Model Checking with IC3

Sebastian Wolff[1]

1 **Technische Universität Kaiserslautern**
Germany
s_wolff09@cs.uni-kl.de

Abstract

Model checking has become popular in the past years and is used to verify both hardware and software. Previous algorithms solve the model checking problem by rephrasing it as a reachability problem which can be reduced to SAT. Those reductions use an unwinding of the transition relation and confront the SAT solver with huge and complex formulas. A novel model checking algorithm, called IC3, follows a different approach for the reduction to SAT. Instead of an unwinding, it computes stepwise over-approximations of reachability information. It then focuses on single states and proves or disproves their reachability in a backward-search fashion. During this procedure only single steps of the transition relation are considered. As a result, IC3 confronts the SAT solver with many simple queries. This new approach proved to be highly efficient and able to outperform state-of-the-art model checkers.

This paper gives an introduction to IC3. Its original version is, however, limited to finite state systems. Since software usually introduces infinite data domains, like the integers, an adaptation is needed to handle infinite state systems. To that end, two CEGAR approaches are presented which employ predicate abstraction to lift IC3 from finite to infinite state transition systems.

Lastly, an approach for combining IC3 and a state-of-the-art model checking algorithm is presented.

1998 ACM Subject Classification D.2.4 Model Checking

Keywords and phrases Model Checking – IC3 – SAT based – finite state systems – abstraction

1 Introduction

While systems grow more and more complex their correctness is more and more in question. One possibility to address this question is model checking. It allows to prove or disprove some desired property of a given state transition system. To that end, a reachability problem is solved asking for the reachability of states violating that property.

However, model checking has always been a discipline where the approach of a human verifier differed fundamentally from the approach of model checking algorithms. The latter one is described as monolithic, whereas the human approach has an incremental character [4]. Humans tend to build up some basic knowledge of a system, like lemmas about the variable domains. Then, more abstract, yet low-level, lemmas follow which directly invoke the previously generated ones. This principle is continued with the resulting lemmas becoming more and more general, being no longer restricted to local aspects of the system. If a lemma cannot be proved with the set of lemmas generated so far, further intermediate lemmas are added. Only then, these high-level lemmas are combined to show a desired property [4].

Additionally, there is no limitation to the diversity of those lemmas. Everything which might be useful can be phrased as a lemma to support the progress of proving an overall property. This approach differs a lot from the approach taken by a model checking algorithm. Most notably, the diversity of the lemmas used by an algorithm is limited. But also the strategy of modern model checkers is not incremental. They focus on global aspects and try to prove or disprove some property with few but very large and complex SAT queries [3, 5, 2].

IC3 address these disadvantages and implements a human inspired algorithm. Many lemmas are computed which are based on each other and generated with simple SAT queries [3]. But naturally, IC3 also suffers from a limited diversity of lemmas.

The rest of the paper is structured as follows. Section 2 summarizes important definitions used throughout the paper. An introduction to IC3 based on *SAT-based Model Checking Without Unrolling* and *Understanding IC3* is given in Section 3. Section 4 presents three extension to IC3: a CEGAR implementation according to *IC3 Modulo Theories via Implicit Predicate Abstraction*, a CEGAR variation due to *Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR)*, and a combination of state-of-the-art model checking and IC3 following *Interpolating Property Directed Reachability*. Section 5 concludes the paper.

2 Definitions

A *logic formula* is considered to be in conjunctive normal form (CNF). That is, it is a conjunction of clauses where a *clause* is a disjunction of literals. A *literal* is either a negated or a non-negated variable. We write $F(X)$ to indicate that formula F ranges over the set of variables X. If clear from the context, we simply write F.

A *finite state transition system* is a triple $S = (X, I, T)$ consisting of: *(i)* a set of state variables X; *(ii)* a set of initial states described by the logic formula $I(X)$; and *(iii)* a transition relation expressed by the logic formula $T(X, X')$.

With X^i we denote the set obtained by adding i primes to each variable from X. For brevity, we sometimes write X' instead of X^1 and F' instead of $F(X^1)$.

A *state* s of a system is an assignment to all variables in X represented as a conjunction of literals. Note that the negation of an assignment, $\neg s$, is a clause. We say that s is an F state if s satisfies F, i.e. $s \models F$. An *inductive generalization* of s is a (sub-) clause c of s with $c \Rightarrow s$.

A *safety property* is a logic formula which is satisfied by all reachable states of a given transition system. We say that a state is a *bad state* if it violates a given safety property.

An *inductive assertion* for a transition system is a formula F which satisfies $I \Rightarrow F$ and $F \wedge T \Rightarrow F'$. The former condition, called *initiation*, ensures that all initial states are covered by F, the latter one, called *consecution*, states that F is closed under the transition relation. We also say that F is *inductive* when both conditions are met. A *Counterexample To Induction (CTI)* is a state which can reach a bad state in one step.

A formula F is an *inductive strengthening* of a safety property P if $F \wedge P$ is inductive. A formula F is *inductive relative* to another formula G if both $I \Rightarrow F$ and $G \wedge T \wedge F \Rightarrow F'$ hold.

3 IC3 – Model Checking Finite State Transition Systems

IC3[1] checks whether a finite state transition system $S = (X, I, T)$ obeys a safety property P. The basic algorithmic idea is rather simple. In order to prove P being a safety property, IC3 generates an inductive strengthening F of P covering all initial states [3, 4]. The existence of such a strengthening proves P, because: *(i)* $F \wedge P$ can not reach a $\neg P$ state since $F \wedge P$ is closed under T by the definition of inductiveness; and *(ii)* $F \wedge P$ covers all initial states by construction.

[1] short for **I**ncremental **C**onstruction of **I**nductive **C**lauses for **I**ndubitable **C**orrectness

Listing 1 shows IC3s top-level function. First of all, the absence of 0-step and 1-step counterexamples is ensured via the two SAT queries at line 2. That is, all initial states have to satisfy P and may not reach a $\neg P$ state in one step. Otherwise execution stops.

Listing 1 IC3s main loop [3]. $S = (X, I, T)$, P and k are assumed to be in scope globally.

```
1  bool prove():
2      if unsat(I ⇒ P) or unsat(I ∧ T ⇒ P): return False
3      F_0 := I, F_1 := P, k := 1
4      while True:
5          if not extendFrontier(): return False
6          propagateClauses()
7          if F_i = F_{i+1} for some 1 ≤ i ≤ k: return True
8          k := k + 1
```

Towards an inductive strengthening, IC3 maintains a sequence $F_0, \ldots, F_k$ of frames. Each such frame F_i is a formula representing an over-approximation of the states reachable in up to i steps. After the initial checks, this sequence is initialized to $F_0 = I, F_1 = P$ and the level of the frontier is set to $k = 1$. The main loop then steadily extends and refines the frames while preserving the following properties [3, 4, 2]:

$(\mathbf{P_1})$	$I \Rightarrow F_0$	
$(\mathbf{P_2})$	$F_i \Rightarrow F_{i+1}$	for $0 \leq i < k$
$(\mathbf{P_3})$	$F_i \Rightarrow P$	for $0 \leq i \leq k$
$(\mathbf{P_4})$	$F_i \wedge T \Rightarrow F_{i+1}$	for $0 \leq i < k$

Before going into the details of `extendFrontier` and `propagateClauses`, consider the following three questions. First, does the initial sequence already obey the above properties? Yes, as (P_1) holds due to $F_0 = I$; (P_2) and (P_3) hold due to $F_0 \Rightarrow P$ (by first initial check) and $F_1 = P$; and (P_4) holds due to $F_0 \wedge T \Rightarrow F_1$ (by second initial check).

Secondly, why are we interested in those properties? Assume we are given a sequence $F_0, \ldots, F_i, F_{i+1}, \ldots F_k$ satisfying the above properties. Further assume that $F_i = F_{i+1}$ holds. Then F_i is closed under the transition relation, i.e. $F_i \wedge T \Rightarrow F_i'$, by ($P_4$) and $F_i = F_{i+1}$. Furthermore, F_i contains all initial states, i.e. $I \Rightarrow F_i$, by (P_1) and (P_2). Thus, F_i is inductive. And by (P_3) also $F_i \wedge P$ is inductive. Altogether, we identified F_i as inductive strengthening of P and understand why line 7 is a proper stopping rule.

Thirdly, why is the loop guaranteed to terminate? By (P_2), the frame sequence is non-decreasing. Suppose the loop does not terminate. This implies that the sequence is not just non-decreasing, but strictly increasing due to the check in line 7. As the transition system is finite, this sequence must stabilize after at most $\#states + 1$ steps [3, 4].

Altogether, properties (P_1)-(P_4) and termination yield total correctness of the algorithm.

▶ **Theorem 1.** *Given a finite state transition system S and a formula P, IC3 terminates and returns* `true` *if and only if P is a safety property of S.*

Extending the Frontier Assume we are in iteration k in a call of `extendFrontier` the code of which is given in Listing 2. The sequence $F_0, \ldots, F_k$ satisfies properties (P_1)-(P_4).

Listing 2 IC3s CTI detection [3].

```
9  bool extendFrontier():
10     F_{k+1} := P
11     while sat(F_k ∧ T ∧ ¬P'):
12         try:
13             s := predecessor of ¬P extracted from SAT witness
14             removeCTI(s)
15         except Counterexample:
16             return False
17     return True
```

First, a new frame $F_{k+1} = P$ is added. Next, we want the extended sequence to obey properties (P_1)-(P_4) again. Obviously, (P_1) is not affected and (P_3) holds by construction. The reason for (P_2) is a bit more involved. It will actually hold in the end, since we will modify frames only by adding a clause to all frames $F_0, \ldots, F_i$, for some i [3]. So the crucial part is preserving (P_4) for F_k, i.e. $F_k \wedge T \Rightarrow F'_{k+1}$.

Assume the implication holds. Then the new sequence obeys (P_1)-(P_4) and `extendFrontier` returns without taking any other action.

Now assume the implication does not hold. Hence, the SAT query to $F_k \wedge T \wedge \neg P'$ from line 11 reveals a proper F_k state[2] s which can reach a $\neg P$ state in one step – s is a CTI. To establish (P_4) we remove s from F_k without violating (P_1)-(P_3) [3, 4, 11]. The removal is done by a call to `removeCTI` the description of which is in order.

Removing a CTI In IC3 the occurrence of a CTI s in the frontier F_k can have two reasons: either s is reachable from I or it is not yet discovered that s is not reachable [2]. The first case actually disproves P as a path from I via s to $\neg P$ exists. In the second case, we have to prove that s is not reachable by augmenting the frame sequence such that s is excluded from F_k [3]. We say that we have the *proof obligation* $\langle s, k \rangle$ [2].

Listing 3 CTI removal [3, 2], i.e. proof obligation handling in backward-search fashion.

```
18 void removeCTI(s : state)
19     states = {⟨s, k⟩}
20     while states not empty:
21         ⟨q, i⟩ = pop element of states that minimizes i
22         if sat(F_0 ∧ T ∧ ¬q ∧ q'):
23             raise Counterexample
24         // here, q is at least inductive relative to F_0
25         j := maximal j with not sat(F_i ∧ T ∧ ¬q ∧ q')
26         c := inductiveGeneralization(¬q)
27         for l from 0 to j + 1:
28             F_l := F_l ∧ c
29         if j ≥ i − 1:
30             break  // proof obligation fulfilled
31         w := witness for sat(F_{j+1} ∧ T ∧ ¬q ∧ q')
32         t := predecessor of q extracted from w
33         states := states ∪ ⟨t, j + 1⟩
34         // ⟨t, j + 1⟩ might not resolve ⟨q, i⟩ ⇝ check again
35         states := states ∪ ⟨q, i⟩
```

[2] s is said to be a proper F_k state if it is no F_{k-1} state

IC3 maintains a whole set of such proof obligations. Each entry $\langle q, i\rangle$ tells us to eliminate state q from frame F_i. To do so, we seek $\neg q$ to be inductive relative to F_{i-1}. With this relation established we can conjoin c to frames $F_0, \ldots, F_i$, with $\neg q \Rightarrow c$.[3][4] This removes q from F_i. One could chose $c = \neg q$, but this might be inefficient as only the single state q would be removed. This is why IC3 chooses c to be a (minimal) inductive generalization of $\neg q$. This generalization removes not just q but also other q-*like* states [3, 4, 2].

Let us now turn to the processing of a proof obligation according to Listing 3. Therefore, consider an obligation $\langle q, i\rangle$, with q being able to reach a $\neg P$ state. Our goal is to remove q from F_i.

First, check whether $\neg q$ is inductive relative to F_0. If it is not, then an initial state exits which can reach q and thus $\neg P$. This disproves P and we can abort. Otherwise, find a maximal j such that $\neg q$ is inductive relative to F_j.[5] Due to the first check, such j must exist.

Next, we can conjoin an inductive generalization of $\neg q$ to $F_0, \ldots, F_{j+1}$, as described above. If $j \geq i - 1$, we are done. Otherwise, we know that consecution failed. That is, $F_{j+1} \wedge T \wedge \neg q \wedge q'$ is satisfiable and a SAT query provides a witness. This witness is key for the procedure as it reveals a F_{j+1} state t which is a predecessor of q. We now have to refocus on t as q is reachable through it. Put differently: for the overall goal of proving that q is unreachable, we have to prove that t is unreachable. That is, we add another proof obligation, namely $\langle t, j + 1\rangle$, and repeat the above argument [3].

Eventually, obligation $\langle t, j + 1\rangle$ resolves and t is removed from F_{j+1}. Then $\langle q, i\rangle$ is considered again: the procedure is repeated. However, a SAT query to $F_{j+1} \wedge T \wedge \neg q \wedge q'$ will not produce t again as it is no longer contained in F_{j+1} due to the previous strengthening. Thus, the obligation is resolved this time, or a new predecessor is revealed which is treated just as t. [3, 2]

To finish the discussion of CTI removal, two last issues are addressed. First, note that the above algorithm is guaranteed to terminate, as shown in [3]. This is because no circular dependency chain among proof obligations is established by the choice of the next obligation (see line 21).

Secondly, towards correctness, let us briefly discuss why properties (P_1)-(P_4) are maintained. (P_1) is preserved as no I state is eliminated from F_0. This is due to the fact that every clause c added to F_0 is actually inductive relative to it (enforced by check in line 22). Next, (P_2) is not violated as clauses are always added to all frames $F_0, \ldots, F_i$, for some i. Property (P_3) continues to hold since the frames are only restricted. Lastly, (P_4) is enforced as predecessors are revealed (lines 31 and 32) and enqueued to be deleted in form of new proof obligations (line 33) until no more predecessors are found (line 35).

Clause propagation In order to steadily refine frames, i.e. reachability information, particular clauses are propagated forward. That is, some clause c of frame F_i is added to frame F_{i+1}. According to the implementation of `propagateClauses`, given in [3] and Listing 4, this procedure focuses on a single clause at time. That is, it checks for all clauses c of frame F_i whether c is inductive relative to F_i. If so, no F_i state can reach a state satisfying $\neg c$. Thus c can be added to F_{i+1} refining the over-approximation F_{i+1} since we proved $\neg c$ being unreachable in $i + 1$ steps [3].

[3] If $\neg q$ is inductive relative to some F_l and $\neg g \Rightarrow c$, then also c is inductive relative to F_l.

[4] As c is inductive relative to F_{i-1}, no F_{i-1} state can reach a $\neg c$ state, by definition. Hence, conjoining c to F_i is valid.

[5] If some formula F is inductive relative to some F_l, then it is also inductive relative to $F_0, \ldots, F_{l-1}$.

Listing 4 IC3s clause pushing [3].

```
36 void propagateClauses(state s, frame i)
37     for i = 1 to k:
38         for each clause c ∈ F_i:
39             if not sat(F_i ∧ T ∧ ¬c'): # same as not sat(F_i ∧ c ∧ T ∧ ¬c')
40                 F_{i+1} := F_{i+1} ∧ c
```

However, this is not the strongest possible refinement. It might be the case that two clauses are together inductive relative to F_i although neither is on its own. So one could compute a maximal set of F_i clauses which is inductive relative to F_i and add it to F_{i+1} [4].

4 IC3 Extensions for Infinite State Transition Systems

The previous section introduced IC3 for model checking finite state systems. Such systems are of major interest when dealing with hardware verification [3], for example. However, we want to extend this algorithm to handle infinite state systems as this allows for verifying more complex systems like software. To that end, we employ predicate abstraction which generates a finite abstraction from an infinite transition system. But model checking results for the abstract system might not carry over to the concrete system as the abstraction might be too imprecise. Thus, two CEGAR approaches are introduced which solve this issue by steadily refining the abstraction until a model checking result for the abstract system carries over to the concrete system.

Lastly, a state-of-the-art model checking approach is combined with IC3. Although it does only support finite state systems, like the original version of IC3, it may profit from previous results on model checking infinite state systems.

4.1 Predicate Abstraction

Predicate abstraction allows to generate a finite abstraction from an infinite state transition system $S = (X, I, T)$ [6]. The main idea is to divide the state space of S into finitely many equivalence classes. These equivalence classes resemble the states of the abstract system and are induced by a set of given predicates $\mathbb{P}$ [5]. That is, a state of the abstract system entails a truth value for each predicate. Intuitively, the abstraction of a state from S is computed by evaluating the predicates under the assignment that is represented by the given state.

Technically, to augment IC3 with predicate abstraction the following modifications are required [5]:

- **($\mathbf{M_1}$)** A SMT solver replaces the SAT solver.
- **($\mathbf{M_2}$)** Clauses, frames and states have to use predicates instead of propositional formulas in their literals.
- **($\mathbf{M_3}$)** An abstract transition relation replaces the concrete one.
- **($\mathbf{M_4}$)** Frames need to be initialized to a set of abstract states rather than concrete ones.

In the following we focus on (M_3) and (M_4) as the other modifications can be integrated easily [5, 2]. First, let us discuss how to compute the abstract transition relation $\hat{T}$. Clearly, it relates abstract states such that $\hat{t} \to_{\hat{T}} \hat{s}$ holds if there are concretizations t, s of $\hat{t}, \hat{s}$ with $t \to_T s$ as depicted in Figure 1 [5]. Technically, the abstract transition relation is defined as follows

$$\hat{T}(X, X') := EQ(X, \bar{X}) \wedge T(\bar{X}, \bar{X}') \wedge EQ(\bar{X}', X')$$

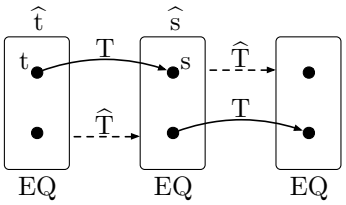
where $\bar{X}$ introduces a copy of all variables X, similar to X', and with

$$EQ(X,Y) := \bigwedge_{p\in\mathbb{P}} p(X) \leftrightarrow p(Y).$$

relating two concrete states to correspond to the same abstract state [5].

Secondly, consider the initialization of frames, i.e. $F_0 = I$ and $F_i = P$, for $i \geq 1$. Instead of initializing new frames to concrete states we rather need to initialize them to abstract states, i.e. a conjunction of predicates. For simplicity, however, we do not compute abstractions of I and P but require them to be composable with the predicates of $\mathbb{P}$. Put differently: we require $\mathbb{P}$ to contain all predicates that occur in I and P [5]. Hence, I and P represent both concrete and abstract states and the initialization does not need to be changed.

Figure 1 An Abstract path [5].



4.2 CEGAR

CEGAR[6] is an iterative approach for model checking infinite state systems. It checks whether a given infinite system S obeys a safety property P. To that end, it employs predicate abstraction to generate a finite abstraction $\hat{S}$. The CEGAR loop [5] then checks whether $\hat{S}$ obeys P. If so, also S does. Otherwise, a counterexample in $\hat{S}$ is found which is simulated in S. That is, it is checked whether the counterexample is present in S, too. If it is, then P is in fact disproved. Otherwise, the abstraction $\hat{S}$ is refined and the loop is repeated.

Listing 5 presents updated versions of the `extendFrontier` and `removeCTI` functions from Listings 2 and 3 from Section 3 which implement predicate abstraction and the CEGAR loop.

A crucial part of CEGAR is the simulation of an abstract counterexample path $\hat{s}_0\hat{s}_1\hat{s}_2\ldots\hat{s}_k$. To this end, it is checked whether a concretization $s_0s_1s_2\ldots s_k$ exists. According to [5], this can be done by checking the satisfiability of

$$I(X^0) \wedge \bigwedge_{0\leq i\leq k} T(X^i, X^{i+1}) \wedge P(X^{k+1}) \wedge \bigwedge_{0\leq i\leq k} s_k(X^k).$$

But this formula seems to be erroneous and should be changed to:

$$\underbrace{I(X^0)}_{(1)} \wedge \underbrace{\bigwedge_{0\leq i\leq k} T(X^i, X^{i+1})}_{(2)} \wedge \underbrace{\neg P(X^{k+1})}_{(3)} \wedge \underbrace{\bigwedge_{0\leq i\leq k} \hat{s}_i(X^i)}_{(4)}.$$

The satisfiability of the second formula entails a concrete counterexample as a witness path $s_0s_1s_2\ldots s_k$ has to satisfy (1)-(4). That is, the path has to start in an initial state, due to (1), two consecutive elements s_i and s_{i+1} have to be connected via the concrete transition relation, due to (2), and the last element reaches a violation of P, by (3). Condition (4) additionally restricts the search space to concretizations of the abstract counterexample path.

[6] short for **C**ounter-**E**xample-**G**uided **A**bstraction-**R**efinement

Listing 5 IC3 with predicate abstraction and the CEGAR loop, based on [5].

```
41 bool extendFrontier():
42     F̂_{k+1} = P
43     while sat_SMT(F̂_k ∧ T̂ ∧ ¬P'):
44         try:
45             ŝ := predecessor of ¬P extracted from SMT witness
46             removeCTI(ŝ)
47         except CounterexamplePath π:
48             if π has concretization: return False
49             else: ℙ := ℙ ∪ refine(π)
50     return True
51
52 void removeCTI(ŝ : state)
53     states = {⟨ŝ, k, ϵ⟩}
54     while states not empty:
55         ⟨q̂, n, π⟩ = pop element of states that minimizes n
56         if sat(F̂_0 ∧ T̂ ∧ ¬q̂ ∧ q̂'):
57             raise CounterexamplePath(π)
58         i := maximal i with not sat_SMT(F̂_i ∧ T̂ ∧ ¬q̂ ∧ q̂')
59         c := inductiveGeneralization(¬q̂)
60         for j from 0 to i+1:
61             F̂_j := F̂_j ∧ c
62         if i+1 ≥ n:
63             break
64         w := witness for sat_SMT(F̂_{i+1} ∧ T̂ ∧ ¬q̂ ∧ q̂')
65         t̂ := predecessor of q̂ extracted from w
66         states := states ∪ ⟨t̂, i+1, t̂ π⟩
67         states := states ∪ ⟨q̂, n, π⟩
```

After the simulation of an abstract counterexample, refinement is needed if no concretization could be revealed. The refinement has to be strong enough to rule out the abstract counterexample it was triggered by [5, 2]. Despite suggesting interpolation, no actual refinement approach is discussed in [5] as the refinement is independent of the overall approach.

4.3 CTIGAR

Traditional CEGAR approaches, as discussed above, focus on complete counterexample traces. As these traces can be arbitrarily long, the concretization can become very expensive requiring huge, complex SAT queries. While it might correspond conceptually to monolithic model checking approaches, it does not coincide with IC3s spirit of considering single step, single state counterexamples. Following this spirit, CTIGAR[7] applies CEGAR-like abstraction-refinement based on single CTIs [3, 5, 2]. In the following, we present a simplified version of CTIGAR which addresses refinement eagerly.

The key insight of CTIGAR is that the need for refinement can be recognized by considering single abstract CTIs instead of considering a complete counterexample path. As CTIs are handled by proof obligations, this process needs to be adapted to detect whether refinement is needed. According to [2] there are two such scenarios.

[7] short for **C**ounterexample **T**o **I**nduction-**G**uided **A**bstraction-**R**efinement

First, focus on the consecution query for abstract states. For an abstract state $\hat{s}$ and a frame F_i it is checked whether $\neg\hat{s}$ is inductive relative to F_i. This check, however, can fail although $\neg s$ is inductive relative to F_i. That is, $F_i \wedge T \wedge \neg\hat{s} \wedge \hat{s}'$ might be satisfiable while $F_i \wedge T \wedge \neg s \wedge s'$ is not [2]. This occurs if the abstraction is too weak to separate all concrete states represented by $\hat{s}$ from F_i. Put differently: $\hat{s}$ resembles some concrete states that are reachable in up to $i-1$ steps and states that are reachable in at least i steps. Hence, refinement is needed such that all states described by an abstract state are reachable in the same number of steps.

Secondly, a more involved problem arises. An abstract state $\hat{t}$ could represent both reachable and unreachable states at once [2]. This could be problematic if the unreachable subset of $\hat{t}$ reaches an abstract CTI $\hat{s}$, while the reachable subset does not. Thus, $\hat{t}$ would be classified as a bad state, too, and IC3 would try to delete it from the frame sequence. But since $\hat{t}$ covers reachable states, IC3 would erroneously conclude that a counterexample was discovered. As we do not want to expensively examine the counterexample trace, as done by CEGAR, we have to detect such misleading abstract states and suggest refinement to remove them. According to [2], this can be done by recognizing such abstract states $\hat{t}$ having concretizations t_1, t_2 with $t_1 \neq t_2$, $t_1 \to_T \hat{s}$ and $t_2 \to_T \neg\hat{s}$. Technically, this check can be accomplished by a SMT query to

$$RC(\hat{t},\hat{s}) := EQ(X,\bar{X}) \wedge \hat{t}(X) \wedge (T(X) \to \hat{s}(X')) \wedge \big(T(\bar{X}) \to \neg\hat{s}(\bar{X}')\big) \wedge \left(\neg \bigwedge_{x \in X} x \leftrightarrow \bar{x}\right).$$

Listing 6 IC3s proof obligation handling enriched with CITIGAR. Based on [2].

```
68  void removeCTI(ŝ : state)
69      states = {⟨ŝ, k⟩}
70      while states not empty:
71          ⟨q̂, n⟩ = pop element of states that minimizes n
72          if sat(F̂_0 ∧ T̂ ∧ ¬q̂ ∧ q̂'):
73              raise Counterexample
74          i := maximal i with not sat_SMT(F̂_i ∧ T̂ ∧ ¬q̂ ∧ q̂')
75          c := inductiveGeneralization(¬q̂)
76          for j from 0 to i+1:
77              F̂_j := F̂_j ∧ c
78          if i+1 ≥ n:
79              break
80          w := witness for sat_SMT(F̂_{i+1} ∧ T̂ ∧ ¬q̂ ∧ q̂')
81          t̂ := predecessor of q̂ extracted from w
82          if not sat(RC(t̂, ŝ)):
83              refine()
84          states := states ∪ ⟨t̂, i+1⟩
85          states := states ∪ ⟨q̂, n⟩
```

Listing 6 presents an updated version of the `removeCTI` function from Listing 3. A check for the second scenario is implemented and refinement is issued eagerly if needed. However, the first scenario is skipped. This is on purpose, as according to *Counterexample to Induction-guided abstract-refinement (CTIGAR)*, the eager refinement prevents the first scenario. Unfortunately, neither a discussion nor a reference supporting this statement is given in the previously mentioned paper.

Similar to the CEGAR case, refinement is not addressed here as it can be conducted independently of the overall approach.

4.4 Interpolation

The previous sections discussed IC3 and extensions which pursued IC3s concept. That is, they employed a local backward-search focusing on single states and guiding the search towards finding an inductive strengthening. This contrasts previous monolithic model checking techniques which focus on global properties in an unguided search [3, 4, 5, 2].

Following [11], an approach for combining both concepts is discussed. Therefore a brief introduction to monolithic model checking is given first, followed by a description of the combination of both techniques.

IMC To get a notion of monolithic model checking we briefly review IMC[8]. The overall structure of IMC is, however, somewhat similar to IC3. A sequence of formulas $I = F_0, F_1, \ldots, F_k$ is maintained that over-approximates the reachable states in up to k steps. Compared to IC3, this sequence is weaker as it has to satisfy only (P_3) and (P_4) from Section 3 [2, 11, 10].

IMC proceeds by extending the sequence to $I, F_1, \ldots, F_k, F_{k+1} = P$ and trying to reestablish the properties mentioned above. Therefore, a SAT query to

$$I(X^0) \wedge \left(\bigwedge_{0 \leq i \leq k} T(X^i, X^{i+1}) \right) \wedge \neg P(X^{k+1})$$

is issued [11, 10] which is similar to the CEGAR check but without any restrictions to the search space. A counterexample is revealed in case the above formula is satisfiable. Otherwise a strengthening can be computed with sequence interpolants [11].

▶ **Definition 2** (Sequence Interpolant). Let $A = (A_0, \ldots, A_n)$ be a tupel of formulas with $(A_0 \wedge \cdots \wedge A_n)$ being unsatisfiable. A sequence interpolant [10] for A is then a tuple $J = (J_1, \ldots, J_n)$ with: *(i)* $A_0 \Rightarrow J_1$; *(ii)* $J_i \wedge A_i \Rightarrow J_{i+1}$, for $0 < i < n$; *(iii)* $J_n \wedge A_n$ unsatisfiable; and *(iv)* J_i does only contain symbols appearing in both $A_0 \wedge \cdots \wedge A_i$ and $A_{i+1} \wedge \cdots \wedge A_n$.

To obtain a strengthening, compute an interpolant $J = (J_1, \ldots, J_{k+1})$ for

$$A = (\underbrace{I(X^0) \wedge T(X^0, X^1)}_{A_0}, \ldots, \underbrace{T(X^i, X^{i+1})}_{A_i}, \ldots, \neg \underbrace{P(X^{k+1})}_{A_{k+1}})$$

and form the new sequence $I, F_1 \wedge J_1, \ldots, F_{k+1} \wedge J_{k+1}$. The new sequence clearly obeys (P_3) if the old one did. And by $F_i \wedge T \Rightarrow F_{i+1}$ and property *(ii)* of the definition of sequence interpolants, also (P_4) is established.

The algorithm terminates, if a pair (i, j) is found with $i > j$ and $F_i \Rightarrow F_j$ [11]. Together with (P_4) we get $F_{i-1} \wedge T \Rightarrow F_i \Rightarrow F_j$. That is, F_i cannot reach a $\neg P$ state. Hence, P has been proved.

Integration with IC3 In the combination of IMC and IC3 as proposed in *Interpolating Property Directed Reachability* IC3s algorithmic flow and even its invariants are maintained [2, 3]. The change is restricted to the `extendFrontier` function the updated version of which is given in Listing 7. Instead of computing single CTIs a monolithic IMC strengthening is

[8] short for **I**nterpolation-sequence based **M**odel **C**hecking

employed. However, the approach is refined in the sense that the search space of the SAT query from above is restricted with respect to the frame sequence [2, 5, 11]. That is, all states that are known to be unreachable are removed from the search space. The modified SAT query is as follows [11]:

$$\left(\bigwedge_{0 \leq i \leq k} F_i(X^i) \wedge T(X^i, X^{i+1}) \right) \wedge \neg P(X^{k+1}).$$

Just like in IMC, a counterexample is revealed or a strengthening with respect to a sequence interpolant is generated.

Lastly, consider the IMC strengthening $F_0, F_1 \wedge J_1, \ldots, F_k \wedge J_k$. As this sequence might violate (P_2), a proper frame sequence needs to be constructed from the strengthening. An easy but inefficient solution [11] is to set

$$F_i = CNF\left(\bigvee_{i \leq j \leq k} F_j \wedge J_j \right)$$

where $CNF(\varphi)$ computes the conjunctive normal form of φ. In [11] a more efficient approach employing IC3s `extendFrontier` function is presented which we skip for brevity.

Listing 7 Monolithic frontier extension [11].

```
86 bool extendFrontier():
87     if sat(F_0(X^0) ∧ T(X^0, X^1) ∧ ··· ∧ F_k(X^k) ∧ T(X^k, X^{k+1}) ∧ ¬P(X^{k+1})):
88         return false
89     J_1, ..., J_k = sequenceInterpolant(F_0(X^0) ∧ T(X^0, X^1), ...,
90                                         F_k(X^k) ∧ T(X^k, X^{k+1}), ¬P(X^{k+1}))
91     for i from 0 to k:
92         F_i := cnf(F_i ∧ J_i ∨ ··· ∨ F_k ∧ J_k)
93     return True
```

5 Conclusion and Future Work

IC3 is a novel model checking algorithm which focuses on an incremental construction of an inductive strengthening while considering single counter examples in an backward-search fashion. By ranking third in the HWMCC'10 competition it proved to be competitive with state-of-the-art model checkers in both fast computations and number of solved benchmarks [3, 4, 11].

A further interesting aspect of IC3 is its adaptability to existing approaches. Implicit predicate abstraction is easily injected to IC3 while keeping all of its algorithmic flow and allowing to model check finite abstractions of infinite systems [2, 5]. With this abstraction it is possible to implement an incremental CEGAR version which seemingly integrates into IC3. However, CEGAR employs complex SAT queries to verify counterexamples [5]. To avoid this conceptual diversity, CTIGAR removes these complex checks in favor of simple SAT queries while preserving the overall CEGAR concept [2].

IMC follows the opposite approach of CTIGAR and introduces a monolithic aspect to IC3 yet preserving its algorithmic flow [11]. Lifting this approach to infinite state systems is possible future work.

This paper focused on the concepts of IC3 itself and possible extensions. However, all presented techniques were evaluated positively. That is, they are competitive to existing state-of-the-art approaches and are even able to outperform them on various benchmarks [3, 5, 2, 11]. Investigating the performance of the presented approaches and comparing them with each other is considered to be future work.

References

1 Armin Biere and Roderick Bloem, editors. *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*. Springer, 2014.

2 Johannes Birgmeier, Aaron R. Bradley, and Georg Weissenbacher. Counterexample to induction-guided abstraction-refinement (CTIGAR). In Biere and Bloem [1], pages 831–848.

3 Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.

4 Aaron R. Bradley. Understanding IC3. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2012.

5 Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 modulo theories via implicit predicate abstraction. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2014.

6 Ranjit Jhala and Rupak Majumdar. Software model checking. *ACM Comput. Surv.*, 41(4), 2009.

7 Kenneth L. McMillan. Interpolation and sat-based model checking. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

8 Markus Müller-Olm and Helmut Seidl. A generic framework for interprocedural analysis of numerical properties. In Chris Hankin and Igor Siveroni, editors, *Static Analysis, 12th International Symposium, SAS 2005, London, UK, September 7-9, 2005, Proceedings*, volume 3672 of *Lecture Notes in Computer Science*, pages 235–250. Springer, 2005.

9 Stefano Tonetta. Abstract model checking without computing the abstraction. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2009.

10 Yakir Vizel and Orna Grumberg. Interpolation-sequence based model checking. In *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*, pages 1–8. IEEE, 2009.

11 Yakir Vizel and Arie Gurfinkel. Interpolating property directed reachability. In Biere and Bloem [1], pages 260–276.