Abstract—IC3, a well-known model checker, proves a property of a transition system $\xi$ by building a sequence of formulas $F_0, \ldots, F_k$. Formula $F_i$, $0 \leq i \leq k$ over-approximates the set of states reachable in at most $i$ transitions. The basic algorithm of IC3 cannot guarantee that the value of $k$ never exceeds the reachability diameter of $\xi$. We describe an algorithm called IC4 that gives such a guarantee. (IC4 stands for "IC3 + Improved Convergence"). One can argue that the average convergence rate of IC4 is better than for IC3 as well. Improving convergence can facilitate some other variations of the basic algorithm. As an example, we describe a version of IC4 employing property decomposition. The latter means replacing an original (strong) property with a conjunction of weaker properties to prove by IC4. We argue that addressing the convergence problem is important for making the property decomposition approach work.

I. INTRODUCTION

IC3 is a model checker [2] that has become very popular due to its high scalability. Let $\xi$ be a transition system and $P$ be a safety property of $\xi$. IC3 builds a sequence of formulas $F_0, \ldots, F_k$ where $F_i$ over-approximates the set of states reachable from an initial state of $\xi$ in at most $i$ transitions. Property $P$ is proved when $F_i$ becomes an inductive invariant of $\xi$ for some $0 \leq i \leq k$.

One of the reasons for high performance of IC3 is that the value of $k$ above is typically much smaller than $\text{Diam}(\xi)$ (i.e. the reachability diameter of $\xi$). So, on average, IC3 converges to an inductive invariant much faster than an RA-tool (where RA stands for “reachability analysis”). Interestingly, the worst case behavior of an RA-tool and IC3 is quite different from their average behavior. Namely, IC3 cannot guarantee that $k$ never exceeds $\text{Diam}(\xi)$. We introduce a modification of IC3 called IC4 that fixes the problem above. (IC4 stands for “IC3 + Improved Convergence”). On one hand, IC4 has the same worst case behavior as an RA-tool. On the other hand, the average convergence rate of IC4 is arguably better than that of IC3 as well.

The main difference between IC4 and IC3 is as follows. IC3 checks if formula $F_k$ is an inductive invariant by “pushing” the clauses of $F_k$ to $F_{k+1}$. If every clause of $F_k$ can be pushed to $F_{k+1}$, the former is an inductive invariant. Otherwise, there is at least one clause $C \in F_k$ that cannot be pushed to $F_{k+1}$. In this case, IC3 moves on re-trying to push $C$ to $F_{k+1}$ when new clauses are added to $F_k$. In contrast to IC3, IC4 applies extra effort to push $C$ to $F_{k+1}$. Namely, it derives new inductive clauses to exclude states that prevent $C$ from being pushed to $F_{k+1}$. This extra effort results either in successfully pushing $C$ to $F_{k+1}$ or in proving that $C$ is “unpushable”.

The proof of unpushability consists of finding a reachable state $s$ that satisfies formula $F_{k+1}$ and falsifies clause $C$.

The existence of $s$ means that $F_k$ cannot be turned into an inductive invariant by adding more clauses. Thus, semantically, the difference between IC4 and IC3 is that the former starts building a new over-approximation $F_{k+1}$ only after it proved that adding one more time frame is mandatory. Operationally, IC4 and IC3 are different in that IC4 generates a small set of reachable states.

An appealing feature of IC3 is its ability to generate property-specific proofs. So it seems natural to decompose a hard property $P$ into a conjunction $P_1 \land \ldots \land P_m$ of weaker properties and then generate $m$ property-specific proofs for $P_i$. However, the convergence issues of IC3 are arguably more pronounced for weak properties (see Subsection [VI.B]). So, to make property decomposition work, one should use IC4 rather than IC3 to prove properties $P_i$. In this paper, we describe a variation of IC4 employing property decomposition.

At the time of writing the first version of the paper we were not aware of QUIP, a version of IC3 published at [1]. We fix this omission and describe the relation between IC4 and QUIP in Subsection [VI.A]. QUIP more aggressively than the basic IC3 pushes clauses to future time frames and generates reachable states as a proof that a clause cannot be pushed. However, no relation of QUIP’s good performance with improvement of its convergence rate has been established either theoretically or experimentally.

The contribution of this paper is as follows. First, we show the reason why IC3 has a poor upper bound on the convergence rate (Section [III]). Second, we formulate a new version of IC3 called IC4 (Section [IV]) that is meant for fixing this problem. In particular, we show that IC4 indeed has a better upper bound than IC3 (Section [V]). We also give an estimate of the number of reachable states IC4 has to generate (Section [VI]). Third, we discuss arguments in favor of IC4 (Section [VII]). Fourth, we describe IC4-PD, a version of IC4 meant for solving hard problems by property decomposition (Section [VIII]).

II. A BRIEF OVERVIEW OF IC3

Let $I$ and $T$ be formulas specifying the initial states and transition relation of a transition system $\xi$ respectively. Let $P$ be a formula specifying a safety property of $\xi$. IC3 proves $P$ by building a set of formulas $F_0, \ldots, F_k$. Here formula $F_i$, $0 \leq i \leq k$ depends on the set of state variables of $i$-th time frame (denoted as $S_i$) and over-approximates the set of states.

1We assume that all formulas are propositional and are represented in CNF (conjunctive normal form)
2A state is an assignment to the set of state variables.
reachable in at most \(i\) transitions. That is every state reachable in at most \(i\) transitions is an \(F_i\)-state.

IC3 builds formula \(F_k\) as follows. Formula \(F_0\) is always equal to \(I\). Every formula \(F_k, k > 0\) is originally set to \(P\). (So \(F_0 \rightarrow P\) is always true because the only modification applied to \(F_k\) is adding clauses.) Then IC3 tries to exclude every \(F_k\)-state that is a predecessor of a bad state \(^4\). That is a state \(s\) that breaks \(F_k \land T \rightarrow P'\). Here \(T\) is a short for \(T(S_k, S_{k+1})\) and \(P'\), as usual, means that \(P\) depends on next-state variables i.e. those of \(S_{k+1}\). Exclusion of \(s\) is done by derivation of a so-called inductive clause \(C\) falsified by \(s\). Adding \(C\) to \(F_k\) excludes \(s\) from consideration. (If \(s\) cannot be excluded, IC3 generates a counterexample.)

One of the properties of formulas \(F_i\) maintained by IC3 is \(F_i \rightarrow F_{i+1}\). To guarantee this, IC3 maintains two stronger properties of \(F_i\): a) \(\text{Clauses}(F_{i+1}) \subseteq \text{Clauses}(F_i)\) and b) \(F_i \neq F_{i+1}\) implies that \(F_i \neq F_{i+1}\). That is the set of clauses of \(F_i\) contains all the clauses of \(F_{i+1}\) and the fact that \(F_i\) contains at least one clause that is not in \(F_{i+1}\) means that \(F_i\) and \(F_{i+1}\) are logically inequivalent. Since every formula \(F_i\) implies \(P\), one cannot have more than \(|P\text{-states}|\) different formulas \(F_0, \ldots, F_k\). That is if the value of \(k\) exceeds \(|P\text{-states}|\), there should be two formulas \(F_{i-1}, F_i, i < k\) such that \(F_{i-1} = F_i\). This means that \(F_{i-1}\) is an inductive invariant and property \(P\) holds.

### III. Convergence Rate Of IC3 And Clause Pushing

We will refer to the number of time frames one has to unroll before proving property \(P\) as the convergence rate. We will refer to the latter as \(\text{ConvRate}(P)\). As we mentioned in Section II an upper bound on \(\text{ConvRate}(P)\) of the basic version of IC3 formulated in [2] is \(|P\text{-states}|\). Importantly, the value of \(|P\text{-states}|\) can be much larger than \(\text{Diam}(\xi)\) (i.e. the reachability diameter of \(\xi\)). Of course, on average, \(\text{ConvRate}(P)\) of IC3 is much smaller than \(\text{Diam}(\xi)\), let alone \(|P\text{-states}|\). However, as we argue below, a poor upper bound on \(\text{ConvRate}(P)\) is actually a symptom of a problem.

Recall that formula \(F_k\) specifies an over-approximation of the set of states reachable in at most \(k\) transitions. So, it cannot exclude a state \(s\) reachable in \(j\) transitions where \(j < k\). (That is such a state \(s\) cannot falsify \(F_k\).) On the other hand, \(F_k\) may exclude states reachable in at least \(k+1\) transitions or more.

Suppose IC3 just finished constructing formula \(F_k\). At this point \(F_k \land T \rightarrow P'\) holds i.e. no bad state can be reached from an \(F_k\)-state in one transition. After constructing \(F_k\), IC3 invokes a procedure for pushing clauses from \(F_k\) to \(F_{k+1}\). In particular, this procedure checks for every clause \(C\) of \(F_k\) if implication \(F_k \land T \rightarrow C'\) holds. We will refer to this implication as the pushing condition. If the pushing condition holds for clause \(C\), it can be pushed from \(F_k\) to \(F_{k+1}\). If the pushing condition holds for every clause\(^5\) of \(F_k\), then \(F_k \land T \rightarrow F'_k\) and \(F_k\) is an inductive invariant.

Suppose that the pushing condition does not hold for a clause \(C\) of \(F_k\). Below, we describe two different reasons for the pushing condition to be broken. IC3 does not try to identify which of the reasons takes place. This feature of IC3 is the cause of its poor upper bound on \(\text{ConvRate}(P)\). Moreover, intuitively, this feature should affect the average value of \(\text{ConvRate}(P)\) as well.

The first reason for breaking the pushing condition is that clause \(C\) excludes a state \(s\) that is \(\text{reachable}\) in \((k+1)\)-th time frame from an initial state. In this case, formula \(F_k\) cannot be turned into an inductive invariant by adding more clauses. In particular, the broken pushing condition cannot be fixed for \(C\).

The second reason for breaking the pushing condition is that clause \(C\) excludes a state \(s\) that is \(\text{unreachable}\) in \((k+1)\)-th time frame from an initial state. In this case, every \(F_k\)-state \(q\) that is a predecessor of \(s\) can be excluded by deriving a clause falsified by \(q\). So in this case, the broken pushing condition can be fixed. In particular, by fixing broken pushing conditions for \(F_k\) one may turn the latter into an inductive invariant.

### IV. Introducing IC4

#### A. A high-level view of IC4

We will refer the version of IC3 with a better convergence rate described in this paper as IC4. The main difference between IC3 and IC4 is that the latter makes an extra effort in pushing clauses to later time frames. This new feature of IC4 is implemented in a procedure called NewPush (see Figure [1]). It is invoked after IC4 has built \(F_k\) where the predecessors of bad states are excluded i.e. as soon as \(F_k \land T \rightarrow P'\) holds. For every clause \(C\) of \(F_k\), NewPush checks the pushing condition (see Section III). If this condition is broken, NewPush tries to fix it or proves that it cannot be fixed and hence \(C\) is “unpushable”.

Depending on the clause-pushing effort, one can identify three different versions of IC4: minimal, maximal and heuristic. The minimal IC4 stops fixing pushing conditions as soon as NewPush finds a clause of \(F_k\) that cannot be pushed. After that the minimal IC4 switches into the “IC3 mode” where the pushing conditions are not fixed for the remaining clauses of \(F_k\). The maximal IC4 tries to fix the pushing condition for every inductive clause of \(F_k\). That is if a clause \(C \in F_k\) cannot be pushed to \(F_{k+1}\), the maximal IC4 tries to fix the pushing condition (regardless of how many unpushable clauses of \(F_k\) has already been identified). Moreover, if an inductive clause \(C\) is added to \(F_i, i < k\), the maximal IC4 tries to fix the pushing condition for \(C\) if it cannot be immediately pushed to \(F_{i+1}\).

A heuristic IC4 uses a heuristic to stay between minimal and maximal IC4 in terms of the clause-pushing effort. In this paper, we describe the minimal IC4 unless otherwise stated. So, when we just say IC4 we mean the minimal version of it.

---

3Given a formula \(H(S)\), a state \(s\) is said to be an \(H\)-state if \(H(s) = 1\).

4Given a property \(P\), a \(\overline{P}\)-state is called a bad state.
Let $F_k$ be a formula for which $NewPush$ is called when $k \geq Diam(\xi)$. At this point $F_k \land T \rightarrow P$ holds. Let $s$ be a state breaking the pushing condition for a clause $C$ of $F_k$. That is $s$ falsifies $C$ (and hence it is not an $F_k$-state) but is reachable from an $F_k$-state in one transition.

Recall that $F_k$ is an over-approximation of the set of states that can be reached in at most $k$-transitions. Since $s$ falsifies $F_k$, reaching it from an initial state of $\xi$ requires at least $k+1$ transitions. However, this is impossible since $k+1 > Diam(\xi)$ and hence state $s$ is unreachable. This means that every $F_k$-state that is a predecessor of $s$ can be excluded by an inductive clause added to $F_k$. So eventually, $NewPush$ will fix the pushing condition for $C$. After fixing all broken pushing conditions for clauses of $F_k$, $NewPush$ will turn $F_k$ into an inductive invariant.

VI. NUMBER OF REACHABLE STATES TO GENERATE

The number of generated reachable states depends on which of the three versions of IC4 is considered (see Subsection IV-A). Let $k$ denote the maximal number of time frames unfolded by IC4. In the case of the minimal IC4, the upper bound on the number of reachable states for proving property $P$ is equal to $k \times (k + 1)/2$. For the maximal IC4, the upper bound is $k \times |Unpush(F)|$ where $F = F_1 \cup \ldots \cup F_k$ and $Unpush(F)$ is the subset of $F$ consisting of unreachable clauses. Indeed, an inductive clause $C \in F_i$ is proved unpushable only once. This proof consists of a trace to a state falsified by $F_i$. The length of this trace is equal to $i$ and hence bounded by $k$. The upper bound for the maximal IC4 above is loose because one assumes that

- the length of every trace proving unpushability equals $k$
- two (or more) clauses cannot be proved unpushable by the same reachable state.

Re-using reachable states can dramatically reduce the total number of reachable states one needs to generate. For instance, for the minimal IC4, this number can drop as low as $k$. For the maximal IC4, the total number of reachable states can go as low as $m + k$ where $m$ is the total number of reachable states generated to prove the unpushability of clauses of $Unpush(F)$.

VII. A FEW ARGUMENTS IN FAVOR OF IC4

In this section, we give some arguments in favor of IC4. The main argument is given in Subsection VII-A where we relate IC4 with a model checker called QUIP. The latter was introduced in [11] in 2015. In Subsections VII-B and VII-C we describe a few potential advantages of IC4 that were not discussed in [11] (in terms of QUIP).

6Recall that at this point of the algorithm, no bad state can be reached from an $F_k$-state in one transition.

7For every formula $F_i$, $i = 1, \ldots, k$, IC4 generates one reachable state $s$ falsifying a clause of $F_i$. To reach $s$, one needs to generate a trace of $i$ states. So the number of reachable states generated for $F_i$ is equal to $i$. The total number of reachable states is equal to $1 + 2 + \ldots + k$.

8As we mentioned in the introduction, at the time of writing the first version of our paper we were not aware of QUIP.
A. IC4 and QUIP

As we mentioned in the introduction, QUIP makes an extra effort to push clauses to future time frames. To show that a clause cannot be pushed, QUIP generates a reachable state. Although the premise of QUIP is that the strategy above may lead to a faster generation of an inductive invariant, this claim has not been justified theoretically. The advantage of QUIP over IC3 is shown in Table 1 in terms of better run times and a greater number of solved problems. So, no direct experimental data is provided on whether QUIP has a better convergence rate than IC3. (As mentioned in Table 1 and in the first version of our paper, having at one’s disposal reachable states facilitates construction of better inductive clause.\textsuperscript{9} So one cannot totally discard the possibility that the performance of QUIP is mainly influenced by this “side effect”.) Nevertheless, great experimental results of QUIP is an encouraging sign.

B. Proving weak properties

In this subsection, we argue that IC4 should have more robust performance than IC3 on weak properties. Let $F_i$ be an over-approximation of the set of states reachable in at most $i$ transitions and $P$ be the property to prove. As we mentioned earlier, there are two conditions one needs to satisfy to turn $F_i$ into an inductive invariant: $F_i \land T \to P'$ and $F_i \land T \to F_i'$. We will refer to a state $s$ breaking the first condition (respectively second condition) as a state of the first kind (respectively second kind). Only states of the first kind (i.e. $F_i$-states from which there is a transition to a bad state) are explicitly excluded by IC3. States of the second kind are excluded implicitly via generalization of inductive clauses. On the other hand, IC4 excludes states of both kinds explicitly and implicitly (via generalization of inductive clauses).

First, assume that $P$ is a strong property meaning that there is a lot of bad states. Then by excluding states of the first kind coupled with generalization of inductive clauses, IC3 also excludes many states of the second kind. Now assume that $P$ is a weak property that has, say, only one bad state. Let us also assume that excluding states reaching this bad state is easy. Intuitively, in this case, IC3 is less effective in excluding the states of the second kind (because their exclusion is just a side effect of excluding states of the first kind). On the other hand, IC4 does not have this problem and so arguably should have a more robust behavior than IC3 when proving weak properties.

C. Test generation

Formal verification of some properties of transition system $\xi$ does not guarantee that the latter is correct.\textsuperscript{10} In this case, testing is employed to get more confidence in correctness of $\xi$. Traces generated by IC4 can be used as tests in two scenarios. First, one can check that reachable states found by IC4 satisfy

\begin{Verbatim}
\textbf{IC4-PD}(1, T, P)\{
1 \textbf{Inv} := \emptyset
2 \textbf{while (true) } \{
3 \textbf{s} := \text{CheckSat}(\text{Inv} \land \text{P})
4 \text{if } (\text{s} = \text{nil}) \text{return}(\text{Inv}, \text{nil})
5 \text{Q} := \text{FormProp(s)}
6 \text{(J, Cex)} := \text{IC4}^* (1, T, \text{P, Inv, Q})
7 \text{if } (\text{Cex} = \text{nil}) \text{return}(\text{nil, Cex})
8 \text{if } (\text{J} = \text{Q})
9 \text{J} := \text{Strengthen}(1, T, \text{Inv, J})
10 \text{Inv} := \text{Inv} \land \text{J} \} \}
\end{Verbatim}

the properties that formal verification tools failed to prove. Second, one can just inspect the states visited by $\xi$ and the outputs produced in those states to check if they satisfy some (formal or informal) criteria of correctness.

VIII. INTRODUCING IC4-PD

In this section, we present IC4-PD, a version of IC4 employing property decomposition. In Subsection VIII-A we describe two obstacles one has to overcome to make property decomposition work. Subsection VIII-B introduces a straightforward implementation of IC4-PD.

A. Property decomposition: two obstacles to overcome

As we mentioned in the introduction, an appealing feature of IC3 is its ability to generate property-specific proofs. Let $P$ be a hard property to prove. Let $P$ be represented as $P_1 \land \cdots \land P_k$ (i.e. $P$ is decomposed into $k$ weaker properties). Let $J_k$ be an inductive invariant for property $P_k$. Then $J_1 \land \cdots \land J_k$ is an inductive invariant for property $P$. So one can prove $P$ via finding property-specific proofs $J_i$, $i = 1, \ldots, k$.

To make the idea of property decomposition work one has to overcome at least two obstacles. The first obstacle is that the search space one has to examine to prove $P_i$ is, in general, not a subsume of the search space for $P$. In Section VII-B, weak properties are more likely to expose the convergence rate problem of IC3. For that reason, replacing a strong property $P$ with weaker properties $P_i$ may actually lead to performance degradation if properties $P_i$ are proved by IC3. On the other hand, IC4 should be more robust when solving weak properties. So one can address the second obstacle by using IC4 (rather than IC3) to prove properties $P_i$.

\textsuperscript{9}This obstacle is of a general nature and is not caused by using IC3.
\textsuperscript{10}The reason is that when proving $P_i$ one may need to consider traces that contain two and more $T$-states. These traces break property $P$ without breaking property $P_i$.
\textsuperscript{11}To prove that $P_i$ holds globally one needs to show that no trace of $P_i$-states reaches a $P_i$-state. Proving $P_i$ locally means showing that no trace of $P$-states (rather than $P_i$-states) reaches a $P_i$-state. As we show in Section VII-B, if $P$ is false, there is property $P_i$ that breaks both globally and locally. So if every $P_i$ holds locally, then it does globally too and $P$ is true.
B. Description of IC4-PD

The pseudocode of IC4-PD is shown in Fig. 2. IC4-PD accepts formulas $I, T, P$ specifying the initial states, the transition relation and the property to prove respectively. IC4-PD returns either an inductive invariant $Inv$ or a counterexample $Cex$. Computation is performed in a while loop. First, IC4-PD checks if there is a $\overline{P}$-state $s$ breaking $Inv \rightarrow P$ (line 3). If not, then $Inv$ is an inductive invariant proving $P$ (line 4). Otherwise, IC4-PD forms a new property $Q$ to prove (line 5). $Q$ consists of one clause, namely, the longest clause falsified by $s$. So, the latter is the only $Q$-state.

Then IC4-PD calls IC4*, a version of IC4 that proves $Q$ locally with respect to the target property $P$ (see Subsection VIII-A). That is IC4* checks is there is a trace of $P$-states (rather than $Q$-states) leading to the $\overline{Q}$-state. If not, then $Q$ holds locally. IC4* uses the current $Inv$ as a constraint. Namely, IC4* looks for a formula $J$ satisfying $Inv \land J \land T \rightarrow J'$ (rather than $J \land T \rightarrow J'$).

If IC4* finds a counterexample $Cex$, then $Q$ and hence $P$ fail (line 7). Otherwise, IC4* returns an inductive invariant $J$. If $Q$ is itself an inductive property (and so $J = Q$), IC4 tries to strengthen $J$ like an inductive clause is strengthened by IC3 (line 9). This is done to avoid enumerating $\overline{P}$-states one by one if many properties $Q$ turn out to be inductive. If $J$ is already strengthened (and so $J \neq Q$), then $Inv$ is replaced with $Inv \land J$ and a new iteration begins.

REFERENCES

[1] A. Ivrii and A. Gurfinkel. Pushing to the top. FMCAD-15, pages 65–72, 2015.
[2] A. R. Bradley. Sat-based model checking without unrolling. In VMCAI, pages 70–87, 2011.
[3] E. Goldberg. Complete test sets and their approximations. In FMCAD-18. To be published.
[4] E. Goldberg. Complete test sets and their approximations. Technical Report arXiv:1808.05750 [cs.LO], 2018.
[5] E. Goldberg, M. Gudemann, D. Kroening, and R. Mukherjee. Efficient verification of multi-property designs (the benefit of wrong assumptions). In DATE '18, pages 43–48, Dresden, Germany, 2018.

---

14 Proving $Q$ locally addresses the first obstacle mentioned in Subsection VIII-A. The second obstacle is addressed by using IC4 instead of IC3.

15 It is safe to do because all reachable states satisfy $Inv$. 