A glimpse of IC3
(and SAT-based model checking)

Santi Enrico
santi.enrico@spes.uniud.it

University of Udine

Sept. 1st, 2023

Presentation overview

Preliminary concepts

Let’s start from afar

Lattice

A structure $\mathcal{L} = \langle L, \sqcap, \sqcup \rangle$ in which :

- L is a non empty partial ordered set.
- $\sqcap$ (meet) : L$\times L \longrightarrow L$, $x \vee y$ is the greatest lower bound to x and y.
- $\sqcup$ (join) : L$\times L \longrightarrow L$, $x \wedge y$ is the least upper bound of x and y.

Let's start from afar

Lattice

A structure $\mathcal{L} = \langle L, \sqcap, \sqcup \rangle$ in which :

- L is a non empty partial ordered set.
- $\sqcap$ (meet) : L$\times L \longrightarrow L$, $x \vee y$ is the greatest lower bound to x and y.
- $\sqcup$ (join) : L$\times L \longrightarrow L$, $x \wedge y$ is the least upper bound of x and y.

Subclause lattice

Given a clause c we can define it's the lattice $\mathcal{L}_c = \langle L, \sqcap, \sqcup \rangle$ where L is now the partial order induced by the subclauses of c with respect to the subclause relation.

Subclause lattice example

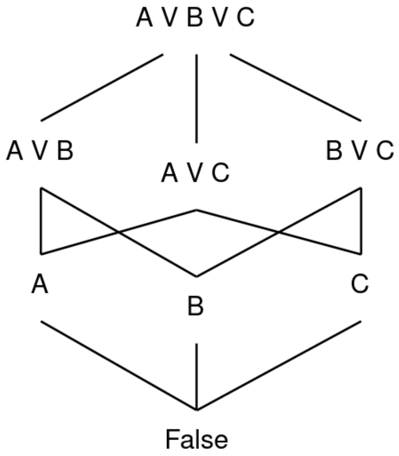


Figure – An example of the subclause lattice on the clause C = *A* ∨ *B* ∨ *C*.

SAT and SAT Solvers

A clause is a dsjunction of literals, a literal is a propositional variable or it's negation and a conjunction of clauses is a formula in CNF.

Example : $(A \vee B \vee C) \wedge (D \vee E) \wedge (C \vee D)$

Solving the CNF-SAT, means finding whether there exist an assignment for the variables satisfying the formula. **In general it's a computational expensive problem.**

SAT and SAT Solvers

A clause is a dsjunction of literals, a literal is a propositional variable or it's negation and a conjunction of clauses is a formula in CNF.

Example : $(A \vee B \vee C) \wedge (D \vee E) \wedge (C \vee D)$

Solving the CNF-SAT, means finding whether there exist an assignment for the variables satisfying the formula. **In general it's a computational expensive problem.**

Just remember that there are programs called **SAT-solvers** (black boxes for us) that can usually **sovle such problems fast** and that we will exploit them.

Symbolic transition systems

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 which explicitates the possible initial states and T is the finite set of transitions.

Symbolic then ?

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 which explicitates the possible initial states and T is the finite set of transitions.

Symbolic then ?

Symbolic transition system

A triple $M = \langle \bar{x}, \Theta, T\rangle$ where :

- $\bar{x}$: set of propositional (boolean) variables
- Θ : a propositional formula over $\bar{x}$
- T : a propositional formula over $(\bar{x}, \bar{x}')$ where $\bar{x}'$ represents the values of the variables in the successor state

The states now are the possible assignments over $\bar{x}$, seen as **cubes**.

BMC and safety

Bounded model checking is a technique for doing model checking consisting in (usually) unrolling the model for an incremental number of steps (BFS style) verifying that all the computations of increasing lengths satisfy the required property.

What properties ?

BMC and safety

Bounded model checking is a technique for doing model checking consisting in (usually) unrolling the model for an incremental number of steps (BFS style) verifying that all the computations of increasing lengths satisfy the required property.

What properties ?

$\downarrow$

Safety properties

$\downarrow$

Express that only “safe" states are reachable.

P it’s *M-invariant* if only states satisfying it are reachable during a computation

Induction and FSIS

Induction

Induction principle $\longrightarrow$ if the first falls, and so does the $(i+1)^{th}$ (assuming the i^{th} does), then everyone falls.

The same works for safety properties :

P is inductive if

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

Induction

Induction principle $\longrightarrow$ if the first falls, and so does the $(i+1)^{th}$ (assuming the i^{th} does), then everyone falls.

The same works for safety properties :

P is inductive if

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

P is inductive relative to P' if

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

What is this all for ?

The reason we care about induction is :

$$P \textbf{ inductive} \Rightarrow P \textbf{ M-invariant}$$

What is this all for ?

The reason we care about induction is :

$$P \textbf{ inductive} \Rightarrow P \textbf{ M-invariant}$$

Given a safety property P check if inductive, otherwise

$\downarrow$

Find other property P' to use as assumption at the next step.

$\downarrow$

if P' is inductive on M with respect to P, and P is inductive on M with respect to P', then $P \wedge P'$ is inductive.

What is this all for ?

The reason we care about induction is :

$$P \textbf{ inductive} \Rightarrow P \textbf{ M-invariant}$$

Given a safety property P check if inductive, otherwise

$\downarrow$

Find other property P' to use as assumption at the next step.

$\downarrow$

if P' is inductive on M with respect to P, and P is inductive on M with respect to P', then $P \wedge P'$ is inductive.

How do we find P' ? $\rightarrow$ **Monolitic** or **incremental** way

Monolithic vs incremental

Two main kind of processes aiming to prove the inductiveness of P :

- Prove the inductiveness by incrementally adding at each iteration a stronger formula based on the previous invariants. **Incremental proofs or incremental methods**.
- Aim to get a strengthening formula to conjoin with P in a single step usually making just a single (much more complex) call to a SAT solver. **Monolithic methods**.

Monolithic vs incremental

Two main kind of processes aiming to prove the inductiveness of P :

- Prove the inductiveness by incrementally adding at each iteration a stronger formula based on the previous invariants. **Incremental proofs or incremental methods**.
- Aim to get a strengthening formula to conjoin with P in a single step usually making just a single (much more complex) call to a SAT solver. **Monolithic methods**.

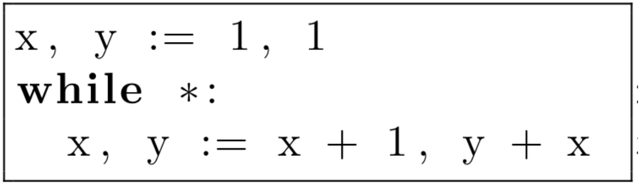
Incremental methods and usually run several small calls to SAT solvers.

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.

An example

Consider the example from [3].

Given the program M :



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

We want to establish whether $y >= 1$ is M-invariat.

The incremental way

We wouldn't be here if P was inductive, so we have to strengthen it.

The incremental way

We wouldn't be here if P was inductive, so we have to strengthen it.

Let $\mathcal{L}_c$ be the subclause lattice for c, consider the monotonic function down$(\mathcal{L}_c, d) :=$ e where d is an element of the partial order of $\mathcal{L}_c$ and e is the largest subclause of d making $P \wedge e \wedge T[\bar{x}, \bar{x}'] \Rightarrow d'$ true.

Knaster-Tarski theorem

$\downarrow$

Down$(\mathcal{L}_c, d)$ is a complete lattice as well, so admits a greatest and a least fixpoint

The incremental way

We wouldn't be here if P was inductive, so we have to strengthen it.

Let $\mathcal{L}_c$ be the subclause lattice for c, consider the monotonic function $down(\mathcal{L}_c, d) := e$ where d is an element of the partial order of $\mathcal{L}_c$ and e is the largest subclause of d making $P \land e \land T[\bar{x}, \bar{x}'] \Rightarrow d'$ true.

Knaster-Tarski theorem

$\downarrow$

$Down(\mathcal{L}_c, d)$ is a complete lattice as well, so admits a greatest and a least fixpoint

$\downarrow$

If the greatest fix point ($\bar{c}$) of $down(\mathcal{L}_c, c)$ satisfies the initiation, it's the largest subclause of c inductive relative to P (**recall $\bar{c}$ is stronger than c**)

The incremental way (cont.) - down algorithm

The greatest fixpoint of down$(\mathcal{L}_c, c)$ can be found by using $O(|c|)$ calls to a SAT solver.

This is done by computing the fix point of the $LIC(\mathcal{L}_c, d)$ function which returns the largest subclause of d satisfying initiation and consecution.

The main idea :

To Check whether $P \wedge d \wedge T[\bar{x}, \bar{x}'] \Rightarrow d'$ holds we check whether $P \wedge d \wedge T[\bar{x}, \bar{x}'] \wedge \neg d'$ is unsatisfiable, if so we also check the initation condition for d. Otherwise there exist an assignment $(\bar{x}, \bar{x}') = (s, s')$ satisfying it, 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 $LIC(\mathcal{L}_c, d \sqcap \neg t)$.

The incremental way (cont.) - down algorithm

The greatest fixpoint of down$(\mathcal{L}_c, c)$ can be found by using $O(|c|)$ calls to a SAT solver.

This is done by computing the fix point of the $LIC(\mathcal{L}_c, d)$ function which returns the largest subclause of d satisfying initiation and consecution.

The main idea :

To Check whether $P \wedge d \wedge T[\bar{x}, \bar{x}'] \Rightarrow d'$ holds we check whether $P \wedge d \wedge T[\bar{x}, \bar{x}'] \wedge \neg d'$ is unsatisfiable, if so we also check the initation condition for d. Otherwise there exist an assignment $(\bar{x}, \bar{x}') = (s, s')$ satisfying it, 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 $LIC(\mathcal{L}_c, d \sqcap \neg t)$.

At each iteration we remove at least one literal since t (conjunction) and d (disjunction) can't have all literals different (e.g. not possible $t = a \wedge b$ $d = \neg a \vee \neg b$).

The incremental way (cont.) - mic algorithm

We want the minimal inductive subclause of d (an inductive subclause with no further inductive subclauses).
We use a procedure called $MIC(M, P, d)$ returning a minimal M-inductive (relative to P) subclause of d or *True* if none exists (we will treat is as a black box to focus on other main aspects).

P at the beginning is the property we want to prove, but what clause d do we use ?

$\downarrow$

Negations of counter examples to induction (CTIs). P-sates that can lead to states which don't satisfy P.

$\downarrow$

So we would like to find smaller formulas excluding those "bad" (and possibly other) states.

FSIS in a nutshell

Having introduced all these procedures, we sum up FSIS (finite state inductive strengthening).

FSIS(M,P) :

FSIS in a nutshell

Having introduced all these procedures, we sum up FSIS (finite state inductive strengthening).

FSIS(M,P) :

1. Let $P' := P, F := True$.

FSIS in a nutshell

Having introduced all these procedures, we sum up FSIS (finite state inductive strengthening).

FSIS(M,P) :

1. Let $P' := P, F := \mathit{True}$.
2. Check if P' satisfies initiation, if not, P' is not M-invariant.

FSIS in a nutshell

Having introduced all these procedures, we sum up FSIS (finite state inductive strengthening).

FSIS(M,P) :

1. Let $P' := P, F := \mathit{True}$.
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 P, see next step) is M-invariant.

FSIS in a nutshell

Having introduced all these procedures, we sum up FSIS (finite state inductive strengthening).

FSIS(M,P) :

1. Let $P' := P, F := \mathit{True}$.
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 P, see next step) is M-invariant.
4. Otherwise we strengthen P' with some inductive (relative to $P' \wedge F$) 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 $MIC(M, P' \wedge F, \neg s)$. 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 $F = F \wedge X$ 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

FSIS example

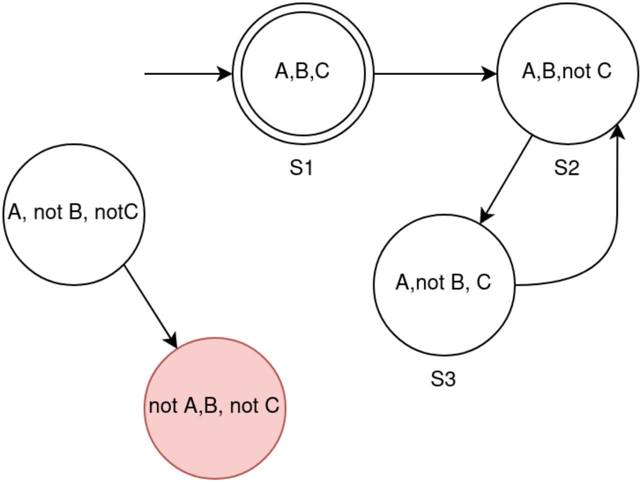


Figure – A an example of a system

FSIS example (cont.)

Suppose we want to verify $P = A \vee C$ and we have three variables in $\bar{x}$, A, B and C.

- Start by checking whether $A \wedge B \wedge C \Rightarrow A \vee C$, such formula is true, thus initiation holds, now check $A \vee C \wedge T[A, B, C, A', B', \neg C'] \Rightarrow A' \vee C'$, again it's holds, also at the next step $A \vee C \wedge T[A, B, \neg C, A', \neg B', C'] \Rightarrow A' \vee C'$ holds.The same is true for the last step, thus we can conclude that $S1$, $S2$, $S3$, are safe.

FSIS example (cont.)

Suppose we want to verify $P = A \vee C$ and we have three variables in $\bar{x}$, A, B and C.

- Start by checking whether $A \wedge B \wedge C \Rightarrow A \vee C$, such formula is true, thus initiation holds, now check $A \vee C \wedge T[A, B, C, A', B', \neg C'] \Rightarrow A' \vee C'$, again it's holds, also at the next step $A \vee C \wedge T[A, B, \neg C, A', \neg B', C'] \Rightarrow A' \vee C'$ holds.The same is true for the last step, thus we can conclude that $S1$, $S2$, $S3$, are safe.
- But $A \vee C \wedge T[A, \neg B, \neg C, \neg A', B', \neg C'] \nRightarrow A \vee C$ 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.

IC3 in a nutshell

Introduction to IC3

FSIS seems to work fine, why would we need another algorithm ?

In FSIS we aren't just removing the counterexamples to P' at each iteration but we are excluding many more states, this is its strength with respect to the naive algorithm.

Introduction to IC3

FSIS seems to work fine, why would we need another algorithm ?

In FSIS we aren't just removing the counterexamples to P' at each iteration but we are excluding many more states, this is its strength with respect to the naive algorithm.

$\downarrow$

This happens 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) used only this technique. In unlucky cases FSIS can just behave like the naive algorithm.

IC3 in a nutshell

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

IC3 works by keeping a sequence of over-approximating sets F_{i_s} (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.

IC3 in a nutshell

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

IC3 works by keeping a sequence of over-approximating sets F_{i_s} (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.

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

IC3 in a nutshell (cont.)

We begin the iteration step by setting $F_{i+1} = P$ after checking that $F_i \wedge T[\bar{x}, \bar{x}'] \Rightarrow P[\bar{x}']$ holds.

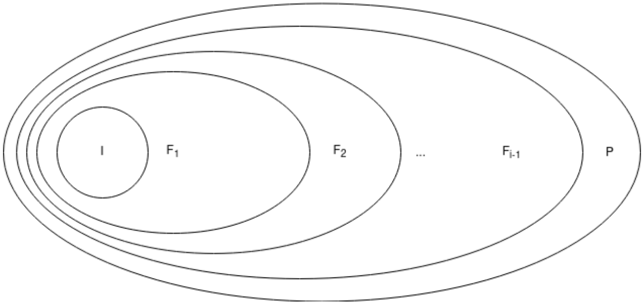


Figure – The overview of the sets if P is M-invariant for at least i-1 steps

Two cases $F_i \wedge T[\bar{x}, \bar{x}'] \Rightarrow P[\bar{x}']$ holds or not.

IC3 in a nutshell (cont.) - the property holds

Since the previous property holds, we can directly refine the frames found and then start a new iteration. To refine all the sets, we loop through all $F_1..F_i$ checking whether any clause in the frame F_k considered it satisfies consecution relative to $F_{1..k}$, if so, we add such clause to F_{k+1} refining it.

IC3 in a nutshell (cont.) - the property holds

Since the previous property holds, we can directly refine the frames found and then start a new iteration. To refine all the sets, we loop through all $F_1..F_i$ checking whether any clause in the frame F_k considered it satisfies consecution relative to $F_{1..k}$, if so, we add such clause to F_{k+1} refining it.

$\downarrow$

This procedure is called clause propagation and allows to keep under control the size of the sets while i grows

$\downarrow$

At the end of this process we can start a new iteration over F_{i+2}.

IC3 in a nutshell (cont.) - the property doesn't hold

If the property doesn't hold, it 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.

$\downarrow$

This though doesn't mean that P isn't M-invariant, s could be one of the sates in F_i but that's not actually reachable in at most i steps. Incremental proof is then used to (try) prove a lemma stating the non reachability of s in i steps.

IC3 in a nutshell (cont.) - the property doesn't hold

If the property doesn't hold, it 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.

$\downarrow$

This though doesn't mean that P isn't M-invariant, s could be one of the sates in F_i but that's not actually reachable in at most i steps. Incremental proof is then used to (try) prove a lemma stating the non reachability of s in i steps.

$\downarrow$

We look for a (minimal) subclause c of $\neg s$ inductive relative to F_i. If we find it we add it to all the $F_0..F_i$ effectively disproving the reachability of the CTI in i steps.

$\downarrow$

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. The problem is now shifted towards the possible removal of t from F_{i-1}.

$\downarrow$

IC3 in a nutshell (cont.) - the property doesn't hold (cont.)

To do so we seek the (minimal) subclause c_1 of $\neg s$ inductive relative to F_{i-1}. We conjoin c_1 to the frames and if it removes t from F_{i-1} we can do clause propagation and begin a new iteration.

$\downarrow$

Otherwise the process is then repeated iteratively going up to all the frames looking for predecessor states to remove. **As stated in [3] 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.**

IC3 in a nutshell (cont.) - how do we stop

Does IC3 always terminate ?

IC3 in a nutshell (cont.) - how do we stop

Does IC3 always terminate ? Yes.

IC3 in a nutshell (cont.) - how do we stop

Does IC3 always terminate ? Yes.

$\downarrow$

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 fixpoint (returning “true”) or returning “false” since a chain of states leads to a sate satisfying $\neg P$.

IC3 example

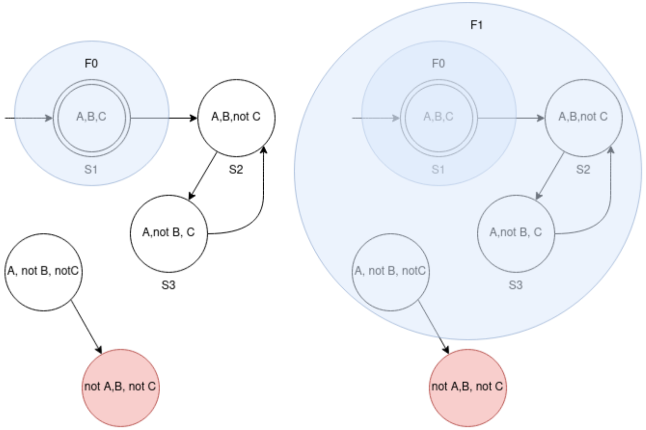
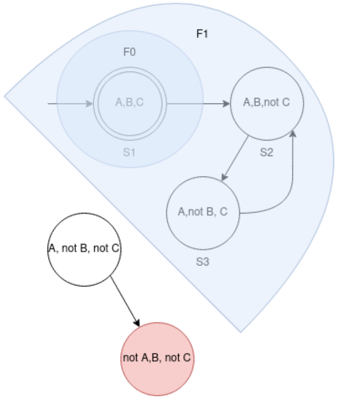


Figure – The initialization of F0 = I and since P holds in the successors, in the second step we initialize F1

IC3 example (cont.)



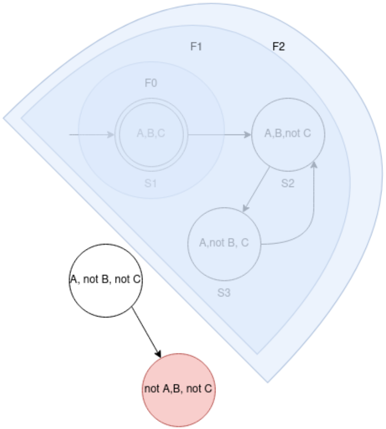


Figure – From F1 we can reach a counterexample so we refine it, and at step four (and five) we introduce F2 and we refine it. Fixpoint found

Conclusions

Conclusions - it’s not over yet

Model checking

$\downarrow$

Fundamental formal verification technique used throughout industry to make software more reliable.

Conclusions - it's not over yet

Model checking

↓

Fundamental formal verification technique used throughout industry to make software more reliable.

↓

Comes in different flavours, and benefits from many different improvements, **SAT based model checking has been positively affected by the evolution of SAT solvers.**

↓

Research is still going on and new inductive model checking algorithms such as FAIR and IICTL have been developed.

References I

Aaron Bradley and Zohar Manna.
Checking safety by inductive generalization of counterexamples to induction.
pages 173–180, 12 2007.

Aaron R. Bradley.
Sat-based model checking without unrolling.
In *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'11, page 70–87, Berlin, Heidelberg, 2011. Springer-Verlag.

Aaron R Bradley.
Understanding ic3.
In *International Conference on Theory and Applications of Satisfiability Testing*, pages 1–14. Springer, 2012.

Francesco Franzini.
Model checking through inductive partitioning of the state space (http ://hdl.handle.net/10589/166741).

Zyad Hassan, Aaron R Bradley, and Fabio Somenzi.
Better generalization in ic3.
In *2013 Formal Methods in Computer-Aided Design*, pages 157–164. IEEE, 2013.

Zohar Manna and Amir Pnueli.
Temporal Verification of Reactive Systems : Safety.
Springer-Verlag, Berlin, Heidelberg, 1995.

C.H. Papadimitriou.
Computational Complexity.
Theoretical computer science. Addison-Wesley, 1994.

References II

Hilary A. Priestley.
Ordered Sets and Complete Lattices, chapter Chapter 2.
Mathematical Institute, University of Oxford.

Sebastian Wolff.
Model checking with ic3.

Fin.