Model Checking:
From BDDs to Interpolation

Orna Grumberg
Technion
Haifa, Israel

Summer school at Bayrischzell 2011

Why (formal) verification?

- safety-critical applications: Bugs are unacceptable!
  - Air-traffic controllers
  - Medical equipment
  - Cars
- Bugs found in later stages of design are expensive, e.g. Intel's Pentium bug in floating-point division
- 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

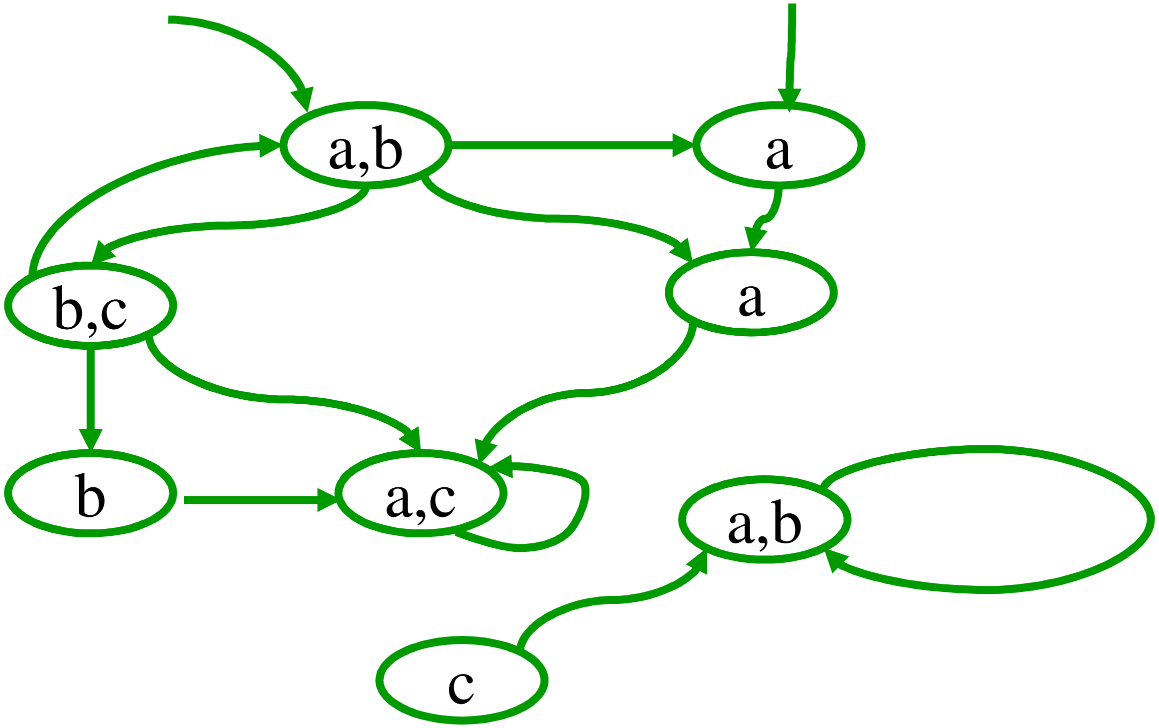
no + Counterexample, otherwise

Model Checking

- Emerging as an industrial standard tool for verification of hardware designs: Intel, IBM, Synopsis, …
- Recently applied successfully also for software verification: SLAM (Microsoft), Java PathFinder and SPIN (NASA), BLAST (EPFL), CBMC (Oxford),…
  - **SLAM won the 2011 CAV award**

Model of a system

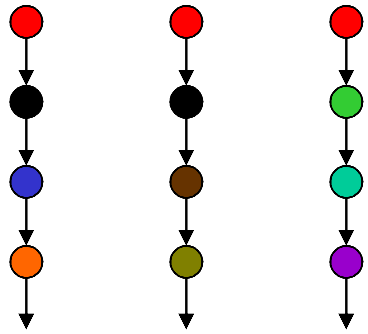
Kripke structure / transition system



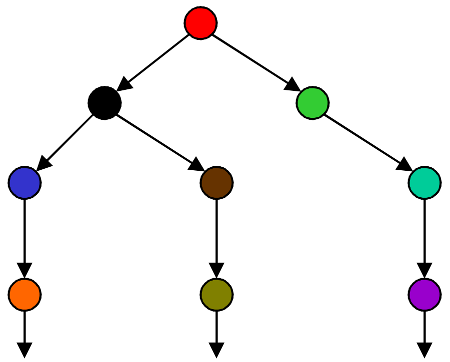
Temporal Logics

- 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)

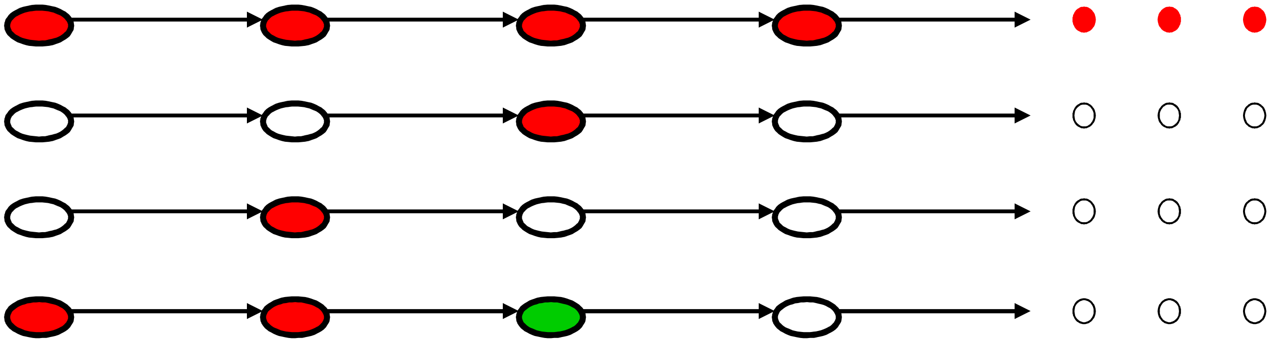


Propositional temporal logic

In **Negation Normal Form**

AP - a set of atomic propositions

Temporal operators:



Gp

Fp

Xp

pUq

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

E there **exists** a path

CTL/CTL*

- **LTL** – interpreted over infinite computation paths
- **CTL** – interpreted over infinite computation trees
- **CTL*** - Allows any combination of temporal operators and path quantifiers. Includes both LTL and CTL

ACTL / ACTL*

The **universal** fragments of CTL/CTL* with only universal path quantifiers

CTL formulas: Example

- **mutual exclusion:** $AG \neg(cs_1 \wedge cs_2)$
- **non starvation:** $AG(request \Rightarrow AF\ grant)$
- **"sanity" check:** $EF\ request$

Model checking

A basic operation: Image computation

Given a set of states Q, Image(Q) returns the set of successors of Q

Image(Q) = **$\{ s' \mid \exists s\, [R(s,s') \wedge Q(s)] \}$**

Model checking AGq 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 $\neg q$.
  - 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.

Model checking **f = AG p**

Given a model **M= < S, I, R, L >**
and a set S_p of states satisfying **q** in M

```
procedure CheckAG ($S_p$ )
    Reach = $\emptyset$
    $S_0$ = I
    k = 0
    while $S_k \not\subset$ Reach do
        If $S_k \cap S_p \neq \emptyset$ return (M $|\neq$ AGq)
        $S_{k+1}$ = Image($S_k$)
        Reach = Reach $\cup S_k$
        k = k+1
    end while
return( Reach, M |= AGp)
```

Model checking AGq

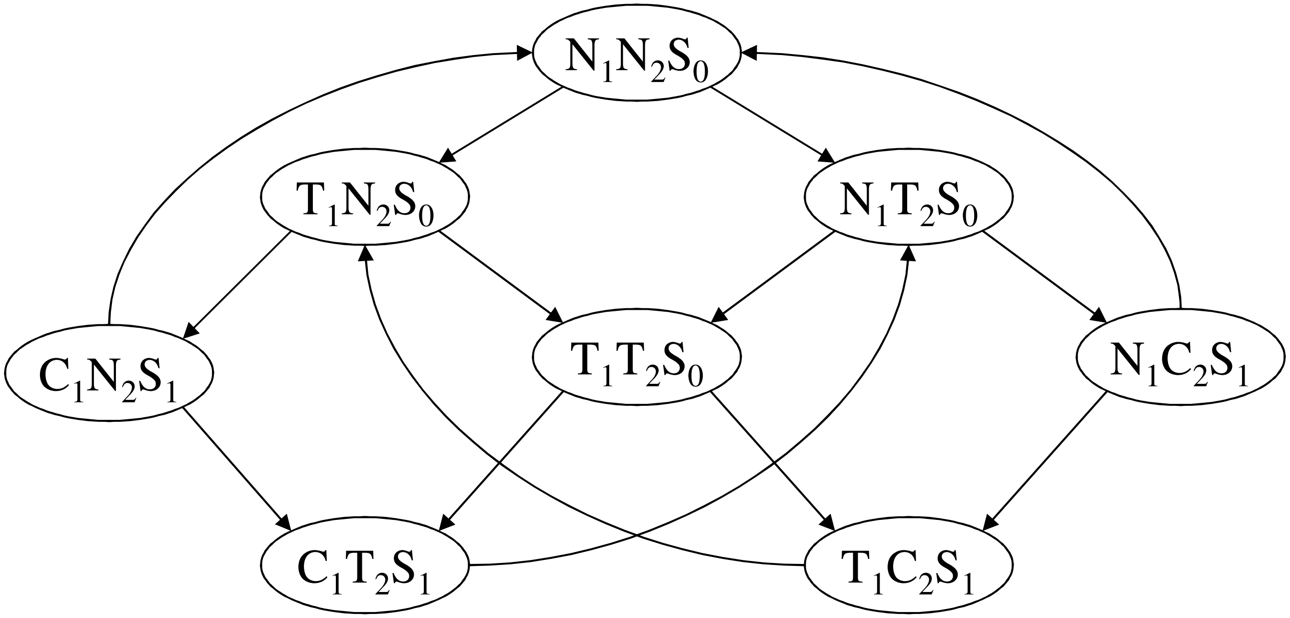
- Also called
  forward reachability analysis

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 --- $N_1 N_2 S_0$

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

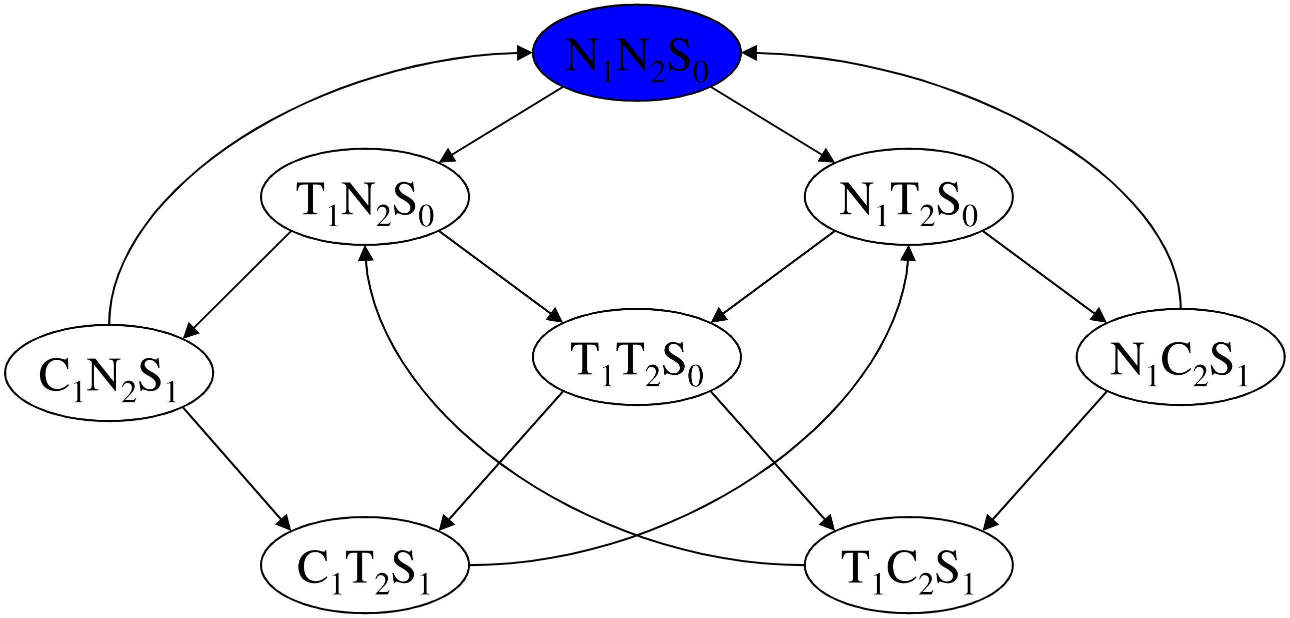
Mutual Exclusion Example



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

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

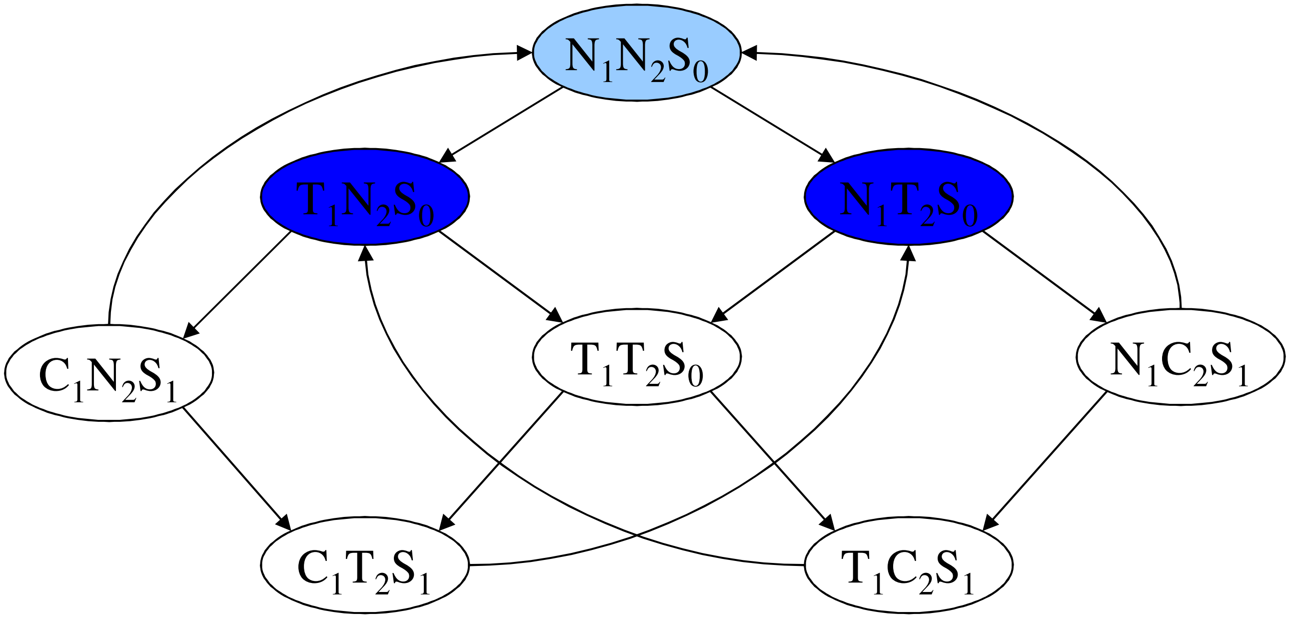
Mutual Exclusion Example



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

S_0

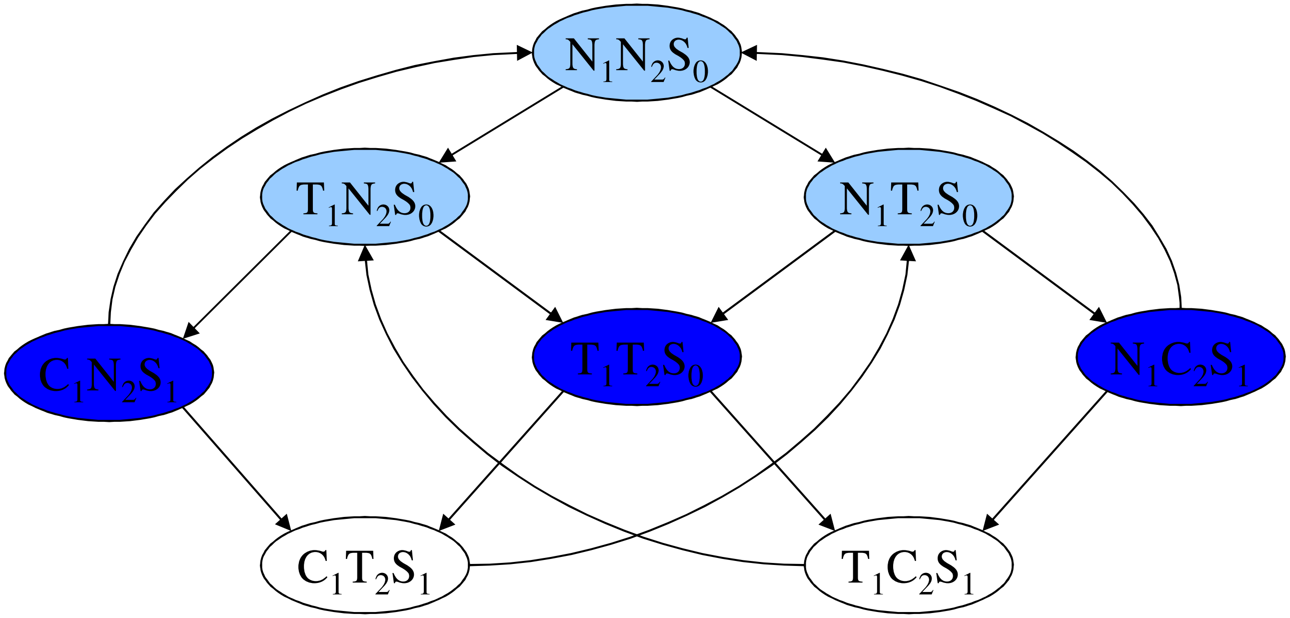
Mutual Exclusion Example



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

S_1

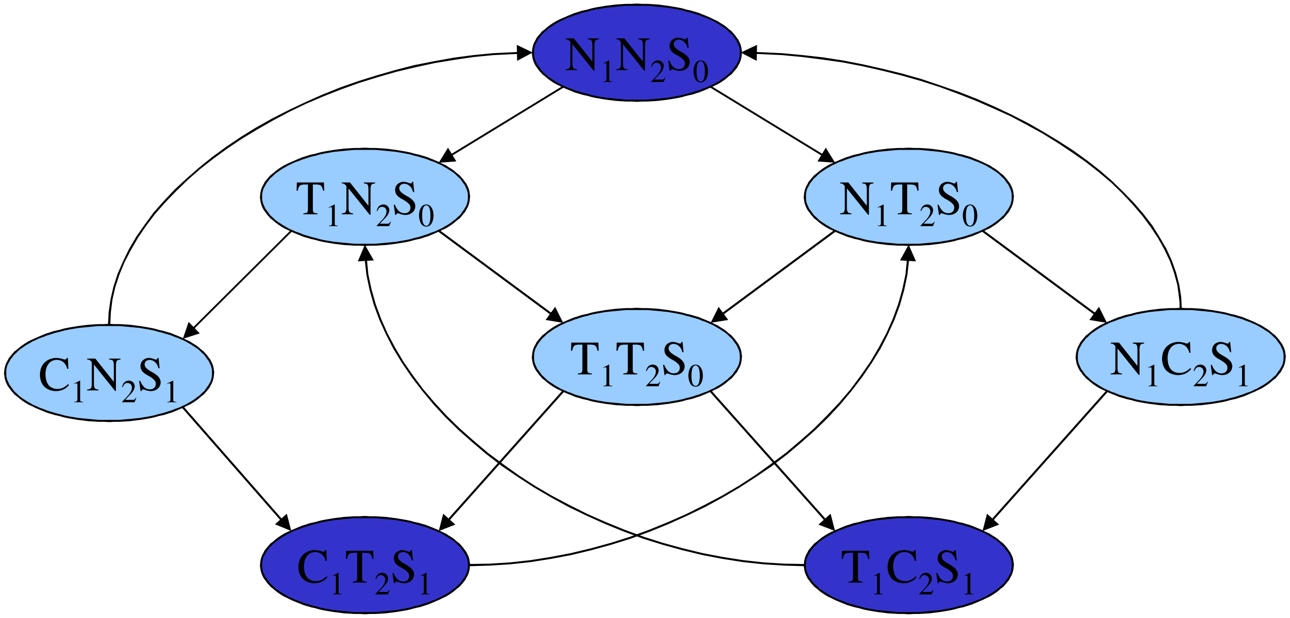
Mutual Exclusion Example



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

S_2

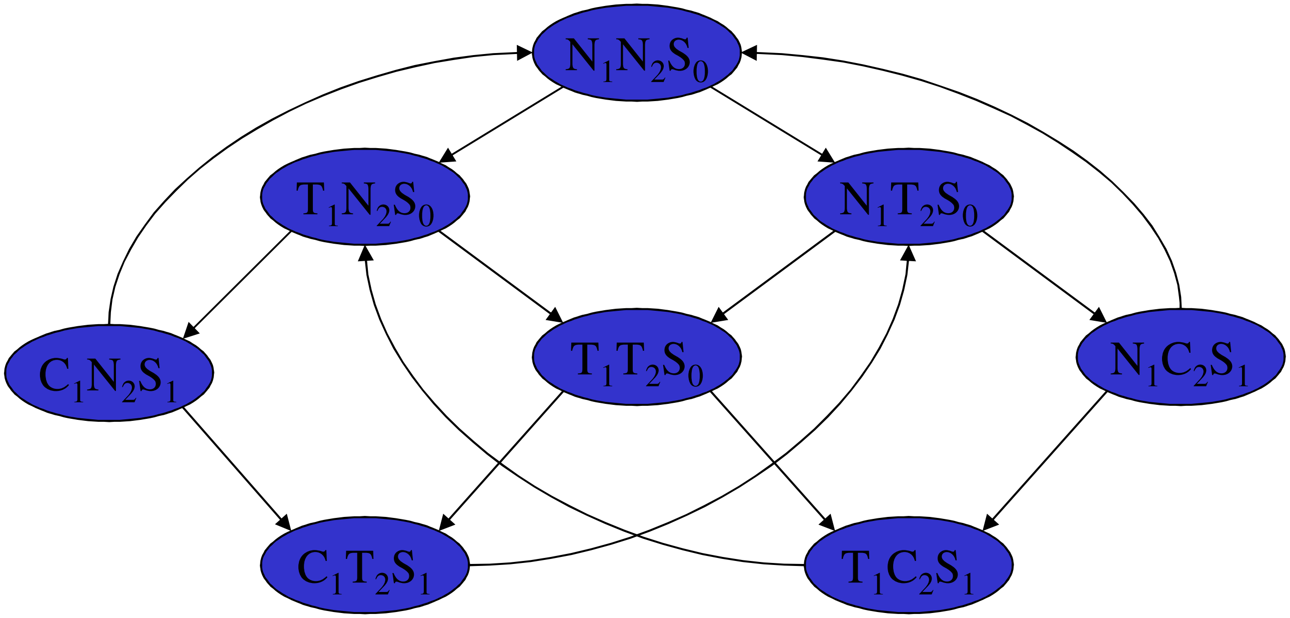
Mutual Exclusion Example



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

S_3

Mutual Exclusion Example



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

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

Main limitation:

The state explosion problem:

Model checking is efficient in time but suffers from high space requirements:

The number of states in the system model grows exponentially with

- the number of variables
- the number of components in the system

Symbolic model checking

A solution to the state explosion problem which uses **Binary Decision Diagrams (BDDs)** to represent the **model** and **sets of states**.

- Suitable mainly for hardware
- Can handle systems with **hundreds** of Boolean variables

Binary decision diagrams (BDDs)

- Data structure for representing Boolean functions
- Often **concise** in memory
- **Canonical** representation
- Most **Boolean operations** on BDDs can be done in **polynomial time** in the BDD size

BDDs in model checking

- Every set $A \subseteq U$ can be represented by its **characteristic function**

$$f_A(u) = \begin{cases} 1 & \text{if } u \in A \\ 0 & \text{if } u \notin A \end{cases}$$

- If the elements of A are encoded by sequences over $\{0,1\}^n$ then f_A is a **Boolean function** and can be represented by a BDD

Representing a model with BDDs

- Assume that **states** in model M are **encoded by $\{0,1\}^n$** and described by Boolean variables **$v_1 \ldots v_n$**
- **Reach, S_k** can be represented by BDDs over **$v_1 \ldots v_n$**
- **R** (a set of pairs of states **(s,s')**) can be represented by a BDD over **$v_1 \ldots v_n$ $v_1' \ldots v_n'$**

Example: representing a model with BDDs

$S = \{ s_1, s_2, s_3 \}$

$R = \{ (s_1,s_2), (s_2,s_2), (s_3,s_1) \}$

State encoding:

s_1: v_1v_2=00 s_2: v_1v_2=01 s_3: v_1v_2=11

For *A* = $\{s_1, s_2\}$ the Boolean formula representing *A*:

$f_A(v_1,v_2) = (\neg v_1 \wedge \neg v_2) \vee (\neg v_1 \wedge v_2) = \neg v_1$

$f_R(v_1,\ v_2,\ v'_1,\ v'_2\ ) =$

$(\neg v_1 \wedge \neg v_2 \wedge \neg v'_1 \wedge v'_2)\ \vee$

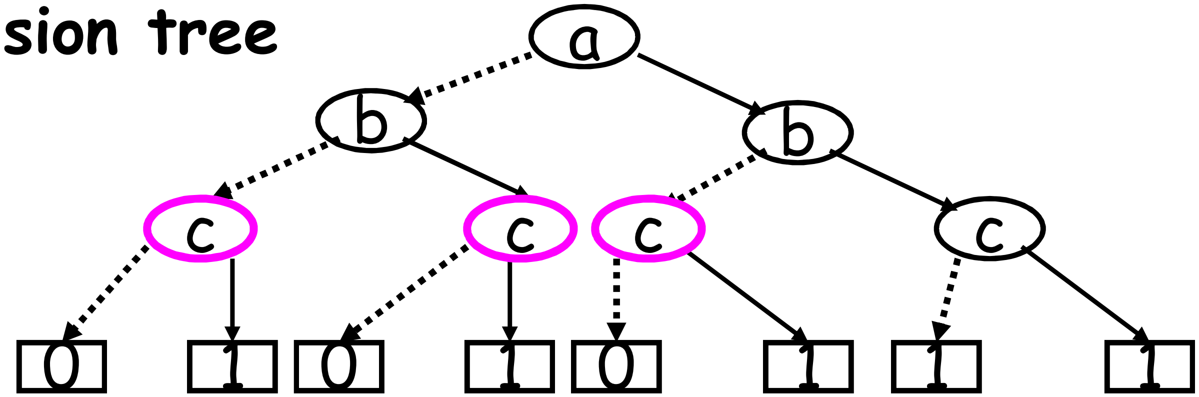
$(\neg v_1 \wedge v_2 \wedge \neg v'_1 \wedge v'_2) \vee$

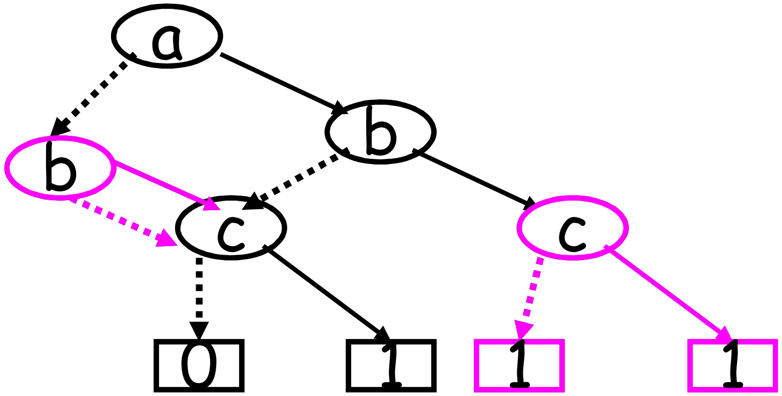
$(v_1 \wedge v_2 \wedge \neg v'_1 \wedge \neg v'_2\ )$

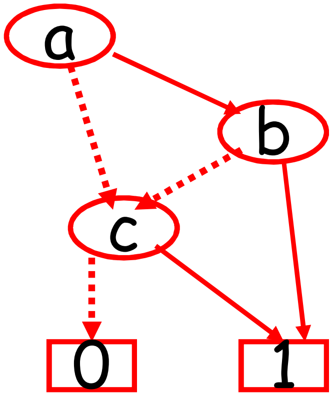
f_A and f_R can be represented by **BDDs**.

BDD for $f(a,b,c) = (a \wedge b) \vee c$

Decision tree







BDD

State explosion problem (cont.)

- state of the art symbolic model checking can handle only systems with a few hundreds of Boolean variables

Other solutions for the state explosion problem are needed

SAT-based model checking

- Translates the model and the specification to a propositional formula
- Uses efficient tools for solving the satisfiability problem

Since the satisfiability problem is **NP-complete**, SAT solvers are based on **heuristics**.

SAT solvers

- Using heuristics, SAT tools can solve very large problems fast.
- They can handle systems with 1000 variables that create formulas with a few thousands of variables.

GRASP (Silva, Sakallah)
Prover (Stalmark)
Chaff (Malik)
MiniSat, …

Model Checking:
From BDDs to Interpolation

Lecture 2

Orna Grumberg
Technion
Haifa, Israel

Summer school at Bayrischzell 2011

SAT-based model checking

- Translate the model and the specification to a propositional formula
- Use efficient tools (SAT solvers) for solving the satisfiability problem

Bounded model checking for checking AGp

- Unwind the model for k levels, i.e., construct all computation of length k
- If a state satisfying $\neg$p is encountered, then produce a counterexample

The method is suitable for **falsification**, not verification

Bounded model checking with SAT

- Construct a formula $\mathbf{f}_{M,k}$ describing all possible computations of M of length k
- Construct a formula $\mathbf{f}_{\varphi,k}$ expressing that $\varphi = \mathbf{EF}\neg\mathbf{p}$ holds within k computation steps
- Check whether $\mathbf{f} = \mathbf{f}_{M,k} \wedge \mathbf{f}_{\varphi,k}$ is satisfiable

If f is satisfiable then $M \not\models AGp$

The satisfying assignment is a **counterexample**

Example - shift register

Shift register of 3 bits: $\langle x, y, z \rangle$

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

$$
\begin{aligned}
f_M = {} & (x_0=0 \vee y_0=0 \vee z_0=0) \wedge \\
& (x_1=y_0 \wedge y_1=z_0 \wedge z_1=1) \wedge \\
& (x_2=y_1 \wedge y_2=z_1 \wedge z_2=1)
\end{aligned}
$$

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

Satisfying assignment: 101 011 111

This is a counter example!

A remark

In order to describe a computation of length k by a propositional formula we need k copies of the state variables.

With BDDs we use only two copies of current and next states.

Bounded model checking

- Can handle **LTL** formulas, when interpreted over finite paths
- 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

BDDs versus SAT

- SAT-based tools are mainly useful for **bug finding** while BDD-based tools are suitable for **full verification**

- some examples work better with BDDs and some with SAT.

Verification with SAT solvers

Interpolation-Sequence Based Model Checking [VG09]

Inspired by:

- forward reachability analysis

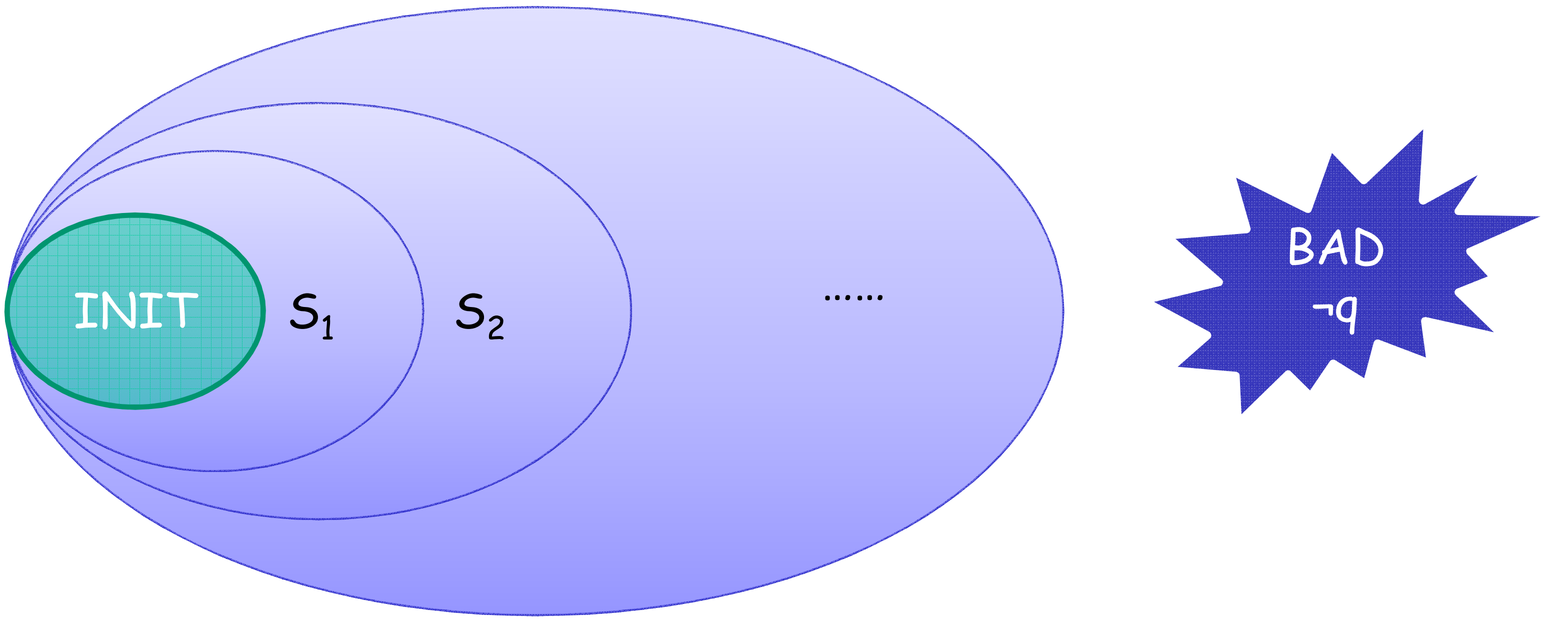
Combines:

- Bounded Model Checking
- Interpolation-sequence

Obtains:

- SAT-based model checking algorithm for full verification

Forward Reachability Analysis



Forward reachability analysis

- S_j is the set of states reachable from some initial state in j steps
- termination when
  - either a bad state satisfying $\neg$**q** is found
  - or a fixpoint is reached:

    $S_j \subseteq \cup_{i=0,j-1} S_i$

Bounded Model Checking

- Does the system have a counterexample of length k?

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

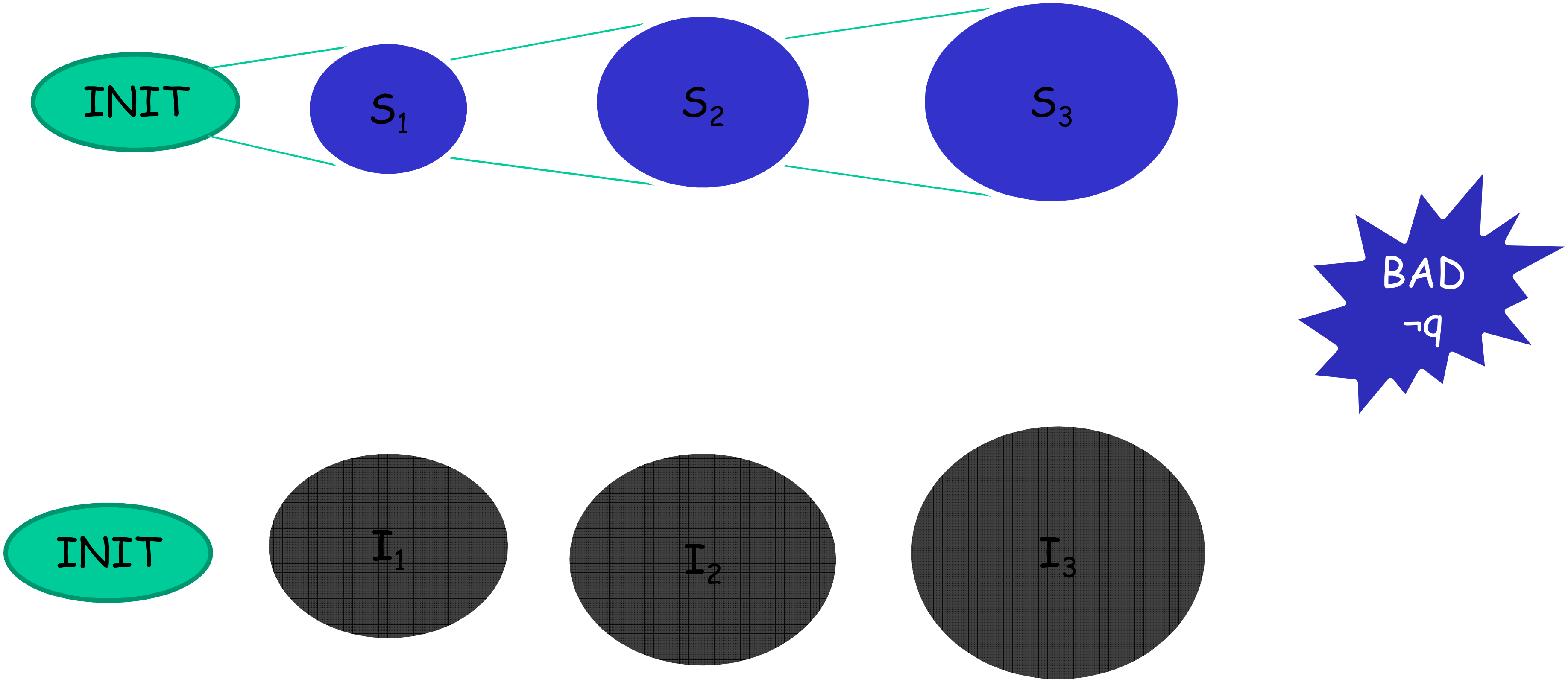
$$INIT(V_0) \wedge T(V_0, V_1) \wedge \neg q(V_1)$$

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

$$\vdots$$

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

A Bit of Intuition



Interpolation

(Craig,57)

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

$$A \Rightarrow I$$

$$I \wedge B = \text{false}$$

I refers only to common variables of A,B

Interpolation (cont.)

- Example:

  $A = p \wedge q, \quad B = \neg q \wedge r, \quad I = q$

- Interpolants from proofs

  given a resolution refutation (proof of unsatisfiability) of $A \wedge B$,
  I can be derived in linear time.

(Pudlak,Krajicek,97)

Interpolation In The Context of Model Checking

- Given the following BMC formula φ^k

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

$$\Big\Downarrow$$

$$\mathbf{I}$$

$$A \Rightarrow I$$

$$I \wedge B \equiv F$$

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

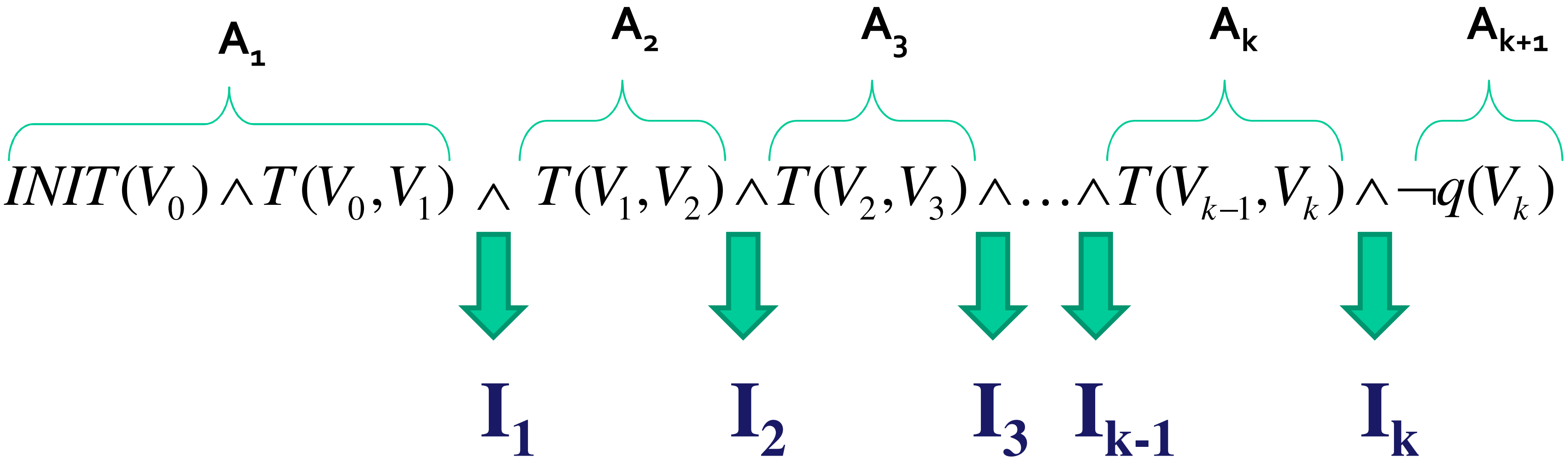
Interpolation in the context of model checking

- I is over V_1
- $A \Rightarrow I$
  - I over-approximates the set S_1

- $I \wedge B \equiv F$
  - States in I cannot reach a bug in k-1 steps

Interpolation-Sequence

- The same BMC formula partitioned in a different manner:



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

$$I_0 = T, I_{k+1} = F$$

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

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

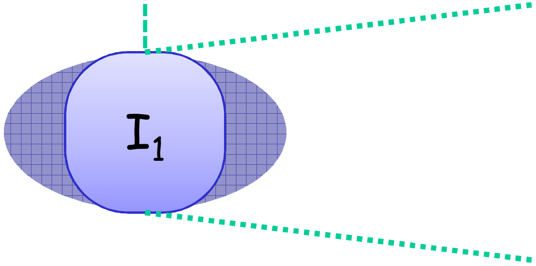
Interpolation-Sequence (2)

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

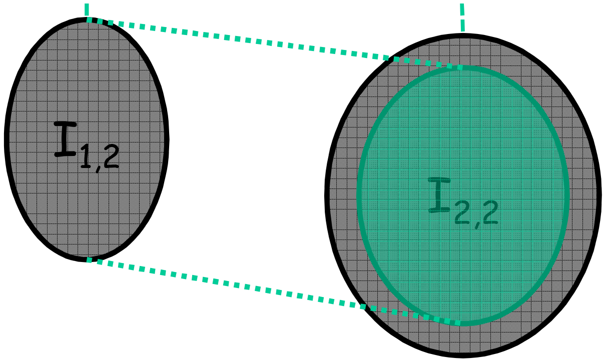
Interpolation-Sequence Based Model Checking

Using Interpolation-Sequence

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



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

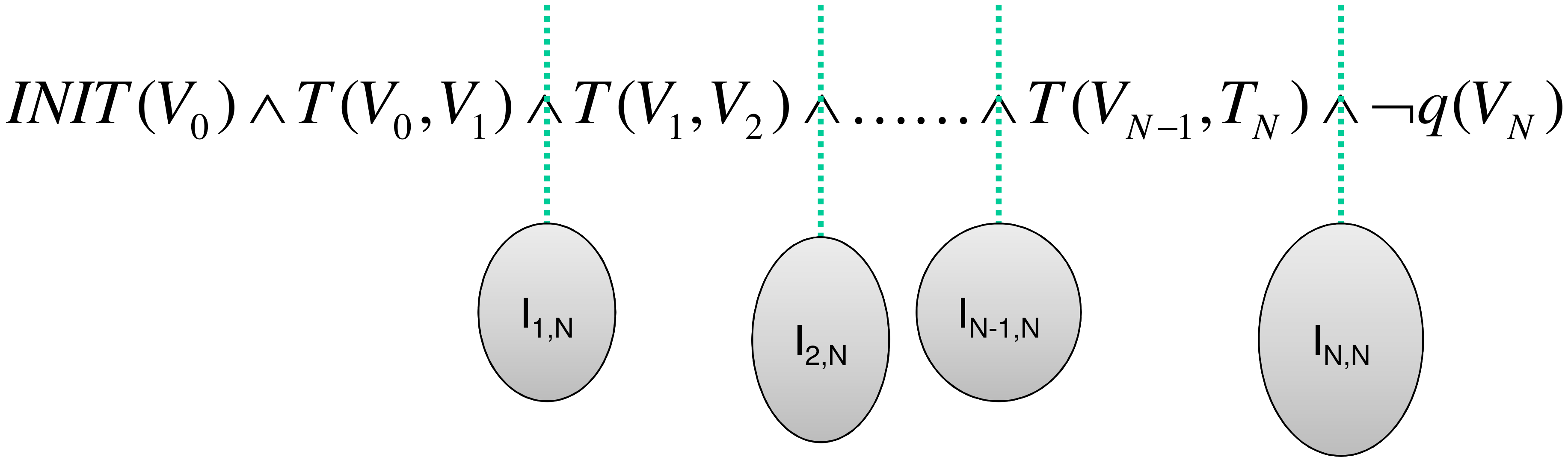


Combining Interpolation-Sequence and BMC

- A way to do reachability analysis using a SAT solver.
- Uses the original BMC loop and adds an inclusion check for full verification.
- Similar sets to those computed by Forward Reachability Analysis but over-approximated.

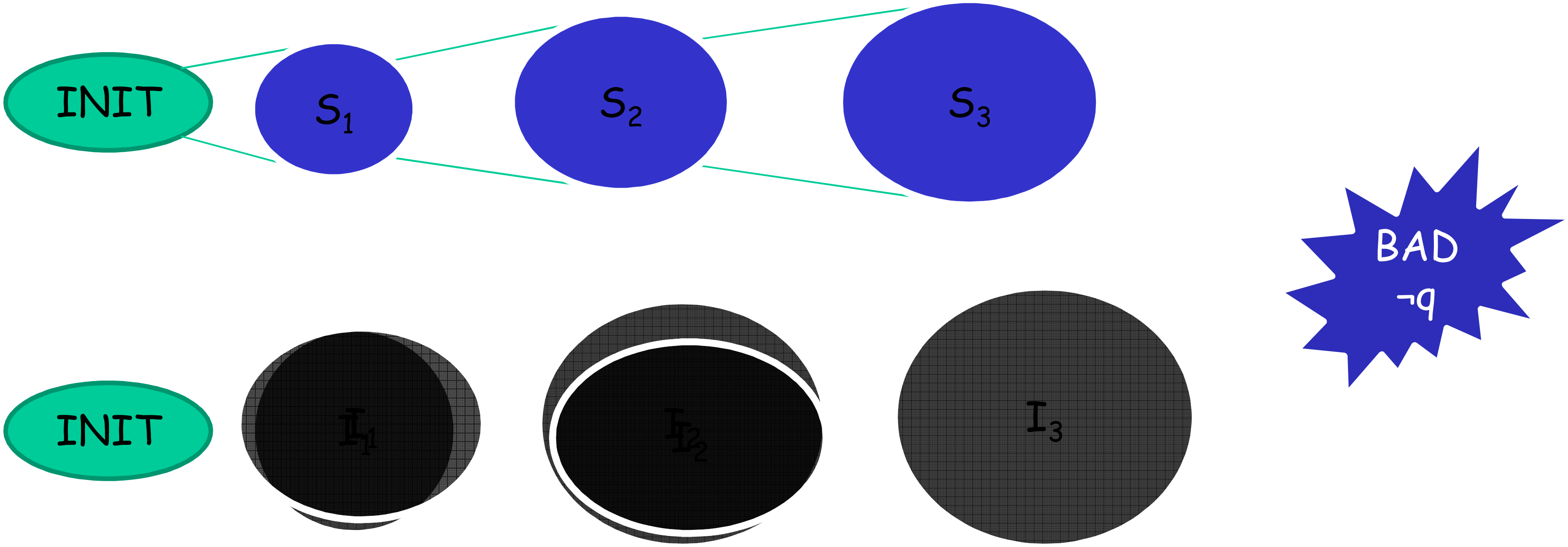
Computing Reachable States with a SAT Solver

- Use BMC to search for bugs.
- Partition the checked BMC formula and extract the interpolation sequence

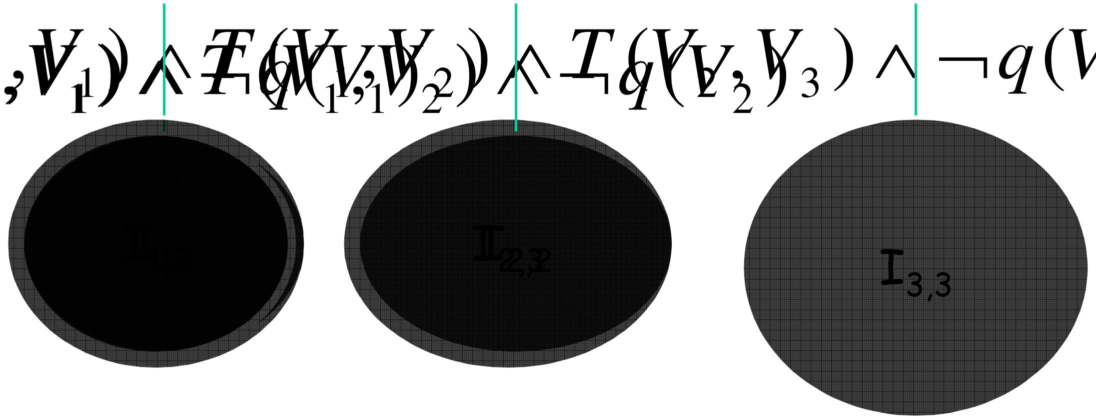


$$INIT(V_0) \wedge T(V_0,V_1) \wedge T(V_1,V_2) \wedge \ldots\ldots \wedge T(V_{N-1},T_N) \wedge \neg q(V_N)$$

The Analogy to Forward Reachability Analysis



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



Model Checking:
From BDDs to Interpolation

Lecture 3

Orna Grumberg
Technion
Haifa, Israel

Summer school at Bayrischzell 2011

Verification with SAT solvers

Combining Interpolation-Sequence and BMC

- Uses BMC for bug finding
- Uses Interpolation-sequence for computing over-approximation of sets S_j of reachable states
- Uses SAT solver for inclusion check for full verification

Combining Interpolation-Sequence and BMC

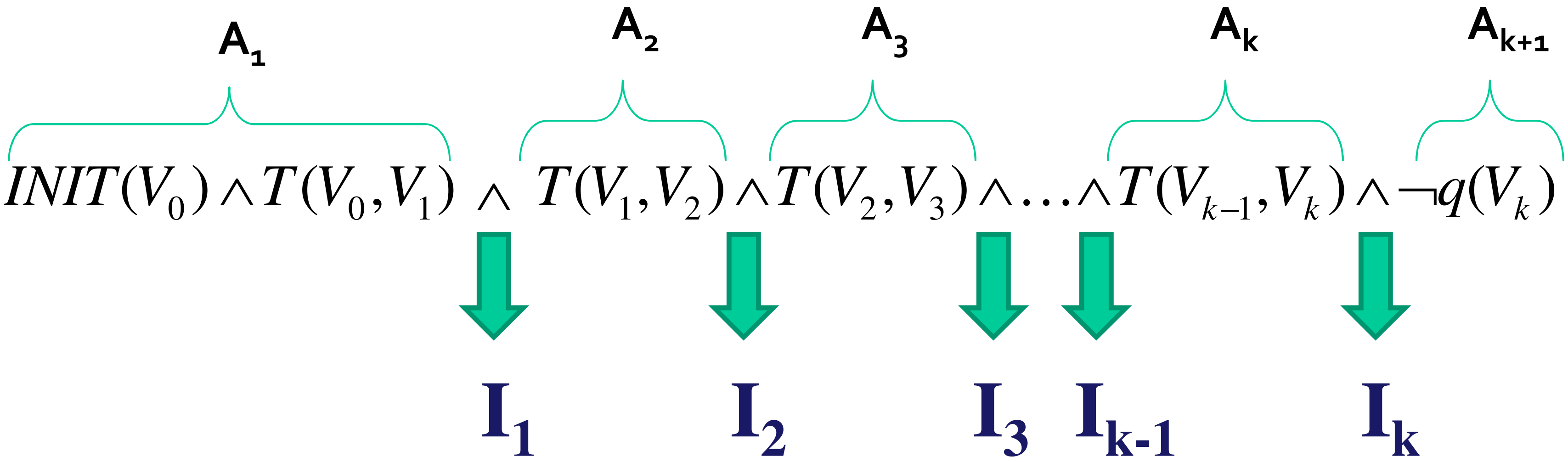
Always terminates

- either when BMC finds a bug:
  $M \mid\neq AGq$

- or when all reachable states has been found:
  $M \mid= AGq$

Interpolation-Sequence

- The same BMC formula partitioned in a different manner:



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

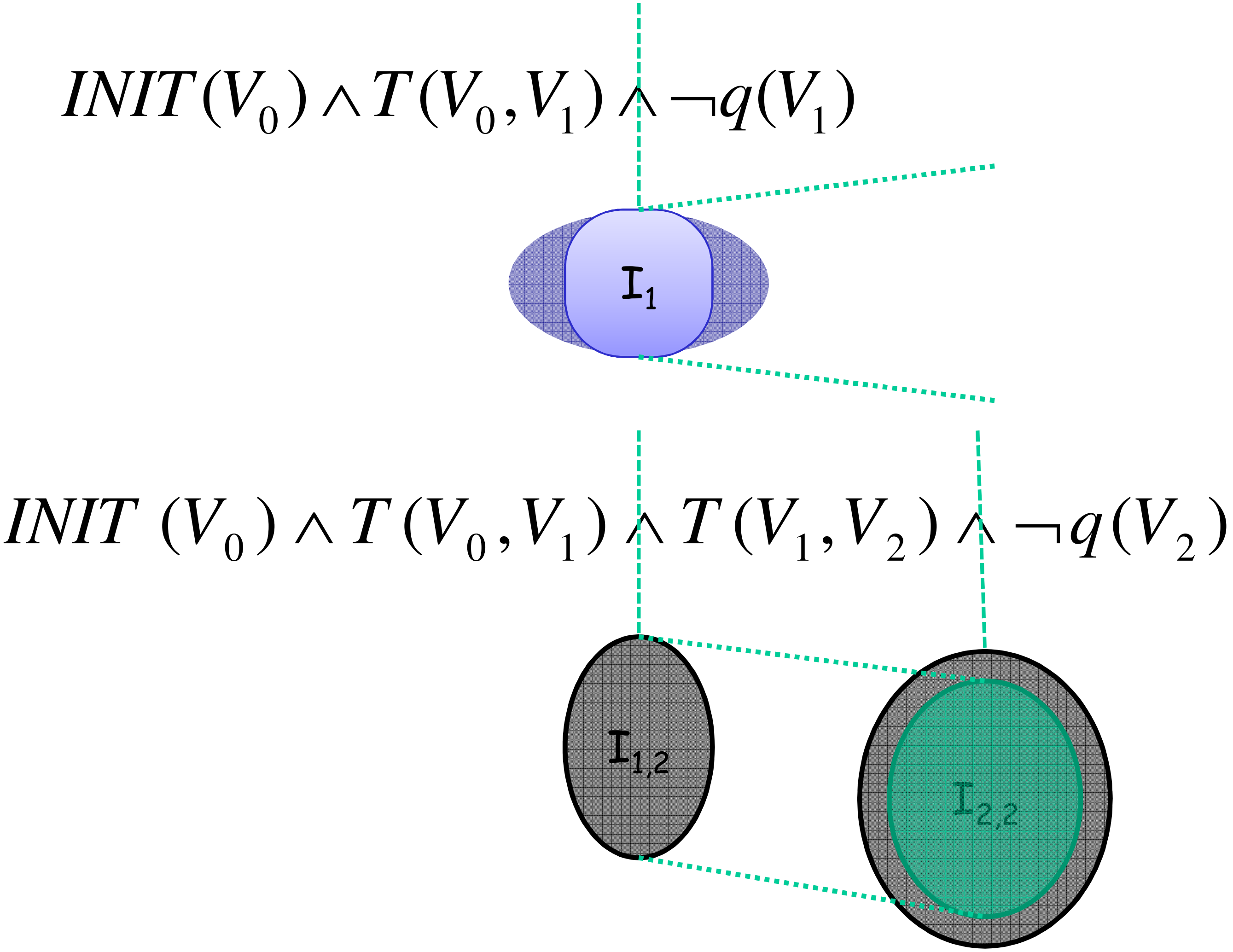
$\mathbf{I_1}$ $\mathbf{I_2}$ $\mathbf{I_3}$ $\mathbf{I_{k-1}}$ $\mathbf{I_k}$

$$I_0 = T, I_{k+1} = F$$

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

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

Using Interpolation-Sequence



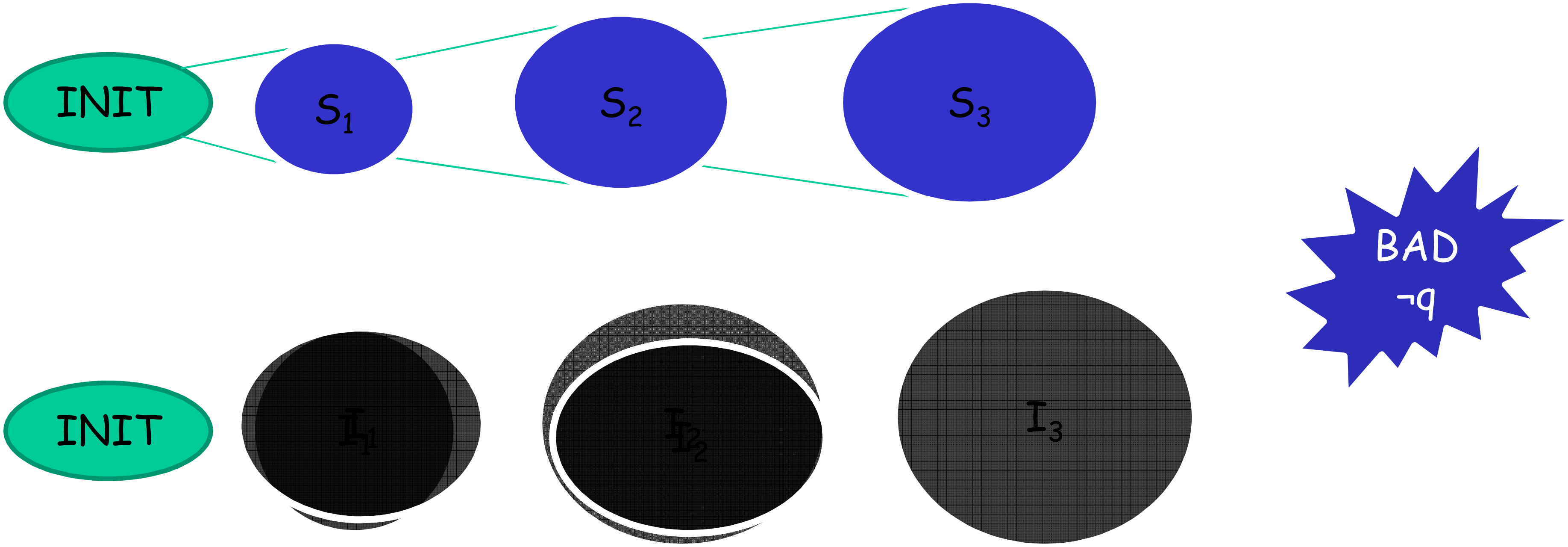
$$INIT(V_0) \wedge T(V_0,V_1) \wedge \neg q(V_1)$$

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

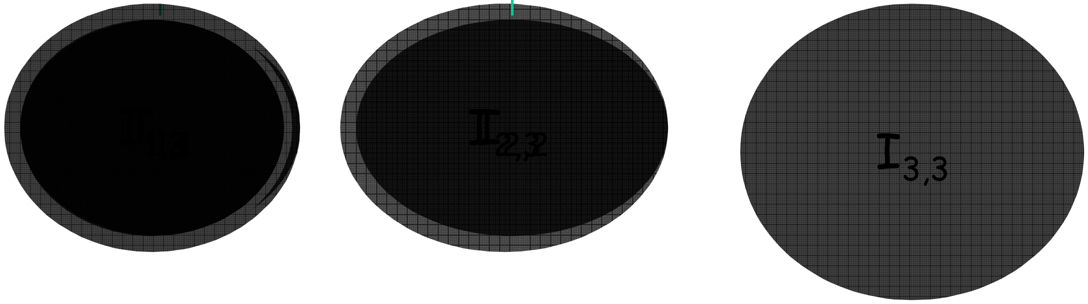
Checking if a "fixpoint" has been reached

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



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



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:
  $\hat{I} = (I_1, I_2, \dots , I_N)$

```
function UpdateReachable( $\hat{I}$, $\hat{I}^k$ )
    j=1
    while (j < k ) do
        $I_j = I_j \wedge I_j^k$
        $\hat{I}[j] = I_j$
    end while
    $\hat{I}[k] = I_k^k$
end function
```

```
function FixpointReached ($\hat{I}$) // check $I_j \Rightarrow \bigvee_{k=1,j-1} I_k$
   j=2
   while (j $\leq$ $\hat{I}$.length) do
      R = $\bigvee_{k=1,j-1} I_k$
      $\alpha = I_j \wedge \neg R$  // negation of $I_j \Rightarrow R$
      if (SAT($\alpha$)==false) then return true
      end if
      j = j+1
   end while
   return false
end function
```

```
Function ISB(M, f)    // f = AGq
    k = 0
    result = BMC (M, f, 0)
    if (result == cex) then return cex
    $\hat{I}$ = $\phi$  // the reachability vector
    while (true) do
        k = k+1
        result = BMC (M, f, k)
        if (result==cex) then return cex
        $\hat{I}^k = ( T, I_1^k, \ldots , I_k^k, F )$
        UpdateReachable ($\hat{I}$, $\hat{I}^k$)
        if ( FixpointReached ($\hat{I}$) == true) then
            return true
        end if
    end while
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

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

$$\Big\Downarrow$$

I

$$A\Rightarrow I$$

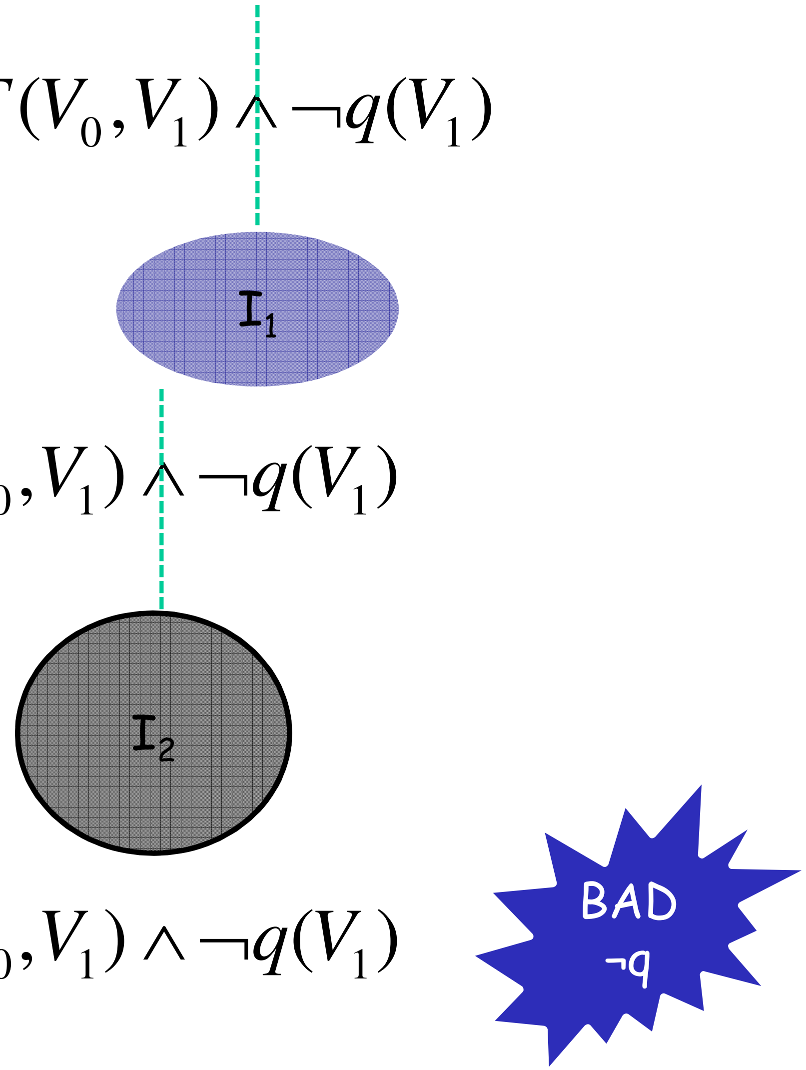
$$I\wedge B\equiv F$$

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

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



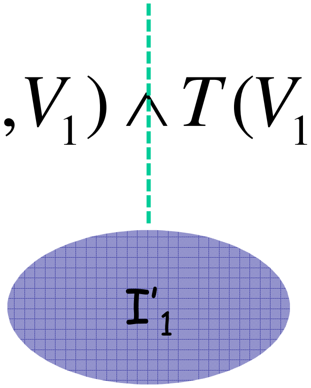
$$INIT(V_0) \wedge T(V_0, V_1) \wedge \neg q(V_1)$$

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

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

Using Interpolation (2)

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

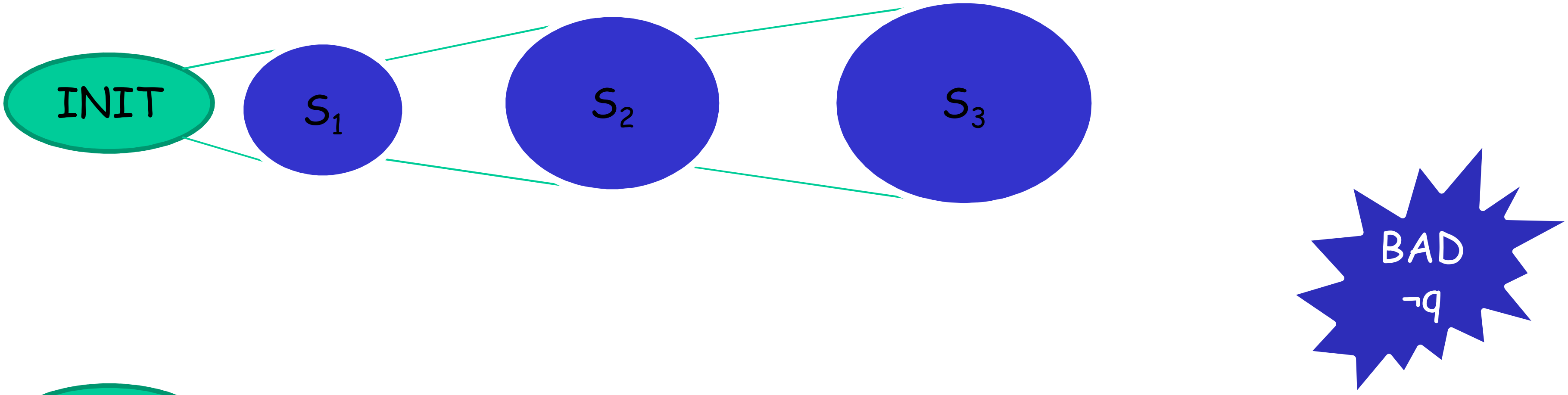


$$I_1'(V_0) \wedge T(V_0,V_1) \wedge T(V_1,V_2) \wedge (\neg q(V_1) \vee \neg q(V_2))$$

$$\vdots$$

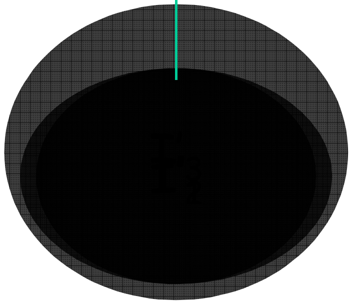
$$I_k'(V_0) \wedge T(V_0,V_1) \wedge T(V_1,V_2) \wedge (\neg q(V_1) \vee \neg q(V_2))$$

The Analogy to Forward Reachability Analysis



INIT

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



Charectaristics

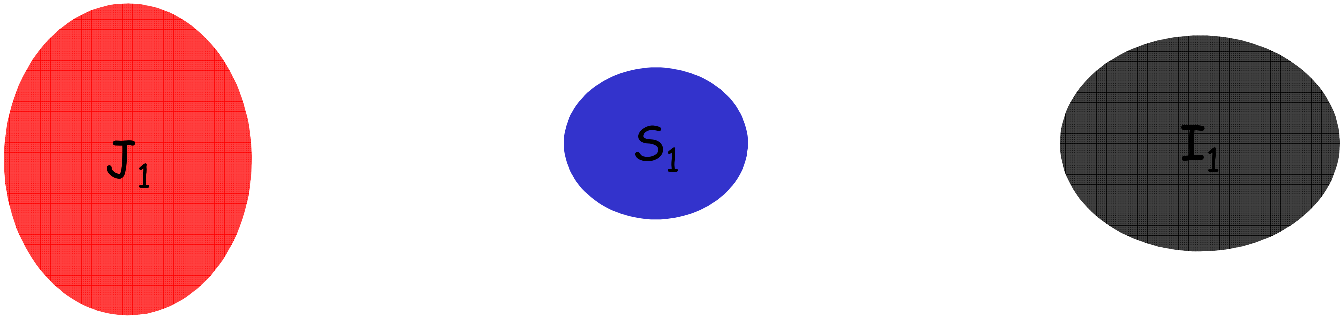
- When calculating the interpolant for the i-th iteration, for bound *k* the following holds:
  - The interpolant represents an over-approximation of reachable states after *i* transitions.
  - Also, it cannot reach a bad state in *k-1+i* steps or less.
    - It is similar to I_i calculated in ISB after *k+i* iterations.

Algorithm

```
Check the INIT states.
N = 1
Reachable = INIT
While (true)
    while ( BMC(M,f,Reachable,1,N) == false )
        I = getInterpolant();
        if ( I $\Rightarrow$ Reachable )
            return true;
        else
            Reachable = Reachable $\vee$ I;
    if (Reachable == INIT)
        return false;
    else
        N++;
```

McMillan’s Method

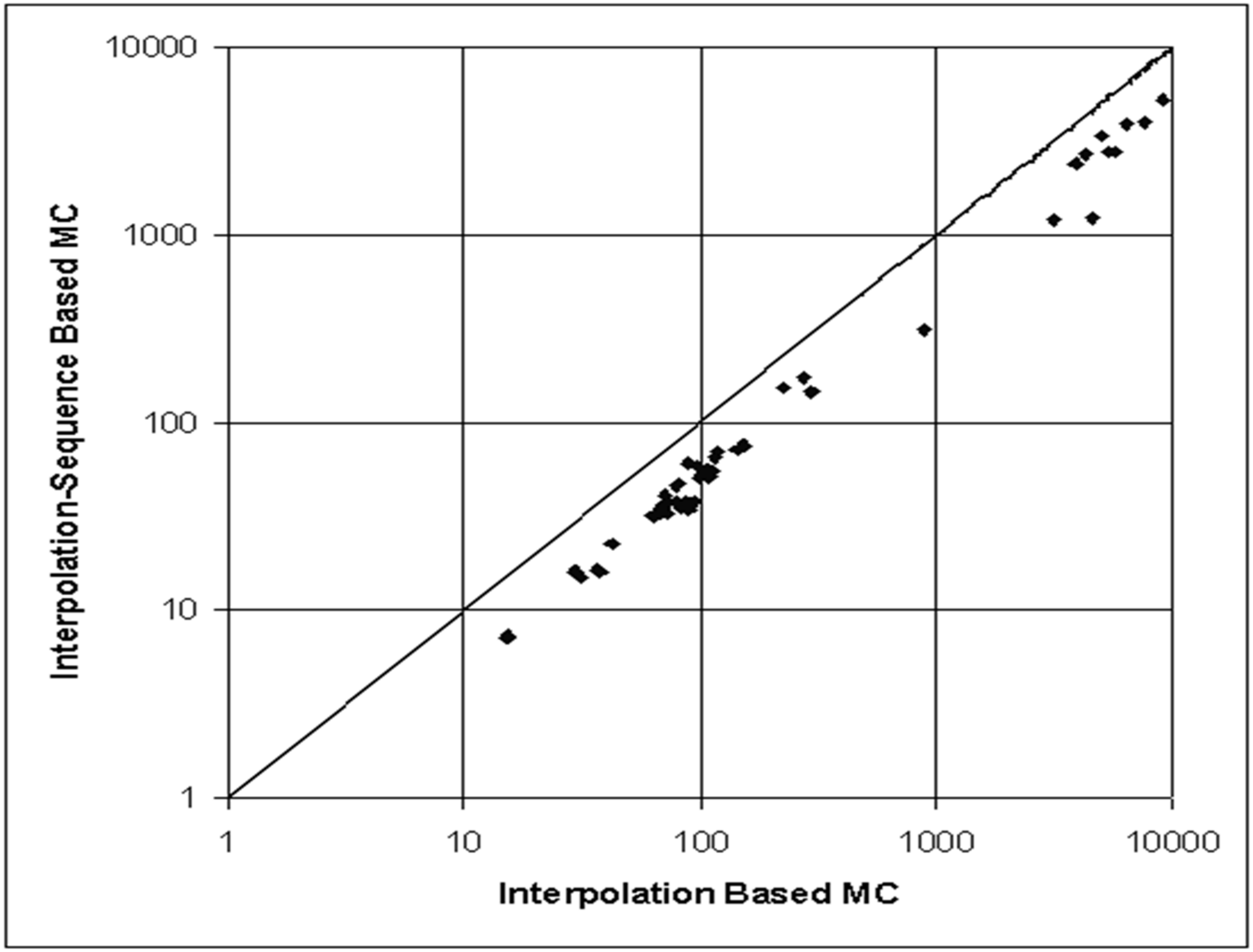
- The computation itself is different.
  - Uses basic interpolation.
  - Successive calls to BMC for the same bound.
  - Not incremental.
- The sets computed are different.



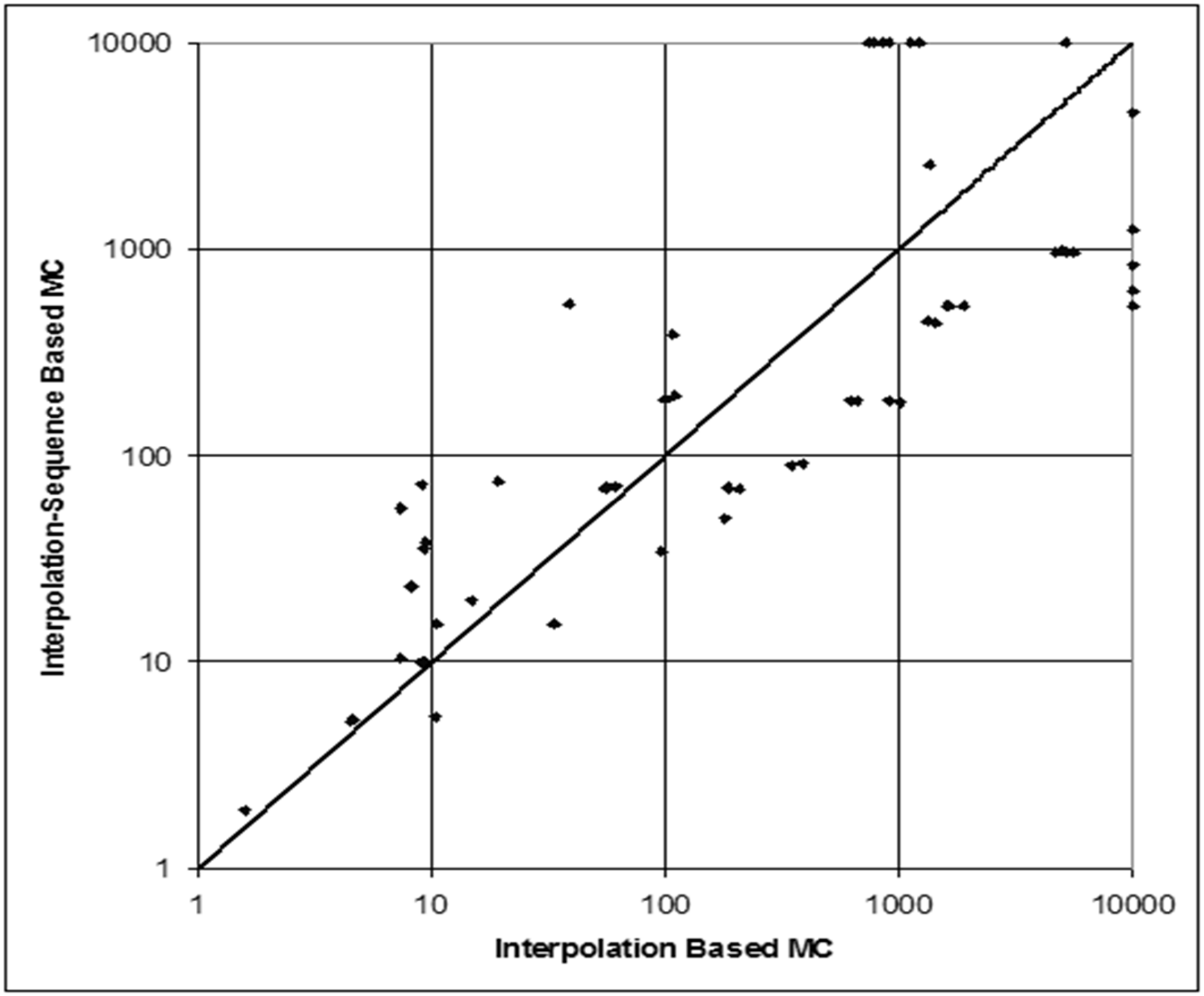
Experimental Results

- Experiments were conducted on two future CPU designs from Intel (two different architectures)

Experimental Results - Falsification



Experimental Results - Verification



Experiments Results - Analysis

Spec	#Vars	Bound (Ours)	Bound (M)	#Int (Ours)	#Int (M)	#BMC (Ours)	#BMC (M)	Time [s] (Ours)	Time [s] (M)
F_1	3406	**16**	15	**136**	80	**16**	80	**970**	5518
F_2	1753	**9**	8	**45**	40	**9**	40	**91**	388
F_3	1753	**16**	15	**136**	94	**16**	94	**473**	1901
F_4	3406	**6**	5	**21**	13	**6**	13	**68**	208
F_5	1761	**2**	1	**3**	2	**2**	2	**5**	4
F_6	3972	**3**	1	**6**	3	**3**	3	**19**	14
F_7	2197	**3**	1	**6**	3	**3**	3	**2544**	1340
F_8	4894	**5**	1	**15**	3	**5**	3	**635**	101

Analysis

- False properties is always faster.
- True properties – results vary. Heavier properties favor ISB where the easier favor IB.
- Some properties cannot be verified by one method but can be verified by the other and vise-versa.

Conclusions

- 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.

End of lecture 3

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

- **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

- 3-Valued BMC:
  A. Yadgar, A. Flaisher, O. Grumberg, and M. Lifshits, High Capacity (Bounded) Model Checking Using 3-Valued Abstraction

- A. Yadgar, New Approaches to Model Checking and to 3-valued abstraction and Refinement, Ph.d. Thesis, Technion, March 2010

Interpolation based model checking:

- K. McMillan, Interpolation and SAT-Based Model Checking, CAV'03
- *T.* Henzinger, R. Jhala, R. Majumdar, K. McMillan, *A*bstractions from Proofs, POPL'04
- Y. Vizel and O. Grumberg, Interpolation-Sequence Based Model Checking, FMCAD'09

Exercise 1

Write 2 CTL formulas.

1. f_1 is true in a state iff
   the state is the start of a path along
   which p holds at least twice

2. f_2 is true in a state iff
   the state is the start of a path along
   which p holds exactly twice