A GLIMPSE OF IC3 AND SAT-BASED MODEL CHECKING

Verification and validation techniques

Santi Enrico
University of Udine
Academic year 2022/23

Contents

1 Introduction

We begin by recalling and introducing some fundamental concepts such lattices, transition systems, their symbolic counterpart and inductive formulas. We also list some basic concepts that will be given for granted, i.e. literals in a formula, CNF formulas, SAT problem and how it can be solved (See [1]).

1.1 Lattices

A lattice is a structure $\mathcal{L} = \langle L, \sqcap, \sqcup \rangle$ in which L is a non empty partial ordered set and $\sqcap : L \times L \to L, \sqcup : L \times L \to L$ are two binary operators on the set elements. Let's now define the properties of the join and meet operators (i.e. $\sqcup$ and $\sqcap$), $x \sqcup y$ is defined to be the least upper bound of x and y, namely an element $s \in L$ such that $x \preceq s$, $y \preceq s$ and $\nexists s'\ s' \preceq s \wedge x \preceq s' \wedge y \preceq s'$. The meet operator is the greatest lower bound to x and y.
Let's now see two other concepts, the *up-sets* and *down-sets* of the elements.
Given an element $x \in L$ we define the up-set $\uparrow x := \{y \in L | x \preceq y\}$ and the down-set $\downarrow x := \{y \in L | x \succeq y\}$[2].

1.1.1 Subclause lattice

We now focus on a specific kind of lattice, the subclause lattice. Given a clause c we can define the lattice $\mathcal{L}_c = \langle L, \sqcap, \sqcup \rangle$ [3] where L is now the partial order induced by the subclauses of c with respect to the subclause relation[1], the join operator is the disjunction of the literals while the meet is the disjunciton only of the common literals.

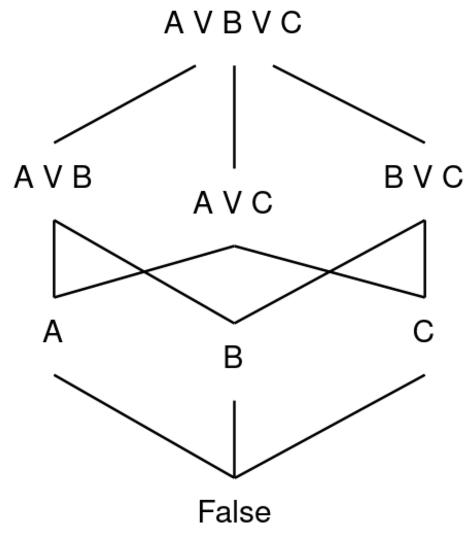

Figure 1: An example of the subclause lattice on the clause $\mathcal{C} = A \vee B \vee C$.

Notice that $\mathcal{L}_c$ is complete, thus the Knaster-Tarski Theorem applies, meaning that every monotonic function on $\mathcal{L}_c$ induces fixpoints which form a complete lattice, thus, there exist a greatest and least fixpoint.

[1]Such relation contains all possible clauses with a subset of literals of the original one[4].

1.2 Symbolic transition systems

A finite state transition system is a triple $M = \langle S, \Theta, T \rangle$ where S is the finite set of states, Θ is an assertion (e.g. a formula) which describes the possible initial states and T is the finite set of transitions (i.e. a map from states to sets of states) [5]. For simplicity they can be thought as graphs on which we have some "marked" (initial) nodes.

The description above it's quite abstract, let's now see a slightly different way to define such systems, by introducing symbolic transition systems [3]. These systems are again triples $M = \langle \bar{x}, \Theta, T \rangle$[2] where this time though the first component is a set of propositional (boolean) variables which will be used to model the set of states S, Θ is a propositional formula over $\bar{x}$, and T is a propositional formula over $(\bar{x}, \bar{x}')$ where $\bar{x}'$ represents the values of the variables in the successor state[3]. The states now are the possible assignments over $\bar{x}$. Related to this concept there is the idea of computation or trace, which is a sequence of assignments $s_0, s_1 ...$ to the variables $\bar{x}$ that satisfy initiality ($s_0 \models \Theta$) and consequentiality ($s_i, s_{i+1} \models T \ \forall i$) [6].

It's easy to see that the number of the states grows exponentially in the number of variables, and thus efficient model checking techniques are needed.

1.3 Model checking, BMC and safety

Model checking in general is a problem consisting in receiving in input a model M (with an initial state s[4]) and a formula/property P for which we want to check $M, s \models P$, meaning that we want to verify if all computations of M starting at s satisfy P. Simplifying a lot bounded model checking (BMC) is a technique for doing model checking consisting in (usually) unrolling the model for an incremental number of steps verifying that all the computations of increasing lengths satisfy the required property, BMC is tied to the concept of SAT solvers since such tools are used to establish whether a computation (of an increasing number of steps) falsifying the property exists.
In this context we will focus on safety properties, i.e. properties which express that only "safe" states are reachable.
It's intuitive to come up with examples of why such properties are important to verify in a system, for instance suppose we have an industrial machine which works within a certain temperature range. In the system characterization there's a variable representing its temperature and the machine has a component that when the temperature is too high cools the machine down. A safety property we could want may be the one stating that just states within the established temperature range are reachable (i.e. checking that the cooling system works properly).

In general given a formula P we say it's *M-invariant* if only states satisfying it are reachable during any computation. If P is not M-invariant then it exists a trace containing a state s_i such that $s_i \not\models P$ we call such computation counterexample[3].

[2]Also similar definitions where M is a quadruple exist, the additional component represents a set of input variables under the environment control, in such case also the transition function is slightly re-adapted.

[3]The system evolves, a successor of a state is a state immediately reachable from it.

[4]Suppose wlog just one initial state.

But how do we compute if P is M-invariant? Giving an answer to this questions would solve the model checking problem for safety properties since if we knew that only states $s_i \models P$ were reachable we would simply answer affirmatively to the original problem. The answer to this question is *induction*.

1.4 Reachability won't work

Before seeing induction let's think why reachability won't (in real scenarios) be a solution in finding whether P is M-invariant. In principle it should be sufficient to check whether the states falsifying P wouldn't be reachable from the starting states, by Savitch's theorem we know reachability can be done in $O(\log_2(n)^2)$ space[5]. The problem is that the graph in real word scenario would certainly be too big to be stored explicitly, thus making such algorithm not applicable.

[5]Where n is the size, or number of nodes in the graph.

2 Induction

A formula P is inductive if[6]:

- It holds for the initial states: $\Theta[\bar{x}] \Rightarrow P[\bar{x}]$
- Is closed under T (only P-states from P-states[4]): $T[\bar{x}, \bar{x}'] \wedge P[\bar{x}] \Rightarrow P[\bar{x}']$

The formula P describes a set of states, so being inductive means that in such set in order to satisfy the first condition (*initiation*) all initial states must be present[6]. For what concerns the second condition (*consecution*) we have that if in a given state the variables are such that satisfy P and they also represent a (admissible) partial assignment for T then also the possible successor states (i.e. the ones in which the variables complete the assignment for T) must satisfy P as well.

There's also the possibility for a formula P to be inductive with respect to another one (P') if the initiation condition is satisfied and $P'[\bar{x}] \wedge P[\bar{x}] \wedge T[\bar{x}, \bar{x}'] \Rightarrow P[\bar{x}']$ (i.e. P satisfies the consecution under the assumption of P' [4]). This idea of adding formulas to our property will be used to determine if P' is M-invariant in the cases it's not inductive, by exploiting the following statement:

P inductive $\Rightarrow$ P M-invariant

Be aware that the opposite isn't true[7], except for the base case, meaning that **if initiation doesn't hold, then P isn't M-invariant**.

Why is this important? The idea is that given a safety property P, if we find other property P' stronger than P (i.e. $P' \models P$) and we find that P' is inductive on M relative to P, then P and P' hold in all reachable states of M thus we would have proven that $P \wedge P'$ is invariant [7].

2.1 Monolithic vs incremental methods

There are two main kind of processes aiming to prove the inductiveness of the property P. One is to prove the inductiveness by incrementally adding at each iteration a stronger formula based on the previous invariants. Such methods are called incremental proofs or incremental methods and usually run several small calls to SAT solvers[8]. On the other hand Monolithic approaches aim to get a strengthening formula to conjoin with P in a single step usually making just a single (but much more complex) call to a SAT solver[7]. We can think to the latter approach as a human stating a complex theorem and trying to prove it all at once, while the former can be seen as a human first developing small lemmas and building upon them construct a theorem. From now on we will just focus on incremental methods, a more in depth analysis is carried out in Paragraph 1.2 of [5].

2.2 The down algorithm, MIC and incremental induction

In Section 2 we introduced the concept of inductive clause relative to another, now we want a procedure that finds such clauses.

[6]We can think to Θ being a disjunction of conjunctions (cubes) each representing an initial state. So if we consider an initial state the value of the variables satisfy the disjunction and by the initiation implication P must be true.

[7]See example and Figure 3.

Recall the concept of subclause lattice for c and consider the monotonic function $down(\mathcal{L}_c, d)$ where d is an element of the partial order of $\mathcal{L}_c$[3] and $down(\mathcal{L}_c, d) := e$ where e is the largest subclause of d making $P \land e \land T[\bar{x}, \bar{x}'] \Rightarrow d'$ true. The clause e is then stronger (i.e has less dinjuctions) than d. To fix this concept consider the example of the system in Figure 2.

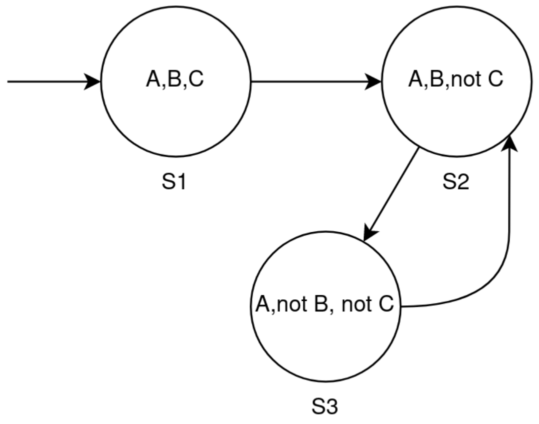

Figure 2: A an example of a system

Consider $d = B \lor C$, $P = A \lor B$, by looking at Figure 1 we can see that $down(\mathcal{L}_{A \lor B \lor C}, d)$ is C.

By Knaster-Tarski theorem we know $down(\mathcal{L}_c, d)$ is a complete lattice as well, so admits a greatest and a least fixpoint, consider $down(\mathcal{L}_c, c)$ if its greatest fixpoint, let's call it $\bar{c}$ satisfies the initiation ($\Theta \Rightarrow \bar{c}$) $\bar{c}$ is the **largest subclause of c inductive relative to** P[3]. We are interested on the other hand in finding the minimal inductive subclause of c (i.e. an inductive subclause with no further inductive subclauses), in some sense the best clause we can get in order to exclude the highest number of states possible. To obtain such clauses we use a procedure called $MIC(M, P, c)$ returning a minimal M-inductive (relative to P) subclause of c or $True$ if none exists (we will treat this function as a black box to focus on other main aspects, we just mention that exploits $down$)[3].
The question now is what clause c[8] do we provide MIC? The answer is negations of counter examples to induction (CTIs). Basically while computing the induction we will find (otherwise we are done) some P-sates that can lead to states which don't satisfy P. By providing such negations (remember states can be seen as cubes) we would like to find smaller formulas excluding those "bad" (and possibly other) states. By following the approach of [6] let's call X the strengthening formula relative to P generated (by adding a new conjunct) at the previous iteration, now we check if P is not M-inductive relative to X (otherwise we are again done), this can happen for two reasons:

- *initiation fails*: $\Theta \land (\neg P)$ is satisfied [9]. In this case P isn't M-invariant.
- *consecution fails*: $P \land X \land T[\bar{x}, \bar{x}'] \land (\neg P')$ is satisfied.

[8]M will our model so will be fixed and P will be the previous M-inductive subclause added, at the beginning is simply the property we want to check.

[9]Recall that $A \Rightarrow B \equiv \neg A \lor B \equiv \neg(A \land \neg B)$ thus if $A \land \neg B$ is true, it means $\neg(A \land \neg B)$ is false so the implication isn't true.

Now, in the second case a there's a state s falsifying consecution, s is a CTI and it's represented by a cube, with a bit of abuse of notation we refer to it by s as well. Notice that $\neg s$ is a clause and we can call $MIC(M, P \land X, \neg s)$, check whether returns $True$ and in such case we update our $P = P \land \neg s$[10] or returns $\bar{c}$ which excludes s (alongside other states[11]) and it's inductive relative to $P \land X$.

2.3 FSIS in a nutshell

The last part of Section 2.2 describes one iteration of the finite-state inductive strengthening algorithm $FSIS(M, P)$. Such procedure returns an inductive strengthening formula of P if P is invariant or returns a counter example trace for P (conjoined with other eventual subgoals) [3].
We now sum up in a more schematic way the procedure:

1. Let $P' := P, F := True$.
2. Check if P' satisfies initiation, if not, P' is not M-invariant.
3. Check if P' is inductive relative to F, if so we are done, P' (which could be stronger than $True$, see next step) is M-invariant.
4. Otherwise we strengthen P' with some inductive (relative to $P' \land F$) formula X coming from the counterexample trace (i.e. we would like to exclude the states that during the induction falsified P', X describes a superset of such states). To do so we call $MIC(M, P' \land F, \neg s)$. If no such clause X exists (i.e. MIC returns $True$) strengthen P' by conjoining $\neg s$ (where s is the cube of the state not satisfying P'). Otherwise $F = F \land X$ and we go back to step 2. Note that when the clause exists, this removes not only the "bad" states but also many unreachable states since it's the minimal M-inductive clause subclause of $\neg s$.

By observing the values assumed by F over the iterations, we have a list of formulas each stronger than its predecessor and at every iteration either we generate a formula X removing the states that are counterexamples or we add $\neg s$ to the target. So at each iteration we are removing some states "hoping" that P' may be inductive on such (smaller) set of new states. As stated in [6] we keep track of the states in which the formula P' isn't satisfied and augment such property by conjoining to it some negation of the states mentioned or a stronger formula. This can be done in a stepwise/incremental manner.
For example suppose we want to verify a property $P = A \lor C$ we have three variables in $\bar{x}$, A, B and C while T is described in Figure 3.

We can see that the red state in Figure 3 doesn't satisfy P but it's not reachable by $S1$ which is the initial state, so, P is M-invariant, let's now see how the algorithm behaves. We start by checking whether $A \land B \land C \Rightarrow A \lor C$, such formula is true, thus initiation holds, now check $A \lor C \land T[A, B, C, A', B', \neg C'] \Rightarrow A' \lor C'$, again it's holds, also at the next step $A \lor C \land T[A, B, \neg C, A', \neg B', C'] \Rightarrow A' \lor C'$ holds. The same is true for the last step, thus we can conclude that $S1, S2, S3$, are "safe". On the other hand we see that

[10]We move to the property to prove the fact that s can't be reached.
[11]MIC returns a small inductive subclause smaller than the largest inductive subclause of $\neg s$ which means that it's negation will remove more unreachable states from the space.

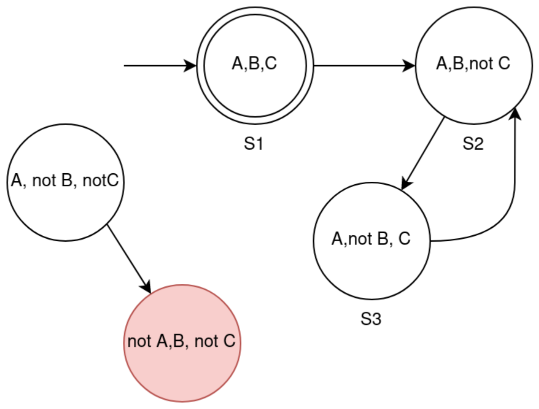

Figure 3: A an example of a system

$A \vee C \wedge T[A, \neg B, \neg C, \neg A', B', \neg C'] \not\Rightarrow A' \vee C'$ for the red state. So we run $MIC(M, A \vee C, \neg s = \neg(A \wedge \neg B \wedge \neg C))$ which should return C, so now we run the algorithm again with $F = True \wedge C = C$, and we find that induction holds.

2.3.1 A bit of complexity

The procedure $down(\mathcal{L}_c, c)$ described in 2.2 can be implemented by using $O(|c|)$ calls to a SAT solver. This is done by computing the fix point of the $LIC(\mathcal{L}_c, c)$ function described in [6]. Such function returns the largest subclause of c satisfying initiation and consecution. The main idea is the following:

- To check whether $P \wedge c \wedge T[\bar{x}, \bar{x}'] \Rightarrow c'$ holds we check whether $P \wedge c \wedge T[\bar{x}, \bar{x}'] \wedge \neg c'$ is unsatisfiable, if so we just check the initation condition for c. Otherwise exist an assignment $(\bar{x}, \bar{x}') = (s, s')$ satisfying it, we consider $\neg s$ and look for the largest clause sharing the most literals with it in $\mathcal{L}_c$, call it $\neg t$ (i.e. the best approximation). We then call $LIC(\mathcal{L}_c, c \sqcap \neg t)$.

The high level idea for the complexity is to notice that at each iteration we remove at least one literal.

2.4 Can we do better?

The strength of FSIS resides in the fact that we aren't just removing the conterexamples to P' at each iteration but we are excluding many more states. This though is done when the MIC procedure doesn't return $True$, otherwise we "remove" cube by cube the states falsifying P'. The naive approaches before FSIS (Exact SAT-based symbolic model checkers [8]) used only this technique. In unlucky cases FSIS can just behave like the naive algorithm, luckily we can do better with IC3.

3 IC3

IC3 (Incremental Construction of Inductive Clauses of Indubitable Correctness) is a SAT based inductive model checking algorithm used to verify safety properties.

The definition above will fit also FSIS, indeed IC3 and FSIS are not two complete different things. IC3 aims to improve FSIS in some of the cases in which takes a long time to search the next relatively inductive clauses. Let's now see the main concepts behind IC3 and how it works.

3.1 The idea

IC3 works by keeping a sequence of over-approximating sets F_i[12] (I,F_1,F_2...F_k) where each set represents a superset of the actual reachable states within i steps from the initial states [8, 9].

Suppose that we are at the $(i+1)th$ iteration step of the incremental procedure, we find ourselves having a sequence I, F_1...F_i of sets such that $\forall k \leq i \quad F_k \Rightarrow P$ meaning that all sets are subsets of P (otherwise we would have found in less than $i+1$ steps a reachable state falsifying P).

A property we want for the sets is that (for each k) $F_k(\bar{x}) \wedge T[\bar{x}, \bar{x}'] \Rightarrow F_{k+1}(\bar{x}')$ meaning

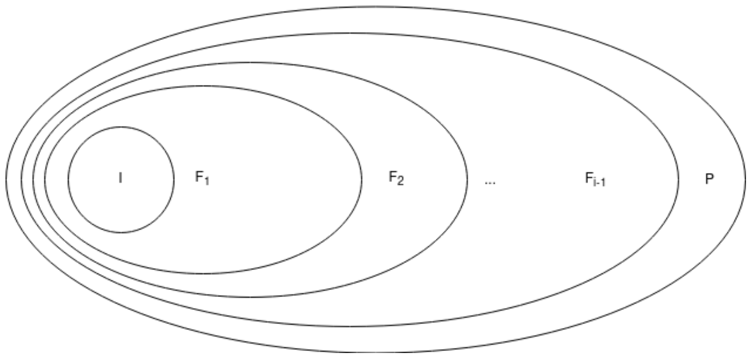

Figure 4: The overview of the sets if P is M-invariant for at least i-1 steps

that in F_{k+1} we would have at least all the states we can reach from F_k with an additional step[8]. As the first part of the $i+1$ iteration step we want to check whether in F_{i+1} we would have states falsifying P (i.e. we want to check $F_i \wedge T[\bar{x}, \bar{x}'] \Rightarrow P[\bar{x}']$). A series of scenarios are now possible:

Scenario 1. $F_i \wedge T[\bar{x}, \bar{x}'] \Rightarrow P[\bar{x}']$ holds, then we can directly refine the set F_{i+1} found (we set $F_{i+1} = P$) and go on (go to step 3).

Scenario 2. $F_i \wedge T[\bar{x}, \bar{x}'] \Rightarrow P[\bar{x}']$ doesn't hold, this means that from a state s in F_i we can reach a state satisfying $\neg P$, s is a CTI. An equivalent formulation is $SAT(F_i \wedge T[\bar{x}, \bar{x}'] \wedge \neg P[\bar{x}'])$ answering affirmatively, which shows[13] a CTI s proper of F_i[14]. This though doesn't mean that P isn't M-invariant since s could be one of the sates in F_i but that's not

[12]A formula.
[13]Shows through the values assigned to the variables.
[14]$s \in F_i$, $s \notin F_{i-1}$.

actually reachable in at most i steps (remember F_i is an over-approximation). For this reason incremental proof is used, meaning that we would like to prove a lemma stating the non reachability of s in i steps. To do this we look for a (minimal) subclase c of $\neg s$ inductive relative to F_i[15]. If we are able to find it we add it to all the $F_0..F_i$ [4] effectively disproving the reachability of the CTI in $i+1$ steps. The clause c may not exist though, the reason is that there may exist a predecessor of s, t in F_{i-1} preventing us from removing s in F_i. So the problem is now shifted towards the possible removal of t from F_{i-1}[16]. To do so we seek the (minimal) subclause of $\neg s$, namely c_1 which inductive relative to F_{i-1}. We conjoin c_1 to it to the frames and if it removes t from F_{i-1} we can continue to step 3. Otherwise the process is then repeated iteratively going up to all the frames looking for predecessor states to remove. Notice that as stated in [8] adding c_1 to the frames allows IC3 (if it didn't remove t) at the next iteration to focus only on the predecessors of s that matter in the frames.

Step 3. We refine all the sets, meaning that we loop through all $F_1..F_i$ checking whether any clause in the frame F_k considered is satisfies consecution relative to F_k, if so, we add such clause to F_{k+1} refining it (See propagate clauses [9]). At the end of this process we can start a new iteration over F_{i+2}.

Whenever we find a fixed point (i.e. $F_i = F_{i+1}$ for some i) the procedure stops answering affirmatively to P being M-inductive. The process of generating a new F_{i+j} and refining the sets is not infinite though. At each iteration we have that the current frame has at least one element more compared to the previous frame. Since the number of states of M is finite (even though it can be huge) this procedure always terminate either because we have a fixed point or returning "false" since a chain of states leads to a sate satisfying $\neg P$. The pseudocode of the idea described above can be found in [9].
A partial execution example of IC3 is now presented:

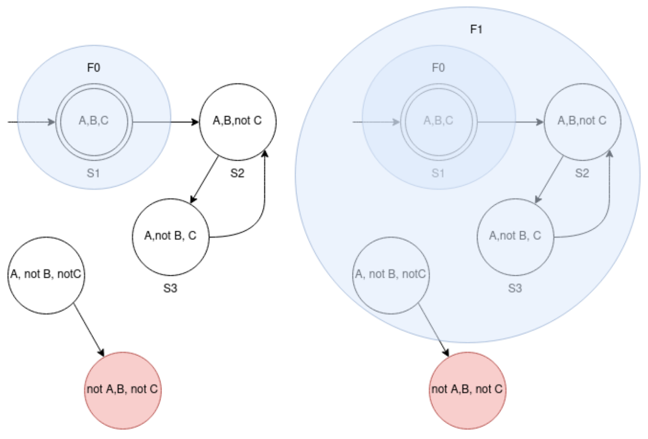

Figure 5: The initialization of $F_0 = I$ and since P holds in the successors, in the second step we initialize F1.

[15]$I \Rightarrow c$, $F_i(\bar{x}) \wedge c(\bar{x}) \wedge T[\bar{x}, \bar{x}'] \Rightarrow c(\bar{x}')$.

[16]i.e. to prove s in unreachable we must show t is, for all predecessors t of s [9]. We can find t by calling $SAT(F_i \wedge \neg s \wedge T \Rightarrow \neg s')$.

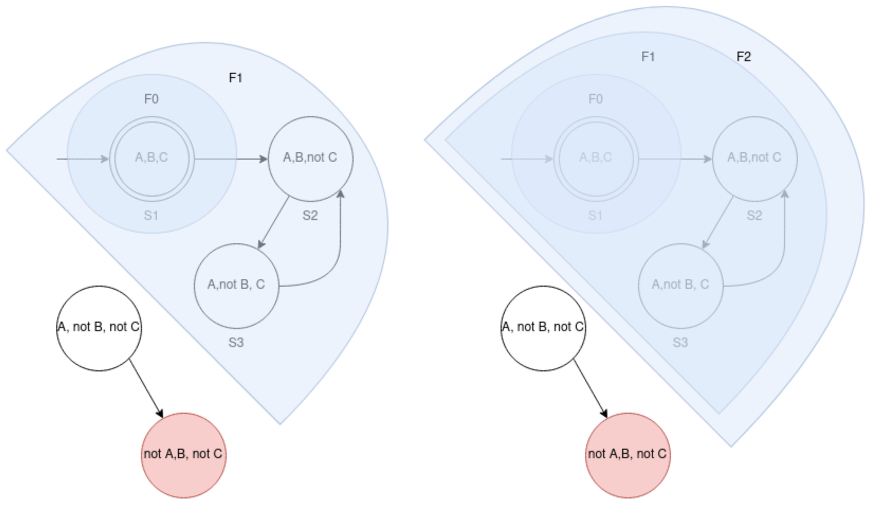

Figure 6: From F_1 we can reach a counterexample so we refine it, and at step four (and five) we introduce F_2 and we refine it. Fixpoint found.

4 Conclusions

Model checking is a fundamental formal verification technique used throughout industry to make software more reliable, comes in different flavours and as we have seen it keeps evolving and improving. In particular SAT based model checking has been positively affected by the evolution of SAT solvers (e.g. the improvements carried from the introduction of CDCL with respect to DPLL).
Although IC3 reaches noticeable performances, research is still going on and new inductive model checking algorithms such as FAIR and IICTL have been developed [8].

References

[1] C. Papadimitriou, *Computational Complexity*, ser. Theoretical computer science. Addison-Wesley, 1994. [Online]. Available: https://books.google.it/books?id=JogZAQAAIAAJ

[2] H. A. Priestley, *Ordered Sets and Complete Lattices.* Mathematical Institute, University of Oxford, ch. Chapter 2.

[3] A. Bradley and Z. Manna, "Checking safety by inductive generalization of counterexamples to induction," 12 2007, pp. 173–180.

[4] Z. Hassan, A. R. Bradley, and F. Somenzi, "Better generalization in ic3," in *2013 Formal Methods in Computer-Aided Design.* IEEE, 2013, pp. 157–164.

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

[6] A. R. Bradley, "Sat-based model checking without unrolling," in *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation*, ser. VMCAI'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 70–87.

[7] F. Franzini, "Model checking through inductive partitioning of the state space (http://hdl.handle.net/10589/166741)."

[8] A. R. Bradley, "Understanding ic3," in *International Conference on Theory and Applications of Satisfiability Testing.* Springer, 2012, pp. 1–14.

[9] S. Wolff, "Model checking with ic3."