SAT-Based Model Checking: Interpolation, IC3 and Beyond

Orna Grumberg
Technion, Israel
Based on presentations by Yakir Vizel

Marktoberdorf 2013

outlines

• Model checking
• SAT-based bug finding
  - Bounded Model Checking (BMC)
• SAT-based verification with
  - Interpolation
  - Interpolation-sequence
  - IC3
  - IC3 + lazy abstraction
  - Forward and backward interpolation
Why (formal) verification?

- safety-critical applications: Bugs are unacceptable!
  - Air-traffic controllers
  - Medical equipment
  - Cars
- Bugs found in later stages of design are expensive
- Hardware and software systems grow in size and complexity: Subtle errors are hard to find by testing
- Pressure to reduce time-to-market

Automated tools for formal verification are needed

---

Formal Verification

Given
- a model of a (hardware or software) system and
- a formal specification

does the system model satisfy the specification? Not decidable!

To enable automation, we restrict the problem to a decidable one:
- Finite-state reactive systems
- Propositional temporal logics
Finite state systems - examples

- Hardware designs
- Controllers (elevator, traffic-light)
- Communication protocols (when ignoring the message content)
- High level (abstracted) description of non finite state systems

Properties in temporal logic - examples

- mutual exclusion:
  always $\neg (cs_1 \land cs_2)$

- non starvation:
  always (request $\Rightarrow$ eventually granted)

- communication protocols:
  $\neg$ get-message until send-message
**Model Checking** \([CE81,QS82]\)

An efficient procedure that receives:
- A finite-state model describing a system
- A temporal logic formula describing a property

It returns
- yes, if the system has the property
- no + Counterexample, otherwise

Model of a system

Kripke structure / transition system
Temporal Logics

- **Temporal Logics**
  - Express properties of event orderings in time

- **Linear Time**
  - Every moment has a unique successor
  - Infinite sequences (words)
  - Linear Time Temporal Logic (LTL)

- **Branching Time**
  - Every moment has several successors
  - Infinite tree
  - Computation Tree Logic (CTL)

Propositional temporal logic

**AP** - a set of atomic propositions

**Temporal operators:**

- **Gp**
- **Fp**
- **Xp**
- **pUq**

**Path quantifiers:**

- **A** for all path
- **E** there exists a path
Model checking $AG \ p$ on $M$

- Iteratively compute the sets $S_j$ of states reachable from an initial state in $j$ steps
- At each iteration check whether $S_j$ contains a state satisfying $\neg p$.
  - If so, declare a failure
- Terminate when all states were found. $S_k \subseteq \bigcup_{i=0}^{k-1} S_i$
  - Result: the set $Reach$ of reachable states.

Model checking $AG \ p$

- Also called forward reachability analysis
**Mutual Exclusion Example**

- Two process mutual exclusion with shared semaphore
- Each process has three states
  - Non-critical (N)
  - Trying (T)
  - Critical (C)
- Semaphore can be available ($S_0$) or taken ($S_1$)
- Initially both processes are in the Non-critical state and the semaphore is available --- $N_1 N_2 S_0$

\[
\begin{align*}
N_1 & \rightarrow T_1 \\
T_1 \land S_0 & \rightarrow C_1 \land S_1 \\
C_1 & \rightarrow N_1 \land S_0 \\
N_2 & \rightarrow T_2 \\
T_2 \land S_0 & \rightarrow C_2 \land S_1 \\
C_2 & \rightarrow N_2 \land S_0
\end{align*}
\]

\[M \models AG \neg(C_1 \land C_2)\]

*The two processes are never in their critical states at the same time*
Mutual Exclusion Example

\[ M \models AG \neg (C_1 \land C_2) \]

\[ S_0 \]

Mutual Exclusion Example

\[ M \models AG \neg (C_1 \land C_2) \]

\[ S_1 \]
Mutual Exclusion Example

\[ M \models AG \rightarrow (C1 \land C2) \]

\[ S_2 \]

Mutual Exclusion Example

\[ M \models AG \rightarrow (C1 \land C2) \]

\[ S_3 \]
**Mutual Exclusion Example**

\[ M \vDash AG \neg (C_1 \land C_2) \]

\[ S_4 \subseteq S_0 \cup \ldots \cup S_3 \]

---

**Forward Reachability Analysis**

- terminates when
  - either a bad state satisfying \( \neg p \) is found
  - or a fixpoint is reached: \( S_j \subseteq \bigcup_{i=0,j-1} S_i \)
Main limitation

The state explosion problem:

Space and time requirements grow with the size of the model

SAT-based model checking:
A solution for the state explosion problem

Main idea

• Translate the model and the specification to propositional formulas

• Use efficient tools (SAT solvers) for solving the satisfiability problem
Bounded Model Checking (BMC) for checking AGp

- Unwind the model for k levels, i.e., construct all computations of length k

- If a state satisfying \( \neg p \) is encountered, produce a counterexample; Otherwise, increase k

[BCCZ 99]
Bounded Model Checking

Terminates
• with a counterexample or
• with time- or memory-out

The method is suitable for falsification, not verification

Example - shift register

Shift register of 3 bits: \( <x, y, z> \)

Transition relation:
\[ R(x,y,z,x',y',z') = \ x' = y \land y' = z \land z' = 1 \]

Initial condition:
\[ I(x,y,z) = \ x = 0 \lor y = 0 \lor z = 0 \]

Specification: \( AG \ ( x = 0 \lor y = 0 \lor z = 0 ) \)
Propositional formula for $k=2$

$$f_M = (x_0=0 \lor y_0=0 \lor z_0=0) \land$$

$$\quad (x_1=y_0 \land y_1=z_0 \land z_1=1) \land$$

$$\quad (x_2=y_1 \land y_2=z_1 \land z_2=1)$$

$$f_\varphi = V_{i=0,..,2} (x_i=1 \land y_i=1 \land z_i=1)$$

Satisfying assignment: 101 011 111
This is a counter example!

Bounded model checking

- Can be used for verification by choosing $k$ which is large enough so that every path of length $k$ contains a cycle

- Using such a $k$ is often not practical due to the size of the model
Verification with SAT solvers

Two successful methods for SAT-based verification are based on:
• Interpolation [McMillan 03]
• IC3 [Bradley 11]

In this series of talks
we present methods for enhancing interpolation- and IC3-based model checking
Interpolation-Sequence Based Model Checking [Vizel, Grumberg 09]

Inspired by:
- forward reachability analysis

Combines:
- Bounded Model Checking
- Interpolation-sequence [Jhala, McMillan 05]

Obtains:
- SAT-based model checking algorithm for full verification

Interpolation [Craig 57]

- If $A \land B = \text{false}$, there exists an interpolant $I$ for $(A,B)$ such that:

  $$A \Rightarrow I$$
  $$I \land B = \text{false}$$
  $I$ refers only to common variables of $A,B$
Interpolation (cont.)

Interpolants from proofs

• When \( A \land B \) is unsatisfiable, SAT solvers return a proof of unsatisfiability in the form of a resolution graph

• Given a resolution graph, \( I \) can be derived in linear time.

\[ \text{[Pudlak,Krajicek 97]} \]

Interpolation in the context of model checking

• Given the following BMC formula \( \varphi^k \)

\[
\begin{align*}
\text{A} & \quad \text{B} \\
INIT(V_0) \land T(V_0,V_1) \land T(V_1,V_2) \land \ldots \land T(V_{k-1},V_k) \land \neg p(V_k) \\
\downarrow \\
I \\
A \Rightarrow I \\
I \land B \equiv \text{false}
\end{align*}
\]

I is over the common variables of A and B, i.e. \( V_1 \)
Interpolation in the context of model checking

- $I$ is over $V_1$
- $A \Rightarrow I$
  - $I$ over-approximates the set $S_1$
- $I \land B \equiv \text{false}$
  - States in $I$ cannot reach a bug in $k-1$ steps

Interpolation-Sequence

- The same BMC formula partitioned in a different manner:

\[
\begin{align*}
&INIT(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \ldots \land T(V_{k-1}, V_k) \land \neg p(V_k) \\
&I_1 \quad I_2 \quad I_3 \quad I_{k-1} \quad I_k
\end{align*}
\]

$I_0 = \text{true}, I_{k+1} = \text{false}$

$I_{j-1} \land A_j \Rightarrow I_j$

$I_j$ is over the common variables of $A_1, \ldots, A_j$ and $A_{j+1}, \ldots, A_{k+1}$, i.e. $V_j$
Interpolation-Sequence

• $I_j$ - over-approximation of the set of states reachable in $j$ steps

• $I_k \land A_{k+1} \Rightarrow \text{false}$
  the states in $I_k$ do not violate $p$

Interpolation-Sequence

• Can easily be computed. For $1 \leq j < n$
  - $A = A_1 \land \ldots \land A_j$
  - $B = A_{j+1} \land \ldots \land A_n$
  - $I_j$ is the interpolant for the pair $(A,B)$
Combining Interpolation-Sequence and BMC

- Uses BMC for bug finding

- Uses Interpolation-sequence for computing over-approximation of sets $S_j$ of reachable states

Always terminates
- either when BMC finds a bug:
  $M \not\models AG\beta$

- or when all reachable states has been found:
  $M \models AG\beta$
Using Interpolation-Sequence

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land \neg p(V_1) \]

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land \neg p(V_2) \]

Checking if a “fixpoint” has been reached

- \[ I_j \Rightarrow V_{k=1,j-1} I_k \]

- Similar to checking fixpoint in forward reachability analysis:
  \[ S_j \subseteq U_{k=1,j-1} S_k \]

- But here we check inclusion for every \( 2 \leq j \leq N \)
  - No monotonicity because of the approximation

- “Fixpoint” is checked with a SAT solver
The Analogy to Forward Reachability Analysis

INIT $S_1$ $S_2$ $S_3$

$\text{BAD } \neg p$

Notation:
If no counterexample of length $N$ or less exists in $M$, then:

- $I_j^k$ is the $j$-th element in the interpolation-sequence extracted from the BMC-partition of $\varphi^k$

- $I_j = \land_{k=j,N} I_j^k \ [V_j \leftarrow V]$

- The reachability vector is: $\hat{I} = (I_1, I_2, ..., I_N)$
function FixpointReached (\( \hat{I} \)) // check \( I_j \Rightarrow V_{k=1,j+1} I_k \)
    j=2
    while (\( j \leq \hat{I}.\text{length} \)) do
        R = \( V_{k=1,j-1} I_k \)
        \( \alpha = I_j \land \neg R \) // negation of \( I_j \Rightarrow R \)
        if (SAT(\( \alpha \))==false) then return true
        end if
        j = j+1
    end while
    return false
end function

Interpolation-Based Model Checking [McM03]
Interpolation In The Context of Model Checking

- We can check several bounds with one formula.
- Given a BMC formula with possibly several bad states.

\[
\begin{align*}
I & \equiv \Box (\phi) \\
\Box (\phi) & \equiv (\phi_0) \land (\phi_1) \\
\phi_0 & \equiv \text{INIT}(V_0) \land \neg (V_1) \\
\phi_1 & \equiv \text{T}(V_1, V_2) \land \text{T}(V_2, V_3) \land \ldots \land \text{T}(V_{k-1}, V_k) \land (\neg q(V_1) \lor \ldots \lor \neg q(V_k))
\end{align*}
\]

\[ A \Rightarrow I \]
\[ I \land B \equiv F \]

I is over the common variables of A and B, i.e. V_i

Interpolation In The Context of Model Checking

- The interpolant represents an over-approximation of reachable states after one transition.
- Also, there is no path of length \( k-1 \) or less that can reach a bad state.
Using Interpolation

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]

\[ I_1(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]

\[ I_2(V_0) \land T(V_0, V_1) \land \neg q(V_1) \]

Using Interpolation

\[ \text{INIT}(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land (\neg q(V_1) \lor \neg q(V_2)) \]

\[ I_1'(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land (\neg q(V_1) \lor \neg q(V_2)) \]

\[ \vdots \]

\[ \vdots \]

\[ I_k'(V_0) \land T(V_0, V_1) \land T(V_1, V_2) \land (\neg q(V_1) \lor \neg q(V_2)) \]
The Analogy to Forward Reachability Analysis

\[ \text{INIT} \quad S_1 \quad S_2 \quad S_3 \]

\[ \neg q \]

If BMC finds a satisfying assignment the counterexample might be spurious
- The set of initial states is over-approximated

Increase k and start with the original INIT
Characteristics

- When calculating the interpolant for the $i$-th iteration, for bound $k$ the following holds:
  - The interpolant represents an over-approximation of reachable states after $i$ transitions
  - Also, it cannot reach a bad state in $k-I+i$ steps or less
    - It is similar to $I_i$ calculated in ISB after $k+i$ iterations

Comparison to interpolation-sequence based model checking

- The computation itself is different
  - Uses interpolation, not interpolation sequence
  - Based on nested loops
  - Not incremental
- The computed over-approximated sets are different.
Experimental Results

Compared interpolation based and interpolation-sequence based model checking
• Experiments were conducted on two (then) future CPU designs from Intel (two different architectures)
• Real-life properties were checked on each

Comparison Analysis

• Results vary
• Some properties cannot be verified by one method but can be verified by the other and vise-versa
Summary

• A new SAT-based method for **unbounded** model checking.
  - BMC is used for falsification.
  - Simulating forward reachability analysis for verification.

• Method was successfully applied to industrial sized systems.
**Incremental Construction of Inductive Clauses for Indubitable Correctness**

or simply: IC3

[Bradley, VMCAI’11]

A Simplified Description

Recall: Following reachability analysis
Notations

• System is modeled as \((V, I, T)\), where:
  - \(V\) is a finite set of variables
  - \(I \subseteq 2^V\) is the set of initial states
  - \(T \subseteq 2^V \times 2^V\) is the set of transitions

• A safety property of the form \(AG P\)
  - \(P\) is a propositional formula over \(V\)

Induction for proving \(AG P\)

• The simple case: \(P\) is an inductive invariant
  - \(I \Rightarrow P\)
  - \(P \land T \Rightarrow P'\)

• \(P'\) – the value of \(P\) in the next state

• \(I(V) \Rightarrow P(V)\)
• \(P(V) \land T(V, V') \Rightarrow P(V')\)
Induction for proving $AG\; P$

- Usually, $P$ is not an inductive invariant
- BUT – a stronger inductive invariant $R$ may exist (strengthening)
  - $I \Rightarrow R$
  - $R \land T \Rightarrow R'$
  - $R \Rightarrow P$
- $R$ can be computed in various ways (BDDs, k-induction, Interpolation-Sequence,…)

Inductive invariant
IC3

• The Goal: Find an Inductive Invariant stronger than P by learning relatively inductive facts (incrementally)

  - Recall: F is inductive invariant if
    - \( I \Rightarrow F \)
    - \( F \land T \Rightarrow F' \)
  - If F is stronger than P, i.e., \( F \Rightarrow P \), then
    - \( F \land P \land T \Rightarrow F' \Rightarrow P' \)

What Makes IC3 Special?

• No unrolling of the transition relation \( T \) is required

• All previous approaches require unrolling
  - Searching for an inductive invariant
  - Unrolling = A form of strengthening

• IC3 strengthen in a different way
  - Learning relatively inductive facts locally
IC3 Basics

• Iteratively compute Over-approximated Reachability Sequence (OARS) \( \langle F_0, F_1, \ldots, F_k \rangle \) s.t.
  - \( F_0 = INIT \)
  - \( F_i \models p \) : p is an invariant up to k
  - \( F_i \models F_{i+1} \) : \( F_i \subseteq F_{i+1} \)
  - \( F_i \land T \models F'_{i+1} \) : Simulates one forward step

\( F_i \) - over-approximates the set of states reachable within i steps

• If \( F_{i+1} \Rightarrow F_i \) then fixpoint

IC3 Basics

• P is inductive relative to F if
  - \( I \models P \)
  - \( F \land P \land T \models P' \)

• Notations:
  - Cube \( s \): conjunction of literals
    - \( v_1 \land v_2 \land \neg v_3 \) - Represents a state
  - \( s \) is a cube \( \Rightarrow \neg s \) is a clause (DeMorgan)
A Backward Search

• Search for a predecessor \( s \) to some error state: \( P \land T \land \neg P' \)
  - If none exists, property \( P \) holds:
    • \( (P \land T \land \neg P') \) unsat  IFF  \( (P \land T \Rightarrow P') \) valid

• Otherwise, try to block \( s \)
  - \( P = P \land \neg s \)
  - BUT, first need to show the \( s \) is not reachable
IC3 - Initialization

• Check satisfiability of the two formulas:
  - $I \land \neg P$
  - $I \land T \land \neg P'$

• If both are unsatisfiable then:
  - $I \Rightarrow P$
  - $I \land T \Rightarrow P'$

• Therefore
  - $F_0 = I, F_1 = P$
  - $\langle F_0, F_1 \rangle$ is OARS
IC3 - Iteration

- Our OARS contains $F_0$ and $F_1$
  - If $P$ is an inductive invariant - done! 😊
  - Otherwise:
    - $F_1$ should be strengthened

IC3 - Iteration

- $P$ is not an inductive invariant
  - $F_1 \land T \land \neg P'$ is satisfiable
  - From the satisfying assignment get the state $s$ that can reach the bad states

\[ F_0 \quad I \quad P \quad F_1 \]

\[ I \quad F_0 \quad P \quad F_1 \quad s \]
IC3 - Iteration

• Is $s$ reachable or not?
  - Hard to know
  - If it is reachable a CEX exists
    • Why?

\[
\begin{align*}
F_0 & \quad p \quad F_1 \\
\text{I} & \quad s
\end{align*}
\]

IC3 - Iteration

• Is $s$ reachable in one transition from the previous set? (Bounded reachability)
  - Check $F_0 \land T \land s'$
  - If satisfiable, $s$ is reachable from $F_0$ (CEX)
  - Otherwise, block it = remove it from $F_1$
    • $F_1 = F_1 \land \neg s$

\[
\begin{align*}
F_0 & \quad p \quad F_1 \\
\text{I} & \quad s
\end{align*}
\]
IC3 - Iteration

- Iterate this process until $F_1 \land T \land \neg P'$ becomes unsatisfiable
  - $F_1 \land T \Rightarrow P'$ holds
  - $F_2$ can be defined to be $P$
    - Any problems/issues with that?

IC3 - Iteration

- New iteration, check $F_2 \land T \land \neg P'$
  - If satisfiable, get $s$ that can reach $\neg P$
  - Now check if $s$ can be reached from $F_1$ by $F_1 \land T \land s'$
  - If it can be reached, get $t$ and try to block it
IC3 - Iteration

- To block $t$, check $F_0 \land T \land t'$
  - If satisfiable, a CEX
  - If not, $t$ is blocked, get a "new" $t$ by $F_1 \land T \land s'$
  - If it can be reached, get $t^*$ and try to block it
  - ......you get the picture 😊

---

General Iteration

---
IC3 - Iteration

- Given an OARS \( \langle F_0, F_1, \ldots, F_k \rangle \), define \( F_{k+1} = P \)
- Apply a backward search
  - Find predecessor \( s \) in \( F_k \) that can reach a bad state
    - Check \( F_k \land T \land \neg P' \)
  - If none exists \( (F_k \land T \Rightarrow P') \), move to next iteration
  - If exists, try to find a predecessor \( t \) to \( s \) in \( F_{k-1} \)
    - \( (F_{k-1} \land T \land s') \)
  - If none exists \( (F_{k-1} \land T \Rightarrow \neg s') \), \( s \) is removed from \( F_k \)
    - \( F_k = F_k \land \neg s \)
  - Otherwise: Recur on \( (t, F_{k-1}) \)
    - We call \( (t, k-1) \) a proof obligation
- If we can reach \( I \), a CEX exists

That Simple?

- Looks simple
- But this “simple” solution does NOT work
- It amounts to States Enumeration
  - Too many states...
- Does IC3 enumerate states?
  - In general - No.
    - It applies generation for removing more than one state at a time
  - Sometimes, yes (when IC3 does not perform well)
Consider the case:

- State $s$ in $F_k$ can reach a bad state in one transition
- $s$ in not reachable (in $k$ transitions):
  - Therefore $F_{k-1} \land T \Rightarrow \neg s'$ holds
- We want to generalize this fact
  - $s$ is a single state
  - Goal: Find a set of states, unreachable in $k$ transitions

**Generalization**

- We know $F_{k-1} \land T \Rightarrow \neg s'$
- And, $\neg s$ is a clause
- Generalization: Find a sub-clause $c \subseteq \neg s$ s.t. $F_{k-1} \land T \Rightarrow c'$
  - Sub clause means less literals
  - Less literals implies less satisfying assignments
    - $(a \lor b \lor c)$ vs. $(a \lor b)$
  - $c \Rightarrow \neg s$ - $c$ is a stronger fact
- $F_k = F_k \land c$
  - More states are removed from $F_k$ making it stronger/more precise (closer to $R_k$)
Generalization

• How do we find a sub-clause \( c \subseteq \neg s \) s.t.
  \( F_{k-1} \land T \Rightarrow c' \)?

• Trial and Error
  - Try to remove literals from \( \neg s \) while \( F_{k-1} \land T \land \neg c' \)
    remains unsatisfiable

• Use the UnSAT Core
  - \( F_{k-1} \land T \land s' \) is unsatisfiable

Observation 1

• Assume a state \( s \) in \( F_k \) can reach a bad state in
  one transition

• Important Fact: \( s \) is not in \( F_{k-1} \) (!!)
  - \( F_{k-1} \land T \Rightarrow F_k \)
  - \( F_k \Rightarrow P \)
  - If \( s \) was in \( F_{k-1} \) we would have found it in an earlier
    iteration

• Therefore: \( F_{k-1} \Rightarrow \neg s \)
Inductive Generalization

- Assume a state $s$ in $F_k$ can reach a bad state in one transition
- Assume $s$ is not reachable (in $k$ transitions):
  - We get $F_{k-1} \land T \Rightarrow \neg s'$ holds
  - BUT, this is equivalent: $F_{k-1} \land \neg s \land T \Rightarrow \neg s'$
    - Since $F_{k-1} \Rightarrow \neg s$
- This looks familiar!
  - $I \Rightarrow \neg s$
    - Otherwise, CEX! ($I \not\Rightarrow \neg s \iff s$ is in $I$)
  - $\neg s$ is inductive relative to $F_{k-1}$

Inductive Generalization

- Find $c \subseteq \neg s$ s.t.
  $F_{k-1} \land c \land T \Rightarrow c'$ and $I \Rightarrow c$ hold

- Define $F_k^* = F_k \land c$

- Since $F_i \Rightarrow F_{i+1}$,
  $c$ is inductive relative to $F_{k-1}, F_{k-2}, \ldots, F_0$
  - Add $c$ to all of these sets
    - $F_i^* = F_i \land c$
  - $F_i^* \land T \Rightarrow F_{i+1}^*$ hold
Observation 2

- Assume a state $s$ in $F_i$ can reach a bad state in a number of transitions.
- $s$ is also in $F_j$ for $j > i$, since $F_i \Rightarrow F_j$.
- A longer CEX may exist:
  - $s$ may not be reachable in $i$ steps, but it may be reachable in $j$ steps.
- If $s$ is blocked in $F_i$, it must be blocked in $F_j$ for $j > i$.
  - Otherwise, a CEX exists.

Push Forward
Push Forward - summary

- s is removed from $F_i$
  - by conjoining a sub-clause c:
    $$F_i = F_i \land c$$
- c is a clause learnt at level i
  Try to push it forward to $j \geq i$
  - If $F_j \land \mathcal{T} \Rightarrow c'$ holds
    - c is implied by $F_j$ in level $j+1$.
      $$F_{j+1} = F_{j+1} \land c$$
  - Else: s was not blocked at level $j > i$
    - Add a proof obligation $(s,j)$
    - If s is reachable from I, CEX!

IC3 - Key Ingredients

- Backward Search
  - Find a state s that can reach a bad state in a number of steps
  - s may not be reachable (over-approximations)
- Block a State
  - Do it efficient, block more than s
    - Generalization
- Push Forward
  - An inductive fact at frame i may also be inductive at higher frames
  - If not, a longer CEX is found
IC3 - High Level Algorithm

If I \land \neg P \text{ is SAT return false; // CEX}
If I \land T \land \neg P' \text{ is SAT return false; // CEX}
OARS = <I,P>; // <F_0,F_1>
k=1
while (OARS.is_fixpoint() == false) do
  while (F_k \land T \land \neg P' \text{ is SAT}) do
    s = get_state();
    If (block_state(s, k) == false) return cex; // recursive function
    extend(OARS);
    push_forward();
return valid;
Lazy Abstraction and SAT-Based Reachability \textit{(with IC3)} in Hardware Model Checking

[Vizel, Grumberg, Shoham 12]

Abstraction

• Fights the state explosion problem
• Removes or simplifies details that are irrelevant

• Abstract model contains less states
• Often - more behaviors
  
  - Over-approximation
Visible Variables Abstraction

Abstraction-Refinement

- Abstract model may contain spurious behaviors
  - Spurious counterexample may exist

- Refinement is applied to remove the spurious behavior
Lazy Abstraction

• Different abstractions at different steps of verification

• Refinement is applied locally, where needed

Locality in IC3

• IC3 applies checks of the form
  - $F_k \land T \land \neg P'$
    • Finds a state in $F_k$ that can reach $\neg P$
  - $F_i \land T \land s'$
    • Finds a predecessor in $F_i$ to the state $s$

• Using only one $T$
  - No unrolling
Our Approach - L-IC3

- Use IC3's local checks for *Lazy Abstraction*
  - Different abstraction at different time frames
  - Use visible variables abstraction
    - Different variables are visible at different time frames

Concrete Model
Using Abstraction

Using Lazy Abstraction
**Lazy Abstraction + IC3 = L-IC3**

- \( <F_0, F_1, \ldots, F_{k+1}> \) - Reachable states

- \( <U_1, U_2, \ldots, U_{k+1}> \) - Abstractions
  - \( U_i \) - set of visible variables
    - \( U_i \) variables have a next state function
    - The rest, inputs
  - \( U_i \subseteq U_{i+1} \)
    - \( U_{i+1} \) is a refinement of \( U_i \)

**L-IC3 Iteration**

- Initialize \( F_{k+1} \) to \( P \)
- Initialize \( U_{k+1} \) to \( U_k \)
- Same problem, the sequence may not be an OARS
Abstract Counterexample

\[ F_i \land T_{i+1} \land s' \quad F_k \land T_{k+1} \land \neg P' \]

Check Spuriousness

- An abstract CEX of length \( k+1 \) exists
- Use an IC3 iteration with the concrete \( T \)
- If a real CEX exists, it will be found
Check Spuriousness (2)

• If no real CEX exists:
  - Compute a strengthened sequence
    \(\langle F^r_0, F^r_1, \ldots, F^r_{k+1} \rangle\)
    • Strengthening by IC3 algorithm
  - The strengthened sequence is an OARS
  - Strengthening eliminates all (real) CEXs of length \(k+1\)

Lazy Abstraction Refinement

• If no real CEX is found by (concrete) IC3 even though (abstract) L-IC3 strengthening failed
  - Abstraction is too coarse

• Refine the sequence \(\langle U_1, U_2, \ldots, U_{k+1} \rangle\) as follows:

• Since \(F^r_i \land T \Rightarrow F^r_{i+1}\)
  - \(F^r_i \land T \land \lnot F^r_{i+1}\) is unsatisfiable
  - Use the UnSAT Core to add visible variables
    • \(U^r_{i+1} = U_{i+1} \cup UCore_i\)
Incrementality

• The concrete IC3 iteration works on the already computed sequence \( <F_0, F_1, ..., F_{k+1}> \)

• At the end of refinement, L-IC3 continues from iteration \( k+2 \)

Experiments - Laziness

<table>
<thead>
<tr>
<th>Test</th>
<th>#Vars</th>
<th>#TF</th>
<th>#TV</th>
<th>#TF</th>
<th>#AV</th>
<th>#TF</th>
<th>#AV</th>
<th>#TF</th>
<th>#AV</th>
<th>#TF</th>
<th>#AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ind 2</td>
<td>5693</td>
<td>7-1</td>
<td>31</td>
<td>8</td>
<td>42</td>
<td>9</td>
<td>51</td>
<td>10-14</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ind 3</td>
<td>11866</td>
<td>1</td>
<td>323</td>
<td>2</td>
<td>647</td>
<td>3</td>
<td>686</td>
<td>4</td>
<td>699</td>
<td>5</td>
<td>705</td>
</tr>
<tr>
<td>Ind 5</td>
<td>3854</td>
<td>1</td>
<td>428</td>
<td>2</td>
<td>453</td>
<td>3</td>
<td>495</td>
<td>4</td>
<td>499</td>
<td>5</td>
<td>503</td>
</tr>
</tbody>
</table>

112
Summary

• Lazy abstraction algorithm for hardware model checking
• Abstraction-Refinement is done incrementally
• We compared our method (L-IC3) to Bradley’s method (IC3)
  - Up to two orders of magnitude runtime improvement

Conclusions

• L-IC3 combines two approaches to fight the state-explosion problem
• L-IC3 exposes and exploits the abstraction, implicit in IC3
Intertwined Forward-Backward Reachability Analysis Using Interpolants

[Vizel, Grumberg, Shoham, TACAS 2013]
Interpolants

- Given an inconsistent pair \((A,B)\) of propositional formulas
- There exists a formula \(I\) such that:
  - \(A \rightarrow I\)
  - \(I \land B\) is unsatisfiable
  - \(I\) is over the common variables of \(A\) and \(B\)
- \(I = \text{Itp}(A,B)\)

Approximated Forward Reachability

- \(F(V)\) - a set of states
- For the unsatisfiable formula \(F(V) \land T(V,V') \land \neg P(V')\), define:
  - \(A = F(V) \land T(V,V')\)
  - \(B = \neg P(V')\)

- Approximated forward reachability:
  \(\text{ApxImg}(F,T) = \text{Itp}(A, B)\)
Backward Reachability Analysis

Does AGp hold?

- \(B_n = \text{PreImg}(B_{n-1}, T)\)
- \(B_2 = \text{PreImg}(B_1, T)\)
- \(B_1 = \text{PreImg}(\neg P, T)\)

Starting states:
Starting from the initial states and making one step forwards, do we reach the bad states?

Duality In a SAT Query

- \(\text{INIT}(V) \land T(V, V') \land \neg P(V')\)
- We tend to read it "Forward"
  - From left to right

Do we reach the bad states?
**Duality In a SAT Query**

- $\text{INIT}(V) \land T(V,V') \land \neg P(V')$
- **We tend to read it "Forward"**
  - From left to right

- **We can also read it "Backward"**
  - From right to left
  - Does the pre-image of the bad states intersect the initial states?

**Approximated Backward Reachability**

- $B(V)$ - a set of states
- For the unsatisfiable formula $\text{INIT}(V) \land T(V,V') \land B(V')$, define:
  
  $A = T(V,V') \land B(V')$
  
  $B = \text{INIT}(V)$

- Approximated backward reachability: $\text{ApxPreImg}(B,T) = \text{Itp}(A, B)$
**Dual Approximated Reachability (DAR)**

- Compute two sequences of reachable states
  - Forward Sequence: \(<F_0,F_1,...,F_n>\)
  - Backward Sequence: \(<B_0,B_1,...,B_n>\)

- Sequences are over-approximations
  - For the forward sequence:
    - \(F_i(V) \land T(V,V') \Rightarrow F_{i+1}(V')\)
    - \(F_i(V) \Rightarrow P(V)\)
  - For the backward sequence
    - \(B_{i+1}(V) \leftarrow T(V,V') \land B_i(V')\)
    - \(B_i(V) \Rightarrow \neg INIT(V)\)

**Dual Approximated Reachability (DAR)**

- Two main phases during the computation
  - Local Strengthening
    - No unrolling
  - Global Strengthening
    - Limited unrolling
    - In case the Local Strengthening fails
Dual Approximated Reachability

• Check the formula:
  \[ \text{INIT}(V) \land T(V, V') \land \neg P(V') \]

\[ F_0 = \text{INIT} \quad \quad \quad B_0 = \neg P \]

• If SAT then CEX is found

Dual Approximated Reachability

• UNSAT:
  \[ \begin{cases} \text{INIT}(V) \land T(V, V') \land \neg P(V') \\ A \end{cases} \quad \begin{cases} B \end{cases} \]

\[ f_0 = \text{INIT} \quad \quad \quad B_1 \quad B_0 = \neg P \]
Local Strengthening - Intuition

What if $F_1$ and $B_1$ intersect each other?

There may be a counterexample

$F_0 = \text{INIT}$

$F_1$  

$B_1$  

$B_0 = \neg P$

Local Strengthening - Intuition

What if $F_1$ and $B_1$ intersect each other?

$F_1(V) \land T(V,V') \land B_0(V')$

$F_0(V) \land T(V,V') \land B_1(V')$

$F_0 = \text{INIT}$

$F_1$

$B_1$

$B_0 = \neg P$
Local Strengthening - Intuition

- Compute forward and backward interpolants
  - $F_2$ is the forward interpolant
  - Backward interpolant strengthens the already existing $B_1$

Local Strengthening - Intuition

- Compute forward and backward interpolants
  - $B_2$ is the backward interpolant
  - $F_1'$ is strengthening the already existing $F_1$
Local Strengthening Fails

\[ F_0(V) \land T(V,V') \land B_0(V) \]

Global Strengthening

- Apply unrolling gradually
  - Start from the initial states
  - Try to reach the backward sequence using an increasing number of T's
**Global Strengthening**

\[ F_0(V_0(V)) \land (V_1(V)) \land (V_2(V)) \land (V_3(V)) \land (V_4(V')) \land (V_5(V'')) \land (B_1(V')) \land (B_2(V'')) \land (B_3(V''')) \land (B_4(V')) \land (B_5(V'')) \land (B_6(V''')) \land \neg P(V''') \]

**Interpolantion-Sequence**

- Given a sequence \(<A_1,...,A_n>\) such that its conjunction is unsatisfiable.
- Then, there exists an interpolation sequence \(<I_0,...,I_n>\) such that:
  - \(I_0 = \text{TRUE}, I_n = \text{FALSE}\)
  - \(I_i \land A_{i+1} \rightarrow I_{i+1}\)
  - \(I_i\) is over the common variables of \(A_1,...,A_i\) and \(A_{i+1},...,A_n\)
Global Strengthening

• If a CEX exists - Full unrolling
• Otherwise, gradually unroll the model
  – Try to reach the Backward sequence
• When the backward sequence is not reachable
  – Extract interpolation sequence
  – Strengthen forward sequence
  – Reapply Local Strengthening
Summary

- Interpolation-based model checking algorithm
- Uses both Forward and Backward traversals
- Two main phases during the computation
  - Local Strengthening
    - No unrolling
  - Global Strengthening
    - Limited unrolling
    - In case the Local Strengthening fails
- Mostly local - No unrolling
  - When unrolling is used, it is restricted

Summary

We presented several methods for SAT-based (unbounded) model checking

- Over-approximate the (forward) reachability analysis
- Apply different methods for making the over-approximation more precise
Thank You

Model checking:

- E.M. Clarke, A. Emerson, Synthesis of Synchronization Skeletons for Branching Time Temporal Logic, workshop on Logic of programs, 1981


- E.M. Clarke, O. Grumberg, D. Peled, Model Checking, MIT press, 1999
• **BDDs:**

• **BDD-based model checking:**

• **SAT-based Bounded model checking:**
  Symbolic model checking using SAT procedures instead of BDDs, A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, Y. Zhu, DAC'99

• **Visible variables abstraction:**

• **Lazy abstraction:**
Interpolation based model checking:

- K. McMillan, Interpolation and SAT-Based Model Checking, CAV’03

- T. Henzinger, R. Jhala, R. Majumdar, K. McMillan, Abstractions from Proofs, POPL’04

- Y. Vizel and O. Grumberg, Interpolation-Sequence Based Model Checking, FMCAD’09

- Y. Vizel, O. Grumberg, S. Shoham, Intertwined Forward-Backward Reachability Analysis Using Interpolants, TACAS’13

Model checking with IC3:

- A. Bradley, SAT-based model checking without unrolling, VMCAI’11

- Y. Vizel, O. Grumberg, S. Shoham, Lazy abstraction and SAT-based reachability in hardware model checking, FMCAD’12