Stepping Forward with Interpolants in Unbounded Model Checking

Gianpiero Cabodi Marco Murciano Sergio Nocco Stefano Quer

Politecnico di Torino
Dip. di Automatica e Informatica
Turin, ITALY

ABSTRACT

This paper addresses SAT-based Unbounded Model Checking based on Craig Interpolants. This recently introduced methodology is often able to outperform BDDs and other SAT-based techniques on large verification instances. Based on refutation proofs generated by SAT solvers, interpolants provide compact circuit representations of state sets, and abstract away several details non relevant for proofs. We propose three main contributions, aimed at controlling interpolant size and traversal depth. First of all, we introduce interpolant-based dynamic abstraction to reduce the support of the computed interpolant. Second, we propose new advances in interpolant compaction by redundancy removal. Both techniques rely on an effective application of the incremental SAT paradigm. Finally, we also introduce interpolant computation exploiting circuit quantification, instead of SAT refutation proofs. Experimental results are specifically oriented to prove properties, rather than disproving them (bug hunting). They show how the methodology is able to extend the applicability of interpolant based Model Checking to larger and deeper verification instances.

1. INTRODUCTION

SAT-based Bounded Model Checking (BMC) has been shown to be more robust and scalable than symbolic model checking methods based on Binary Decision Diagrams (BDDs). Unlike BDD-based methods, BMC focuses on finding bugs of bounded length, successively increasing the bound, to search for longer traces. Given a design and a correctness property, it generates a Boolean formula by unrolling the design for k time frames so that the formula is satisfiable if and only if there is a counter-example of length k. Although BMC can find bugs in larger designs than BDD-based methods, verification is not complete, as the correctness of a property is guaranteed only for the given bound.

To extend the methodology to full verification, a completeness check was proposed by Sheeran et al. [1], providing a proof of correctness for safety properties based on the longest loop-free path between states. Unfortunately, the longest loop-free path can be exponentially longer than the diameter of the reachable state space.

To overcome this problem, McMillan [2] adopted quantifier elimination, through the enumeration of SAT solutions (*all-solutions SAT*), achieved by the introduction of the so-called "blocking clauses". For each new solution, a blocking clause is added to the original problem database. This prevents the SAT-solver to run twice into the same solution.

Ganai et al. [3] replaced blocking clauses by cofactorized circuits, i.e., they adopted a circuit graph representation to capture a larger set of states for each SAT solution.

McMillan [4] recently proposed a novel and quite promising Model Checking approach, based on Craig interpolants. Modern SAT solvers are able to generate proofs of unsatisfiability (refutation proofs) which can be used to produce over-approximations of the reachable state space of the system. Given the above possibility, the basic idea is to exploit refutation proofs of (unsatisfied) BMC checks, in order to compute over-approximate state sets. The approach can be viewed as an iterative refinement of proof based abstractions to narrow down a proof to relevant facts. Its convergence is bound by the diameter of the state graph, and experimental results on large circuits showed impressive results in several cases.

On the same path, Silva [5] proposed some effective compaction optimizations for interpolant based traversals.

Our initial experiences with interpolants in Model Checking showed that they can be very effective whenever symbolic traversals are able to converge at low depths (number of reachability iterations), and interpolant sizes stay within tractable ranges. Unfortunately, this is not always the case. We thus propose three main contributions, under the general idea of compacting interpolant sizes and reducing traversal depths:

- A dynamic abstraction procedure, based on identifying optimal subsets of state variables, to be safely abstracted before each reachability step. The proposed abstraction generally reduces the amount of iterations, thus anticipating the convergence of the process.
- An interpolant circuit optimization technique, based on redundancy removal under Observability and External Don't Care conditions. The strategy we propose is applied on top of other optimization techniques, see [6], and effectively exploits the benefits of incremental SAT.
- Interpolant computation based on circuit based quantification, rather than on refutation proofs. We intro-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICCAD'06, November 5-9, 2006, San Jose, CA
Copyright 2006 ACM 1-59593-389-1/06/0011 ...$5.00.

duce this technique (combined with dynamic abstraction) in backward reachability, where circuit based quantification is more effective.

The main intuition under dynamic abstraction is that variable quantification projects traversals over subspaces (of non quantified variables). Over-approximations are generally stabler than SAT based interpolants, and produce shorter traversals. Our techniques are implemented by successfully exploiting incremental SAT, i.e., by properly formulate a set of SAT problems which are able to share large portions of their clause database. The main idea of incremental SAT is to forbid clause removal (and just allow the addition of new clauses) across different SAT solver runs.

Our experimental results concentrate on proving correct properties, and showing that the proposed method improves the original one by making it faster, more robust and scalable. We show experiments where we are able to complete difficult instances, not achievable without our techniques.

2. BACKGROUND

The sequential systems we address are usually modeled either as Finite State Machines (FSMs), or as the more general Kripke Structures.

A Kripke structure is defined as a 4–tuple M = (S, I, T, L), where: S is the set of states, I $\subseteq$ S is the set of initial states, T $\subseteq$ S $\times$ S is the transition relation between the states, L: S $\rightarrow 2^A$ is the function that labels each state with a set of atomic propositions. We will use x for present state variables, the primed notation (x') for next state variables, and T^k to indicate a combinational unrolling of T of depth k.

An invariant property P over a sequential model is checked by attempting to prove (or disprove) the reachability of its complement F (the target state set, F = $\neg$P) from I.

Given two inconsistent formulas A and B (A $\wedge$ B = 0), the interpolant C is a formula such that: (1) it is implied by A, (2) it is inconsistent with B, and (3) it is expressed over the common variables of A and B.

Starting from a refutation proof RP of A $\wedge$ B (easily obtained from a SAT solver), the interpolant C = Itp (A, B) is an AND/OR circuit that can be computed from RP. Albeit the computation can be performed in linear time with respect to the size of RP, the size of RP itself can be exponential compared to A and B.

An over-approximate image Img^+ is such that, for all state sets S, Img (T, S) implies Img^+ (T, S). A k–adequate over-approximate image Img^+_{Adq} (T, S, F, k) is an Img^+ (T, S) that does not intersect any state on paths of length k to F. Img^+_{Adq} (T, S, F, k) is undefined iff

$$\text{Img (T, S)} \wedge T^k \wedge F \neq 0.$$

A possible way of computing Img^+_{Adq} is interpolation:

$$\text{Img}^+_{Adq}\text{ (T, S, F, k)} = \text{Itp (S} \wedge \text{T, T}^k \wedge \text{F)}$$

An image is called adequate if it is k–adequate for any k, i.e., no path of any length can lead from a state within the image to states in 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.

Working with increasing values of k and with the k–adequate (interpolant-based) image operator, McMillan [4] introduces a finite, fully SAT, algorithm for unbounded model checking. The algorithm is sketched in Figure 1.

```
InterpolantMC (I, T, F)
    k = 0
    repeat
        res = FiniteRun (I, T, F, k)
        k = k + 1
    until (res ≠ undecided)

FiniteRun (I, T, F, k)
    if (SAT (I ∧ T^k ∧ F))
        return (reachable)
    R = I
    while (true)
        to = Img+_Adq (T, R, F, k)
        if (to = undefined)
            return (undecided)
        if (to ⊆ R)
            return (unreachable)
        R = R ∨ to
```

Figure 1: Interpolant based Verification.

For each value of k, starting from 1, function FiniteRun is called. It first solves a BMC problem of bound k, to tentatively prove F reachable. If this is not the case, it performs an approximate traversal until a fixed point (property proved), or F is reached by over-approximate state sets (interpolant undefined).

Since k increases at every iteration, if I and F are mutually unreachable, eventually k $\geq$ d. In such a case, Img^+_{Adq} is adequate, hence we terminate. For smaller values of k, a k–adequate set can produce a non k–adequate image.

According to [5], in order to avoid a quadratic number of image computations, the depth of the last FiniteRun execution can be used to increment k.

3. DYNAMIC ABSTRACTION

Let us start from the observation that interpolation basically provides an over–approximate image, k–adequate with respect to a given target set F, or, in other words, guaranteed not to reach F through paths of length k.

Two nice effects are related to the use of Craig interpolants in verification: (1) they achieve over-approximation and variable existential quantification (i.e., image computation) at the same time, and (2) the over-approximation process is based on the property under check, so that the abstraction concentrates on relevant facts.

At the opposite side of the spectrum, problems may arise due to:

- **Deep traversals.** Let d^+ be the sequential depth of the over-approximate FiniteRun procedure, and d the diameter of the state transition graph. In general, $d^+ < d$, i.e., over-approximate "bypasses" long state transition paths and converges faster than exact reachability. Nevertheless, $d^+ > d$ is also possible, whenever interpolants trigger long walks over chains of "unreachable" but "adequate" states.

- **Interpolant size.** The size of interpolants is generally difficult to predict, as interpolant circuits come from refutation proofs, and their initial size is strongly related to the SAT solving process. It has been observed [4, 5] that interpolants are highly redundant. Furthermore, different refutation graphs are possible

for a given proof, producing different interpolant circuits. McMillan [7] explores a full range of refutation graph transformations, and the resulting interpolant circuits, basically looking for tighter over-approximations.

In order to face the above problems, we exploit abstraction, as a well known (and successful) way of analyzing over-approximate behaviors, by removing some details and/or mutual dependencies, such that the proof is still achievable.

Although interpolation itself can be viewed as an abstraction technique, we specifically consider over-approximation by state variable (existential) abstraction. Our aim is to find a subset α of the x state variables, whose abstraction still guarantees k–adequacy. We consider abstraction itself as an interpolation procedure, able to produce k–adequate over-approximations. We call our abstraction process dynamic, as we restart abstraction before each image computation, instead of keeping a given abstraction over an entire traversal (as in standard abstraction–refinement methods [8, 9]).

Given $\alpha \subseteq x$, we compute the abstract transition relation

$$\mathrm{T}_\alpha = \exists_{\alpha'}\ \mathrm{T}$$

by existentially quantifying out the α' variables in the next state space.

The abstraction is k–adequate iff

$$\textsc{Img}\ (\mathrm{T}, \mathrm{S}) \wedge \mathrm{T}^k \wedge \mathrm{F} = 0 \quad \Rightarrow \quad \textsc{Img}\ (\mathrm{T}_\alpha, \mathrm{S}) \wedge \mathrm{T}^k \wedge \mathrm{F} = 0$$

A k–adequate abstraction can be used within the FiniteRun procedure, either as a preprocessing step of interpolant based image, or as the only over-approximate operator, followed by exact image (using the abstract T_α).

The main advantages we expect from such an abstraction are: (1) a minimal set of support variables, and hopefully a minimal circuit size, of the image state set, and (2) shorter reachability depths d^+.

Intuitively, we start from the observation that minimal support is often related to lower circuit size. SAT solvers are unlikely to achieve minimum support of the refutation proof by themselves, as the heuristics driving them are based on different targets. Although we generally expect that refutation proofs do not include variables that are not relevant for a proof, we don't expect minimal support interpolants.

A specifically targeted abstraction, paying some overhead, is more likely to get to such a minimal support interpolant. Moreover, shorter traversal depths come from abstraction by variable quantification, that projects the state transition graph over a subspace. This tends to produce more compact graphs and shorter state transition paths.

3.1 Selecting the Abstraction

Let us consider now the process of finding the α set of variables. Minimum–Cost Unsatisfiability proof should be considered, in order to address this particular issue. Unfortunately (to the best of our knowledge), just Minimum–Cost Satisfiability Tools (MinCostSAT) [10] have been studied, not the former ones.

This is an optimization problem, that we can face by trading off performance for optimality. Iterating through all candidate subsets, in order to find k–adequate abstractions, and select the optimal one, is clearly too expensive. We chose a sub–optimal greedy approach that incrementally builds the α set by looping through all next state variables. Figure 2 shows the procedure. The generic x'_i variable is tentatively added to the abstraction set α (initially empty), and validated for k–adequacy. The result is sensitive to the chosen variable order. Given a state set S and a transition relation T, k–adequacy of their image w.r.t. F is tested by a SAT check:

```
DynAbstr (S, T, F, k)
   // k–adequacy check for exact image
   if (SAT(S ∧ T∧ T^k ∧ F) ≠ 0)
      return (undecided)
   α = {}
   foreach next state var x'_i
      α = α + x'_i
      // k–adequacy check for abstract image with T_α
      if (SAT(S ∧ T_α ∧ T^k ∧ F) ≠ 0)
         // Not adequate: undo abstraction
         α = α - x'_i
   return (α)
```

Figure 2: Dynamic Abstraction.

$$\textsc{Img}\ (\mathrm{T}, \mathrm{S}) \wedge \mathrm{T}^k \wedge \mathrm{F} \neq 0 \quad \Leftrightarrow \quad \mathrm{SAT}\ (\mathrm{S} \wedge \mathrm{T} \wedge \mathrm{T}^k \wedge \mathrm{F}) \neq 0$$

This is done by a preliminary test with T (to filter out undefined interpolants), then repeated for all candidate abstract transition relations T_α.

The process requires a linear (in the number of state variables) number of SAT calls, which can still be too expensive. To further reduce the cost of the iterated k–adequacy SAT checks, we adopted *incremental SAT*, a well known approach for multiple, though related, SAT calls.

Let x' be the set of (common) next state variables shared by T_α and T^k. In order to relationally express different T_α (with different α sets), we use *wire cutting* instead of explicit existential quantification. We generate two sets of fresh variables $\hat{x}$ and γ. Variable $\hat{x}_i$ replaces x'_i in T_α, so that now T_α and T^k do not share state variables any more. Conditional variable sharing is captured by an additional term

$$\bigwedge_i (\gamma_i \Rightarrow (\hat{x}_i \Leftrightarrow x'_i))$$

where $\gamma_i = 1$ means wire connection between $\hat{x}_i$ and x_i, $\gamma_i = 0$ means no wire connection (no relation) between them. In other words, $\gamma_i = 0$ is equivalent to the existential quantification of x'_i in T_α (as $\hat{x}_i$ is an unconstrained free variable). We now express T_α as follows:

$$T_\alpha(x'/\hat{x}) \wedge \bigwedge_i (\gamma_i \Rightarrow (\hat{x}_i \Leftrightarrow x'_i))$$

We now load all terms (including T_α in relational form) once and for all into the SAT solver (in terms of CNF clauses). The generic SAT call, to test k–adequacy of a given α set, is done incrementally:

$$\mathrm{Assume}_\alpha = \bigwedge_i (\gamma_i = \neg(x_i \in \alpha))$$
$$\mathrm{SAT}_{Assume_\alpha}\ (\mathrm{A} \wedge \mathrm{B})$$

Each SAT call is performed exploiting incremental learning and assuming proper values for all γ variables. Any γ_i is set to 0 if x_i belongs to the α set, 1 otherwise. A further improvement we adopted (omitted here for sake of simplicity) is the introduction of a time bound on SAT calls. This improves scalability, but leaves undecided k–adequacy tests.

The undecided variables are further refined by a second loop, starting from the assumption that all undecided variables are abstracted. This set is then refined until the abstraction is proved k–adequate.

4. CIRCUIT COMPACTION

Circuit compaction is another way to attack the size of interpolant representations, as interpolants are potentially highly redundant.

Among the available possibilities, we concentrated on redundancy removal, whose basic steps consist in identifying circuit nodes replaceable by constant nodes. Although testing techniques have been successfully applied in this framework [11], as redundancies are related to untestable stuck–at faults, we resorted to SAT-based algorithms [12, 13]. We expected major improvements from the recent developments in the SAT technology, the incremental approach, and an aggressive use of don't care based simplifications.

First of all, we looked at incremental SAT, as redundancy removal is essentially a set of SAT checks over several circuit nodes, to prove/disprove candidate substitutions with constant nodes. Given a circuit f (identified by the set of its outputs), and a subset N of its nodes (to be checked for redundancy) we build up a product machine PM for each redundancy test, comparing the (outputs of the) original f with the new one (with constant injected). More formally, let $n_i \in N$ be the node to be checked for 0–redundancy, we do the following check:

$$SAT(f \neq f(n_i/0)) = 0 \quad \Rightarrow \quad n_i \; is \; 0-redundant$$

Where the $f(n_i/0)$ expression means that the n_i node is assigned the 0 constant value, and the dependency from its fanin circuit is removed. Dually for 1–redundancy. In order to guarantee incrementality for the SAT calls over the set of N nodes, we basically need to express every constant injection in terms of just variable decisions. So for each $n_i \in N$, we generate two new variables: c_i, selecting the proper constant value (0 or 1), and γ_i, selecting between non redundancy ($\gamma_i = 1$, n_i connected to its original fanin) and redundant behavior ($\gamma_i = 0$, n_i fed by c_i). So all n_i nodes in N are replaced by ITE (γ_i, n_i, c_i). Now each redundancy check over N can be achieved incrementally, through proper assumptions on the γ and c variables.

We experimentally found that it was not worthwhile to work with too large N sets, e.g., the whole set of f nodes. Not only different redundancies are related/implied each–other (so it might be unnecessary to prove all of them), but a given circuit can be recomputed and highly compacted, after a subset of ITE redundancies is known. So we decided to work with "clusters" of nodes for the N set. Nodes are added to the N cluster following their topological depth, and a cluster size threshold. Limiting the size of N allows us to control the overall amount of variables for the SAT problem, and to rebuild f after redundancies over N are found, yet exploiting incremental SAT for all checks on a given N.

Our redundancy removal procedure is shown in Figure 3. Let f be the function (represented by a combinational network) we want to optimize, and Care be an External Care condition. The RedRemoval procedure loops through successive redundancy removal iterations, each one considering a wider set of candidate nodes. We sort nodes by topological level, and we first consider nodes near the inputs. The

```
RedRemoval (f, Care)
  Th = minTh
  repeat
    f̂ = f
    N = SelectCluster (f̂, Th)
    for each node n_i ∈ N
      replace n_i with ITE (γ_i, c_i, c_i)
      // Initially no redundancy assumed
      Assume_γi = 1
    PM = (f̂ ≠ f) ∧ Care
    for each node n_i ∈ N
      Assume_γi = 0 // Try redundancy
      Assume_ci = 0 // Constant 0
      if (¬SAT_Assume (PM))
        // Redundancy found
        Set n_i = 0 in f
      else
        Assume_ci = 1 // Constant 1
        if (¬SAT_Assume (PM))
          // Redundancy found
          Set n_i = 1 in f
        else
          Assume_γi = 1 // Not redundant
    RecomputeAIG (f)
    Th = Th · 2
  until ((N > maxTh) ∨ (No Redundancy Found))
```

Figure 3: Redundancy Removal under external care conditions.

procedure loops until no redundancy is found, or the upper bound (maxTh) on tested nodes is reached.

A relevant contribution for stronger optimizations is provided by a careful and effective use of don't cares, that quite often allow finding far more redundancies than with the original circuit alone.

We exploit External Don't Cares (EDC) optimization with the idea that already reached states can work as EDC conditions for further reachability steps ($Care = \neg EDC = \neg Reached$). The above simplification was key for advances in BDD–based reachability, where the "restrict" cofactor was used to minimize the size of reachable state sets feeding image operators.

We then implicitly take into consideration both Input Don't Cares (IDC) and Output (or Observability) Don't Cares (ODC), within our product machine model. Historically, most practical redundancy removal approaches are quite often limited to Input Don't Cares (IDC), as redundancy removal under IDCs preserves the circuit function: a node is replaced with a constant node whenever it is constant under all allowed inputs. Practically speaking, the product machine includes a comparison output exactly on the redundant node n_i.

ODC conditions take into account more aggressive node transformations: they allow different node values, for a given node, from the original and the transformed circuit, provided that they are not observable on the checked outputs. The main problem with this kind of redundancies is that two ODC based redundancies are not mutually unrelated (any accepted redundancy modifies the circuit for next redundancy checks). This makes simulation based preprocessing less effective, and efficient SAT processing crucial. Furthermore, the overall result depends on the ordering of the re-

dundancy checks. Incremental SAT is key to effectively handle ODC conditions, as it allows different checks to exploit common ODC conditions (and the related conflict clauses). Concerning the order of checks, we presently rely on topological heuristics, though we expect that further improvements are possible.

5. FORWARD/BACKWARD VERIFICATION

It is well known that forward and backward reachability can be selectively applied to get better results, as each one can beat the other depending on the model under check.

```
FiniteRun_Fwd (I, T, F, k)
    if (SAT (I ∧ T^k ∧ F))
        return (reachable)
    R = From = I
    while (true)
        α = DynAbstr (From, T, F, k))
        if (SAT (α) = undecided)
            return (undecided)
        to = Img^+_Adq (T_α, From, F, k)
        if (to ⊆ R)
            return (unreachable)
        From = RedRemoval (to, ¬R)
        R = R ∨ to
```

Figure 4: Forward traversal.

The advantage of either approach can rely on the depth of forward/backward reachability, and the ease to express the corresponding state sets. We implemented both forward and backward reachability in our interpolant based framework. The two approaches are quite similar, and both correspond to the main traversal scheme represented in Figure 1.

Forward verification is shown in Figure 5. Whenever the transition relation comes from a circuit, it can be represented as:

$$T = \bigwedge_i (x'_i \Leftrightarrow \delta_i)$$

This means that existential quantification of next state variables is straightforward: It is done by removing the corresponding component in T. We exploit this fact while generating the abstract transition relation T_α, by simple component elimination from T.

Backward reachability is similar, as the roles of the initial (I) and target (F) states are swapped, and the pre–image operator uses the transition relation in the reverse direction. Anyway, it is possible to exploit different optimizations (see Figure 5).

We cannot generate T_α by component removal, as true quantification (before or within pre-image computation) is required. Nevertheless, we can apply state variable quantification by composition within the pre–image computation. This can drastically reduce the amount of variables to be quantified in pre–image, and allows us to (at least partially) adopt exact pre–image (using) T_α. The LazyPreImg procedure performs exact circuit quantification, for primary inputs and α variables, whenever convenient (otherwise the variable is kept) with a method inspired to [14] and [15]. If we are able to fully complete exact pre-image with LazyPreImg, then we avoid SAT based over–approximation by interpolant, thus providing a tighter pre–image. Otherwise we interpolate.

```
FiniteRun_Bwd (I, T, F, k)
    if (SAT (I ∧ T^k ∧ F))
        return (reachable)
    R = From = F
    while (true)
        α = DynAbstr (From, T, I, k)
        if (α = undecided)
            return (undecided)
        to_0 = LazyPreImg (T_α, From)
        if (more variables to quantify)
            to = Itp (to_0, I ∧ T^k)
        if (to ⊆ R)
            return (unreachable)
        From = RedRemoval (to, ¬R)
        R = R ∨ to
```

Figure 5: Backward traversal.

The peculiar aspect of the above procedure is to often achieve full interpolation by dynamic abstraction and circuit quantification, without resorting to SAT–based Craig interpolants. Even though less general, the procedure showed very good results in some experimental cases, where it was able to outperform forward interpolation.

6. EXPERIMENTAL RESULTS

We compare interpolant based Model Checking, with and without the optimizations described in this paper. Our procedures are implemented on top of the Minisat [16] SAT tool. We also use our own implementation of an AIG library, and we exploit the ABC tool [6] for different kinds of logic synthesis optimizations.

Our experiments ran on a Dual Core Pentium IV 3 GHz Workstation with 3 GByte of main memory, running Debian Linux. We performed extensive experiments on selected benchmarks, with both forward and backward interpolation, by specifically addressing proofs of correctness.

We present results on:

- Some standard benchmarks belonging to the VIS distribution.
- The Sun PicoJava II microprocessor as presented in [4]. It includes 20 true safety properties with a number of state variables (after cone of influence reduction) ranging from about 50 to 350.
- The IBM Formal Verification Benchmark Library [17]. This library includes 75 circuits, each one with one property, with a size ranging from 95 to 917 memory elements.
- Some industrial circuits coming from STMicroelectronics.

Table 1 includes detailed data comparing the running time of the different strategies implemented. Among all the previously indicated problems, we only consider the ones which needed more than 20 second of CPU time, and that we were able to complete, with at least one strategy, in less than 1800 seconds. Each line reports data on one single property. More lines labeled with the same circuit name indicate that more properties are verified on the same circuit. The table reports the number of memory elements for each circuit (after COI extraction), and the verification time for the

Model	# FF	Fwd	Fwd+DA	Fwd+DA+RR	Bwd+DA+RR	Fwd+DA+RR-IS
$Ns2_1$	67	–	–	–	101	–
$Ns2_2$	67	639	–	–	48	–
$Ns2_3$	67	726	42	111	130	–
$PicoJava_5$	88	83	42	22	20	–
Blackjack	103	976	–	265	793	1687
31_2_batch_1	122	1249	57	78	–	–
$Soap_1$	140	–	–	–	268	–
$Soap_2$	140	–	–	–	205	–
$Soap_3$	140	–	–	1535	–	–
$Industrial_1$	186	64	50	25	55	586
$Industrial_2$	202	–	–	280	321	1567
$PicoJava_{16}$	290	61	85	63	–	132
Feistel	296	722	199	311	748	–
$PicoJava_6$	322	118	1122	81	–	–
$PicoJava_{15}$	364	44	50	49	–	55

Table 1: CPU Time (in seconds) for the different strategies. – means time out after 1800 seconds.

different interpolation strategies. The bottom-line strategy is represented by Fwd, where we indicate the time for the original interpolation method [4] re-implemented in our tool. +DA presents results by adopting dynamic abstraction, and +RR indicates the presence of redundancy removal. Finally, Bwd shows the strategy in the backward direction, and -IS the results without incremental SAT. Notice that in all the strategies we adopted synthesis optimization techniques as delivered by the ABC tool [6].

Notice that the last column clearly witnesses the relevant role of incremental SAT. Forward reachability is generally better, though in a few cases the backward approach is the key to complete otherwise not achievable problems. Redundancy removal has a larger impact on backward reachability. In the forward direction it represents, at least in some cases, just an overhead. We are often able to attain low state set AIG sizes in problems where they grow much larger with the original procedure. After all, we were able to complete 5 problems that were not achievable with the original approach.

Figure 6 and 7 show data on the 20 properties of PicoJava II. Notice that in [4] all properties could not be verified by standard symbolic model checking, within a limit of 1800 seconds and that one property was not achievable by any technique. We were able to complete *all* 20 properties in a total time of 528 seconds, and a maximum time (for property 6) of 118 seconds.

Figure 6 reports some data concerning the number of reachability iterations (within the FiniteRun procedure) for the Fwd+DA+RR method against the Fwd strategy. The figure plots the number of FiniteRun calls (external interpolation loop) and the number of FiniteRun inner iterations (inner interpolation loop). For the last one, we report both the maximum value (maximum number of inner iterations on all main iterations) and the total value (total number of inner iterations performed on all main iterations).

Figure 7 reports the maximum size of the support of the to set, for the same two strategies compared in Figure 6.

A direct comparison between the original and the proposed method shows that the latter one is able to reduce both the amount of traversal iterations and the support of the computed state sets, supporting the original claims of the paper.

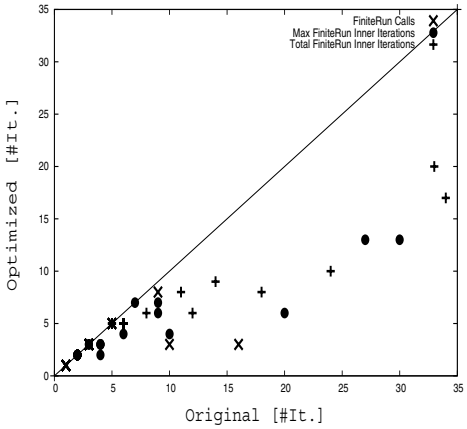

Figure 6: Number of FiniteRun calls (external interpolation loop) and number of FiniteRun inner iterations (maximum and total values): Fwd+DA+RR method against Fwd.

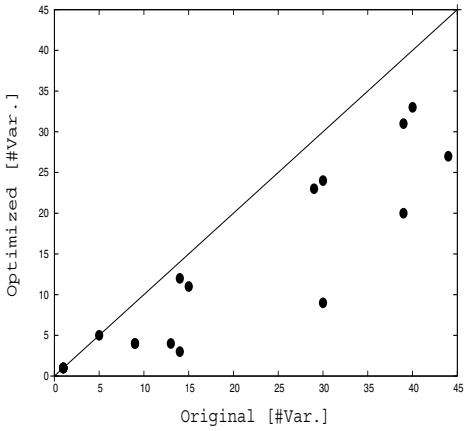

Figure 7: Maximum support of the to set along the whole traversal process: Fwd+DA+RR method against Fwd.

7. CONCLUSIONS

This paper shows how Craig interpolants, derived from SAT proofs, can be further optimized by (1) extending the idea of interpolation to variable abstraction, (2) by effectively exploiting redundancy removal under External and Observability Don't Care conditions, and (3) circuit based quantification adopted in the backward direction.

Our optimizations heavily rely on an efficient use of incremental SAT, that allows grouping several related problems with common learning.

To conclude, the interpolation approach still shows its main limits with sequentially deep verification instances correlated with large interpolant circuits. Solving the above problems seem to be the main challenge for future works in interpolant based verification.

Another interesting options is to better integrate and combine interpolant verification with other state-of-the-art methods such as abstraction-refinements, inductive verification and automated generation of lemmas.

8. ACKNOWLEDGMENTS

The authors would like to thank K. L. McMillan for the source descriptions of the Sun PicoJava II microprocessor, and the VTT group of the STMicroelectronics, Agrate, Italy, for their industrial benchmarks.

9. REFERENCES

[1] M. Sheeran, S. Singh, and G. Stålmarck. Checking Safety Properties Using Induction and SAT Solver. In W. A. Hunt and S. D. Johnson, editors, *Proc. Formal Methods in Computer-Aided Design*, volume 1954 of *LNCS*, pages 108–125. Springer-Verlag, November 2000.

[2] K. L. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proc. Computer Aided Verification*, volume 2404 of *LNCS*, pages 250–264, Copenhagen, Denmark, 2002.

[3] M. K. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based Unbounded Symbolic Model Checking Using Circuit Cofactoring. In *Proc. Int'l Conf. on Computer-Aided Design*, San Jose, California, November 2004.

[4] K. L. McMillan. Interpolation and SAT-Based Model Checking. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Proc. Computer Aided Verification*, volume 2725 of *LNCS*, pages 1–13, Boulder, CO, USA, 2003.

[5] J. Marque-Silva. Improvements to the implementation of Interpolant–Based Model Checking. In D. Borrione and W. Paul, editors, *Proc. Computer Aided Verification*, volume 3725 of *LNCS*, pages 367–370, Edimburgh, Scotlan, UK, 2005.

[6] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, http://www.eecs.berkeley.edu/∼alanmi/abc/.

[7] K. L. McMillan and R. Jhala. Interpolation and SAT-Based Model Checking. In *Proc. Computer Aided Verification*, Edimburgh, Scotlan, UK, 2005.

[8] B. Lin, C. Wang, and F. Somenzi. A Satisfiability-Based Approach to Abstraction Refinement in Model Checking. In *BMC'03: First International Workshop on Bounded Model Checking*, Boulder, Colorado, July 2003.

[9] P. Chauhan, E. Clarke, J. Kukula, S. Sapra, H. Veith, and D. Wang. Automated Abstraction Refinement for Model Checking Large State Spaces Using SAT Based Conflict Analysis. In M. D. Aagaard and J. W. O'Leary, editors, *Proc. Formal Methods in Computer-Aided Design*, volume 2517 of *LNCS*, pages 35–51, November 2002.

[10] V. Manquinho and J. Marques-Silva. Search Pruning Techniques in SAT–based Branch-and-Bound Algorithms for the Binate Covering Problem. *IEEE Trans. on Computer-Aided Design*, 21:505–516, 2002.

[11] M. Berkelaar and K. van Eijk. Efficient and Effective Redundancy Removal for Million-Gate Circuits. In *Proc. Design Automation & Test in Europe Conf.*, pages 1088–1088, Paris, France, March 2002.

[12] A. Mishchenko and R. K. Brayton. Scalable Logic Synthesis using a Simple Circuit Structure. In *Proc. Int'l Workshop on Logic Synthesis*, Lake Tahoe, California, May 2006.

[13] A. Mishchenko and R. K. Brayton. Improvements to Combinational Equivalence Checking. In *Proc. Int'l Conf. on Computer-Aided Design*, San Jose, California, November 2006.

[14] P. A. Abdulla, P. Bjesse, and N. Een. Symbolic Reachability Analysis based on SAT-Solvers. In *TACAS 2000 - Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 1785, Springer Verlag*, pages 411–425, Upsala University, Prover Technology, Chalmers University, Sweden, April 2000.

[15] G. Cabodi, S. Nocco, and S. Quer. Circuit Based Quantification: Back to State Set Manipulation within Unbounded Model Checking. In *Proc. Design Automation & Test in Europe Conf.*, Munich, Germany, March 2005.

[16] N. Eén and N. Sörensson. Minisat SAT Solver, http://www.cs.chalmers.se/Cs/Research/Formal-Methods/MiniSat/.

[17] IBM Formal Verification Benchmark Library. http://www.haifa.il.ibm.com/projects/verification/-rb_homepage/fvbenchmarks.html.