Beyond Safety: Customized SAT-based Model Checking

Malay K Ganai, Aarti Gupta and Pranav Ashar[1]
NEC Laboratories America, Princeton, NJ USA 08540
{malay | agupta}@nec-labs.com

ABSTRACT

Model checking of safety properties has taken a significant lead over non-safety properties in recent years. To bridge the gap, we propose dedicated SAT-based model checking algorithms for properties beyond safety. Previous bounded model checking (BMC) approaches have relied on either converting such properties to safety checking, or finding proofs by deriving termination criteria using loop-free path analysis. Instead, our approach uses a customized SAT-based formulation for bounded model checking of non-safety properties, and determines the completeness bounds for liveness using unbounded SAT-based analysis. Our main contributions are: 1) Customized property translations for LTL formulas for BMC, with novel features that utilize partitioning, learning, and incremental formulation. Customized translations not only improve the BMC performance significantly in comparison to standard monolithic LTL translations, but also allow efficient derivation and use of completeness bounds. Though we discuss the translation schemas for liveness, they can be easily extended to handle other LTL properties as well. 2) Customized formulations for determining completeness bounds for liveness using SAT-based unbounded model checking (UMC) rather than using loop-free path analysis. These formulations comprise greatest fixed-point and least fixed-point computations to efficiently handle nested properties using SAT-based quantification approaches. We show the effectiveness of our overall approach for checking liveness on public benchmarks and several industry designs.

Categories and Subject Descriptors

B.6.3 [**Design Aids**]: Verification

General Terms

Algorithms, Verification

Keywords

Formal verification, bounded model checking, unbounded model checking, LTL, liveness, SAT, circuit cofactoring

1. INTRODUCTION

A liveness property [1] expresses that eventually "somethinggood will take place" along an execution path. In contrast, a safety property expresses that "something bad will take place" during system execution.

Safety properties have generated a lot of interest among researchers due to their relative importance in practice. Recently, there have been several effective SAT-based techniques – BMC [2, 3], induction [4, 5], unbounded model checking (UMC) [6-9], and proof-based abstraction [10, 11] – that can handle safety properties very efficiently compared to BDD-based approaches like symbolic model checking [12, 13]. Though many of these techniques can be applied to checking all properties in principle, there has been a general lack of efforts dedicated to liveness (barring a few described later).

As reported in a recent workshop [14], one school of thought believes that rigorous reasoning about liveness is tedious and therefore, it is not worth the effort given finite resource for verification. Another school of thought, along similar lines, believes that liveness without a bound is not useful and bounded liveness can be reduced to a safety property; therefore, we should use safety checking only. We believe, like an alternative school of thought, that liveness is important and that it too deserves dedicated model checking techniques. We list some of the main reasons cited:

- If we restrict verification to only safety checking, it would be impossible to detect subtle design errors that prevent some behavior from happening eventually.
- Translation of bounded liveness to safety is expensive especially when the bound is large. Moreover, designers may not always provide a safe bound.
- Fairness constraints are used to capture the interaction of the design with the environment. Since formal verification is usually applied at the block level, fairness is needed to model the environment correctly. Therefore, liveness with fairness is important to capture subtle system behaviors.
- Model checking for all LTL properties can be reduced to checking of a fairness constraint.

In general, liveness is handled as a standard LTL property, by converting the negated formula to a Buchi automaton and checking for language emptiness of its product with the design model [15]. Such a general translation may not be very effective, and many improved translations [16-18] have been proposed. SAT-based BMC [2] handles liveness as part of a standard translation of an LTL formula to a propositional satisfiability problem where the challenge is to find a state loop. BMC is incomplete without a completeness bound [2], and generally it is difficult to obtain such a bound [19]. However for safety checking, the completeness has been addressed by several other methods: induction [4, 5], SAT-based UMC [6-9], and proof-based abstraction [10, 11]. This inspired some researchers [20] to reduce the liveness property to a safety property and use the complete safety checking algorithms. However, this translation is based on a state recording approach that doubles the number of state bits in the design. Such a translation adds significant overhead to safety checking algorithms and is therefore, not very

[1]The author is now at Real Intent Inc.

DAC 2005, June 13-17, 2005, Anaheim, California, USA.

practical. In another recent work [21], authors reason about a model composed with Buchi automata obtained from translation of the negated LTL formula, and provide completeness bound for a fairness constraint based on loop-free path analysis. Such bounds may be quite large in practice, and are therefore not very useful.

In this work, we describe SAT-based techniques geared towards verifying liveness properties of the form $F(q)$ and $G(p \rightarrow F(q))$ with fairness $B=\{f^1,...,f^b\}$. These techniques can be extended to handle all LTL properties. We use customized SAT-based translations for bounded model checking of non-safety properties, and determine the completeness bounds for liveness using SAT-based unbounded analysis. We show the effectiveness of our approach for checking liveness on public benchmarks and several industry designs.

Outline In Section 2 we give relevant background on liveness, SAT-based BMC and UMC; in Section 3 we give the motivation; in Section 4 we describe our contributions; in Section 5 we discuss our experiments on several case studies and conclude in Section 6.

2. BACKGROUND

2.1 Bounded Model Checking: Liveness

In BMC, the specifications are expressed in LTL (Linear Temporal Logic). Given a Kripke structure M, an LTL formula f, and a bound k, the translation task in BMC is to construct a propositional formula $[M, f]_k$ such that the formula is satisfiable if and only if there exists a witness of length k [2]. $[M, f]_k$ is defined as follows:

$[M, f]_k = [M]_k \wedge [f]_k$, where

$[M]_k = I(s_0) \wedge \wedge_{0 \le i < k} T(s_i, s_{i+1})$ :: initial and circuit constraints

$[f]_k = ((\neg L_k \wedge [f]_k^0) \vee (\vee_{0 \le l \le k} ({}_lL_k \wedge {}_l[f]_k^0)))$:: property translation

${}_lL_k = T(s_k, s_l)$ :: loop transition from state s_k to s_l $(l \le k)$

$L_k = \vee_{0 \le l \le k} {}_lL_k$ :: disjunction of loops of length up to k

The detailed definitions for $[f]_k^0$ and ${}_l[f]_k^0$ are given in [2, 3]. Note, $s_0,..,s_k$ denotes a finite $(k+1)$-length sequence of states.

Liveness properties $F(\neg q)$, $G(p \rightarrow F(\neg q))$ in presence of fairness constraints $B = \{f^1,...,f^b\}$ expressed as $G(\wedge_{1 \le r \le b} F(f^r))$ are:

$$G(\wedge_{1 \le r \le b} F(f^r)) \rightarrow F(\neg q)$$

$$G(\wedge_{1 \le r \le b} F(f^r)) \rightarrow G(p \rightarrow F(\neg q))$$

The standard LTL translations [2, 3] for the negation of the above properties where p, q are Boolean propositional are as follows:

$[\neg(G(\wedge_{1 \le r \le b} F(f^r)) \rightarrow F(\neg q))]_k = [G(q \wedge (\wedge_{1 \le r \le b} F(f^r)))]_k$

$= [\vee_{0 \le l \le k} ({}_lL_k \wedge (\wedge_{1 \le r \le b} (\vee_{l \le j \le k} f_j^r)) \wedge (\wedge_{0 \le i \le k} q_i))]_k$

$[\neg(G(\wedge_{1 \le r \le b} F(f^r)) \rightarrow G(p \rightarrow F(\neg q)))]_k$

$= [F(p \wedge G(q \wedge (\wedge_{1 \le r \le b} F(f^r))))]_k$

$= [\vee_{0 \le l \le k} ({}_lL_k \wedge (\vee_{0 \le t \le k} p_t \wedge (\wedge_{min(j,l) \le i \le k} q_i) \wedge (\wedge_{1 \le r \le b} (\vee_{l \le j \le k} f_j^r))))]_k$

A counter-example to a liveness property $G(F(f)) \rightarrow G(p \rightarrow F(\neg q))$ in presence of a fairness constraint f is shown in Figure 1. Note, the fairness requirement is that f should hold at least once between the looping states i.e. s_l and s_k. The standard translation for these properties is monolithic, i.e., the entire propositional formula is generated for a given k which is then checked for satisfiability using a standard SAT solver. In practice, the search for longer witnesses is conducted by incrementing the bound k.

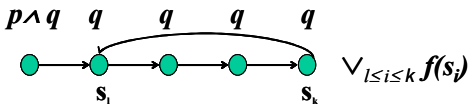

Figure 1. Debug trace for $G(p \rightarrow F(\neg q))$ with fairness f

2.2 SAT-based Unbounded Model Checking

SAT-based UMC techniques [6-9] can be used to obtain completeness bounds for safety properties. The approach is based on exact computation of a least fixed-point operation (*lfp*) using a SAT-based existential quantification procedure *SAT-EQ*. The procedure *SAT-EQ(f, A, B)* takes a function *f(A, B)* with support variables from sets A and B and returns $C=\exists_B f(A,B)$ after quantifying the variables in set B. A SAT solver is called repeatedly on the constrained problem $f=1 \wedge C=0$ where C represents the set of solutions captured at some enumeration step. The state solution set can be obtained using several approaches such as: BDD-based quantification [6], cube-wise enumeration [7], or circuit cofactor-based enumeration [9]. This state solution set is then used as a blocking constraint to prevent enumeration of the same solution again. Successive enumeration stops when there are no more solutions and C represents $\exists_B f(A,B)$ exactly.

A standard *lfp* computation using *SAT-EQ* for LTL operator *F(p)*, where p is a Boolean proposition, is shown in Figure 2. It uses a pre-image computation, $Pre\text{-}Image(X)=\exists_{U,Y} \delta(X,Y,U) \wedge T(Y)$ where $X,Y,$ and U represent the set of current state, next state, and primary input variables respectively; $T(Y)$ and $\delta(X,Y,U)$ represent the set of states and the transition relation respectively.

```
1 Fixed_point_F(p) { //compute least fixed-point
2   R(X)=∅; T(X)=p(X);
3   while (T(X) !=0) {//lfp reached?
4     R(X) = R(X) ∨ T(X);
5     //get pre-image states for T but not in R
6     T(X)=SAT-EQ(δ ∧T(<X←Y>)∧¬R(X), X, U ∪ Y);}
7   return R(X); //return states satisfying F(p)
```

Figure 2. Fixed-point computation for *F(p)*

Note that the procedure *SAT-EQ* (line 6) returns the *pre-image* of the set $T(Y)$. $R(X)$ is the set of states that satisfy *F(p)* and is updated with pre-images in line 4. The exclusion of $R(X)$ in the pre-image computation using *SAT-EQ* (line 6) corresponds to a typical fixed-point check.

3. MOTIVATION

We are motivated by the following shortcomings of a standard monolithic LTL translation for BMC [2, 3], described earlier:

- The standard monolithic translation provides little scope for an incremental formulation within a k-instance BMC problem. Therefore, the past efforts [22, 23] on incremental learning for BMC have been limited to sharing of constraints across k-instances only.
- Finding a completeness bound for a standard monolithic formulation is not easy. Past efforts using SAT-based UMC [6-9] can be used to find bounds on safety properties. Nested properties, however, pose additional challenges to SAT-based UMC and have not been addressed so far.

4. OUR CONTRIBUTIONS

In this section, we describe our two main contributions:

1. We use customized property translations for LTL formulas for BMC, with novel features that utilize partitioning, learning, and incremental formulation. Our customized translations allow not only incremental learning across *k*-instances but also within *k*-instances of BMC problems and thereby, greatly improve overall performance. Moreover, in comparison to the standard translation, our customized translations provide additional opportunities for deriving completeness bounds for nested properties. Though we discuss such translations for liveness, we can easily extend this to handle other LTL properties as well.
2. We propose customized formulations for determining completeness bounds for liveness using SAT-based UMC rather than using loop-free path analysis. These formulations comprise greatest fixed-point and least fixed-point computations to handle nested properties efficiently using SAT-based quantification approaches.

4.1 Customized Translations: Novel Features

Most BMC methods use the standard monolithic translation originally proposed by Biere *et al.* [2]. In this section, we describe the customized translations we use instead. Rather than generating a monolithic SAT formula, our translation schemas can be viewed as building the formula incrementally, by lazily indexing over the bounded conjunctions/disjunctions, terminating early when possible. The main motivation is to use an incremental formulation, employing learning, and partitioning to generate multiple simpler SAT sub-problems. Though the standard BMC procedure also partitions the overall problem into separate *k*-instances, in our customized translation we further partition each *k*-instance problem into multiple, smaller SAT sub-problems. Furthermore, our partitioning is based on formulating sub-problems in a way that enables the following kinds of SAT-based incremental learning:

Learning from shared constraints (L1): Given two SAT instances S_1 and S_2, conflict clauses that are deduced solely from the set of constraints shared between S_1 and S_2 can be used as learned clauses while solving S_1 *or* S_2 [22, 23].

Learning from satisfiable results (L2): Suppose $\{\varphi_{1,...,}\varphi_n\}$ represents a series of SAT problems where problem φ_i is built incrementally by adding and removing constraints to and from φ_{i-1} respectively. The satisfying solution for φ_{i-1} (if it exists) can be used to guide SAT decision engine to solve φ_i [23].

Learning from unsatisfiable results (L3): Let a SAT problem Φ be a disjunction of sub-problems $\{\varphi_{1,...,}\varphi_n\}$. Instead of solving Φ as a monolithic problem, one can solve φ_i starting from $i=1$ with the additional constraint $C_i=\neg\varphi_1\wedge...\wedge\neg\varphi_{i-1}$ where each φ_j is unsatisfiable for all $j<i$. More benefit will be potentially obtained when the problem φ_i shares more with the additional constraint C_i, thereby, allowing *L1* learning.

In a standard BMC procedure, there is a considerable overlap of circuit constraints due to unrolled transition relations between a *k*-instance and a *(k+1)*-instance of the BMC problem. Some researchers have applied L1 learning across *k*-instances [22, 23], while some researchers have used L3 learning for safety properties only to reduce the overall verification time [24].

In our customized BMC translations, sharing occurs not just between the circuit constraints due to the unrolled transition relation, but also between the constraints arising from our property translations and between constraints learned from unsatisfiable SAT sub-problems. In order to take advantage of incremental SAT techniques and to mitigate the overheads of partitioning into multiple SAT sub-problems, our translation schemas are geared toward an incremental formulation, i.e. reuse of variables and constraints wherever possible. This allows use of incremental SAT learning techniques L1, L2 and L3 in the SAT solver very effectively. In contrast, the standard translation does not provide such an incremental formulation of the property constraints.

To highlight the partitioning, learning, and incremental aspects of our customized property translations, the pseudo code for handling LTL properties, in particular, *G(q)* and $F(p\wedge G(q))$ in presence of fairness constraints $B=\{f^1,...f^b\}$ are shown in Figure 3. Here, *p, q* and f^i denote Boolean constraints; *is_sat(C)* denotes SAT check of the Boolean formula *C*; *L{1-3}* denote the kind of incremental SAT-based learning; L_{ij} denote the loop transition constraint i.e. $L_{ij} = T(s_i, s_j)$; and FC_{ij} denotes the fairness constraints i.e. $FC_{ij} = \wedge_{1\leq k\leq b} (\vee_{i\leq k\leq j} f_k^r)$; *M, N* denote the loop and non-loop bound, both under user's control. Due to incremental formulation, learning *L1* is always active. For ease of description, the circuit constraints are not shown in these translations – they are always added to the SAT sub-problems.

Customized Translation of *G(q)*

Consider the translation of nested *G(q)* with fairness constraints, shown by the function call *G(IC,q,start)* where *IC* denotes the initial path constraint, and *start* is the current time frame. Non-nested *G(q)* is a special case with *IC=1* and *start=0* (line 2). The function *G(IC,q,start)*, searches for a path satisfying the constraint *IC* starting at s_{start}, such that *q=true* at each state in the path, and the path loops back to a previous state. The outer *for-loop* on index *i* (line 6) corresponds to incrementing the time-frames for BMC up to the user-provided bound *M*. It incrementally builds up the property database *C*, which is initially set to *IC* (line 5). For each *i*, we first check the satisfiability of $(C \& q_i)$ (line 8). If it is unsatisfiable, it returns false, as clearly there is no way to find a witness loop. If it is satisfiable, we collect satisfying decision variables for L2 learning. We then check the satisfiability of $(C \& FC_{ij})$ (line 10), in presence of fairness constraints. If it is unsatisfiable, we skip the witness loop check; else we proceed to check for witness loops incrementally (line 11) so that we can terminate early at the first detection. However, if such a loop is not found, this fact is learned (L3 at line 12). For non-zero *i*, checks for a loop back state at time frame $j<i$ is also done (lines 14-28). In this case, it must additionally prove that *q=true* at each such state. If it is found to be unsatisfiable at any *j*, $!q_j$ is learned (L3 at line 17) and *t* is updated accordingly (line 17) so that we do not need to loop back to a state before *t* (line 15). If the current path cannot be extended to remain loop-free (lines 25-28), we conclude the property to be false.

```
1   G(q){
2     return G(1,q,0);
3   }
4   G(IC,q,start){// L1 is active always
5     C = IC; t = 0; C''= 1;
6     for (i=start;i<M;i++) {
7       C = C & q_i;
8       if (!is_sat(C)) return false;//L2
9       for (j=i;j>=start;j--) {
10        if (!is_sat(C & FC_ij)) continue;//L2
11        if (is_sat(C &L_ij & FC_ij)) return true;
12        C = C & !L_ij; //L3
13       }
14       C' = C & C'';
15       for(j=start-1;j>=t;j--) {
16         if (!is_sat(C' & q_j)) {
17           t = j+1; C'' = C'' & !q_j ; //L3
18           break;
19         }//else L2
20         C' = C' & q_j;
21         if (!is_sat(C' & FC_ij)) break;//L2
22         if (is_sat(C'&L_ij& FC_ij)) return true;
23         C' = C' & !L_ij;    //L3
24       }
25       for(l=start-1;l>j;l--){//loopFree check
26         C = C & !L_il; //L3
27         if (!is_sat(C)) return false;
28       }
29    }
30    return undetermined;
31  }
32  F(p ∧ G(q)){//L1 is active always
33    C = 1; lfc=1;
34    for (i=0;i<N;i++) {
35      C = C & p_i;
36      if (!is_sat(C & p_i)) {
37        C = C & !p_i; //L3
38      }else {//L2
39        R = G(C & p_i,q,i));
40        if (R=true) return true;
41        else if (R=undetermined) lfc=0;
42      }
43      for (j=i;j>=0;j--) {//loop-free check
44        C = C & !L_ij; //L3
45        if (!lfc &&!is_sat(C)) return false;
46      }
47    }
48    return undermined;
49  }
```

Figure 3. Customized translation for liveness properties

Customized Translation of *F(p∧G(q)*

Nested properties like *F(p∧G(q)* with fairness constraints provide the opportunity to partition further, allowing greater learning and sharing across the partitions. The algorithm for *F(p∧G(q)* is shown in Figure 3 (lines 32-49). There are two user-provided bounds *N* and *M* for *F* and *G* checks respectively. The outer *for-loop* on *i* (line 34) increments the bound *k* for BMC. The first SAT sub-problem (line 36) is to check *(C & p_i)*. If it is unsatisfiable, *!p_i* is learned (L3) and added to *C* (line 37); otherwise, the algorithm checks for nested *G(q)* using the function call *G(C & p_i, q, i)* (line 39) which looks for a path starting at s_i, such that *q=true* at each state in the path, and the path loops back to a previous state. If the current path cannot be extended to remain loop-free in lines 43-46, we stop further. Moreover, if the *G* checks in line 39 have been false at all depths < *i* i.e. *lfc=1,* the property is proved to be false.

Note the large number of SAT calls made in the process of analyzing this property, and how the databases *(C, C', C'')* are incremented between successive calls. Besides L1 learning due to sharing of constraints and conflict clauses, SAT calls on sub-problems provides additional L2 and L3 learning opportunities. Also, due to reuse of variables and clauses in our incremental formulation, there is a large sharing of constraints that make L1-L3 very effective. The analysis of properties translated with our approach is much faster than with the standard translation as observed in our experiments.

4.2 SAT-based UMC for *G(q)*

We discuss the greatest fixed-point computation to obtain the completeness bound for *G(q)* property. Let $G^k(q)$ denote the k-bounded *G(q)* property. It is defined as:

$$G^k(q) = q \wedge X(q) \wedge \ldots \wedge X^k(q)$$

where *X* is a next state LTL operator and X^k is shorthand for *X…X* for *k*-times. Let H^k denote the set of states that satisfy $G^k(q)$, i.e. the set of states from which there is a path $s_0 \ldots s_k$ such that *q* is true in all *k+1* states. Note that, $H^{k+1} \subseteq H^k$. For non-empty *G(q)*, there exists some *k=m* such that $H^{m+1} = H^m$. We call *m* the greatest fixed-point bound (*gfp bound*). We make the following observations:

- If there is a finite state-path of length *m+1* where *q=true* in all the *m+1* states, then this finite segment is also the prefix to an infinite path where *q* is globally true (a witness loop).
- In other words, if there is no path from initial states *I(s)* of length *m+1* where *q* is true in all the *m+1* states, then the property *F(¬q)* is proved correct. Note, this is equivalent to satisfiability checking of $(I(s) \wedge H^m)$. Thus the *gfp bound* is a completeness bound for BMC on *G(q)*.

Using the *gfp bound m* one can determine if a witness *exists* by checking satisfiability of *q* for *m* time frames, without checking for any witness loops, which are quite expensive in practice. Though the *gfp bound* does not provide any bound on the length of a witness loop, it may not be worth the effort to find such a bound. If an *m*-length witness prefix exists, it is a matter of expending enough resources to find the witness loop. Note, *gfp bound* could be much smaller than the witness loop length as determined by longest path analysis. We have seen examples where it is much easier to find *m*-length witness prefixes confirming the existence of a witness, than the witness loop itself. We present the greatest fixed-point algorithm, *Fixed_Point_G*, as shown in Figure 4 for *G(q)*. It uses the *SAT-EQ* algorithm (line 7) for pre-image computation.

```
1 Fixed_Point_G(q) {//greatest fixed-point
2   i=0; T(X)=1;
3   do {
4     H(X) = T(X); i = i+1;
5     q^i(X,U)=Unroll(q, i);//get q at i^th depth
6     //Compute states satisfying q for depths ≤i
7     T(X) = SAT-EQ(q^i ∧H(X), X, U);
8     if (T(X)==0) return (NULL,0);//gfp is zero
9   }while (SAT_Solve(H(X)∧¬T(X))==SAT));//gfp?
10  return (T(X),i);} //states satisfying G(q)
```

Figure 4. SAT-based UMC for *G(q)*

Note that, unlike a standard algorithm where a pre-image computed in the previous step is used to compute the next pre-image [7, 13], we use the conjunction of unrolled q^i (*q* at i^{th} unroll depth) with *H(X)* to represent the target states (satisfying $G^i(q)$). This allows efficient handling of a nested *G(q)* without modifying the overall algorithm (discussed in the next section). The greatest fixed-point check is done by satisfiability check of *H(X)\T(X)* (line

9). Note that if no state satisfies *G(q)*, then it will be discovered at line 8 (an empty *T(X)* corresponds to zero enumeration steps in *SAT-EQ*). For a top-level (i.e. un-nested) property *G(q)*, an additional check for containment of the initial state(s) in the resulting set *T(X)* is required.

4.3 SAT-based UMC for $F(p \wedge G(q))$

Our customized translation of the property $F(p \wedge G(q))$ uses two bounds: *N* for finding a state where *p=true* and *M* for finding a witness loop where *q=true* always (and where the fairness constraints are also satisfied). In Section 4.2, we provided the completeness bound for *G(q)*, i.e., *M* as the *gfp bound m*. Here, we provide the completeness bound for *N* using a least fixed-point (lfp) computation. As *m* is the *gfp bound* for the nested *G(q)*, states satisfying $G^m(q)$ are equivalent to states satisfying *G(q)*. Note, that we do not represent $G^m(q)$ directly as a set of states. Instead, we unroll the transition relation *m* times and impose the constraint *q=true* on each time frame. In our experience, SAT-based UMC does not work well with state set representations for nested properties, but works better on unrolled transition relations. For obtaining the *lfp bound*, say *n*, we can simply use the algorithm *Fixed_Point_F* in Figure 2 on $p \wedge G^m(q)$ (instead of on *p*). If the lfp computation completes, then we can check for containment of the initial state(s) in the resulting state set. Alternately, the *lfp bound n* and the *gfp bound m* together give the completeness bounds for BMC on the property $F(p \wedge G(q))$. In other words, for our customized formulation of property $F(p \wedge G(q))$ (in Figure 3) we need to search for a path prefix (called *stem*) of length at most *n* to a state, where *p=true* and from which there exists a finite path of length *m* where q=*true* always. This is done efficiently in the procedure *G(IC,q,start)* using steps *a-c* as follows: *a)* set *M=m*, *b)* skip the expensive loop checks (lines 9-29), *c)* interpret *undetermined* result (line 30) as *true*. If no such *stem* of length $\leq n$ exists, $\neg F(p \wedge G(q))$ is proved correct.

5. EXPERIMENTS

We implemented these techniques in our SAT-based model checking framework called *DiVer* [28] that uses a hybrid SAT solver [25]. We experimented on a workstation with 2.8 GHz Xeon Processors with 4GB running Red Hat Linux 7.2.

Industry Examples: In the first set of experiments we show the benefits of our customized translation on liveness properties in the (negated) form $F(p \wedge G(q))$ on industry designs D1-D3. These are bus core designs with multiple masters and slaves. The properties to falsify are "*request should be eventually followed by acknowledge or error*". We compare various learning schemes i.e. NL (no learning), L1+L3, L2+L3, and L1+L2+L3 in our customized approach, called *custom*. We also compare them against standard BMC in VIS [26], called *standard*. For fair comparison, we use the same SAT heuristics in our SAT solver as those used in the backend SAT solver (zChaff [27]) in VIS BMC. We present the comparison results in Table 1. Columns 2-4 show the characteristics of the designs in Column 1, i.e., number of flip-flops, primary inputs and gates, respectively; Column 5 shows the length of counter example (CEX); Columns 6-9 and 10 show time taken by our customized translation with different learning schemes, and the standard translation, respectively. Clearly, our customized translation is able to find counter examples far quicker than the standard monolithic translation. Moreover, the various learning schemes improve the performance of BMC significantly.

Table 1. Improvements due to Customized Translations

D	#FF	#PI	#G	CEX (D)	Custom (DiVer)				Standard (VIS)
					NL	L1,3	L2,3	L1,2,3	
D1	2316	76	14655	19	2.3	2.2	2.3	2	77
D2	2563	88	16686	22	11.2	8.9	11.7	8	201
D3	2810	132	18740	28	730	290	862	240	2728

Public Benchmarks: In the second set of experiments, we used publicly available benchmarks [26]. We used the circuit-based cofactoring approach [9] for SAT-based UMC.

We first used our SAT-based UMC techniques for liveness to find completeness bounds – greatest fixed-point and least fixed-point – for liveness properties of the form $F(p \wedge G(q))$. We compare our results with bounds obtained using longest loop-free path analysis [21] for the sub-formula *G(q)*. We used a time limit of 1800s for each run. The results are shown in Table 2 (Note, 'TO' ≡ timeout, 'NA' ≡ Not Applicable.) Column 1 shows the designs and the number in the bracket denotes the property; Columns 2 and 3 show the completeness bound obtained for the sub-formula *G(q)* using the longest loop-free path analysis with standard monolithic translations and the corresponding time taken, respectively. Bound *k* with '*' indicates that the analysis timed out after that depth. The 0 value denotes that there is no state from which there is an infinite path where q holds always, where the number in the bracket indicates the number of unrolling required to discover it. The remaining columns show result for deriving bounds related to customized translations. Columns 4 and 5 show the *gfp bound* as obtained by the greatest fixed-point algorithm described in Figure 4 and the time taken, respectively. The 0 value has a similar meaning as in Column 2. Columns 6 and 7 show the *lfp bound* for $F(p \wedge G(q))$ computed using the algorithm described in Section 3.4 and the time taken, respectively. We do not compute *lfp bound* when the corresponding *gfp bound* is 0 (indicated by NA). It is not surprising that the loop-free analysis needs to search much deeper than the *gfp bound* analysis. However, note that deriving completeness bounds is much more efficient (requiring less resources) for customized partitioned translations for $F(p \wedge G(q)$ than the corresponding monolithic standard BMC translations.

Next, we used the *gfp* and *lfp bounds* as completeness bounds to find violation/proof for the properties using BMC. We omitted those properties for which the *gfp bound* is 0, since that corresponds to a proof of correctness. We compare our customized translation (w/ L1+L2+L3) with the standard monolithic translation. For our customized translation, we set *N* to the *lfp bound* and *M* to the *gfp bound,* and for the monolithic translation we set *k* to *lfp+gfp bound.* We present the results in Table 3. Column 1 shows the designs (with non-zero gfp bound). Columns 2-5 show the results using our customized translation for BMC; specifically, Columns 2-3 indicate the existence (*Y*) of counter-example (*CEX*) or proof (*N*) and the corresponding time taken, respectively; Columns 4-5 show the *CEX* length (if any) and the corresponding time taken to find the *CEX* loop, respectively. Columns 6-7 show the results for standard translation for BMC in VIS; specifically, Column 6 shows the *CEX* length or proof (*N*) and Column 7 shows the corresponding

time taken. Note that, in the standard translation, computation of existence of *CEX* is not separated from finding the actual *CEX*. (Note, '?' denotes result is undetermined.) Clearly, our translation is able to use completeness bound much more efficiently to find violation/proofs than the standard translation. Also, as shown in the *Miim* examples, it is easier to detect the existence of a loop without actually finding one.

Table 2. Completeness Bounds for $F(p \wedge G(q)$

Design	Loop-free G(.)		Fixed-point G(.)		Fixed-point F(.)	
	k	sec	gfp	sec	lfp	sec
Feistal	0(8)	<1	0(8)	<1	NA	NA
Miim(1)	172*	TO	2	<1	5	<1
Miim(2)	169*	TO	2	<1	5	<1
Coherence(1)	56*	TO	11	56	10	1169
Coherence(2)	55*	TO	13	146	11	1128
Palu (1)	95*	TO	3	<1	5	<1
Palu (2)	93*	TO	3	<1	5	<1
Pathfinder	9	< 1	2	<1	9	<1
s1269 (1)	0(7)	<1	0 (7)	<1	NA	NA
Smult	134*	TO	1	<1	3	<1
Unidec	0(12)	<1	0(12)	6	NA	NA
VsaR	112*	TO	79	262	7	34

Table 3. BMC with Completeness Bounds

Design	Custom (N=lfp, M=gfp)				Standard (k=lfp+gfp)	
	CEX?	sec	CEX	sec	CEX	sec
Miim (1)	Y	<1	?	TO	?	TO
Miim (2)	Y	<1	?	TO	?	TO
Coherence (1)	N	5	NA	NA	N	288
Coherence (2)	N	10	NA	NA	N	479
Palu (1)	Y	<1	1	<1	1	<1
Palu (1)	Y	<1	1	<1	1	<1
Pathfinder	Y	<1	2	<1	2	<1
Smult	Y	<1	4	<1	4	<1
VsaR	N	7	NA	NA	N	665

6. CONCLUSIONS

In this paper, we have proposed techniques to bridge the technology gap between model checking of safety properties and non-safety properties that has widened in recent years. We provided dedicated SAT-based model checking algorithms for such properties and discussed them specifically for liveness. We provided customized property translations for BMC with novel features that utilize partitioning, learning, and incremental formulation. These customized translations allow not only incremental learning within and across *k*-instances BMC problems, thereby improving performance in comparison to standard LTL translations, but also allow efficient derivation and use of completeness bounds. We also proposed formulations for determining completeness bounds for liveness using SAT-based UMC, rather than using loop-free path analysis. These comprise greatest fixed-point and least fixed-point computations in a way that allows efficient handling of nested properties. We showed the effectiveness of our approach on several public benchmarks and industry designs.

7. REFERENCES

[1] B. Alpern and F. B. Schneider, "Defining liveness," *Information Processing Letters*, 1985.

[2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," in *Proceedings of TACAS*, 1999.

[3] A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani, "Improving the Encoding of {LTL} Model Checking into SAT," *Proceedings of VMCAI*, 2002.

[4] M. Sheeran, S. Singh, and G. Stalmarck, "Checking Safety Properties using Induction and a SAT Solver," in *Proceedings of FMCAD*, 2000.

[5] A. Gupta, M. Ganai, C. Wang, Z. Yang, and P. Ashar, "Abstraction and Bdds Complement SAT-Based BMC in DiVer," in *Proceedings of CAV*, 2003.

[6] A. Gupta, Z. Yang, P. Ashar, and A. Gupta, "SAT-based Image Computation with Application in Reachability Analysis," in *Proceedings of FMCAD*, 2000.

[7] K. McMillan, "Applying SAT methods in Unbounded Symbolic Model Checking," in *Proceedings of CAV*, 2002.

[8] K. McMillan, "Interpolation and SAT-based Model Checking," in *Proceedings of CAV*, 2003.

[9] M. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based Symbolic Unbounded Model Checking Using Circuit Cofactoring," in *Proceedings of ICCAD*, 2004.

[10] K. McMillan and N. Amla, "Automatic Abstraction without Counterexamples," in *Proceedings of TACAS*, 2003.

[11] A. Gupta, M. Ganai, P. Ashar, and Z. Yang, "Iterative Abstraction using SAT-based BMC with Proof Analysis," in *Proceedings of ICCAD*, 2003.

[12] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*: MIT Press, 1999.

[13] K. L. McMillan, *Symbolic Model Checking: An Approach to the State Explosion Problem*: Kluwer Academic Publishers, 1993.

[14] "Liveness Manifesto. Beyond Safety International Workshop. 2004."

[15] P. Wolper, M. Y. Vardi, and A. P. Sistla, "Reasoning about infinite computation paths.," *Proceedings of Symposium on FCS*, 1983.

[16] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," *Protocol Specification, Testing and Verification*, 1995.

[17] M. Daniele, F. Giunchiglia, and M. Y. Vardi, "Improved automata generation for linear time temporal logic," *Proceedings of CAV*, 1999.

[18] F. Somenzi and R. Bloem, "Efficient Buchi Automata from LTL formulae," *Proceedings of CAV*, 2000.

[19] D. Kroening and O. Shtrichman, "Efficient computation of recurrence diameter," *Proceedings of VMCAI*, 2003.

[20] A. Biere, C. Artho, and V. Schuppan, "Liveness Checking as Safety Checking," *Proceedings of FMICS*, 2002.

[21] M. Awedh and F. Somenzi, "Proving more properties with Bounded Model Checking," *Proceedings of CAV*, 2004.

[22] O. Shtrichman, "Pruning Techniques for the SAT-based bounded model checking," in *Proceedings of TACAS*, 2001.

[23] J. Whittemore, J. Kim, and K. Sakallah, "SATIRE: A New Incremental Satisfiability Engine," *Proceedings of DAC*, 2001.

[24] M. Ganai and A. Aziz, "Improved SAT-based Bounded Reachability Analysis," in *Proceedings of VLSI Design Conference*, 2002.

[25] M. Ganai, L. Zhang, P. Ashar, and A. Gupta, "Combining Strengths of Circuit-based and CNF-based Algorithms for a High Performance SAT Solver," in *Proceedings of DAC*, 2002.

[26] "The VIS Home Page. http://www-cad.eecs.berkeley.edu/Respep/Research/vis/."

[27] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of DAC*, 2001.

[28] M. Ganai, A. Gupta, and P. Ashar, "DiVer: SAT-based Model Checking Platform for Verifying Large Scale Systems", in *Proceedings of TACAS*, 2005.