any given problem, one of them wins over the others. Our combined approach often produces better results than any component strategy running separately, as each component method is able to exploit (and contribute to) part of the generated state sets.

To sum up, the main contributions of this paper are:

1) an integrated approach combining interpolant-based verification with quantifier elimination methods, using SAT solvers;
2) a set of partitioning strategies, aimed at decomposing circuit-based representations into disjunctive formulations. Among them, we present novel SAT-based techniques for partial quantification and/or under-estimation of the state space, by means of *lazy quantification*;
3) a partitioned formulation of interpolant-based image computations, working with disjunctively partitioned representations of backward circuit unrollings.

This paper is a revised version of [4], which has been improved from both the theoretical and algorithmic perspectives, then from the experimental results point of view.

The new set of experiments takes into consideration the suite collected for the 2008 Model Checking Competition [16], which includes more than 600 benchmarks. We compare results with and without the proposed methodology, and we are able to show improvements in terms of robustness, scalability, and speed-up.

B. Related Works

Our work follows the UMC approaches based on SAT rather than BDDs.

Inductive proofs are the starting point of the majority of those approaches [5]–[8], all following the seminal work of Sheeran *et al.* [9]. Fixed-point checks are proved inductively, whereas completeness is based on uniqueness constraints, expressing loop-free paths between states. Unfortunately, the largest simple path between any two reachable states can be exponentially larger than the reachability diameter. As a consequence, most of the research in this field has concentrated on finding tight sets of inductive invariants, i.e., over-approximations of reachable states, quite often sufficient for inductive proofs.

In order to attain completeness, it is possible to adopt alternative representations of state sets. Unfortunately, these representations are generally difficult to manipulate within non BDD-based frameworks, as both conjunctive normal form (CNF) and circuit-based representations can lead to memory explosion. Williams *et al.* [10] first adopted Boolean expression diagrams (BEDs), for the removal of quantifiers. Abdulla *et al.* [11] exploited reduced Boolean circuits, i.e., a variant of BEDs, to represent formulas on which they performed existential quantifier elimination through substitution, scope reduction, and so on. McMillan [12], later followed by Kang and Park [13], proposed quantifier elimination through the enumeration of SAT solutions (*all-solutions SAT*). Ganai *et al.* [14] extended the previous approaches by using "circuit co-factoring" to capture a large set of states in every SAT enumeration step. All those methods potentially converge faster than [9], but a widespread applicability is still out of reach, due to the complexity of quantifier elimination which often ends up into exponential state set representations.

Abstraction techniques represent an orthogonal direction to tackle complexity, as they seek and remove those parts of a circuit/system that are not relevant for the proof. Craig's interpolants and their usage within UMC can be framed within this general path of research. According to several researchers, this method is the most promising one, and extensive experimental sessions, see for example [15], [16], showed its robustness and effectiveness. Nevertheless, Jhala and McMillan [17] reported that interpolants are "often unnecessary weak," and introduced an interpolant strengthening method. Marques-Silva [18], [19] showed that interpolants can be reused under reasonable assumptions, allowing a better search pruning along subsequent SAT solver calls. Unfortunately, he was not able to make any conclusion regarding the relationship of subsequent approximations. To cope with this problem, D'Silva *et al.* [20] considered an abstract and approximation-oriented view of interpolant-based model checking, which provided better counterexample-free approximations within an integrated algorithm for forward and backward analyses.

C. Roadmap

This paper is organized as follows. Section II introduces background notions on bounded, unbounded, and interpolant model checking. Section III describes how to obtain mixed representations with partial (i.e., incomplete) state sets starting from circuit-based unrollings. It presents our contributions to partial quantification strategies, discussing how different methods can be activated under limited resource bounds and how those methods exploit/produce partial circuit and/or state set representations. Section IV explains how to use data

partitioning to split circuits and state set representations for improved scalability. Section V shows our integrated approach to UMC. Finally, Section VI discusses the experiments we performed, and Section VII concludes this paper with some summarizing remarks.

II. Background

A. Model and Notation

We address systems modeled as labeled state transition structures, and represented implicitly by Boolean formulas. The state space and the primary inputs are defined by indexed sets of Boolean variables $V = \{v_1, \ldots, v_n\}$ and $W = \{w_1, \ldots, w_m\}$, respectively. States correspond to the valuations of variables in V, whereas transition labels correspond to evaluations of variables in W. We indicate next states with the primed variable set $V' = \{v'_1, \ldots, v'_n\}$. Whenever we explicitly need time frame variables, we use $V^i = \{v^i_1, \ldots, v^i_n\}$ and $W^i = \{w^i_1, \ldots, w^i_m\}$ for variable instances at the ith time frame. We also adopt the short notation $V^{i..j}$ ($W^{i..j}$) for $V^i \cup V^{i+1} \cup \ldots \cup V^j$ ($W^i \cup W^{i+1} \cup \ldots \cup W^j$).[1] We denote with $\top$ and $\bot$ the logical *true* and *false* constants, respectively.

A set of states is expressed by a state predicate $S(V)$ (or $S(V')$ for the next state space). $I(V)$ is the initial state predicate. We use $P(V)$ to denote an invariant property, and its complement, $F(V) = \neg P(V)$, as target for bug search. Notice that, for the sake of simplicity, all the algorithms presented in the sequel assume that $I \wedge F = \bot$. However, extending them to verify this condition is straightforward.

$T(V, W, V')$ is the transition relation of the system. We assume T given by a *circuit graph*, with state variables mapped to latches. Present and next state variables correspond to latch outputs and inputs, respectively. The input of the ith latch is fed by a combinational circuit, described by the Boolean function $\delta_i(V, W)$. Hence, the transition relation can be expressed as

$$\begin{aligned} T(V, W, V') &= \textstyle\bigwedge_i t_i(V, W, v'_i) \\ &= \textstyle\bigwedge_i (v'_i \Leftrightarrow \delta_i(V, W)). \end{aligned}$$

A state path of length k (with $k > 0$) is a sequence of states $\sigma_0, \ldots, \sigma_k$ such that $T(\sigma_i, \nu_i, \sigma_{i+1})$ is true, given some input pattern ν_i, for all $0 \leq i < k$.

A state set $\widehat{S}$ is reachable from the state set S in k steps if there exists a path of length k, in the labeled state transition structure, connecting a state in S to another state in $\widehat{S}$. In other words, the following formula has to be satisfiable:

$$S(V^0) \wedge \left(\bigwedge_{i=0}^{k-1} T(V^i, W^i, V^{i+1}) \right) \wedge \widehat{S}(V^k).$$

The image operator $\mathrm{Img}(T, S)$ computes the set of states *Image* reachable in one step from the states in S

$$\begin{aligned} Image(V') &= \mathrm{Img}(T(V, W, V'),\ \ S(V)) \\ &= \exists_{V,W}.S(V) \wedge T(V, W, V'). \end{aligned}$$

[1] $V^{i..j}$ and $W^{i..j}$ are appropriately defined if $i \leq j$, otherwise we conventionally specify them as empty variable sets.

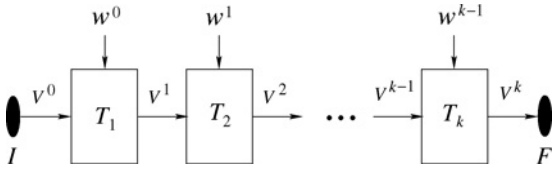

Fig. 1. BMC propositional formula based on the relation T.

Pre-image is dual, with the only difference that the existential quantification of functionally computed state variables can be obtained by composition

$$\begin{aligned} PreImage(V) &= \mathrm{PreImg}(T(V, W, V'),\ \ S(V')) \\ &= \exists_{W,V'}.S(V') \wedge T(V, W, V') \\ &= \exists_W.S(\delta(V, W)). \end{aligned} \tag{1}$$

An over-approximate image is any state set including the exact image

$$Image^+(V') = \mathrm{Img}^+(T,\ \ S) \supseteq \mathrm{Img}(T,\ \ S).$$

Finally, given a Boolean function f, we will use the notation $supp(f)$ to denote its variable set of support.

B. BMC

SAT-based BMC [21] considers only k-bounded reachability, as expressed by the propositional formula

$$f = I(V^0) \wedge \left(\bigwedge_{i=0}^{k-1} T(V^i, W^i, V^{i+1}) \right) \wedge F(V^k)$$

that is satisfiable *iff* a counter-example (a path from I to F) of length k exists. The formula f can be graphically represented as shown in Fig. 1, where each box represents an entire transition relation T.

Existential quantification can be applied to intermediate sets of state variables

$$\begin{aligned} g &= \exists_{V^{1..k}}.f \\ &= I(V^0) \wedge \exists_{V^{1..k}}.\textstyle\bigwedge_{i=0}^{k-1} T(V^i, W^i, V^{i+1}) \wedge F(V^k) \\ &= I(V^0) \wedge Cone_k(V^0, W^{0..k-1}) \end{aligned}$$

where $Cone_k$ represents a combinational single output circuit unrolling, formally defined by exploiting quantification by functional composition[2]

$$\begin{aligned} Cone_k &= Cone_k(V^0, W^{0..k-1}) \\ &= \exists_{V^{1..k}}.\textstyle\bigwedge_{i=0}^{k-1}(V^{i+1} \Leftrightarrow \delta(V^i, W^i)) \wedge F(V^k) \\ &= F(\delta(\ldots\delta(\delta(V^0, W^0), W^1)\ldots, W^{k-1})). \end{aligned} \tag{2}$$

The formula g may be depicted as in Fig. 2, where the right-end side cone-like shape represents the intrinsic reduction obtained by composition. In other words, when we are close

[2] $Cone_k$ is appropriately defined for all $k > 0$. $Cone_0$ is conventionally specified as equal to F.

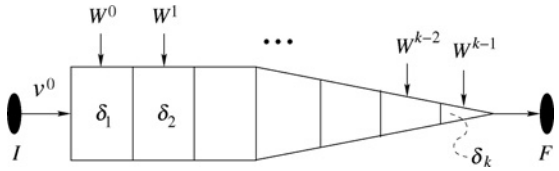

Fig. 2. BMC propositional formula based on *Cone*.

```
1   BACKWARDMC (I, T, F)
2     k = 0
3     BckR_0 = F
4     do
5       k = k + 1
6       BckR_k = PREIMG (T, BckR_{k-1})
7       if (SAT (I ∧ BckR_k))
8         return (reachable)
9     while (SAT (BckR_k ∧ ¬ BckR_{0..k-1}))
10    return (unreachable)
```

Fig. 3. AIG-based backward verification.

to the target F, it is likely that only a few δs are relevant for the target itself, whereas the remainder can be discarded. This reduction comes out in a natural way when using *Cone*, whereas it can be obtained only exploiting *ad hoc* algorithms with the formalism T.

However, the main advantage of using $Cone_k$ instead of $\bigwedge_i T_i$ is that $Cone_k$ is a single circuit. As a consequence, several combinational optimizations (covering even different time frames) can be applied before converting it into CNF. On the contrary, this is not possible with $\bigwedge_i T_i$, since in this case only the set of functions δ within each single time step can be optimized.

In the sequel, we will use the short notation $Cone_{0..k}$ to indicate the disjunction of all the cones up to the kth time frame: $Cone_{0..k} = \bigvee_{i=0}^{k} Cone_i$.

C. Backward SAT-Based Unbounded Model Checking

Fig. 3 shows an SAT-based version of an UMC procedure, derived from [22], where and-inverter graphs (AIGs) (or cognate, non canonical, circuit-based representations [23]) are used to represent and manipulate logic functions.

Starting from the failure state set, each iteration of the loop computes the set of states $BckR_k$ backward reachable in exactly k steps from the target state set F (line six). If the intersection of $BckR_k$ with the initial set of states I is non-empty (line seven), then the `reachable` result is returned, meaning that the property has been disproved. The loop ends when the fixed-point has been found, i.e., when all states in $BckR_k$ have been previously discovered. This check is performed at line nine, where the notation $BckR_{0..k}$ is used to indicate the set of states reachable up to depth k, i.e., $BckR_{0..k} = \bigvee_{i=0}^{k} BckR_i$. In this case, the property has been proved, and the `unreachable` result is returned (line ten).

The main weakness of the algorithm is due to the complexity of performing the quantifications hidden in the PreImg operator [see (1)].

Notice that the set of states $BckR_k$ can be also obtained by primary input quantification over the circuit unrolling $Cone_k$ defined by (2)

$$BckR_k(V) \quad = \quad \exists_{W^{0..k-1}}.Cone_k(V, W^{0..k-1}). \tag{3}$$

A similar consideration holds for $BckR_{0..k}$

$$\begin{aligned} BckR_{0..k}(V) &= \bigvee_{i=0}^{k} BckR_i(V) \\ &= \bigvee_{i=0}^{k} \exists_{W^{0..i-1}}.Cone_i(V, W^{0..i-1}) \\ &= \exists_{W^{0..k-1}}.\bigvee_{i=0}^{k} Cone_i(V, W^{0..i-1}) \\ &= \exists_{W^{0..k-1}}.Cone_{0..k}(V, W^{0..k-1}) \end{aligned} \tag{4}$$

where distributivity of existential quantification over union has been applied.

We finally remark the fact that the fixed-point check in function BACKWARDMC (line nine) consists of a single SAT run because $BckR_k$ and $BckR_{0..k}$ are state sets, i.e., no primary input variable appears in their support. Replacing them with the corresponding expressions *Cone* given in (3) and (4) turns this test into a quantified Boolean formula (QBF) problem.

D. Craig Interpolants in Model Checking

Definition 1: Given two inconsistent formulas A and B (i.e., $A \wedge B = \bot$), an interpolant C is a formula such that:

1) it is implied by A, i.e, $A \Rightarrow C$;
2) it is inconsistent with B, i.e., $C \wedge B = \bot$;
3) it is expressed over the common variables of A and B, i.e, $supp(C) \subseteq supp(A) \cap supp(B)$.

A Craig interpolant $C = \text{Itp}\ (A,\ B)$ is an AND/OR circuit that can be directly derived from refutation proof of $A \wedge B$. Albeit the computation of Itp from the proof has a linear cost [24], the refutation proof can be exponentially larger than A and B.

An over-approximate image of a state set S is k-adequate, with respect to F, if it does not intersect any state on paths of length k to F. It can be computed as follows:

$$\text{Img}^{+}_{Adq}(T,\ S,\ Cone_{0..k}) = \text{Itp}\big(S(V^{-1}) \wedge T(V^{-1}, W^{-1}, V^0),\ Cone_{0..k}(V^0, W^{0..k-1})\big)$$

where the circuit unrolling $Cone_{0..k}$ encodes all k-step paths to F. A k-adequate over-approximate image Img^{+}_{Adq} is `undefined` *iff* the exact image is not k-adequate, i.e, *iff*

$$S(V^{-1}) \wedge T(V^{-1}, W^{-1}, V^0) \wedge Cone_{0..k}(V^0, W^{0..k-1})$$

is satisfiable. An image is called adequate if it is k-adequate for any k, i.e., no state within the image is backward reachable from F. Since the model is finite, a k-adequate image is adequate if $k \geq d$, where d is the diameter of the state transition graph.

Fig. 4 shows a fully SAT-based UMC procedure exploiting Craig interpolants, derived from the original one proposed by McMillan [3] by exploiting the *Cone* representation instead of T.

```
1  INTERPOLANTMC (I, T, F)
2    k = 0
3    Cone_0(V^0) = F(V^0)
4    do
5      res = FINITERUN (I, T, Cone_{0..k})
6      k = k + 1
7      Cone_k(V^0, W^{0..k-1}) = Cone_{k-1}(δ(V^0, W^0), W^{1..k-1})
8    while (res = undecided)
9    return (res)

10 FINITERUN (I, T, Cone)
11   if (SAT(I ∧ T ∧ Cone))
12     return (reachable)
13   R = I
14   while (⊤)
15     Image^+ = IMG^+_{Adq} (T, R, Cone)
16     if (Image^+ = undefined)
17       return (undecided)
18     if (Image^+ ⇒ R)
19       return (unreachable)
20     R = R ∨ Image^+
```

Fig. 4. Interpolant-based verification.

INTERPOLANTMC is the entry point of the algorithm. Each iteration of the loop executes a call to routine FINITERUN (line five), which performs the interpolant-based over-approximate traversal, ending up with one out of three possible results:

1) `reachable`, if it proves F reachable from I in k steps, hence the property has been disproved;
2) `unreachable`, if the approximate traversal using the Img^+_{Adq} image computation reaches a fixed-point. In this case the property has been proved;
3) `undecided`, if F is intersected by the over-approximate state sets.

In case a definite conclusion is not achieved by routine FINITERUN, k is increased and a deeper cone is generated (lines six and seven), thus preparing the next loop iteration.

It is possible to prove that the algorithm is sound and complete [3]. If I and F are mutually reachable, sooner or later the BMC check at line 11 will report a reachable result. Otherwise, if I and F are mutually unreachable, sooner or later k will become larger than d, Img^+_{Adq} will become adequate, and the algorithm will terminate with an approximate reachability fixed-point.

III. PARTIAL CIRCUITS AND STATE SETS

As we introduced in Section II, a circuit unrolling $Cone(V, W)$[3] represents paths, i.e., backward reachable states coupled with the related input assignments, whereas $BckR(V)$ is a set of reachable states with no information on paths to the target F. Obviously $Cone(V, W) \Rightarrow BckR(V)$, due to existential quantification [see (3) and (4)]. Since state sets support SAT fixed-point checks, we prefer them to backward circuit unrollings, at least as long as they can be computed with an acceptable cost. Nonetheless, this rarely happens. We thus accept *partial* representations, where both state sets and circuits are used to express parts of the reachable states.

A partial state set (circuit) is a sub-set (i.e., an incomplete representation) of a full state set (circuit). We denote such sub-sets with the "$^-$" super-script

$$BckR^-(V) \Rightarrow BckR(V)$$

$$Cone^-(V, W) \Rightarrow Cone(V, W).$$

A (backward) behavior can be completely represented as a disjunction of partial representations. Let us define Γ as

$$\Gamma(V, W) = BckR^-(V) \vee Cone^-(V, W). \tag{5}$$

We say that the representation Γ is *complete* if

$$\begin{aligned} BckR(V) &= \exists_W.\Gamma(V, W) \\ &= BckR^-(V) \vee \exists_W.Cone^-(V, W). \end{aligned} \tag{6}$$

In other words, a complete representation Γ is based on partitioned/partial backward cones and reachable state sets, such that their union still covers the whole set of states. A complete Γ may fully replace $BckR$ and/or $Cone$, in backward reachability and/or in the FINITERUN function, still preserving the completeness of the approach. As input paths explicitly appear just for a sub-set of the states in Γ, we have that

$$Cone(V, W) \Rightarrow \Gamma(V, W) \Rightarrow BckR(V).$$

Thus, the representation Γ lies in between two extremes:

1) an exact state set representation with *full quantification*, within a standard backward traversal scheme;
2) state sets and related input values, i.e., paths, through backward circuit unrollings with *no quantification* at all.

In the rest of this section, we describe how we can obtain (partial) state sets from circuit unrollings.

A. Circuit-Based Quantification

State sets can be computed from circuit unrollings by applying circuit-based quantification [22], according to (3) and (4). Our quantification algorithms, presented in this section, are controlled by size and time bounds in order to prevent blow-ups. When these bounds are not exceeded, the algorithms compute full state sets, otherwise they obtain partial state sets. We first present a straightforward solution for time and size control, by enabling full quantification as far as its cost is acceptable, then disabling it. Then, we show how we can operate at a finer level, by accepting/rejecting individual variable quantifications, and performing state sub-setting.

1) *Bounded Traversal:* We just enable full quantification up to a depth h, such that the computation of $BckR_{h+1}$ exceeds a given time/space threshold. We then disable the complete quantification at any depth $k > h$. Therefore, whenever h is smaller than the backward diameter, $BckR_{0..h}$ is a proper sub-set of the exact $BckR_{0..k}$. This simple heuristic handles all cases where the complexity of exact traversals is accepted up to a given depth.

In this phase, we also enable BDD-based computation of state sets, whenever the number of domain variables is acceptable (up to 100 in our present implementation). The

[3]For the sake of simplicity, hereinafter we do not explicitly indicate the bound k whenever not strictly necessary. As a consequence, $Cone$ stands for either $Cone_k$ or $Cone_{0..k}$, $BckR$ stands for either $BckR_k$ or $BckR_{0..k}$, and W stands for $W^{0..k-1}$.

```
1  LAZYEXIST (Cone, W)
2    G = Cone
3    forall w_i ∈ W
4      tmp = ∃_{w_i}.G
5      if (|tmp| < α · |G|)
6        // Accept quantification
7        G = tmp
8    return (G)
```

Fig. 5. Lazy quantification.

system transition relation T and the state sets $BckR$ are converted from AIGs to BDDs, then exact BDD-based pre-images are computed under time and space control. BDDs are generated by enabling cut-points and auxiliary variables as pseudo-inputs. Cut-points are heuristically selected to keep the size of the BDDs below a given threshold (selected by the user) and to prevent memory blow-up [25], [26].

2) *Lazy Quantification:* We introduce *lazy quantification* as an additional and more accurate control strategy for the size of state set representations. The lazy quantification operator accepts or rejects each variable quantification, based on space and time limits. The pseudo-code of this procedure is given in Fig. 5.

For each variable $w_i \in W$ that has to be quantified, we optionally accept or reject the quantification, based on a target maximum size increase α. A given quantification is maintained (line seven) only if the circuit size (evaluated in terms of AIG nodes) after quantification is acceptable (line five).

The global effect of the procedure is to filter out critical variable quantifications, i.e., those responsible of size explosion. Lazy quantification can thus produce every intermediate result in the range going from no quantification at all to full quantification.

3) *Lazy Quantification With Sub-Setting:* The final result of the LAZYEXIST procedure is either a state set or a cone, where some of the input variables have been ruled out (quantified), but some other (or all) are still in the support. In order to obtain a partial state set in any case, we introduce a variant of the lazy quantification, that always removes quantification variables, but produces sub-sets of the exact state sets. The method is inspired by [14] and [27], whose common idea is assigning constant values to a set of variables, thus restricting the represented state set within a given sub-space.

Following this idea, a sub-set $BckR^{-}(V)$ can be computed by assigning constant values to the W variables appearing in $Cone(V, W)$. Let σ be a variable assignment to W satisfying $Cone$ (generated by an SAT solver run). Then

$$BckR^{-}_{\sigma}(V) = Cone(V, \sigma) \Rightarrow BckR(V).$$

The overall level of under-estimation of $BckR(V)$ by $BckR^{-}_{\sigma}(V)$ strongly depends on the choice of σ. Most of the heuristics described in [14] are oriented to drive a circuit-based SAT solver toward more representative input solutions. To reduce the dependence on σ, we propose an alternative hybrid approach to sub-setting, where some primary inputs are existentially quantified and the others are set to a constant value.

A simplified implementation of such a procedure is shown in Fig. 6.

```
1  LAZYSUBSET (Cone, W)
2    G = Cone
3    σ = SAT (Cone)
4    forall w_i ∈ W
5      tmp = ∃_{w_i}.G
6      if (|tmp| < β · |G|)
7        // Accept quantification
8        G = tmp
9      else
10       // Reject quantification and perform sub-setting
11       G = G|_{w_i=σ[w_i]}
12   return (G)
```

Fig. 6. Lazy quantification with sub-setting.

The function LAZYSUBSET first computes a variable assignment σ satisfying $Cone$ (line three). Then, for each variable $w_i \in W$, it evaluates $\exists_{w_i}.G = G|_{w_i=0} \vee G|_{w_i=1}$ (line five) and compares the size of the result (evaluated in terms of AIG nodes) with the size of G scaled by the factor β (line six). If the result is too large, the quantification is rejected, and only one of the cofactors is retained (line 11). The selection of the cofactor is based on the value of σ, that is computed as previously described. The produced sub-set includes σ by construction (some variables are existentially quantified, the other ones are assigned according to σ). Since the variable assignment σ satisfies $Cone$, sub-setting will never end finding an empty sub-set. An alternative choice would be computing both cofactors and dynamically selecting one of them, based on support or size heuristics. Anyhow, in this case the procedure may incur in dead-ends, i.e., it may find an empty sub-set, unless a backtracking schema is introduced.

Our procedure can obtain potentially denser sub-sets than other sub-setting operators, since it assigns constant values only to variables that are critical for existential quantification.

Notice that the previous pseudo-codes hide some implementation details, such as:

1) variable scoring based on history of accepted/rejected quantifications: we directly reject quantification of variables with high rejection rate;
2) two-step quantification: a first cycle just quantifies accepted variables, with no sub-setting, a second loop through the previously rejected variables enables both quantification and sub-setting;
3) time thresholds: the cost in terms of central processing unit (CPU) time is limited for single variable quantifications and for the entire quantification procedure, with inner SAT sweepings to simplify the result.

B. Bounded SAT Enumeration

As outlined in Section II-C, a fixed-point check consists of a single SAT run when state sets are available. However, this is no longer true when working with a mixed representation of partial state sets and circuit unrollings, as the test turns into a QBF instance. We present in this section an SAT approach to solve this problem, which produces partial state sets as a by-product. The method is inspired by Mneimneh *et al.* [28], who evaluated the diameter of a state transition graph by performing state enumerations with an SAT solver. For every state reachable from the initial state set in $k + 1$ steps (found

```
1  CHECKFP ($Cone_{k+1}$, $Cone^{-}_{0..k}$, $BckR^{-}_{0..k}$)
2    for $j = 1$ to $maxIter$
3      // Find (new?) state and related input values ($\sigma$)
4      $\sigma$ = SAT ($Cone_{k+1} \wedge \neg BckR^{-}_{0..k}$)
5      if ($\sigma = \emptyset$)
6        // No new state: fixed-point
7        return ($\top$)
8      // Isolate state
9      $\sigma_V = \exists_W . \sigma$
10     // Check if state already in $Cone^{-}_{0..k}$
11     $\gamma$ = SAT ($Cone^{-}_{0..k}|_{V=\sigma_V}$)
12     if ($\gamma = \emptyset$)
13       // New state: no fixed-point
14       return ($\bot$)
15     // Already reached state: enlarge $BckR^{-}_{0..k}$
16     $\gamma_W = \exists_V . \gamma$
17     $BckR^{-}_{0..k} = BckR^{-}_{0..k} \vee Cone^{-}_{0..k}|_{W=\gamma_W}$
18   // Not decided after $maxIter$ iterations: abort
19   return (undecided)
```

Fig. 7. Checking fixed-point based on SAT enumeration.

through an SAT run), an SAT call verifies whether such a state is reachable at depth less or equal to k.

1) If the state is reachable, then a blocking clause (forbidding that state) is added at depth $k+1$, and the process is iterated.
2) If the state is not reachable, k is not the diameter, then the entire reasoning is repeated with a larger value of k.

The process terminates when no new state is obtained at depth $k+1$ (reachability fixed-point), meaning that k is the system diameter. The authors showed that a complete SAT approach is more efficient than the one based on a QBF solver.

We adopt a similar SAT enumeration strategy, where we perform an incomplete (bounded) search in order to limit the iterative quest for new states and the related generation of (blocking) state sets. Moreover, instead of working at the level of blocking CNF clauses (as in [28]), we exploit circuit-based cofactoring [14], so that whenever an already reached state is found, we *enlarge* it to all states backward reachable through the same input values. Finally, backward paths up to depth k are represented in our case by a *complete* representation $BckR^{-}/Cone^{-}$ of partial state sets/circuit unrollings.

A simplified implementation of such a procedure is shown in Fig. 7. The function receives as parameters the backward circuit unrolling at depth $k+1$ and the (partial) unrollings and state sets up to depth k. If the previous quantifications completed successfully, then $Cone^{-}_{0..k}$ is empty and $BckR^{-}_{0..k} = BckR_{0..k}$. In this case, a single iteration of the loop (line two) is sufficient to produce the result. In all other cases (mixed state sets and circuit unrollings), the function performs a heuristically limited number (*maxIter*) of SAT runs, trying to find a state reachable at depth $k+1$ (through $Cone_{k+1}$) but not present in the previously reached state set ($BckR^{-}_{0..k}$ or $Cone^{-}_{0..k}$). Whenever a state included in both $Cone_{k+1}$ and $Cone^{-}_{0..k}$ is found, the set of states backward reachable through the same input assignment (γ_W) is copied from $Cone^{-}_{0..k}$ to $BckR^{-}_{0..k}$ (lines 16 and 17). If the bound on the SAT calls is exceeded, CHECKFP ends up with an `undecided` result (line 19).

IV. Interpolation Using State Sets and Partitioning

When the SAT-based backward reachability approach shows its limits, we resort to interpolation with the enhancements described as follows. We exploit partial state sets (originated as a sub-product of circuit-based quantification and SAT enumeration), and disjoined cone decompositions. We extensively adopt partitioning within complex interpolant-based image computations and traversals. Furthermore, partial state sets are exploited as don't care sets to simplify/restrict interpolant-based traversals.

A. Partitioned Interpolation

We apply a divide-and-conquer approach to adequate image computation. We generate a conjunctively partitioned image starting from a disjunctively partitioned circuit unrolling.

For the sake of simplicity, let us concentrate in this section on a backward unrolling disjunctively decomposed into a set of partial sub-cones (the presence of a partial state set will be analyzed in the next subsection). Notice that this is the usual situation when using the algorithm reported in Fig. 4, since the parameter *Cone* of the function FINITERUN actually corresponds to $Cone_{0..k} = \bigvee_{i=0}^{k} Cone_i$. Furthermore, another possible source for disjunctive decompositions is represented by the property under check. Whenever $P(V) = \bigwedge_i p_i(V)$, we have that $F(V)$ is a disjunction, as $F(V) = \neg P(V) = \bigvee_i \neg p_i$.

Definition 2: Given two inconsistent formulas A and B, we say that any two Boolean functions Ψ_1 and Ψ_2 are *equivalent modulo interpolation* if both of them represent a valid interpolant for (A, B). In this case we use the short notation $\Psi_1 \approx_{A,B} \Psi_2$.

Theorem 1: Given two inconsistent formulas A and B, such that $B = \bigvee_i b_i$, then

$$\mathrm{Itp}(A, B) \approx_{A,B} \bigwedge_i \mathrm{Itp}(A, b_i).$$

Proof. The left-end side of the relation $\approx$ is obviously an interpolant for (A, B). As a consequence, we only need to demonstrate that the right-end side also satisfies the three interpolant properties of Definition 1.

1) $A \Rightarrow \mathrm{Itp}(A, b_i)$, for any i, by definition. Hence, it is also true that $A \Rightarrow \bigwedge_i \mathrm{Itp}(A, b_i)$.
2) We have that

$$\begin{aligned}\bigwedge_i \mathrm{Itp}(A, b_i) \wedge B &= \bigwedge_i \mathrm{Itp}(A, b_i) \wedge \bigvee_j b_j \\ &= \bigvee_j (\bigwedge_i \mathrm{Itp}(A, b_i) \wedge b_j) \\ &= \bigvee_j (\bigwedge_{i \neq j} \mathrm{Itp}(A, b_i) \wedge \mathrm{Itp}(A, b_j) \wedge b_j) \\ &= \bot\end{aligned}$$

because $\mathrm{Itp}(A, b_j) \wedge b_j = \bot$ by definition.
3) By hypothesis, $supp(b_i) \subseteq supp(B)$. Furthermore, we know that $supp(\mathrm{Itp}(A, b_i)) \subseteq supp(A) \cap supp(b_i)$. Thus

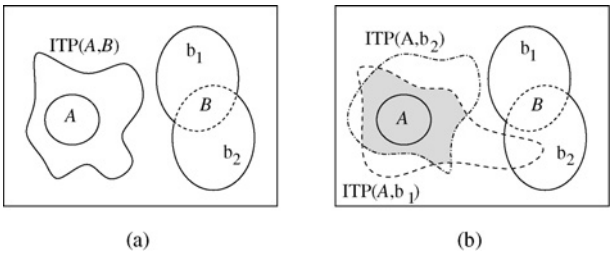

Fig. 8. Partitioned interpolant computation. Computing Itp(A, B): (a) standard (monolithic) case and (b) partitioned approach.

$$\begin{aligned} supp(\textstyle\bigwedge_i \text{Itp}(A, b_i)) &= \textstyle\bigcup_i supp(\text{Itp}(A, b_i)) \\ &\subseteq \textstyle\bigcup_i supp(A) \cap supp(b_i) \\ &= supp(A) \cap \textstyle\bigcup_i supp(b_i) \\ &= supp(A) \cap supp(B) \end{aligned}$$

as $\bigcup_i supp(b_i) = supp(B)$ by hypothesis. □

The meaning of the previous theorem is graphically depicted in Fig. 8, which represents the projections of A and B in the space defined by their common variables (such a space is schematically pictured as a rectangle). For the sake of simplicity, we assume that $B = b_1 \vee b_2$. Fig. 8(a) shows the interpolant as computed with respect to (A, B), i.e., the left-end side of Theorem 1. On the other hand, Fig. 8(b) illustrates the two interpolants computed with respect to (A, b_1) and (A, b_2). Their conjunction (right-end side of Theorem 1 and shadowed area) is still an interpolant for (A, B).

Corollary 1: Given $Cone(V, W) = \bigvee_i Cone_i(V, W)$, then

$$\text{Img}^+_{Adq}(T, S, Cone) \approx_{S \wedge T, Cone} \bigwedge_i \text{Img}^+_{Adq}(T, S, Cone_i).$$

Proof. To prove the corollary, it is sufficient to observe that $\text{Img}^+_{Adq}(T, S, Cone)$ is computed as $\text{Itp}(S \wedge T, Cone)$, and then to apply Theorem 1. □

We heuristically exploit the above formulation for complex images when the monolithic interpolation is too expensive in terms of time or space. We proceed as follows. First of all, we compute the monolithic interpolant under limited memory/time resource bounds. If the process does not complete, or the result is larger than a threshold, we compute the partitioned interpolant as well. Finally, we take the best result between the monolithic and the partitioned interpolants. The intuition behind the above heuristic strategy is that we spend an extra overhead (to compute the partitioned interpolant) only when the monolithic procedure already had a high time/memory cost.

In order to compute a partitioned interpolant, we sort disjunctive components by size, evaluating easier sub-images first and harder ones later, or vice-versa. The latter scheme showed to be the most promising one, as larger components often subsume several smaller ones, which do not have to be considered anymore.

Our decomposed image differs from usual partitioned images in model checking, which typically follow either the disjunctive model (disjunctive partitioned image working with disjunctive state sets and/or transition relation) or the conjunctive one (conjunctively partitioned transition relation with monolithic state sets). Our scheme is specifically oriented to interpolant-based model checking: we use disjunctive decomposed backward unrollings, and we produce conjunctively decomposed images.

B. Interpolation and Partial State Sets

Partial state sets could simply replace the corresponding cones in partitioned interpolation. In other words, interpolants computed with respect to $BckR$ or $Cone$ are equivalent, since

$$\text{Sat}(S \wedge T \wedge BckR) \quad \Leftrightarrow \quad \text{Sat}(S \wedge T \wedge Cone).$$

However, since input variables have already been ruled out (quantified) from $BckR$, a possible option for interpolation is to directly take the complement of $BckR$ as interpolant[4]

$$\text{Img}^+_{Adq}(T, S, BckR) \quad \Rightarrow \quad \neg BckR.$$

Obviously, computing $\neg BckR$ is faster than resorting to SAT refutation proofs, although the result is over-estimated at its maximum allowed level. Anyhow, this solution can still be a good option when working at the backward diameter.

As an additional consideration, we experimentally observed that SAT runs are generally more efficient (and refutation proofs more compact) when using backward cones rather than the corresponding state sets. This may be related to the fact that state sets are usually larger than their original circuit counterparts, and that input variables (existentially quantified while computing state sets) may effectively help the SAT solver by guiding the decision process and leading to search space reductions.

Taking into account the above observations, we decided to use state sets as substitutes of backward cones for interpolant computation just in a few cases, i.e., when their size is smaller than the corresponding expressions $Cone$. In all other cases, we consider them as redundant information, to be exploited as *don't cares*, for circuit-based optimization, or as *space* constraints to be added to SAT runs for better performance.

1) *Circuit Simplifications:* Circuit optimizations can be very effective when don't care conditions (either input, output or external don't cares) are given. In our context, various optimization methodologies are possible, based on BDD and/or SAT operators, such as BDD [25], [29] and SAT [30] sweeping, redundancy removal [31], or other circuit-based optimizations [23], [32]. Let us refer to a generic Simplify procedure, which compacts a cone under external care conditions

$$Cone^- = \text{Simplify}(Cone, care).$$

Then, any partial state set $BckR^-$ can be used as don't care set (i.e., $care = \neg BckR^-$) for other cones, or for state sets (and transition relations) to be used within the function FINITERUN. This is allowed by the fact that cones and state sets will

[4]We just consider the case of unsatisfiable runs. The interpolant is still undefined when the SAT run returns satisfiable.

be finally OR-ed together, in order to obtain a complete representation Γ.

2) *Sub-Space Constraining:* Another optimization, directly related to partitioned image computation, is the possibility to use image over-approximations and/or state sets as search space constraints for interpolant computation.

Definition 3: Given two inconsistent formulas A and B and a constraint function C, we say that any two Boolean functions Ψ_1 and Ψ_2 are *equivalent modulo interpolation under constraint* C if $C \Rightarrow (\Psi_1 \approx_{A,B} \Psi_2)$. In this case we use the short notation $\Psi_1 \approx^{C}_{A,B} \Psi_2$.

In other words, in order to be equivalent modulo interpolation under constraint C, two functions Ψ_1 and Ψ_2 have both to represent a valid interpolant for (A, B), *but only in the sub-space identified by* C. More specifically, this means that for $i \in \{1, 2\}$:

1) $C \Rightarrow (A \Rightarrow \Psi_i)$;
2) $C \Rightarrow (\Psi_i \wedge B = \bot)$;
3) $supp(\Psi_i) \subseteq supp(A) \cap supp(B)$.

Given the previous definition, we can present the following theorem.

Theorem 2: Given three formulas A, B, and C such that $A \wedge B = \bot$ and $supp(C) \subseteq supp(B)$, then

$$\text{Itp}(A, B) \approx^{C}_{A,B} \text{Itp}(A, B \wedge C).$$

Proof. Since the left-hand side of the relation $\approx$ is obviously an interpolant for (A, B), regardless of the value of C, we only need to prove the three (constrained) interpolant conditions for the right-end side.

1) $A \Rightarrow \text{Itp}(A, B \wedge C)$ by definition, so it is also true that $C \Rightarrow (A \Rightarrow \text{Itp}(A, B \wedge C))$.
2) We need to demonstrate that $C \Rightarrow \neg(\text{Itp}(A, B \wedge C) \wedge B)$ is a tautology

$$\begin{aligned} C \Rightarrow \neg(\text{Itp}(A, B \wedge C) \wedge B) &= \neg C \vee \neg \text{Itp}(A, B \wedge C) \vee \neg B \\ &= \neg(\text{Itp}(A, B \wedge C) \wedge B \wedge C) \\ &= \top \end{aligned}$$

as $\text{Itp}(A, B \wedge C) \wedge B \wedge C = \bot$ by definition.

3) We have that

$$\begin{aligned} supp(\text{Itp}(A, B \wedge C)) &\subseteq supp(A) \cap (supp(B) \cup supp(C)) \\ &= supp(A) \cap supp(B) \end{aligned}$$

since $supp(C) \subseteq supp(B)$ by hypothesis. □

Using the same graphic conventions of Fig. 8, Fig. 9 illustrates the meaning of the Theorem 2. In Fig. 9(a), we have the standard case: the interpolant $\text{Itp}(A, B)$ includes A but does not overlap with B. The sub-space of this interpolant included in C is then shadowed. On the other side, Fig. 9(b) represents the case in which the interpolant is computed with respect to $(A, B \wedge C)$. Thus, it includes A, but it may intersect B. However, this fact cannot occur in the C sub-space.

Corollary 2: Whenever the result of an adequate image, computed by interpolation, is intersected with a given constraint *care*, the adequacy constraint can be conjoined with *care* before doing interpolation

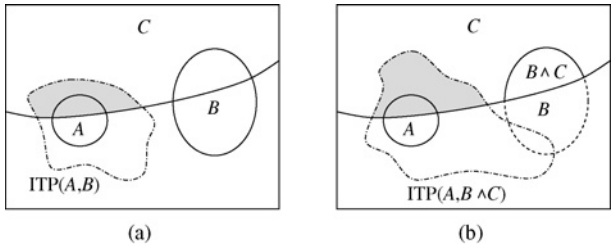

Fig. 9. Constrained interpolant computation. Computing $\text{Itp}(A, B)$ with constraint C: (a) standard case and (b) under the application of Theorem 2.

$$\text{Img}^{+}_{Adq}(T, S, Cone) \wedge care \approx^{care}_{S \wedge T, Cone} \text{Img}^{+}_{Adq}(T, S, Cone \wedge care) \wedge care.$$

Proof. The previous formula is a direct consequence of Theorem 2. □

We exploit the previous result for complexity reduction, as constraining a cone can turn out to reduce the overall cost of the SAT run, and the size of the related proof and interpolant as a consequence. We heuristically choose the above technique whenever a *care* constraint is available as the complement of backward reachable states

$$care = \neg Bck R^{-}.$$

Moreover, when $Cone_{0..k}$ is large, an adequate image computed with respect to backward cones at a smaller depth $h < k$ can be used as *care*

$$care = \text{Img}^{+}_{Adq}(T, S, Cone_{0..h}).$$

Therefore, the adequate image is finally computed as $\text{Img}^{+}_{Adq}(T, S, Cone \wedge care) \wedge care$.

V. An Integrated Approach

In this section, we describe our model checking procedure combining the previously described techniques. We compute partial state sets in the backward direction. Whenever they are not able to support complete reachability fixed-point checks, we exploit them (with partitioned cones) for improved interpolant-based forward traversals. Backward reachable partial state sets (Section III) are obtained by circuit-based quantification (Section III-A) and SAT enumeration (Section III-B), controlled by time and size bounds. The main goal of these steps is to capture *easy* backward fixed-points and/or to provide search space restrictions (care sets) for interpolation-based runs, which are activated exploiting the optimizations described in Section IV. More in detail, state sets are used as don't cares, as described in Section IV-B (Theorem 2), whereas circuit unrollings can be used in disjunctive partitioned form and benefit from Theorem 1 (Section IV-A). Partitioned cones are derived both from conjunctive properties and from iterative cone computations.

Fig. 10 shows the pseudo-code for the top-level verification function IntegratedMC. The loop (line five) follows the standard backward traversal scheme of Fig. 3. At each iteration, it computes a deeper backward cone (line six), as circuit unrolling starting from the target F (line three). Then, it tests

```
1   IntegratedMC (I, T, F)
2     k = 1
3     BckR_0 = Cone_0 = F
4     res = undecided
5     do
6       Cone_k(V, W^{0..k-1}) = Cone_{k-1}(δ(V, W^0), W^{1..k-1})
7       if (Sat(I ∧ Cone_k))
8         return (reachable)
9       BckR_k^- = PartialStateSets (Cone_k, W^{0..k-1})
10      fp = CheckFP (Cone_k, Cone_{0..k-1}, BckR^-_{0..k-1})
11      if (fp = ⊤)
12        return (unreachable)
13      if (fp = undecided)
14        Cone^-_{0..k} = Simplify(Cone_{0..k}, ¬BckR^-_{0..k})
15        res = PartFiniteRun (I, T, Cone^-_{0..k}, BckR^-_{0..k})
16      k = k + 1
17    while (res = undecided)
18    return (res)

19  PartialStateSets (Cone, W)
20    if (BDDs enabled)
21      BckR = ∃_W.Cone
22      if (BckR ≠ ∅)
23        return (BckR)
24      disable BDDs
25    BckR = Cone = LazyExist(Cone, W)
26    if (partial Exist done)
27      BckR = LazySubset (Cone, W)
28    return (BckR)
```

Fig. 10. Verification procedure: an integrated approach dovetailing SAT-based backward model checking and interpolation.

the condition for failure by checking the intersection of $Cone_k$ with the initial state I (line seven). If such an intersection is not empty, the result `reachable` is returned (line eight).

Then, circuit-based quantification is executed, calling the function PartialStateSets (line line). A detailed description of this procedure is reported at lines 19–28. BDDs are tentatively used as long as they are able to achieve full quantification. To this respect, it should be observed that, according to the given pseudo-code, $BckR$ is obtained by quantifying the input variables of $Cone$. In the real implementation, however, the function PartialStateSets also receives T and the previously computed state sets, and a pure BDD-based pre-image computation is attempted. If BDDs are able to perform quantification, the resulting state set is returned (line 23), otherwise they are disabled and the function LazyExist is called (line 25). If some circuit-based quantifications are not accepted, sub-setting (LazySubset, line 27) is then performed. The resulting state set $BckR$ is finally returned. The procedure thus implements a bounded backward traversal using BDDs and/or circuit-based quantification (see Sections III-A1 and III-A2), where sub-setting (Section III-A3) is applied only when full quantification is not achieved.

After that, CheckFP (i.e., SAT enumeration, line ten) verifies, under controlled time and memory conditions, whether the fixed-point is hit. If this happens, the `unreachable` result is returned (line 12). Otherwise, if the fixed-point could not be decided, $BckR^-_{0..k}$ is used as a don't care set to optimize $Cone_{0..k}$, computing its sub-set $Cone^-_{0..k}$ (line 14). As mentioned in Section IV-B1, any redundancy-removal procedure that is able to exploit an external *care* set can be used instead of the function Simplify.

Finally, the interpolant-based forward procedure PartFiniteRun (line 15) is activated. This is a variant of FiniteRun, which uses the optimized (divide-and-conquer) image computation based on partial sub-sets and exploits $BckR^-_{0..k}$ for search space restriction, as discussed in Section IV.

Theorem 3: The function IntegratedMC is sound and complete.

Sketch of proof. Soundness. A `reachable` result may be delivered only by the BMC check of line seven. This check is sound because each $Cone_i$ is only transformed by (partial) existential quantifications, which do not alter the result of SAT calls. An `unreachable` result may be given by CheckFP or PartFiniteRun. CheckFP finds a fix-point whenever no new states are backward reachable at depth k with respect to $k-1$. The result is sound as our state representation does not alter backward state reachability (we just operate partial existential quantification of input variables). A similar argument holds for PartFiniteRun, where the original backward cone is replaced by a complete representation $BckR/Cone$.

Completeness. The function CheckFP does not ensure completeness, as we activate it with limited resources, so it might return `undecided`. Completeness of our method is guaranteed by the complete interpolant-based approach (i.e., the function PartFiniteRun), that will converge at the backward diameter, in the worst case. □

VI. Experimental Results

We implemented our approach on top of the PdTrav tool, a state-of-the-art verification framework which won two of the sub-categories at the Hardware Model Checking Competition 2007 (HWMCC'07) [15], and ranked among the first three HWMCC'08 verification engines.

Our experiments ran on a Quad Core 2.4 GHz Workstation with 8 GB of main memory, hosting a Debian Linux distribution. Memory limit was always set to 1 GB.

The software configuration of our framework features CUDD [33], which provides functions to manipulate BDDs, MiniSAT [34], that is an satisfiability solver with unsatisfiability proof-logging capabilities, and ABC [35], that in our environment provides synthesis and optimization procedures.

We present results on circuits derived from the HWMCC'08 suite, and some industrial benchmarks coming from STMicroelectronics. We focus on *true* (i.e., proved) properties, since they are the most interesting ones for UMC (*false* instances are usually captured by BMC runs). As we mainly target difficult property proofs, and the original HWMCC'08 suite features several hundreds of circuits (including many very easy/small ones), we selected a sub-set of them, according to the following criteria.

1) We ran a preliminary *in/out* experimental session, using our own implementation of the standard interpolant model checking procedure [3]. We consider this algorithm as a *reference* for all subsequent experiments. We used the same time limit adopted at the HWMCC competitions, i.e., 900 s, on all benchmark instances not

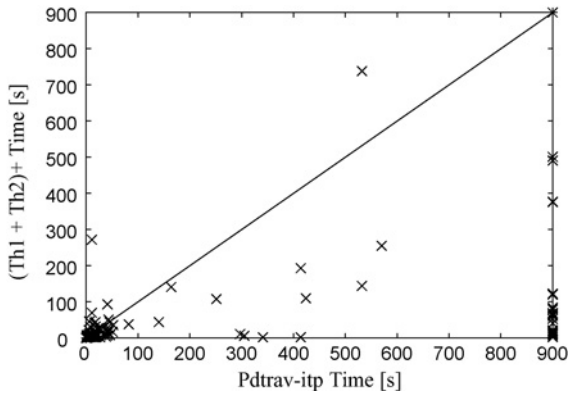

Fig. 11. Result statistics: HWMCC'08 `PdTrav-itp` versus our new *integrated approach*, with a time limit of 900 s.

tagged as *sat*.[5] All properties solved in less than 200 s were removed before moving to the next step.
During this preliminary experimental analysis, we also compared our new integrated method[6] with the one submitted to the HWMCC'08 competition (referenced as `pdtrav-itp`). Albeit our target is mainly represented by hard-to-solve instances, Fig. 11 presents the comparison on the entire set of verification properties, including both easy and hard-to-solve ones. The graph shows that our new method does present some overhead on the easier instances, but it is also much faster on the majority of the problems requiring more than 100 seconds of CPU time. Moreover, a conspicuous number of previously timed out instances are now verified.

2) For all properties filtered from the previous step, we ran a more extensive experimental session, featuring the reference setting and all the optimizations proposed in this paper. In this case, we adopted a higher timeout threshold, i.e., 3600 s. We further discarded all benchmarks that ran out-of-time in all approaches.
3) Some of the remaining benchmarks are very hard-to-solve instances when resorting to SAT-based methods, although they are known to be easy with BDD-based approaches. As a consequence, we removed most of these trivial properties, while we kept a few representative cases (the ones whose name is marked with a star, *, in Table I), for the purpose of showing our *adaptive* method's ability to outperform pure SAT-based algorithms by exploiting inner BDD-based steps.

As a final result of this selection process, we ended up considering ten Intel benchmarks, four industrial designs from STMicroelectronics, and 14 other circuits coming from the HWMCC'08 public suite.

Our experimental analysis is divided into two parts. In the first part (see Section VI-A), we present our results on property verification, detailing the contributions of the individual optimizations presented in this paper. In the second one (see Section VI-B), we provide detailed statistics on improved interpolation, based on partitioned image (Theorem 1) and sub-space constraining (Theorem 2).

[5]In the HWMCC notation, *uns*, i.e., unsatisfiable, designs are those whose property is proved, whereas *sat*, i.e., satisfiable, indicates that the property is falsified.

[6]This method will be indicated with $(Th_1 + Th_2)^+$ in Table I.

A. Verification Results

We compare the reference strategy (our own implementation of the standard interpolant model checking procedure) with and without our proposed optimizations.

Table I reports detailed data on the verification instances, selected as previously described. Columns 1–4 give information about the benchmarks. Model is the instance name, #PI, #FF, and #Nodes represent the number of primary inputs, memory elements, and AIG nodes of the circuit, respectively. We then show CPU running times obtained with standard interpolant computation (column Std ITP), and with the integrated method described in Section V. In order to provide a detailed analysis (and a better understanding) of individual contributions, we incrementally enabled our optimizations. Columns Th_1, Th_2, and $Th_1 + Th_2$ show the impact of Theorem 1 (partitioned interpolation) and Theorem 2 (space constraining), individually and combined, on purely interpolant-based UMC (with quantification strategies disabled). Circuit-based quantification and SAT enumeration (Section III-A) are enabled in the last two columns, where they are indicated by the $^+$ super-script. Columns Th_2^+ and $(Th_1 + Th_2)^+$ show the results obtained by our integrated approach, without and with Theorem 1 (partitioned interpolation) enabled. The acronyms ITP, CBQ, and SE, in the last column of the table, indicate the method providing termination (fixed-point) for the integrated verification procedure: ITP stands for interpolation, CBQ for circuit-based quantification (possibly resorting to BDDs, whenever this is allowed by the size of the representation), and SE for SAT enumeration.

Data show that we obtain average speed-ups of about 15, and up to a factor larger than 100 in a few cases. In particular, the full integrated approach (column $(Th_1 + Th_2)^+$) clearly delivers the best results, as it is the winner in 20 out of 28 cases. The original interpolation technique and the methods enabling only a few of the proposed optimizations are able to obtain better results on a few benchmarks. However, they are individually less robust and scalable than the fully integrated approach.

Overall, the presented data show that our combined strategy produces better results than any component optimization running separately.

Furthermore, a detailed analysis of our verification runs showed that the interpolant approach is extremely sensitive to the SAT solver decisions and proofs. A few benchmarks within the intel suite, for instance, moved from solved to unsolved, due to the level of over-approximation attained by SAT-based interpolants. This is shown by proofs achieved by individual methods, with execution times not far from the timeout threshold. It is well known that convergence of over-approximate traversals and abstraction approaches strongly rely on the level of accuracy of abstractions/approximations. To this respect, we can state that our optimizations usually make the verification algorithm more stable.

TABLE I
VERIFICATION RESULTS: DETAILED ANALYSIS OF SOME HARD-TO-SOLVE DESIGNS

Model	#PI	#FF	#Nodes	Std ITP (s)	New Methods (s)					
					Th_1	$Th_1 + Th_2$	Th_2	Th_2^+	$(Th_1 + Th_2)^+$	
bjrb07amba4andenv	11	33	17 448	3325.28	–	–	–	266.69	**242.79**	CBQ
cmuperiodic*	32	34	1555	889.05	891.68	1187.08	–	1360.00	**55.75**	CBQ
eijk3271	26	305	2546	–	–	681.65	686.83	671.60	**631.10**	SE
eijk3384	43	689	3069	–	–	–	–	647.19	**415.60**	SE
eijk6669	83	506	4879	–	3256.50		3085.79	3449.45	**1796.25**	SE
intel006	345	350	3265	218.36	228.48	**182.76**	279.76	322.84	–	–
intel018	548	491	6647	–	–	–	–	–	**2436.06**	ITP
intel020	349	354	5735	3154.96	–	–	–	**2697.49**	–	–
intel021	360	365	5882	–	–	–	**2263.23**	–	–	–
intel024	352	357	5710	2782.31	2561.57	2808.00	2746.94	1806.75	**1619.93**	ITP
intel026	486	492	6263	**1062.74**	1237.41	–	1633.22	1677.22	1180.76	ITP
intel044	642	647	6859	–	–	–	–	–	**680.23**	ITP
intel047	642	647	6859	–	–	–	–	–	**581.52**	ITP
intel049	136	141	1305	257.53	–	–	–	**137.16**	139.63	ITP
intel059	280	285	2705	–	–	–	–	–	**1210.50**	ITP
nusmvbrp*	11	52	527	–	–	–	–	–	**27.66**	CBQ
nusmvguidancep7	84	86	2001	233.89	1873.83	695.76	586.61	566.45	**173.35**	ITP
nusmvguidancep8*	84	86	1919	1392.38	1325.14	1210.19	–	2185.14	**85.45**	CBQ
nusmvreactorp2	74	76	1394	–	–	–	–	459.45	**447.72**	CBQ
pdtvisbakery0*	3	35	6252	2943.20	–	–	–	2768.35	**68.85**	CBQ
pdtvisbakery1*	3	35	6209	–	–	–	–	3105.26	**73.34**	CBQ
pdtvisbakery2*	3	35	6214	–	–	–	–	3138.04	**90.59**	CBQ
pdtvisns3p04	21	112	3718	**1586.87**	3407.38	–	–	1864.16	–	–
pdtvissoap2	11	160	3495	**3044.20**	–	–	–	–	–	–
Industrial_ A1	119	76	1075	–	–	**3305.27**	–	–	–	–
Industrial_A2	180	771	9357	1123.41	1132.43	1205.70	1233.88	1080.41	**383.71**	ITP
Industrial_A3	21	116	3879	–	–	–	–	–	**100.74**	ITP
Industrial_B1	19	425	3907	–	–	–	–	–	**824.06**	ITP

The symbol "–" means time overflow with a time limit of 3600 s.

B. Improved Interpolation

In this section, we present detailed results on the improved interpolation approach, based on the partitioning scheme, as described in Section IV-A (Theorem 1), and the sub-space constraining strategy presented in Section IV-B2 (Theorem 2). We concentrate on the verification instances analyzed in Table I.

The scattered plots in Figs. 12–14 summarize the statistics collected for Theorem 1. The partitioned image computation was disabled with small monolithic interpolants. Hence, it is worth noticing that our plots report only those instances[7] in which partitioning was enabled.

Fig. 12 represents the size (in terms of AIG nodes) of the partitioned cones. The x-axis refers to the instances. The y-axis indicates the size. Data are sorted by increasing y values. The figure shows how we usually have to deal with backward cones whose size ranges from 10 000 to 50 000 nodes. Larger unrollings are usually found in circuits that we were unable to solve within the adopted CPU time limit. Instances with the same y value often indicate cones that are re-used for several partitioned images within the same call PARTFINITERUN).

Fig. 13 plots the number of partitions per instance. The y-axis refers to the actual number of partitions. Analogously to Fig. 12, the x-axis indicates different instances sorted by increasing value on the y-axis. Usually the number of partitions is close to the depth of the backward unrolling we are considering. When the number is larger, the extra-

[7]In this context, an instance is the occurrence of a partitioned interpolant computation, for a given circuit. Therefore, some of the points in the plot may refer to the same benchmark.

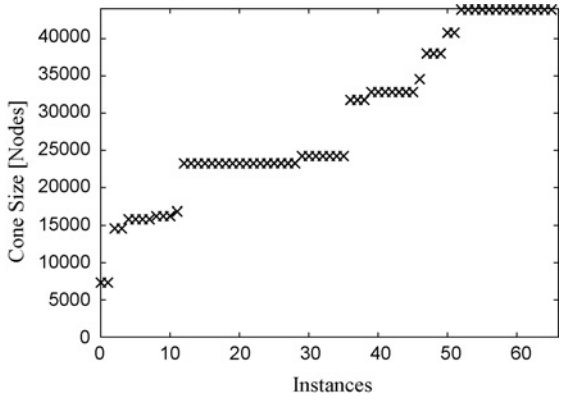

Fig. 12. Result statistics: partitioned cone sizes (AIG nodes) per instance.

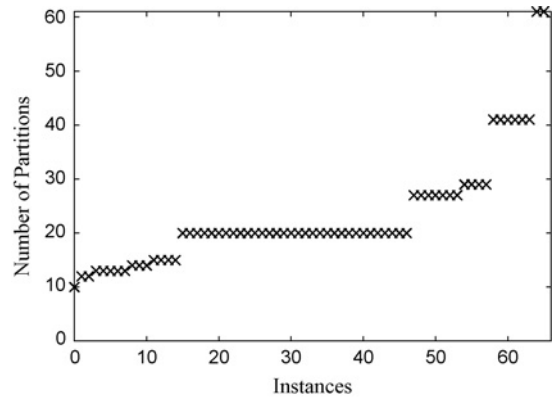

Fig. 13. Result statistics: number of partitions per instance.

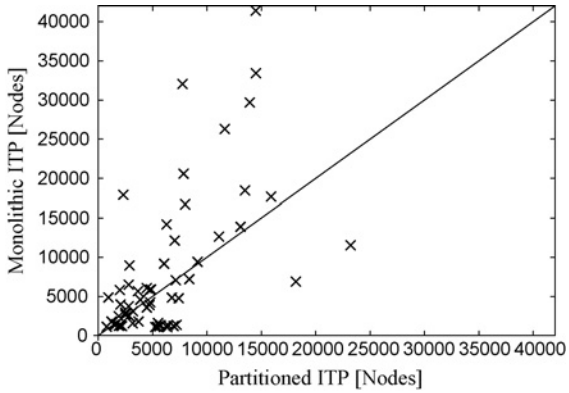

Fig. 14. Result statistics: size of the partitioned versus monolithic interpolants (AIG nodes).

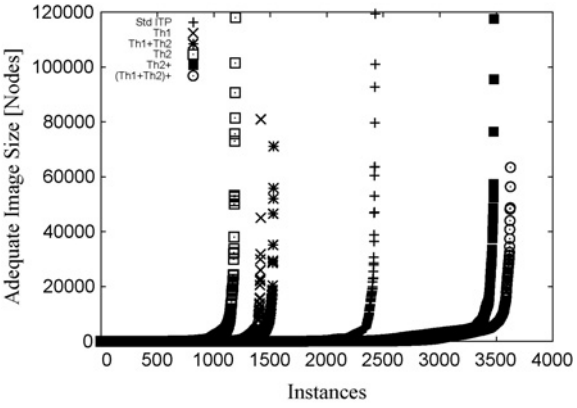

Fig. 15. Result statistics: size for all adequate images (AIG nodes) computed in Table I.

partitions are due to properties that can be conjunctively decomposed.

Fig. 14 plots the size of partitioned interpolants versus monolithic ones, for all the available instances. The x-axis (y-axis) represents the size of the interpolant when this is computed with the partitioned (monolithic) strategy. This plot is very expressive. It shows that for monolithic interpolants (ITP_{mon}) with size ranging from 10 000 to 50 000 nodes, their partitioned counterparts (ITP_{part}) are about two to five times smaller. The average size ratio ($|ITP_{\text{mon}}|/|ITP_{\text{part}}|$) for all considered instances is 1.97. Notice that the plot does not represent two instances in which either the monolithic or the partitioned interpolant size is far outside the area considered in the figure (<50 000 nodes). For the sake of completeness, for these cases the ratio $|ITP_{\text{mon}}|/|ITP_{\text{part}}|$ is equal to 0.36 and 40.61, respectively.

Finally, the cactus plot of Fig. 15 presents statistics on the number and size of all adequate images computed with the analyzed methods, in all the cases in which those methods resort to interpolant computation. We compare all methods shown in Table I. The plots Th_1, Th_2, and $Th_1 + Th_2$ stand on the left of the one generated by the original method. The values on the y-axis show that interpolant sizes are generally smaller, except for the case labeled as Th_2, where partitioned interpolants are disabled. Smaller values on the x-axis are partly due to aborted verifications, as well as to different over-approximation levels attained by the optimized methods. In other words, we often reach state sub-spaces in fewer forward traversal steps. Moreover, the diagrams labeled with Th_2^+ and $(Th_1 + Th_2)^+$ confirm that the use of (state-based) care sets improves interpolant computations. Images are usually smaller and the number of computations larger, as the methods can proceed deeper into the verification analysis (eventually proving more properties, as detailed in Table I).

VII. Conclusion

This paper shows how to improve interpolant-based model checking by means of an integrated approach combining partial quantification, sub-setting, disjunctive partitioning, and interpolation. The core idea of this combination is to adopt quantifications and circuit-based representations of sub-spaces whenever convenient (and not too expensive). The obtained advantages derive from a limitation and a restriction of the search space of interpolant-based methods.

Experimental results, specifically oriented to hard verification problems, show the robustness of our approach, with improvements in terms of CPU time up to two orders of magnitude.

Among the possible future works, we report the possibility to extend the approach presented in this paper to word-level, or at least to combine it with other word-level strategies.

References

[1] W. Craig, "Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory," *J. Symbolic Logic*, vol. 22, no. 3, pp. 269–285, 1957.

[2] R. C. Lyndon, "An interpolation theorem in the predicate calculus," *Pacific J. Math.*, vol. 9, no. 1, pp. 129–142, 1959.

[3] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proc. Comput.-Aided Verification*, LNCS 2725. Boulder, CO, 2003, pp. 1–13.

[4] G. Cabodi, P. Camurati, L. Garcia, M. Murciano, S. Nocco, and S. Quer, "Trading-off SAT search and variable quantifications for effective unbounded model checking," in *Proc. Formal Methods Comput.-Aided Design*, Portland, OR, Nov. 2008, pp. 205–212.

[5] P. Bjesse and K. Claessen, "SAT-based verification without state space traversal," in *Proc. Formal Methods Comput.-Aided Design*, LNCS 1954. Austin, TX, 2000, pp. 409–426.

[6] M. L. Case, A. Mishchenko, and R. K. Brayton, "Inductively finding a reachable state space over-approximation," in *Proc. Int. Workshop Logic Synthesis*, Lake Tahoe, CA, May 2006.

[7] F. Lu and K. T. Cheng, "IChecker: An efficient checker for inductive invariants," in *Proc. High-Level Design Validation Test Workshop*, 2006, pp. 176–180.

[8] G. Cabodi, S. Nocco, and S. Quer, "Boosting the role of inductive invariants in model checking," in *Proc. Design Autom. Test Eur. Conf.*, Nice, France, Apr. 2007, pp. 1319–1324.

[9] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and an SAT solver," in *Proc. Formal Methods Comput.-Aided Design*, LNCS 1954. Austin, TX, Nov. 2000, pp. 108–125.

[10] P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta, "Combining decision diagrams and SAT procedures for efficient symbolic model checking," in *Proc. Comput.-Aided Verification*, LNCS 2102. Chicago, IL, Jul. 2000, pp. 124–138.

[11] P. A. Abdulla, P. Bjesse, and N. Een, "Symbolic reachability analysis based on SAT-solvers," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 1785, S. Graf and M. I. Schwartzbach, Eds. Berlin, Germany: Springer-Verlag, Apr. 2000, pp. 411–425.

[12] K. L. McMillan, "Applying SAT methods in unbounded symbolic model checking," in *Proc. Comput.-Aided Verification*, LNCS 2404. Copenhagen, Denmark, 2002, pp. 250–264.

[13] H. J. Kang and L. C. Park, "SAT-based unbounded symbolic model checking," in *Proc. 40th Design Autom. Conf.*, Anaheim, CA, 2003, pp. 840–843.

[14] M. K. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, Nov. 2004, pp. 510–517.
[15] A. Biere and T. Jussila. (2007, Jul. 7). *The model checking competition web page* [Online]. Available: http://fmv.jku.at/hwmcc07
[16] A. Biere and T. Jussila. (2008, Jul. 13). *The model checking competition web page* [Online]. Available: http://fmv.jku.at/hwmcc08
[17] K. L. McMillan and R. Jhala, "Interpolation and SAT-based model checking," in *Proc. Comput.-Aided Verification*, LNCS 3725. Edinburgh, U.K., 2005, pp. 39–51.
[18] J. Marques-Silva, "Improvements to the implementation of interpolant-based model checking," in *Proc. Correct Hardware Design Verification Methods*, LNCS 3725. Edinburgh, U.K., 2005, pp. 367–370.
[19] J. Marques-Silva, "Interpolant learning and reuse in SAT-based model checking," *Electron. Notes Theor. Comput. Sci.*, vol. 174, no. 3, pp. 31–43, May 2007.
[20] V. D'Silva, M. Purandare, and D. Kroening, "Approximation refinement for interpolation-based model checking," in *Proc. 9th Int. Conf. Verification Model Checking Abstract Interpretation*, LNCS 4905. 2008, pp. 68–82.
[21] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. 36th Design Autom. Conf.*, New Orleans, LA, Jun. 1999, pp. 317–320.
[22] G. Cabodi, M. Crivellari, S. Nocco, and S. Quer, "Circuit based quantification: Back to state set manipulation within unbounded model checking," in *Proc. Design Autom. Test Europe Conf.*, Munich, Germany, Mar. 2005, pp. 688–689.
[23] P. Bjesse and A. Boralv, "DAG-aware circuit compression for formal verification," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, Nov. 2004, pp. 42–49.
[24] P. Pudlák, "Lower bounds for resolution and cutting plane proofs and monotone computations," *J. Symbolic Logic*, vol. 62, no. 3, pp. 981–998, 1997.
[25] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proc. 34th Design Autom. Conf.*, Anaheim, CA, Jun. 1997, pp. 263–268.
[26] G. Cabodi, P. Camurati, and S. Quer, "Auxiliary variables for BDD-based representation and manipulation of Boolean functions," *ACM Trans. Design Autom. Electron. Syst.*, vol. 3, no. 3, pp. 309–340, Jul. 1998.
[27] K. Ravi and F. Somenzi, "High-density reachability analysis," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, Nov. 1995, pp. 154–158.
[28] M. Mneimneh and K. Sakallah, "SAT-based sequential depth computation," in *Proc. Int. Conf. Asia South Pacific Design Autom.*, New York, 2003, pp. 87–92.
[29] A. Kuehlmann, M. K. Ganai, and V. Paruthi, "Circuit-based Boolean reasoning," in *Proc. Design Automation Conf.*, Las Vegas, NV, Jun. 2001, pp. 232–237.
[30] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't-cares," in *Proc. Design Autom. Conf.*, 2006, pp. 229–234.
[31] G. Cabodi, M. Murciano, S. Nocco, and S. Quer, "Boosting interpolation with dynamic localized abstraction and redundancy removal," *ACM Trans. Design Autom. Electron. Syst.*, vol. 13, no. 1, pp. 309–340, Jan. 2008.
[32] R. K. Brayton, S. Chatterjee, and A. Mishchenko, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *Proc. Design Autom. Conf.*, 2006, pp. 532–536.
[33] F. Somenzi. (2009, Aug. 1). *CUDD: CU decision diagram package—release 2.4.1* [Online]. Available: http://vlsi.colorado.edu/~fabio/CUDD
[34] N. Eén and N. Sörensson. (2009, Aug. 1). *The minisat SAT solver* [Online]. Available: http://minisat.se
[35] A. Mishchenko. (2009, Aug. 1). *ABC: A system for sequential synthesis and verification* [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc

Gianpiero Cabodi received the M.S. degree in electrical engineering and computer science and the Ph.D. degree in information and system engineering from the Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy, in 1984 and 1989, respectively.

He previously was a System Manager, from 1989 to 1992, and a Research Assistant, from 1992 to 1998, with the Politecnico di Torino. Since 1998, he has been an Associate Professor with the Dipartimento di Automatica e Informatica, Politecnico di Torino. His research interests include hardware description languages, binary decision diagrams, satisfiability, formal verification, logic and high level synthesis, and embedded systems.

Luz Amanda Garcia received the M.S. degree in electrical engineering and computer science and the Ph.D. degree in information and system engineering from the Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy, in 2005 and 2009, respectively.

As a Ph.D. student, she was Assistant to computer engineering courses and Master courses. Currently, she is a Consultant with AleniaSIA, Torino, Italy, where she is working on the implementation of the computer control within the Galileo precise timing facility project. Her research interests include hardware symbolic model checking, scheduling, and verification and testing.

Marco Murciano received the M.S. degree in electrical engineering and computer science and the Ph.D. degree in information and system engineering from the Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy, in 2005 and 2009, respectively.

During his Ph.D. studentship, he joined the Formal Methods Group, Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy. As a Ph.D. student, he was involved both in national and international projects and also acted as an Assistant and Tutor for several daily university classes. From August to October 2006, he was an Intern with Cadence Design Systems, Inc., San Jose, CA. Since January 2009, he has been working as non-tenure track Post-Doctoral Fellow with the Formal Methods Group, Dipartimento di Automatica e Informatica, Politecnico di Torino. His research interests include hardware symbolic model checking, high level scheduling, specification, design, verification, and testing of embedded systems.

Sergio Nocco received the M.S. degree in electrical engineering and computer science and the Ph.D. degree in information and system engineering from the Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy, in 2001 and 2005, respectively.

During his studies, he performed several summer internships with companies, working with Nippon Electric Company, Minato, Tokyo, Japan, in 2001, Cadence Berkeley Labs, Berkeley, CA, in 2002, and Intel Corporation, Santa Clara, CA, in 2003, 2004, and 2005. Currently, he is a Post-Doctoral Fellow with the Formal Methods Group, Dipartimento di Automatica e Informatica, Politecnico di Torino. His research interests include the application of symbolic techniques, based on both Boolean satisfiability and binary decision diagrams, to hardware formal verification, high level synthesis, and game theory.

Stefano Quer received the M.S. degree in electrical engineering and computer science and the Ph.D. degree in information and system engineering from the Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy, in 1991 and 1996, respectively.

In 1994, he was with the Department of Electronic Engineering and Computer Science, University of California, Berkeley. In 1998, he collaborated with the Advanced Technology Group, Synopsys, Inc., Mountain View, CA, and in 1999 with the Alpha Development Group, Compaq Computer Corporation, Shrewsbury, MA. He has been a Consultant for Compaq Computer Corporation, and an Assistant Professor with the Dipartimento di Automatica e Informatica, Politecnico di Torino. He is currently an Associate Professor with the Dipartimento di Automatica e Informatica, Politecnico di Torino. His research interests include hardware description languages, logic synthesis, formal verification, and the simulation and testing of digital circuits and systems.