Interpolation with Guided Refinement: revisiting incrementality in SAT-based Unbounded Model Checking

G. Cabodi, M. Palena, P. Pasini
Dipartimento di Automatica ed Informatica
Politecnico di Torino - Torino, Italy
Email: {gianpiero.cabodi, marco.palena, paolo.pasini}@polito.it

***Abstract*—This paper addresses model checking based on SAT solvers and Craig interpolants. We tackle major scalability problems of state-of-the-art interpolation-based approaches, and we achieve two main results: (1) a novel model checking algorithm; (2) a new and flexible way to handle an incremental representation of (over-approximated) forward reachable states. The new model checking algorithm (IGR: Interpolation with Guided Refinement), partially takes inspiration from IC3 and interpolation sequences. It bases its robustness and scalability on incremental refinement of state sets, and guided unwinding/simplification of transition relation unrollings. State sets, the central data structure of our algorithm, are incrementally refined, and they represent a valuable information to be shared among related problems, either in concurrent or sequential (multiple-engine or multiple property) execution schemes. We provide experimental data, showing that IGR extends the capability of a state-of-the-art model checker, with a specific focus on hard-to-prove properties.**

I. Introduction

Craig interpolants (ITPs for short) [1], [2], introduced by McMillan [3] in the Unbounded Model Checking (UMC) field, have shown to be effective on difficult verification instances. Though recently challenged by new techniques (IC3, Incremental Construction of Inductive Clauses for Indubitable Correctness [4]), our experience within the field of HWMCC competitions [5] and industrial co-operations shows that interpolation-based approaches still play an important role within a portfolio-based tool.

From a high-level Model-Checking perspective, Craig interpolation is an operator able to compute over-approximated images. The approach can be viewed as an iterative refinement of proof-based abstractions, to narrow down a proof to relevant facts. Over-approximations of the reachable states are computed from refutation proofs of unsatisfied BMC-like runs, in terms of AND/OR circuits, generated in linear time and space, w.r.t. the proof.

Craig interpolants most interesting features are their completeness and the automated abstraction mechanism. Whereas one of their major challenges is the inherent redundancy of interpolant circuits, as well as the need for fast and scalable techniques to compact them. Improvements over the base method [3] were proposed in [6], [7], [8], [9], [10] and [11], in order to push forward applicability and scalability of the technique.

Interpolant compaction is a potential approach that we have specifically addressed in [12]. We follow here a second track of research: alternative ITP-based traversal schemes for model checking algorithms, under the underlying purpose of incrementally computing state sets and reducing the complexity of their computation. We also follow the idea of incrementality in order to support optimal data structures for the verification of multiple properties, and for a tighter integration with counterexample- and/or proof-based [13], [14] abstraction/refinement approaches.

A. Contributions

The main contributions of this work are: (1) A novel model checking algorithm based on interpolation and characterized by: incremental computation of state sets, guided deployment and simplification of transition relation unrollings; (2) Internal optimizations to image computation, exploiting the incremental state representation; (3) A new and flexible way to compute and refine state set representations.

B. Related works

Our work is related to many recent papers on SAT-based Model Checking. Among others, let us mention that the idea of guided search and refinement is clearly present in some past BDD-based works (see for instance [15]), in IC3 [4], as well as in interpolation sequences (ITPSEQ [16], [17]). More recently, Vizel et al. [18] have proposed *Dual Approximated Reachability* (DAR), an evolution of interpolation sequences that considers mixing forward and backward reachability. Our approach takes ideas from all above works, it is based on interpolation, it computes just forward approximations of state sets, which allows us to potentially reuse them for multiple properties (or sub-properties) of the same model.

Our scheme of incremental refinement of state sets takes equal inspiration from IC3 and ITPSEQ. Compared to IC3, we represent state sets by circuits instead of clauses, and our state sets relax inductiveness constraints. Compared to interpolation sequences, though our refinement scheme is similar, we never compute an interpolation sequence from a single SAT run (and

[1]This work was supported in part by SRC contract 2012-TJ-2328.

proof), but we activate sequences of standard interpolation and/or approximate image calls.

Many other internal details, at the level of SAT and circuit-based reasoning, take inspiration from the above, as well as other existing works. Let us mention for instance clause propagation by *pushing*, redundancy removal by subsumption, that we brought from IC3 and re-implemented on circuit-based (AIG) representations.

C. Outline

Section II introduces background notions and notation about BMC and UMC, SAT-based Craig interpolant Model Checking, IC3. Sections III, IV, V introduce our contributions. Section III discusses incremental state sets in interpolation, section IV introduces base concept on guiding cones through state sets, section V presents the overall IGR algorithm. Section VII discusses the experiments we performed. Section VIII concludes with some summarizing remarks.

II. Background

A. Model and Notation

We address systems modeled by labeled state transition structures and represented implicitly by Boolean formulas. From our standpoint, a system M is a triplet $M = (S, S_0, T)$, where S is a finite set of states, $S_0 \subseteq S$ is the set of initial states, and $T \subseteq S \times S$ is a total transition relation. The system state space is encoded with an indexed set of Boolean variables $X = \{x_1, \ldots, x_n\}$, so that a state $s \in S$ corresponds to a valuation of the variables in X, and a set of states can be represented with a Boolean formula over X. We use the primed notation (X') for the *next state* of a variable (so a transition relation is $T(X, X')$). Whenever more time frames are involved, we use a superscript notation: e.g., in circuit unrollings, we use X^i for the X variables instantiated at the i-th time frame. Support variables will be omitted for simplicity when easily guessed from the context. A literal is a Boolean variable or its negation. A clause is a disjunction of literals. A CNF formula is a conjunction of clauses. Most modern SAT solvers [19], [20] adopt clauses as their main representation and manipulation formalism for Boolean functions. Given a Boolean formula $\mathcal{F}$, whenever we need to explicitly indicate its *before/after* version, w.r.t. an evaluation (e.g., a refinement step), we use a -1 superscript for the *before* version: $\mathcal{F}^{-1}$. We will use overlined letters for arrays of functions: e.g., $\overline{\mathcal{F}} = (\mathcal{F}_0, \mathcal{F}_1, ...)$.

B. Bounded and Unbounded Model Checking

Given a sequential system M and an invariant property p, SAT-based BMC [21] is an iterative process to check the validity of p up to a given bound. To perform this task, the transition relation T is unrolled k times

$$T^k(X^{0..k}) \quad = \quad \bigwedge_{i=0}^{k-1} T(X^i, X^{i+1})$$

in order to implicitly represent all state paths of length k. BMC tools use SAT checks such as:

$$bmc^k(X^{0..k}) \quad = \quad S_0(X^0) \wedge T^k(X^{0..k}) \wedge \bigvee_{i=0}^{k} \neg p(X^i)$$

to look for counterexamples (of length $\leq k$) that start from the initial states S_0 and falsify p. The same formula can be rewritten, in a simpler form, by omitting support variables, as follows:

$$bmc^k \quad = \quad S_0 \wedge T^k \wedge \bigvee_{i=0}^{k} \neg p$$

Though BMC tools are effective at finding bugs, their verification method is not complete. Therefore, specific techniques are required in order to support Unbounded Model Checking (UMC). The ability to check reachability fix-points and/or to find inductive invariants, is thus the main difference, and additional complication, between BMC and UMC.

C. Craig Interpolants

Let A and B be two inconsistent Boolean formulas, i.e., such that $A \wedge B \equiv \bot$. An ITP I for (A, B) is a formula such that: (1) $A \Rightarrow I$, (2) $I \wedge B \equiv \bot$, and (3) $supp(I) \subseteq supp(A) \cap supp(B)$.

```
InterpolantMC ($S_0$, $T$, $\neg p$)
   k = 0
   do
      $Cone^k$ = ConeUnroll($\neg p, T, k$)
      res = FiniteRun ($S_0$, $T$, $Cone^k$)
      k = k + 1
   while (res = undecided)
   return (res)

FiniteRun ($S_0$, $T$, $Cone$)
   if Sat($S_0 \wedge T \wedge Cone$) return (reachable)
   R = $S_0$
   while ($\top$)
      $Image$ = Itp($R \wedge T, Cone$)
      if ($Image$ = undefined)
         return (undecided)
      if ($Image \Rightarrow$ R) return (unreachable)
      R = R $\vee$ $Image$
```

Fig. 1. Interpolant-based Verification.

An interpolant $I = \text{Itp}(A, B)$ can be derived, as an AND/OR circuit, from the refutation proof of $A \wedge B$. McMillan [3] proposed an effective fully SAT-based Unbounded Model Checking algorithm, exploiting interpolants, as sketched in Figure 1.

Routine FiniteRun operates a forward traversal, where interpolation is used as an over-approximate image operator. The degree of accuracy or abstraction of the operation is tied to the bound K of the $Cone^{0..k}$ transition relation unrolling. Whenever the product ($S_0 \wedge T \wedge Cone$) is UNSAT, we say that S_0 and $Cone$ are mutually *adequate*. The function may end up with three possible results:

- *reachable*, if it proves $\neg p$ reachable in k steps, hence the property has been disproved;
- *unreachable*, if the approximate traversal using the Img^{+}_{Adq} image computation reaches a fix-point. In this case the property is proved;
- *undecided*, if $\neg p$ intersects the over-approximate state sets. Then, k in increased for a new FiniteRun call.

Routine INTERPOLANTMC, on top of FINITERUN, loops through increasing k bound values. The previous algorithm is sound and complete [3].

D. IC3

IC3 [4] is based on incrementally refining and extending a sequence $\mathcal{F}_1, ..., \mathcal{F}_k$ of sets of reachable states (*bounded invariants*) represented by sets of clauses, under the following rules:

$$\begin{array}{lll} \mathcal{F}_0 & = & S_0 \\ \mathcal{F}_i & \Rightarrow & \mathcal{F}_{i+1},\ for 1 \leq i \leq k-1 \\ \mathcal{F}_i & \supseteq & \mathcal{F}_{i+1} \text{ as sets of clauses, for} 1 \leq i \leq k-1 \\ \mathcal{F}_i \wedge T & \Rightarrow & \mathcal{F}_{i+1}, \text{ for } 1 \leq i \leq k-1 \\ \mathcal{F}_i & \Rightarrow & p. \end{array}$$

The introduction of IC3 [4] suggested a different way to compute information about reachable states, as (unlike other ITP-based approaches) IC3 requires no unrolling of the transition relation. One of the major contributions of IC3 is an inductive reasoning, where induction is exploited under stepwise assumptions-assertions. IC3 is incremental in that it finds inductive subclauses of the negations of states. The main limitation of IC3 is the potential clause-based state set enumeration. Some interesting ideas of IC3, that partially influenced our work, are:

- the incremental representation of state sets;
- the *push* operation, that possibly re-uses clauses from inner state sets to outer ones;
- redundancy removal by subsumption.

III. INCREMENTAL STATE SETS IN ITP

In this section we describe our model of incremental state sets. Instead of directly introducing the overall IGR algorithm (see section V), we just propose here some modifications to the standard interpolation algorithm of [3], that would allow reusing and refining previously computed interpolants.

As already pointed out, incremental state sets are present in ITPSEQ [16], [17] and DAR [18]. Compared to those works, our approach, as described in the sequel, is much closer to standard interpolation. More in detail:

- we just work on approximations of forward reachable states, with no attempt to mix forward and backward state sets (as in DAR);
- we keep the standard interpolation scheme, extended by saving and reusing previously computed state sets;
- we always refine (i.e., strengthen) state sets, which does not prevent us from possibly simplifying their representation by using ad–hoc redundancy removal.

We use for state sets a notation taken from IC3 and ITPSEQ. $\overline{\mathcal{F}} = \mathcal{F}_1, ..., \mathcal{F}_k$ is a sequence of sets of reachable states represented by circuits (AIGs) instead of sets of clauses. Let $\mathcal{R}_i^E$ represent the set of states reachable in *exactly* i steps, and $\mathcal{R}_i = \cup_{j=0..i} \mathcal{R}_j^E$ the sets of all states reachable in at most i steps. $\mathcal{R}_i$ includes all previous state sets, whereas $\mathcal{R}_i^E$ does not necessarily.

Our implementation supports both versions:

$$\begin{array}{lll} \mathcal{R}_i^E & \Rightarrow & \mathcal{F}_i \\ \mathcal{R}_i & \Rightarrow & \mathcal{F}_i \end{array}$$

the choice being a user selected option[1]. On the one hand, the fully inclusive ($\mathcal{R}_i$) representation has nice properties, which are at the base of the IC3 inductive reasoning. On the other hand, state set strengthening is generally more powerful using $\mathcal{R}_i^E$. In the sequel we will assume the first (non inclusive) model. So our assumptions for the $\mathcal{F}_i$ sets are the following:

$$\begin{array}{lll} \mathcal{F}_0 & = & S_0 \\ \mathcal{F}_i(X) \wedge T(X, X') & \Rightarrow & \mathcal{F}_{i+1}(X'), \text{ for } 1 \leq i \leq k-1 \end{array}$$

In order to represent an incremental refinement of $\mathcal{F}_i$ sets, we use notation $\mathcal{F}_i^{-1}$ for denoting the version of $\mathcal{F}_i$ prior to refinement. A refinement of $\mathcal{F}_i^{-1}$ is thus the result of a strengthening step, such that: $\mathcal{F}_i \Rightarrow \mathcal{F}_i^{-1}$.

```
INCRITPMC ($S_0$, $T$, $\neg p$)
  k = 0
  $\overline{\mathcal{F}} = (S_0)$
  do
    $Cone^k$ = CONEUNROLL($\neg p, T, k$)
    res = INCREMENTALFINITERUN ($\overline{\mathcal{F}}$, $T$, $Cone^k$)
    k = k + 1
  while (res = undecided)
  return (res)

INCRFINITERUN ($\overline{\mathcal{F}}$, $T$, $Cone$)
  if SAT($\mathcal{F}_0 \wedge T \wedge Cone$) return (reachable)
  R = $\mathcal{F}_0$
  $i = 0$
  while ($\top$)
    if ($\mathcal{F}_{i+1}$ = void) $\mathcal{F}_{i+1}$ = $\top$
    $Image$ = IMGREF($\mathcal{F}_i, T, Cone, \mathcal{F}_{i+1}$)
    if ($Image$ = undefined)
      return (undecided)
    $\mathcal{F}_{i+1}$ = $Image$
    if ($\mathcal{F}_{i+1} \Rightarrow$ R) return (unreachable)
    R = R $\vee$ $\mathcal{F}_{i+1}$
    $i = i + 1$

IMGREF ($\mathcal{F}_i$, $T$, $Cone$, $\mathcal{F}_{i+1}^{-1}$)
  $C$ = SIMPLIFY ($Cone, \mathcal{F}_{i+1}^{-1}$)
  $Image$ = ITP($\mathcal{F}_i \wedge T, C$)
  if ($Image$ = undefined) return ($Image$)
  return ($Image \wedge$ SIMPLIFY ($\mathcal{F}_{i+1}^{-1}, Image$)
```

Fig. 2. Interpolant-based Image with refinement.

Figure 2 shows a variant of the algorithm in Figure 1. We explicitly use $\overline{\mathcal{F}}$ to represent state sets. $\overline{\mathcal{F}}$ is initialized to an empty array, with the exception of $\mathcal{F}_0 = S_0$. The standard interpolation operator is replaced here by IMGREF. In this new operator interpolation is preceded by cone simplification, based on previously available state sets, and followed by a refinement step. Refinement is a strenghtening step, done by

[1]The $\mathcal{R}_i$ option is internally handled by properly transforming the transition relation.

conjoining the previous set with a new term. This is done by embedding a simplification step. SIMPLIFY is based on the general notions of redundancy removal under *external* or *observability don't cares*. Strictly speaking, whenever two functions are conjoined, either one could be pre-simplified using the other one as *care*:

$$A \wedge B \quad = \quad A \wedge \text{SIMPLIFY}(B, A) \qquad (1)$$

BDDs offered nice operators (cofactor, constrain, restrict [22]) for function simplification, that have no counterpart in gate-based rapresentations. Though many redundancy removal operators have been proposed, our experience shows that most of them are too expensive (and poorly scalable). As we need a fast operator, we limit ourselves to equivalences involving state variables, exploited for simplifications based on circuit merging.

We now prove that incrementality is guaranteeing the correctness of the Model Checking procedure. The proof is based on **Theorem 1**, stating that IMGREF (including the refinement step) is returning a correct over-approximated image.

Theorem 1 IMGREF is correct $\mathcal{F}_i \wedge T \Rightarrow$ IMGREF ($\mathcal{F}_i$, T, $Cone$, $\mathcal{F}_{i+1}^{-1}$)

Proof. Let us start by the observation that both the previous $\mathcal{F}_{i+1}^{-1}$ (assumed as $\top$ if not yet available) and $Image$ in IMGREF are implied by the exact image.

$$\begin{array}{lccl} (a) & \mathcal{F}_i \wedge T & \Rightarrow & \mathcal{F}_{i+1}^{-1} \\ (b) & \mathcal{F}_i \wedge T & \Rightarrow & Image \end{array}$$

(a) is true because $\mathcal{F}_i$ is a strengthening of its previous version $\mathcal{F}_i^{-1}$ ($\mathcal{F}_i \Rightarrow \mathcal{F}_i^{-1}$), so its image implies the image of $\mathcal{F}_i^{-1}$. (b) comes from $Image$ being an interpolant. By conjoining (a) and (b), we can derive:

$$(c) \quad \mathcal{F}_i \wedge T \quad \Rightarrow \quad \mathcal{F}_{i+1}^{-1} \wedge Image$$

From the definition of the SIMPLIFY operator 1:

$$(d) \quad Image \wedge \text{SIMPLIFY}(\mathcal{F}_{i+1}^{-1}, Image) \equiv Image \wedge \mathcal{F}_{i+1}^{-1}$$

The thesis comes from combining (c) and (d).

IV. GUIDED CONE

Let us identify a refinement step as a strengthening of a state set $\mathcal{F}_{i+1}^{-1}$, such that the new version implies the previous one: ($\mathcal{F}_{i+1} \rightarrow \mathcal{F}_{i+1}^{-1}$). We describe here how incrementality can exploit the fact that any subset of adequate backward cones can be used for refinement, based on two observations: (A) convergence of the approach is guaranteed by the fact that at worst a full cone of bound equal to the diameter is eventually used (see [3]), or the full enumeration of used cone subsets could completely cover the space backward reachable from the target ($\neg p$) (see [4]); (B) performance issues require a good balance between the opposite needs, to (1) keep small cones, for easier BMC-like SAT checks, and (2) to avoid activating too many refinement steps. We also need to avoid using cones that do not help refining previously computed state sets.

Let us thus start from the observation that any (subset of a) backward cone is acceptable by a refinement step (a call to IMGREF), as the cone is not required by the proof of **Theorem 1**. Of course, no refinement ($\mathcal{F}_{i+1} = \mathcal{F}_{i+1}^{-1}$) could come from a wrongly chosen cone, leading to explosion in the number of iterations. As an extreme option, any state cube backward reachable from the target (or known not to be forward reachable) could be used, as in IC3. Though cone subsetting is an option in view of scalability, it is not a primary focus of this work. Cone partitioning and/or subsetting would obviously reduce the size and depth of BMC-like checks, whose number whould increase. In this paper limit ourselves to cone simplification and guided rewinding/unwinding, see IV-B, exploiting previously computed $\overline{\mathcal{F}^{-1}}$ whenever available in order to:

- simplify $Cone$, using available $\overline{\mathcal{F}^{-1}}$ sets (in other words, restricting cones to *go into* the known state set rings);
- drive $Cone^k$ computation to a proper k depth, i.e., the minimum required in order to produce a strengthening.

A. Cone simplification

Whenever we are computing the image of $\mathcal{F}_i$, exploiting previously computed $\mathcal{F}_j^{-1}$ ($j > i$), we can use all available $\mathcal{F}_j^{-1}$ as *care sets* for $Cone$ simplification, based on the fact that the image will be conjoined with $\mathcal{F}_{i+1}^{-1}$.

So, $Cone^k$ in IMGREF can be replaced by FSIMPLIFY($Cone^k$, $\overline{\mathcal{F}}$, $i+1$), under the constraint that:

$$\text{FSIMPLIFY}(Cone^k, \overline{\mathcal{F}^{-1}}, j) \wedge \mathcal{F}_j^{-1} \quad \equiv \quad Cone^k \wedge \mathcal{F}_j^{-1}$$

A straightforward application of the previous formula is based on the so called *latch correspondences*, i.e., couples of latches that are known to be equivalent in $\mathcal{F}_j^{-1}$. For all of them, latches can be merged in $\mathcal{F}_j^{-1}$. More formally, for each couple of state variables (x_p, x_q) such that $\mathcal{F}_j^{-1} \Rightarrow (x_p \leftrightarrow x_q)$, the substitution $x_p \rightarrow x_q$ can be done in $Cone^k$. A similar operation can be done for all latch correspondences at intermediate transition relation boundaries in $Cone$. So for any known $\mathcal{F}_l^{-1}$ ($j < l < j + k$), implied equivalences can be used to simplify $Cone$.

A proof of correctness of the above steps is omitted for conciseness.

B. Guiding cones through state sets

Whenever INCREMENTALFINITERUN hits $Cone^k$ (a possibly false counterexample) at step i, standard interpolation would expand the cone by incrementing k, possibly by more than 1, and restart a new run from the initial state $\mathcal{F}_0$. Different ideas are followed in ITPSEQ and DAR, where refinements can be triggered based on BMC-like runs with growing depth. IC3, instead, drives refinements based on a prioritized selection of backward reachable cubes. In IGR, we follow two directions, that share the common goal of potentially expanding, by adding new frames, and refining, by strengthening, $\overline{\mathcal{F}}$:

- resuming forward traversal (and state refinements) with a *smaller* cone;
- restarting a new traversal at an intermediate step, such that a strengthening of the current $\overline{\mathcal{F}}$ is guaranteed.

1) Cone Rewinding: We call refinement sequence an iterated tail of INCREMENTALFINITERUN, that optionally resumes, after hitting $Cone^k$ at iteration i, by iteratively using cones with decreasing bounds. The operation is inspired by interpolation sequences, with a specific reference to [17].

More formally, let us assume that:

$$\mathcal{F}_i \wedge T \wedge Cone^k \not\equiv \bot$$

We could resume the forward iteration using $Cone^{k-1}$, as it is guaranteed that:

$$\mathcal{F}_i \wedge T \wedge Cone^{k-1} \equiv \bot \tag{2}$$

based on the observation that $\mathcal{F}_i \wedge Cone^k \equiv \bot$ and that $Cone^k = T \wedge Cone^{k-1}$. We could thus operate up to k iterations, until $Cone^0$, and generate or refine state sets $\mathcal{F}_{i+1}..\mathcal{F}_{i+k}$. As an alternative, one could compute an interpolation sequence, directly from a single BMC call on problem 2. As observed in [17], we prefer an iterative computation of interpolants starting from previously computed ones.

Any sub-setting of $Cone^k$, that guarantees unsatisfiability, could be selected (instead of decrementing k). E.g., if the cone is generated by a set of properties/targets, one could remove the satisfied ones, and just keep the unsat subset.

2) Cone Unwinding: Given an abstract counterexample (a cone hit) at iteration i, the *cone rewinding* strategy has the effect of refining state sets from $\mathcal{F}_{i+1}$ to $\mathcal{F}_{i+k}$. Let us now find a (minimal) unwinding of $Cone^k$ that insures to refine $\mathcal{F}_i$ and other ones at lower i values.

Starting from the observation that $bmc^{i+k} \not\equiv \bot$ (i.e., the BMC problem of depth $i + k$ is SAT) would confirm the abstract counterexample as a concrete one, and that $bmc^{i+k} = 0$ would refute it, we can iteratively produce BMC problems of increasing bound, starting from k, until we obtain UNSAT.

Let ν ($0 < \nu < i$) be the minimum cone unwinding, from $Cone^k$ to $Cone^{\nu+k}$, such that, with $j = i - \nu - 1$:

$$\mathcal{F}_j \wedge T \wedge Cone^{\nu+k} \equiv \bot \tag{3}$$

We can restart the next INCREMENTALFINITERUN from $\mathcal{F}_j$, using cone $Cone^{\nu+k}$. From a more practical point of view, we are unwinding $Cone$ in a guided way through $\mathcal{F}_j$ state sets, in order to fine the outermost one able to provide an UNSAT BMC problem (against the unwinded $Cone$).

Alternative options, for the choices of j and ν, include going to larger ν values, combined with j values such that $j < i - \nu$, and that Equation 3 is still UNSAT.

Overall, guided cone unwinding/rewinding allows us to dynamically tune unrollings. In this respect, standard interpolation is too rigid, as refinement is always done by expanding cones and using them for newly restarted traversals. ITPSEQ introduces incrementality, but with a fixed and rigid scheme. Much more flexibility is present in DAR where local and global strengthening techniques introduce the notion of refinement just when and where needed. Although backward refinement in DAR has similarities with our approach, it is based on the idea of using over-approximated backward reachable states when refining forward reachable ones. Our approach, instead, is fully based on backward cones (i.e., T unrollings), in order to represent the backward exact behavior.

V. IGR: INTERPOLATION WITH GUIDED REFINEMENT

We now describe the overall IGR model checking procedure which combines the techniques mentioned in the previous sections. Figure 3 shows the top level function IGRMC, that iteratively chooses the bound k for an unwinded cone and activates IGRFINITERUN. The latter is a variant of INCRFINITERUN, that receives as additional parameters the index i of the $\mathcal{F}_i$ state set where to start a forward traversal, and the bound k, to be used for cone unwinding. The function returns the index i_{hit} of the state set where reachability hits a cone. At each iteration, i and k are properly computed by SEEKBESTUNSAT, starting from i_{hit} and k_{hit} (related to the previous abstract counterexample) Following the *cone unwinding* strategy described in section IV-B, the cone bound k is extended, and i is decremented, until an UNSAT BMC check is obtained. As a side effect, function SEEKBESTUNSAT also detects true failures whenever the unwinded cone hits $\mathcal{F}_0$ (this check has been removed from IGRFINITERUN). The overall task of IGRMC can thus be summarized as:

- Iteratively choose a starting $\mathcal{F}_i$ set and a cone $Cone^k$, unwinded in a guided manner throughout the (abstract) $\overline{\mathcal{F}}$ sets. This is done by function SEEKBESTUNSAT;
- Start a new forward traversal (INCRFINITERUN), that is expected to refine $\overline{\mathcal{F}}$ and filter out the last (abstract) counterexemple found within the $\mathcal{F}_{i_{hit}}$ state set.

INCRFINITERUN, though heavily based on the skeleton of FINITERUN and INCRFINITERUN (its variant supporting incremental state sets), is more flexible in selecting the starting point for a traversal and the backward cone:

- Traversals start at $\mathcal{F}_i$, with i received as parameter (see IGRMC), and reachable states are initialized as the union of all $\mathcal{F}_0..\mathcal{F}_i$ state sets;
- The backward cone is not kept constant as in FINITERUN. As in INCRFINITERUN, it is simplified exploiting $\mathcal{F}$ sets at outer indexes. It is kept until an abstract counterexample is generated, or a maximum number of iterations is reached. After that, $Cone$ is rewinded by one time frame at each iteration (see section IV-B).

Convergence is tested as in all interpolation-based approaches, based on set containment. The value of variable $ForceRewind$ is assigned as a set-up parameter that heuristically controls activation of cone rewinding. Whenever $ForceRewind = 0$, rewinding is always active, so the approach obtains a minimal refinement, and it mimics the effect of interpolation sequences. High values of $ForceRewind$ keep the k value constant until a hit, a scheme much closer to standard interpolation. We empirically observed that small values are better at small sequential depths, as they can produce more light-cost refinement steps.

Figures 4 and 5 report experimental data on a case study, circuit INTEL015 from [5], that we selected among the ones

```
IgrMC ($S_0$, $T$, $\neg p$)
    $i_{hit}$ = $k_{hit}$ = 0
    $\overline{\mathcal{F}} = (S_0)$
    do
        (res, $i$, $k$) = SeekBestUnsat($\neg p$, $T$, $i_{hit}$, $k_{hit}$)
        if (res = reachable)
            return (res)
        (res, $i_{hit}, k_{hit}$) = IgrFiniteRun ($\neg p$, $T$, $\overline{\mathcal{F}}$, $i$, $k$)
    while (res = undecided)
    return (res)

IgrFiniteRun ($\neg p$, $T$, $\overline{\mathcal{F}}$, $i$, $k$)
    R = $\bigcup_{l=0..i} \mathcal{F}_l$
    $rewindEnabled = \bot$
    while ($\top$)
        if ($\mathcal{F}_{i+1}$ = {}) $\mathcal{F}_{i+1}$ = $\top$
        if ($rewindEnabled \wedge k > 0$) k = k-1
        $Cone^k$ = ConeUnroll($\neg p, T, k$)
        $Cone^{\mathcal{F}}$ =FSimplify($Cone^k$, $\overline{\mathcal{F}}$, $i+1$)
        $Image$ = ImgRef($\mathcal{F}_i \wedge T, Cone^{\mathcal{F}}, \mathcal{F}_{i+1}$)
        if ($Image$ = undefined)
            if ($rewindEnabled$) return (undecided, $i$, $k$)
            $rewindEnabled = \top$
        else
            $\mathcal{F}_{i+1}$ = $Image$
            if ($\mathcal{F}_{i+1} \Rightarrow$ R) return (unreachable, -, -)
            R = R $\vee$ $\mathcal{F}_{i+1}$
            $i = i + 1$
            if ($i > ForceRewind$) $rewindEnabled = \top$

SeekBestUnsat($\neg p$, $T$, $i_{hit}$, $k_{hit}$)
    $i = i_{hit}$
    $k = k_{hit}$
    while ($i \geq 0 \wedge$ Sat($\mathcal{F}_i \wedge T \wedge$ ConeUnroll($\neg p, T, k$)))
        $i = i - 1$
        $k = k + 1$
    if ($i < 0$) return (reachable, -, -)
    return (undecided, $i$, $k$)
```

Fig. 3. Interpolation with Guided Refinement.

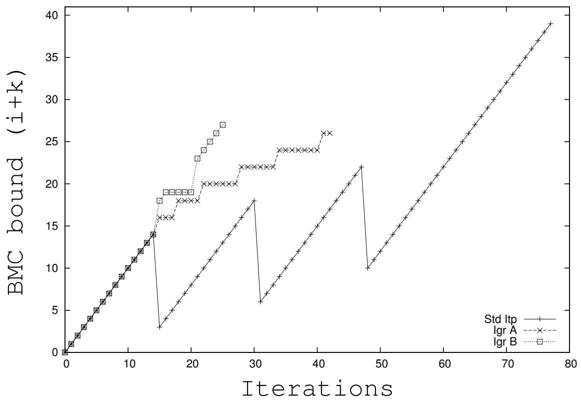

Fig. 4. BMC bound comparison in intel015, between standard interpolation and IGR in two versions: Igr A (rewind always enabled), Igr B (rewind disabled until hit).

where standard interpolation could be compared with IGR. Figure 4 plots $i + k$, the sum of state set indexes (i) and cone bounds (k). This is usually logged as an equivalent BMC

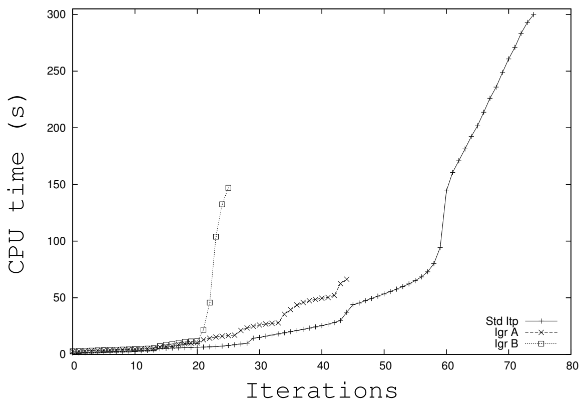

Fig. 5. CPU time comparison in intel015, between standard interpolation, Igr A and Igr B.

bound. Iterations (on the X axis) indicate algorithm iterations (with image computation). The standard interpolation line clearly shows that BMC bounds grow linearly within FiniteRun, and they restart from the newly adjusted cone bound[2] at new FiniteRun calls. The IGR A line plots a run of IGR with cone rewinding always enabled: this means that the iterative decrease of k compensates the increase of i, keeping the BMC bound constant within IgrFiniteRun (except when we reach $k = 0$). The IGR B line plots a run of IGR with cone rewinding disabled until a BMC hit. In this case we observe an initial increase of BMC bounds, followed by a phase with constant BMC bound. Overall, IGR exploits its ability to avoid restarting from low bounds and seeking for optimal restarts, which can provide convergence at lower iteration indexes.

A comparison between IGR A and B shows that the latter can converge in fewer iterations, due to its ability to increase BMC bounds. However figure 5, that plots cumulative CPU times, shows that IGR A can be faster.

Intuitively, guided and simplified cones in IGR can produce cheaper BMC problems, as compared to standard interpolation. IGR A benefits from triggering more, but possibly simpler, refinement steps (SAT calls). Although this is a good way to avoid highly expensive BMC problems, IGR B often performs better in case of models with higher diameters (e.g., in the range of hundredths).

3) Other Implementation Issues: A few more points are worth being noticed, as having an impact on performance:

- We implemented a light weight redundancy removal procedure used for Simplify when applied to state sets, inspired by clause subsumption. Whenever a set is a conjunction of several terms, the procedure iteratively finds redundant ones through an incremental SAT formulation;
- We implemented a SAT-based procedure able to partially reuse and *push* forward components of $\mathcal{F}_i$ to $\mathcal{F}_{i+1}$, whenever $\mathcal{F}_i$ is a conjunction. This process, which is similar to *clause pushing* in IC3, relies on an efficient incremental SAT formulation.

[2]Following [7], we heuristically increment cone bounds by more that 1, based on the depth of the previous FiniteRun run.

VI. Laxy Abstraction and Multiple Properties

Due to the fully incremental representation chosen for reachable states, IGR can be tightly embedded in lazy abstraction, as well as multiple property verification loops.

Typical abstraction-refimenent loops [13], [14] are based on the idea of looping through incremental model refinements, and restarting a new model checking problem after each new refinement step. Recent work [23] has explored a tighter integration of a model checking algorithm (IC3) within a lazy abstraction algorithm. As IGR is based on a similar data structure (the $\mathcal{F}_i$ over-approximations of forward reachable state sets), its integration within a lazy abstraction loop is straightforward: $\mathcal{F}_i$ state sets can be inherited by all refined models, as refinements can be considered as model strengthening steps. Let M^j and M^{j+1} be two abstract models (after refinement steps j and $j+1$). Let R_i^j and R_i^{j+1} be the states reachable by them in i steps. As refinement strengthens a model, $R_i^{j+1} \subseteq R_i^j$, so state set overapproximations for M^j also overapproximate states in M^{j+1}.

A similar framework can be adopted in multiple property verification, where $\mathcal{F}_i$ can be inherited and reused by all properties under check on the same model. Reusability of state sets is guaranteed here by sharing the same model over different property checks.

Though we already implemented both the above mentioned frameworks, their detailed description goes beyond the scope of this paper.

VII. Experimental Results

We implemented a prototype version of our methodology on top of the PdTRAV tool [24], a state-of-the-art verification framework. The experimental data in this section provide an evaluation of the techniques proposed, as well as a comparison with standard interpolation. Our experiments ran on a Quad-core workstation, with 2.5 GHz CPU frequency and 16 GB of main memory. We set time and memory limits to 1200 seconds and 2 GB, respectively.

We performed an extensive experimentation on a selected sub-set of publicly available benchmarks from the HWMCC'12 and HWMCC'13[5] suites. We selected them by excluding problems that PdTRAV could originally solve in less than 1 minute, and those that we could not solve with any technique (including the one presented here). It is worth noticing that all of the selected benchmarks are from industrial origin (IBM, Intel). In most cases, we operated a pre-processing using the ABC tool for combinational and/or sequential light weight optimizations, i.e., latch and signal correspondence, rewriting and refactoring. For the intel benchmarks, we also operated implicit invariant extraction and phase abstraction.

Table I provides detailed data, showing (column Best ITP) the best results we could obtain through standard interpolation, without the techniques described in this paper. Column Best IGR shows the best we could obtain with Igr, whereas column Best HWMCC shows best results attained during past HWMCCs. To this respect, it is worth noticing that time statistics from competitions were measured on a different machine, with a time limit of 900 seconds, by portfolio based (concurrent) model checkers. In the Best ITP and IGR experiments we used a single engine and we increased our time limit to 7200 seconds (2 hours), in an attempt to observe potentially difficult problems.

Table I highlights IGR as a clear winner with respect to standard interpolation, in most challenging problems. Higher runnign times in some of the easier examples simply witness some overhead for state set handling and cone winding/unwinding phases. Overall, IGR proves more scalable. The comparison with other engines is not as easy. To this respect it is worth noticing that the best model checkers at HWMCCs highly rely on aggressive transformational techniques, that seek to pre-simplify problems under various equivalence-preserving notions, before getting to Model Checking engines.

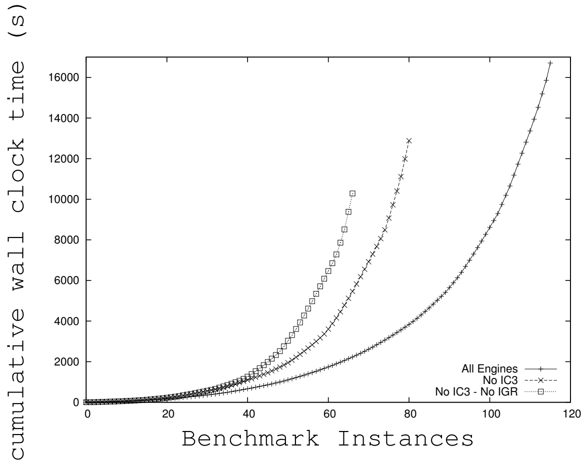

Fig. 6. Wall clock cumulative time comparison on hwmcc'12 instances solved by PdTrav (concurrent multi-engine version), with all engines active, without IC3, and without IC3 and IGR. Time limit 900 seconds per instance.

In order to gather more data, we did a second experimental evaluation of IGR, extended to the full set of HWMCC'12 (single property track, including more and easier benchmarks than HWMCC'13) benchmark instances. We repeated a competition run with 900 seconds time limit, using our multi-engine portfolio in three different setups: with the full set of engines, excluding IC3 and excluding both IC3 and IGR.

The results are plotted in figure 6, which clearly confirms IC3 as the most powerful engine. But it also shows a good impact of IGR, as a relevant contribution to the portfolio. The run with the full set of engines solved 116 problems, of which 47 were covered by IC3, and 10 by IGR. When disabling $IC3$, the overall result decreased to 81, with IGR solving 18 problems. Data also show that IGR is still not oriented to fast runs (within minutes). As seen in table I, a 2 hours timeout better shows the gain of IGR over ITP.

VIII. Conclusions

We addressed the problem of optimizing interpolants for SAT-based Unbounded Model Checking. Our main contribution is to provide a new approach, that improves over standard interpolation, by exploiting the ideas of incremental

Model				Best IGR	Best ITP	Best HWMCC	
Name	#PI	#FF	#AIG	Time	Time	Time	# of Solver
6s8	86	396	3016	835.40	-	147.82	4
6s34	77	1565	11098	2002.76	-	87.18	9
6s35	77	1571	11504	525.54	-	-	0
6s38	343	1931	10847	392.22	-	606.89	2
6s102	72	1108	7700	488.47	726.62	10.58	8
6s144	480	3337	45470	291.48	160.62	155.98	6
6s148	480	3337	45470	2011.54	1713.52	-	0
6s189	479	2436	39830	214.46	282.66	110.48	3
6s194	532	2131	13617	423.45	852.17	54.38	7
6s366r	86	1998	20560	612.28	-	-	0
6s428rb093	410	3790	29084	746.75	-	273.34	2
intel010	1111	280	10156	200.91	265.70	96.37	3
intel011	1024	273	9362	190.73	899.89	440.09	4
intel015	1024	273	9362	130.30	-	272.22	3
6s160(*)	149	559	8716	97700.21	-	-	0

TABLE I

RESULTS ON SELECTED HWMCC BENCHMARKS. COMPARING OUR BASIC VS. OPTIMIZED INTERPOLATION VERSIONS. (*) 6S160 WAS SOLVED WITHOUT TIME LIMIT, USING LAZY ABSTRACTION (STANDARD INTERPOLATION WENT OUT OF MEMORY). THE # of Solver COLUMN REPORTS HOW MANY MODEL CHECKERS SOLVED THE PROBLEM, OUT OF 21 (17) IN HWMCC'12 (HWMCC'13).

refinement and guidance through state sets. We experimentally observed that the proposed optimizations have improved both performance and scalability of our existing UMC approaches. Albeit we need to put some extra effort in a better engineering and overall integration of the proposed techniques, as well as more experimental work, we deem that current experimental data clearly witness the improvements attained.

REFERENCES

[1] W. Craig, "Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory," *The Journal of Symbolic Logic*, vol. 22, no. 3, pp. 269–285, 1957.

[2] R. C. Lyndon, "An Interpolation Theorem in the Predicate Calculus," *Pacific Journal of Mathematics*, pp. 155–164, 1959.

[3] K. L. McMillan, "Interpolation and SAT-Based Model Checking," in *Proc. Computer Aided Verification*, ser. LNCS, vol. 2725. Boulder, CO, USA: Springer, 2003, pp. 1–13.

[4] A. R. Bradley, "Sat-based model checking without unrolling," in *VMCAI*, Austin, Texas, Jan. 2011, pp. 70–87.

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

[6] K. L. McMillan and R. Jhala, "Interpolation and SAT-Based Model Checking," in *Proc. Computer Aided Verification*, ser. LNCS, vol. 3725. Edinburgh, Scotland, UK: Springer, 2005, pp. 39–51.

[7] J. Marques-Silva, "Improvements to the implementation of Interpolant–Based Model Checking," in *Proc. Correct Hardware Design and Verification Methods*, ser. LNCS, vol. 3725. Edinburgh, Scotland, UK: Springer, 2005, pp. 367–370.

[8] V. D'Silva, M. Purandare, and D. Kroening, "Approximation Refinement for Interpolation-Based Model Checking," in *Verification, Model Checking and Abstract Interpretation*, ser. Lecture Notes in Computer Science, vol. 4905. Springer, 2008, pp. 68–82.

[9] G. Cabodi, M. Murciano, S. Nocco, and S. Quer, "Boosting Interpolation with Dynamic Localized Abstraction and Redundancy Removal," *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 1, pp. 309–340, Jan. 2008.

[10] G. Cabodi, P. Camurati, and M. Murciano, "Automated Abstraction by Incremental Refinement in Interpolant-based Model Checking," in *Proc. Int'l Conf. on Computer-Aided Design*. San Jose, California: ACM Press, Nov. 2008, pp. 129–136.

[11] B. Li and F. Somenzi, "Efficient Abstraction Refinement in Interpolation-Based Unbounded Model Checking," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 3920, 2006, pp. 227–241.

[12] G. Cabodi, C. Loiacono, and D. Vendraminetto, "Optimization techniques for Craig Interpolant compaction in Unbounded Model Checking," in *Proc. Design Automation & Test in Europe Conf.* Grenoble, France: IEEE Computer Society, Mar. 2013, pp. 1417–1422.

[13] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *CAV*, 2000, pp. 154–169.

[14] A. Gupta, M. Ganai, Z. Yang, and P. Ashar, "Iterative Abstraction using SAT-based BMC with Proof Analysis," in *Proc. Int'l Conf. on Computer-Aided Design*, San Jose, California, Nov. 2003, pp. 416–423.

[15] G. Cabodi, S. Nocco, and S. Quer, "Mixing Forward and Backward Traversals in Guided-Prioritized BDD-Based Verification," in *Proc. Computer Aided Verification*, ser. LNCS, E. Brinksma and K. G. Larsen, Eds., vol. 2102. Copenhagen, Denmark: Springer-Verlag, Jul. 2002, pp. 471–484.

[16] Y. Vizel and O. Grumberg, "Interpolation-Sequence based Model Checking," in *Proc. Formal Methods in Computer-Aided Design*, ser. LNCS, vol. 2517. Austin, Texas, USA: Springer, Nov. 2009, pp. 1–8.

[17] G. Cabodi, S. Nocco, and S. Quer, "Interpolation Sequences Revisited," in *Proc. Design Automation & Test in Europe Conf.* Grenoble, France: IEEE Computer Society, Mar. 2011, pp. 316–322.

[18] Y. Vizel, O. Grumberg, and S. Shoham, "Intertwined Forward-Backward Reachability Analysis Using Interpolants," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 7795. Rome, Italy: Springer, Mar. 2013, pp. 308–323.

[19] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proc. 38th Design Automation Conf.* Las Vegas, Nevada: IEEE Computer Society, Jun. 2001.

[20] N. Eén and N. Sörensson, "The Minisat SAT Solver, http://minisat.se," Apr. 2009.

[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 Automation Conf.* New Orleans, Louisiana: IEEE Computer Society, Jun. 1999, pp. 317–320.

[22] O. Coudert, C. Berthet, and J. C. Madre, "Verification of Sequential Machines Based on Symbolic Execution," in *Lecture Notes in Computer Science 407*, Berlin, Germany, 1989, pp. 365–373.

[23] Y. Vizel and S. S. O. Grumberg, "Lazy Abstraction and SAT-Based Reachability in Hardware Model Checking," in *Proc. Formal Methods in Computer-Aided Design*. Cambridge, UK: IEEE, Oct. 2012, pp. 173–181.

[24] G. Cabodi, S. Nocco, and S. Quer, "Benchmarking a model checker for algorithmic improvements and tuning for performance," *Formal Methods in System Design*, vol. 39, no. 2, pp. 205–227, 2011.