Tightening BDD-based Approximate Reachability with SAT-based Clause Generalization*

G. Cabodi and P. Pasini and S. Quer and D. Vendraminetto
Dipartimento di Automatica ed Informatica
Politecnico di Torino - Torino, Italy

Abstract—**In the framework of symbolic model checking, BDD-based approximate reachability is potentially much more scalable than its exact counterpart. However, its practical applicability is highly limited by its static approach to abstraction, and the intrinsic difficulty to find an acceptable trade-off between accuracy and memory/time complexity.**

In this paper, we apply SAT-based cube generalization, a core step of the IC3 model checking algorithm, to BDD-based over-approximate reachability analysis. More specifically, we use cube generalization, in both its inductive and non-inductive versions, to tighten BDD-based over-approximate representations of state sets computed by Machine by Machine (MBM) and Frame by Frame (FBF) algorithms. The resulting approach benefits from the orthogonal power of BDD and CNF representations, and it improves the scalability and applicability in verification of BDD-based methods.

Experimental results confirm that this approach can provide tighter representations of reachable state sets and more powerful fully BDD-based engines, as well as potential applications of BDDs as invariants or constraints in SAT-based model checking.

I. Introduction

Binary Decision Diagrams [1] (BDDs) represented a major technology in symbolic model checking for about a decade, before the advent of SAT solvers, and SAT-based model checking methods, in the late 90s. Whereas BDD-based reachability analysis tends to become impractical if the design under verification cannot be reduced or abstracted below several hundred state variables, SAT-based techniques can scale beyond tens of thousands of state variables. Nevertheless, BDDs may dramatically outperform SAT-based techniques for selected classes of problems, and are still needed for specific verification instances [2]. As a consequence, albeit BDDs are typically considered as a second level engine in a modern Model Checking portfolio, a well-tuned BDD-based reachability engine is an essential component of a state-of-the-art verification tool.

Numerous techniques have been developed to boost the scalability of BDD-based reachability engines. Approximate reachability techniques [3], [4] obtain a higher degree of scalability than exact BDD traversal. They represent state sets as partitioned BDDs of bounded support, and work on partitioned (abstract) transition relations. Scalability is paid in terms of over-approximation and incompleteness of the approach in the stand-alone mode, as it is quite rare that an over-approximate set of reachable states proves a property. Anyhow, approximate reachability may play a key role as a slave technique to guide exact backward traversals in pure BDD-based model checking, or as a pre-processing step producing invariants for Bounded Model Checking, Craig interpolants or IC3.

Moving to SAT-based model checking, interpolant-based verification [5] and IC3 [6] are the most recent and notable achievements. Craig interpolants are usually computed by transforming a refutation proof into a Boolean circuit with the same structure. Unfortunately, since modern SAT solvers are not specifically aimed to generate compact refutation proofs, the produced interpolants are often very large even for simple designs. To solve this problem, Chockler et al. [7] use cubes generalization to evaluate interpolants without refutation proofs. The authors essentially compute an interpolant incrementally, by taking the disjunction (union) of *point interpolants*. Each point interpolant is obtained with cube generalization.

IC3 (Incremental Construction of Inductive Clauses for Indubitable Correctness) is based on incrementally refining, and extending, a sequence of over-approximated reachable states. It does not require unrollings of the transition relation, as it uses inductive reasoning to incrementally find those state sets. One of the most important steps performed by the tool is the so called *cube generalization*. Cube generalization is the process of finding a minimal sub-clause, by removing as many literals as possible from it, such that it over-approximates the set of reachable states while excluding the cube. Generalization is used in IC3 to refine clause-based representations of state sets.

Following the work by Chockler et al. [7], we tighten BDD-based approximate reachable states with sets of clauses, properly strengthened by generalization. We apply this tightening scheme to two approximate reachability algorithms, namely the *Machine by Machine* (MBM) and the *Frame by Frame* (FBF) algorithms [3], [4]. Though the overall idea, i.e., to use clauses to strengthen representations of reachable state sets, is common to IC3 and [7], we integrate the clause generation and generalization steps within the inner steps of BDD-based reachability. We thus propose an intertwined approach where SAT-generated clauses refine and strengthen the BDD-based reachability. In the case of Machine by Machine (MBM), whose top level algorithm iterates through exact traversals of component transition relations, clause-based strengthening is activated within the main loop, after one step of reachability on all component sub-models. In the Frame by Frame (FBF) over-approximate reachability, that performs partitioned image steps, generalized clauses are added at the end of each approximate image computation. We also propose a light-weight

* This work was supported in part by SRC contract 2012-TJ-2328.

strengthening step, achieved by a pure post-processing of over-approximate reachability, in both MBM or FBF approaches.

We present a set of encouraging experimental results showing how cube generalization is able to strengthen (in terms of number of reached states) over-approximate reachability state sets computed with the MBM and FBF approaches. Moreover, we demonstrate how tighter estimates can be used in a BDD-based verification framework mixing approximate forward and exact backward reachability analysis.

II. Background

A. Model, Notation, and Property Definition

The sequential systems we address are usually modeled as Finite State Machines (FSMs). Each FSM is described by a Transition Relation $\mathsf{TR}(s, y)$, which indicates its present–next state behavior, and an initial state set I. In our notation, B indicates the Boolean space. Symbols $\wedge$, $\vee$, and $\neg$ are used for Boolean conjunction (AND), disjunction (OR), and negation (NOT), respectively. The $\downarrow$ symbol denotes the generalized cofactor function. We make no distinction between the BDD representing a set of states, the characteristic function of that set, and the set itself. We thus use Boolean operators for set operations, implemented by Boolean operators on BDDs.

In our approximate BDD-based reachability pseudo-codes, *subscripts* will be used for iteration, and *superscripts* will be adopted to identify decomposed machines.

B. Approximate Reachability Analysis

Approximate Traversals [3], [4] attain scalability and approximation adopting the two following steps:

1) Performing a (static) *State Space Decomposition*, i.e., heuristically partitioning state variables in subsets. Each partition corresponds to a Boolean subspace, and to a sub-FSM of the original FSM.
2) Computing super-sets (R^+) of the reachable states (R), by performing separate traversals and/or images on sub-FSMs.

Different strategies can be used to coordinate traversal and image computations of the different sub-FSMs, and to model the interaction among them. In this paper, we will concentrate on the *Machine by Machine* (MBM) and the *Frame by Frame* (FBF) approaches. Their description will be embedded in our contributions in Section III-A and III-B, respectively.

Notice that, although over-approximate reachable state sets include more states than their exact counterparts, their BDD representations are usually much smaller, as many mutual interactions and dependencies among state variables disappear because of the approximation. Moreover, notice that albeit approximate techniques are computationally cheaper and more scalable, they provide sufficient but not necessary checks, i.e., they can prove correctness but they cannot disprove it.

C. Satisfiability

Given a propositional formula, the Boolean Satisfiability Problem (commonly abbreviated as *SAT*) consists of determining a variable assignment such that the formula evaluates to true, or establishing that no such assignment exists. In case no assignment exists, we would say that the function is *unsatisfiable*. On the contrary, if we are able to find an assignment, we would say that the Boolean formula is *satisfiable*.

SAT solvers generally operate on problems for which a Boolean function is specified in *Conjunctive Normal Form* (*CNF*). A formula in CNF form is the logical conjunction (AND) of one or more *clauses*, each of which consists of the logical disjunction (OR) of one or more *literals*. A *literal* is merely an instance of a variable or its complement. A *cube* is the negation of a clause, i.e., a conjunction of literals.

D. Generalization

Inductive generalization [8], [6] of a cube s, with respect to a set of states $\mathcal{F}_i$, is the process of finding a minimal inductive sub-clause d of $\neg s$, if one exists. The resulting sub-clause over-approximates the set of reachable states while excluding s and all states that can reach s. More in detail, given two sets of states $\mathcal{F}_{i-1}$ and $\mathcal{F}_i$, such that:

- Over-approximate states reachable from I in $i-1$ (resp. i) steps,
- Satisfy the relation $\mathcal{F}_i \supseteq \textsc{Img}(\mathsf{TR}, \mathcal{F}_{i-1})$.

the generalized clause is a clause such that:

$$\mathcal{F}_i \wedge d \supseteq \textsc{Img}(\mathsf{TR}, \mathcal{F}_{i-1} \wedge d)$$

In practice, a minimal inductive sub-clause is typically substantially smaller than the cube s from which it is extracted and it excludes states that are not necessarily related to s by TR, which is why we say that the inductive sub-clause generalizes that s is unreachable.

III. Over-approximate Reachability with SAT-based Strengthening

The driving idea, for our variants of MBM and FBF approximate reachability algorithms, is to intertwine BDD-based approximation steps with SAT-based strengthening computations. Strengthening steps, adopting a bounded number of generalized clauses, are introduced to filter-out unreachable states. Figure 1 shows the core idea of our approach. Let us

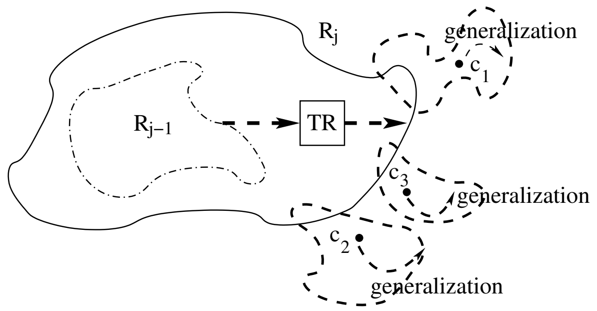

Fig. 1. Tightening the R_j set using cube extraction and generalization.

suppose that during a step of over-approximated reachability analysis the state set R_j is computed starting from R_{j-1}. Our target is to tighten R_j by selecting a limited number of cubes outside Rj, and by generalizing them. Each cube generalization is used to restrict the original R_j set. When available R_{j-1} can be used for inductive generalization.

In the rest of the section, we will describe the MBM and FBF algorithms in more details.

A. The Machine by Machine (MBM) Approach

Our *Machine by Machine* (MBM) approach, improved with SAT-based clause generalization, is reported in Figure 2.

The algorithm starts with all R^i sets initialized to the tautology $\top$. This indicates that all the states of each sub-machine are assumed to be reachable. Moreover, it gives a trivial upper bound for the computation.

After that, the procedures enters a main iteration in which both BDD-based and SAT-based contributions are computed. BDD-based contributions are evaluated by functions TRAV$^+$. This procedure computes non-increasing over-estimates of the global reachable state set until a greatest fixed point is found. Each greatest-fixed point is evaluated through a sequence of non-decreasing exact least fix-points performed on each sub-FSMs by procedure TRAV. SAT-based contributions are computed by procedure SATSTRENGTHEN.

```
MBM (TR, I, MAXCL)
    for all sub-machines $i$
        $R^i$ = $C^i$ = $\top$
        traverse$^i$ = TRUE
    do
        converged = TRUE
        newR = TRAV$^+$ (I, TR, C, R, traverse)
        R = newR
        From = To = C $\wedge$ R
        NewC = SATSTRENGTHEN (From, TR, To, MAXCL)
        C = C $\wedge$ NewC
        for all sub-machines $i$
            if (traverse$^i$ = TRUE)
                converged = FALSE
    while (converged = FALSE)
    return (R $\wedge$ C)

TRAV$^+$(I, TR, C, R, traverse)
    for all sub-machines $i$
        if (traverse$^i$ == FALSE) continue
        oldR$^i$ = R$^i$
        newTR = TR $\downarrow$ C $\wedge$ R
        R$^i$ = TRAV (newTR, I)
        traverse$^i$ = FALSE
        if (oldR$^i$ $\neq$ R$^i$)
            for all sub-machines $j$ in fan-out of M$^i$
                traverse$^j$ = TRUE
    return (R)
```

Fig. 2. MBM Approximate Reachability with Clause Generalization.

To compute over-estimations of the reachable states, i.e.,

$$\text{TRAV}^+(\mathsf{TR}, \mathsf{From}) \supseteq \text{TRAV}(\mathsf{TR}, \mathsf{From})$$

function TRAV$^+$ essentially traverses all sub-FSMs *serially*. It uses previously computed R and C sets as constraints for each new computation performed by function TRAV. When the reachable state set of a sub-FSMs is updated by function TRAV, all sub-FSMs in its fan-out cones are marked (i.e., the traverse^j flag is set to TRUE) to be traversed again in the future.

SAT-based contributions are computed by SATSTRENGTHEN. Those contributions are accumulated by conjoining strengthening terms in C. Notice that C is stored as a set of clauses, even if we represent it as a single term. The user-defined integer value MAXCL, is used to specify the number of cubes generalized by this function.

The SAT-based C contribution is finally used to strengthen the reachable state sets computed by TRAV$^+$ as $(\mathsf{R} \wedge \mathsf{C})$. Notice that, the same set is used for both the domain and co-domain sets within the generalization procedure, and strictly inductive clauses will be selected by SATSTRENGTHEN. In the next section, within the FBF procedure, we will see how to apply inductive generalization more efficiently having two distinct reachable sets.

B. The Frame by Frame (FBF) Approach

The *Frame by Frame* (FBF) approach handles sub-FSMs in *parallel* performing a traversal step (an image) on each sub-machine. Interactions among the sub-FSMs are more fine-grained, so FBF is usually more expensive, but it results in stronger estimates of the reachable state set. Our modified FBF approach with clause generalization is shown in Figure 3. The codes adopts an *approximate image* (IMG$^+$) operator, which returns over-estimations of exact images:

$$\text{IMG}^+(\mathsf{TR}, \mathsf{From}) \supseteq \text{IMG}(\mathsf{TR}, \mathsf{From})$$

Variable j is the iteration counter, whereas i is used to specify sub-machines. As a consequence, R^i_j specifies the set of states reachable by sub-machine i at iteration j. R_j is the set of all R^i_j.

The key operation of FBF is over-approximate image computation. Function IMG$^+$ computes over-approximate images by taking the Cartesian product of images on TR partitions.

Once computed $\mathsf{R}_j = \text{IMG}^+(\mathsf{TR}, \mathsf{R}_{j-1})$ the standard FBF algorithm would proceed to compute the next image, by applying the IMG$^+$ function to R_j. In our case, function SATSTRENGTHEN (analyzed in Section IV) takes care of clause selection and generalization, i.e., our strengthening step is activated on the R_j, right after over-approximate image computation.

```
FBF (TR, I, MAXCL)
    $R_0$ = I
    j = 1
    do
        $R_j$ = IMG$^+$(TR, $R_{j-1}$)
        C = SATSTRENGTHEN ($R_{j-1}$, TR, $R_j$, MAXCL)
        $R_j$ = $R_j$ $\wedge$ C
        j = j + 1
    while ($R_j$ $\not\subseteq$ $R_{j-1}$)
    return (R)

IMG$^+$ (TR, From)
    To = $\top$
    for all sub-machines $i$
        newTR = TR $\downarrow$ From$^i$
        To$^i$ = IMG (newTR, From$^i$)
        To = To $\wedge$ To$^i$
    return (To)
```

Fig. 3. FBF Approximate Reachability with Clause Generalization.

An interesting scenario is provided here by the fact that we have both a R_j, i.e., the over-approximated image to be strengthened, and its exact counterpart, though just implicitly represented by $\mathsf{TR} \wedge \mathsf{R}_{j-1}$. This gives us the opportunity to operate both non-inductive and inductive generalizations, as the former is not allowed to strengthen R_j, whereas the inductive one could.

IV. Image Strengthening by Generalization

Figure 4 shows our SAT strengthening routine.

```
SatStrengthen (From, TR, To, MAXCL)
    SP = $\neg$ To
    C = $\top$
    if (pushEnabled)
        for all $\mathsf{cl}_j \in$ From
            if ($\mathsf{cl}_j \supseteq$ Img(TR, From))
                C = C $\wedge$ $\mathsf{cl}_j$
                SP = SP $\wedge$ $\mathsf{cl}_j$
    for (i=0; i<MAXCL; i++)
        $cube$ = SAT(SP)
        if ($cube$ == NULL) break
        cl = GeneralizeCube (From, TR, cube)
        C = C $\wedge$ cl
        SP = SP $\wedge$ cl
    return ($\neg$ C)
```

Fig. 4. SAT strengthening with clause generalization.

The procedure operates in the present and next state spaces, using a CNF encoding of the transition relation. Though some SAT details are hidden in GeneralizeCube, $\mathsf{TR} \wedge \mathsf{From}$ is an implicit representation of the image of From, whereas To is its over-approximation to be strengthened.

The first part of the function (if enabled by a user selectable flag pushEnabled) tries to recycle as many clauses as possible from the domain set of states (From). This is inspired by the *push* operation in IC3, and it is done though an iterative process with incremental SAT.

The second part of function SatStrengthen enumerates through candidate cubes for generalization. The cube search is performed in the SP co-domain space in a random way, as the selection cannot be "property-directed" as in IC3. Although the random choice may appear to be a weak point of the algorithm, it has to be noticed that even in IC3 the cube selection phase is a critical phase and it is still an open issue how to perform it.

We allow clauses to span on the entire set of variables, and we use the full (non-partitioned) transition relation, as scalability is guaranteed by two factors:

- The number of clauses is bounded.
- The transition relation is not expressed by BDDs, but directly converted from circuit to CNF clauses.

It may be useful to point out that in terms of SAT solving, following the experience of several researchers working on the IC3 methodology, it is possible to use one or two SAT solvers. On the one hand, is it possible to use just one SAT solver if an efficient implementation of incremental SAT and activation literals is put in place. On the other hand, two solvers can be easier to deal with in term of work-load distribution. In our case, the first solver is used during the search space exploration, to select cubes in the state space outside R_j. The second one is employed within the clause generalization phase, in a similar fashion to what happens within the IC3 algorithm, i.e., such that the obtained clause includes $\textsc{Img}(\mathsf{TR}, \mathsf{R}_{j-1})$ (exact image) and strengthens R_j.

The procedure returns the overall set of clauses (C) able to strengthen To and From, if clauses are inductive.

V. Experimental Results

In order to analyze the full potential benefit of our methodology, we ran experiments on a selected benchmark set extracted from the Hardware Model Checking Competition (HWMCC [9]) suite. Our selection followed the one adopted by Xu et al. [2], and it consists of 35 benchmarks on which BDD-based verification showed to have an edge over (or at least compare very well with) other verification techniques (such as interpolation and IC3).

Our prototype tool ran on an Intel i7 3370 Workstation with 8 MB cache memory, a clock speed of 3.40 GHz, 4 cores, 8 threads, 16 GBytes of main memory DDR III 1333, and hosting a Ubuntu 12.04 LTS Linux distribution.

Table I shows detailed data for both the MBM and FBF approaches. For this set of experiments the time limit was set to 1000 seconds. It essentially compares the base algorithm (columns Base) with the improved one (columns Gen) in terms of reachable states. The value of MAXCL, i.e., the number of generalized cubes, has been selected to obtain a significant strengthening without incurring in too high overheads in terms of CPU time and final BDD node size. In general, we make MAXCL vary in the range of a few thousands.

As strengthening, at least in some cases, makes the overall reachability effort heavier, we also report a lighter weight strengthening experiment (column GenR) where function SatStrengthen is not enabled within inner steps but just at the approximate reachability fix point. Column GenPush represents the variant with the push operation enabled (see Section IV). Column Ratio reports the improvements obtained, i.e., the ratio between the states reached without and with SAT-based strengthening (a value larger than one thus means that the number of states is smaller, i.e., strengthening was successful). To give a schematic overview of the data collected in Table I all columns Ratio are also plotted in Figure 5. The graph essentially gives an idea of the relative improvements obtained by SAT-based refinement, with respect to the purely BDD-based versions, in terms of number of states. Overall,

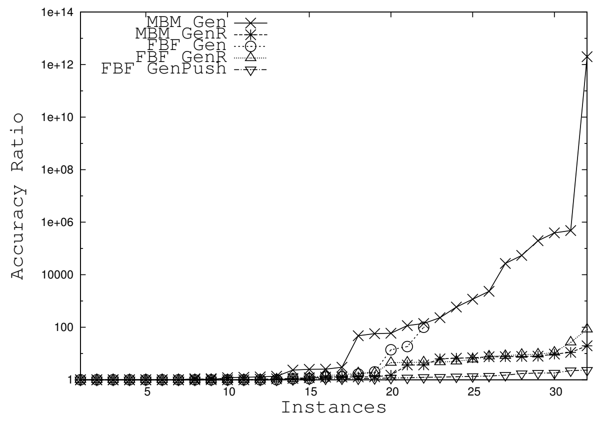

Fig. 5. Improvement achieved with our alternative MBM and FBF approaches.

results show that our sets of reachable states are often much stronger, on average from 1 to 2 orders of magnitude, and up to 12 orders of magnitude in one case.

TABLE I. Number of states reached by standard approximate reachability analysis and our improved versions with generalization. ovf means overflow on time, a dash ($-$) means data not available.

Model	#FF	MBM					FBF						
		Base	Gen	Ratio	GenR	Ratio	Base	Gen	Ratio	GenR	Ratio	GenPush	Ratio
6s105	151	3.91E+48	1.69E+46	232.13	4.93E+47	7.93	3.86E+48	2.79E+47	13.84	8.22E+47	4.69	2.14E+48	1.80
6s177	160	2.70E+41	ovf	–	2.44E+40	11.07	2.70E+41	ovf	–	5.73E+40	4.71	1.94E+41	1.39
6s179	185	1.89E+60	1.37E+60	1.37	1.47E+59	1.28	1.89E+59	1.74E+60	1.09	1.74E+60	1.08	1.81E+60	1.04
6s197	208	3.09E+60	2.22E+58	139.08	4.63E+59	6.67	2.94E+60	ovf	–	3.13E+57	9.40	2.61E+60	1.13
6s198	208	3.08E+60	ovf	–	8.34E+59	3.69	3.08E+60	ovf	–	6.70E+59	4.60	2.63E+60	1.17
6s48p0	66	2.64E+22	4.50E+19	587.60	3.48E+20	7.59	1.49E+22	8.07E+19	18.52	1.40E+20	106.00	1.19E+20	125.00
6s4	202	1.24E+56	ovf	–	3.39E+55	3.66	2.82E+56	ovf	–	5.60E+54	5.03	2.27E+56	1.24
6s52	208	5.17E+60	4.44E+57	1164.96	6.67E+59	7.75	4.85E+60	ovf	–	6.17E+59	7.85	2.81E+60	1.73
6s53	208	3.09E+60	1.57E+55	1.97E+05	4.11E+59	7.51	2.94E+60	ovf	–	4.89E+59	6.01	1.92E+60	1.50
bjrb07amba10andenv	63	2.66E+21	2.65E+21	1.005	2.66E+21	1.00	2.66E+21	2.64E+21	1.01	2.65E+21	1.005	2.38E+21	1.11
bjrb07amba7andenv	50	5.33E+17	4.68E+16	1.14	4.71E+17	1.13	5.33E+17	3.23E+17	1.65	3.61E+17	1.47	5.33E+17	1.00
bjrb07amba9andenv	53	2.97E+19	2.97E+19	1.00	2.97E+19	1.00	2.97E+19	2.97E+19	1.00	2.97E+19	1.00	2.97E+19	1.00
boblivear	77	1.27E+24	1.10E+22	115.72	1.34E+23	9.50	1.39E+23	ovf	–	1.70E+22	8.21	1.03E+23	1.35
boblivea	102	2.52E+29	5.29E+28	47.59	4.01E+29	6.27	9.08E+29	ovf	–	2.03E+29	4.48	8.27E+29	1.10
eijkbs3330	246	6.06E+44	1.27E+39	4.79E+05	1.49E+43	40.74	2.44E+44	ovf	–	1.42E+41	1715.91	9.59E+43	2.54
intel055	227	1.35E+42	5.32E+41	2.534	1.34E+42	1.00	1.35E+42	1.34E+42	1.00	1.33E+42	1.01	1.35E+42	1.00
intel059	285	1.05E+47	3.48E+46	3.00	8.71E+46	1.20	8.72E+46	8.72E+46	1.00	4.31E+46	2.02	3.54E+46	2.46
nusmvqueue	84	2.43E+30	9.67E+29	2.51	2.43E+30	1.00	4.86E+30	2.43E+30	2	4.86E+30	1.00	4.86E+30	1.00
pdtfifo1to0	142	6.62E+47	6.62E+47	1.00	6.62E+47	1.00	6.62E+47	6.62E+47	1.00	6.62E+47	1.00	6.62E+47	1.00
pdtpmsgigamax	123	6.81E+18	1.25E+14	5.43E+04	3.36E+17	20.20	6.81E+18	ovf	–	2.13E+16	318.00	3.01E+18	2.25
pdtpmsbufferalloc	66	5.73E+24	2.46E+21	2332.28	8.31E+23	6.89	5.73E+24	ovf	–	6.29E+22	9.10	2.45E+23	2.33
pdtpmseisenberg	201	1.35E+36	3.43E+30	3.92E+05	3.18E+34	42.27	1.34E+36	ovf	–	1.61E+34	83.25	7.39E+35	1.82
pdtpmstimeout	334	5.67E+28	5.67E+28	1.00	5.67E+28	1.00	5.67E+28	5.67E+28	1.00	5.67E+28	1.00	5.67E+28	1.00
pdtswvqis10x6p1	94	4.01E+30	7.00E+28	57.256	3.10E+30	1.29	4.01E+30	3.07E+30	1.31	2.22E+29	1.81	3.68E+29	1.09
pdtswvqis10x6p2	94	2.58E+29	4.35E+27	59.329	1.73E+29	1.497	2.58E+29	2.58E+29	1.00	1.69E+29	1.53	2.58E+29	1.00
pdtswvqis8x8p1	100	8.40E+30	3.57E+31	2.35	6.67E+31	1.259	8.40E+30	6.59E+31	1.28	4.62E+31	1.82	6.42E+31	1.31
pdtswvqis8x8p2	100	6.96E+31	6.96E+31	1.00	6.96E+31	1.00	6.96E+31	6.96E+31	1.00	6.96E+31	1.00	6.96E+31	1.00
pdtswvrod6x8p1	84	5.58E+26	4.69E+27	1.189	5.14E+27	1.09	5.58E+26	5.58E+26	1.00	4.69E+27	1.189	4.69E+27	1.19
pdtswvrod6x8p2	84	3.54E+26	3.54E+26	1.00	3.54E+26	1.00	3.54E+26	3.54E+26	1.00	3.54E+26	1.00	3.54E+26	1.00
pdtswvroz10x6p2	81	1.09E+27	1.09E+27	1.00	1.09E+27	1.00	1.09E+27	1.09E+27	1.00	1.09E+27	1.00	1.09E+27	1.00
pdtswvsam6x8p4	128	2.49E+39	2.49E+39	1.00	2.49E+39	1.00	2.49E+39	1.19E+39	1.25	2.49E+39	1.00	2.49E+39	1.00
pdtswvtma6x4p2	49	1.32E+18	1.32E+18	1.00	1.32E+18	1.00	1.32E+18	1.32E+18	1.00	1.45E+17	91.03	1.02E+17	12.90
pdtswvtma6x4p3	49	1.15E+18	1.04E+18	1.11	1.10E+18	1.05	1.15E+18	1.04E+18	1.11	1.10E+18	1.05	1.04E+18	1.11
pdtvissoap1	220	6.44E+30	2.40E+26	2.68E+04	3.26E+29	19.70	3.25E+29	ovf	–	1.21E+28	26.80	2.57E+29	1.26
pdtvsarmultip00	130	1.64E+16	8.19E+03	2.00E+12	1.26E+16	1.30	1.64E+16	1.64E+14	100.00	1.48E+15	11.00	5.20E+15	3.16

To complete the analysis, Figure 6 plots run times of the Gen, GenR, and GenPush cases (MBM and FBF) against the Base (MBM and FBF) ones. Approximation with generalization is usually slower than without. Anyhow, the difference is often quite small as the majority of the graph points are around or just above the main diagonal.

As far as verification is concerned, we concentrate on very hard verification instances on which BDDs perform extremely well when compared to other techniques. We start our discussion on verification with a case study. Figure 7 shows data on circuit 6s48p0, which our verification tool was previously unable to solve with any BDD-based technique within less that 1 hour of CPU time. The figures plot the BDD size of reachable states (top) and the CPU time (bottom) during each single backward traversal iteration, for three verification methods: Pure backward, forward/backward, and forward/backward with SAT-based strengthening. The figure shows that backward reachability, exploiting a strengthened over-approximate reached set as care set [10] computed in the forward analysis, could complete the problem in less than one hour.

Table II shows detailed data on a few very hard-to-prove verification instances. In this case we extended the time limit to 3600 seconds, i.e., 1 hour. The table reports run times for pure BDD-based verification based on pure forward (column Fwd), pure backward (column Bwd), approximate forward followed by exact backward (column Fwd^+/Bwd), and approximate forward followed by exact backward with our generalization strengthening. In both Fwd^+/Bwd columns we use the best among the MBM and FBF strategies, and we indicate the method adopted (column Method) for our improved strategy. Although the backward approach always runs out of time, we report it for the sake of completeness. The Fwd^+/Bwd approach (without generalization) is often better than Fwd but for the first design. Anyhow, our approximate forward and exact backward approach with generalization improves it up to a factor of 6. As a last remark, let us notice that the MBM approach seems to obtain larger benefits than FBF. We are trying to understand whether this is due to the intrinsic nature of the algorithms, to our specific implementation, or to the experiments we ran so far.

TABLE II. Performance comparison among standard approximate reachability analysis and our improved versions with generalization. ovf means overflow on time.

Model	Fwd	Bwd	Fwd^+/Bwd	Fwd^+/Bwd with generalization	
				Time	Method
6s52	19.83	ovf	ovf	24.50	MBM
6s179	ovf	ovf	1472.92	989.89	MBM
intel59	131.97	ovf	89.87	2.99	MBM
pdtswvqis10x6p1	402.93	ovf	203.54	2.59	MBM
pdtswvqis10x6p2	294.85	ovf	45.67	2.39	MBM
pdtswvqis8x8p1	42.23	ovf	34.34	6.78	FBF

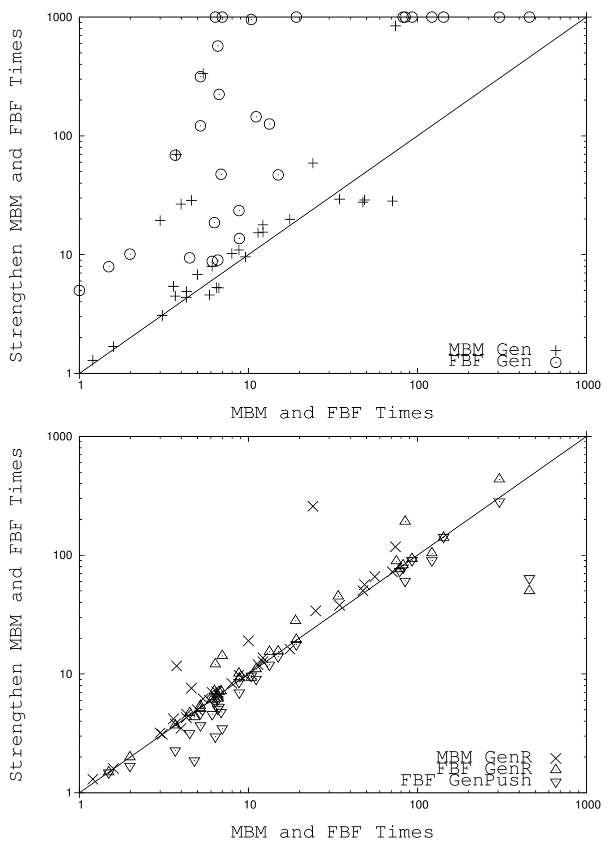

Fig. 6. CPU Times (in seconds) required by the reachability algorithms: Strengthened versions against original ones.

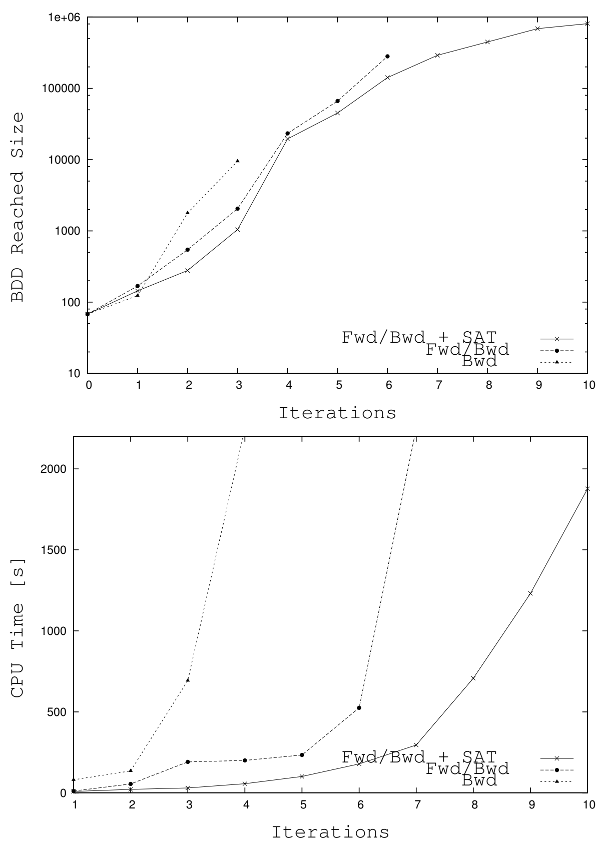

Fig. 7. Verification of model 6s48p0: Comparative analysis of reached size and elapsed CPU time using pure backward, forward/backward or forward/backward with generalization.

VI. Conclusions and Future Work

In this paper, we explore the use of inductive clause generalization as an additional player in BDD-based reachability. Cube generalization, a core step of the IC3 model checking algorithm, is used to strengthen BDD-based over-approximations of state sets, computed modifying standard algorithm such as the Machine By Machine (MBM) and the Frame by Frame (FBF) strategies. The resulting approach benefits from the orthogonal power of BDD and CNF representations. Preliminary experimental results confirm that this approach can provide tighter representations of reachable state sets and better BDD-based forward/backward verification.

As a last comments, albeit we do not have experimental results available yet, let us notice that we are in the process of implementing a tight interaction and exploitation of strengthened reachable states within interpolation and IC3 based methods.

References

[1] R. E. Bryant, "Graph–Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, vol. C–35, no. 8, pp. 677–691, Aug. 1986.

[2] J. Xu, M. Williams, H. Mony, and J. Baumgartner, "Enhanced reachability analysis via automated dynamic netlist-based hint generation," in *FMCAD*, 2012, pp. 157–164.

[3] H. Cho, G. D. Hatchel, E. Macii, B. Plessier, and F. Somenzi, "Algorithms for Approximate FSM Traversal Based on State Space Decomposition," *IEEE Trans. on Computer-Aided Design*, vol. 15, no. 12, pp. 1465–1478, Dec. 1996.

[4] S. G. Govindaraju, D. L. Dill, A. Hu, and M. A. Horowitz, "Approximate Reachability Analysis with BDDs using Overlapping Projections," in *Proc. 35th Design Automation Conf.* San Francisco, California: IEEE Computer Society, Jun. 1998, pp. 451–456.

[5] K. L. McMillan, "Interpolation and SAT-based Model Checking," in *Proc. Computer Aided Verification*, ser. LNCS, W. A. H. Jr. and F. Somenzi, Eds., vol. 2725. Boulder, CO, USA: Springer, 2003, pp. 1–13.

[6] A. R. Bradley, "Sat-based model checking without unrolling," in *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI*, ser. Lecture Notes in Computer Science, R. Jhala and D. A. Schmidt, Eds., vol. 6538. Springer, Jan. 2011, pp. 70–87.

[7] H. Chockler, A. Ivril, and A. Matsliah, "Computing Interpolants without Proofs," in *Proceedings of the 7th International Haifa Verification Conference on Hardware and Software: Verification and Testing*, ser. HVC '12, 2012.

[8] A. R. Bradley and Z. Manna, "Checking safety by inductive generalization of counterexamples to induction," in *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings*, ser. FMCAD, 2007, pp. 173–180.

[9] A. Biere, K. L. Claessen, and T. Jussila, "The Hardware Model Checking Competition Web Page, http://fmv.jku.at/hwmcc," 2012.

[10] G. Cabodi, P. Camurati, and S. Quer, "Symbolic Forward/Backward Traversals of Large Finite State Machines," *JSA Journal of System Architecture, The EUROMICRO Journal*, vol. 46, no. 12, pp. 1137–1158, Oct. 2000.