Bounded Model Checking and Induction:
From Refutation to Verification *

(Extended Abstract, Category A)

Leonardo de Moura, Harald Rueß, and Maria Sorea**

SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, CA 94025, USA
{demoura, rues, sorea}@csli.sri.com
http://www.csli.sri.com/

Abstract. We explore the combination of bounded model checking and
induction for proving safety properties of infinite-state systems. In particu-
lar, we define a general $k$-induction scheme and prove completeness
thereof. A main characteristic of our methodology is that strengthened
invariants are generated from failed $k$-induction proofs. This strengthen-
ing step requires quantifier-elimination, and we propose a lazy quantifier-
elimination procedure, which delays expensive computations of disjunc-
tive normal forms when possible. The effectiveness of induction based on
bounded model checking and invariant strengthening is demonstrated
using infinite-state systems ranging from communication protocols to
timed automata and (linear) hybrid automata.

1 Introduction

Bounded model checking (BMC) [4, 3, 6] is often used for refutation, where one
systematically searches for counterexamples whose length is bounded by some
integer $k$. The bound $k$ is increased until a bug is found, or some pre-computed
completeness threshold is reached. Unfortunately, the computation of complete-
ess thresholds is usually prohibitively expensive and these thresholds may be
too large to effectively explore the associated bounded search space. In addition,
such completeness thresholds do not exist for many infinite-state systems.

In deductive approaches to verification, the invariance rule is used for es-
establishing invariance properties $\varphi$ [10, 9, 12, 2]. This rule requires a property $\psi$
which is stronger than $\varphi$ and inductive in the sense that all initial states satisfy
$\psi$ and $\psi$ is preserved under each transition. Theoretically, the invariance rule is
adequate for verifying a valid property of a system, but its application usually

** Also affiliated with University of Ulm, Germany.
requires creativity in coming up with a sufficiently strong inductive invariant. It is also nontrivial to detect bugs from failed induction proofs.

In this paper, we explore the combination of BMC and induction based on the \textit{k-induction} rule. This induction rule generalizes BMC in that it requires demonstrating the invariance of \( \varphi \) in the first \( k \) states of any execution. Consequently, error traces of length \( k \) are detected. This induction rule also generalizes the usual invariance rule in that it requires showing that if \( \varphi \) holds in every state of every execution of length \( k \), then every successor state also satisfies \( \varphi \). In its pure form, however, \( k \)-induction does not require the invention of a strengthened inductive invariant. As in BMC, the bound \( k \) is increased until either a violation is detected in the first \( k \) states of an execution or the property at hand is shown to be \( k \)-inductive. In the ideal case of attempting to prove correctness of an inductive property, 1-induction suffices and iteration up to a possibly large, complete threshold, as in BMC, is avoided. The \( k \)-induction rule is sound, but further conditions, such as the restriction to acyclic execution sequences, must be added to make \( k \)-induction complete even for finite-state systems [15].

One of our main contributions is the definition of a general \( k \)-induction rule and a corresponding completeness result. This induction rule is parameterized with respect to suitable notions of simulation. These simulation relations induce different notions of path \textit{compression} in that an execution path is compressed if it does not contain two similar states. Many completeness results, such as \( k \)-induction for timed automata, follow by simply instantiating this general result with the simulation relation at hand. For general transition systems, we develop an \textit{anytime} algorithm for approximating adequate simulation relations for \( k \)-induction.

Whenever \( k \)-induction fails to prove a property \( \varphi \), there is a counterexample of length \( k + 1 \) such that the first \( k \) states satisfy \( \varphi \) and the last state does not satisfy \( \varphi \). If the first state of this trace is reachable, then \( \varphi \) is refuted. Otherwise, the counterexample is labeled \textit{spurious}. By assuming the first state of this trace is unreachable, a spurious counterexample is used to automatically obtain a strengthened invariant. Many infinite-state systems can only be proven with \( k \)-induction enriched with invariant strengthening, whereas for finite systems the use of strengthening decreases the minimal \( k \) for which a \( k \)-induction proof succeeds.

Since our invariant strengthening procedure for \( k \)-induction heavily relies on eliminating existentially quantified state variables, we develop an effective quantifier elimination algorithm for this purpose. The main characteristic of this algorithm is that it avoids a potential exponential blowup in the initial computation of a disjunctive normal form whenever possible, and a constraint solver is used to identify relevant conjunctions. In this way the paradigm of \textit{lazy} theorem proving, as developed by the authors for the ground case [6], is extended to first-order formulas.

The paper is organized as follows. Section 2 contains background material on encodings of transition systems in terms of logic formulas. In Section 3 we develop the notions of reverse and direct simulations together with an anytime
algorithm for computing these relations. Reverse and direct simulations are used in Section 4 to state a generic $k$-induction principle and to provide sufficient conditions for the completeness of these inductions. Sections 5 and 6 discuss invariant strengthening and lazy quantifier elimination. Experimental results with $k$-induction and invariant strengthening for various infinite-state protocols, timed automata, and linear hybrid systems are summarized in Section 7. Comparisons to related work are in Section 8.

2 Background

Let $V := \{x_1, \ldots, x_n\}$ be a set of variables interpreted over nonempty domains $D_1$ through $D_n$, together with a type assignment $\tau$ such that $\tau(x_i) = D_i$. For a set of typed variables $V$, a variable assignment is a function $\nu$ from variables $x \in V$ to an element of $\tau(x)$. The variables in $V := \{x_1, \ldots, x_n\}$ are also called state variables, and a program state is a variable assignment over $V$.

All the developments in this paper are parametric with respect to a given constraint theories $C$, such as linear arithmetic or a theory of bitvectors. We assume a computable function for deciding satisfiability of a conjunction of constraints in $C$. A set of Boolean constraints, $\text{Bool}(C)$, includes all constraints in $C$ and is closed under conjunction $\land$, disjunction $\lor$, and negation $\neg$. Effective solvers for deciding the satisfiability problem in $\text{Bool}(C)$ have been previously described [6,5].

A tuple $(V, I, T)$ is a C-program over $V$, where interpretations of the typed variables $V$ describe the set of states, $I \in \text{Bool}(C(V))$ is a predicate that describes the initial states, and $T \in \text{Bool}(C(V \cup V'))$ specifies the transition relation between current states and their successor states ($V$ denotes the current state variables, while $V'$ stands for the next state variables). The semantics of a program is given in terms of a transition system $M$ in the usual way.

For a program $M = (V, I, T)$, a sequence of states $\pi(s_0, s_1, \ldots, s_n)$ forms a path through $M$ if $\bigwedge_{0 \leq i < n} T(s_i, s_{i+1})$. A state $s$ is reachable in $M$ if there is a path $\pi(s_0, s_1, \ldots, s_{n-1}, s)$ through $M$ and $I(s_0)$, and a state property $\varphi \in C(V)$ is invariant in $M$ iff $\varphi(s)$ holds for every reachable state $s$ in $M$. A counterexample for a property $\varphi$ is a path $\pi(s_0, \ldots, s_n)$ such that $I(s_0)$ and $\neg \varphi(s_n)$, and the length $\text{len}(\pi)$ of such a counterexample is given by the number of states in this path.

Typical programming constructs can be rewritten into the program syntax presented above. For example, Dijkstra’s guarded commands are encoded in terms of a disjunction of conjunctions of guards $g(x_1, \ldots, x_n)$ and updates $x'_i = f_i(x_1, \ldots, x_n)$ for all variables $x_i$. Programs with external, non-deterministic inputs are defined by partitioning the set of variables $X$ into the input variables $\text{input}(X)$, which are unconstrained, and the other state variables, whose next-state values are constrained by the transition relation.

Throughout this paper we use timed automata [1], which are state-transition graphs augmented with a finite set of real-valued clocks, as a prototypical class of infinite-state systems. Decidability of the model-checking problem for timed
automata rests on the fact that the space of clock valuations is partitioned into finitely many clock regions. Two clock valuations \( v_1, v_2 \) that belong to the same region are (region) equivalent, denoted as \( v_1 \sim_{TA} v_2 \). This region equivalence is a stable quotient relation, that is, whenever \( q \sim_{TA} u \) and \( T(q, q') \), there exists a state \( u' \) such that \( T(u, u') \) and \( q' \sim_{TA} u' \) [1]. Encoding of timed automata in terms of logical programs with linear arithmetic constraints are described in [17]. In particular, program states consist of a location and nonnegative real interpretations of clocks. For timed automata we restrict ourselves to proving so-called clock constraints \( \varphi \), such that \( q \sim_{TA} u \) implies that \( \varphi(q) \) iff \( \varphi(u) \).

3 Direct and Reverse Simulation

The notions of direct and reverse simulation as developed here lay out the foundation for the completeness results in Section 4.

**Definition 1 (Direct / Reverse Simulation).** Let \( M = (V, I, T) \) be a program and \( \varphi \) a state formula over \( V \). We define the functors \( F_d \) and \( F_r \) that map binary relations \( R \) over \( V \) in the following way:

\[
F_d(R)(s_1, s_2) := \begin{cases} \text{if } \neg \varphi(s_1) \text{ then } \neg \varphi(s_2) \\ \text{else } \forall s_1'. \ T(s_1, s_1') \Rightarrow \exists s_2'. \ R(s_1', s_2') \land T(s_2', s_2) \end{cases}
\]

\[
F_r(R)(s_1, s_2) := \begin{cases} \text{if } I(s_1) \text{ then } I(s_2) \\ \text{else } \forall s_1'. \ T(s_1', s_1) \Rightarrow \exists s_2'. \ R(s_1', s_2') \land T(s_2', s_2) \end{cases}
\]

A direct (reverse) simulation over \( V \) with respect to \( \varphi \) is any binary relation \( \preceq \) over \( V \) that satisfies \( \preceq \subseteq F_d(\preceq) \) (\( \preceq \subseteq F_r(\preceq) \)).

In contrast to reverse simulations, direct simulations depend on a state formula \( \varphi \). Also, the definition of direct simulation is inspired by the notion of stable relations above. Direct (reverse) simulations are usually denoted by \( \preceq_d \) (\( \preceq_r \)). The following direct and reverse simulation are used as running examples throughout the paper.

**Example 1.** The empty relation \( a \preceq_d b := \text{false} \) is a direct and a reverse simulation.

**Example 2.** Equality \( (=) \) between states is both a direct and a reverse simulation.

**Example 3.** The relation \( s_1 \preceq_T s_2 := I(s_1) \land I(s_2) \) is a reverse simulation, where \( I \) is the predicate for describing the set of initial states of the given program.

**Example 4.** Now, consider programs \( (V, I, T) \) with inputs such that \( \text{input}(x) \) holds iff \( x \) is an input variable. The relation

\[
s_1 =_i s_2 := \text{for all variables } x \in V \cdot \text{input}(x) \lor s_1(x) = s_2(x),
\]
with \( s(x) \) denoting the value of the variable \( x \) in the state \( s \), is a reverse simulation, since the values of the input variables are not constrained by the predicate \( I \) and their next values are not constrained by \( T \). Obviously, for transition systems with inputs, the relation \( s_1 =_i s_2 \) is stronger than \( = \), and therefore gives rise to shorter paths.

**Example 5.** We now consider timed automata programs and clock constraints. The region equivalence \( \sim_{\tau A} \), which give rise to finitely many clock regions, is stable, and therefore a direct simulation.

The notions of direct and reverse simulation are modular in the sense that the union of direct (reverse) simulations is also a direct (reverse) simulation.

**Proposition 1 (Modularity).** If \( \preceq_1 \) and \( \preceq_2 \) are direct (reverse) simulations, then \( \preceq_1 \cup \preceq_2 \) is also a direct (reverse) simulation.

This property follows directly from the definitions of direct (reverse) simulations in Definition 1 and from the monotonicity of the functors \( F_d \) and \( F_r \). For example, the reverse simulations \( \preceq_{d} \) and \( =_i \) in Examples 3 and 4 may be combined to obtain a new reverse simulation.

Given an arbitrary program \( M = (V, I, T) \) and a property \( \varphi \), the associated largest direct (reverse) simulation relation \( \preceq_{D} \) (\( \preceq_{R} \)) is obtained as the greatest fixpoint of the functor \( F_d \) (\( F_r \)) in Definition 1. These fixpoints exist, since \( F_d \) and \( F_r \) are monotonic. However, the fixpoint iterations are often prohibitively expensive, and a direct (reverse) simulation is only obtained on convergence of the iteration. The iteration in Proposition 2 provides a viable alternative in that a reverse (direct) simulation is refined to obtain a stronger reverse (direct) simulation. The proof of the proposition below follows from the definitions of reverse (direct) simulations, from the monotonicity of the functors \( F_r \) (\( F_d \)), and from modularity (Proposition 1).

**Proposition 2 (Anytime Iteration).** If \( \preceq_r \) (\( \preceq_d \)) is a reverse (direct) simulation, then for all \( n \geq 0 \) the relation \( \preceq_{r,n} \) (\( \preceq_{d,n} \)) is also a reverse (direct) simulation:

\[
\begin{align*}
\preceq_{r,0} := & \preceq_r \\
\preceq_{r,n} := & \preceq_{r,n-1} \cup F_r(\preceq_{r,n-1}) \\
\preceq_{d,0} := & \preceq_d \\
\preceq_{d,n} := & \preceq_{d,n-1} \cup F_d(\preceq_{d,n-1})
\end{align*}
\]

Consequently, this iteration gives rise to an anytime algorithm for computing direct (reverse) simulations, and equality \( = \), for example, may be used as seed, since it is both a direct and a reverse simulation (see Example 2).

4 Completeness of \( k \)-Induction

Given the notions of direct and reverse simulations, we develop sufficient conditions for proving completeness of \( k \)-induction. These results are based on restricting paths to not contain states equivalent with respect to a given direct or reverse simulation. For direct (reverse) simulations we define a compressed
path w.r.t. to the given direct (reverse) simulation as a path $\pi(s_0, s_1, \ldots, s_n)$ not containing any $s_i, s_j$ with $j < i$ ($i < j$) such that $s_i$ directly (reversely) simulates $s_j$.

**Definition 2 (Path Compression).**

- A path $\pi_{d}(s_0, s_1, \ldots, s_n)$ is *compressed* w.r.t. the direct simulation $\preceq_d$ if:
  $$\pi_{d}(s_0, s_1, \ldots, s_n) := \pi(s_0, s_1, \ldots, s_n) \land \bigwedge_{0 \leq j < n} s_i \not\prec_d s_j.$$  

- A path $\pi_{r}(s_0, s_1, \ldots, s_n)$ is *compressed* w.r.t. the reverse simulation $\preceq_r$ if:
  $$\pi_{r}(s_0, s_1, \ldots, s_n) := \pi(s_0, s_1, \ldots, s_n) \land \bigwedge_{0 \leq i < n} s_i \not\prec_r s_j.$$  

A path that is compressed with respect to the reverse and the direct simulations $\preceq_r$ and $\preceq_d$ is denoted by $\pi_{s-d}$.

For example, a path $\pi(s_0, \ldots, s_n)$ is compressed w.r.t. the reverse simulation (=) from Example 2 iff it is acyclic. Moreover, given the reverse simulation $\preceq_r$ from Example 3, a path $\pi(s_0, \ldots, s_n)$ is compressed w.r.t. $\preceq_r$ iff it contains at most one initial state. Obviously, for transition systems with inputs, the relation (=) (see Example 4) is stronger than (=), and therefore give rise to shorter compressed paths. We have collected all ingredients for defining $k$-induction for arbitrarily compressed paths.

**Definition 3 ($k$-Induction).** Let $M = (V, I, T)$ be a program, $k$ an integer, $\preceq_r$ a reverse simulation, and $\preceq_d$ a direct simulation. The induction scheme of depth $k$, $\text{IND}^{\preceq_r \cdot \preceq_d}(k)$ allows one to deduce the invariance of $\varphi$ in $M$ if the following holds.

- $I(s_0) \land \pi_{d}(s_0, \ldots, s_{k-1}) \Rightarrow \varphi(s_0) \land \ldots \land \varphi(s_{k-1})$
- $\varphi(s_n) \land \ldots \land \varphi(s_{n+k-1}) \land \pi_{d}(s_n, \ldots, s_{n+k}) \Rightarrow \varphi(s_{n+k})$

For example, given the empty relationship $\preceq_0$ from Example 1, $\text{IND}^{\preceq_r}$ reduces to the naive, incomplete $k$-induction on arbitrary paths. Consider, for example, the system in Figure 1 and a property $\varphi$ which is assumed to hold only in $q_1$. Now, the execution sequence $q_3 \leadsto q_3 \leadsto \ldots \leadsto q_1 \leadsto q_1$ is not $k$-inductive, but it is ruled out under the acyclic path restriction. The complete $k$-induction
schemes in [15], which consider only acyclic paths and paths that only visit initial states once can be recovered by instantiating Definition 3 with the relations (=) (Example 2) and (\leq_{T}) (Example 3), respectively. Since both (=) and (\leq_{T}) are reverse simulations, an induction scheme restricted to acyclic paths visiting initial states at most once is obtained by modularity (Proposition 1).

Completeness of k-induction relies heavily on the notion of path compression. We now state the main lemma.

**Lemma 1 (Compressing non-\pi^{\leq_{r}} paths).** Let \pi(s_{0}, \ldots, s_{n}) be a given path; then:

1. There exists a \pi^{\leq_{r}}-compressed path \pi^{\leq_{r}}(q_{0}, \ldots, q_{m}) such that q_{m} = s_{n} and \(m \leq n\).
2. There exists a \pi^{\leq_{s}}-compressed path \pi^{\leq_{s}}(q_{0}, \ldots, q_{m}), such that q_{0} = s_{0} and \(m \leq n\).

**Proofsketch.** Assume a path \pi(s_{0}, \ldots, s_{n}), which is not compressed w.r.t. \leq_{r}. By Definition 1 it follows that there are states \(s_{i}, s_{j} \in \pi(s_{0}, \ldots, s_{n})\) such that \(s_{i} \leq_{r} s_{j}\), and \(i < j\). We distinguish two cases. First, if \(s_{i}\) is an initial state, then so is \(s_{j}\), and therefore a shorter path \pi(s_{j}, \ldots, s_{n}) is obtained as a counterexample. Second, if \(s_{i}\) is not an initial state, then \(s_{i} \neq s_{0}\), and there exists a \(s'_{i} \neq s_{0}\) such that \(T(s'_{i}, s_{i})\). Since \(s'_{i} \leq_{r} s_{j}\) it follows by Definition 1 that there is a state \(s'_{i} \leq_{T} s'_{j}\), such that \(s'_{i} \leq_{r} s'_{j}\) and \(T(s'_{j}, s_{j})\). If \(s'_{i}\) is initial state, then so is \(s'_{j}\), and since \(i < j\) a shorter path \pi^{\leq_{r}}(s'_{j}, s_{j}, \ldots, s_{n}) is obtained. If \(s'_{i}\) is not initial, by repeating the above argument a shorter path is constructed. In both cases a shorter path is obtained, if such path is not a compressed path, then it is further reduced. The proof for \pi^{\leq_{s}}-compressed paths works analogously.

IND^{\leq_{r}}(k) is complete if \(\varphi\) is an invariant of \(M\) iff there is a \(k\) such that IND^{\leq_{r}}(k)(\varphi). Now, completeness of k-induction follows from the main lemma 1 above.

**Theorem 1 (Completeness).** IND^{\leq_{r}}(k) is a complete proof method iff there is an upper bound on the length of the paths \pi^{\leq_{r}}(s_{0}, \ldots, s_{n}).

Using the simulation from Example 2, Theorem 1 is instantiated to obtain the following complete k-induction for finite-state systems.

**Corollary 1.** Let \(M\) be a finite-state program over \(V\) and \(\varphi\) a state property in \(V\); then IND^{\leq_{r}}(k) induction is complete.

In general, k-induction for (=) is not complete for infinite-state systems. Consider, for example, the program \(M = (I, T)\) over the integer state variable \(x\) with \(I = (x = 0)\) and \(T = (x' = x + 2)\), and the formula \(x \neq 3\). Obviously, it is the case that \(x \neq 3\) is invariant in \(M\), but there exists no \(k \in \mathbb{N}\) such that the property is proven by IND^{\leq_{r}}(k). However, k-induction is complete for timed automata, since the equivalence relation \(\sim_{TA}\) is a direct simulation (Example 5), and an upper bound on the length of the paths \pi^{\sim_{TA}}(s_{0}, \ldots, s_{n}) is given by the number of clock regions.
Corollary 2. Let $M$ be a timed automata program over the clock evaluations $C$ and $\varphi$ a clock constraint in $C$; then $\text{IND}^{-\tau}(k)$ induction is complete.

Similar results are obtained for other direct and reverse simulations and combinations thereof.

5 Invariant Strengthening

Whenever $k$-induction fails to prove a property $\varphi$, there is a counterexample $\pi = s_n, s_{n+1}, \ldots, s_{n+k}$ such that the first $k$ states satisfy $\varphi$ whereas the last state $s_{n+k}$ does not satisfy this property. If $s_n$ is indeed reachable, then $\varphi$ is not invariant. Otherwise, the counterexample is labeled as spurious and it is inconclusive whether $\varphi$ is invariant or not. However, by assuming $s_n$ to be unreachable, such a spurious counterexample is used to obtain a strengthened invariant $\varphi \land \neg(s_n)$.

Consider, for example, the property $\neg(q_1)$ for the system in Figure 1. Induction of depth $k = 1$ fails, and the counterexample $q_2 \sim q_1$ is obtained. Now, $\neg(q_1)$ is strengthened to obtain $\neg(q_1) \land \neg(q_3)$, which is proven using 1-induction. More generally, whenever the induction step of $\text{IND}^{-\tau}(k)$ fails, the formula

$$Q(s_n, \ldots, s_{n+k}) := \varphi(s_n) \land \ldots \land \varphi(s_{n+k-1}) \land \pi^{-\tau}(s_n, \ldots, s_{n+k}) \land \neg\varphi(s_{n+k})$$

is satisfiable, and each satisfying assignment describes a counterexample for the induction step. Thus, we define the predicate $U(s)$ for representing the set of possibly unreachable states, which may reach the bad state in $k$ steps by means of a $\pi^{-\tau}$ path.

$$U(s) = \exists s_{n+1}, \ldots, s_{n+k} \cdot Q(s, \ldots, s_{n+k})$$

Now, $\varphi$ is strengthened as $\varphi \land \neg U(s)$, and quantifier elimination is used for transforming this strengthened formula into an equivalent Boolean constraint formula. For the general case, we use the quantifier elimination procedure in Section 6. Notice, however, that for special cases such as guarded command languages, the quantifiers in $U(s)$ are eliminated using purely syntactic operations such as substitution, since all quantifications are over “next-state” variables $x$ for which there are explicit solutions $f(x)$. An example might help to illustrate the combination of $k$-induction, strengthening, and quantifier elimination.

Example 6. Consider the usual stripped-down version of Lamport’s Bakery protocol in Figure 2 with the initial value 0 for both counters $y1$ and $y2$ and the mutual exclusion property $MX$ defined by $\neg(x1 = a3 \land y2 = b3)$. We apply 3-induction with the empty simulation relation $\preceq_0$. The base step holds and the induction step fails to obtain

$$U(s_n) := \exists s_{n+1}, s_{n+2}, s_{n+3} \cdot$$

$$MX(s_n) \land MX(s_{n+1}) \land MX(s_{n+2}) \land$$

$$\pi^{-\tau}(s_n, s_{n+1}, s_{n+2}, s_{n+3}) \land \neg MX(s_{n+3})$$
Fig. 2. Bakery Mutual Exclusion Protocol.

with states $s_i$ of the form $(pc_i, y_1, pc_2, y_2)$. Since the transitions of the Bakery protocol are in terms of guarded commands, simple substitution is used to obtain a quantifier-eliminated form.

$$R(s) := (pc1 = a1 \land pc2 = b2 \land y2 = 0) \lor (pc1 = a2 \land pc2 = b1 \land y1 = 0)$$

Now, the strengthened property $MX(s) \land \neg R(s)$ is proven using 3-induction.

## 6 Quantifier elimination

Given a quantified formula $\exists \text{vars. } \varphi$ with $\varphi \in \text{Bool}(C)$, quantifier-elimination procedures usually work by transforming $\varphi$ into disjunctive normal form (DNF) and distributing the existential quantifiers over disjunctions. Thus, one is left with eliminating quantifiers from a set of existentially quantified conjunctions of literals. We assume as given such a procedure $C$-qe. The main drawback of these procedures is that there is a potential exponential blowup in the initial transformation to DNF and $C$-qe might even return further disjunctions (as is the case for Presburger arithmetic).

The quantifier elimination problem for invariant strengthening, as discussed in Section 5, however, allows for a purely syntactic quantifier elimination as long as we are restricting ourselves to guarded command programs. In these cases, $C$-qe just applies the substitution rule $(x \not\in \text{vars}(\psi))$

$$(\exists x. (x = \psi) \land \varphi(x)) \iff \varphi(\psi);$$

possibly followed by simplification. Another $C$-qe function is used in McMillan's [13] quantifier elimination algorithm based on propositional SAT solving, in that his $C$-qe$(\text{vars}, c)$ simply deletes the literals in $c$, which contain a variable in $\text{vars}$.\(^1\)

However, the initial DNF computation should usually be avoided when possible. Given a set of existentially quantified variables $\text{vars}$ and a quantifier-free formula $\varphi$ in $\text{Bool}(C)$, the algorithm $qe(\text{vars}, \varphi)$ in Figure 3 returns a formula in $\text{Bool}(C)$ which is equivalent to $\exists \text{vars. } \varphi$. The procedure $qe$ relies on a satisfiability solver for formulas $\varphi \in \text{Bool}(C)$, which is assumed to enumerate representations of sets of satisfiable models in terms of conjunctions of literals in $\varphi$. Such a solver

\(^1\) Actually, McMillan considers the dual problem of eliminating universal quantifications from a conjunctive normal form.
procedure $\text{qe}(\text{vars}, \varphi)$

\begin{verbatim}
$\psi := false$

\textbf{loop}
$c := \text{next-solution}(\varphi)$
\textbf{if $c := false$ then return $\psi$}
$c' := \text{C-qe}(\text{vars}, c)$
$\psi := \psi \lor c'$
$\varphi := \varphi \land \neg c'$
\end{verbatim}

**Fig. 3.** Lazy Quantifier Elimination.

is described, for example, in [6, 5]. These solutions are supposed to be enumerated by successive calls to $\text{next-solution}$ in Figure 3. Since there are only a finite number of solutions in terms of subsets of literals, the function $\text{qe}$ is terminating. Moreover, minimal solutions or good over-approximations thereof, as produced by the lazy theorem proving algorithm [6, 5], accelerate convergence.

The variable $c$ in Figure 3 stores the current solution obtained by $\text{next-solution}$, and the procedure $\text{C-qe}$ applies quantifier elimination for conjunction. In many cases, $\text{C-qe}$ just applies the substitution rule to remove quantified variables. In order to obtain the next set of solutions, we rule out the current solutions by updating $\varphi$ with the value $\neg c'$ instead of $\neg c$, since $\neg c'$ is more restrictive.

Thus, the quantifier elimination procedure in Figure 3 avoids eager computation of a disjunctive normal form. Moreover, a solver for $\text{Bool}(C)$ is used to guide the search for relevant “conjunctions” in $\varphi$. In this way, the $\text{qe}$ algorithm extends the lazy theorem proving paradigm described in [6, 5] to the case of first-order reasoning.

**Example 7.** Consider

$$
\exists x_1, y_1 (x_0 = 1 \lor x_0 = 3 \lor y_0 > 1) \land x_1 = x_0 - 1 \land y_1 = y_0 + 1 \\
\lor ((x_0 = -1 \lor x_0 = -3) \land x_1 = x_0 + 2 \land y_1 = y_0 - 1)) \land x_1 < 0
$$

A first satisfiable conjunction of literals is obtained by, say

$$c := y_0 > 1 \land x_1 = x_0 - 1 \land y_1 = y_0 + 1 \land x_1 < 0.$$

Now, application of the substitution rule yields

$$c' := y_0 > 1 \land x_0 - 1 < 0$$

and, after updating $\varphi$ with $\neg c'$ a second solution is obtained as

$$c := x_0 = -3 \land x_1 = x_0 + 2 \land y_1 = y_0 - 1 \land x_1 < 0.$$

Again, applying the substitution rule, one gets $c' := x_0 = -3 \land x_0 + 2 < 0$, and, since there are no further solutions, the quantifier-eliminated formula is

$$[y_0 > 1 \land x_0 - 1 < 0] \lor (x_0 = -3 \land x_0 + 2 < 0)$$
7 Experiments

We describe some of our experiments with $k$-induction and invariant strengthening. Our benchmark examples include infinite-state systems such as communication protocols, timed automata and linear hybrid systems.  \cite{5}. In particular, Table 1 contains experimental results for the Bakery protocol as described earlier, Simpson's protocol \cite{13} to avoid interference between concurrent reads and writes in a fully asynchronous system, well-known timed automata benchmarks such as the train gate controller and Fischer's mutual exclusion protocol, and three linear hybrid automata benchmarks for water level monitoring, the leaking gas burner, and the multi-rate Fischer protocol. Timed automata and linear hybrid systems are encoded as in \cite{14}. Starting with $k=1$ we increase $k$ until $k$-induction succeeds. We are using invariant strengthening only in cases where syntactic quantifier elimination based on substitution suffices. In particular, we do not use strengthening for the timed and hybrid automata examples, that is, $\mathcal{C}$-	extit{ge} tries to apply the substitution rule, if the resulting satisfiability problems for Boolean combinations of linear arithmetic constraints are solved using the lazy theorem proving algorithm described in \cite{15} and implemented in the ICS decision procedures \cite{16}.

<table>
<thead>
<tr>
<th>System Name</th>
<th>Proved with $k$</th>
<th>Time</th>
<th>Refinements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bakery Protocol</td>
<td>3</td>
<td>0.21</td>
<td>1</td>
</tr>
<tr>
<td>Simpson Protocol</td>
<td>2</td>
<td>0.16</td>
<td>2</td>
</tr>
<tr>
<td>Train Gate Controller</td>
<td>5</td>
<td>0.52</td>
<td>0</td>
</tr>
<tr>
<td>Fischer Protocol</td>
<td>4</td>
<td>0.71</td>
<td>0</td>
</tr>
<tr>
<td>Water Level Monitor</td>
<td>1</td>
<td>0.08</td>
<td>0</td>
</tr>
<tr>
<td>Leaking Gas Burner</td>
<td>6</td>
<td>1.13</td>
<td>0</td>
</tr>
<tr>
<td>Multi Rate Fischer</td>
<td>4</td>
<td>0.84</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. Results for $k$-induction. Timings are in seconds.

The experimental results in Table 1 are obtained on a 2GHz Pentium-IV with 1Gb of memory. The second row in Table 1 lists the minimal $k$ for which $k$-induction succeeds, the third row includes the total time (in seconds) needed for all inductions from 0 to $k$, and the fourth row the number of strengthenings. Timings do not include the one for quantifier elimination, since we restricted ourselves to syntactic quantifier elimination only\cite{3}. Notice that invariant strengthening is essential for the proofs of the Bakery protocol and Simpson's protocol, since $k$-induction alone does not succeed.

\footnote{2} These benchmarks are available at http://www.csl.sri.com/~demoura/cav03examples

\footnote{3} Note to the reviewer: we plan to include further experiments by using strengthening on all benchmarks.
Simpson’s protocol for avoiding interference between concurrent reads and writes in a fully asynchronous system has also been studied using traditional model checking techniques. Using an explicit-state model checker, Rushby [14] demonstrates correctness of a finitary version of this potentially infinite-state problem. Whereas it took around 100 seconds for the model checker to verify this stripped-down problem, $k$-induction together with invariant strengthening proves the general problem in a fraction of a second. Moreover, other nontrivial problems such as correctness of Illinois and Futurebus cache coherence protocols, as given by [7], are easily established using $1$-induction with only one round of strengthening.

8 Related Work

We restrict this comparison to work we think is most closely related to ours. Sheeran, Singh, and Stalmarck's [15] also use $k$-induction, but their approach is restricted to finite-state systems only. They consider $k$-induction restricted to acyclic paths and each path is constrained to contain at most one initial state. These inductions are simple instances of our general induction scheme based on reverse and direct simulations. Moreover, invariant strengthening is used here to decrease the minimal $k$ for which $k$-induction succeeds.

Our path compression techniques can also be used to compute tight completeness thresholds for BMC. For example, a compressed recurrence diameter is defined as the smallest $n$ such that $I(s_0) \land \pi^{\leq r, s}(s_0, \ldots, s_n)$ is unsatisfiable. Using equality $=$ for the simulation relation, this formula is equivalent to the recurrence diameter in [3]. A tighter bound of the recurrence diameter, where values of input variables are ignored, is obtained by using the reverse simulation $\Rightarrow$. In this way, the results in [11] are obtained as specific instances in our general framework based on reverse and direct simulations. In addition, the compressed diameter is defined as the smallest $n$ such that

$$I(s_0) \land \pi^{\leq r, s}(s_0, \ldots, s_n) \land \bigwedge_{i=0}^{n-1} \neg \pi^{\leq r, s}(s_0, s_i)$$

is unsatisfiable, where $\pi^{\leq r, s}(s_0, s_i) := \exists s_1, \ldots, s_{i-1}. \pi^{\leq r, s}(s_0, s_1, \ldots, s_{i-1}, s_i)$ holds if there is a relevant path from $s_0$ to $s_i$ with $i$ steps. Depending on the simulation relation, this compressed diameter yields tighter bounds for the completeness thresholds than the ones usually used in BMC [3].

9 Conclusion

We developed a general $k$-induction scheme based on the notion of reverse and direct simulation, and we studied completeness of these inductions. Although any $k$-induction proof can be reduced to a $1$-induction proof with invariant strengthening, there are certain advantages of using $k$-induction. In particular, bugs of
length \( k \) are detected in the initial step, and the number of strengthenings required to complete a proof is reduced significantly. For example, a 1-induction proof of the Bakery protocol requires three successive strengthenings each of which produces 4 new clauses. There is, however, a clear trade-off between the additional cost of using \( k \)-induction and the number of strengthenings required in 1-induction, which needs to be studied further.

References