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, ruess, sorea}@csl.sri.com
http://www.csl.sri.com/

Abstract. We explore the combination of bounded model checking and induction for proving safety properties of infinite-state systems. In particular, 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 strengthening step requires quantifier-elimination, and we propose a *lazy* quantifier-elimination procedure, which delays expensive computations of disjunctive 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) [5,4,7] 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 completeness 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 establishing invariance properties φ [11,10,13,3]. This rule requires a property ψ which is stronger than φ and *inductive* in the sense that all initial states satisfy ψ, and ψ is preserved under each transition. Theoretically, the invariance rule is adequate for verifying a valid property of a system, but its application usually

[*] Funded by SRI International, by NSF Grant CCR-0082560, DARPA/AFRL-WPAFB Contract F33615-01-C-1908, and NASA Contract B09060051.

[**] 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 *k-induction* rule. This induction rule generalizes BMC in that it requires demonstrating the invariance of φ 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 φ holds in every state of every execution of length k, then every successor state also satisfies φ. 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 [17].

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 *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 *anytime* algorithm for approximating adequate simulation relations for k-induction.

Whenever k-induction fails to prove a property φ, there is a counterexample of length $k + 1$ such that the first k states satisfy φ and the last state does not satisfy φ. If the first state of this trace is reachable, then φ is refuted. Otherwise, the counterexample is labeled *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 *lazy* theorem proving, as developed by the authors for the ground case [7], 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 $\mathcal{D}_1$ through $\mathcal{D}_n$, together with a type assignment τ such that $\tau(x_i) = \mathcal{D}_i$. For a set of typed variables V, a *variable assignment* is a function ν 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 $\mathcal{C}$, such as linear arithmetic or a theory of bitvectors. We assume a computable function for deciding satisfiability of a conjunction of constraints in $\mathcal{C}$. A set of *Boolean constraints*, $\mathsf{Bool}(\mathcal{C})$, includes all constraints in $\mathcal{C}$ and is closed under conjunction $\wedge$, disjunction $\vee$, and negation $\neg$. Effective solvers for deciding the satisfiability problem in $\mathsf{Bool}(\mathcal{C})$ have been previously described [7,6].

A tuple $\langle V, I, T\rangle$ is a *$\mathcal{C}$-program* over V, where interpretations of the typed variables V describe the set of states, $I \in \mathsf{Bool}(\mathcal{C}(V))$ is a predicate that describes the initial states, and $T \in \mathsf{Bool}(\mathcal{C}(V \cup V'))$ 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 $M = \langle V, I, T\rangle$, a sequence of states $\pi(s_0, s_1, \ldots, s_n)$ forms a *path* through M if $\bigwedge_{0 \le i < n} T(s_i, s_{i+1})$. 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 $\varphi \in \mathcal{C}(V)$ is *invariant* in M iff $\varphi(s)$ holds for every reachable state s in M. A *counterexample* for a property φ is a path $\pi(s_0, \ldots, s_n)$ such that $I(s_0)$ and $\neg\varphi(s_n)$, and the length $len(\pi)$ 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 $x_i' = f_1(x_1, \ldots, x_n)$ for all variables x_i. Programs with external, non-deterministic inputs are defined by partitioning the set of variables into input variables, 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 [2], 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 $q' \sim_{TA} u'$ [2]. Encoding of timed automata in terms of logical programs with linear arithmetic constraints are described in [19]. 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 φ, 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 $M = \langle V, I, T\rangle$ be a program and φ 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.

$$F_d(R)(s_1, s_2) := \begin{cases} \textit{if } \neg\varphi(s_1) \textit{ then } \neg\varphi(s_2) \\ \textit{else } \forall s'_1 \,.\, T(s_1, s'_1) \Rightarrow \exists s'_2 \,.\, R(s'_1, s'_2) \,\wedge\, T(s_2, s'_2) \end{cases}$$

$$F_r(R)(s_1, s_2) := \begin{cases} \textit{if } I(s_1) \textit{ then } I(s_2) \\ \textit{else } \forall s'_1 \,.\, T(s'_1, s_1) \Rightarrow \exists s'_2 \,.\, R(s'_1, s'_2) \,\wedge\, T(s'_2, s_2) \end{cases}$$

A *direct simulation* over V with respect to φ is any binary relation $\preceq$ over V that satisfies $\preceq \subseteq F_d(\preceq)$. Similarly, a *reverse simulation* over V with respect to φ is any binary relation $\preceq$ over V that satisfies $\preceq \subseteq F_r(\preceq)$.

In contrast to reverse simulations, direct simulations depend on a state formula φ. 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 simulations are used as running examples throughout the paper.

Example 1. The empty relation $a \preceq_\emptyset b := \textit{false}$ is a direct and a reverse simulation.

Example 2. Equality $(=)$ between states is a direct and a reverse simulation.

Example 3. The relation $s_1 \preceq_I s_2 := I(s_1) \wedge I(s_2)$ 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 $\langle V, I, T\rangle$ with inputs such that $\textit{input}(x)$ holds iff x is an input variable. The relation

$$s_1 =_i s_2 := \textit{for all variables } x \in V \,.\, \textit{input}(x) \textit{ or } s_1(x) = s_2(x),$$

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 weaker than $=$, and therefore gives rise to shorter paths.

Example 5. We now consider timed automata programs and clock constraints. The region equivalence $\sim_{TA}$, 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 $\preceq_I$ and $=_i$ in Examples 3 and 4 may be combined to obtain a new reverse simulation.

Given a program $M = \langle V, I, T \rangle$ and a property φ, 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:

$$\begin{array}{ll} \preceq_{r,0} := \preceq_r & \preceq_{d,0} := \preceq_d \\ \preceq_{r,n} := \preceq_{r,n-1} \cup F_r(\preceq_{r,n-1}) & \preceq_{d,n} := \preceq_{d,n-1} \cup F_d(\preceq_{d,n-1}) \end{array}$$

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). Also, quantifier elimination algorithms such as the one in Section 6 may be used in this iteration.

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 that are similar with respect to a given *direct* or *reverse* simulation. For direct (reverse) simulations we define a compressed

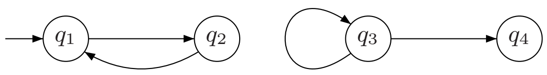

Fig. 1. Incompleteness of k-induction.

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 $\pi^{\preceq_d}(s_0, s_1, \ldots, s_n)$ is *compressed* w.r.t. the direct simulation $\preceq_d$ if:

$$\pi^{\preceq_d}(s_0, s_1, \ldots, s_n) := \pi(s_0, s_1, \ldots, s_n) \ \wedge \bigwedge_{0 \le j < i \le n} s_i \not\preceq_d s_j .$$

- A path $\pi^{\preceq_r}(s_0, s_1, \ldots, s_n)$ is *compressed* w.r.t. the reverse simulation $\preceq_r$ if:

$$\pi^{\preceq_r}(s_0, s_1, \ldots, s_n) := \pi(s_0, s_1, \ldots, s_n) \ \wedge \bigwedge_{0 \le i < j \le n} s_i \not\preceq_r s_j .$$

A path that is compressed with respect to the reverse and the direct simulations $\preceq_r$ and $\preceq_d$ is denoted by $\pi^{\preceq_{r,d}}$.

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 $\preceq_I$ from Example 3, a path $\pi(s_0, \ldots, s_n)$ is compressed w.r.t. $\preceq_I$ iff it contains at most one initial state. Obviously, for transition systems with inputs, the relation $(=_i)$ (see Example 4) is weaker 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 $M = \langle V, I, T \rangle$ be a program, k an integer, $\preceq_r$ a reverse simulation, and $\preceq_d$ a direct simulation. The induction scheme of depth k, $\text{IND}^{\preceq_{r,d}}(k)$ allows one to deduce the invariance of φ in M if the following holds.

- $I(s_0) \ \wedge \ \pi^{\preceq_{r,d}}(s_0, \ldots, s_{k-1}) \rightarrow \varphi(s_0) \ \wedge \ \ldots \ \wedge \ \varphi(s_{k-1})$
- $\varphi(s_n) \ \wedge \ \ldots \ \wedge \ \varphi(s_{n+k-1}) \ \wedge \ \pi^{\preceq_{r,d}}(s_n, \ldots, s_{n+k}) \rightarrow \varphi(s_{n+k})$

For example, given the empty relationship $\preceq_\emptyset$ from Example 1, $\text{IND}^{\preceq_\emptyset}$ reduces to the naive, incomplete k-induction on arbitrary paths. Consider, for example, the system in Figure 1 and a property φ, which is assumed to hold only in q_4. Now, the execution sequence $\underbrace{q_3 \rightsquigarrow q_3 \rightsquigarrow \ldots \rightsquigarrow q_3}_{k} \rightsquigarrow q_4$ is not k-inductive, but it is ruled out under the acyclic path restriction. The complete k-induction schemes in [17], 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 $(\preceq_I)$ (Example 3), respectively. Since both $(=)$ and $(\preceq_I)$ 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-$\pi^{\preceq_{r,d}}$ paths). Let $\pi(s_0, \ldots, s_n)$ be a given path; then:

1. There exists a $\pi^{\preceq_r}$- compressed path $\pi^{\preceq_r}(q_0, \ldots, q_m)$, s.t. $q_m = s_n$ and $m \leq n$.
2. There exists a $\pi^{\preceq_d}$- compressed path $\pi^{\preceq_d}(q_0, \ldots, q_m)$, s.t. $q_0 = s_0$ and $m \leq n$.

Proofsketch. Assume a path $\pi(s_0, \ldots, s_n)$, which is not compressed w.r.t. $\preceq_r$. By Definition 1 it follows that there are states $s_i, s_j \in \pi(s_0, \ldots, s_n)$ such that $s_i \preceq_r s_j$, 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-1} such that $T(s_{i-1}, s_i)$. Since $s_i \preceq_r s_j$ it follows by Definition 1 that there is a state s'_{i-1}, such that $s_{i-1} \preceq_r s'_{i-1}$ and $T(s'_{i-1}, s_j)$. If s_{i-1} is initial state, then so is s'_{i-1}, and since $i < j$ a shorter path $\pi^{\preceq_r}(s'_{i-1}, s_j, \ldots, s_n)$ is obtained. If s_{i-1} 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 $\pi^{\preceq_d}$- compressed paths works analogously.

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

Theorem 1 (Completeness). $\text{IND}^{\preceq_{r,d}}(k)$ is a complete proof method iff there is an upper bound on the length of the paths $\pi^{\preceq_{r,d}}(s_0, \ldots, s_n)$.

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 φ a state property in V; then $\text{IND}^{=}(k)$ induction is complete.

In general, k-induction for $(=)$ is not complete for infinite-state systems. Consider, for example, the program $M = \langle I, T\rangle$ 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 $\text{IND}^{=}(k)$. 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 φ a clock constraint in C; then $\text{IND}^{\sim_{TA}}(k)$ induction is complete.

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

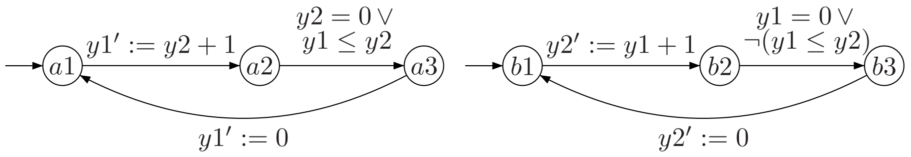

Fig. 2. Bakery Mutual Exclusion Protocol.

5 Invariant Strengthening

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

Consider, for example, the property $\neg(q_4)$ for the system in Figure 1. Induction of depth $k = 1$ fails, and the counterexample $q_3 \rightsquigarrow q_4$ is obtained. Now, $\neg(q_4)$ is strengthened to obtain $\neg(q_4) \wedge \neg(q_3)$, which is proven using 1-induction. More generally, whenever the induction step of $\mathrm{IND}^{\preceq_{r,d}}(k)$ fails, the formula $Q(s_n, \ldots, s_{n+k}) := \varphi(s_n) \wedge \ldots \wedge \varphi(s_{n+k-1}) \wedge \pi^{\preceq_{r,d}}(s_n, \ldots, s_{n+k}) \wedge \neg\varphi(s_{n+k})$ 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 $\pi^{\preceq_{r,d}}$ path, $U(s) = \exists\, s_{n+1}, \ldots, s_{n+k}.Q(s, s_{n+1}, \ldots, s_{n+k})$. Now, φ is strengthened as $\varphi \wedge \neg U(s)$, 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 $f(.)$. 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 $\neg(pc1 = a3 \wedge pc2 = b3)$. We apply 3-induction with the empty simulation relation $\preceq_\emptyset$. The base step holds and the induction step fails, thus we obtain

$$\begin{aligned} U(s_n) := \exists\, s_{n+1}, s_{n+2}, s_{n+3}.\ & MX(s_n) \wedge MX(s_{n+1}) \wedge \\ & MX(s_{n+2}) \wedge \pi^{\preceq_\emptyset}(s_n, s_{n+1}, s_{n+2}, s_{n+3}) \wedge \neg MX(s_{n+3}) \end{aligned}$$

with states s_i of the form $(pc1_i, y1_i, pc2_i, y2_i)$. Since the transitions of the Bakery protocol are in terms of guarded commands, simple substitution is used to obtain a quantifier-eliminated form, $R(s)$, defined as

$$R(s) := (pc1 = a1 \wedge pc2 = b2 \wedge y2 = 0) \vee (pc1 = a2 \wedge pc2 = b1 \wedge y1 = 0).$$

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

6 Quantifier Elimination

Given a quantified formula $\exists vars.\ \varphi$ with $\varphi \in \mathsf{Bool}(\mathcal{C})$, quantifier-elimination procedures usually work by transforming φ 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 $\mathcal{C}$-*qe*. The main drawback of these procedures is that there is a potential exponential blowup in the initial transformation to DNF and $\mathcal{C}$-*qe* might even return further disjunctions (as is the case for Presburger arithmetic); this problem has been addressed for the Boolean case by McMillan [14].

The quantifier elimination problem for invariant strengthening, as discussed in Section 5 allows for a purely syntactic quantifier elimination as long as we are restricting ourselves to guarded command programs. In these cases, $\mathcal{C}$-*qe* just applies the *substitution rule* ($x \notin vars(\psi)$)

$$(\exists x.(x = \psi) \wedge \varphi(x)) \text{ iff } \varphi(\psi);$$

possibly followed by simplification. Quantifier elimination by substitution has already been used in the context of model checking, for example, by Coudert, Berthet, and Madre [15] and more recently by Williams, Biere, Clarke, Gupta [20], and Abdulla, Bjesse, Eén [1]. Another $\mathcal{C}$-*qe* function is used in McMillan's [14] quantifier elimination algorithm based on propositional SAT solving, in that his $\mathcal{C}$-$qe(vars, c)$ simply deletes the literals in c, which contain a variable in $vars$. In contrast, depending on the background theory, arbitrary complex quantifier elimination procedures, such as the ones for Presburger arithmetic or real-closed fields, can also be used here.

As motivated above, the initial DNF computation should usually be avoided when possible. Given a set of existentially quantified variables $vars$ and a quantifier-free formula φ in $\mathsf{Bool}(\mathcal{C})$, the algorithm $qe(vars, \varphi)$ in Figure 3 returns a formula in $\mathsf{Bool}(\mathcal{C})$ which is equivalent to $\exists vars.\ \varphi$. The procedure qe relies on a satisfiability solver for formulas $\varphi \in \mathsf{Bool}(\mathcal{C})$, which is assumed to enumerate representations of sets of satisfiable models in terms of conjunctions of literals in φ. Such a solver is described, for example, in [7,6]. These solutions are supposed to be enumerated by successive calls to *next-solution* in Figure 3. Since there are only a finite number of solutions in terms of subsets of literals, the function qe is terminating. Moreover, minimal solutions or good over-approximations thereof, as produced by the lazy theorem proving algorithm [7,6], accelerate convergence.

The variable c in Figure 3 stores the current solution obtained by *next-solution*, and the procedure $\mathcal{C}$-*qe* applies quantifier elimination for conjunction. In

```
procedure qe($vars$, $\varphi$)
  $\psi := false$
  loop
    $c := next\text{-}solution(\varphi)$
    if $c = false$ then return $\psi$
    $c' := \mathcal{C}\text{-}qe(vars, c)$
    $\psi := \psi \vee c'$
    $\varphi := \varphi \wedge \neg c'$
```

Fig. 3. Lazy Quantifier Elimination.

many cases, $\mathcal{C}$-*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 φ 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 $\mathsf{Bool}(\mathcal{C})$ is used to guide the search for relevant "conjunctions" in φ. In this way, the *qe* algorithm extends the lazy theorem proving paradigm described in [7,6] to the case of first-order reasoning.

Example 7. Consider

$$\begin{array}{ll}\exists x_1, y_1 & ((x_0 = 1 \vee x_0 = 3 \vee y_0 > 1) \wedge x_1 = x_0 - 1 \wedge y_1 = y_0 + 1) \\ \vee & ((x_0 = -1 \vee x_0 = -3) \wedge x_1 = x_0 + 2 \wedge y_1 = y_0 - 1)) \wedge x_1 < 0\end{array}$$

A first satisfiable conjunction of literals is obtained by, say

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

Now, application of the substitution rule yields $c' := y_0 > 1 \wedge x_0 - 1 < 0$, and, after updating φ with $\neg c'$ a second solution is obtained as

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

Again, applying the substitution rule, one gets $c' := x_0 = -3 \wedge x_0 + 2 < 0$, and, since there are no further solutions, the quantifier-eliminated formula is $(y_0 > 1 \wedge x_0 - 1 < 0) \vee (x_0 = -3 \wedge x_0 + 2 < 0)$.

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.[1] In particular, Table 1 contains experimental results for the Bakery protocol as described earlier, Simpson's protocol [18] to avoid interference between concurrent reads and

[1] These benchmarks are available at http://www.csl.sri.com/~demoura/cav03examples

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 [19]. 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, $\mathcal{C}$-*qe* 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 [7] and implemented in the ICS decision procedures [9].

System Name	Proved with k	Time	Refinements
Bakery Protocol	3	0.21	1
Simpson Protocol	2	0.16	2
Train Gate Controller	5	0.52	0
Fischer Protocol	4	0.71	0
Water Level Monitor	1	0.08	0
Leaking Gas Burner	6	1.13	0
Multi Rate Fischer	4	0.84	0

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 column in Table 1 lists the minimal k for which k-induction succeeds, the third column includes the total time (in seconds) needed for all inductions from 0 to k, and the fourth column the number of strengthenings. Timings do not include the one for quantifier elimination, since we restricted ourselves to syntactic quantifier elimination only. Notice that invariant strengthening is essential for the proofs of the Bakery protocol and Simpson's protocol, since *k-induction* alone does not succeed for any k.

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 [16] 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 [8], 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 Stålmarck's [17] 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 $I(s_0) \wedge \pi^{\preceq_{r,d}}(s_0, \dots, s_n)$ is unsatisfiable. Using equality ($=$) for the simulation relation, this formula is equivalent to the *recurrence diameter* in [4]. A tighter bound of the recurrence diameter, where values of input variables are ignored, is obtained by using the reverse simulation $=_i$. In this way, the results in [12] 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

$$I(s_0) \wedge \pi^{\preceq_{r,d}}(s_0, \dots, s_n) \wedge \bigwedge_{i=0}^{n-1} \neg \pi_i^{\preceq_{r,d}}(s_0, s_i)$$

is unsatisfiable, where $\pi_i^{\preceq_{r,d}}(s_0, s_i) := \exists s_1, \dots, s_{i-1}. \pi^{\preceq_{r,d}}(s_0, s_1, \dots, s_{i-1}, s_i)$ 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 [4].

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

1. P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In S. Graf and M. Schwartzbach, editors, *TACAS 2000*, volume 1785 of *LNCS*, pages 411–425. Springer-Verlag, 2000.

2. R. Alur. Timed automata. In *Computer-Aided Verification, CAV 1999*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22, 1999.
3. S. Bensalem and Y. Lakhnech. Automatic generation of invariants. *Formal Methods in System Design*, 15:75–92, 1999.
4. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zh. Symbolic model checking without BDDs. *Lecture Notes in Computer Science*, 1579, 1999.
5. E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
6. L. de Moura and H. Rueß. Lemmas on demand for satisfiability solvers. *Annals of Mathematics and Artificial Intelligence*, 2002. Accepted for publication.
7. L. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *Conference on Automated Deduction (CADE)*, volume 2392 of *LNCS*, pages 438–455. Springer-Verlag, July 27-30 2002.
8. G. Delzanno. Automatic verification of parameterized cache coherence protocols. In *Computer Aided Verification (CAV'00)*, pages 53–68, 2000.
9. J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonization and Solving. In *Proceedings of CAV'2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer-Verlag, 2001.
10. S. M. German and B. Wegbreit. A synthesizer of inductive assertions. *IEEE Transactions on Software Engineering*, 1(1):68–75, Mar. 1975.
11. S. M. Katz and Z. Manna. A heuristic approach to program verification. In N. J. Nilsson, editor, *Proceedings of the 3rd IJCAI*, pages 500–512, Stanford, CA, Aug. 1973. William Kaufmann.
12. D. Kroening and O. Strichman. Efficient computation of recurrence diameters. In *Proceedings of VMCAI'03*, Jan. 2003.
13. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, Jan. 1995.
14. K. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer-Aided Verification, CAV 2002*, volume 2404 of *LNCS*. Springer-Verlag, 2002.
15. O. Coudert, C. Berthet, and J.C. Madre. Verification of synchronous sequential machines using symbolic execution. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 365–373, Grenoble, France, June 1989. Springer-Verlag.
16. J. Rushby. Model checking Simpson's four-slot fully asynchronous communication mechanism. Technical report, CSL, SRI International, Menlo Park, Menlo Park, CA, July 2002.
17. M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. *LNCS*, 1954:108, 2000.
18. H. R. Simpson. Four-slot fully asynchronous communication mechanism. *IEE Proceedings, Part E: Computers and Digital Techniques*, 137(1):17–30, Jan. 1990.
19. M. Sorea. Bounded model checking for timed automata. In *Proceedings of MTCS 2002*, volume 68 of *Electronic Notes in Theoretical Computer Science*, 2002.
20. P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Proc. Computer Aided Verification (CAV)*, volume 1855 of *LNCS*. Springer-Verlag, 2000.