IC3: Where Monolithic and Incremental Meet

Fabio Somenzi
Dept. of Electrical, Computer, and Energy Engineering
University of Colorado at Boulder
Email: fabio@colorado.edu

Aaron R. Bradley
Summit Charter Middle School
Email: arbrad@cs.stanford.edu

***Abstract*—IC3 is an approach to the verification of safety properties based on relative induction. It is incremental in the sense that instead of focusing on proving one assertion, it builds a sequence of small, relatively easy lemmas. These lemmas are in the form of clauses that are derived from counterexamples to induction and that are inductive relative to reachability assumptions. At the same time, IC3 progressively refines approximations of the states reachable in given numbers of steps. These approximations, also made up of clauses, are among the assumptions used to support the inductive reasoning, while their strengthening relies on the inductive clauses themselves. This interplay of the incremental and monolithic approaches lends IC3 efficiency and flexibility and produces high-quality property-driven abstractions. In contrast to other SAT-based approaches, IC3 performs very many, very inexpensive queries. This is another consequence of the incrementality of the algorithm and is a key to its ability to be implemented in highly parallel fashion.**

I. Introduction

This paper discusses the IC3 technique for model checking safety properties. It is meant as a companion to [13]. Section II illustrates the approach on examples, while the rest of this introduction and Section III put the algorithm in its historical and ideological context by showing its relation to other methods for finite-state verification.

A. Induction

Induction is fundamental to the verification of safety properties [1], [2]. The only question is how it should be applied.

Consider a finite state system, $S : (\overline{i},\ \overline{x},\ I(\overline{x}),\ T(\overline{i}, \overline{x}, \overline{x}'))$, consisting of primary inputs $\overline{i}$, state variables $\overline{x}$, a propositional formula $I(\overline{x})$ describing the initial configurations of the system, and a propositional formula $T(\overline{i}, \overline{x}, \overline{x}')$ describing the transition relation. Primed state variables $\overline{x}'$ represent the next state.

Suppose that one wants to prove that every reachable state satisfies state assertion $P(\overline{x})$. Beginner's luck might allow one to proceed as follows:

- Show that the initial configuration of the system satisfies P: $I(\overline{x}) \Rightarrow P(\overline{x})$, where $\Rightarrow$ corresponds to implication. That is, all states that satisfy the initial condition I also satisfy P.
- Show that a P-state can only be followed by another P-state: $P(\overline{x}) \wedge T(\overline{i}, \overline{x}, \overline{x}') \Rightarrow P(\overline{x}')$.

These two steps—sometimes called *initiation* and *consecution*, respectively—comprise induction over S.

B. Monolithic and Incremental Methods

Outside of a classroom, such a direct application of induction is bound to fail. The development of safety model checking has essentially been the study of what one should do when, as usual, it does fail. In *Temporal Verification of Reactive Systems: Safety* [3], Manna and Pnueli write,

> We present two solutions to this problem, which can be summarized by the following strategies:
> 1) Use a stronger assertion, or
> 2) Conduct an incremental proof, using previously established P-invariants.

They go on to endorse the latter approach when engaging in manual or computer-aided verification:

> We strongly recommend [an incremental proof] whenever applicable. Its main advantage is that of *modularity*.

The former approach, however, is the one that has been most pursued from an algorithmic point of view in the context of hardware model checking. The formal basis for this approach is the following. If

- $I(\overline{x}) \Rightarrow F(\overline{x})$
- $F(\overline{x}) \wedge T(\overline{i}, \overline{x}, \overline{x}') \Rightarrow F(\overline{x}')$
- $F(\overline{x}) \Rightarrow P(\overline{x})$

then P is an invariant of S. In words, if F is inductive over S and implies P, then both F and P are invariants.

Traditional model checkers, based on BDDs [4] or SAT [5], explicitly compute post-conditions to compute the strongest possible strengthening of P, namely the reachable set of states, or pre-conditions to compute the weakest possible strengthening of P, namely all states except those that can lead to a violation of P. Bounded model checkers (BMC) exploit the finiteness of the state graph to enable a complete approach based on unrolling T and searching, with a SAT solver, for a counterexample trace [6]. An alternative to relying on a property of the state graph is to strengthen consecution simply by considering multiple time steps at once: k-induction assumes that P holds over multiple time steps to increase the likelihood that P holds in the next time step [7]. BDD-based algorithms that compute backward reachability can also be interpreted as computing increasingly strong consecutions: the number of iterations required for the fixpoint computation to converge to the weakest possible invariant that implies P gives the number of time steps to be considered to turn P into an inductive assertion.

Finally, one can abstract the post-condition in order to ease the computation, as in abstract interpretation [8]. Even better, one can abstract it with respect to the property, as in interpolation-based model checking, in which interpolants are derived from failed BMC queries [9].

We refer to these methods as *monolithic* because they spend all of their resources in computing one inductive assertion. Furthermore, their success is fundamentally tied to the reasoning engines—either the BDD package or the SAT solver. The representation of states reachable from the initial ones or states that can reach the target ones often entails prohibitively large BDDs. BMC, k-induction, and interpolant-based model checking fail when the SAT solver is overwhelmed by the number of unrollings of T.

One must then wonder whether an *incremental* approach, which is so successful for humans, might not be a bad idea as the basis for an algorithm. An incremental approach would compute many inductive assertions that all together strengthen P. It would thus have the modularity that Manna and Pnueli highlight—each assertion need only refer to an aspect of S—as well as the potential of not taxing the reasoning engines quite so much. Moreover, the incremental approach would be property directed, like the interpolant-based method: each intermediate assertion would arise to eliminate some hypothesized error.

The formal basis for an incremental approach is the following. Consider a sequence $\varphi_1(\overline{x}), \ldots, \varphi_n(\overline{x})$ of assertions such that

- every assertion is satisfied by the initial states: for each j, $I(\overline{x}) \Rightarrow \varphi_j(\overline{x})$,
- each assertion obeys consecution under the assumption that its predecessors hold: for each j,
$$\bigwedge_{1 \le k \le j} \varphi_k(\overline{x}) \;\wedge\; T(\overline{i}, \overline{x}, \overline{x}') \;\Rightarrow\; \varphi_j(\overline{x}') \;,$$
- and all together they imply P:
$$\bigwedge_{1 \le j \le n} \varphi_j(\overline{x}) \;\Rightarrow\; P(\overline{x}) \;.$$

If P also satisfies initiation, then it is an invariant of S. In this version of consecution (the second condition), we say that φ_j is inductive *relative to* $\varphi_1, \ldots, \varphi_{j-1}$.

In the incremental approach, one might as well assume P. If

- P is satisfied by the initial states: $I(\overline{x}) \Rightarrow P(\overline{i})$,
- every assertion is satisfied by the initial states: for each j, $I(\overline{x}) \Rightarrow \varphi_j(\overline{x})$,
- each assertion obeys consecution under the assumption that its predecessors and P hold: for each j,
$$\bigwedge_{1 \le k \le j} \varphi_k(\overline{x}) \;\wedge P(\overline{x}) \wedge T(\overline{i}, \overline{x}, \overline{x}') \;\Rightarrow\; \varphi_j(\overline{x}') \;,$$
- and P is inductive relative to the assertions,
$$\bigwedge_{1 \le j \le n} \varphi_j(\overline{x}) \;\wedge P(\overline{x}) \wedge T(\overline{i}, \overline{x}, \overline{x}') \;\Rightarrow\; P(\overline{x}') \;,$$

then P is an invariant of S.

Bradley and Manna proposed the first incremental safety model checking algorithm [10], [11]. It discovers inductive subclauses of the negation of states that lead, not necessarily directly, to violations of P. Such clauses eliminate the states from which they are derived while generalizing to eliminate many other states as well. Each clause is an assertion φ_j that is indeed typically inductive only relative to prior assertions but not on its own. As expected, deriving the clauses is relatively easy: the employed SAT solver solves many, often hundreds or thousands, of queries per second, in stark contrast to BMC, k-induction, and the interpolant method. An unexpected benefit is that this instance of the incremental approach is effectively parallelizable—and easily so. This characteristic has carried through in subsequent work.

Besides modularity and reduced labor, the incremental approach has one more benefit: induction-based generalization is a powerful mechanism for property-directed abstraction. Induction tends to find semantic relationships among states rather than simply adjacency, or structural, relationships, as in traditional model checking. The clause that eliminates a state s may well eliminate states that are far, or even disconnected, from s in the state graph. When induction is applied throughout the analysis rather than being the goal of a monolithic propagation, it abstracts the system in a property-directed fashion.

Unfortunately, this algorithm suffers from a common pitfall of incremental methods. Manna and Pnueli write:

> There are cases in which the conjunction $\varphi_1 \wedge \varphi_2$ is inductive, but it is not the case that φ_1 is inductive and φ_2 is inductive relative to φ_1.

In the context of the algorithm, a state s can be encountered such that $\neg s$ does not contain a subclause that is inductive relative to known information. In such situations, the algorithm falls back on state enumeration until sufficient information is acquired to resume inductive clause construction. Yet when such a situation does not occur, the algorithm is extremely effective [11].

This weakness of the incremental method is not an issue for manual or computer-assisted verification, as the human can provide an insight. But in an algorithmic context, one typically limits the form of assertions in order to control computational costs [8]. Is an algorithmic incremental method thus doomed from the start?

C. IC3: A Monolithic-Incremental Hybrid

While an incremental method may be limited in the form of its assertions, Bradley eventually realized that the constructed clauses need not be truly inductive. The machinery of induction can be applied just as well when stronger information is assumed—information that is not necessarily valid for the entire state space. In particular, stepwise assumptions—assertions that hold for some number of timesteps rather than for all time—could be combined with relative inductive clause generation to yield a hybrid monolithic-incremental method in which *relatively inductive clauses are guaranteed to exist* if P is invariant. IC3 is the result of this insight [12], [13].

IC3 is incremental in that it finds inductive subclauses of the negations of states, just as the first approach does—except that these clauses are now inductive relative to certain assumptions. Its use of SAT solvers is thus similar: hundreds to thousands of queries are solved per second. Additionally, the clauses are the right compromise between effort and information content, so that they can be traded effectively among parallel processes.

IC3 is monolithic in that it computes over-approximations to the sets of states reachable in one step, two steps, etc., until it converges upon an inductive strengthening assertion. Each major iteration propagates the clauses that comprise the timestep approximations forward in time as much as possible. These over-approximations are the information relative to which new clauses are generated.

Hence, IC3 alternates between an incremental mode, in which it uses states that lead, not necessarily directly, to violations of P to discover new relatively inductive clauses, and a monolithic mode, in which it propagates clauses forward across time steps. Models on which the original method [10] devolves into enumerating states cause IC3 to go through more major iterations, yielding long sequences of stepwise over-approximations. Models on which the original method succeeds are just as easy, and often easier, for IC3, and result in short sequences of stepwise over-approximations before the final inductive strengthenings are formed. And many other models cause IC3 to adapt either a more monolithic or a more incremental strategy at various stages. The power of IC3 is that it can quickly deduce lemmas for certain aspects of a model while working harder—and, at times, more monolithically—for other lemmas that require more clauses.

II. Examples

This section presents IC3 by way of two examples. The objective is to show the nature of the algorithm. Certain optimizations omitted from this exposition are essential in practice for good performance.

A. A Passing Property

Figure 1 shows the state transition graph of a system S with no primary inputs and state variables $\overline{x} = \{x_1, x_2\}$ such that

$$\begin{aligned} I(\overline{x}) &= \neg x_1 \wedge \neg x_2 \\ T(\overline{x}, \overline{x}') &= (x_1 \vee \neg x_2 \vee x_2') \wedge (x_1 \vee x_2 \vee \neg x_1') \\ &\quad \wedge (\neg x_1 \vee x_1') \wedge (\neg x_1 \vee \neg x_2') \wedge (x_2 \vee \neg x_2') \\ P(\overline{x}) &= \neg x_1 \vee x_2 \ \ . \end{aligned}$$

Each state in the figure is annotated with its encoding. The incoming arrow designates q_0 as initial, while the shaded state (q_3) violates P. Inspection of Fig. 1 reveals that the only reachable state of S is q_0 and that $S \models P$. This example is not meant to highlight the efficiency of IC3. On the contrary, it provides the opportunity for a rather extensive tour of the algorithm in spite of its simplicity. (The reader is however cautioned that interesting aspects of IC3, like its ability to quickly compute long counterexamples, or to find large sets of mutually inductive clauses, are better understood via the algorithm's fundamental intentions. This section only provides a stepping stone in that direction.)

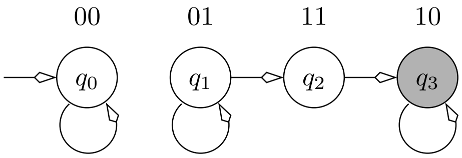

Fig. 1. The state transition graph of a simple system.

The initial check performed by IC3 establishes that there are no counterexamples of length 0 or 1. Therefore, the over-approximations (or stepwise assumptions)

$$\begin{aligned} F_0 &= I = \neg x_1 \wedge \neg x_2 \\ F_1 &= P = \neg x_1 \vee x_2 \end{aligned}$$

satisfy the fundamental IC3 invariants for $k = 1$:

$$\begin{aligned} I &\Rightarrow F_0 && \\ F_i &\Rightarrow F_{i+1} && 0 \le i < k \\ F_i &\Rightarrow P && 0 \le i \le k \\ F_i \wedge T &\Rightarrow F'_{i+1} && 0 \le i < k \ \ . \end{aligned}$$

Together, these invariants assert the "reasonableness" of the stepwise assumptions. In particular, since no counterexample of length up to k exists, all states reachable in at most k steps are P-states. Taking F_k to be P is therefore a valid over-approximation. If IC3 eventually increases k to 2, it is because it has established that there are no counterexamples of length up to 2. In general, if IC3 increases k from n to $n+1$, it is because it has established that there are no counterexamples of length up to $n+1$. It does so by proving that there are no counterexamples-to-induction (CTI) states that are reachable in at most n steps from some initial state. For that, it checks whether $F_n \wedge T \Rightarrow P'$ can be violated.

The check $F_1 \wedge T \Rightarrow P'$ produces $s = x_1 \wedge x_2$ as CTI. (Note that this check is equivalent to $P \wedge T \Rightarrow P'$, the inductive step of a simple inductive proof.) If $S \models P$, a CTI must be unreachable from the initial states. If $\neg s \wedge F_1 \wedge T \Rightarrow \neg s'$, unreachability is proved. If, however, the implication does not hold, the CTI may still be unreachable (as in this case) and IC3 tries to learn something useful about it: specifically, it tries to bound the length of a counterexample that goes through the CTI. Hence, $\neg s = \neg x_1 \vee \neg x_2$ is checked for inductiveness relative to the various F_i's. It is not inductive relative to F_1 because of the transition between q_1 and q_2. It is, however, inductive relative to F_0. (Otherwise, P would not hold.)

The inductiveness check has established that the CTI is not reachable in one step. Therefore, it would be possible to remove it from F_1 by adding the clause $\neg s$ to it. However, removing one CTI at a time is not practical for all but the simplest systems. Instead, IC3 looks for more states, be they CTIs or not, that, like the one at hand, are not reachable in one step and such that they are all described by a subclause of $\neg s$. That is, IC3 tries to generalize $\neg s$.

Generalization of $\neg s$ is thus attempted at level 0. The algorithm may find either $\neg x_1$ or $\neg x_2$ as subclauses of $\neg s$, because both satisfy both initiation and consecution. In fact, the conjunction of either clause with F_0 yields F_0 itself, from which no state violating either $\neg x_1$ or $\neg x_2$ may be reached. For the execution of the algorithm, however, which clause is the result of generalization makes a difference. Suppose $\neg x_2$ is found. Then the update of F_1 produces

$$F_1 = (\neg x_1 \vee x_2) \wedge \neg x_2 \ \ ,$$

which is equivalent to F_0. While this observation suffices to prove termination, IC3 first checks whether $F_1 \wedge T \Rightarrow P'$; that is, it checks whether the strengthening of F_1 has gotten rid of the CTI. Since the answer is positive, it increases k to 2, instantiates $F_2 = \neg x_1 \vee x_2$, and then propagates $\neg x_2$ from F_1. That is, it adds $\neg x_2$ to F_2 because $F_1 \wedge T \Rightarrow \neg x_2'$. The addition causes F_1 and F_2 to be identical and terminates the proof because $F_1 = (\neg x_1 \vee x_2) \wedge \neg x_2$ has been shown to be inductive ($I \Rightarrow F_1$ and $F_1 \wedge T \Rightarrow F_1'$) and is known to imply P. (F_1 is initially P and can only get stronger through the run of IC3.)

If, instead of $\neg x_2$, the generalization of $\neg x_1 \vee \neg x_2$ produces $\neg x_1$, the update of the reachability over-approximations results in

$$F_1 = (\neg x_1 \vee x_2) \wedge \neg x_1 \ \ ,$$

which is equivalent to $\neg x_1$. This F_1 is not as strong as in the previous case, and in particular does not exclude q_1, but it is still sufficient to satisfy $F_1 \wedge T \Rightarrow P'$. Therefore, IC3 sets k to 2, instantiates $F_2 = \neg x_1 \vee x_2$ and tries to strengthen it by propagating clause $\neg x_1$ from F_1. However,

$$F_1 \wedge T \not\Rightarrow \neg x_1' \ \ ,$$

because of the transition from q_1 to q_2; hence, no strengthening takes place. State $s = x_1 \wedge x_2$ is found once again as a CTI. The difference from the previous iteration is that it is now known that no counterexample of length less than 3 may go through it. IC3 then tries to prove that no counterexample of length 3 exists. The next step is therefore finding i such that

$$(\neg x_1 \vee \neg x_2) \wedge F_i \wedge T \Rightarrow (\neg x_1' \vee \neg x_2') \ \ .$$

Since $F_2 = P$ and F_0 has not changed, the answers for $i = 2$ and $i = 0$ are already known. It remains to ascertain whether F_1 is strong enough to support $\neg s$. Once again, the transition between q_1 and q_2 causes the answer to be negative. Therefore, $\neg s$ is inductive at level 0, but not at level 1. Generalization of this clause also proceeds as in the previous iteration and may result in either literal being dropped. If $\neg x_2$ is found, then its addition to F_1 makes it inductive, so that both $\neg x_1$ and $\neg x_2$ are propagated to F_2 causing termination.

If, on the other hand, $\neg x_1 \vee \neg x_2$ is generalized to $\neg x_1$, then no changes to F_1 result and no clause propagation ensues. Since F_2 has not changed, the CTI has not been removed. To guarantee termination, IC3 identifies a predecessor of $s = q_2$ that is an F_1 state, but not an F_0 state. The only choice is $t = \neg x_1 \wedge x_2$. If this state is proved unreachable, progress is made. More generally, if all predecessors of s in F_1 are shown to be unreachable in at most one step, then s is not reachable in at most two steps and hence there is no counterexample of length up to 3 through it.

IC3 therefore recurs on t to find which is the least i (if any) such that

$$\neg t \wedge F_i \wedge T \Rightarrow \neg t' \ \ .$$

Since $\neg t$ is itself inductive (q_1 in Fig. 1 has no incoming transitions from other states) $i = 2$. Since x_1 does not satisfy initiation, the only generalization of $\neg t$ is $\neg x_2$. The addition of this clause to both F_1 and F_2 makes them identical and causes termination.

In this case, F_1 is exact at termination. That is, F_1 describes exactly the states reachable in at most one step from the initial states. Oftentimes, though, the ability to prove properties quickly stems from the ability to keep the over-approximations loose. This is one reason why IC3 does not decompose the initial condition into a set of strong clauses that can be propagated.[1]

In contrast to IC3, the approach of [10] focuses on removing each CTI by generalizing its negation to an inductive clause. For the system of Fig. 1, this entails generalizing $s = \neg x_1 \vee \neg x_2$ by checking whether it contains a subclause d such that

$$d \wedge P \wedge T \Rightarrow d' \ \ .$$

The solution is in this case $d = \neg x_2$. Once this clause is discovered, it is possible to prove that $\neg x_2 \wedge P \wedge T \Rightarrow P'$, which in turn proves $S \models P$. However, if the encoding of the states is changed so that $q_2 = x_1 \wedge \neg x_2$ and $q_3 = x_1 \wedge x_2$, then the negation of the CTI $\neg s = \neg x_1 \vee x_2$ has no inductive generalization and the approach of [10] falls back on removing the CTI alone from further consideration. While this is hardly a disadvantage when there are only four states, it is the main weakness of that method. IC3 is also affected by the change of encoding, in that $\neg s = \neg x_1 \vee x_2$ can only be generalized to $\neg x_1$, but relatively inductive clauses can always be found.

B. A Failing Property

Figure 2 shows the transition graph of a system U with no primary inputs and state variables $\overline{x} = \{x_1, x_2, x_3\}$ defined by

$$\begin{aligned} I(\overline{x}) &= \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \\ T(\overline{x}, \overline{x}') &= (x_1 \vee \neg x_2') \wedge (\neg x_1 \vee x_2') \\ &\quad (x_2 \vee \neg x_3') \wedge (\neg x_2 \vee x_3') \\ P(\overline{x}) &= \neg x_1 \vee \neg x_2 \vee \neg x_3 \ \ . \end{aligned}$$

State q_i has code i. For example, q_3 is $\neg x_1 \wedge x_2 \wedge x_3$. As in Fig. 1, the shaded state violates property P.

[1] Implementations rely on pre-analysis of the model that easily discovers most state variables that can only take one value. Using any remaining literals from the initial condition typically lengthens the analysis because it overconstrains the early over-approximations.

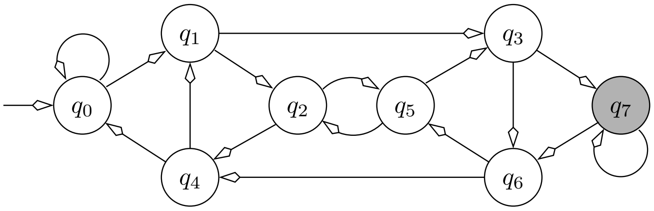

Fig. 2. Transition graph for a system with a failing property.

Having checked that there are no counterexamples of length up to 1, IC3 sets $k = 1$ and chooses

$$F_0 = I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$$
$$F_1 = P = \neg x_1 \vee \neg x_2 \vee \neg x_3$$

as stepwise assumptions. Checking whether $F_1 \wedge T \Rightarrow P'$ yields $s = \neg x_1 \wedge x_2 \wedge x_3$ as CTI. Inductiveness of $\neg s$ is established at level 0 and the generalization of $\neg s$ is $\neg x_2$. After the strengthening of F_1,

$$F_1 = P \wedge \neg x_2 \ ,$$

$F_1 \wedge T \Rightarrow P'$. Therefore, k increases to 2 and $F_2 = P$ is instantiated. No clause is propagated from F_1 to F_2. Therefore, the same CTI as before is found when $F_2 \wedge T \Rightarrow P'$ is tested. Since $\neg s \wedge F_1 \wedge T \not\Rightarrow \neg s'$, inductiveness is again established at level 0 and the generalization is again $\neg x_2$. Nothing changes in the stepwise assumptions, and the CTI remains an F_2 state. IC3 therefore looks for a predecessor of s that is in F_1. The choice is between $\neg x_1 \wedge \neg x_2 \wedge x_3$ and $x_1 \wedge \neg x_2 \wedge \neg x_3$. The former is immediately shown to be a successor of the initial state because its negation is not inductive even at level 0. Therefore the minimum-length counterexample q_0, q_1, q_3, q_7 is found.

If, instead of $\neg x_1 \wedge \neg x_2 \wedge x_3$, IC3 chooses $t = x_1 \wedge \neg x_2 \wedge \neg x_3$ as F_1 predecessor of the CTI, $\neg t$ is proved inductive at level 1 because the two predecessors of t are not in F_1. Generalization of $\neg t$ produces $\neg x_1$, which is added to both F_1 and F_2 eliminating $x_1 \wedge \neg x_2 \wedge \neg x_3$ from both. This forces the choice of $\neg x_1 \wedge \neg x_2 \wedge x_3$ as F_1 predecessor of the CTI and leads to the same counterexample as before. It should be noted how the refinement of the stepwise assumptions acted as guidance in the search for the counterexample.

IC3 does not guarantee counterexamples of minimum length. While k cannot increase beyond the length of a shortest counterexample, IC3 may find a counterexample well before k matches its length. This ability proves an important advantage when the transition relation is such that refining the stepwise assumptions beyond a certain point becomes difficult. This may be the case when the CTIs and the states that should be removed from one of the F_i's to get out of an impasse have codes that are different enough that the generalized inductive clauses do not cover the "problem" states.

When refinement of the stepwise assumptions proves difficult, IC3 often finds that the negation of the target state (CTI or one of its predecessors) is inductive at the level immediately preceding that of the target state. It then chooses a predecessor at the same level, producing a path with several states for one F_i until either the path eventually crosses into F_{i-1} or new clauses are generated that cause a refinement of the stepwise assumptions. Under these circumstances IC3 may still discover a deep counterexample even though k is small.

III. Discussion

A. What Problem is IC3 Trying to Solve?

Interpolation and k-induction address the practical incompleteness of BMC. The latter combines BMC with a consecution check:

$$P \wedge \bigwedge_{i=0}^{k-1} (T^{(i)} \wedge P^{(i)}) \Rightarrow P^{(k)} \ .$$

When that check fails, k is increased, corresponding to a further unrolling of T. In practice, k can be prohibitively large.

The interpolant method goes further: it suggests forming over-approximate stepwise reachability sets F_i using a fixed unrolling. It addresses the failure of the following implication by increasing k:

$$F_i \wedge \bigwedge_{i=0}^{k-1} (T^{(i)} \wedge P^{(i)}) \Rightarrow P^{(k)} \ .$$

Because the implication does not hold, no interpolant exists that lies between the i-step over-approximation F_i and the k-step unrolling leading to a violation of P. The interpolant method thus increases k for the next round, yielding better over-approximations F_i.

Hence, neither k-induction nor the interpolant method drop the regime of unrolling that BMC introduced. While they attempt to reduce the number of necessary unrollings, their completeness—both practically and theoretically—is still fundamentally tied to unrolling.

IC3 entirely sidesteps the need for unrolling and thus sets out on a new trail than that blazed by BMC. When confronted with a problem similar to the one in interpolation (though lacking any unrolling), that is, the failure of the implication

$$F_i \wedge T \Rightarrow P' \ ,$$

it refines the i-step over-approximation F_i itself—and typically earlier stepwise over-approximations—in order to make the refined implication come closer to holding. It accomplishes this refinement by incrementally generating stepwise-relative inductive clauses in reaction to the CTI that the implication's failure reveals. In the end, the sequence of over-approximate stepwise assertions F_i can be seen as a possible outcome of the interpolant method—though derived in a fundamentally different manner.

B. The Incremental Method: Beyond IC3

The purely incremental method fails when the space of assertions is too poor to provide lemmas for all possible situations. In the case of safety model checking, clauses are too weak to be the basis of a robust algorithm. IC3 provides a stronger framework in which to use a weak, but expressively complete, assertion domain. However, a pure incremental approach can work on its own in other settings.

In this conference, we present an incremental approach to model checking LTL properties of systems [14]. The fundamental insight is that *SCC-closed regions* of the state graph, which are a fundamental characterization used in BDD-based techniques [15], can be discovered through induction. Hence, inductive assertions, as discovered by IC3, are the intermediate lemmas of this approach. Unlike the relationship between error states and clauses in safety model checking, every hypothesized error—which we call a *skeleton*—that does not correspond to an actual error has a corresponding inductive proof. Thus, the algorithm is purely incremental, and it enjoys the usual benefits: modular reasoning, natural abstraction, and opportunities for parallelization.

Acknowledgments. This material is based on work supported in part by the National Science Foundation under grant No. 0952617 and by the Semiconductor Research Corporation under contract GRC 1859. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] R. W. Floyd, "Assigning meanings to programs," in *Symposia in Applied Mathematics*, vol. 19. American Mathematical Society, 1967, pp. 19–32.

[2] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, October 1969.

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

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Information and Computation*, vol. 98, no. 2, pp. 142–170, 1992.

[5] K. L. McMillan, "Applying SAT methods in unbounded symbolic model checking." in *CAV*, ser. LNCS, vol. 2404. Springer-Verlag, 2002, pp. 250–264.

[6] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Fifth International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, Amsterdam, The Netherlands, Mar. 1999, pp. 193–207, lNCS 1579.

[7] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a SAT-solver," in *Formal Methods in Computer Aided Design*, W. A. Hunt, Jr. and S. D. Johnson, Eds. Springer-Verlag, Nov. 2000, pp. 108–125, lNCS 1954.

[8] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *POPL*. ACM Press, 1977, pp. 238–252.

[9] K. L. McMillan, "Interpolation and SAT-based model checking," in *Fifteenth Conference on Computer Aided Verification (CAV'03)*, W. A. Hunt, Jr. and F. Somenzi, Eds. Berlin: Springer-Verlag, Jul. 2003, pp. 1–13, lNCS 2725.

[10] A. R. Bradley and Z. Manna, "Checking safety by inductive generalization of counterexamples to induction," in *Formal Methods in Computer Aided Design (FMCAD'07)*, Austin, TX, 2007, pp. 173–180.

[11] A. R. Bradley, "Safety analysis of systems," Ph.D. dissertation, Stanford University, May 2007.

[12] ——, "k-step relative inductive generalization," CU Boulder, Tech. Rep., March 2010, http://arxiv.org/abs/1003.3649.

[13] ——, "SAT-based model checking without unrolling," in *Verification, Model Checking, and Abstract Interpretation (VMCAI'11)*, Austin, TX, 2011, pp. 70–87, lNCS 6538.

[14] A. R. Bradley, F. Somenzi, Z. Hassan, and Y. Zhang, "An incremental approach to model checking progress properties," in *Formal Methods in Computer Aided Design (FMCAD'11)*, Austin, TX, 2011.

[15] R. Bloem, H. N. Gabow, and F. Somenzi, "An algorithm for strongly connected component analysis in $n \log n$ symbolic steps," in *Formal Methods in Computer Aided Design*, W. A. Hunt, Jr. and S. D. Johnson, Eds. Springer-Verlag, Nov. 2000, pp. 37–54, lNCS 1954.