Understanding IC3*

Aaron R. Bradley

ECEE Department, University of Colorado at Boulder
Email: bradleya@colorado.edu

Abstract. The recently introduced model checking algorithm IC3 has proved to be among the best SAT-based safety model checkers. Many implementations now exist. This paper provides the context from which IC3 was developed and explains how the originator of the algorithm understands it. Then it draws parallels between IC3 and the subsequently developed algorithms FAIR and IICTL, which extend IC3's ideas to the analysis of ω-regular and CTL properties, respectively. Finally, it draws attention to certain challenges that these algorithms pose for the SAT and SMT community.

1 Motivation

In *Temporal Verification of Reactive Systems: Safety*, Zohar Manna and Amir Pnueli discuss two strategies for strengthening an invariant property to be inductive [13]: "(1) Use a stronger assertion, or (2) Conduct an incremental proof, using previously established invariants." They "strongly recommend" the use of the second approach "whenever applicable," its advantage being "modularity." Yet they note that it is not always applicable, as a conjunction of assertions can be inductive when none of its components, on its own, is inductive. In this paper, the first method is referred to as "monolithic"—all effort is focused on producing one strengthening formula—while the second method is called "incremental."

1.1 Monolithic and Incremental Proof Methods

A simple pair of transition systems clarifies the two strategies and the limitations of the second:

```
x, y := 1, 1                    1
while *:                        2
  x, y := x + 1, y + x          3
```

```
x, y := 1, 1                    1
while *:                        2
  x, y := x + y, y + x          3
```

The star-notation indicates nondeterminism. Suppose that one wants to prove, for both systems, that $P : y \geq 1$ is invariant.

Consider the first system. To attempt to prove the invariant property P, one can apply induction:

* Work supported in part by the Semiconductor Research Corporation under contract GRC 2271.

– It holds initially because

$$\underbrace{x = 1 \wedge y = 1}_{\text{initial condition}} \Rightarrow \underbrace{y \geq 1}_{P} \ .$$

– But it does not hold at line 3 because

$$\underbrace{y \geq 1}_{P} \wedge \underbrace{x' = x + 1 \wedge y' = y + x}_{\text{transition relation}} \not\Rightarrow \underbrace{y' \geq 1}_{P'} \ .$$

The first step of an inductive proof of an invariant property is sometimes called *initiation*; the second, *consecution* [13]. In this case, consecution fails. Hence, an *inductive strengthening* of P must be found.

The first step in strengthening P is to identify why induction fails. Here, it's obvious enough: without knowing that x is nonnegative, one cannot know that y never decreases. The assertion $\varphi_1 : x \geq 0$ is inductive:

– it holds initially: $x = 1 \wedge y = 1 \Rightarrow x \geq 0$, and
– it continues to hold at line 3, where x is updated:

$$\underbrace{x \geq 0}_{\varphi_1} \wedge \underbrace{x' = x + 1 \wedge y' = y + x}_{\text{transition relation}} \Rightarrow \underbrace{x' \geq 0}_{\varphi_1'} \ .$$

Now $P : y \geq 1$ is inductive *relative to* φ_1 because consecution succeeds in the presence of φ_1:

$$\underbrace{x \geq 0}_{\varphi_1} \wedge \underbrace{y \geq 1}_{P} \wedge \underbrace{x' = x + 1 \wedge y' = y + x}_{\text{transition relation}} \Rightarrow \underbrace{y' \geq 1}_{P'} \ .$$

This use of "previously established invariants" makes for an "incremental proof": first establish φ_1; then establish P using φ_1. Here, each assertion is simple and discusses only one variable of the system. The inductive strengthening of $P : y \geq 1$ is thus $x \geq 0 \wedge y \geq 1$. Of course, the stronger assertion $x \geq 1 \wedge y \geq 1$ would work as well.

In the second transition system, neither $x \geq 0$ nor $y \geq 1$ is inductive on its own. For example, consecution fails for $x \geq 0$ because of the lack of knowledge about y:

$$x \geq 0 \wedge x' = x + y \wedge y' = y + x \not\Rightarrow x' \geq 0 \ .$$

Establishing $y \geq 1$ requires establishing the two assertions together:

– initiation: $x = 1 \wedge y = 1 \Rightarrow x \geq 0 \wedge y \geq 1$
– consecution: $x \geq 0 \wedge y \geq 1 \wedge x' = x + y \wedge y' = y + x \Rightarrow x' \geq 0 \wedge y' \geq 1$.

An incremental proof seems impossible in this case, as only the conjunction of the two assertions is inductive, not either on its own. Thus, for this system, one must invent the inductive strengthening of P all at once: $x \geq 0 \wedge y \geq 1$.

Notice that the assertion $x \geq 0 \wedge y \geq 1$ is inductive for the first transition system as well and so could have been proposed from the outset. However, especially in more realistic settings, an incremental proof is simpler than inventing a single inductive strengthening, when it is possible.

1.2 Initial Attempts at Incremental, Inductive Algorithms

IC3 is a result of asking the question: if the incremental method is often better for humans, might it be better for algorithms as well? The first attempt at addressing this question was in the context of linear inequality invariants. Previous work had established a constraint-based method of generating individual inductive linear inequalities [7]. Using duality in linear programming, the constraint-based method finds instantiations of the parameters $a_0, a_1, \ldots, a_n$ in the template

$$a_0x_0 + a_1x_1 + \cdots + a_{n-1}x_{n-1} + a_n \geq 0$$

that result in inductive assertions. A practical implementation uses previously established invariants when generating a new instance [17]. However, an enumerative algorithm generates the strongest possible over-approximation—for that domain—of the reachable state space, which may be far stronger than what is required to establish a given property.

A property-directed, rather than enumerative, approach is to guide the search for inductive instances with counterexamples to the inductiveness (CTIs) of the given property [5]. A CTI is a state (more generally, a set of states represented by a cube; that is, a conjunction of literals) that is a counterexample to consecution.

In the first system above, consecution fails for $P : y \geq 1$:

$$y \geq 1 \wedge x' = x + 1 \wedge y' = y + x \not\Rightarrow y' \geq 1 \ .$$

A CTI, returned by an SMT solver, is $x = -1 \wedge y = 1$. Until this state is eliminated, P cannot be established. The constraint system for generating an inductive instance of the template $ax+by+c \geq 0$ is augmented by the constraint $a(-1) + b(1) + c < 0$. In other words, the generated inductive assertion should establish that the CTI $x = -1 \wedge y = 1$ is unreachable. If no such assertion exists, other CTIs are examined instead. The resulting lemmas may be strong enough that revisiting this CTI will reveal an assertion that is inductive relative to them, finally eliminating the CTI. But in this example, the instance $x \geq 0$ ($a = 1$, $b = 0$, $c = 0$) is inductive and eliminates the CTI.

In the context of hardware model checking, this approach was developed into a complete model checker, called FSIS, for invariance properties [4]. Rather than linear inequality assertions, it generates clauses over latches. While the algorithm for generating strong inductive clauses is not trivial, understanding it is not essential for understanding the overall model checking algorithm, which is simple. The reader is thus referred to previous papers to learn about the clause-generation algorithm [4, 3]. Consider finite-state system $S : (\overline{i}, \overline{x}, I(\overline{x}), T(\overline{x}, \overline{i}, \overline{x}'))$ with primary inputs $\overline{i}$, state variables (latches) $\overline{x}$, a propositional formula $I(\overline{x})$ describing the initial configurations of the system, and a propositional formula $T(\overline{x}, \overline{i}, \overline{x}')$ describing the transition relation, and suppose that one desires to establish the invariance of assertion P. First, the algorithm checks if P is inductive with two SAT queries, for initiation and consecution, respectively:

$$I \Rightarrow P \qquad \text{and} \qquad P \wedge T \Rightarrow P' \ .$$

If they hold, P is invariant. If the first query fails, P is falsified by an initial state, and so it does not hold. If consecution fails—the likely scenario—then there is a state s that can lead in one step to an error; s is a CTI.

The inductive clause generation algorithm then attempts to find a clause c that is inductive and that is falsified by s. If one is found, c becomes an incremental lemma, φ_1, relative to which consecution is subsequently checked:

$$\varphi_1 \wedge P \wedge T \Rightarrow P' \ .$$

If consecution still fails, another CTI t is discovered, and again the clause generation algorithm is applied. This time, however, the generated clause need only be inductive relative to φ_1, in line with Manna's and Pnueli's description of incremental proofs. In this manner, a list of assertions, $\varphi_1, \varphi_2, \ldots, \varphi_k$, is produced, each inductive relative to its predecessors, until $P \wedge \bigwedge_i \varphi_i$ is inductive.

But what is to be done if no clause exists that both eliminates s and is inductive? In this case, the target is expanded: the error states grow from $\neg P$ to $\neg P \vee s$; said otherwise, the property to establish becomes $P \wedge \neg s$. Every CTI is handled in this way: either a relatively inductive clause is generated to eliminate it, or it is added to the target. The algorithm is complete for finite-state systems.

One important, though subtle, point in applying the incremental method is that the invariance property, P, that is to be established can be assumed when generating new inductive assertions. That is, a generated assertion need only be inductive relative to P itself. For suppose that auxiliary information ψ is inductive relative to P, and P is inductive relative to ψ:

$$\psi \wedge P \wedge T \Rightarrow \psi' \qquad \text{and} \qquad \psi \wedge P \wedge T \Rightarrow P' \ .$$

Then clearly $\psi \wedge P$ itself is inductive.

In the second transition system of Section 1.1, this extra information makes a difference. Consider consecution again for P:

$$y \geq 1 \wedge x' = x + y \wedge y' = y + x \Rightarrow y' \geq 1 \ .$$

It fails with, for example, the CTI $x = -1 \wedge y = 1$. While $x \geq 0$ eliminates this CTI, it is not inductive on its own. However, it is inductive relative to P:

$$\underbrace{y \geq 1}_{P} \ \wedge \ \underbrace{x \geq 0}_{\varphi_1} \wedge \underbrace{x' = x + y \wedge y' = y + x}_{\text{transition relation}} \Rightarrow \underbrace{x' \geq 0}_{\varphi_1'} \ .$$

By assuming P, an incremental proof is now possible. Once sufficient strengthening information is found, this seemingly circular reasoning straightens into an inductive strengthening.

However, this trick does not fundamentally strengthen the incremental proof methodology. There are still many situations in which the purely incremental approach is impossible.[1] Experiments with FSIS made it clear that this weakness had to be addressed.

[1] Consider, for example, a similar transition relation with three variables updated according to `x, y, z := x + y, y + z, z + x`. Neither $x \geq 0$ nor $z \geq 0$ is inductive relative to $P : y \geq 1$.

1.3 Other SAT-Based Approaches

This section considers the strengths and weaknesses, which motivate IC3, of other SAT-based approaches.

At one extreme are solvers based on backward search. Exact SAT-based symbolic model checking computes the set of states that can reach an error, relying on cube reduction to accelerate the analysis [14]. Conceptually, it uses the SAT solver to find a predecessor, reduces the resulting cube, and then blocks the states of that cube from being explored again. At convergence, the blocking clauses describe the weakest possible inductive strengthening of the invariant. Sequential SAT similarly reduces predecessor cubes, but it also reduces state cubes lacking unexplored predecessors via the implication graph of the associated (unsatisfiable) SAT query [12]. This latter approach computes a convenient inductive strengthening—not necessarily the weakest or the strongest. FSIS is like this latter method, except that, when possible, it uses induction to reduce a predecessor state cube, which can allow the exploration of backward paths to end earlier than in sequential SAT, besides producing stronger clauses.

The strength of pure backward search is that it does not tax the SAT solver. Memory is not an issue. Its weakness is that the search is blind with respect to the initial states. In the case of FSIS, its selection of new proof obligations is also too undirected; some predecessors trigger more informative lemmas than others, but FSIS has no way of knowing which. Perhaps because of this lack of direction, successful modern SAT-based model checkers, other than IC3, derive from BMC [1]. BMC is based on unrolling the transition relation between the initial and error states. Thus, the SAT solver considers both ends in its search.

While BMC is strong at finding counterexamples, it is practically incomplete. Interpolation (ITP) [15] and k-induction [18] address this practical incompleteness. The latter combines BMC (which becomes initiation) with a consecution check in which the transition relation is unrolled k times and the property is asserted at each non-final level. When that check fails, k is increased; in a finite-state context, there is a k establishing P if P is invariant. In practice, the sufficient k is sometimes small, but it can also be prohibitively large. Like exact model checking, k-induction cannot find a convenient strengthening; rather, its strengthening is based on a characteristic of the transition system.

ITP goes further. Rather than unrolling from the initial states (BMC) or applying induction directly (k-induction), it unrolls from the current frontier F_i, which contains at least all states at most i steps from an initial state. If the associated SAT query is unsatisfiable, the algorithm extracts an interpolant between F_i and the k-unrolling leading to a violation of P, which serves as the $(i+1)$-step over-approximation F_{i+1}. If the query is satisfiable, the algorithm increases k, yielding a finer over-approximating post-condition computation. The size of the unrolling that yields a proof can be smaller in practice than that of k-induction. Tuning the interpolant finder can allow it to find convenient assertions, potentially accelerating convergence to some inductive strengthening.

BMC-based approaches have the advantage of giving meaningful consideration to both initial and error states. However, they have the disadvantage of

being monolithic. They search for a single, often complex, strengthening, which can require many unrollings in practice, overwhelming the SAT solver.

IC3 addresses the weaknesses of both types of solvers while maintaining their strengths. Like the backward search-based methods, it relies on many simple SAT queries (Section 2.1) and so requires relatively little memory in practice. Like the BMC-based methods, it gives due consideration to the initial and error states (Section 2.2). It can be run successfully for extended periods, and—for the same reasons—it is parallelizable. Compared to FSIS, it uses the core idea of incrementally applying relative induction but applies it in a context in which every state cube is inductively generalizable. Hence, induction becomes an even more powerful method for reducing cubes in IC3.

2 IC3

Manna's and Pnueli's discussion of incremental proofs is in the context of manual proof construction, where the ingenuity of the human is the only limitation to the discovery of intermediate lemmas. In algorithms, lemma generation is typically restricted to some abstract domain [8] such as linear inequalities [9] or a fixed set of predicates [10]. Thus, the case in which a CTI cannot be eliminated through the construction of a relatively inductive assertion arises all too frequently, making FSIS, in retrospect, a rather naive algorithm.

The goal in moving beyond FSIS was to preserve its incremental character while addressing the weakness of backward search and the weakness of the incremental proof method: the common occurrence of mutually inductive sets of assertions that cannot be linearized into incremental proofs. In other words, what was sought was an algorithm that would smoothly transition between Manna's and Pnueli's incremental methodology, when possible, and monolithic inductive strengthening, when necessary.

This section discusses IC3 from two points of view: IC3 as a prover and IC3 as a bug finder. It should be read in conjunction with the formal treatment provided in the original paper [3]. Readers who wish to see IC3 applied to a small transition system are referred to [19].

2.1 Perspective One: IC3 as a Prover

IC3 maintains a sequence of stepwise over-approximating sets, $F_0 = I$, F_1, F_2, $\ldots$, F_k, F_{k+1}, where each set F_i over-approximates the set of states reachable in at most i steps from an initial state. Every set except F_{k+1} is a subset of P: $F_i \Rightarrow P$. Once F_k is refined so that it excludes all states that can reach a $\neg P$-state in one transition, F_{k+1}, too, is strengthened to be a subset of P by conjoining P to it. F_k is considered the "frontier" of the analysis. A final characteristic of these sets is that $F_i \wedge T \Rightarrow F'_{i+1}$. That is, all successors of F_i-states are F_{i+1}-states.

This description so far should be relatively familiar. Forward BDD-based reachability [16], for example, computes exact i-step reachability sets, and if any

such set ever includes a $\neg P$-state, the conclusion is that the property does not hold. ITP also computes i-step reachability sets, and like IC3's, they are over-approximating. However, when ITP encounters an over-approximating set that contains a $\neg P$-state, it refines its approximate post-image operator by further unrolling the transition relation, rather than addressing the weaknesses of the current stepwise sets directly. The crucial difference in the use of these sets between IC3 and ITP is that IC3 refines all of the sets throughout its execution.[2]

Putting these properties together reveals two characteristics of the reach sets. First, any state reachable in i steps is an F_i-state. Second, any F_i-state cannot reach a $\neg P$-state for at least $k - i + 1$ steps. For example, an F_{k+1}-state can actually be a $\neg P$-state, and an F_k-state may reach an error in one step. But an F_{k-1}-state definitely cannot transition to a $\neg P$-state (since $F_{k-1} \wedge T \Rightarrow F'_k$ and $F_k \Rightarrow P$).

Now, the property to check is whether P is inductive relative to F_k. Since $F_k \Rightarrow P$, the following query, corresponding to consecution for P relative to F_k, is executed:

$$F_k \wedge T \Rightarrow P' \ . \tag{1}$$

Suppose that the query succeeds and that F_k is itself inductive: $F_k \wedge T \Rightarrow F'_k$. Then F_k is an inductive strengthening of P that proves P's invariance.

Now suppose that the query succeeds but that F_k is not inductive. F_{k+1} can be strengthened to $F_{k+1} \wedge P$, since all successors of F_k-states are P-states. Additionally, a new frame F_{k+2} is introduced. IC3 brings in monolithic inductive strengthening by executing a phase of what can be seen as a simple predicate abstraction (propagateClauses [3]). Every clause that occurs in any F_i is treated as a predicate. A clause's occurrence in F_i means that it holds for at least i steps. This phase allows clauses to propagate forward from their current positions. Crucially, subsets of clauses can propagate forward together, allowing the discovery of mutually inductive clauses. For i ranging from 1 to k, IC3 computes the largest subset $C \subseteq F_i$ of clauses such that the following holds (consecution for C relative to F_i):

$$F_i \wedge T \Rightarrow C' \ .$$

These clauses C are then conjoined to F_{i+1}. Upon completion, F_{k+1} becomes the new frontier. Many of the stepwise sets may be improved as lemmas are propagated forward in time. If $F_k = F_{k+1}$, then F_k is inductive, which explains how F_k is determined to be inductive in the case above.

Finally, suppose that query (1) fails, revealing an F_k-state s (more generally, a cube of F_k-states) that can reach a $\neg P$-state in one transition; s is a CTI. In other words, the problem is not just that F_k is not inductive; the problem is that it is not even strong enough to rule out a $\neg P$-successor, and so more

[2] Of course, one might implement ITP to reuse previous over-approximating sets, so that it too could be seen to refine them throughout execution. Similarly, one might use transition unrolling in IC3. *But for completeness*, ITP relies on unrolling but not continual refinement of all stepwise sets, whereas IC3 relies on continual refinement of all stepwise sets but not unrolling.

reachability information must be discovered. IC3 follows the incremental proof methodology in this situation: it uses induction to find a lemma showing that s cannot be reached in k steps from an initial state. This lemma may take the form of a single clause or many clauses, the latter arising from analyzing transitive predecessors of s.

Ideally, the discovered lemma will prove that s cannot *ever* be reached. Less ideally, the lemma will be good enough to get propagated to future time frames once P becomes inductive relative to F_k. But at worst, the lemma will at least exclude s from frame F_k, and even F_{k+1}[3].

Specifically, IC3 first seeks a clause c whose literals are a subset of those of $\neg s$ and that is inductive relative to F_k; that is, it satisfies initiation and consecution:

$$I \Rightarrow c \qquad \text{and} \qquad F_k \wedge c \wedge T \Rightarrow c' \ .$$

Such a clause proves that s cannot be reached in $k+1$ steps. However, there may not be any such clause. It may be the case that a predecessor t exists that is an F_k-state and that eliminates the possibility of a relatively inductive clause. In fact, t could even be an F_{k-1}-state.[4]

Here is where IC3 is vastly superior to FSIS, and where it sidesteps the fundamental weakness of the incremental proof method. A failure to eliminate s at F_k is not a problem. Suppose that a clause c is found relative to F_{k-1} rather than F_k:

$$I \Rightarrow c \qquad \text{and} \qquad F_{k-1} \wedge c \wedge T \Rightarrow c' \ .$$

Because c is inductive relative to F_{k-1}, it is added to F_k: no successor of an $(F_{k-1} \wedge c)$-state is a $\neg c$-state. If even with this update to F_k, s still cannot be eliminated through inductive generalization (the process of generating a relatively inductive subclause of $\neg s$), then the failing query

$$F_k \wedge \neg s \wedge T \Rightarrow \neg s' \tag{2}$$

reveals a predecessor t that was *irrelevant for F_{k-1} but is a reason why inductive generalization fails relative to F_k*. This identification of a reason for failure of inductive generalization is one of IC3's insights. The predecessor is identified *after* the generation of c relative to F_{k-1} so that c focuses IC3 on predecessors of s that matter for F_k. The predecessor t is not just any predecessor of s: it is specifically one that prevents s's inductive generalization at F_k. IC3 thus has a meaningful criterion for choosing new proof obligations.

Now IC3 focuses on t until eventually a clause is produced that is inductive relative to frame F_{k-1} and that eliminates t as a predecessor of s through frame F_k. Focus can then return to s, although t is not forgotten. Inductive generalization of s relative to F_k may succeed this time; and if it does not, the newly discovered predecessor would again be a reason for its failure.

[3] The clause eventually generated for s relative to F_k strengthens F_{k+1} since it is inductive relative to F_k.

[4] However, it cannot be an F_{k-2}-state, for then s would be an F_{k-1}-state, and its successor $\neg P$-state an F_k-state. But it is known that $F_k \Rightarrow P$.

It is important that, during the recursion, all transitive predecessors of s be analyzed all the way through frame F_k. This analysis identifies mutually inductive (relative to F_k) sets of clauses. Only one of the clauses may actually eliminate s, but the clauses will have to be propagated forward together since they support each other. It may be the case, though, that some clauses are too specific, so that the mutual support breaks down during the clause propagation phase. This behavior is expected. As IC3 advances the frontier, it forces itself to consider ever more general situations, until it finally discovers the real reasons why s is unreachable. It is this balance between using stepwise-specific information and using induction to be as general as possible that allows IC3 to synthesize the monolithic and incremental proof strategies into one strategy.

2.2 Perspective Two: IC3 as a Bug Finder

Although IC3 is often inferior to BMC for finding bugs quickly, industry experience has shown that IC3 can find deep bugs that other formal techniques cannot. This section presents IC3 as a guided backward search. While heuristics for certain decision points may improve IC3's performance, the basic structure of the algorithm is already optimized for finding bugs. In particular, IC3 considers both initial and error states during its search. The following discussion develops a hypothetical, but typical, scenario for IC3, one which reveals the motivation behind IC3's order of handling proof obligations. Recall from the previous section that IC3 is also intelligent about choosing new proof obligations.

Suppose that query (1) revealed state s, which was inductively generalized relative to F_{k-2}; that query (2) revealed t as an F_{k-1}-state predecessor of s; and that t has been inductively generalized relative to F_{k-3}. At this point, IC3 has the proof obligations $\{(s,\ k-1),\ (t,\ k-2)\}$, indicating that s and t must next be inductively generalized relative to F_{k-1} and F_{k-2}, respectively. As indicated in the last section, neither state will be forgotten until it is generalized relative to F_k, even if s happens to be generalized relative to F_k first.

At this point, with the proof obligations $\{(s,\ k-1),\ (t,\ k-2)\}$, it is fairly obvious that until t is addressed, IC3 cannot return its focus to s; t would still cause problems for generalizing s relative to F_{k-1}. While focusing on t, suppose that u is discovered as a predecessor of t during an attempt to generalize t relative to F_{k-2}. Although t cannot be generalized relative to F_{k-2}, u may well be; it is, after all, a different state with different literals. Indeed, it may even be generalizable relative to F_k. In any case, suppose that it is generalizable relative to F_{k-2} but not higher, resulting in one more proof obligation: $(u,\ k-1)$. Overall, the obligations are now $\{(s,\ k-1),\ (t,\ k-2),\ (u,\ k-1)\}$.

The question IC3 faces is which proof obligation to consider next. It turns out that correctness requires considering the obligation $(t,\ k-2)$ first [3]. Suppose that t and u are mutual predecessors.[5] Were $(u,\ k-1)$ treated first, t could be discovered as an F_{k-1}-state predecessor of u, resulting in a duplicate proof

[5] The current scenario allows it. For t's generalization relative to F_{k-3} produced a clause at F_{k-2} that excludes t, which means that the generalization of u relative to

obligation and jeopardizing termination. But one might cast correctness aside and argue that u should be examined first anyway—perhaps it is "deeper" than s or t given that it is a predecessor of t.

Actually, the evidence is to the contrary. The obligations $(u,\ k-1)$ and $(t,\ k-2)$ show that u is at least k steps away from an initial state (recall that u has been eliminated from F_{k-1}), whereas t is at least only $k-1$ steps away. That is, IC3's information predicts that t is "closer" to an initial state than is u and so is a better state to follow to find a counterexample trace.

Thus, there are two characteristics of a proof obligation $(a,\ i)$ to consider: (1) the length of the suffix of a, which leads to a property violation, and (2) the estimated proximity, $i+1$, to an initial state. In the example above, s, t, and u have suffixes of length 0, 1, and 2, respectively; and their estimated proximities to initial states are k, $k-1$, and k, respectively. Both correctness and intuition suggest that pursuing a state with the lowest proximity is the best bet. In the case that multiple states have the lowest proximity, one can heuristically choose from those states the state with the greatest suffix length (for "depth") or the shortest suffix length (for "short" counterexamples)—or apply some other heuristic.

From this perspective, IC3 employs inductive generalization as a method of dynamically updating the proximity estimates of states that lead to a violation of the property. Inductive generalization provides for not only the update of the proximities of explicitly observed CTI states but also of many other states. When $F_k \wedge T \Rightarrow P'$ holds, all proximity estimates of $\neg P$-predecessors are $k+1$, and so another frame must be added to continue the guided search.

The bug-finding and proof-finding perspectives agree on a crucial point: even if the initial CTI s has been inductively generalized relative to F_k, its transitive predecessors should still be analyzed through F_k in order to update their and related states' proximity estimates. A consequence of this persistence is that IC3 can search deeply even when k is small.

3 Beyond IC3: Incremental, Inductive Verification

Since IC3, the term *incremental, inductive verification* (IIV) has been coined to describe algorithms that use induction to construct lemmas in response to property-driven hypotheses (e.g., the CTIs of FSIS and IC3). Two significant new incremental, inductive model checking algorithms have been introduced. One, called FAIR, addresses ω-regular (e.g., LTL) properties [6]. Another, called IICTL, addresses CTL properties [11]. While this section does not describe each in depth, it attempts to draw meaningful parallels among IC3, FAIR, and IICTL. Most superficially, FAIR uses IC3 to answer reachability queries, and IICTL uses IC3 and FAIR to address reachability and fair cycle queries, respectively.

An IIV algorithm can be characterized by (1) the form of its hypotheses, (2) the form of its lemmas, (3) how it uses induction, and (4) the basis of generalization. For IC3, these characterizations are as follows:

F_{k-2} ignores t. Therefore, it is certainly possible for u to be generalized at F_{k-2}, leaving obligation $(u,\ k-1)$.

1. *Hypotheses*: Counterexamples to induction (CTIs). When consecution fails, the SAT solver returns a state explaining its failure, which IC3 then inductively generalizes, possibly after generating and addressing further proof obligations.
2. *Lemmas*: Clauses over state variables. A clause is generated in response to a CTI, using only the negation of the literals of the CTI.
3. *Induction*: Lemmas are inductive relative to stepwise information.
4. *Generalization*: Induction guides the production of minimal clauses—clauses that do not have any relatively inductive subclauses. The smaller the clause, the greater is the generalization; hence, induction is fundamental to generalization in IC3.

FAIR searches for reachable fair cycles, or "lasso" counterexamples. The fundamental insight of FAIR is that SCC-closed sets can be described by sequences of inductive assertions. In other words, an inductive assertion is a barrier across the state space which the system can cross in only one direction. A transition from one side to the other is a form of progress, since the system can never return to the other side. FAIR is characterized as an IIV algorithm as follows:

1. *Hypotheses*: Skeletons. A skeleton is a set of states that together satisfy all Büchi fairness conditions and that all appear on one side of every previously generated barrier. The goal is to connect the states into a "lasso" through reachability queries.
2. *Lemmas*: An inductive assertion. Each lemma provides one of two types of information: (1) global reachability information, which is generated when IC3 shows that a state of a skeleton cannot be reached; (2) SCC information, which is generated when IC3 shows that one state of the skeleton cannot reach another. In the latter case, all subsequent skeletons must be chosen from one "side" of the assertion.
3. *Induction*: SCC-closed sets are discovered via inductive assertions.
4. *Generalization*: Proofs constructed by IC3 can be refined to provide stronger global reachability information or smaller descriptions of one-way barriers. Furthermore, new barriers are generated relative to previous ones and transect the entire state space, not just the "arena" from which the skeleton was selected. Exact SCC computation is not required.

IICTL considers CTL properties hierarchically, as in BDD-based model checking [16], but rather than computing exact sets for each node, it incrementally refines under- and over-approximations of these sets. When a state is undecided for a node—that is, it is in the over-approximation but not in the under-approximation—its membership is decided via a set of SAT (for EX nodes), reachability (for EU nodes), or fair cycle (for EG nodes) queries. IICTL is characterized as an IIV algorithm as follows:

1. *Hypotheses*: A state is undecided for a node if it is included in the upper-bound but excluded from the lower-bound. If it comes up during the analysis, its status for the node must be decided.

2. *Lemmas*: Lemmas refine the over- and under-approximations of nodes, either introducing new states into under-approximations or removing states from over-approximations.
3. *Induction*: Induction is used to answer the queries for EU and EG nodes.
4. *Generalization*: Generalization takes two forms. For negatively answered queries, the returned proofs are refined to add (remove) as many states as possible to the under-approximations (from the over-approximations) of nodes, rather than just the motivating hypothesis state. For positively answered queries, the returned traces are generalized through a "forall-exists" generalization procedure, again to decide as many states as possible in addition to the hypothesis state.

All three algorithms are "lazy" in that they only respond to concrete hypotheses but are "eager" in that they generalize from specific instances to strong lemmas. Furthermore, all hypotheses are derived from the given property, so that the algorithms' searches are property-directed.

4 Challenges for SAT and SMT Solvers

The queries that IIV methods pose to SAT solvers differ significantly in character from those posed by BMC, k-induction, or ITP. There is thus an opportunity for SAT and SMT research to directly improve the performance of IIV algorithms.

IC3 is the first widespread verification method that requires highly efficient incremental solvers. An incremental interface for IC3 must allow single clauses to be pushed and popped; it must also allow literal assumptions. IIV algorithms pose many thousands to millions of queries in the course of an analysis, and so speed is crucial. FAIR requires even greater incrementality: the solver must allow sets of clauses to be pushed and popped.

IIV methods use variable orders to direct the generation of inductive clauses. An ideal solver would use these variable orders to direct the identification of the core assumptions or to direct the lifting of an assignment.

The inductive barriers produced in FAIR provide opportunities for generalization (in the *cycle queries* [6]) but are not required for completeness. Using all such barriers overwhelms the solver, yet using too few reduces opportunities for generalization. Therefore, currently, a heuristic external to the solver decides whether to use a new barrier or not. Ideally, a solver would provide feedback on whether a group of clauses has been used or not for subsequent queries. Those clause groups that remain unused for several iterations of FAIR would be removed. This functionality would allow direct identification of useful barriers.

IIV algorithms gradually learn information about a system in the form of lemmas. Thus, a core set of constraints, which includes the transition relation, grows and is used by every worker thread. On a multi-core machine, replicating this set of constraints in each thread's solver instance uses memory—and memory bandwidth—poorly, and this situation will grow worse as the number of available cores grows. An ideal solver for IIV algorithms would provide access to every thread to a growing core set of constraints. Each thread would then

have a thread-specific view in which to push and pop additional information for incremental queries.

Finally, IC3 has shown itself to be highly sensitive to the various behaviors of SAT solvers. Swapping one solver for another, or even making a seemingly innocuous adjustment to the solver, can cause widely varying performance—even if the average time per SAT call remains about the same. For example, a SAT solver that is too deterministic can cause IC3 to dwell on one part of the state space by returning a sequence of similar CTIs, so that IC3 must generate more lemmas. Identifying the desirable characteristics of a solver intended for IC3 will be of great help.

5 What's Next?

Fundamentally, IC3 should not be seen as a clause-generating algorithm. Rather, the insight of IC3 is in how it can harness seemingly weak abstract domains to produce complex inductive strengthenings. At first glance, it seems that IC3's abstract domain is CNF over state variables. In fact, the abstract domain is *conjunctions of state variables* (over the inverse transition relation; see next paragraph), which is, practically speaking, the simplest possible domain.

If that perspective seems unclear, think about how IC3 works: to address CTI s, it performs what is essentially a simple predicate abstraction over the inverse of the transition relation, where the predicates are the literals of s. This process produces a cube $d \subseteq s$—that is, a conjunction of a subset of the predicates—that lacks $\neg d$-predecessors (within a stepwise context F_i); therefore, the clause $\neg d$ is inductive (relative to F_i). It is the incremental nature of IC3 that produces, over time, a conjunction of clauses.

Recalling the linear inequality domain [5] seems to confuse the issue, however. Where are the disjunctions in that context? To understand it, consider the polyhedral domain [9]. If it were to be used in the same way as the state variable domain of IC3, then each CTI would be analyzed with a full polyhedral analysis, each lemma would take the form of a disjunction of linear inequalities, and IC3 would produce proofs in the form of a CNF formula of linear inequalities. That approach would usually be unnecessarily expensive. Clearly, the polyhedral domain is not being used. Instead, the domain is much simpler: it is a domain of half-spaces.[6]

Therefore, the next step, in order to achieve word-level hardware or software model checking, is to introduce new abstract domains appropriate for IC3—domains so simple that they could not possibly work outside the context of IC3, yet sufficiently expressive that IC3 can weave together their simple lemmas into complex inductive strengthenings.

[6] A Boolean clause can be seen as a half-space over a Boolean hypercube. The author first pursued inductive clause generation (in FSIS) because of the parallel with linear inequalities.

Acknowledgments. Thanks to Armin Biere, Zyad Hassan, Fabio Somenzi, and Niklas Een for insightful discussions that shaped my thinking on how better to explain IC3, and to the first three for reading drafts of this paper.

References

[1] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, 1999.
[2] A. R. Bradley. k-step relative inductive generalization. Technical report, CU Boulder, Mar. 2010. `http://arxiv.org/abs/1003.3649`.
[3] A. R. Bradley. SAT-based model checking without unrolling. In *VMCAI*, pages 70–87, Jan. 2011.
[4] A. R. Bradley and Z. Manna. Checking safety by inductive generalization of counterexamples to induction. In *FMCAD*, Nov. 2007.
[5] A. R. Bradley and Z. Manna. Verification Constraint Problems with Strengthening. In *ICTAC*, Nov. 2006.
[6] A. R. Bradley, F. Somenzi, Z. Hassan, and Y. Zhang. An incremental approach to model checking progress properties. In *FMCAD*, Nov. 2011.
[7] M. A. Colon, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, 2003.
[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*, 1977.
[9] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, January 1978.
[10] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *CAV*, 1997.
[11] Z. Hassan, A. R. Bradley, and F. Somenzi. Incremental, inductive CTL model checking. In *CAV*, July 2012.
[12] F. Lu, M. K. Iyer, G. Parthasarathy, L.-C. Wang, K.-T. Cheng, and K.C. Chen. An Efficient Sequential SAT Solver With Improved Search Strategies. In *DATE*, 2005.
[13] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
[14] K. L. McMillan, Applying SAT Methods in Unbounded Symbolic Model Checking. In *CAV*, July 2002.
[15] K. L. McMillan, Interpolation and SAT-based model checking. In *CAV*, July 2003.
[16] K. L. McMillan. *Symbolic Model Checking*. Kluwer, Boston, MA, 1994.
[17] S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable Analysis of Linear Systems using Mathematical Programming. In *VMCAI*, 2005.
[18] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *FMCAD*, Nov. 2000.
[19] F. Somenzi and A. R. Bradley. IC3: Where monolithic and incremental meet. In *FMCAD*, Nov. 2011.