k-Step Relative Inductive Generalization

Aaron R. Bradley

Dept. of Electrical, Computer & Energy Engineering
University of Colorado at Boulder
Boulder, CO 80309
bradleya@colorado.edu

Abstract. We introduce a new form of SAT-based symbolic model checking. One common idea in SAT-based symbolic model checking is to generate new clauses from states that can lead to property violations. Our previous work suggests applying induction to generalize from such states. While effective on some benchmarks, the main problem with inductive generalization is that not all such states can be inductively generalized at a given time in the analysis, resulting in long searches for generalizable states on some benchmarks. This paper introduces the idea of inductively generalizing states relative to k-step over-approximations: a given state is inductively generalized relative to the latest k-step over-approximation relative to which the negation of the state is itself inductive. This idea motivates an algorithm that inductively generalizes a given state at the highest level k so far examined, possibly by generating more than one mutually k-step relative inductive clause. We present experimental evidence that the algorithm is effective in practice.

1 Introduction

Several themes for SAT-based symbolic model checking [6] have been explored over the past decade [3,18,14,15,17,5]. A subset of these methods [14,17,5] derive new search-constraining clauses from discovered states that lead to property violations. In previous work, we introduced induction as one means of generalizing from such states. Given a cube c that one would like to exclude because the states that it describes lead to violations of a desired property, a *minimal inductive subclause* d of $\neg c$ is a clause whose literals are negations of those appearing in c ($d \subseteq \neg c$) and that is inductive relative to known reachability information [5]. Not all cubes can be inductively generalized at a given time during proof construction, however. This inability to inductively generalize any given cube (whose satisfying states lead to property violations) limits the applicability of the technique as previously developed [5]: on some benchmarks, the model checker becomes embroiled in long fruitless searches for generalizable cubes. However, its success on some nontrivial benchmarks indicates that the fundamental idea of inductive generalization from states is worth exploring [4].

We describe in this paper a method based on induction for generalizing all cubes (unless the asserted property does not hold). The algorithm maintains a sequence $F_0, F_1, F_2, \ldots, F_k$ of over-approximations of sets of states reachable in at most $0, 1, 2, \ldots, k$ steps, for increasing k. It iteratively generalizes cubes: a cube s that implies F_k and that leads in one step to violating the property is inductively generalized relative to the most general over-approximation F_i relative to which the negation of the state, $\neg s$, is itself inductive. If $i < k$, predecessors of s are treated recursively until s can be inductively generalized relative to F_k. We call this process *k-step relative inductive generalization*. Once F_k is strengthened to the point that no F_k-state can transition into a property-violating state, k is incremented and the generated clauses are propagated forward through $F_0, F_1, F_2, \ldots, F_{k+1}$ via implication checks. The iterations continue until convergence (if the property is invariant) or until discovery of a counterexample trace (if the property is not invariant). Section 3 presents this algorithm in detail.

The symbolic model checker based on k-step relative inductive generalization is robust. Section 4 details our implementation and experiments on the HWMCC 2008 benchmarks [2]. Our symbolic model checker outperforms the winner of the *unsat* division and the overall winner of the competition.

2 Preliminaries

2.1 Definitions

A *finite-state transition system* $S : (\bar{x}, I, T)$ is described by a pair of propositional logic formulas: an initial condition $I(\bar{x})$ and a transition relation $T(\bar{x}, \bar{x}')$ over a set of Boolean variables $\bar{x}$ and their next-state primed forms $\bar{x}'$ [8]. Applying prime to a formula, F', is the same as priming all of its variables.

A state of the system is an assignment of Boolean values to all $\bar{x}$ and is described by a *cube* over $\bar{x}$, which is a conjunction of literals, each *literal* a variable or its negation. The negation of a cube is a *clause*. An assignment s to all variables of a formula F either satisfies the formula, denoted $s \models F$, or falsifies it, denoted $s \not\models F$. A formula F *implies* another formula G, written $F \Rightarrow G$, if every satisfying assignment of F satisfies G.

A *trace* $s_0, s_1, s_2, \ldots$ of a transition system S, which may be finite or infinite in length, is a sequence of states such that $s_0 \models I$ and for each adjacent pair (s_i, s_{i+1}) in the sequence, $s_i \wedge s'_{i+1} \models T$. That is, a trace is the sequence of assignments in an execution of the transition system. A state that appears in some trace of the system is *reachable*.

A safety property $P(\bar{x})$ asserts that only P-states (states satisfying P) are reachable. P is *invariant* for the system if indeed only P-states are reachable. If P is not invariant, then there exists a finite *counterexample* trace $s_0, s_1, \ldots, s_k$ such that $s_k \not\models P$.

An *inductive* assertion $F(\bar{x})$ describes a set of states that (1) includes all initial states: $I \Rightarrow F$, and that (2) is closed under the transition relation: $F \wedge T \Rightarrow F'$. An assertion F is inductive *relative to* another assertion G if instead of (2), we have that $G \wedge F \wedge T \Rightarrow F'$.

An inductive *strengthening* of a safety property P is a formula F such that $F \wedge P$ is inductive. Since $F \wedge P \Rightarrow P$, F is a proof of P's invariance.

2.2 Inductive Generalization

In previous work, we introduced a technique for discovering a *minimal inductive subclause* d of a given clause c if one exists [5]. Such a clause d (1) consists only of literals of c ($d \subseteq c$), (2) is inductive (possibly relative to known reachability information), and (3) is minimal in that it does not contain any strict subclauses that are also inductive.

Inductive generalization of a cube s is the process of finding a minimal inductive subclause d of $\neg s$, if one exists. The resulting subclause (if one exists) over-approximates the set of reachable states while excluding s. In practice, a minimal inductive subclause is typically substantially smaller than the cube s from which it is extracted. Hence, it excludes many other states as well, which is why we say that the inductive subclause generalizes that s is unreachable.

Substantially
大幅地

3 Algorithm and Analysis

We describe a complete symbolic model checking algorithm for safety properties. Given a transition system $S : (\bar{x}, I, T)$ and safety property P, it either generates a formula F such that $F \wedge P$ is inductive or it discovers a counterexample trace.

Section 3.1 presents the algorithm informally, while Section 3.2 provides an example of its application. Then Section 3.3 formally describes and proves the correctness of the algorithm.

3.1 Informal Description

The algorithm constructs a sequence $F_0, F_1, F_2, \ldots$ of over-approximations of the state sets reachable in at most $0, 1, 2, \ldots$ steps. It incrementally refines the sequence until some F_i converges to an inductive strengthening of P, or until it encounters a counterexample trace.

Initially, $F_0 = I$, and $F_i = P$ for $i > 0$, corresponding to the assumption that P is invariant. Let k be the level of F_k, the frontier of the sequence. The sequence satisfies the following invariants: (1) $F_0 = I$, (2) $\forall\ 0 \leq i < k,\ F_i \Rightarrow F_{i+1}$, and (3) $\forall\ 0 \leq i < k,\ F_i \wedge T \Rightarrow F'_{i+1}$. If $F_k \wedge T \Rightarrow P'$, then F_{k+1} becomes the new frontier. Otherwise, there is a state s that leads in one step to a violation of P.

Given such a state s, the algorithm finds the highest level $0 \leq i \leq k$ such that $\neg s$ is inductive relative to F_i. If P is invariant, such a level exists. At this level, s can be inductively generalized relative to F_i.

reachability

Inductive generalization produces a clause $c \subseteq \neg s$ that is inductive relative to F_i. It asserts that s — and any other state t such that $t \not\models c$ — is not reachable within $i + 1$ steps. Because $\neg s$ has been generalized to c, c may exclude states that were previously admitted by some F_j for $j \leq i + 1$. In other words, c potentially represents new j-step reachability information at every level j up to $i + 1$. Therefore, each F_j, for $1 \leq j \leq i + 1$, is strengthened to $F_j \wedge c$.

$F_i \wedge \neg s \wedge T \Rightarrow \neg s'$

$i = k \quad F_k \wedge \neg s \wedge T \Rightarrow \neg s'$

If $i = k$, then s has been inductively generalized at the highest possible level, and F_k no longer admits the state s, bringing the algorithm one step closer to strengthening F_k such that $F_k \wedge T \Rightarrow P'$.

$P \wedge T \Rightarrow S$

$F_i \wedge \neg S \wedge T \Rightarrow \neg S'$

If $i < k$, then the generalization of s at level i must be pushed to level k. There must exist some predecessor p of s admitted by F_{i+1} but excluded by F_i. This predecessor is one of the reasons that $\neg s$ is not inductive relative to F_{i+1}. Now p is considered recursively for inductive generalization. This recursion continues until s can be inductively generalized relative to F_k.

Once $F_k \wedge T \Rightarrow P'$ holds, the clauses that have been generated so far are propagated forward through $F_0, F_1, F_2, \ldots, F_k$: for each clause $d \in \texttt{clauses}(F_i)$, if $F_i \wedge d \wedge T \Rightarrow d'$, then d is conjoined to F_{i+1}. If the clause sets of two adjacent levels, F_i and F_{i+1}, become equal, then F_i is an inductive strengthening of P that proves P's invariance.

If P is not invariant, the algorithm discovers a counterexample trace, though not necessarily a shortest. Let $s_0, s_1, \ldots, s_n$ be a shortest counterexample trace. The algorithm finds a counterexample trace when $k = n$, if not earlier. For when $k = n$, each s_i, for $2 \leq i \leq n$, can be shown to be inductive relative to at most F_{i-2}. Hence, s_1 (or another 1-step state from another counterexample trace) must eventually be analyzed during the recursion associated with inductively strengthening s_n (or another state from another counterexample trace) relative to F_n, at which point it would be found to be reachable from an initial state.

3.2 An Illustrative Example

$P: x \wedge y$

Consider the contrived transition system $S : (\bar{x}, I, T)$ with variables $\bar{x} = \{x_0, x_1, x, y_0, y_1, y, z\}$, initial condition

$$I : \; x_0 \wedge \neg x_1 \wedge x \wedge (y_0 = \neg y_1) \wedge y \wedge z \; ,$$

$x \wedge y \wedge (\underline{x_0 \vee x_1 \vee x \vee y_0 \vee y_1 \vee}$

$\underline{y \vee \neg z)}$

and transition relation

$$T : \begin{bmatrix} (x_0' = \neg x_0) \wedge (x_1' = \neg x_1) \wedge (x' = x_0 \vee x_1) \\ \wedge\, (y_0' = x \wedge \neg y_0) \wedge (y_1' = x \wedge \neg y_1) \wedge (y' = y_0 \vee y_1) \\ \wedge\, (z' = x \wedge y) \end{bmatrix} .$$

$\wedge(\; - \cdot - \;)$

$\underline{(x_0 \vee x_1) \wedge (y_0 \vee y_1)}$

$\underline{\wedge(\; \text{-} \; \text{---} \; \text{--} \;)}$

$\underline{\neg x_0 \vee \neg x_1 \vee x_0 \vee x_1}$

The intention is that x and y — and thus z — are always $\texttt{true}$. This intention is asserted as the safety assertion $P : z$. We apply the algorithm to this transition system to prove the invariance of P.

1. F_0 is initialized to I, each of $F_1, F_2, F_3, \ldots$ to P, and k to 1.
2. $F_1 \wedge T \wedge \neg P'$ is satisfiable. One satisfying assignment yields the $\neg P$-predecessor $s_1 : \neg x_0 \wedge \neg x_1 \wedge \neg x \wedge \neg y_0 \wedge \neg y_1 \wedge \neg y \wedge z$. Is $\neg s_1$ inductive relative to F_1? Yes, as $F_1 \wedge \neg s_1 \wedge T$ implies $\neg s_1'$. Inductive generalization of s_1 relative to F_1 yields the clause $c_1 : x_0 \vee x$, where (1) $c_1 \subset \neg s_1$, and (2) c_1 is inductive relative to F_1. As Table 1 illustrates, c_1 is conjoined at both levels 1 and 2 while still maintaining the invariants on the sequence $F_0, F_1, F_2, \ldots$ discussed above. The clause c_1 not only excludes s_1 but also many other states, which is the purpose of inductive generalization.
3. $F_1 \wedge T \wedge \neg P'$ is still satisfiable. One satisfying assignment yields the $\neg P$-predecessor $s_2 : x_0 \wedge \neg x_1 \wedge \neg x \wedge \neg y_0 \wedge \neg y_1 \wedge \neg y \wedge z$. $\neg s_2$ is inductive relative to F_1. Inductive generalization yields from $\neg s_2$ the clause $c_2 : x_1 \vee x$, which is also inductive relative to F_1.

Table 1. Incremental construction of an inductive strengthening assertion

Level	0	1	2	3	4	5	6	7
F_0	I							
F_1	P	c_1	c_2	c_3	c_4	c_5	c_4	c_6
F_2	P	c_1	c_2			c_5	c_4	c_6

$c_1 : x_0 \vee x$ $\quad c_4 : x_0 \vee x_1$
$c_2 : x_1 \vee x$ $\quad c_5 : \neg x_0 \vee \neg x_1$
$c_3 : \neg y_0 \vee y$ $\quad c_6 : x$

4. $F_1 \wedge T \wedge \neg P'$ is still satisfiable. One satisfying assignment yields the $\neg P$-predecessor $s_3 : x_0 \wedge x_1 \wedge \neg x \wedge y_0 \wedge y_1 \wedge \neg y \wedge z$, which has predecessor $s_4 : \neg x_0 \wedge \neg x_1 \wedge x \wedge \neg y_0 \wedge \neg y_1 \wedge y \wedge z$ at level 1. Hence, $\neg s_3$ is not inductive relative to F_1. However, it is inductive relative to F_0, and inductive generalization yields from $\neg s_3$ the clause $c_3 : \neg y_0 \vee y$ at level 0. As Table 1 indicates, c_3 is only placed at level 1 (and implicitly at level 0).
5. The state s_3 is again considered at level 1, but as c_3 does not exclude s_4, $\neg s_3$ is still not inductive relative to F_1. Therefore s_4 is considered. But it, too, has a predecessor $s_5 : x_0 \wedge x_1 \wedge x \wedge y_0 \wedge y_1 \wedge y \wedge z$ at level 1. However, it is inductive relative to F_0, and inductive generalization yields $c_4 : x_0 \vee x_1$ at level 0.
6. Now either s_3 or s_4 must be considered at level 1. Choosing s_3 reveals that $\neg s_3$ is now inductive relative to F_1, and inductive generalization yields $c_5 : \neg x_0 \vee \neg x_1$ at level 1. Notice how the deduction of c_4 at level 0 is crucial to the deduction of c_5 at level 1.
7. To finish this iteration, it remains to address s_4 at level 1. With the addition of c_5, $\neg s_4$ is inductive relative to F_1, and inductive generalization yields again the clause $c_4 : x_0 \vee x_1$, but now at level 1 instead of level 0. Inductively generalizing cubes at the highest possible levels until convergence at k makes it possible to deduce the equivalence $x_0 = \neg x_1$, which requires two clauses to express.
8. $F_1 \wedge T \wedge \neg P'$ is still satisfiable. One satisfying assignment yields the $\neg P$-predecessor $s_6 : x_0 \wedge \neg x_1 \wedge \neg x \wedge y_0 \wedge y_1 \wedge \neg y \wedge z$, which is inductive relative to F_1. Inductive generalization yields the clause $c_6 : x$ at level 1.
9. With x at level 1, analysis of the y component of the transition system proceeds similarly until $F_1 \wedge T \wedge \neg P'$ becomes unsatisfiable.
10. Propagation from F_1 to F_2 and from F_2 to F_3 reveals that all clauses are inductive and inductively strengthen z. Simplifying through subsumption and rewriting the formula yields the expected inductive strengthening

$$x_0 = \neg x_1 \wedge x \wedge y_0 = \neg y_1 \wedge y \wedge z$$

of the safety assertion $P : z$, thus proving its invariance.

$\underline{F_1} \wedge \underline{\neg} S_3 \wedge T \Rightarrow \underline{\neg S_3'}$

$S_4 \subseteq \neg S_3$

$\underline{F_1 \wedge S_4 \wedge T \Rightarrow S}$

$\underline{F_1}$ State $\underline{S_4}$

$\left\{ \begin{array}{l} S_5 \subseteq \neg S_4 \\ F_1 \wedge S_5 \wedge T \Rightarrow S_4 \end{array} \right.$

k-step relative

inductive generation

k-step relative

inductive generation

assignment

3.3 Formal Presentation and Analysis

We present the algorithm and its proof of correctness simultaneously with formally annotated pseudocode in Listings 1.1-1.4 using the classic approach to program verification [11,13]. All assertions are inductive, but the ranking functions require some additional reasoning. For convenience, some assertions are labeled and subsequently referenced in annotations.

Listing 1.1. The main function

```
-post: $rv$ iff $P$ is invariant
bool prove():
  if either $I \wedge \neg P$ or $I \wedge T \wedge \neg P'$ is satisfiable:
    -assert: there exists a counterexample trace
    return false
  $F_0$ := $I$, clauses($F_0$) := $\emptyset$
  $F_i$ := $P$, clauses($F_i$) := $\emptyset$ for all $i > 0$
  for $k$ = 1 to ...:
    -rank: at most $2^{|\bar{x}|} + 1$
    -assert ($A$):
      (1) $\forall\ i \geq 0,\ I \Rightarrow F_i$
      (2) $\forall\ i \geq 0,\ F_i \Rightarrow P$
      (3) $\forall\ i > 0,\ \texttt{clauses}(F_{i+1}) \subseteq \texttt{clauses}(F_i)$
      (4) $\forall\ 0 \leq i < k,\ F_i \wedge T \Rightarrow F'_{i+1}$
      (5) $\forall\ i > k,\ |\texttt{clauses}(F_i)| = 0$
    if not check($k$):
      -assert: there exists a counterexample trace
      return false
    propagate($k$)
    if there exists $1 \leq i \leq k$ such that $\texttt{clauses}(F_i) = \texttt{clauses}(F_{i+1})$:
      -assert:
        (1) $I \Rightarrow F_i$
        (2) $F_i \wedge T \Rightarrow F'_i$
        (3) $F_i \Rightarrow P$
      return true
```

Listing 1.1 presents the top-level function prove, which returns true if and only if P is invariant. First it looks for 0-step and 1-step counterexample traces. If none are found, $F_0, F_1, F_2, \ldots$ are initialized to assume that P is invariant, while their clause sets are initialized to empty. As a formula, F_i for $i > 0$ is interpreted as $P \wedge \bigwedge \texttt{clauses}(F_i)$. Then it constructs the sequence of k-step over-approximations starting with $k = 1$. On each iteration, it first calls check(k) (Listing 1.2), which strengthens F_i for $1 \leq i \leq k$ so that F_i-states are at least $k - i + 1$ steps away from violating P. Then it calls propagate(k) (Listing 1.2) to propagate clauses forward through $F_1, F_2, \ldots, F_{k+1}$ based on their having become inductive relative to higher levels during the call to check. If this propagation yields any adjacent levels that share all clauses (a simple syntactic check, not a validity check), an inductive strengthening of P has been discovered.

While the assertions are inductive, an argument needs to be made to justify the ranking function. By $A.3$, the state sets represented by $F_0, F_1, \ldots, F_k$ are nondecreasing with level. To avoid termination at the if check requires that they be strictly increasing with level, which is impossible when k exceeds the number of possible states. Hence, k is bounded by $2^{|\bar{x}|} + 1$, and, assuming that the called functions always terminate, prove always terminates.

For a given level k, check(k) (Listing 1.2) iterates until F_k excludes all states that can lead to a violation of P in one step. Suppose s is one such state. It is

Listing 1.2. The `check` and `propagate` functions

```
-pre:
  (1) $A$
  (2) $k \geq 1$
-post:
  (1) $A$.1-3
  (2) if $rv$ then $\forall\ 0 \leq i \leq k,\ \ F_i \wedge T \Rightarrow F'_{i+1}$
  (3) $\forall\ i > k+1,\ \ |\texttt{clauses}(F_i)| = 0$
  (4) if not $rv$ then there exists a counterexample trace
bool check($k$ : level):
  try:
    while $F_k \wedge T \wedge \neg P'$ is satisfiable:
      -rank: at most $2^{|\bar{x}|}$
      -assert ($B$):
        (1) $A$.1-4
        (2) $\forall\ c \in \texttt{clauses}(F_{k+1}),\ \ F_k \wedge T \Rightarrow c'$
        (3) $\forall\ i > k+1,\ \ |\texttt{clauses}(F_i)| = 0$
      let $s$ be the predecessor extracted from the witness
      -assert: $k < 2$ or $\neg s$ is inductive relative to $F_{k-2}$
      $n$ := inductive($s$, $k-2$, $k$)
      push($\{(n+1, s)\}$, $k$)
      -assert ($C$): $s \not\models F_k$
    return true
  except Counterexample:
    return false

-pre/post:
  (1) $A$.1-3
  (2) $\forall\ 0 \leq i \leq k,\ \ F_i \wedge T \Rightarrow F'_{i+1}$
  (3) $\forall\ i > k+1,\ \ |\texttt{clauses}(F_i)| = 0$
void propagate($k$ : level):
  for $i$ = 1 to $k$:
    for each $c$ in clauses($F_i$):
      -assert: pre/post
      if $F_i \wedge T \wedge \neg c'$ is unsatisfiable:
        $F_{i+1}$ := $F_{i+1} \wedge c$
```

eliminated by, first, inductively generalizing it at the highest level n at which $\neg s$ is inductive relative to F_n through a call to `inductive`(s, $k-2$, k) (Listing 1.3) and then, second, pushing for a generalization at level k through a call to `push`($\{(n+1, s)\}$, k) (Listing 1.4). At the end of the iteration, F_k excludes s (assertion C). This progress implies that the loop can iterate at most as many times as there are possible states, yielding `check`'s ranking function.

Notice how `check`, according to its postcondition, preserves loop invariants A.1-3 while incrementing A.4-5 to to apply to an additional step (see postconditions (2) and (3)), unless a counterexample is found.

The functions `inductive` and `generate` (Listing 1.3) perform inductive generalization. The details of discovering an inductive subclause are described in

Listing 1.3. *i*-step relative inductive generalization

```
61 -pre:
62   (1) $B$
63   (2) $i \geq 0$
64   (3) $\neg s$ is inductive relative to $F_i$
65 -post:
66   (1) $B$
67   (2) $s \not\models F_{i+1}$
68 void generate($s$ : state, $i$ : level, $k$ : level):
69   $c$ := find subclause of $\neg s$ that is inductive relative to $F_i$
70   for $j$ = 1 to $i+1$:
71     -assert:
72       (1) $B$
73       (2) $s \not\models F_{j-1}$
74     $F_j$ := $F_j \wedge c$
75
76 -pre:
77   (1) $B$
78   (2) $min \geq -1$
79   (3) $min < 0$ or $\neg s$ is inductive relative to $F_{min}$
80   (4) there is a trace from $s$ to a $\neg P$-state
81 -post:
82   (1) $B$
83   (2) $min \leq rv \leq k$, $rv \geq 0$
84   (3) $s \not\models F_{rv+1}$
85   (4) $\neg s$ is inductive relative to $F_{rv}$
86 level inductive($s$ : state, $min$ : level, $k$ : level):
87   if $min < 0$ and $F_0 \wedge T \wedge \neg s \wedge s'$ is satisfiable:
88     -assert: there exists a counterexample trace
89     raise Counterexample
90   for $i$ = max(1, $min+1$) to $k$:
91     -assert:
92       (1) $B$
93       (2) $min < i \leq k$
94       (3) $\forall\ 0 \leq j < i$, $\neg s$ is inductive relative to $F_j$
95     if $F_i \wedge T \wedge \neg s \wedge s'$ is satisfiable:
96       generate($s$, $i-1$, $k$)
97       return $i-1$
98   generate($s$, $k$, $k$)
99   return $k$
```

$F_i \wedge \neg s \wedge T \Rightarrow \neg s'$

~~$\wedge c$~~

$t \subseteq \neg s$

$\underbrace{t \wedge T}$ $\Rightarrow \neg s'$

$F_{n-2} \wedge t$

previous work [5]. One interesting observation, however, is that when calling `inductive`, a minimum level *min* at which $\neg s$ is inductive relative to F_{min} can be supplied. At lines 43-44, $s \not\models F_{k-1}$ by $A.2$ and $A.4$ so that $\neg s$ is inductive relative to F_{k-2} by $A.4$. At lines 127-128, $\neg s$ is inductive relative to F_{n-1} so that $p \not\models F_{n-1}$ and thus $\neg p$ is inductive relative to F_{n-2} by $A.4$. If $min < 0$, then it is possible that s is reachable from an initial state, hence the check at line 87.

Listing 1.4. The push function for k-step relative inductive generalization

```
-pre:
  (1) $B$
  (2) $\forall\ (i,q) \in \mathit{states},\ \ 0 < i \le k+1$
  (3) $\forall\ (i,q) \in \mathit{states},\ \ q \not\models F_i$
  (4) $\forall\ (i,q) \in \mathit{states},\ \ \neg q$ is inductive relative to $F_{i-1}$
  (5) $\forall\ (i,q) \in \mathit{states}$, there is a trace from $q$ to a $\neg P$-state
-post:
  (1) $B$
  (2) $\forall\ (i,q) \in \mathit{states},\ \ q \not\models F_k$
void push($\mathit{states}$ : (level, state) set, $k$ : level):
  while true:
    -rank: at most $(k+1)2^{|\bar{x}|}$
    -assert ($D$):
      (1) $B$
      (2) $\forall\ (i,q) \in \mathit{states}_{\mathsf{prev}},\ \ \exists j \ge i,\ \ (j,q) \in \mathit{states}$
      (3) $\forall\ (i,q) \in \mathit{states},\ \ 0 < i \le k+1$
      (4) $\forall\ (i,q) \in \mathit{states},\ \ q \not\models F_i$
      (5) $\forall\ (i,q) \in \mathit{states},\ \ \neg q$ is inductive relative to $F_{i-1}$
      (6) $\forall\ (i,q) \in \mathit{states}$, there is a trace from $q$ to a $\neg P$-state
    ($n$, $s$) := choose pair from $\mathit{states}$ that minimizes $n$
    -assert: $\forall\ (i,q) \in \mathit{states},\ \ n \le i$
    if $n > k$:
      return
    if $F_n \wedge T \wedge s'$ is satisfiable:
      let $p$ be the predecessor extracted from the witness
      -assert ($E$):
        (1) $\forall\ (i,q) \in \mathit{states},\ \ p \ne q$
        (2) $n < 2$ or $\neg p$ is inductive relative to $F_{n-2}$
      $m$ := inductive($p$, $n-2$, $k$)
      $\mathit{states}$ := $\mathit{states} \cup \{(m+1, p)\}$
    else:
      $m$ := inductive($s$, $n$, $k$)
      -assert ($F$): $m+1 > n$
      $\mathit{states}$ := $\mathit{states} \setminus \{(n,s)\} \cup \{(m+1,s)\}$
```

The push algorithm (Listing 1.4) is the key to "pushing" inductive generalization to higher levels. The insight is simple: if a state s is not inductive relative to F_i, apply inductive generalization to its predecessors that satisfy F_i. The complication is that this recursive analysis must proceed in a manner that terminates despite the presence of cycles in the system's state graph. To achieve termination, a set *states* of pairs (i, s) is maintained such that each pair $(i, s) \in \mathit{states}$ represents the knowledge that (1) s is inductive relative to F_{i-1}, and (2) F_i excludes s. The loop in push always selects a pair (n, s) from *states* such that n is minimal over the set. Hence, none of the states already represented in *states* can be a predecessor of s at level n.

Formally, termination of `push` is established by the inductive assertions $D.\mathtt{2}$, which asserts that the set of states represented in *states* does not decrease; $E.\mathtt{1}$, which asserts that each state in *states* is represented by at most one pair in *states*; and F, which asserts that the level associated with a state can only increase. Given that each iteration either adds a new state to *states* or increases a level for some state already in *states* and that levels peak at $k + 1$, the number of iterations is bounded by the product of $k + 1$ and the size of the state space.

The inductive proof in Listings 1.1-1.4 and the termination arguments yield total correctness:

Theorem 1. *For finite transition system $S : (\bar{x}, I, T)$, the algorithm always terminates and returns true if and only if safety assertion P is invariant.*

3.4 Variations

Notice that `inductive` and `generate` (Listing 1.3) together generate a subclause of $\neg s$ that is inductive relative to F_i, where i is the greatest level for which $\neg s$ is itself inductive relative to F_i. It is actually possible to find the highest level $j \geq i$ for which $\neg s$ has a subclause that is inductive relative to F_j even if $\neg s$ is not itself inductive relative to F_j (that is, $j > i$). The difference between these two approaches is in whether the down function of [5] is ever applied to $\neg s$. In the method of `inductive` and `generate`, it is not; in the variation, it is.

While generalizing at higher levels is desirable, applying down to large clauses, such as $\neg s$, is the most expensive phase of inductive generalization in practice. On particularly large benchmarks with thousands of latches this phase can take prohibitively long; for example, on the `neclaftpX00X` benchmarks from HWMCC'08, this variation does not typically terminate in under 15 minutes.

One might wonder, therefore, if a weaker but faster inductive generalization procedure could be used. An obvious such procedure is the following: rather than using full induction, one could search for clauses that are established in the next state without assuming them as inductive hypotheses — in other words, perform a search for an implicate subclause (that is also inductive) rather than for an inductive subclause. Experiments indicate that using this generalization yields an overall model checker that is rarely faster and often significantly slower despite the superior speed of the individual generalizations. Of course, a positive spin on this disappointing result is that full induction is apparently a powerful generalization technique compared to searching for implicates.

4 Implementation and Experiments

4.1 Implementation

We implemented the algorithm using O'Caml for top-level reasoning, MiniSAT 2.0 for preprocessing the transition relation [9], and ZChaff for SAT-solving because of its incremental solving capability [16]. Notice that the SAT-solving libraries were available before 2008; thus, our performance on the HWMCC'08 benchmarks reported below cannot be attributed to superior SAT solvers.

Preprocessing. MiniSAT 2.0 provides an interface for "freezing" variables that should not be chosen for elimination during preprocessing. We use it to simplify the given transition relation once and for all [10]. Reducing the transition relation according to the cone-of-influence [8] followed by preprocessing yielded significant performance improvements for inductive generalization. It is likely that more sophisticated preprocessing would yield better performance.

Incremental SAT-Solving. Our technique requires solving hundreds to thousands of SAT problems per second in an incremental fashion. While MiniSAT 2.0 provides the ability to maintain context and change assumptions in the form of literals, only ZChaff, as far as we know, provides competitive SAT-solving combined with the ability to push and pop incremental context that includes sets of clauses. It is likely that a fully incremental version of a modern SAT solver would yield better performance.

Optimizations. Given that our algorithm relies on inductive generalization, we implemented a simple method to extract literal invariants that are obvious from the structure of the initial condition and transition relation. This optimization greatly improved performance on the `neclaftpX00X` benchmarks.

We implemented binary, rather than linear, search in the function `inductive`.

In our implementation of inductive generalization [5], we use a simple threshold to end the search for a minimal inductive subclause. If a certain number of randomly chosen literals (three in our implementation) are determined to be necessary to yield an inductive subclause, the search for a smaller inductive subclause ends. While minimality is no longer guaranteed, the resulting clauses are sufficiently strong (and probably minimal).

Finally, we implemented a VSIDS-like literal-ordering heuristic to guide which inductive clauses are discovered from a given cube [16]. Since a given clause can have many minimal inductive subclauses, the idea is to focus on those literals whose negations have appeared most frequently in examined states in recent history. Unfortunately, whether the heuristic has any benefit is unclear.

4.2 Experiments

The benchmarks and results from the Hardware Model Checking Competition 2008 provide a means of comparing different model checking algorithms [2]. We report our performance on these benchmarks.

We performed all experiments on a laptop equipped with an Intel Core 2 Duo 2.2 GHz processor, although only one core was used, and 4 GB of memory. In the HWMCC'08 competition, entries ran on Pentium IV 3 GHz processors with 2 GB of memory. After reading various online forums, we concluded that our processor provides a speed advantage of approximately 1.8× over the hardware used in the competition. Thus, rather than counting the number of benchmarks solved in under 900 seconds, we count only those solved in under 500 seconds.

Our implementation constructs proofs of unsatisfiability for 325 benchmarks in under 500 seconds and using at most 1.5 GB of memory, compared to the 314 solved by abc, the winner of the *unsat* division of the competition. Ten of these benchmarks were not solved during the competition. It finds counterexample traces in 234 cases, surprisingly competitive with BMC [3]. The top four entries for the satisfiable problems, all based on BMC, found 247, 243, 239, and 239 counterexamples, respectively. Our total number of solved problems is thus 559, seven more than abc, the winner of the overall competition.

Table 4.2 presents data for the 38 benchmarks that our implementation proved unsatisfiable in the allotted time (500 seconds) and memory (1.5 GB) that were solved by at most three competitors. The second column lists those competitors who solved the benchmark, their time in seconds (unscaled), and their peak memory consumption in MB. The third through sixth columns list our implementation's time in seconds *scaled by 1.8 to allow for better comparison*, memory consumption in MB, the number of thousands of SAT instances solved, and the number of the clauses in the proof, respectively. Again, the time for our implementation is multiplied by 1.8, so indicated runtime can be over 500 seconds despite our setting the timeout at 500 seconds.

In case the 1.8 scaling to compensate for different processors is considered too low, the results for 3.0 scaling are the following: 317 proofs and 228 counterexamples, with 545 benchmarks solved overall.

5 Related Work

SAT-based unbounded model checking was the first symbolic model checking approach based on generating clauses [14]. It discovers implicates to generalize states leading to property violations. The overall iterative structure is the same as standard symbolic model checking. In our algorithm, induction is a means not only for generalizing from states but also for abstracting the system based on the property, allowing the analysis of large transition systems.

Our algorithm can be seen as an instance of predicate abstraction/refinement [12,7] in that the minor iterations generate new predicates (clauses) while the major iterations propagate them. If the clauses are insufficient for convergence to an inductive strengthening assertion, the next minor iteration generates additional clauses that allow propagation to continue at least one additional step.

The k-step over-approximation structure of $F_0, F_1, F_2, \ldots, F_k$ is similar to that of interpolation-based model checking (ITP) [15], which uses an interpolant from an unsatisfiable K-step BMC query to compute the post-image approximately. All states in the image are at least $K-1$ steps away from violating the property. A larger K refines the image by increasing the minimum distance to violating states. In our algorithm, if the frontier is at level k, then F_i, for $0 \leq i \leq k$, represents states that are at least $k-i$ steps from violating the property. As k increases, the minimum number of steps from F_i-states to violating states increases. In both cases, increasing k (in ours) or K (in ITP) sufficiently for a correct system yields an inductive assertion. However, the algorithms differ in their underlying “technology”: ITP computes interpolants from K-step

Table 2. Solved benchmarks that were solved in HWMCC'08 by at most three solvers

Benchmark	**Solved by** (solver/sec/MB)	**Sec**	**MB**	**SC**(k)	**|Proof|**
bjrb07amba6andenv	abc/309/166 pdtravbdd/19/61	462	364	11	269
bjrb07amba7andenv	abc/203/180 pdtravbdd/242/71	169	253	7	221
intel006	pdtravitp/348/143 tipidi/367/425	32	79	28	931
intel007	pdtravcbq/881/185	541	228	76	2906
intel026		261	277	96	1335
intel037		207	786	2	157
intel054	tipidi/2/8 tipids/2/8 tipind/2/8	414	174	271	4544
intel055	tipidi/43/12 tipids/43/12	39	95	30	615
intel056	tipidi/7/13 tipids/8/13	91	79	93	1597
intel057	tipidi/2/6 tipids/2/6 tipind/2/6	176	129	142	2332
intel059	tipidi/4/8 tipids/4/8	46	74	53	982
neclabakery001	aigtrav/14/95 pdtravbdd/18/54 tipind/422/34	156	233	417	2755
neclaftp1001		84	781	1	669
neclaftp1002		284	1417	3	707
neclaftp2001	tipidi/839/122 tipids/838/122 tipind/834/123	43	466	1	638
neclaftp2002	tipind/898/175	248	816	3	644
neclatcas1a001	tipidi/0/0 tipids/0/0 tipind/0/0	3	56	1	86
neclatcasall001	tipidi/0/0 tipids/0/0 tipind/0/0	45	97	20	279
nusmvbrp	pdtravbdd/456/74 pdtravcbq/187/283	21	50	56	688
nusmvguidancep2	pdtravbdd/478/61 tipidi/873/394	23	78	16	164
nusmvguidancep5	pdtravbdd/59/44	16	70	10	121
nusmvguidancep6	abc/34/35 pdtravbdd/54/44 pdtravitp/92/177	10	69	8	97
pdtvisbakery0	abc/21/97 pdtravbdd/28/60	113	164	36	215
pdtvisbakery1	abc/96/97 pdtravbdd/44/61	144	182	44	308
pdtvisbakery2	abc/57/95 pdtravbdd/114/64	136	191	42	371
pdtvisgoodbakery0	abc/45/98 pdtravbdd/57/64	203	202	65	601
pdtvisgoodbakery1	abc/102/95 pdtravbdd/51/63	142	175	46	458
pdtvisgoodbakery2	abc/118/97 pdtravbdd/49/60	153	193	47	372
pdtvisns3p00		244	138	120	1709
pdtvisns3p01	pdtravcbq/618/266 tipids/670/145	352	131	152	2287
pdtvisns3p02		196	129	100	1169
pdtvisns3p03		230	121	106	1398
pdtvisns3p04		550	115	207	2187
pdtvisns3p06	pdtravcbq/823/278	837	164	289	2845
pdtvisns3p07		311	131	145	1453
pdtvisrethersqo4	abc/23/15	162	157	341	3394
pdtvissoap1	pdtravitp/384/520	70	102	42	807
pdtvissoap2		108	101	65	1789

BMC queries, while our algorithm uses inductive generalization of cubes, which requires only 1-step BMC queries for arbitrarily large k.

Various approaches to generalizing counterexamples to k-induction have been explored [17,1,19]. Our work could in principle be applied as a method of strengthening k-induction. However, the technique already works well on its own and has the distinct advantage of posing small SAT problems.

Finally, we draw on our previous work on inductive generalization [5]. This paper contributes k-step relative inductive generalization, which guarantees that all examined cubes can be inductively generalized if the property is invariant.

6 Conclusion

The empirical data suggest the effectiveness of k-step relative inductive generalization, a technique unlike — and therefore complementary to — other symbolic model checking methods. The most exciting direction for our ongoing research is to parallelize the algorithm. Our earlier work on inductive generalization was easily parallelized and sometimes yielded near-linear scaling with the number of nodes on hard benchmarks [4]. The new algorithm, although more complex in structure, should be similarly parallelizable since the implementation spends the majority of its time generating clauses incrementally.

BMC is faster than our implementation at finding counterexample traces. We plan to investigate a combination of our algorithm with BMC in which generated clauses would constrain the SAT search space.

Another direction for research is to apply the idea of finding k-step relative inductive generalizations of states in an infinite-state setting.

References

1. Awedh, M., and Somenzi, F. Automatic invariant strengthening to prove properties in bounded model checking. In *DAC* (2006), ACM Press, pp. 1073–1076.
2. Biere, A., Cimatti, A., Claessen, K., Jussila, T., McMillan, K., and Somenzi, F. Hardware model checking competition, 2008.
3. Biere, A., Cimatti, A., Clarke, E. M., and Zhu, Y. Symbolic model checking without BDDs. In *TACAS* (London, UK, 1999), Springer-Verlag, pp. 193–207.
4. Bradley, A. R. *Safety Analysis of Systems.* PhD thesis, Stanford University, May 2007.
5. Bradley, A. R., and Manna, Z. Checking safety by inductive generalization of counterexamples to induction. In *FMCAD* (2007).
6. Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. Symbolic model checking: 10^{20} states and beyond. In *LICS* (1990), pp. 428–439.
7. Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM 50*, 5 (2003), 752–794.
8. Clarke, E., Grumberg, O., and Peled, D. *Model Checking.* MIT Press, 2000.
9. Eén, N., and Biere, A. Effective preprocessing in SAT through variable and clause elimination. In *SAT* (2005), vol. 3569, Springer, pp. 61–75.
10. Eén, N., Mishchenko, A., and Sörensson, N. Applying logic synthesis for speeding up SAT. In *SAT* (2007), pp. 272–286.
11. Floyd, R. W. Assigning meanings to programs. In *Symposia in Applied Mathematics* (1967), vol. 19, American Mathematical Society, pp. 19–32.
12. Graf, S., and Saidi, H. Construction of abstract state graphs with PVS. In *CAV* (June 1997), O. Grumberg, Ed., vol. 1254 of *LNCS*, Springer, pp. 72–83.
13. Hoare, C. A. R. An axiomatic basis for computer programming. *Communications of the ACM 12*, 10 (October 1969), 576–580.
14. McMillan, K. L. Applying SAT methods in unbounded symbolic model checking. In *CAV* (2002), vol. 2404 of *LNCS*, Springer-Verlag, pp. 250–264.
15. McMillan, K. L. Interpolation and SAT-based model checking. In *CAV* (2003), vol. 2725 of *LNCS*, Springer, pp. 1–13.
16. Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. Chaff: Engineering an Efficient SAT Solver. In *DAC* (2001).
17. Moura, L. D., Ruess, H., and Sorea, M. Bounded model checking and induction: From refutation to verification. In *CAV* (2003), Springer-Verlag, pp. 14–26.
18. Sheeran, M., Singh, S., and Stålmarck, G. Checking safety properties using induction and a sat-solver. In *FMCAD* (2000), pp. 127–144.

19. Vimjam, V. C., and Hsiao, M. S. Fast illegal state identification for improving SAT-based induction. In *DAC* (2006), ACM Press, pp. 241–246.