2013 20th International Symposium on Temporal Representation and Reasoning

Incremental, Inductive Model Checking

Aaron R. Bradley
Mentor Graphics Corporation &
University of Colorado at Boulder
Email: bradleya@colorado.edu

***Abstract*—IC3, a model checking algorithm for invariance properties, has inspired a fair amount of research since it was first noticed in 2011 and is now widely used in the EDA industry. It is rooted in the deductive approach to verification, central to which is the application of mathematical induction. IC3 applies induction in two ways: in the typical manner, to detect convergence to an inductive strengthening of the property; and in an incremental manner, to discover relatively inductive lemmas in response to concrete error states. Core ideas in IC3 have been lifted to algorithms for model checking LTL and CTL properties and for analyzing infinite-state systems.**

I. Background

Model checking [9], [21], [22], in practice, is the task of deciding whether a design satisfies its logically-provided specification. Specifications have traditionally been written in LTL [21] and CTL [9]—or in practical variants of these temporal logics. Techniques for deciding satisfaction of a model checking query include algorithmic [9], [23] and deductive [16]. Symbolic state exploration is now standard [18].

Common to both logics—and, in practice, most widely used—is the class of invariance properties. An invariance property asserts that all reachable states satisfy some propositional or first-order assertion; alternately, no state that satisfies the negation of the assertion is reachable. Because of the prominence of the invariance-checking problem, there has been steady progress in the scalability of algorithms for it.

For this discussion, consider a finite-state transition system $S : (\bar{i}, \bar{x}, I, T)$ with primary inputs $\bar{i}$, state variables $\bar{x}$, initial condition $I(x)$, and transition relation $T(\bar{x}, \bar{i}, \bar{x}')$, where $\bar{x}'$ represent the values of the state variables in the next state and formulas are propositional. The oldest method of proving an invariance property $P(\bar{x})$ is induction [16], in which one shows that

1) the invariance property P includes the initial condition: $I(\bar{x}) \Rightarrow P(\bar{x})$;
2) and P-states have only P-state successors: $P(\bar{x}) \wedge T(\bar{x}, \bar{i}, \bar{x}') \Rightarrow P(\bar{x}')$.

Here, $F \Rightarrow G$ is shorthand for $\forall * . \ F \rightarrow G$. Failure of the inductive step does not imply that P does not hold, only that P is not "inductive." A basic goal of any induction-based invariance-checker is to find a strengthening formula F such that $P \wedge F$ is inductive.

Bounded model checking (BMC) introduced the use of propositional logic (SAT) solvers to the model checking problem [2]. For checking an invariance property, ever more time-shifted copies of the transition relation are appended to a SAT database—essentially symbolically executing the system starting from the initial states—a technique referred to as "unrolling" the relation. At each iteration, the error is temporarily asserted in the final time frame, directing the SAT solver to search for a counterexample of the given length.

While revolutionary for disproving specifications and finding shortest counterexample witnesses, BMC's practical incompleteness spurred the development of k-induction [19]. The technique of k-induction generalizes standard induction. In k-induction, one proves that all loop-free sequences of k P-states have only P-state successors. If a given k is inadequate, one tries a larger k.

Interpolation-based model checking (ITP) introduced the use of Craig interpolants as a means of achieving a property-directed over-approximating abstraction [17]. For a given set of states represented by formula F, if no F-state can reach a $\neg P$-state (an error) in k steps, then the k-step unrolling rooted in F and targeting $\neg P$ is unsatisfiable. A Craig interpolant then exists over variables $\bar{x}'$ that over-approximates states reachable from F in one time-step, which provides an abstract version of the strongest post-condition operator. Iterating this construction can result in a spurious intersection with error states, in which case k is increased. Convergence to an inductive strengthening of P occurs when the formed over-approximation implies the disjunction of previous over-approximating formulas.

II. An Incremental, Inductive Invariance Checker

Since BMC, k-induction, and ITP unroll the transition relation, they all suffer when k, the number of time frames, becomes large. IC3 (sometimes called PDR) departs from this BMC-inspired line of model checkers, instead refining over-approximating stepwise sets incrementally [3], [4]. Queries to the SAT solver involve only one step of the transition relation and are thus relatively easy. In practice, hundreds to thousands of SAT queries are solved per second, and memory consumption tends to be small and stable.

IC3 maintains a sequence of over-approximations F_i to sets of states reachable within i steps, for $0 \leq i \leq k$, where F_k is the "frontier" and k grows incrementally until convergence or a counterexample is found. It is always the case that $F_i(\bar{x}) \Rightarrow F_{i+1}(\bar{x})$ and $F_i(\bar{x}) \wedge T(\bar{x}, \bar{i}, \bar{x}') \Rightarrow F_{i+1}(\bar{x}')$. Each F_i is a conjunction of the property P with an initially empty set of clauses. For each $k > 0$, IC3 refines the F_i's as needed to prove inductiveness of P relative to F_k, i.e.,

1530-1311/13 $26.00 © 2013 IEEE
DOI 10.1109/TIME.2013.9

that $F_k(\overline{x}) \wedge T(\overline{x}, \overline{i}, \overline{x}') \Rightarrow P(\overline{x}')$. This refinement is property-driven: a counterexample to the inductiveness (CTI) of the property, which is an F_k-state with a $\neg P$-state as a successor, triggers IC3 to derive a clause to block it. If successful, it applies induction to generalize the clause to block many more states than the CTI alone. It then adds the generalized clause to F_i for all $i \leq k$.

Otherwise, it explores (transitive) predecessors of the CTI to derive supporting strengthening clauses until the original CTI can itself be addressed relative to F_k. This exploration of concrete predecessors is guided by a priority queue of pairs of states and frame indices: (s, i) represents the obligation that state s must be inductively excluded relative to F_i, i.e., proved unreachable for at least $i + 1$ steps. Obligations are handled in lowest-index-first order, guaranteeing termination. IC3 aggressively generalizes from states: once it addresses (s, i) by finding a clause $c \subseteq \neg s$ that is inductive relative to some F_j, $j \geq i$, IC3 adds obligation $(s, j+1)$ to the queue if $j < k$. This aggressive strategy not only facilitates early discovery of mutually inductive clauses, it also allows IC3 to find deep counterexamples even when k is small.

When no CTIs remain (for F_k), IC3 checks each clause of each F_i to determine if it can be propagated forward, i.e., if it has become inductive relative to F_i since its creation because of subsequent strengthening of F_i. In the process, IC3 determines whether any F_i has become an inductive strengthening of the property, in which case the property is declared to hold. If not, it "bootstraps" the new frontier F_{k+1} with all clauses that are inductive relative to F_k and increments k. This process continues until IC3 finds an inductive strengthening of the property or finds a counterexample by following a sequence of CTIs back to an initial state.

While complicated in detail, the overall ideas are simple: refine i-step over-approximating sets $F_1, \ldots, F_k$ until they are sufficiently strong to show that F_k's successors are P-states, which allows the frontier to progress one time frame. Generate clauses that are inductive relative to these sets and that block explicit states that can reach errors. Finally, use induction to aggressively generalize the blocking of one or a few states to the blocking of many related states.

III. Incremental, Inductive Model Checking

IC3 continues to inspire new research, some of which is outlined here. Most obvious are refinements of the core algorithm. Lifting CTIs before attempting to block them provides another opportunity for generalization [8], [10]. IC3 can be used within an incremental verification framework, in which the analysis of small changes to designs or families of properties can be accelerated based on previous executions, as the generated clauses of the frames F_i can be filtered and reused [8]. The generalization mechanism has been improved [12].

IC3 offers new opportunities for localization-reduction, including offering direction on how to refine [1] and allowing for lazy abstraction [24].

The ideas of IC3 have been lifted to LTL [6] and CTL [11] model checking. Both algorithms follow the "incremental, inductive" philosophy of refining by constructing, using induction, lemmas from (sets of) explicit states.

IC3-like algorithms have been developed for other decidable problem domains, including timed systems [13], [14], Petri nets (and a broader class of infinite-state transition systems) [15], and finite-state safety games [20]. Incomplete extensions to infinite-state systems are possible when combined with predicate generation [7], [13], [26].

The technique of state-based inductive generalization [5], as opposed to interpolation, is useful in its own right and has been applied in an interpolant-generating algorithm [25].

Finally, IC3 has the potential to impact application domains outside of verification that apply model checking or reachability algorithms, e.g., planning.

References

[1] J. Baumgartner, A. Ivrii, A. Matsliah, H. Mony. IC3-guided abstraction. In *FMCAD*, Nov. 2012.

[2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, 1999.

[3] A. R. Bradley. k-step relative inductive generalization. Technical report, CU Boulder, Mar. 2010. http://arxiv.org/abs/1003.3649.

[4] A. R. Bradley. SAT-based model checking without unrolling. In *VMCAI*, Jan. 2011.

[5] A. R. Bradley and Z. Manna. Checking safety by inductive generalization of counterexamples to induction. In *FMCAD*, Nov. 2007.

[6] A. R. Bradley, F. Somenzi, Z. Hassan, and Y. Zhang. An incremental approach to model checking progress properties. In *FMCAD*, Nov. 2011.

[7] A. Cimatti and A. Griggio. Software model checking via IC3. In *CAV*, July 2012.

[8] H. Chockler, A. Ivrii, A. Matsliah, S. Moran, Z. Nevo. Incremental formal verification of hardware. In *FMCAD*, Nov. 2011.

[9] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs*, May 1981.

[10] N. Een, A. Mishchenko, R. Brayton. Efficient implementation of property directed reachability. In *FMCAD*, Nov. 2011.

[11] Z. Hassan, A. R. Bradley, and F. Somenzi. Incremental, inductive CTL model checking. In *CAV*, July 2012.

[12] Z. Hassan, A. R. Bradley, and F. Somenzi. Better generalization in IC3. Submitted.

[13] K. Hoder and N. Bjørner. Generalized property directed reachability. In *SAT*, June 2012.

[14] R. Kindermann, T. A. Junttila, I. Niemelä. SMT-based induction methods for timed systems. In *FORMATS*, Sept. 2012.

[15] J. Kloos, R. Majumdar, F. Niksic, R. Piskac. Incremental, inductive coverability. In *CAV*, July 2013.

[16] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[17] K. L. McMillan, Interpolation and SAT-based model checking. In *CAV*, July 2003.

[18] K. L. McMillan. *Symbolic Model Checking*. Kluwer, Boston, MA, 1994.

[19] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *FMCAD*, Nov. 2000.

[20] A. Morgenstern, M. Gesell, K. Schneider. Solving games using incremental induction. In *IFM*, June 2013.

[21] A. Pnueli. The temporal logic of programs. In *FOCS*, Nov. 1977.

[22] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming*, April 1982.

[23] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, June 1986.

[24] Y. Vizel, O. Grumberg, S. Shoham. Lazy abstraction and SAT-based reachability in hardware model checking. In *FMCAD*, Nov. 2012.

[25] Y. Vizel, V. Ryvchin, A. Nadel. Efficient generation of small interpolants in CNF. In *CAV*, July 2013.

[26] T. Welp and A. Kuehlmann. QF BV model checking with property directed reachability. In *DATE*, March 2013.