EPiC Series in Computer Science

Volume 36, 2015, Pages 137–150

GCAI 2015. Global Conference on Artificial Intelligence

EPR-based *k*-induction with Counterexample Guided Abstraction Refinement

Zurab Khasidashvili, Konstantin Korovin, and Dmitry Tsarkov

[1] Intel Israel Development Center, Haifa 31015, Israel
zurabk@iil.intel.com
[2] The University of Manchester, UK
{korovin|tsarkov}@cs.man.ac.uk

Abstract

In recent years it was proposed to encode bounded model checking (BMC) into the effectively propositional fragment of first-order logic (EPR). The EPR fragment can provide for a succinct representation of the problem and facilitate reasoning at a higher level. In this paper we present an extension of the EPR-based bounded model checking with *k*-induction which can be used to prove safety properties of systems over unbounded runs. We present a novel abstraction-refinement approach based on unsatisfiable cores and models (UCM) for BMC and *k*-induction in the EPR setting. We have implemented UCM refinements for EPR-based BMC and *k*-induction in a first-order automated theorem prover iProver. We also extended iProver with the AIGER format and evaluated it over the HWMCC'14 competition benchmarks. The experimental results are encouraging. We show that a number of AIG problems can be verified until deeper bounds with the EPR-based model checking.

1 Introduction

SAT-based bounded and unbounded model checking [1, 2] is widely used in the industry for verification of hardware and software systems. Our focus in this paper, both theoretical and experimental, is on hardware verification, which is an important problem where the scalability and performance remain the challenge. To scale this approach further, two basic approaches are (1) extending SAT-based model checking to word level [3, 4], and (2) performing automatic abstraction refinement, as in the CEGAR approach [5], so that the verification problems become smaller with each iteration, and verification remains both sound and complete (no spurious counter-examples are reported).

For safely critical systems, often full correctness proofs are required. While BMC [1] is in principle a full-proof method (for finite-state systems), in practice it would require unrolling to unrealistically high bounds to achieve full proofs. Induction [6] is a basic technique that enables full proof for industrial-sized designs (still limited to modules with one or a few functional blocks). See, e.g., [7] for a successful application of an induction scheme to software verification. A number of advanced methods based on automatic generation of induction invariants exist that drastically increase the class of problems that can be fully verified [2, 8]. In this paper we develop an induction algorithm for EPR-based model checking, at word level.

The EPR fragment, also called the Bernays-Schönfinkel-Ramsey fragment, is a universal fragment of first-order logic where all function symbols are constants. The first encoding of BMC into EPR was

proposed in [9]. This bit-level encoding was extended to word level in [10, 11]. Experimental results reported in these papers regarding EPR-based BMC versus SAT and SMT-based approaches showed the promising potential of EPR-based BMC on industrial hardware model checking and equivalence checking problem instances with memories, which were the driving industrial examples for developing the EPR-based word-level BMC. The EPR fragment is NEXPTIME complete, and recent research has also shown that it can be used to encode bit-vectors exponentially more succinctly [12, 13].

The advantage of encoding the BMC and induction algorithms into EPR is that the rich expressive power of EPR enables a very concise representation of the problem, which, combined with the decidability of the EPR fragment, opens the door for an efficient implementation of BMC where unrolling is somewhat implicit and on-demand. Indeed, the EPR-based BMC in [11] works with a single copy of the transition relation, and the on-demand unrolling is "built-in" in the solving of the problem instances arising with the EPR encoding. For this reason, the BMC algorithm in [11] is called BMC1 (to indicate that unlike the regular SAT-based BMC, the circuit is not copied at each unrolling bound). In the case of bit-level verification, this approach is related in idea with QBF-based model checking, e.g., in [14, 15]. Despite multiple efforts, QBF-based BMC has not yet been able to extend the state of the art of BMC on large real-life examples. And similarly, in the past, the EPR-based BMC could only improve the state of the art on verification problems with large memories represented at word level. This confirms that taking advantage of the conciseness of the encoding by solving through a subtle abstraction-refinement procedure is a very challenging problem. This challenge of defining smart abstraction-refinement approaches is very relevant to subsequent discussions in this paper and we will clarify it farther for the case where instantiation based theorem provers like iProver [16] are used as the EPR solver engine.

The regular SAT-based BMC unrolls the (possibly pruned) circuit to a desired bound n and then solves the BMC formula using a SAT solver. Usually, to solve a BMC problem, not all the variables in the unrolled instance are relevant, even if the cone of influence of the assertion has already been pruned to contain only the relevant parts of the circuit. It is in general much more beneficial to avoid generation of irrelevant variables during the unrolling (but see [17] for discussion). With EPR-based BMC, an instantiation process having the effect of a partial, or more precisely, discrete unrolling, is performed *a*s part of solving the EPR formula. To explain this claim, let us recall briefly how instantiation-based theorem provers work; in particular, let us recall the algorithm employed in the iProver solver used in our experiments.

The basic idea of instantiation-based reasoning is to interleave a smart generation of instances of first-order formulae with propositional reasoning, within an abstraction-refinement scheme. Given a set of first-order clauses S, iProver first produces a ground abstraction of S by mapping all variables into a distinguished constant, say $\bot$, obtaining a set of ground clauses $S\bot$. If $S\bot$ is unsatisfiable then so is S and we are done. Otherwise, we need to refine the abstraction by adding new instances of clauses, witnessing unsatisfiability at the ground level. Instances are generated by an inference system called DSInst-Gen [16]. iProver repeats this process until we obtain either (i) an unsatisfiable ground abstraction (this can be checked by any off-the-shelf SAT solver), or (ii) a saturated set of clauses (that is, no non-redundant inference is applicable), and in this case completeness of the calculus implies that S is satisfiable.

As a result of the iterative abstraction refinement procedure interleaving instantiation and SAT solving, iProver generates variables corresponding to circuit signals for a subset of bounds between 0 and n, and it does this lazily, per demand. Hence, in each iteration, a circuit signal might have fewer than $n+1$ copies. Thus with EPR encoding, and with an instantiation-based procedure for solving the EPR formulas, the BMC problem can (potentially) be solved via solving much smaller propositional SAT instances.

To further take advantage of the EPR-based approach to abstraction refinement, in this paper we propose an unsat-core based iterative scheme that allows us to optimize the implicit on-demand unrolling

discussed above, and as a result the SAT problems generated and solved as part of the Inst-Gen algorithm shrink even further. For EPR-based BMC, this usually implies a speedup in verification and, more importantly, much deeper unrolling bounds can be verified. For induction, in addition to the speedup, this can decrease the unrolling depths required to achieve full proofs. In the context of SAT-based BMC, interpolation [18] and PDR [19], at the bit-level, some of the ideas in this paper are similar to the counter-example and proof-based abstraction refinement procedures in [20, 17, 21, 22, 23, 24], although our algorithm differs significantly from these approaches in technical detail. In abstraction-refinement for SAT-BMC, the abstract model is usually fully unrolled to a current bound k. In our approach each part of the transition relation can be added/removed separately at non-consecutive bounds, guided by the abstraction-refinement process. Our experiments show that usually instantiating different small parts of the transition relation at previous bounds is enough for solving the problem at the current bound. In the EPR framework this can be conveniently achieved by adding/removing relevant next state atoms during the abstraction-refinement process. The EPR framework also gives natural flexibility of how the transition relation is partitioned.

The paper is organized as follows. In Section 2, we quickly recall the BMC1 algorithm form [11] and discuss its extension with property lemmas. In Section 3, we introduce a basic version of EPR-based k-induction, and discuss in Section 4 how to adapt the encoding to always remain within the EPR fragment. Unsat core and model based refinements of BMC1 are introduced in Section 5, and are extended to k-induction in Section 6. Quantified invariants are discussed in Section 7. Implementation and experimental results are reported in Sections 8 and 9, respectively. We conclude in Section 10.

2 BMC1

In this paper we consider verification of safety properties of state transition systems represented using (many-sorted) first-order logic. In our formalisation we use general first-order logic and later show that in the case of bounded domains this formalisation can always be reduced to the effectively propositional fragment (EPR) of first-order logic.

A state transition systems and a verification condition can be represented using first-order logic by three formulas:

- the initial condition $F_{init}(S)$, that should hold in the starting state;
- the transition relation $F_{next}(S,S')$, that describes the changes of the state during execution;
- the target property $F_p(S)$.

In order to succinctly represent unrolling of the system we introduce auxiliary predicates $I(S)$, $N(S,S')$ and $p(S)$ which represent initial states, the transition relation and the target property.

The bounded model checking representation of the system in first-order logic (BMC1) [11] consists of the following formulas:

$$\forall S\ [I(S) \rightarrow F_{init}(S)] \tag{1}$$

$$\forall S,S'\ [N(S,S') \rightarrow F_{next}(S,S')] \tag{2}$$

$$\forall S\ [F_p(S) \rightarrow p(S)]. \tag{3}$$

BMC1 unrolling of the system to a bound n can be represented as

$$I(s_0) \wedge N(s_0,s_1) \wedge \ldots \wedge N(s_{n-1},s_n) \wedge \neg p(s_n), \tag{4}$$

where $s_0,\ldots,s_n$ are constants representing states. If formula (4) is unsatisfiable (together with the BMC1 representation of the system), then the property p holds at all states reachable from the initial state in

exactly n steps. The BMC1 procedure then starts from $n = 0$ and increments n until a state where either i) (4) is satisfiable and therefore the property is falsified, or ii) the diameter of the system, or more generally completeness threshold, is reached [2] and therefore the property holds for all states reachable from an initial state. Let us note that the BMC1 procedure does not change the representation of the system and at each step n only modifies formula (4) by adding a single atom $N(s_{n-1}, s_n)$ and replacing $\neg p(s_{n-1})$ with $\neg p(s_n)$.

2.1 BMC1 with property lemmas.

One modification of the BMC1 procedure is adding property lemmas that we already proved in the previous steps. For this, first the representation of the system should be modified: we need to replace (3) with

$$\forall S\ [F_p(S) \leftrightarrow p(S)]. \tag{5}$$

Now BMC1 unrolling with property lemmas to a bound n can be represented by:

$$I(s_0) \wedge p(s_0) \wedge N(s_0, s_1) \wedge \ldots \wedge p(s_{n-1}) \wedge N(s_{n-1}, s_n) \wedge \neg p(s_n). \tag{6}$$

We have that if (6) together with the system representation (1), (2), (5) is unsatisfiable, then $p(s_n)$ holds and we can add $p(s_n)$ as a lemma in the following iterations. The main difference with the propositional case [2] is that due to expressivity of first-order logic lemmas can be represented by a single atom at each stage.

3 EPR-based k-induction

The k-induction is an approach to unbounded model checking that uses a variant of a mathematical induction over the lengths of the execution runs [6]. In essence, the approach can be adapted to the EPR-based model checking as follows. For a given n the following two formulas are tested for satisfiability together with the system representation (1), (2), (5).

Induction base:

$$I(s_0) \wedge p(s_0) \wedge N(s_0, s_1) \wedge \ldots \wedge N(s_{n-1}, s_n) \wedge \neg p(s_n). \tag{7}$$

Satisfiability in this case shows that there exists a run of length n, starting from the initial state, at the end of which p does not hold. This means that the hypothesis is disproved and the procedure terminates. If the base case is unsatisfiable, then p holds for any chain of states of length n, starting from the initial states.

Induction step:

$$p(s_0) \wedge N(s_0, s_1) \wedge \ldots \wedge p(s_n) \wedge N(s_n, s_{n+1}) \wedge \neg p(s_{n+1}). \tag{8}$$

Unsatisfiability in this case means that, starting with n subsequent states where p holds, the next state will be a state where p also holds. Together with the induction base this completes mathematical induction: p holds in every state reachable from the initial one.

If the formula corresponding to the induction step is satisfiable, this means that we fail to prove the induction hypothesis. Then, we increase n and return to the induction base.

3.1 Complete k-induction

In general this version of k-induction is not complete: it is possible that the induction step will fail for any n even if the property holds for all reachable states. In order to overcome this incompleteness one can add formulas stating that different states in the induction sequence are not equivalent (see, e.g., [2]

for the propositional case). We can define this in first-order logic as follows. Let Σ_B denote the *base signature* which consists of all predicates that are used to represent basic components of the system such as bit-vectors and memories. For each predicate $P(S,\bar{x}) \in \Sigma_B$ we introduce a binary predicate $Noneq_p(S,S')$ and axiomatise that two states are non-equivalent wrt. P as follows:

$$\forall S,S'\ Noneq_p(S,S') \rightarrow \exists \bar{x}[P(S,\bar{x}) \leftrightarrow \neg P(S',\bar{x})]. \tag{9}$$

Then, define when states are non-equivalent wrt. basic signature as

$$\forall S,S'\ Noneq_B(S,S') \rightarrow \bigvee_{p\in\Sigma_B} Noneq_p(S,S'). \tag{10}$$

We assume formulas used in the system description are Skolemized and non-constant functions are replaced by predicates as described in Section 4 and added to the basic signature.

During the unrolling up to a bound n we add atomic formulas stating that visited states are pairwise non-equivalent

$$Noneq_B(s_i,s_j) \text{ for } 1 \leq i < j \leq n. \tag{11}$$

This ensures that k-induction will always terminate in the case of finite state systems.

4 Reducing to the EPR in the case of functions with finite ranges

In some cases Skolemization can introduce functions with non-zero arity, resulting in formulas outside of the EPR fragment. Although our approach is valid for general first-order logic, for efficiency reasons it is desirable to stay within the decidable EPR fragment. Below we summarise how we translate non-EPR formulas containing function symbols of non-zero arity into the EPR fragment, in the case of functions with finite ranges.

Consider a function $f(\bar{x})$ with a finite range. Then, we can replace such a function by a predicate $P_f(\bar{x},y)$ which represents the graph of f. A *function elimination transformation* can be defined as follows:

$$F[f(\bar{t})] \Rightarrow_{felim} P_f(\bar{t},y) \rightarrow F[y]. \tag{12}$$

It can be shown (following [25]) that F is satisfiable if and only if (12) together with the domain axiom for P_f:

$$\bigvee_{v\in D(f)} P_f(\bar{x},v) \tag{13}$$

is satisfiable, where $D(f)$ is the value set of f. Note that values of f can be symbolic, represented by uninterpreted constants. Using the function elimination transformation we can eliminate all functions with finite ranges, replacing them with the corresponding predicates. This transformation is suitable when the value domain is reasonably small such as e.g. a range of bit-vector indexes. For large value domains such as fixed size bit-vectors, more efficient transformations are possible [11]. In applications such as hardware verification all domains are finite and therefore we can eliminate functions as described above or using more intricate transformations for large domains. There are also different clausification methods such as definition simplifications and unfolding that can be used to avoid introduction of Skolem functions in many cases [26].

5 Unsat Core and Model Guided BMC1 (UCM-BMC1)

In this section we describe how BMC1 can be guided by unsat cores and models. Our method is based on approximations of the transition relation.

5.1 Splitting transition relation

Consider a transition system represented by formulas (1), (2), (5). The formula representing the transition relation can be naturally written as a conjunction of weaker conditions (usually as a conjunction of clauses)

$$\forall S,S'\ [N(S,S') \rightarrow (F^1_{next}(S,S') \wedge \ldots \wedge F^m_{next}(S,S'))]. \tag{14}$$

In order to obtain a finer grained representation of the transition relation we replace the predicate N with a ternary predicate N_s where the additional argument represents transition relations corresponding to weaker conditions. Then, (14) can be rewritten as follows.

$$\begin{array}{c} \forall S,S'\ [N_s(1,S,S') \rightarrow F^1_{next}(S,S')] \\ \ldots \\ \forall S,S'\ [N_s(m,S,S') \rightarrow F^m_{next}(S,S')]. \end{array} \tag{15}$$

We assume that the first argument of N_s ranges over a finite domain $\{1,\ldots,m\}$ corresponding to *indexes of the transition conditions* $F^1_{next}(S,S'),\ldots,F^m_{next}(S,S')$. There is a natural flexibility of how the transition relation is partitioned. If all indexes are the same then we obtain the standard representation as in (2), whereas when indexes are different for each clause, we obtain a fine grained splitting. We can also get intermediate representations, e.g., merging indexes corresponding to the same latch.

Let us note that for any partition, (1), (15), (5) represent the same transition system as represented by (1), (2), (5), in particular we can define N using N_s as

$$\forall S,S'\ [N(S,S') \leftrightarrow \forall J\ N_s(J,S,S')].$$

In what follows, we will use $N(S,S')$ as a shorthand for $\forall J\ N_s(J,S,S')$. We call conjunction of formulas (1), (15), (5) as a *split representation of the transition system* and denote it by *STS*. The main idea behind splitting of the transition relation is that it allows one to instantiate different transition conditions at different bounds. The intuition is that for proving a property at a bound n it may be sufficient to instantiate only certain transition conditions at different bounds between 0 and n. In other words, the unrolling is partial, or rather, *discrete*.

A transition set $\mathscr{N}$ is a finite set of atoms of the form $N_s(t,s_i,s_{i+1})$, where t is either a variable or a constant in $\{1,\ldots,m\}$. We also use $\mathscr{N}$ to represent the conjunction of all atoms in $\mathscr{N}$. *A partial BMC1 unrolling w.r.t.* $\mathscr{N}$ *and a bound* n is represented by the following formula:

$$I(s_0) \wedge \mathscr{N} \wedge p(s_0) \wedge \ldots \wedge p(s_{n-1}) \wedge \neg p(s_n). \tag{16}$$

We assume that when $n=0$, (16) stands for $I(s_0) \wedge \neg p(s_0)$. We denote by $\mathscr{U}(\mathscr{N},n)$ the conjunction of (16) together with the split representation of the transition system (1), (15), (5).

A transition set $\mathscr{EN}_n$ and the corresponding unrolling $\mathscr{U}(\mathscr{EN}_n,n)$ are called *exhaustive* if $\mathscr{EN}_n$ consists of atoms $N_s(X,s_i,s_{i+1})$ for $0 \leq i < n$, where X is a universally quantified variable. Exhaustive unrollings directly simulate BMC1 unrollings with lemmas (6). Let us note that in the case of a partial BMC1 unrolling w.r.t. $\mathscr{N}$ we can safely remove the transition conditions in (15) with indexes outside of $\mathscr{N}$, without affecting satisfiability of the unrolling.

5.2 UCM-BMC1 procedure

The main ingredient of the UCM-BMC1 procedure is a method for extending (to the next bound) and expanding (at the same bound) transition sets based on satisfiability of the partial unrolling at the current stage. For this we use unsatisfiable cores and partial models. From now on, we assume that our formulas are in clausal form.

5.2.1 Unsat cores

A *ground unsatisfiable core (or just unsat core)* of an unsatisfiable set of clauses S, is a subset of ground instances of S which is also unsatisfiable. In practice, unsat cores are usually small, representing the reason for unsatisfiability. In the propositional case, unsat cores can be conveniently returned by modern SAT solvers [27, 28] which in turn can be used within instantiation procedures, such as Inst-Gen [16], to produce ground unsat cores for first-order problems. In many cases unsat cores can be assumed minimal, i.e., removing a clause from the core will result in a satisfiable set of clauses, but we do not impose this requirement here. For an unsatisfiable set of clauses S we will denote by S^{uc} one of its ground unsat cores. For an unsatisfiable unrolling $\mathscr{U}(\mathscr{N},n)$ we denote by $\mathscr{N}^{uc}$ the instances of $\mathscr{N}$ which are in the ground unsat core $\mathscr{U}(\mathscr{N},n)^{uc}$.

5.2.2 Partial models

Models correspond to satisfiable traces, or counter-examples. In our approach we over-approximate the transition system and therefore models can also represent spurious counter-examples which are then used to correct the current over-approximation. We do not require models to be complete and generally use partial models, or partial interpretations. Partial interpretations can be represented as sets of ground literals. Let us note that there are also more advanced model representations, e.g., based on cubes used in Z3 [29] or dismatching constraints [30, 31] used in iProver, that allow more compact representation of ground sets of literals. In the Inst-Gen framework in place of an explicit model representation we can enforce partial interpretations by fixing a literal selection in clauses using propositional assumptions, this was used in our implementation. For simplicity of the exposition we will consider representations of partial interpretations based on sets of ground literals. We say that a set of literals M is a partial model for a set of clauses S if $M \cup S$ is satisfiable. We assume that we are given a function *model* which for a satisfiable set of clauses S returns a (partial) model of S and is undefined otherwise. We will use satisfiability checks combining partial interpretations together with sets of clauses. There is a trade-off when considering different partial interpretations: the stronger the interpretations (containing more literals) the easier is reasoning with $M \cup S$, on the other hand weaker interpretations represent larger classes of counter-examples which can be corrected in one step.

Let us informally describe the UCM-BMC1 procedure. We over-approximate unrollings of the transition system by considering partial unrollings based on transition sets. In partial unrollings, different transition conditions are instantiated at different bounds. This has the advantage that the reasoning will involve only relevant transition conditions from (15), corresponding to the current transition set. We start with the empty transition set and expand it as required in order to eliminate spurious counterexamples. At each stage we check satisfiability of the partial unrolling with the current transition set. If the partial unrolling is unsatisfiable then there are no counter-examples reachable with even stronger exhaustive unrolling at the current bound and we can proceed to the bound extension phase. If, on the other hand, the partial unrolling is satisfiable then we consider a model M which is a candidate counter-example. We check M on the exhaustive unrolling at the current bound. If M is satisfiable together with the exhaustive unrolling then M can be extended to a real counter-example and we are done. Otherwise, we extract from the corresponding unsat core relevant instances of the transition relation which are sufficient to eliminate this counter-example. We expand the current transition set by adding obtained new instances of the transition relation and continue with satisfiability check. We expand the transition relation at the current bound until we find a real counter-example or show that there is no counter-example reachable at this bound. In the latter case, we go to the extension phase. In the extension phase we 1) refine the transition set by restricting it to transition relations used in proving unreachability of the counter-example at the current bound (these can be extracted from the unsat core of the latest satisfiability check), 2) extend

selected transition relations from the transition set to the next bound, 3) continue partial unrolling to next bound with the new transition set. We have flexibility in choosing which transition relations to extend to the next bound in 2) above with a natural choice of transition relations which involve the current bound and were used in proving unreachability of the counter-example at the current bound. Let us remark that the choice of which transition relations to extend to the next bound does not affect completeness of the procedure but is relevant to performance.

5.2.3 States

A state of the UCM-BMC1 procedure consists of

1. *STS* – a split representation of the transition system
2. n – the current unrolling bound,
3. $\mathscr{N}$ – the current transition set,
4. M – the current model or possibly the empty set, which will indicate that the model is undefined.

The UCM-BMC1 procedure is presented as Algorithm 1.

Theorem 1. *(soundness and completeness wrt. counter-examples) If UCM-BMC1 returns DISPROVED then there is a counter-example, if returns PROVED then property holds for all bounds. Moreover, if there is a counter-example then UCM-BMC1 will return DISPROVED after a finite number of steps.*

Proof. (Sketch) If the algorithm returns DISPROVED then either 1) already an initial state does not satisfy the property: $I(s_0) \wedge \neg p(s_0)$ is satisfiable (line 5) or 2) at bound n, exhaustive unrolling reaches a state that does not satisfy the property – $M \cup \mathscr{U}(\mathscr{E}\mathscr{N}_n, n)$ is satisfiable (line 16). In both cases a bad state is reachable from the initial states.

If the algorithm returns PROVED at bound n (line 37) then $\mathscr{U}(\mathscr{N}, n)$ is unsatisfiable. Moreover, since $\mathscr{N}^{uc}_{\max} = \emptyset$, $\mathscr{N}^{uc}$ does not contain s_n. Therefore $\mathscr{U}(\mathscr{N}, n)^{uc}$ can be split into two parts $\mathscr{U}(\mathscr{N}, n)^{uc} = U_1 \wedge U_2$ where U_1 does not contain s_n and U_2 is either $\top$ or $\neg p(s_n)$. In both cases we have $U_1 \models \forall x(p(x))$ and for any exhaustive unrolling beyond n we have $U_1 \subseteq \mathscr{U}(\mathscr{E}\mathscr{N}_n, n)$, which implies that the property holds for all reachable states.

In order to show the last part of the theorem, first we note that when the algorithm extends the bound from n to $n+1$ (line 41) then the property holds for all states reachable from the initial states in n or less steps. Indeed, at this stage we have $\mathscr{U}(\mathscr{N}, n)$ is unsatisfiable and since $\mathscr{N} \subseteq \mathscr{E}\mathscr{N}$, the exhaustive unrolling $\mathscr{U}(\mathscr{E}\mathscr{N}, n)$ is also unsatisfiable from which the claim follows. To complete the proof it remains to show that for any bound n the algorithm either returns PROVED, DISPROVED or extends the bound to $n+1$. For this it is sufficient to show that at any bound n, the Expansion phase is not called infinitely often (from line 27). First note that expanding $\mathscr{N}$ (line 18) is always proper. Indeed, at this stage (before line 18) we have $M \models \mathscr{N}$ (due to line 26) but $M \not\models \mathscr{N} \cup \mathscr{E}\mathscr{N}^{uc}_n$. Since there are only a finite number of ground instances of transition predicates, at a fixed bound, there can be only a finite number of proper expansions (visits to line 18) and therefore the Expansion phase can be called only a finite number of times before the Extension phase is called or DISPROVED is returned. □

Let us make a final remark that we can also split initial predicate I and the property p in a similar way as we did with the transition relation.

6 Unsat Core and Model Guided k-induction

In this Section we extend UCM-BMC1 with k-induction. As we saw in Section 3 k-induction can be split into two independent parts: the *Induction base* and the *Induction step*. For solving the *Induction*

Algorithm 1 UCM-BMC1

```
1: function UCM-BMC1
2:     Initial phase
3:         n ← 0, 𝒩 ← ∅, I ← ∅
4:         if ISSAT(𝒰(𝒩, n)) then
5:             return DISPROVED
6:         else
7:             n ← 1
8:             M ← model(STS ∧ I(s_0))
9:             goto Expansion phase
10:        end if
11:
12:    Expansion phase
13:        // Check the current model (trace)
14:        // on exhaustive unrolling.
15:        if ISSAT(M ∪ 𝒰(ℰ𝒩_n, n)) then
16:            return DISPROVED
17:        else
18:            𝒩 ← 𝒩 ∪ ℰ𝒩^uc_n
19:            goto Sat checking phase
20:        end if
21:
22:    Sat checking phase
23:        // Check satisfiability with the collected
24:        // transition predicates.
25:        if ISSAT(𝒰(𝒩, n)) then
26:            M ← model(𝒰(𝒩, n))
27:            goto Expansion phase
28:        else
29:            goto Extension phase
30:        end if
31:
32:    Extension phase
33:        // Extend the transition set to the next bound.
34:        // At this stage 𝒰(𝒩, n) is unsatisfiable.
35:        𝒩^uc_max ← {N_s(j, s_{n−1}, s_n) | N_s(j, s_{n−1}, s_n) ∈ 𝒩^uc}
36:        if 𝒩^uc_max = ∅ then
37:            return PROVED
38:        else
39:            𝒩 ← 𝒩^uc ∪ {N_s(j, s_n, s_{n+1}) |
40:                          N_s(j, s_{n−1}, s_n) ∈ 𝒩^uc}
41:            n ← n + 1
42:            goto Sat checking phase
43:        end if
44:
45: end function
```

base we can directly apply the UCM-BMC1 Algorithm 1. We present a separate algorithm for the *Induction step*. The main difference between two parts is that at almost all bounds, except possibly the last bound, the *Induction base* is unsatisfiable whereas the *Induction step* is satisfiable. Instead of over-approximating the transition relation, we attempt to extend a model at the current bound in the *Induction step* into a model at the next bound. Consider a partial interpretation M and a set of clauses S. Define a *consistent sub-model of M wrt S* to be a subset M' of M such that $M' \cup S$ is satisfiable, if such a subset exists, and $\emptyset$ otherwise. In practice, we aim at maximal or near maximal subset of our model satisfiable with the current *Induction step* unrolling, which can be achieved by iteratively removing literals from M which are in unsat cores of $M \cup S$. We assume that we have a function $M^{sat}(M,S)$ which returns a consistent sub-model of M wrt. S.

Another modification we make is reversing the unrolling (8) of the *Induction step* [32].

$$\neg p(s_0) \wedge N(s_1,s_0) \wedge p(s_1) \wedge \ldots \wedge p(s_n) \wedge N(s_{n+1},s_n) \wedge p(s_{n+1}). \tag{17}$$

We denote by $\mathscr{U}^r(\mathscr{E}\mathscr{N},n)$ the reverse unrolling (17) together with the representation of the transition system (15), (5), and for the complete version also (9), (10), (11). It is easy to see that the reverse unrolling (17) is logically equivalent to the direct unrolling (8). The reason behind considering the reverse unrolling is that in the direct unrolling we flip the value of $p(s_n)$ from positive to negative at each extension. This can adversely affect incremental reasoning which benefits from reasoning at previous stages. Moreover models can not be directly extended from the current stage to the next due to the flip in the value of $p(s_n)$. In the reverse unrolling we always keep $\neg p(s_0)$ at state s_0 and at each extension we add $p(s_{n+1})$ only positively. In practice, we also apply reverse unrolling in the UCM-BMC1 algorithm.

The Algorithm 2 describes *Induction step* of UCM-k-ind-step. The full UCM-k-ind is the standard alternation between UCM-BMC1 as UCM-kind-base and UCM-k-ind-step as described in Section 3.

Algorithm 2 UCM-k-ind (induction step)

```
1:  function UCM-K-IND-STEP
2:      n ← 0, M ← ∅
3:      while n ≤ MaxBound do
4:          if ISSAT(M ∪ U^r(E N, n)) then
5:              M ← model(M ∪ U^r(E N, n))
6:              n ← n + 1
7:          else
8:              if M = ∅ then
9:                  return PROVED
10:             else
11:                 M ← M^sat(M, U^r(E N, n))
12:             end if
13:         end if
14:     end while
15:     return induction step fails upto MaxBound
16: end function
```

7 Quantified invariants

One of the advantages of using first-order logic in verification is that it allows one to represent universally quantified invariants. During the run of UCM-BMC1 and UCM-k-ind it is possible to generate universally quantified invariants from the unsatisfiable cores (line 17 in Algorithm 1 and line 10 of Algorithm 2). Consider line 17 in Algorithm 1. At this stage we have $M \cup \mathscr{U}(\mathscr{E}\mathscr{N}_n,n)$ is unsatisfiable. We extract an unsat core from $M \cup \mathscr{U}(\mathscr{E}\mathscr{N}_n,n)$. Let M^{uc} be the literals in the model M from the unsat core and, as before, $\mathscr{N}^{uc}$ be the instances of the transition relation from the unsat core. Then, the disjunction of the complements of all literals from M^{uc} and $\mathscr{N}^{uc}$ will be a ground lemma implied by the split representation of the transition system. Let us denote this lemma as $lem_{gr}(M^{uc},\mathscr{N}^{uc})$. The ground lemma $lem_{gr}(M^{uc},\mathscr{N}^{uc})$ will contain some state constants. Since the description of our transition system does not contain state constants we can replace them by fresh variables obtaining a universally quantified invariant which holds at all (sequences of) states and is implied by our representation of the transition system. We denote such an invariant as $inv(M^{uc},\mathscr{N}^{uc})$. Let us consider an example.

Let $M^{uc} = \{a(s_1), \neg b(s_1), c(s_2), \neg d(s_3)\}$ and $\mathscr{N}^{uc} = \{N(1,s_1,s_2), N(2,s_2,s_3)\}$. Then the corresponding ground lemma is

$$lem_{gr}(M^{uc},\mathscr{N}^{uc}) = \neg N(1,s_1,s_2) \vee \neg N(2,s_2,s_3) \vee \neg a(s_1) \vee b(s_1) \vee \neg c(s_2) \vee d(s_3)$$

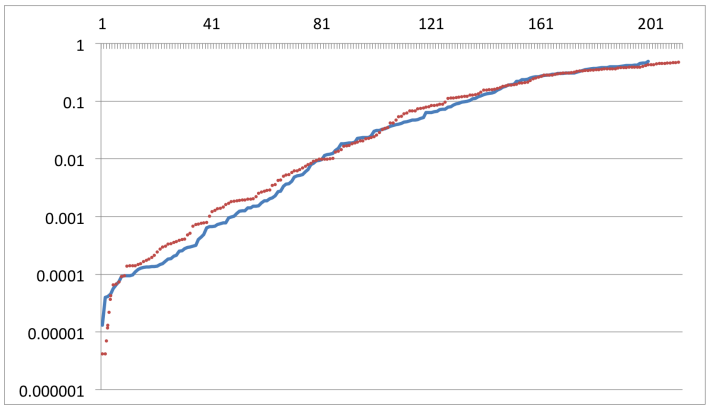

Figure 1: The ratio between the size of the transition relation used by UCM over the full relation. UCM-BMC1 with property lemmas (red dotted line) and UCM k-induction (blue solid line).

and the corresponding quantified invariant:

$$inv(M^{uc},\mathscr{N}^{uc}) = \forall S_1,S_2,S_3[\neg N(1,S_1,S_2) \vee \neg N(2,S_2,S_3) \vee \neg a(S_1) \vee b(S_1) \vee \neg c(S_2) \vee d(S_3)].$$

Such quantified invariants can be added to the system description and can be shared between UCM-BMC1 and UCM-k-ind.

8 Implementation

We implemented the approach described above in the iProver system [16]. iProver is a general purpose theorem prover for first-order logic which incorporates SAT solvers at its core (currently, MiniSAT [33] and optionally PicoSAT [34]). iProver is particularly efficient in the EPR fragment [35].

We implemented UCM-based algorithms, presented as Algorithms 1 and 2, in out system. We also implemented the basic versions of BMC1 and k-induction, as described in Sections 2 and 3.

One of the main features of iProver which we used to implement BMC and k-induction is incrementality wrt. *assumptions*, which in turn is based on incrementality of SAT solvers. Assumptions can be used to assert or retract relevant clauses at each stage of the model checking process, by extending clauses with a fresh literal and either assuming the negation of the literal to assert the clause, or assuming the literal itself to retract the clause. We use the assumption literals $bound_n$ for every n which means that the n'th iteration of the incremental model checking is performed. We also represent a partial model as a set of assumption literals.

In our implementation we use several optimisations that significantly increase performance of the system. One of such optimisations is based on the following observation. If the evaluation of a partial unrolling is satisfiable, the model is used for the exhaustive unrolling check in the expansion phase. Then, if that model is unsatisfiable, the unsat core is added to the partial unrolling (lines 25–27 and 12–20 of Algorithm 1). In practice this scenario repeats several times: the model appears to be too restrictive for the exhaustive unrolling, then at the next iteration slightly modified model is used. We collapse this loop, adjusting partial model in the expansion phase by removing from it all assumptions that are contained in the unsat core and continue with the expansion phase to expand the transition set with several unsat cores.

problem	HWMCC	iProver	method
6s268	129	130	UCM
6s279	14	15	BMC1
6s280r	15	16	both
6s339rb19	91	125	UCM
6s393r[2]	30	15	BMC1
6s402rb0342[2]	14	15	UCM
6s516r	15	16	both
6s517rb0	14	15	both
bobsmcodic	19	20	both

Table 1: Deep Bounds problems where iProver outperforms systems in HWMCC-14

iProver accepts problems in the TPTP first-order format [36]. AIGER [37] is a hardware description language which contains only AND-gates (with possible negations) and latches to describe the state change. A safety verification problem in the AIGER format contains a description of a circuit and a Boolean property that should hold in all computational states. In iProver an AIGER problem is transformed into the EPR verification problem and is passed to the solver. We apply several optimisation techniques at the AIGER level including definition merging, constant propagation and non-growing inlining similar to [26]. We used an AIGER parser provided by AIGER tools [37].

9 Evaluation

For evaluation we used hardware verification benchmarks from HWMCC-14 competition in the single safety property track and deep bounds track. These problems are represented at the bit-level rather than at the word-level, which we believe largely limits the advantages of the EPR-based model checking. For experiments we used 15 min timeout, the same as in the competition. Experiments were performed on the machine with Intel Xeon L5410 2.33 GHz processor and 12 Gb of RAM under Linux.

iProver performed well on the deep bounds track (79 problems), reaching strictly higher bounds than any other system from the competition on 9 problems. iProver also proved validity of a deep bound problem 6s393r and non-validity of 6s402rb0342, which were not previously known. Table 1 shows the problems from Deep Bounds track of the HWMCC-14 for which iProver reached larger bounds than the best bounds reached during the competition. Let us note that for a number of problems UCM-BMC1 was essential to reach deeper bounds. On the problems from safety property track (230 problems), iProver solved 50 problems, 28 SAT and 22 UNSAT. UCM-BMC1 alone can prove UNSAT (line 37 of Algorithm 1) this happens on 11 problems in these benchmarks.

Figure 1 shows a comparison between the size of the transition relation used by the UCM procedure and the full relation on the HWMCC-14 benchmarks. Our abstraction-refinement approach uses under 10% of the transitional relation in 2/3 of cases, therefore we can conclude that on most problems only small parts of the transition relation are involved in our discrete unrollings.

10 Conclusion

In this paper we presented an EPR-based BMC and *k*-induction with counterexample guided abstraction refinement. We implemented our approach in a first-order theorem prover iProver and evaluated our implementation over hardware verification benchmarks from the HWMCC'14 competition. The deep

[2]Problem 6s393r was PROVED using UCM-k-ind and 6s402rb0342 was DISPROVED by iProver.

bounds track at HWMCC represents the most challenging set of publicly available real-life HW verification benchmarks. On these benchmarks, iProver was able to improve bounds on 7 problems and fully solve two additional problems, in comparison, out of 62 deep bounds benchmarks that were used both in 2013 and 2014 competitions, the solvers participating at HWMCC'14 were able to collectively improve the reached bounds on 21 problems. We have shown experimentally that only small parts of the transition relation are involved in our discrete unrollings and we believe other SAT and SMT based verification methods can also benefit from such discrete unrollings. As a by-product of this research, we are also translating AIGER benchmarks into first-order TPTP benchmarks which can be used to evaluate first-order theorem provers on hardware verification problems. As a future work we are planning to compare and combine our approach with other property directed methods such as PDR/IC3 [19] and interpolation [18, 22, 29].

References

[1] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in *TACAS'99*, 1999, pp. 193–207.

[2] A. Biere, "Bounded model checking," in *Handbook of Satisfiability*. IOS Press, 2009, pp. 457–481.

[3] R. E. Bryant, S. K. Lahiri, and S. A. Seshia, "Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions," in *CAV*, 2002, pp. 78–92.

[4] P. Manolios, S. K. Srinivasan, and D. Vroon, "Automatic memory reductions for rtl model verification," in *ICCAD*, 2006, pp. 786–793.

[5] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-Guided Abstraction Refinement," in *CAV*, 2000, pp. 154–169.

[6] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a sat-solver," in *FMCAD'00*, 2000, pp. 108–125.

[7] A. F. Donaldson, L. Haller, D. Kroening, and P. Rümmer, "Software verification using k-induction," in *SAS*, 2011, pp. 351–368.

[8] A. Biere and D. Kröning, "Sat-based model checking," in *Handbook of Model Checking*, E. Clarke, T. Henzinger, and H. Veith, Eds. IOS Press, 2015, pp. 457–481.

[9] J. A. N. Pérez and A. Voronkov, "Encodings of bounded LTL model checking in effectively propositional logic," in *CADE-21*, 2007, pp. 346–361.

[10] M. Emmer, Z. Khasidashvili, K. Korovin, and A. Voronkov, "Encoding industrial hardware verification problems into effectively propositional logic," in *FMCAD*, 2010, pp. 137–144.

[11] M. Emmer, Z. Khasidashvili, K. Korovin, C. Sticksel, and A. Voronkov, "EPR-based bounded model checking at word level," in *IJCAR*, 2012, pp. 210–224.

[12] G. Kovásznai, A. Fröhlich, and A. Biere, "BV2EPR: A Tool for Polynomially Translating Quantifier-Free Bit-Vector Formulas into EPR," in *CADE*, 2013, pp. 443–449.

[13] G. Kovásznai, A. Fröhlich, and A. Biere, "On the complexity of fixed-size bit-vector logics with binary encoded bit-width," in *SMT'12*, 2012, pp. 44–56.

[14] N. Dershowitz, Z. Hanna, and J. Katz, "Bounded model checking with QBF," in *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005*, 2005, pp. 408–414.

[15] T. Jussila and A. Biere, "Compressing BMC encodings with QBF," *Electr. Notes Theor. Comput. Sci.*, vol. 174, no. 3, pp. 45–56, 2007.

[16] K. Korovin, "Inst-Gen - a modular approach to instantiation-based automated reasoning," in *Programming Logics*, 2013, pp. 239–270.

[17] A. Gupta and O. Strichman, "Abstraction refinement for bounded model checking," in *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, 2005, pp. 112–124.

[18] K. L. McMillan, "Interpolation and sat-based model checking," in *CAV*, 2003, pp. 1–13.

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

[20] K. L. McMillan and N. Amla, "Automatic abstraction without counterexamples," in *TACAS*, 2003, pp. 2 – 17.

[21] A. Gupta, M. Ganai, Z. Yang, and P. Ashar, "Iterative abstraction using sat-based bmc with proof analysis," in *ICCAD*, 2003, pp. 416 – 423.

[22] N. Amla and K. L. McMillan, "A hybrid of counterexample-based and proof-based abstraction," in *FMCAD*, 2004, pp. 260 – 274.

[23] N. Een, A. Mishchenko, and N. Amla, "A single-instance incremental sat formulation of proof- and counterexample-based abstraction," in *FMCAD*, 2010, pp. 181 – 188.

[24] N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *FMCAD*, 2011, pp. 125–134.

[25] P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli, "Computing finite models by reduction to function-free clause logic," *J. Applied Logic*, vol. 7, no. 1, pp. 58–74, 2009.

[26] K. Hoder, Z. Khasidashvili, K. Korovin, and A. Voronkov, "Preprocessing techniques for first-order clausification," in *FMCAD'12*, 2012, pp. 44–51.

[27] L. Zhang and S. Malik, "Extracting small unsatisfiable cores from unsatisfiable boolean formula," in *SAT03*, 2011.

[28] E. Goldberg and Y. Novikov, "Verification of proofs of unsatisfiability for cnf formulas," in *DATE03*, 2003, pp. 10 886–10 891.

[29] N. Bjørner, A. Gurfinkel, K. Korovin, and O. Lahav, "Instantiations, zippers and EPR interpolation," in *LPAR'13, short papers*, ser. EPiC Series, vol. 26, 2014, pp. 35–41.

[30] K. Korovin and C. Sticksel, "A note on model representation and proof extraction in the first-order instantiation-based calculus Inst-Gen," in *(ARW'12)*, 2012, pp. 11–12. [Online]. Available: http://arw2012.cs.man.ac.uk/

[31] C. G. Fermüller and R. Pichler, "Model Representation via Contexts and Implicit Generalizations," in *CADE 20*, 2005, pp. 409–423.

[32] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electr. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.

[33] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. of the 6th International Conference SAT'03*, ser. LNCS, vol. 2919. Springer, 2004, pp. 502–518.

[34] A. Biere, "Picosat essentials," *JSAT*, vol. 4, no. 2-4, pp. 75–97, 2008.

[35] G. Sutcliffe, "The CADE-24 automated theorem proving system competition - CASC-24," *AI Com.*, vol. 27, no. 4, pp. 405–416, 2014.

[36] ——, "The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0," *JAR*, vol. 43, no. 4, pp. 337–362, 2009.

[37] A. Biere, K. Heljanko, and S. Wieringa, "AIGER 1.9 and beyond," 2011, http://fmv.jku.at/aiger/.