SAT-Based Model Checking:
Interpolation, IC3 and Beyond

Orna Grumberg
Technion, Israel

Based on presentations by Yakir Vizel

Marktoberdorf 2013

1

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

2

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 $\neg(cs_1 \wedge cs_2)$
- **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

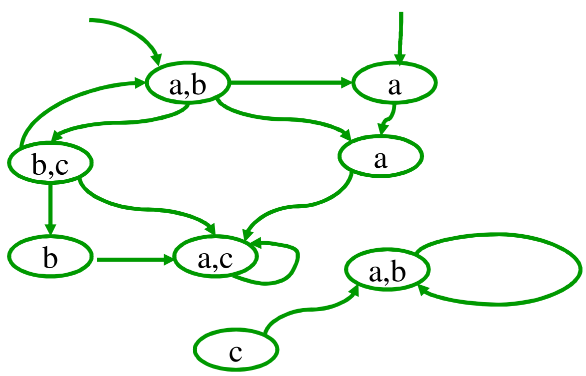
yes, if the system has the property

no + Counterexample, otherwise

7

Model of a system

Kripke structure / transition system



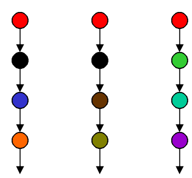
8

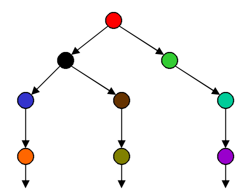
Temporal Logics

- 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)





9

Propositional temporal logic

AP – a set of atomic propositions

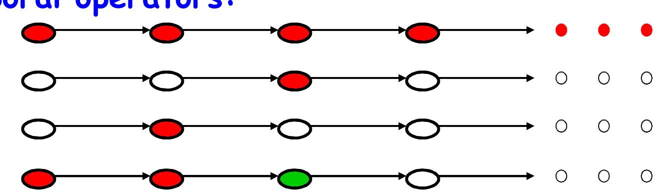
Temporal operators:

Gp

Fp

Xp

pUq



Path quantifiers: **A** for **all** path

E there **exists** a path

10

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.

  $S_k \subseteq \cup_{i=0,k-1} S_i$

  - Result: the set **Reach** of reachable states.

11

Model checking AG p

- *Also called*
  forward reachability analysis

12

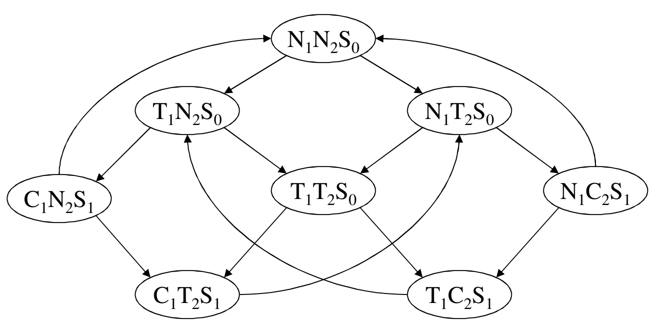
Mutual Exclusion Example

- 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 --- $N_1 N_2 S_0$

$$\begin{array}{lcl} N_1 & \rightarrow & T_1 \\ T_1 \wedge S_0 & \rightarrow & C_1 \wedge S_1 \\ C_1 & \rightarrow & N_1 \wedge S_0 \end{array} \quad \| \quad \begin{array}{lcl} N_2 & \rightarrow & T_2 \\ T_2 \wedge S_0 & \rightarrow & C_2 \wedge S_1 \\ C_2 & \rightarrow & N_2 \wedge S_0 \end{array}$$

13

Mutual Exclusion Example

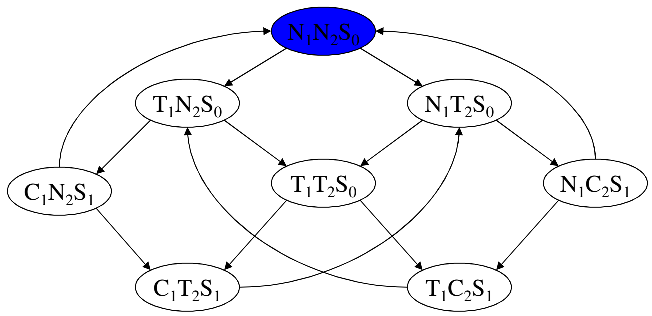


$$M \models AG \neg (C_1 \wedge C_2)$$

The two processes are never in their critical states at the same time

14

Mutual Exclusion Example

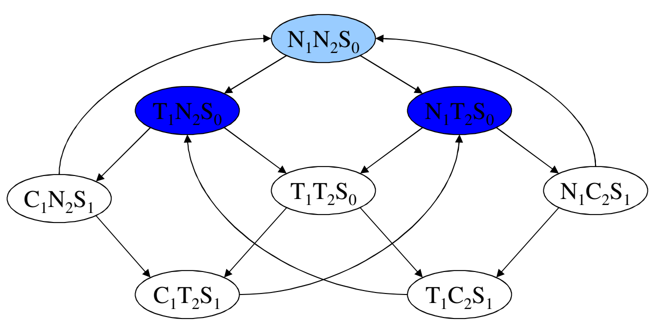


$M \models AG \neg (C1 \wedge C2)$

S_0

15

Mutual Exclusion Example

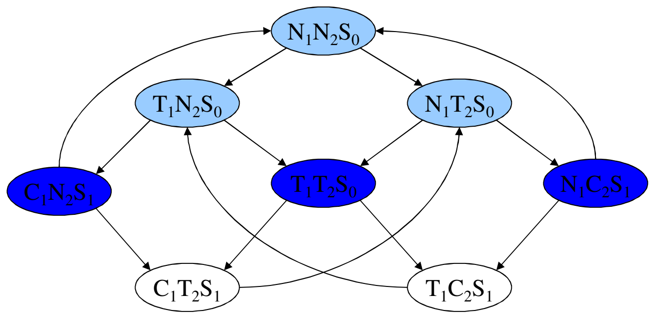


$M \models AG \neg (C1 \wedge C2)$

S_1

16

Mutual Exclusion Example

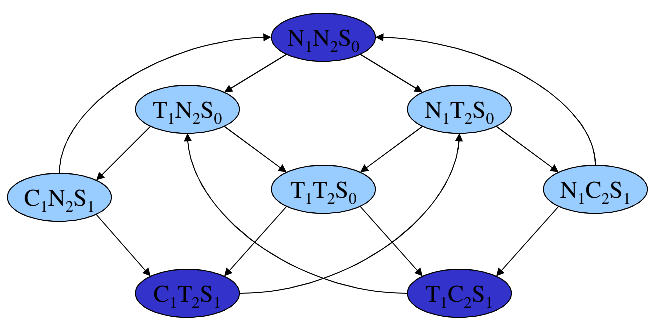


$$M \models AG \neg (C1 \wedge C2)$$

S_2

17

Mutual Exclusion Example

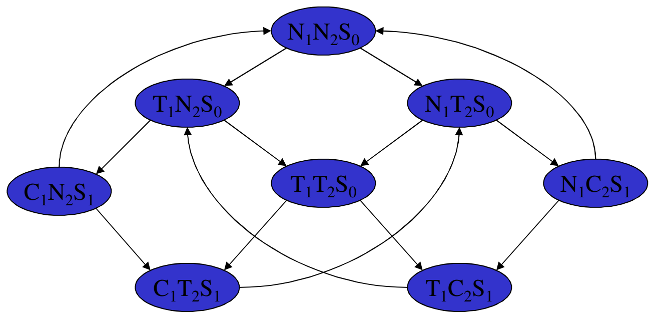


$$M \models AG \neg (C1 \wedge C2)$$

S_3

18

Mutual Exclusion Example

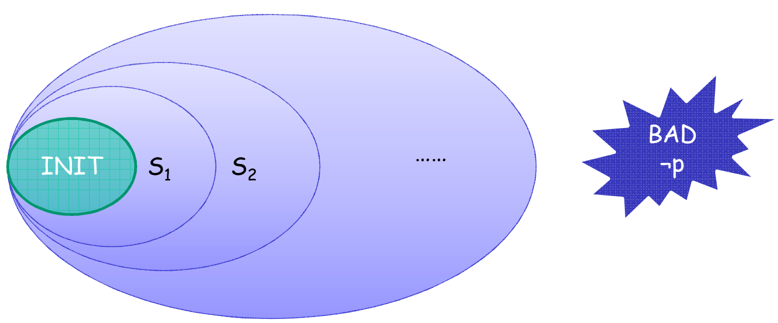


$M \models AG \neg (C1 \wedge C2)$

$S_4 \subseteq S_0 \cup \ldots \cup S_3$

19

Forward Reachability Analysis



- terminates when
  - either a bad state satisfying **¬p** is found
  - or a fixpoint is reached: $S_j \subseteq \cup_{i=0,j-1} S_i$

20

Main limitation

The state explosion problem:

Space and time requirements grow with the size of the model

21

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

22

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]

23

Bounded Model Checking

- Does the system have a counterexample of length k?

$$INIT(V_0) \wedge \neg p(V_0)$$

$$INIT(V_0) \wedge T(V_0, V_1) \wedge \neg p(V_1)$$

$$INIT(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge \neg p(V_2)$$

.

.

.

$$INIT(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge \ldots \wedge T(V_{k-1}, V_k) \wedge \neg p(V_k)$$

24

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:

$R(x,y,z,x',y',z') = \quad x'=y \ \wedge\ y'=z \ \wedge\ \underbrace{z'=1}_{\textbf{error}}$

Initial condition:

$I(x,y,z) = \ x=0 \vee y=0 \vee z=0$

Specification: $AG\ (\ x=0 \vee y=0 \vee z=0)$

Propositional formula for k=2

$f_M = (x_0=0 \vee y_0=0 \vee z_0=0) \wedge$
$(x_1=y_0 \wedge y_1=z_0 \wedge z_1=1) \wedge$
$(x_2=y_1 \wedge y_2=z_1 \wedge z_2=1)$

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

Satisfying assignment: **101 011 111**

This is a counter example!

27

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

28

Verification with SAT solvers

Two successful methods for SAT-based verification are based on:

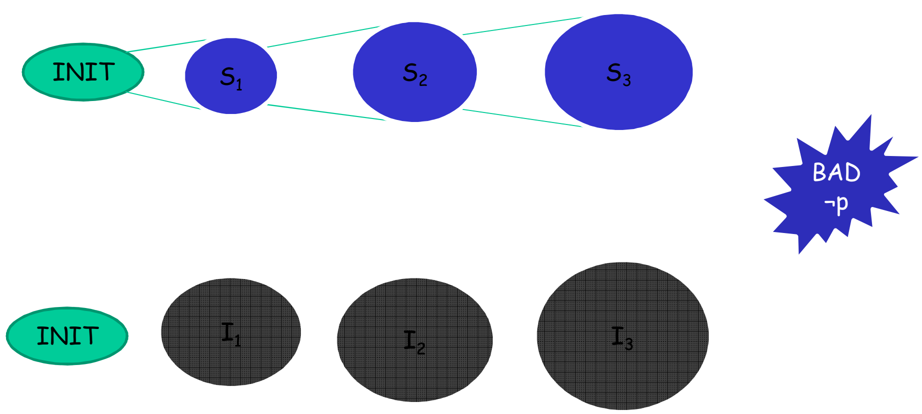
- Interpolation [McMillan 03]
- IC3 [Bradley 11]

In this series of talks

we present methods for enhancing interpolation- and IC3-based model checking

29

A Bit of Intuition



30

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

31

Interpolation [Craig 57]

- If $A \wedge B$ = false, there exists an *interpolant* I for (A,B) such that:

$A \Rightarrow I$

$I \wedge B$ = false

I refers only to common variables of A,B

32

Interpolation (cont.)

Interpolants from proofs

- When $A \wedge B$ 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.

[Pudlak,Krajicek 97]

33

Interpolation in the context of model checking

- Given the following BMC formula φ^k

$$\underbrace{INIT(V_0) \wedge T(V_0,V_1)}_{A} \wedge \underbrace{T(V_1,V_2) \wedge \ldots \wedge T(V_{k-1},V_k) \wedge \neg p(V_k)}_{B}$$

$$\Downarrow$$

I

$$A \Rightarrow I$$

$$I \wedge B \equiv false$$

I is over the common variables of A and B, i.e V_1

34

Interpolation in the context of model checking

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

35

Interpolation-Sequence

- The same BMC formula partitioned in a different manner:

$$\underbrace{INIT(V_0) \wedge T(V_0,V_1)}_{A_1} \wedge \underbrace{T(V_1,V_2)}_{A_2} \wedge \underbrace{T(V_2,V_3)}_{A_3} \wedge \ldots \wedge \underbrace{T(V_{k-1},V_k)}_{A_k} \wedge \underbrace{\neg p(V_k)}_{A_{k+1}}$$

$\mathbf{I_1}$ $\mathbf{I_2}$ $\mathbf{I_3}$ $\mathbf{I_{k-1}}$ $\mathbf{I_k}$

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

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

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

36

Interpolation-Sequence

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

- $I_k \wedge A_{k+1} \Rightarrow false$
  the states in I_k do not violate p

37

Interpolation-Sequence

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

38

Combining Interpolation-Sequence and BMC

- Uses BMC for bug finding
- Uses Interpolation-sequence for computing over-approximation of sets S_j of reachable states

39

Combining Interpolation-Sequence and BMC

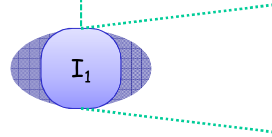
Always terminates

- either when BMC finds a bug:
  $M \mid\neq AGp$
- or when all reachable states has been found:
  $M \models AGp$

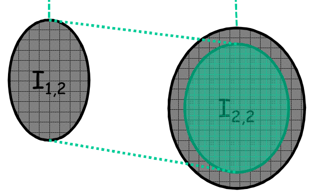
40

Using Interpolation-Sequence

$$INIT(V_0) \wedge T(V_0,V_1) \wedge \neg p(V_1)$$



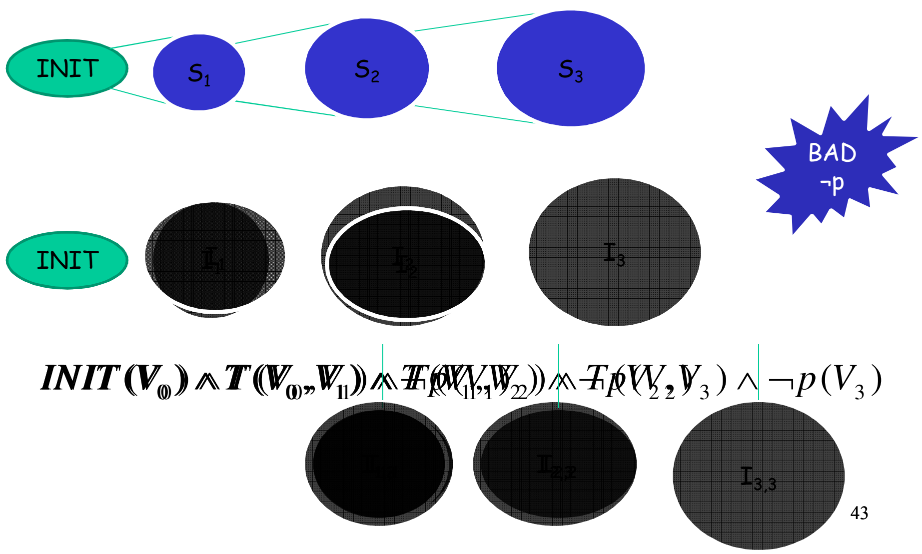
$$INIT(V_0) \wedge T(V_0,V_1) \wedge T(V_1,V_2) \wedge \neg p(V_2)$$



Checking if a "fixpoint" has been reached

- $I_j \Rightarrow \vee_{k=1,j-1} I_k$
- Similar to checking fixpoint in forward reachability analysis :
  $S_j \subseteq \cup_{k=1,j-1} S_k$
- 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



$$INIT(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge T(V_2, V_3) \wedge \neg p(V_3)$$

43

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 φ^k
- $I_j = \Lambda_{k=j,N} I_j^k \; [V^j \leftarrow V]$
- The reachability vector is:
  $\hat{I} = (I_1, I_2, \ldots, I_N)$

44

```
function FixpointReached (Î) // check I_j ⇒ V_{k=1,j-1} I_k
  j=2
  while (j ≤ Î.length) do
     R = V_{k=1,j-1} I_k
     α = I_j ∧ ¬R  // negation of I_j⇒R
     if (SAT(α)==false) then return true
     end if
     j = j+1
  end while
  return false
end function
```

45

Interpolation-Based Model Checking [McM03]

46

Interpolation In The Context of Model Checking

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

$$\underbrace{INIT(V_0) \wedge T(V_0,V_1)}_{A} \wedge \underbrace{T(V_1,V_2) \wedge \ldots \wedge T(V_{k-1},V_k) \wedge (\neg q(V_1) \vee \ldots \vee \neg q(V_k))}_{B}$$

$$\Downarrow$$

I

$$A \Rightarrow I$$

$$I \wedge B \equiv F$$

I is over the common variables of A and B, i.e V_1

47

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.

48

Using Interpolation

$$INIT(V_0) \wedge T(V_0, V_1) \wedge \neg q(V_1)$$

I_1

$$I_1(V_0) \wedge T(V_0, V_1) \wedge \neg q(V_1)$$

I_2

$$I_2(V_0) \wedge T(V_0, V_1) \wedge \neg q(V_1)$$

BAD
¬q

49

Using Interpolation

$$INIT(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge (\neg q(V_1) \vee \neg q(V_2))$$

I'_1

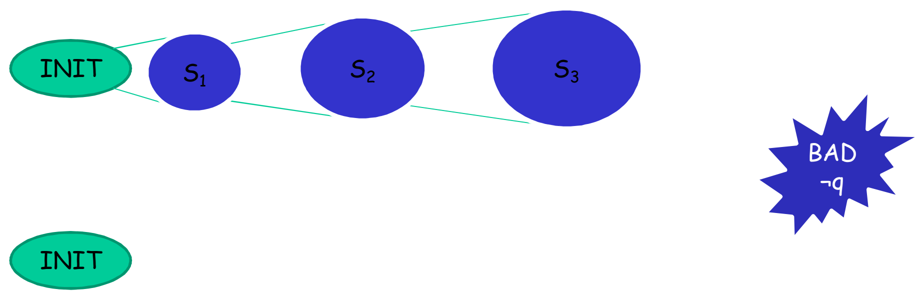
$$I_1'(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge (\neg q(V_1) \vee \neg q(V_2))$$

.
.
.

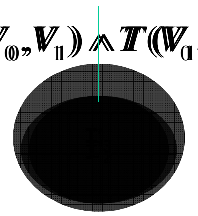
$$I_k'(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge (\neg q(V_1) \vee \neg q(V_2))$$

50

The Analogy to Forward Reachability Analysis



$$INIT'(V_0) \wedge T(V_0,V_1) \wedge T(V_1,V_2) \wedge (\neg q(V_1) \vee \neg q(V_2))$$



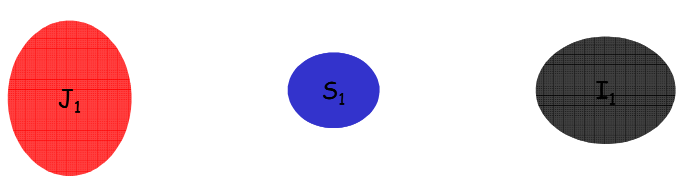
- If BMC finds a satisfying assignment the counterexample might be spurious
  - The set of initial states is over-approximated
- Increase k and start with the original INIT

Charectaristics

- 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-1+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

55

Comparison Analysis

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

56

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.

57

58

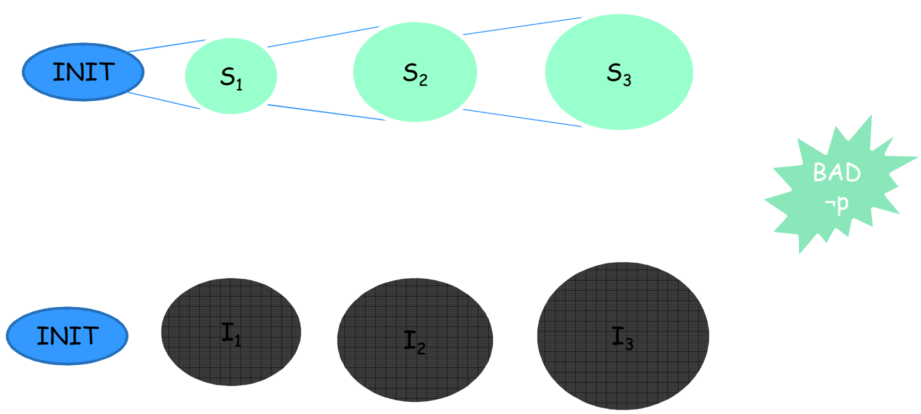
Incremental Construction of Inductive Clauses for Indubitable Correctness

or simply: IC3

[Bradley, VMCAI'11]

A Simplified Description

Recall: Following reachability analysis



60

60

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

61

Induction for proving AG P

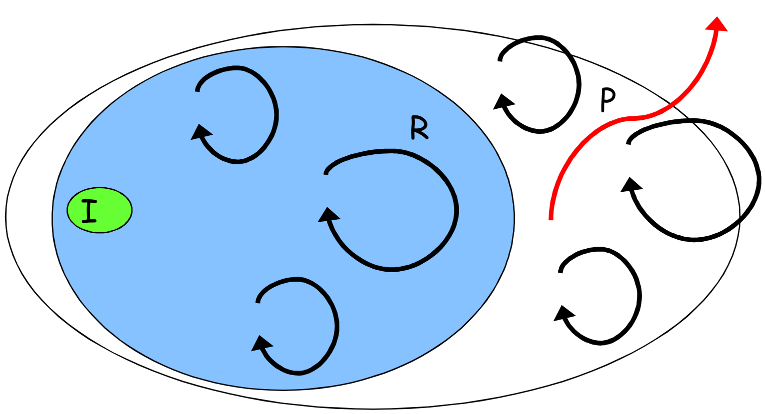
- The simple case: P is an inductive invariant
  - I => P
  - $P \wedge T$ => P'
- P' – the value of P in the next state
- I(V) => P(V)
- $P(V) \wedge T(V, V')$ => P(V')

62

Induction for proving AG P

- Usually, P is not an inductive invariant
- BUT – a stronger inductive invariant R may exist (strengthening)
  - $I \Rightarrow R$
  - $R \wedge T \Rightarrow R'$
  - $R \Rightarrow P$
- 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 => F
    - $F \wedge T \Rightarrow F'$
  - If F is stronger than P, i.e., F => P, then
    - $F \wedge P \wedge T \Rightarrow F' \Rightarrow P'$

65

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

66

IC3 Basics

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

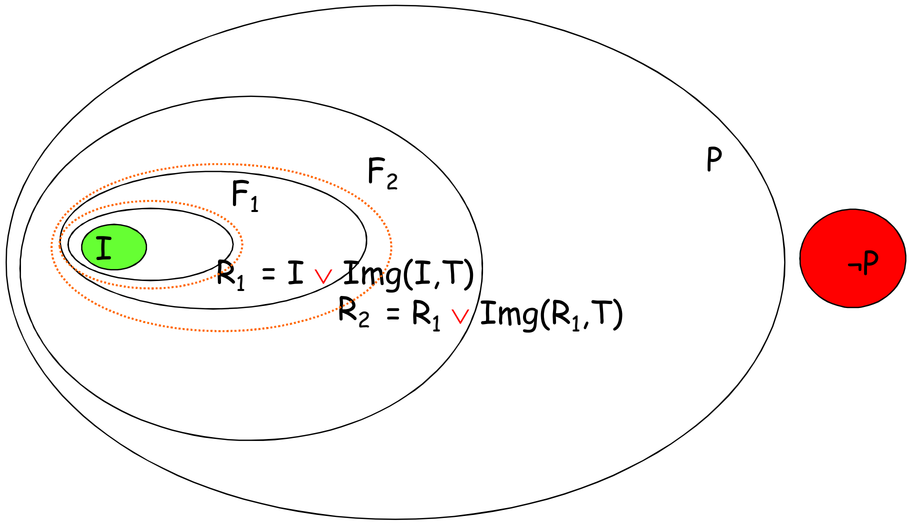
67

IC3 Basics

- P is inductive relative to F if
  - $I \Rightarrow P$
  - $F \wedge P \wedge T \Rightarrow P'$
- Notations:
  - Cube s: conjunction of literals
    - $v_1 \wedge v_2 \wedge \neg v_3$ - Represents a state
  - s is a cube => **¬s is a clause** (DeMorgan)

68

OARS



A Backward Search

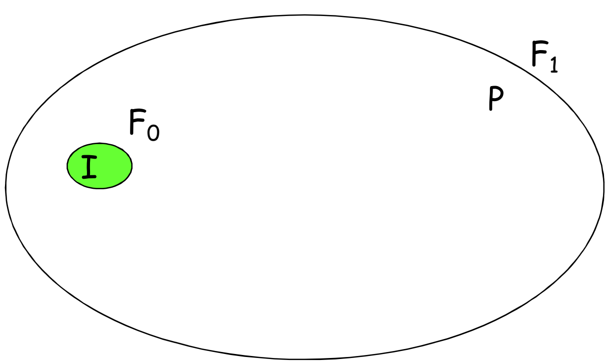
- Search for a predecessor s to some error state: $P \wedge T \wedge \neg P'$
  - If none exists, property P holds:
    - $(P \wedge T \wedge \neg P')$ unsat IFF $(P \wedge T \Rightarrow P')$ valid
- Otherwise, try to block s
  - $P = P \wedge \neg s$
  - BUT, first need to show the s is not reachable

IC3 - Initialization

- Check satisfiability of the two formulas:
  - $I \wedge \neg P$
  - $I \wedge T \wedge \neg P'$
- If both are unsatisfiable then:
  - $I \Rightarrow P$
  - $I \wedge T \Rightarrow P'$
- Therefore
  - $F_0 = I$, $F_1 = P$
    - $\langle F_0, F_1 \rangle$ is OARS

71

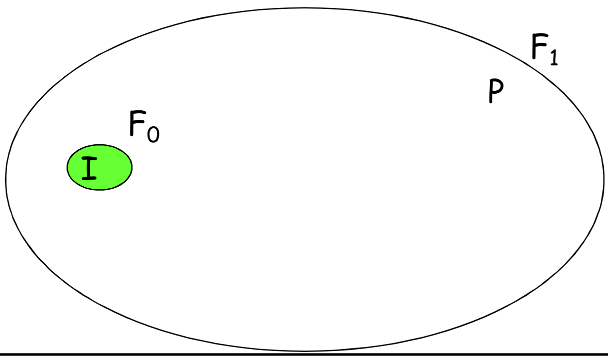
IC3 - Initialization



72

IC3 - Iteration

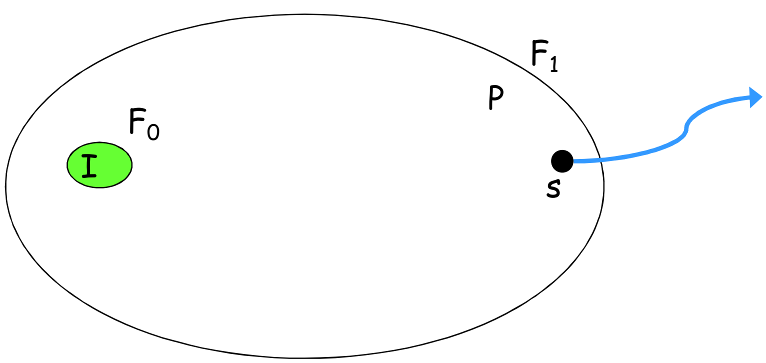
- Our OARS contains F_0 and F_1
  - If P is an inductive invariant – done! ☺
  - Otherwise:
    - F_1 should be strengthened



73

IC3 - Iteration

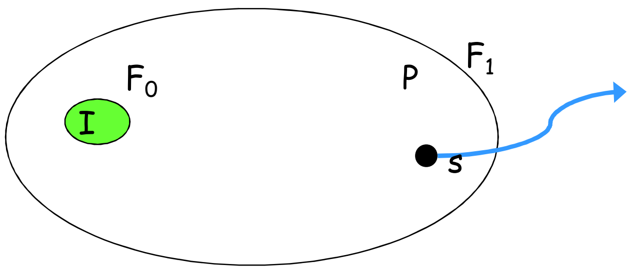
- P is not an inductive invariant
  - $F_1 \wedge T \wedge \neg P'$ is satisfiable
  - From the satisfying assignment get the state s that can reach the bad states



74

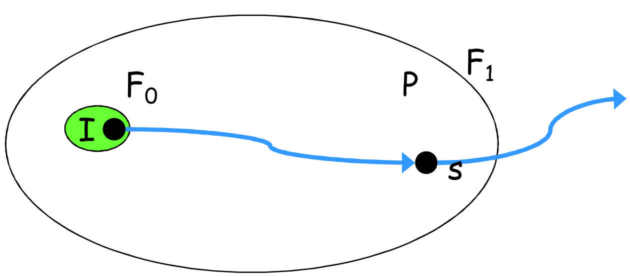
IC3 - Iteration

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



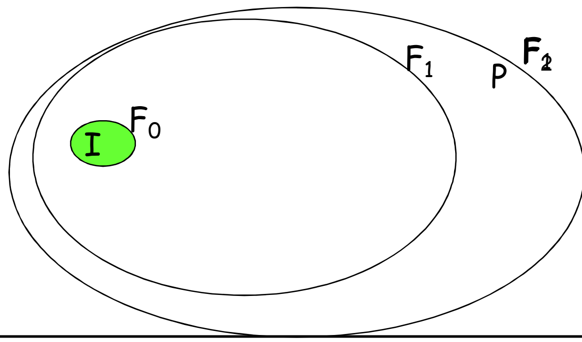
IC3 - Iteration

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



IC3 - Iteration

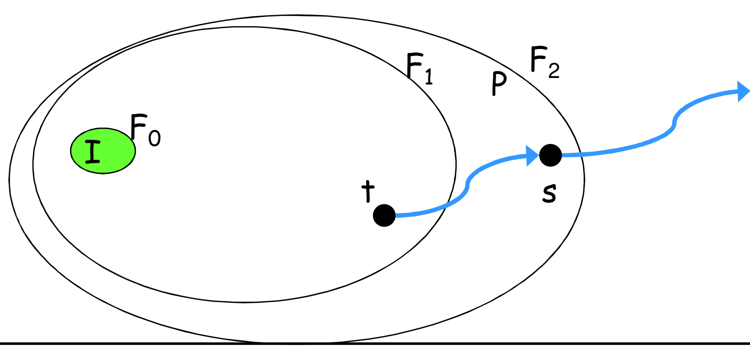
- Iterate this process until $F_1 \wedge T \wedge \neg P'$ becomes unsatisfiable
  - $F_1 \wedge T \Rightarrow P'$ holds
  - F_2 can be defined to be P
    - Any problems/issues with that?



77

IC3 - Iteration

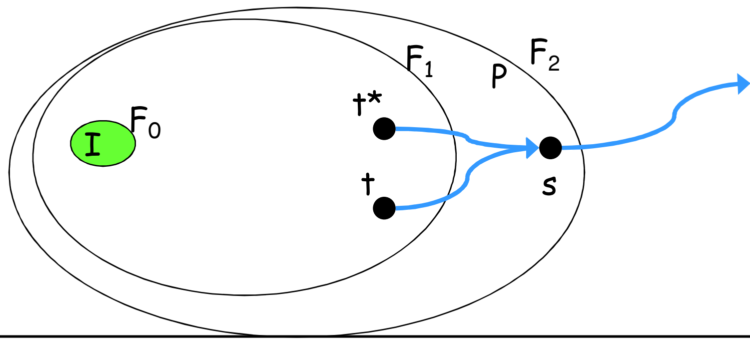
- New iteration, check $F_2 \wedge T \wedge \neg P'$
  - If satisfiable, get s that can reach ¬P
  - Now check if s can be reached from F_1 by $F_1 \wedge T \wedge s'$
  - **If it can be reached, get t and try to block it**



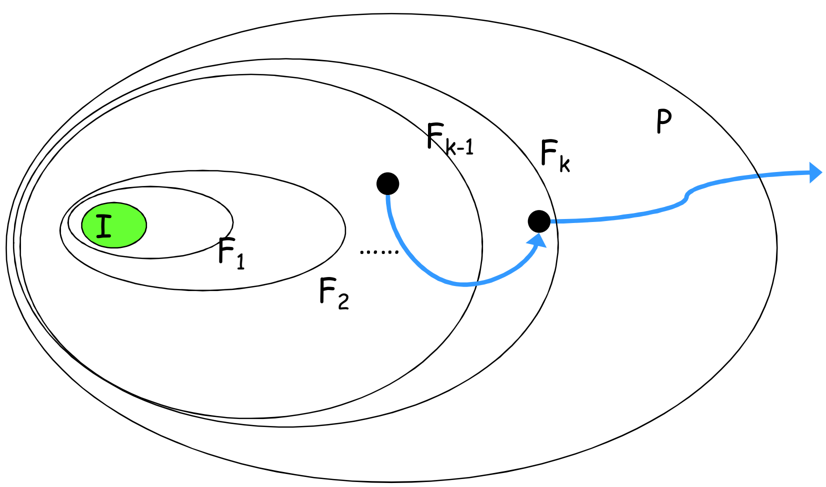
78

IC3 - Iteration

- To block t, check $F_0 \wedge T \wedge t'$
  - If satisfiable, a CEX
  - If not, t is blocked, get a "new" t by $F_1 \wedge T \wedge s'$
  - **If it can be reached, get t* and try to block it**
  - **……You get the picture ☺**



General Iteration



IC3 - Iteration

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

81

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)

82

Generalization

Consider the case:

- State s in F_k can reach a bad state in one transition
- s in not reachable (in k transitions):
  - Therefore $F_{k-1} \wedge T \Rightarrow \neg s'$ 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 $F_{k-1} \wedge T \Rightarrow \neg s'$
- And, $\neg s$ is a clause
- Generalization: Find a sub-clause $c \subseteq \neg s$ s.t. $\mathbf{F_{k-1} \wedge T \Rightarrow c'}$
  - Sub clause means less literals
  - Less literals implies less satisfying assignments
    - $(a \vee b \vee c)$ vs. $(a \vee b)$
  - $c \Rightarrow \neg s$ – c is a stronger fact
- $F_k = F_k \wedge c$
  - 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. $\mathbf{F_{k-1} \wedge T \Rightarrow c'}$?
- Trial and Error
  - Try to remove literals from $\neg s$ while $\mathbf{F_{k-1}} \wedge \mathbf{T} \wedge \neg c'$ remains unsatisfiable
- Use the UnSAT Core
  - $\mathbf{F_{k-1}} \wedge \mathbf{T} \wedge s'$ is unsatisfiable

85

Observation 1

- Assume a state s in F_k can reach a bad state in one transition
- Important Fact: **s is not in $\mathbf{F_{k-1}}$** (!!)
  - $F_{k-1} \wedge T \Rightarrow F_k$
  - $F_k \Rightarrow P$
  - If s was in F_{k-1} we would have found it in an earlier iteration
- Therefore: $F_{k-1} \Rightarrow \neg s$

86

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 $F_{k-1} \wedge T \Rightarrow \neg s'$ holds
- BUT, this is equivalent: $F_{k-1} \wedge \neg s \wedge T \Rightarrow \neg s'$
  - Since $F_{k-1} \Rightarrow \neg s$
- This looks familiar!
  - $I \Rightarrow \neg s$
    - Otherwise, CEX! ($I \not\Rightarrow \neg s \Leftrightarrow s$ is in I)
  - $\neg s$ is inductive relative to F_{k-1}

87

Inductive Generalization

- Find $c \subseteq \neg s$ s.t.
  $F_{k-1} \wedge c \wedge T \Rightarrow c'$ and $I \Rightarrow c$ hold
- Define $F_k^* = F_k \wedge c$
- Since $F_i \Rightarrow F_{i+1}$,
  c is inductive relative to $F_{k-1}, F_{k-2},\dots,F_0$
  - Add c to all of these sets
  - $F_i^* = F_i \wedge c$
- $F_i^* \wedge T \Rightarrow F_{i+1}^*$ hold

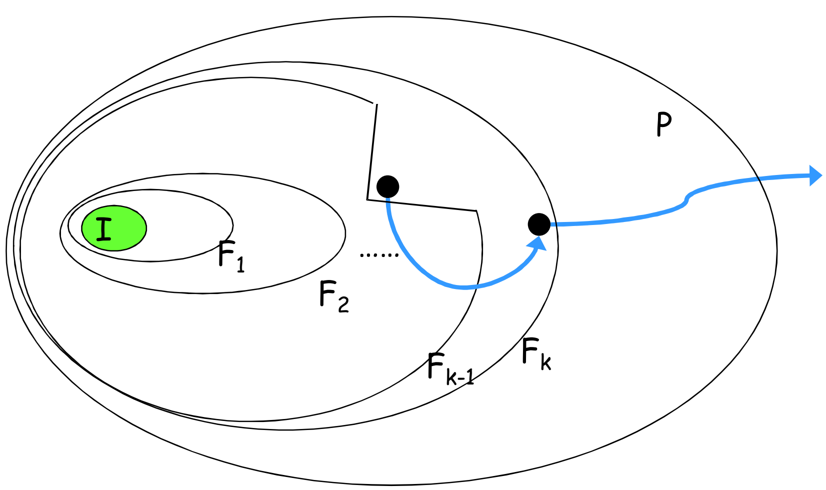
88

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 => 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

89

Push Forward



90

Push Forward - summary

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

91

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

92

IC3 – High Level Algorithm

```
If I ∧ ¬P is SAT return false; // CEX
If I ∧ T ∧ ¬P' is SAT return false; // CEX
OARS = <I,P>;  // <F0,F1>
k=1
while (OARS.is_fixpoint() == false) do
    while (Fk ∧ T ∧ ¬P' is SAT) do
        s = get_state();
        If (block_state(s, k) == false) return cex; //
        recursive function
    extend(OARS);
    push_forward();
return valid;
```

93

94

Lazy Abstraction and SAT-Based Reachability (with IC3) in Hardware Model Checking

[Vizel,Grumberg,Shoham 12]

95

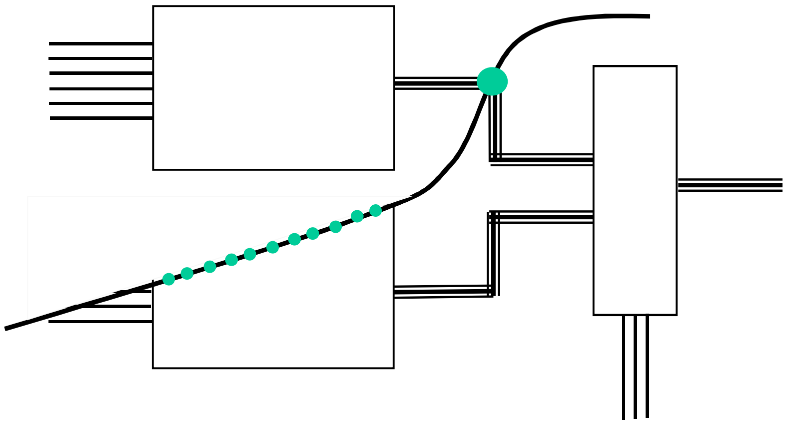
Abstraction

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

- Abstract model contains less states
- Often – more behaviors
  - Over-approximation

96

Visible Variables Abstraction



97

Abstraction-Refinement

- Abstract model may contain spurious behaviors
  - Spurious counterexample may exist
- Refinement is applied to remove the spurious behavior

98

Lazy Abstraction

- Different abstractions at different steps of verification
- Refinement is applied locally, where needed

Locality in IC3

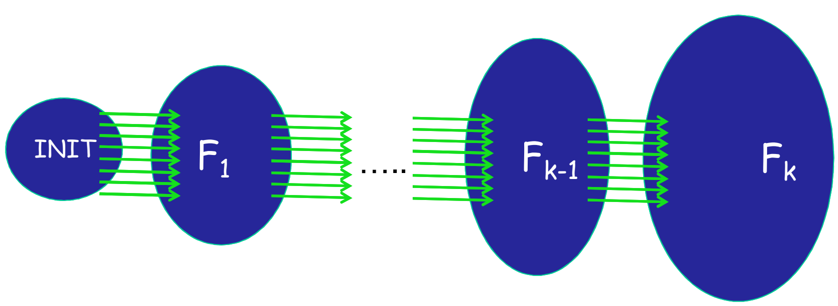
- IC3 applies checks of the form
  - $F_k \wedge T \wedge \neg P'$
    - Finds a state in F_k that can reach $\neg P$
  - $F_i \wedge T \wedge s'$
    - Finds a predecessor in F_i to the state s
- Using only one T
  - No unrolling

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

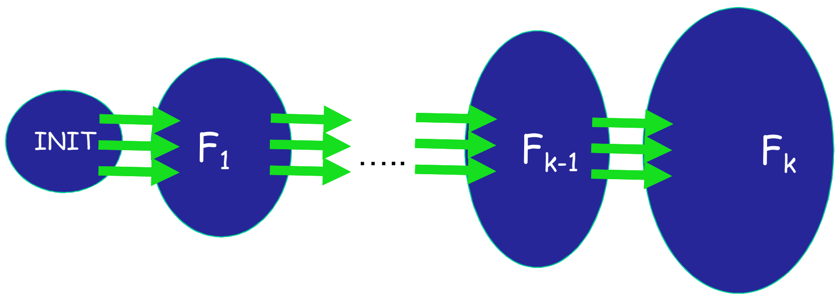
101

Concrete Model



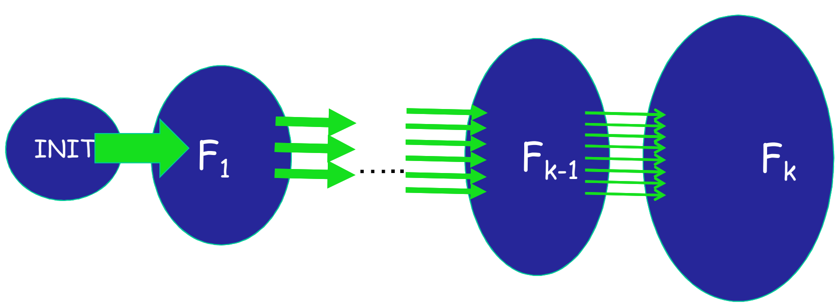
102

Using Abstraction



103

Using Lazy Abstraction



104

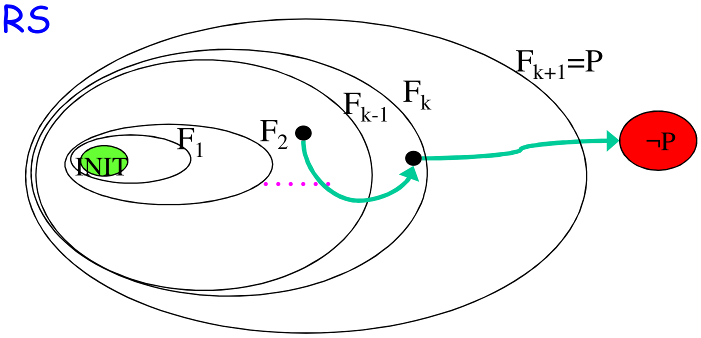
Lazy Abstraction + IC3 = L-IC3

- $\langle F_0, F_1, \ldots, F_{k+1} \rangle$ - Reachable states
- $\langle U_1, U_2, \ldots, U_{k+1} \rangle$ - 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

105

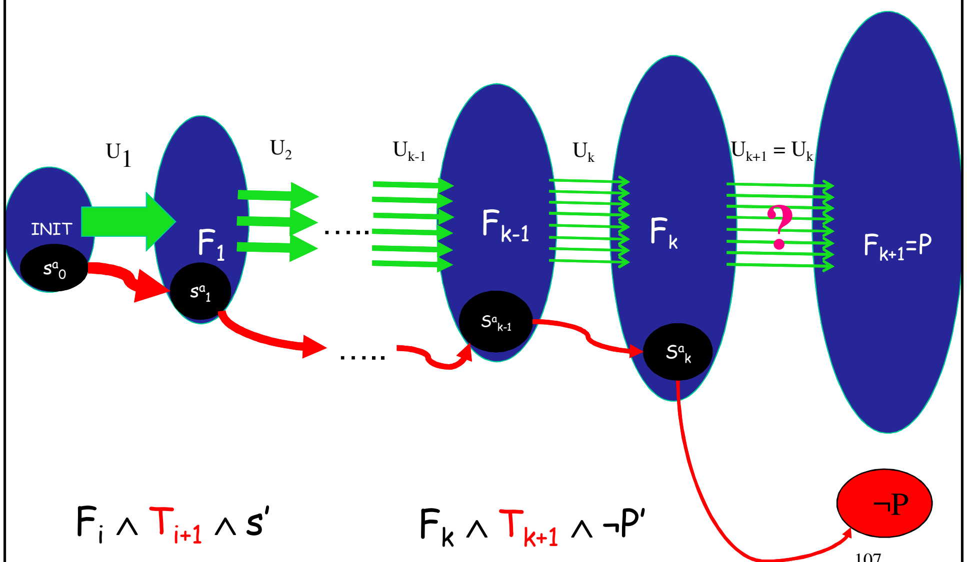
L-IC3 Iteration

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



106

Abstract Counterexample

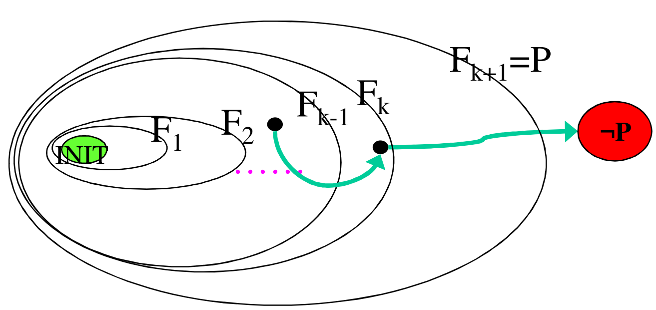


$F_i \wedge T_{i+1} \wedge s'$ $\qquad$ $F_k \wedge T_{k+1} \wedge \neg P'$

107

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



108

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

109

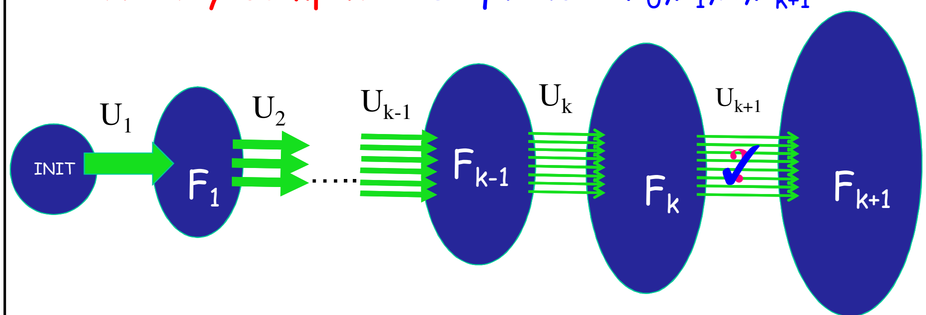
Lazy Abstraction Refinement

- 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 $F^r_i \wedge T \Rightarrow F^{r\prime}_{i+1}$
  - $F^r_i \wedge T \wedge \neg F^{r\prime}_{i+1}$ is unsatisfiable
  - Use the UnSAT Core to add visible variables
    - $U^r_{i+1} = U_{i+1} \cup UCore_i$

110

Incrementality

- The concrete IC3 iteration works on the already computed sequence $\langle F_0, F_1, \ldots, F_{k+1} \rangle$



- At the end of refinement, L-IC3 continues from iteration k+2

Experiments - Laziness

Test	#Vars	#TF	#AV	#TF	#AV	#TF	#AV	#TF	#AV	#TF	#AV
Ind 2	5693	7-1	31	8	42	9	51	10-14	54		
Ind 3	11866	1	323	2	647	3	686	4	699	5	705
		6	713	7	714	8	728	9	743		
Ind 5	3854	1	428	2	453	3	495	4	499	5	503
		6	560	7	574	8	657	9-11	577		

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

113

Conclusions

- L-IC3 combines two approaches to fight the state-explosion problem
- L-IC3 exposes and exploits the abstraction, implicit in IC3

114

115

Intertwined Forward-Backward Reachability Analysis Using Interpolants

[Vizel, Grumberg, Shoham, TACAS 2013]

116

Interpolants

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

Approximated Forward Reachability

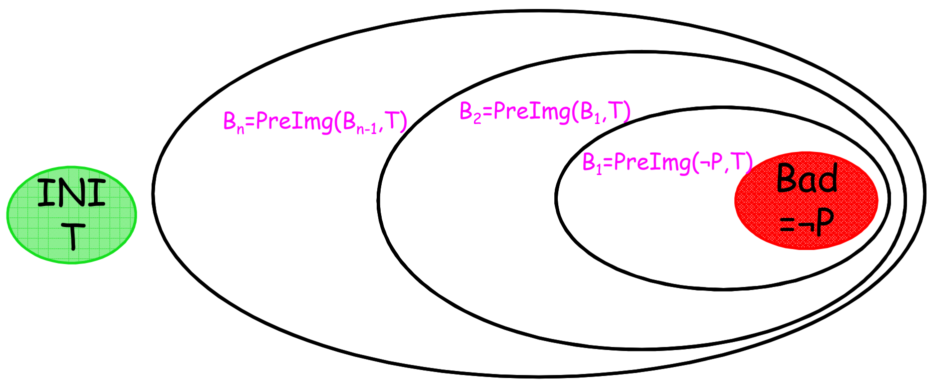
- F(V) – a set of states
- For the unsatisfiable formula $F(V) \wedge T(V,V') \wedge \neg P(V')$, define:

$$A = F(V) \wedge T(V,V')$$
$$B = \neg P(V')$$

- Approximated forward reachability:
  ApxImg(F,T) = Itp(A, B)

Backward Reachability Analysis

Does AGp hold?



119

Duality In a SAT Query

- INIT(V) ∧ T(V,V') ∧ ¬P(V')

Do we reach the bad states?

- We tend to read it "Forward"
  - From left to right

Duality In a SAT Query

- $INIT(V) \wedge T(V,V') \wedge \neg P(V')$

Do we reach the initial states?

- 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 $INIT(V) \wedge T(V,V') \wedge B(V')$, define:

$$A = T(V,V') \wedge B(V')$$
$$B = INIT(V)$$

- Approximated backward reachability: $ApxPreImg(B,T) = Itp(A, B)$

Dual Approximated Reachability (DAR)

- Compute two sequences of reachable states
  - Forward Sequence: $<F_0,F_1,\ldots,F_n>$
  - Backward Sequence: $<B_0,B_1,\ldots,B_n>$
- Sequences are over-approximations
  - For the forward sequence:
    - $F_i(V) \wedge T(V,V') \rightarrow F_{i+1}(V')$
    - $F_i(V) \rightarrow P(V)$
  - For the backward sequence
    - $B_{i+1}(V) \leftarrow T(V,V') \wedge B_i(V')$
    - $B_i(V) \rightarrow \neg INIT(V)$

123

Dual Approximated Reachability (DAR)

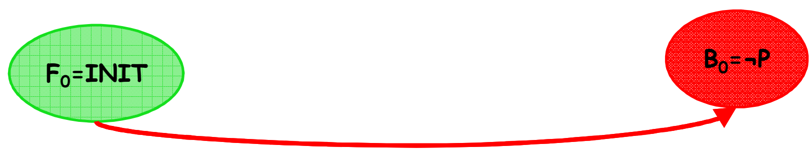
- Two main phases during the computation
  - Local Strengthening
    - No unrolling
  - Global Strengthening
    - Limited unrolling
    - In case the Local Strengthening fails

124

Dual Approximated Reachability

- Check the formula:

$$INIT(V) \wedge T(V,V') \wedge \neg P(V')$$



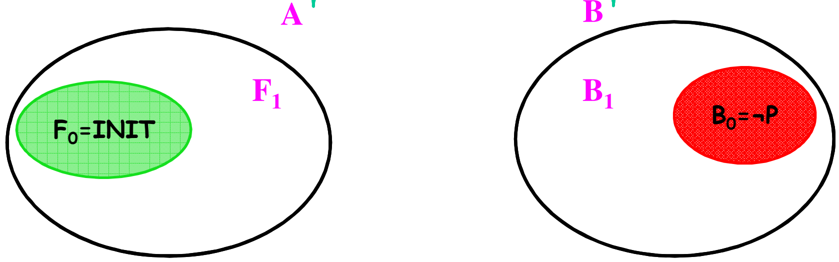
- If SAT then CEX is found

125

Dual Approximated Reachability

- UNSAT:

$$\underbrace{\overbrace{INIT(V)}^{B} \wedge T(V,V')}_{A} \wedge \underbrace{\neg P(V')}_{B}$$

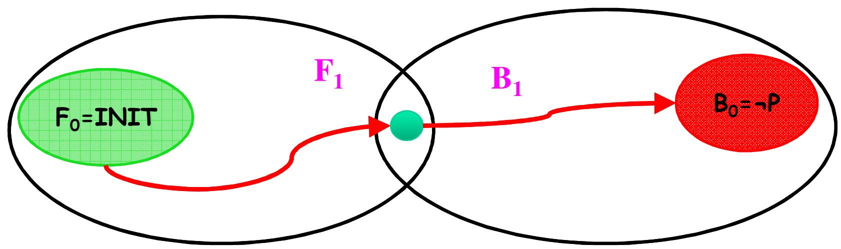


126

Local Strengthening - Intuition

What if F_1 and B_1 intersect each other?

There may be a counterexample



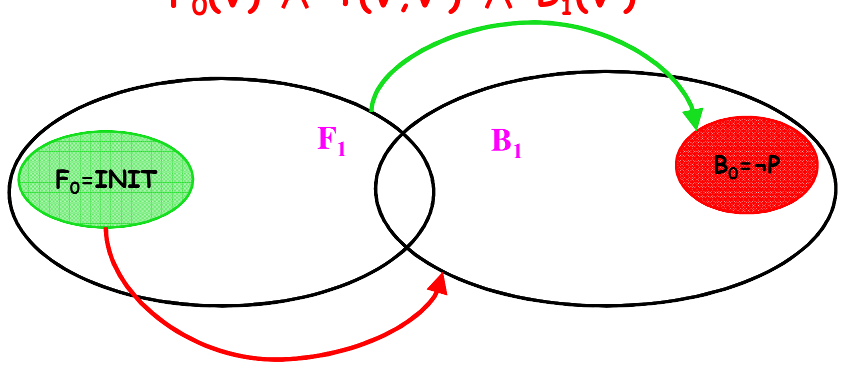
127

Local Strengthening - Intuition

What if F_1 and B_1 intersect each other?

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

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

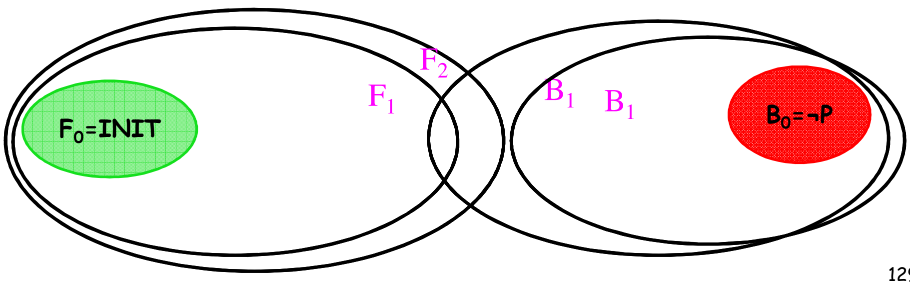


128

Local Strengthening - Intuition

$$\overbrace{F_1(V)}^{B} \overbrace{\land\ T(V,V')}^{A} \overbrace{\land}^{A} \overbrace{B_0(V')}^{B}$$

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

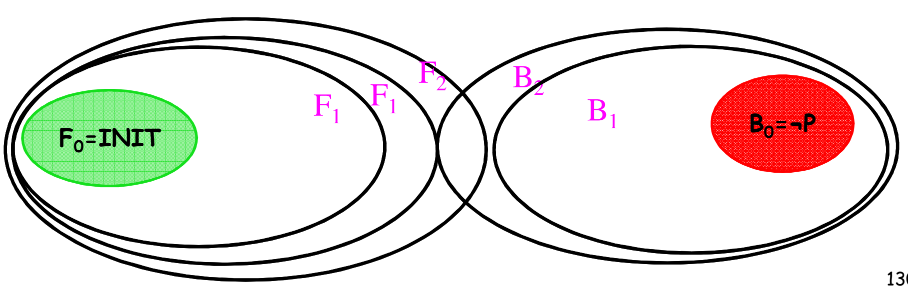


Local Strengthening - Intuition

$$\overbrace{F_0(V)}^{B} \overbrace{\land\ T(V,V')}^{A} \overbrace{\land}^{A} \overbrace{B_1(V')}^{B}$$

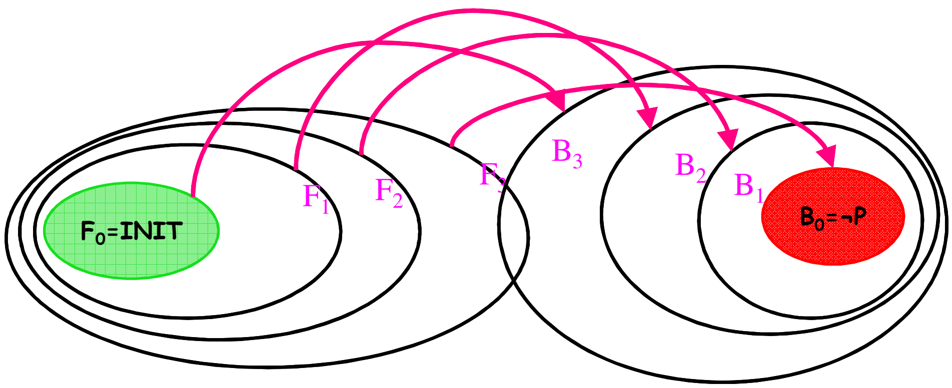
Must **be UnSAT**

- Compute forward and backward interpolants
  - B_2 is the backward interpolant
  - F'_1 is strengthening the already existing F_1



Local Strengthening Fails

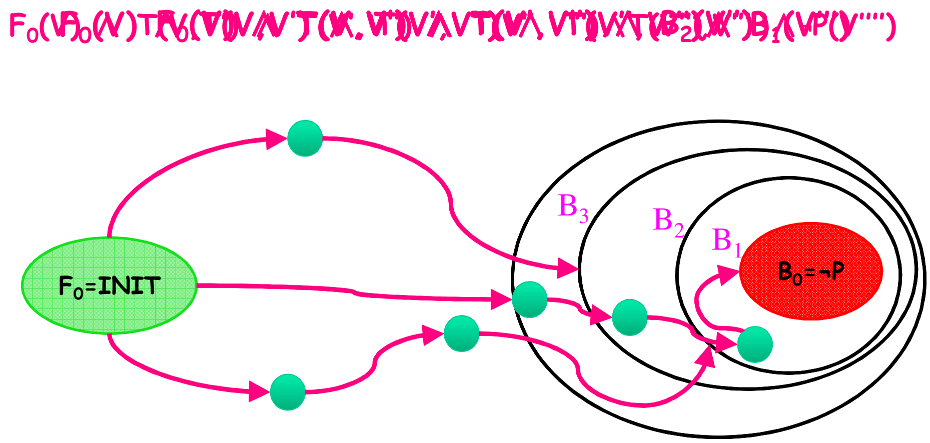
$F(V) \wedge T(V,V') \wedge B(V')$



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



133

Interpolantion-Sequence

- Given a sequence $\langle A_1,\dots,A_n\rangle$ such that its conjunction is unsatisfiable
- Then, there exists an interpolation sequence $\langle I_0,\dots,I_n\rangle$ such that:
  - I_0 = TRUE, I_n=FALSE
  - $I_i \wedge A_{i+1} \rightarrow I_{i+1}$
  - I_i is over the common variables of $A_1,\dots,A_i$ and $A_{i+1},\dots,A_n$

Global Strengthening

$$\overbrace{F_0(V) \wedge T(V,V')}^{A_1} \wedge \overbrace{T(V',V'')}^{A_2} \wedge \overbrace{T(V'',V''')}^{A_3} \wedge \overbrace{B_1(V''')}^{A_4}$$

T $\quad I_1 \quad I_2 \quad I_3 \quad$ F

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

138

Thank You

139

Model checking:

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

- J-P. Queille, J. Sifakis, Specification and Verification of Concurrent Systems in CESAR, international symposium on programming, 1982

- E.M. Clarke, O. Grumberg, D. Peled, Model Checking, MIT press, 1999

140

- **BDDs:**
  R. E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, IEEE transactions on Computers, 1986

- **BDD-based model checking:**
  J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic Model Checking: 10^20 States and Beyond, LICS'90

- **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:**
  R. P. Kurshan, Computer-Aided Verification of coordinating processes – the automata theoretic approach, 1994

- **Lazy abstraction:**
  T. Henzinger, R. Majumdar, R. Jhala, Lazy abstraction, POPL'02.

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

143

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

144