Automated Abstraction by Incremental Refinement in Interpolant-based Model Checking

G. Cabodi, P. Camurati, M. Murciano
Dipartimento di Automatica ed Informatica
Politecnico di Torino - Torino, Italy
Email: {gianpiero.cabodi, paolo.camurati, marco.murciano}@polito.it

***Abstract*—This paper addresses the field of Unbounded Model Checking (UMC) based on SAT engines, where Craig interpolants have recently gained wide acceptance as an automated abstraction technique.**

We start from the observation that interpolants can be quite effective on large verification instances. As they operate on SAT-generated refutation proofs, interpolants are very good at automatically abstract facts that are not significant for proofs. In this work, we push forward the new idea of generating abstractions *without* resorting to SAT proofs, and to accept (reject) abstractions whenever they (do not) fulfill given *adequacy* constraints. We propose an integrated approach smoothly combining the capabilities of interpolation with abstraction and over-approximation techniques, that do not directly derive from SAT refutation proofs.

The driving idea of this combination is to incrementally generate, by refinement, an abstract (over-approximate) image, built up from equivalences, implications, ternary and localization abstraction, then (eventually) from SAT refutation proofs.

Experimental results, derived from the verification of hard problems, show the robustness of our approach.

I. Introduction

Symbolic model checking [1] is a method for verifying temporal properties of finite state systems, which relies on a symbolic representation of sets, typically through Binary Decision Diagrams (BDDs) [2]. Symbolic representations of state sets have been by far the key factor for BDD success in symbolic model checking [1]. Given the canonicity of BDDs, and the efficient implementations of quantification, image and pre-image operators could produce Boolean functions representing state sets. Unfortunately, scalability problems practically limit BDD usage at circuits with tens to few hundreds latches.

By contrast, bounded model checking (BMC) [3] can falsify temporal properties using Boolean satisfiability (SAT). BMC is efficient at producing counter-examples and it has been shown to be more robust and scalable than BDD-based symbolic model checking. As a matter of fact, SAT tools are the core technology to attack larger problems. And Inverted Graphs (AIGs), or similar (non-canonical) circuit-based representations [4] are used in order to model circuit unrollings. Circuits are either converted to CNF clauses, to be processed by SAT solvers, or directly manipulated by circuit-based solvers. However, BMC is not complete, as it only guarantees the correctness of a property up to a given bound.

Therefore, specific techniques are required in order to support Unbounded Model Checking (UMC) in SAT based environments. For the sake of detail, the ability to check reachability fix-points is the main difference between BMC and UMC. All UMC approaches basically rely on one or more methods able to detect that the forward, backward or mixed analysis they perform is complete.

A. Related Works

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

Inductive proofs are at the base of the most of such approaches [5], [6], [7], [8], all following the seminal work of Sheeran et al. [9]. Fix-point checks are proved inductively, whereas completeness is based on uniqueness constraints, expressing loop-free paths between states. Unfortunately, the longest loop-free path can be exponentially longer than the diameter of the reachable state space, thus most of the research in this field has concentrated on finding tight sets of inductive invariants, i.e. over-approximations of reachable states, quite often sufficient for inductive proofs.

In order to avoid the exponential depth, symbolic representations of state sets are an alternative approach. But they are generally difficult to manipulate within non BDD-based frameworks, as both CNF and circuit-based representations can lead to memory explosion. Williams et al. [10] first adopted Boolean Expression Diagrams (BEDs), for the removal of quantifiers. Abdulla et al. [11] adopted Reduced Boolean Circuits (RBCs), i.e., a variant of BEDs, to represent formulas on which they performed existential quantifier elimination through substitution, scope reduction, etc. McMillan [12], later followed by Kang and Park [13], proposed quantifier elimination through the enumeration of SAT solutions (*all-solutions SAT*). Ganai et al. [14] extended the previous approaches by using "circuit co-factoring". The authors adopted a circuit to represent state sets, and they used circuit-based co-factoring to capture a large set of states in every SAT enumeration step.

All the above methods potentially converge faster than [9], although they share the common problem of possibly exponential state set representations.

Abstraction techniques represent an orthogonal direction to tackle complexity, as they seek and remove those parts of a circuit/system that are not relevant for the proof. Within this general path of research, our work follows the ideas first

978-1-4244-2820-5/08/$25.00 ©2008 IEEE

introduced by McMillan in [15], who used Craig interpolants for Unbounded Model Checking. Craig interpolants exploit the ability of Modern SAT solvers to generate proofs of unsatisfiability. Over-approximations of the reachable states are computed, starting from refutation proofs of (unsatisfied) BMC-like runs. The approach can be viewed as an iterative refinement of proof-based abstractions to narrow down a proof to relevant facts and its convergence is bound to the state graph diameter. Craig interpolant's most interesting features are its completeness and the automated abstraction mechanism, while the drawbacks are (again) mainly related to the potential size blow-up of SAT-based interpolant circuits, and to the convergence of over-approximate reachability.

New applications and improvements over the base method [15] were proposed in [16], [17], [18], [19], in order to push forward applicability and scalability of the technique. This paper is a direct follow-up of [18], as we push forward our previous idea, to compute abstractions at the image level, generating interpolant-like over-approximations, without resorting to SAT refutation proofs.

B. Motivations

Our experience [20], [21] shows that SAT-based Craig interpolants, combined with preliminary computations of inductive invariants, can prove a broad range of verification instances. A careful analysis of the unproved problems lead us to the following observations:

- Craig interpolants tend to produce highly redundant circuit representations of state sets, derived from SAT-generated proofs (often of unpredictable size). Circuits can be compacted by means of logic synthesis optimizers, that are limited in terms of scalability, and thus poorly working with big interpolants
- Over-approximation can drastically reduce the forward diameter (as long state transition paths can by bypassed by over-approximation). But it can also trigger state space explorations within unreachable areas, with direct consequences in terms of both traversal depth and reachable state set size
- Craig interpolants combine the power of forward and backward reachability, by over-approximating forward states within the region left "free" by backward reachability. Whenever the backward unrolling representation is complex, and the "free" region shrinks, interpolant representations tend to explode.

Due to the previously mentioned problems, we seek for *alternative ways to generate state set over-approximations*, whose circuit size can be better kept under control, and we focus our efforts on the generation of *tighter over-approximations*.

C. Contributions

In this paper, we propose a set of abstraction techniques not directly derived from SAT refutation proofs, although controlled by SAT-based "adequacy" checks. We incrementally compute sets of atomically simple over-approximations, with increasing complexity and computational effort.

Our approach is a follow-up of our previous experience [18], where we already explored the combination of interpolants and localization abstraction. With the current work we go much beyond that idea, as we introduce an automated abstraction procedure, based on incremental *learning* and *simplification* steps. We learn small atomic constraints such as equivalences and implications between state variables. We simplify circuits representing state sets and combinational unrollings, by exploiting equivalences and implications, plus ternary and localization abstractions. Compared to [19], our method works on abstractions at the image level, whereas they seek for abstraction/refinement steps to be used for entire interpolant-based traversals.

We adopt:

- cube-based over-approximations, by detecting state variables temporarily stuck at constant values
- equivalence classes and implications among state variables, able to provide simple over-approximations, at the base of powerful circuit optimizations
- abstractions based on ternary logic models.

Starting from the above considerations, our approach proposes two main novel contributions:

1) We adopt state set over-approximation techniques that are novel for interpolant-based Model Checking
2) We propose an integrated approach for image computation, incrementally combining and tuning the above over-approximation techniques within a unified SAT-based (complete) Unbounded Model Checking approach.

Our experimental results concentrate on proving correct properties, and they show that the proposed methods improve the original one by making it faster, more robust and scalable. We show experiments where in some cases we are able to complete difficult instances, not achievable with previous techniques.

D. Outline

Section II introduces background notions on notation, BMC and UMC, and SAT-based Craig interpolant Model Checking. Sections III and IV present our main contributions. Section V discusses the experiments we performed. Section VI 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. The state space and the (free) inputs are defined by indexed sets of Boolean variables $V = \{v_1, \ldots, v_n\}$ and $W = \{w_1, \ldots, w_n\}$, respectively. States correspond to the valuations of variables in V, whereas transition labels correspond to the valuations of variables in W. We indicate next states with the primed variable set $V' = \{v'_1, \ldots, v'_n\}$.

Whenever we explicitly need time frame variables, we use $V^i = \{v_1^i, \ldots, v_n^i\}$ and $W^i = \{w_1^i, \ldots, w_n^i\}$ for variable instances at the $i-$th time frame. We also adopt the short notation[1] $V^{i..j}$ for $V^i, V^{i+1}, ..., V^j$. Similarly for W.

A set of states is expressed by a state predicate $S(V)$ (or $S(V')$ for the next state space). $I(V)$ is the initial state predicate. We use $P(V)$ to denote an invariant property, and $F(V) = \neg P(V)$ for its complement (as it is often used as target for bug search). With abuse of notation, in the rest of this paper, we make no distinction between the characteristic function of a set and the set itself.

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

$$T(V, W, V') = \bigwedge_i t_i(V, W, v_i') = \bigwedge_i (v_i = \delta_i(V, W))$$

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

A state set S' is reachable from state set S in k steps if there exists a path of length k, in the labeled state transition structure, connecting a state belonging to S to another one belonging to S'; equivalently

$$S(V^0) \wedge \bigwedge_{i=0}^{k-1} T(V^i, W^i, V^{i+1}) \wedge S'(V^k)$$

The image operator $\text{IMG}(T, From)$ computes the set of states To reachable in one step from the states in $From$

$$\begin{array}{rcl} To(V') & = & \text{IMG}(T(V, W, V'),\ From(V)) \\ & = & \exists_{V,W}(From(V) \wedge T(V, W, V')) \end{array}$$

An over-approximate image (or pre-image) is any state set including the exact image

$$To^+ \quad = \quad \text{IMG}^+(T,\ From) \quad \supseteq \quad \text{IMG}(T,\ From)$$

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

$$\begin{array}{rcl} To(V) & = & \text{PREIMG}(T(V, W, V'),\ From(V')) \\ & = & \exists_{W,V'}(From(V') \wedge T(V, W, V')) \\ & = & \exists_W From(\delta(V, W)) \end{array}$$

B. Bounded Model Checking

SAT-based Bounded Model Checking (BMC) [3] considers only $k-$bounded reachability, as expressed by the propositional formula

$$\text{BMC}_0^k \quad = \quad I(V^0) \wedge \bigwedge_{i=0}^{k-1} T(V^i, W^i, V^{i+1}) \wedge F(V^k)$$

A bounded proof is thus translated into a SAT problem. BMC_0^k is satisfiable *iff* there is a counter-example (a path from I to F) of length k. In the case of circuits, existential quantification is conveniently applied to intermediate sets of state variables

$$\begin{array}{rcl} \text{BMC}_0^k = & I(V^0) \wedge \exists_{V^{1..k}}(\bigwedge_{i=0}^{k-1} T(V^i, W^i, V^{i+1}) \wedge F(V^k)) \\ = & I(V^0) \wedge Cone_k(V^0, W^{0..k-1}) \end{array}$$

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

$$\begin{array}{rcl} Cone_k & = & Cone_k(V^0, W^{0..k-1}) \\ & = & \exists_{V^{1..k}}(\bigwedge_{i=0}^{k-1}(V^{i+1} = \delta_i(V^i, W^i)) \wedge F(V^k)) \\ & = & F(\delta(...(\delta(W^0, V^0))), W^{k-1}) \end{array}$$

The main advantage of using $Cone_k$ (a combinational circuit) instead of transition relation instances ($\bigwedge_i T_i$), is that several circuit-based simplifications, besides Cone-Of-Influence (COI) reductions, are possible on $Cone_k$, from constant propagations to combinational optimizations, before moving to CNF- or circuit-based SAT.

C. State sets and Fix-points in Unbounded Model Checking

Although reachability is usually formulated in terms of the image and/or pre-image operators, we will here express backward reachability using $Cone_k$, as previously defined. SAT-based model checking approaches generally keep explicit representations of circuit unrollings instead of (exact) state set representations, due to the inherent complexity of quantification operators. The set of states (backward) reachable from F in (exactly) k steps can be obtained by primary input quantification over a circuit unrolling

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

The overall set of backward reachable states is the union of all reachable states up to depth k

$$\begin{array}{rcl} BckR_{0..k}(V) & = & \bigcup_{i=0}^{k} BckR_i(V) \\ & = & \bigcup_{i=0}^{k} \exists_{W^{0..i-1}} Cone_i(V, W^{0..i-1}) \\ & = & \exists_{W^{0..k-1}} \bigcup_{i=0}^{k} Cone_i(V, W^{0..i-1}) \end{array}$$

where distributivity of existential quantification over union has been applied. We introduce the short notation $Cone_{0..k}$ for $\bigcup_{i=0}^{k} Cone_i$. So backward reachable states are defined as

$$BckR_{0..k}(V) = \exists_{W_{0..k-1}} Cone_{0..k}(V, W_{0..k-1})$$

A backward reachability fix-point could be checked by a SAT run on the following Boolean formula

$$BckR_{k+1}(V) \wedge \neg BckR_k(V)$$

or in a simpler one, with quantified state sets on the second term only

$$Cone_{k+1}(V, W_{0..k}) \wedge \neg BckR_k(V)$$

Unfortunately, both the above formulations are difficult to manipulate in many practical cases, due to the complexity of SAT-based quantification.

[1] $V^{i..j}$ is defined if $i \leq j$, otherwise we conventionally define it as a void variable set

D. Craig Interpolants in Model Checking

Given two inconsistent formulas A and B ($A \wedge B = 0$), an interpolant C is a formula such that:

1) It is implied by A
2) It is inconsistent with B, i.e., $C \wedge B$ is unsatisfiable
3) It is expressed over the common variables of A and B.

A Craig interpolant $C = \text{ITP}\ (A,\ B)$ is an AND/OR circuit that can be computed in linear time from the refutation proof of $A \wedge B$. Albeit the computation is linear, the refutation proof itself can be exponentially larger than A and B.

A k-adequate over-approximate image is an $\text{IMG}^+(T, From)$ that does not intersect any state on paths of length k to F. Using the $Cone_{0..k}$ circuit unrolling, a k-adequate over-approximate image is

$$\text{IMG}^+_{Adq}(T, From, Cone_{0..k})$$

defined as follows[2]:
IMG^+_{Adq} is $undefined$ *iff*

$$From(V) \wedge T(V, W_0, V') \wedge Cone_{0..k}(V', W^{1..k}) \neq 0$$

Otherwise, it is computed by interpolation:

$$\text{IMG}^+_{Adq}(T, S, Cone_{0..k}) \quad = \quad \text{ITP}(S(V) \wedge T(V, W_0, V'), \\ Cone_{0..k}(V', W_{1..k}))$$

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.

McMillan [15] proposed an effective fully SAT-based Unbounded Model Checking algorithm, exploiting interpolants, as sketched in Figure 1.

```
INTERPOLANTMC (I, T, F)
    k = 0
    do
        Cone_{0..k} = CIRCUITUNROLL(F, δ, k)
        res = FINITERUN (I, T, Cone_{0..k})
        k = k + 1
    while (res = undecided)

FINITERUN (I, T, Cone)
    if (SAT(I ∧ T ∧ Cone)
        return (reachable)
    R = I
    while (true)
        To = IMG+_Adq (T, R, Cone)
        if (To = undefined)
            return (undecided)
        if (To ⇒ R)
            return (unreachable)
        R = R ∨ To
```

Fig. 1. Interpolant-based Verification.

While INTERPOLANTMC is the entry point of the algorithm, routine FINITERUN takes care of the interpolant-based over-approximate traversal. The latter function may end up with three possible results:

- *"reachable"*, if it proves F 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 F is intersected by the over-approximate state sets. Then, k in increased for a new FINITERUN call.

McMillan [15] proved that the previous algorithm is sound and complete. In synthesis, let us assume I and F mutually unreachable: if $k < d$, a k-adequate set can produce a non k-adequate image. In this case, *"undecided"* is returned and k is increased. Otherwise, when $k \geq d$, IMG^+_{Adq} is always adequate. At this point, the algorithm will terminate with an approximate reachability fix-point.

According to [17], k can be incremented by the depth of the last FINITERUN execution to avoid a quadratic number of image computations.

III. SAT-BASED OVER-APPROXIMATE IMAGE

The over-approximate images we are concerned with are defined by the following two conditions:

- An over-approximate image includes the exact image. Avoiding existential quantification, a given $To^+(V')$ is an over-approximate image if:

$$T(V, W_0, V') \ \wedge \ From(V) \Rightarrow To^+(V')$$

- We look for $k-$adequate images, as adequacy (by increasing k values) is the condition adopted to incrementally tighten abstractions. Given $Cone_{0..k}$

$$To^+(V') \wedge Cone_{0..k}(V', W^{1..k}) = 0$$

Given the above observations, we look for an over-approximate image, computed as a Boolean conjunction of several atomic over-approximations

$$To^+(V') = \textstyle\bigwedge_i to_i{}^+(V')$$

characterized by the property that each $to_i{}^+$ factor includes the exact image, but only the $\bigwedge_i to_i{}^+$ product is $k-$adequate (whereas each $to_i{}^+$ is not required to be $k-$adequate)

$$\begin{array}{lcl} \forall_i, T(V, W_0, V') \ \wedge \ From(V) & \Rightarrow & to_i{}^+(V') \\ \bigwedge_i to_i{}^+(V') \wedge Cone_{0..k}(V' W^{1..k}) & = & 0 \end{array}$$

Hence, we look for a product of $to_i{}^+$ factors, where each one is an over-approximate image in itself, and the overall product is $k-$adequate. We choose to possibly find several (small) $to_i{}^+$, instead of a single To^+, as we adopt an iterative algorithm to select among multiple (small) candidate over-approximations. Whereas it might be computationally infeasible to enumerate all functions of the (V') variables as candidate over-approximations, we explore given classes of atomic functions.

[2]Indexes of input variable sets have been shifted up in $Cone_k$, from $W^{0..k-1}$ to $W^{i..k}$, in order to use W^0 variables in the forward T instance.

The second criterion we adopt is the incremental refinement of an initially coarse To^+ abstraction:

- We consider ${to_i}^+$ candidates, selected by incremental complexity. We group them by iterating through classes of functions, so that we first capture the simplest/easiest ones, before moving to more complex candidates
- Whenever we find an over-approximation, we possibly use it to simplify the data structure for the next steps (e.g. the transition relation and the backward cone used for adequacy checks)
- We stop the iterative process whenever the over-approximation is adequate. As the selection we operate is not complete, we possibly end up computing a Craig interpolant (on the simplified backward cone and forward transition relation) from a SAT refutation proof.

Figure 2 shows the skeleton of our iterative image over-approximation algorithm.

```
IMG+_Adq (T, From, Cone)
    To+ = 1
    Class = constClass
    do
        Cand_class = GETCANDIDATES (Class)
        Abstr_class = { }
        foreach c_j (V') ∈ CandClass
            if ACCEPT (c_i, T, From, Cone)
                Abstr_class = Abstr_class ∪ {c_j}
        To+_class = OVERAPPRSET (Abstr_class)
        To+ = To+ ∧ To+_class
        if ¬SAT (To+ ∧ Cone)
            return (To+)
        SIMPLIFY (Cone, Abstr_class)
        SIMPLIFY (T, Abstr_class)
        Class = NEXTCLASS()
    while (Class ≠ emptyClass)
    return (To+ ∧ ITP (From∧ T, Cone))
```

Fig. 2. Over-approximate k-adequate image computation.

Let $Class$ represent the choice for the type of over-approximations to be considered: we start by `constClass`, the equivalence class of *constant variables*, then NEXTCLASS() returns, at each new iteration, equivalences, implications, ternary abstraction and localization abstraction. For each class, we iterate through the c_j candidates, in order to gather the *accepted* ones. A candidate is accepted if it is a valid over-approximation, i.e. adequacy conditions are kept after applying the over-approximation. Once all accepted candidates for a given class have been selected, we generate the class over-approximate image (To^+_{class} = OVERAPPRSET(Abstr$_{class}$)). To^+_{class} is a new refinement for (and it is and-ed to) the overall image To^+. If the refined To^+ is adequate (¬SAT ($To^+ \wedge Cone$)), it is returned as a result, otherwise we exploit the set of atomic abstractions (Abstr$_{class}$) to simplify $Cone$ and T, and we move to the next class.

When we have looped through all classes, without returning an adequate To^+ set, this means that the over-approximation is still too coarse, and we end up calling SAT-based interpolation (which is possibly easier due to all previous simplifications on T and $Cone$).

All of the above mentioned classes will be discussed in section IV, where, for each one, we motivate it, we discuss the abstraction obtained, and how to efficiently compute $k-$adequacy and to simplify $Cone$ and T.

IV. ABSTRACTION CLASSES AND ADEQUACY CHECKS

Abstraction classes are now individually discussed, as well as the related acceptance criteria, set over-approximation and simplification strategies.

A. Constant variables

We first look for variables equivalent to constant values, i.e. variables that are implied to constant values in the next state. More formally, let variable $x'_j \in X'$ be the generic next state variable, we consider two possible over-approximation literal candidates: $\neg x'_j$ and x'_j. Therefore, the candidate class returned by GETCANDIDATES(`constClass`) is the set of all (direct and complemented) literals:

$$\text{GETCANDIDATES}(constClass) = \{x'_1, \neg x'_1, \ldots, x'_n, \neg x'_n\}$$

We efficiently accept/reject individual candidates by an incremental SAT procedure that iterates through all literals, and detects the implied ones. An accepted literal l_j is such that:

$$S \wedge T \Rightarrow l_j$$

The next step is the explicit computation of $To^+_{constClass}$ (function OVERAPPRSET), as the conjunction of all implied literals.

Given a cube, the simplification task (performed by the function SIMPLIFY(T, Abstr$_{constClass}$)) is straightforward. We just need to replace variables by the corresponding constant values, and this action generally reduces the overall AIG node count for $Cone$. SIMPLIFY(T, Abstr$_{constClass}$)) is easier, as we simply need to remove the T components corresponding to constant values.

It is worth noticing that constant state variables typically appear at the initial steps of forward traversals and possibly throughout the traversal of externally constrained (by external assumptions) systems.

Keeping exact values for implied variables is a way to tighten over-approximation vs. other abstraction techniques. Craig interpolants as well as localization abstraction could, for instance, abstract away implied variables (whenever they are not relevant for the proof). But this might result in a looser over-approximation, and possibly trigger visits of unreachable states. By explicitly expressing $To^+_{constClass}$, we also remove implied variables from further computations (as in other abstraction techniques), but the $To^+_{constClass}$ factor in the overall over-approximate image will constrain the corresponding present state variables at their exact value, at the next forward traversal iteration.

B. Equivalences and Implications

After detecting constant state variables, we look for equivalences (modulo complementation) between couples of next state variables.

Let literals l_i and l_j be two literals to be considered for equivalence, the corresponding candidate equivalence is $eq_{ij} = l_i \Leftrightarrow l_j$.

The set of candidate equivalences is obviously quadratic in the number of variables, as well as the iteration for acceptance checks. Again we can exploit efficient solutions based on equivalence classes and incremental SAT, inspired by the ones proposed in [7], [8].

$To^{+}_{eqClass}$ (function OVERAPPRSET) is straightforwardly computed as the conjunction of all proved equivalences.

Function simplification given a set of equivalences is performed by means of variable merging (for each equivalence class we substitute each variable literal with a class representative literal).

Once equivalences have been detected and used to simplify $Cone$ and T, we consider variable implications, again, modulo complementation. Implications are accepted similarly to equivalences, but their computation cannot rely on equivalence classes any more. Furthermore, implications cannot be used for direct function simplification. We just need to explicitly represent them (To^{+}_{Impl}) in order to tighten the candidate To^{+}.

We also explicitly add the To^{+}_{Impl} constrain to $Cone$ as an additional redundant factor, for space search restriction purposes.

C. Ternary abstraction

Three-Valued Logic is at the base of several synthesis and verification approaches, following the general idea to encode an additional logic value, representing either the *unknown*, or the $\{0,1\}$ set. Our approach directly follows an idea proposed by Bres et al. [22], following works by Malik [23] and Shiple et al. [24]. Other related works can be found in the field of equivalence checking [25], where ternary logic is used in order to handle circuit initialization sequences.

The cited works are inspired from Scott's three-valued logic, built upon the usual two-valued Boolean logic by adding a third value $\perp$, that denotes the undefined or unknown value. Bres et al. [22] adopt a two bit encoding (sometimes called dual rail) for ternary constants ($(0,1)$ for $false$, $(1,0)$ for $true$, $(0,0)$ for $\perp$). A Boolean function f is represented by two bit functions f_0 and f_1, such that f_0 (resp. f_1) is the characteristic function of the set for which f evaluates to 0 (resp. 1). $f_\perp = \neg f_0 \wedge \neg f_1$ is the set for which the function is undefined (the *don't care* set). The function is completely defined if $f_1 \vee f_0 = 1$, and $f_\perp = 0$. Let us introduce the notation $f^3 = (f_0, f_1)$ for such a dual rail encoding.

Boolean operators for over the ternary encoding are defined by the following rules:

$$\begin{array}{lcll} \neg f^3 & = & \neg(f_0, f_1) & = & (f_1, f_0) \\ f^3 \vee g^3 & = & (f_0, f_1) \vee (g_0, g_1) & = & (f_0 \wedge g_0, f_1 \vee g_1) \\ f^3 \wedge g^3 & = & (f_0, f_1) \wedge (g_0, g_1) & = & (f_0 \vee g_0, f_1 \wedge g_1) \end{array}$$

Let s_i be a Binary variable, and σ_i the corresponding ternary one. Abstraction of variable σ_i in $f^3(\sigma_1, \ldots, \sigma_i, \ldots, \sigma_n)$ was done in [22], by setting variable σ_i to the unknown value:

$$f^3(\sigma_1, \ldots, \perp, \ldots, \sigma_n)$$

In the dual rail encoding, the σ_i variable is assigned the $\perp = (0,0)$ ternary constant, whereas all other σ_j variables are encoded by the symbolic ternary value $(s_j, \neg s_j)$ (using the corresponding binary variable).

Given the above assignments, function f^3 has now a possibly non-void don't care set $f_\perp$. And a binary over-approximation of f can be obtained by $\neg f_0 = f_1 \vee f_\perp \supseteq f$.

$$\begin{array}{c} TernaryAbs\,(f(s_1, \ldots, s_i, \ldots, s_n), s_i) = \\ \neg f_0((s_1, \neg s_1), \ldots, (0,0), \ldots, (s_n, \neg s_n)) \end{array}$$

Informally, we have replaced the characteristic function of the set "on which f is true" (the onset of f), by a superset "on which f is certainly not false".

We apply ternary abstraction to our over-approximate image computation, by looping through all primary input and state variables, and iteratively selecting them as possible ternary abstraction candidates. Ternary abstraction of the generic state/input variable s_i could violate an adequacy condition. This is the reason why it is *accepted* just if it still guarantees adequacy:

$$\begin{array}{lcll} From(V) & \wedge & ternaryAbs(T(V, W^0, V'), s_i) & \\ & \wedge & ternaryAbs(Cone_k(V', W^{1..k}), s_i) & \neq 0 \end{array}$$

The s_i variable can be one of the V' state variables, or the $W^{0..k}$ input variables. An accepted ternary abstraction does not directly produce any over-approximation of the image (To^{+}) (unless all W^0 variables are quantified). Hence, OVERAPPRSET$_{ternaryClass}$ generally returns no abstraction ($To^{+}_{ternaryClass} = 1$. On the other hand, ternary abstraction is directly applied to simplify T and/or $Cone$.

D. Localization abstraction

Localization abstraction is our last attempt to produce an over-approximation. A candidate abstraction corresponds to letting a state variable be free at the forward/backward boundary, i.e. simply re-labeling the chosen v'_i variable in $Cone(V', W^{1..k})$ by a fresh new variable.

The abstraction process is inspired by our previous work [18], and is managed analogously to ternary abstraction.

V. EXPERIMENTAL RESULTS

We implemented our algorithms on top of the PdTrav tool, a state-of-the-art verification framework which won two of the sub-categories at the 2007 Model Checking competition [20]. We compared results of our tool with and without the proposed methodology.

Our experiments ran on a Dual-Core Pentium IV 3 GHz Workstation with 3 GByte of main memory, running Debian Linux. We performed extensive tests, by specifically addressing proofs of correctness. For each verification instance, we used a 900 seconds time limit.

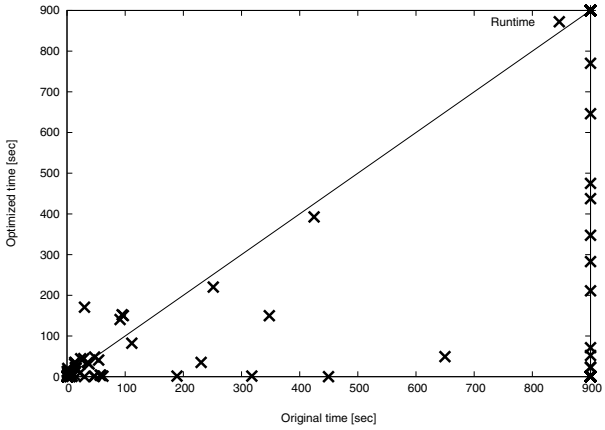

Fig. 3. Verification time on circuits coming from [20], [21].

We present results on circuits derived from the Model Checking competition [20], [21], and a few standard benchmarks from the VIS distribution [26], with particular emphasis on true hard-to-solve instances. The reason for reporting *only* true properties verification data is the ability of BMC in being the most effective technique for checking falsifications (i.e., false properties), as showed by the competition results.

The scattered plot in Figure 3 shows verification times on 152 benchmarks from the Model Checking competition. It compares a standard interpolant-based UMC technique (time on the X-axis) against the same strategy improved as suggested in this paper (time on the Y-axis). Figure 3 clearly shows a set of "easy" benchmarks, i.e., the ones which are solved in few seconds/minutes for both the techniques. The plot also highlights a set of problems that were not solved within the 900 seconds time limit with the standard interpolant computation, while they are solved with the proposed optimization. The overall results clearly show the robustness of the proposed approach.

Table I reports more detailed data on a few selected hard-to-solve verification instances. The meaning of columns follows: Model is instance name (industrial designs names have intentionally been hidden), # PI, # FF and # NODES represent the number of primary inputs, memory elements and AIG nodes of the circuit respectively; finally, Std ITP, Method A and Method B provide the verification time in seconds, with different strategies. Ternary abstraction was disabled in column Method A, while it was enabled in column Method B. The data clearly show that most of the problems could already be completed without resorting to ternary abstraction (see column Method A), whereas the latter plays an important role in a few cases. An inner look at those experiments showed that ternary abstraction requires a time overhead for acceptance checks, which is not worth the advantage gained with the abstraction, in several cases.

Though we still need to further enhance the self-tuning capability of our approach, it is already able to attack large problems, as shown by the results on the large industrial benchmarks.

VI. Conclusions

This paper addresses improvements to Interpolant-based model checking by means of an integrated approach exploiting over-approximation techniques that are novel for this field.

We describe an integrated approach for image computation, incrementally combining and tuning different techniques within a unified SAT-based (complete) Unbounded Model Checking approach.

The method we propose adopts the general skeleton of interpolant-based model checking procedure, and exploits preliminary SAT-based exploration of candidate atomic abstractions, within a global effort to tighten over-approximations and keep state set sizes under control.

Experimental results, specifically oriented to hard verification problems, show the robustness of our approach implemented on a state-of-the-art verification framework.

References

[1] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," *IEEE Trans. on Computer-Aided Design*, vol. 13, no. 4, pp. 401–424, Apr. 1994.

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

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

[4] P. Bjesse and A. Boralv, "DAG-Aware Circuit Compression For Formal Verification," in *Proc. Int'l Conf. on Computer-Aided Design*. San Jose, California: IEEE Computer Society, Nov. 2004.

[5] P. Bjesse and K. Claessen, "SAT–Based Verification without State Space Traversal," in *Proc. Formal Methods in Computer-Aided Design*, ser. LNCS, vol. 1954. Austin, TX, USA: Springer, 2000.

[6] M. L. Case, A. Mishchenko, and R. K. Brayton, "Inductively Finding a Reachable State Space Over-Approximation," in *Proc. Int'l Workshop on Logic Synthesis*, Lake Tahoe, California, May 2006.

[7] F. Lu and K. T. Cheng, "IChecker: An Efficient Checker for Inductive Invariants," in *High-Level Design Validation and Test Workshop*, 2006, pp. 176–180.

[8] G. Cabodi, S. Nocco, and S. Quer, "Boosting the Role of Inductive Invariants in Model Checking," in *Proc. Design Automation & Test in Europe Conf.* Nice, France: IEEE Computer Society, Apr. 2007.

[9] M. Sheeran, S. Singh, and G. Stålmarck, "Checking Safety Properties Using Induction and SAT Solver," in *Proc. Formal Methods in Computer-Aided Design*, ser. LNCS, W. A. Hunt and S. D. Johnson, Eds., vol. 1954. Austin, Texas, USA: Springer, Nov. 2000, pp. 108–125.

[10] P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta, "Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking," in *Proc. Computer Aided Verification*, ser. LNCS, E. A. Emerson and A. P. Sistla, Eds., vol. 2102. Chicago, Illinois: Springer-Verlag, Jul. 2000, pp. 124–138.

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

[12] K. L. McMillan, "Applying SAT Methods in Unbounded Symbolic Model Checking," in *Proc. Computer Aided Verification*, ser. LNCS, E. Brinksma and K. G. Larsen, Eds., vol. 2404. Copenhagen, Denmark: Springer, 2002, pp. 250–264.

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

TABLE I
VERIFICATION DATA ON HARD-TO-SOLVE EXPERIMENTS (− MEANS OVERFLOW, BOLD FONTS ARE USED FOR **best** RESULTS)

Model	#PI	#FF	#NODES	Std ITP	Method A	Method B
intel_005.blif	165	170	1776	(-)	26.88	**22.15**
intel_006.blif	345	350	3265	(-)	**518.82**	(-)
intel_020.blif	349	354	5735	(-)	757.51	**646.22**
intel_021.blif	360	365	5882	(-)	**410.02**	(-)
intel_024.blif	352	357	5710	(-)	(-)	**437.55**
intel_026.blif	486	492	6263	(-)	215.04	**211.10**
intel_029.blif	559	564	8816	(-)	(-)	**347.52**
sfeistel-inv.blif	68	296	6837	304.44	**111.51**	555.18
blackjack-inv.blif	5	103	3979	(-)	96.51	**35.58**
31_2_batch_1.blif	24	122	1506	(-)	23.12	**23.00**
soap-inv.blif	11	140	3605	(-)	199.15	**172.48**
intel_049.blif	136	77	1305	254.29	(-)	**193.83**
nusmvguidancep9.blif	84	86	1902	359.59	**249.91**	357.99
pdtvisns3p09.blif	21	101	3770	386.14	**315.51**	747.10
pdtvisvsa16a29.blif	32	172	7016	663.12	753.26	**306.39**
visprodcellp22.blif	30	63	2771	147.09	216.99	**116.42**
cmu.periodic.N.blif	32	34	1555	230.59	75.84	**35.12**
nusmv.guidance^2.C.blif	84	86	1920	251.45	**134.33**	220.22
nusmv.guidance^6.C.blif	84	86	1901	347.76	185.84	**149.77**
nusmv.guidance^7.C.blif	84	86	2001	**91.26**	140.41	134.88
nusmv.guidance^8.C.blif	84	86	1919	424.77	678.26	**392.52**
nusmv.reactor^6.C.blif	74	76	1396	(-)	(-)	**475.04**
vis.coherence^2.E.blif	6	29	1216	111.35	**60.01**	82.17
vis.coherence^3.E.blif	6	29	1214	649.73	60.57	**49.18**
industrial1.blif	120	76	1089	(-)	**93.43**	289.62
industrial2.blif	119	79	1103	(-)	**96.90**	224.40
industrial3.blif	119	78	1100	(-)	**255.72**	261.15
industrial4.blif	138	97	2172	(-)	422.33	**326.60**
industrial5.blif	113	459	7666	(-)	112.32	**108.67**
industrial6.blif	52	187	3600	126.08	**59.31**	75.16

[14] 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: IEEE Computer Society, Nov. 2004.

[15] 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.

[16] K. L. McMillan and R. Jhala, "Interpolation and SAT-Based Model Checking," in *Proc. Computer Aided Verification*, ser. LNCS, T. Ball and R. B. Jones, Eds., vol. 3725. Edimburgh, Scotlan, UK: Springer, 2005, pp. 39–51.

[17] J. Marques-Silva, "Improvements to the implementation of Interpolant–Based Model Checking," in *Proc. Computer Aided Verification*, ser. LNCS, D. Borrione and W. Paul, Eds., vol. 3725. Edimburgh, Scotlan, UK: Springer, 2005, pp. 367–370.

[18] G. Cabodi, S. Nocco, M. Murciano, and S. Quer, "Stepping Forward with Interpolants in Unbounded Model Checking," in *Proc. Int'l Conf. on Computer-Aided Design*. San Jose, California: ACM Press, Nov. 2006.

[19] 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.

[20] A. Biere and T. Jussila, "The Model Checking Competition Web Page, http://fmv.jku.at/hwmcc07/organizers.html," 2007.

[21] ——, "The Model Checking Competition Web Page, http://fmv.jku.at/hwmcc08/organizers.html," 2008.

[22] A. B. Y. Bres, G. Berry and E. M. Sentovich, "State Abstraction Techniques for the Verification of Synchronous Circuits," in *dcc02: Designing Correct Circuits 2002*, Grenoble, France, Apr. 2002.

[23] S. Malik, "Analysis of Cyclic Combinational Circuits," vol. 13, no. 7, 1994, pp. 950–956.

[24] G. B. T. R. Shiple and H. Touati, "Constructive Analysis of Cyclic e e Circuits," in *IDTC'96: International Design and Testing Conference*, Paris, France, 1996.

[25] Z. Khasidashvili and Z. Hanna, "Sat-based methods for sequential hardware equivalence verification without synchronization," in *BMC'03: First International Workshop on Bounded Model Checking*, Boulder, Colorado, Jul. 2003, pp. 593–607.

[26] R. K. B. et al., "VIS," in *Proc. Formal Methods in Computer-Aided Design*, ser. LNCS, M. Srivas and A. Camilleri, Eds., vol. 1166. Palo Alto, California: Springer, Nov. 1996, pp. 248–256.