

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

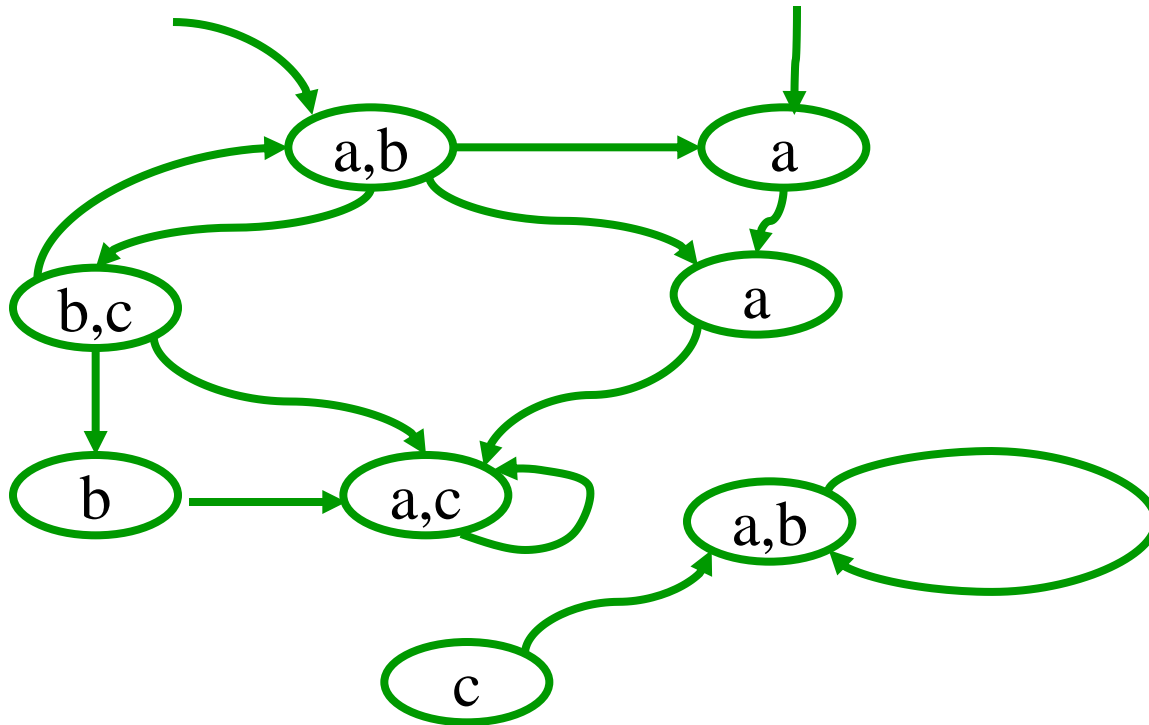
no + Counterexample, otherwise

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

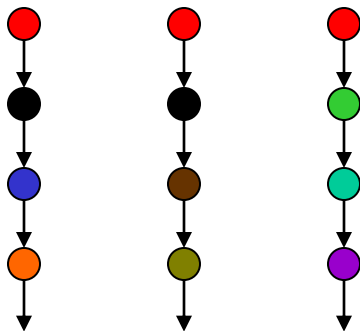


Temporal Logics

- Temporal Logics
 - Express properties of event orderings in time

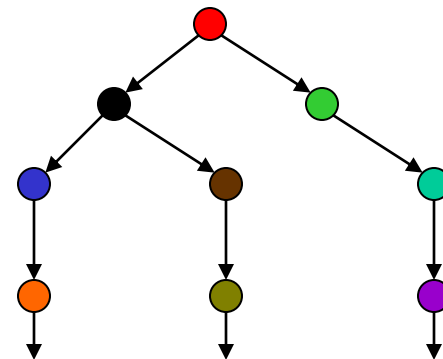
- Linear Time

- 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)

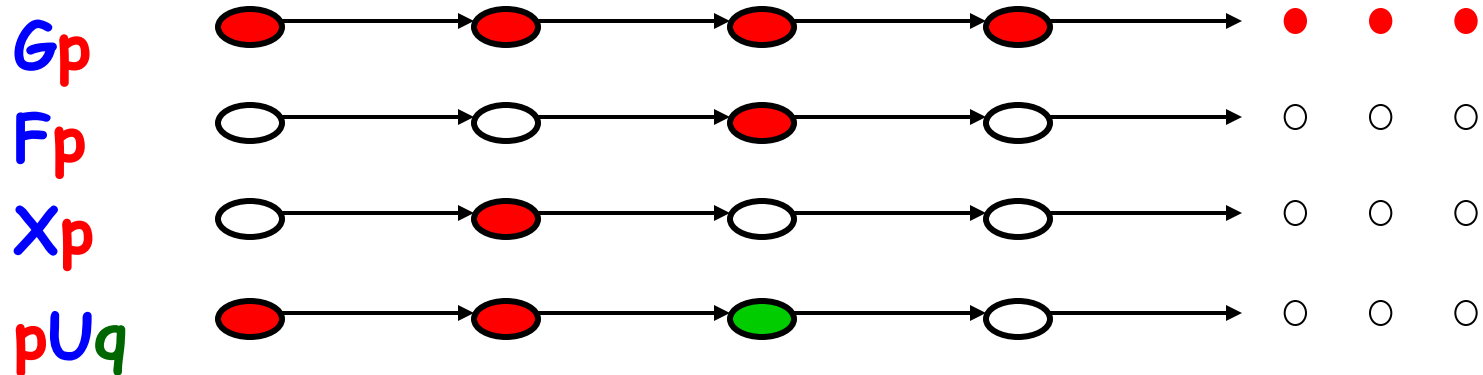


Propositional temporal logic

In Negation Normal Form

AP - a set of atomic propositions

Temporal operators:



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: $AG(\text{request} \Rightarrow AF \text{grant})$
- "sanity" check: $EF \text{request}$

Model checking

A basic operation: **Image computation**

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

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

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 $M = \langle S, I, R, L \rangle$
and a set S_p of states satisfying q in M

procedure **CheckAG** (S_p)

Reach = \emptyset

$S_0 = I$

$k = 0$

while $S_k \not\subseteq \text{Reach}$ do

 If $S_k \cap S_p \neq \emptyset$ return ($M \models \text{AG}q$)

$S_{k+1} = \text{Image}(S_k)$

 Reach = Reach $\cup S_k$

$k = k+1$

end while

return(**Reach**, $M \models \text{AG}p$)

Model checking AGq

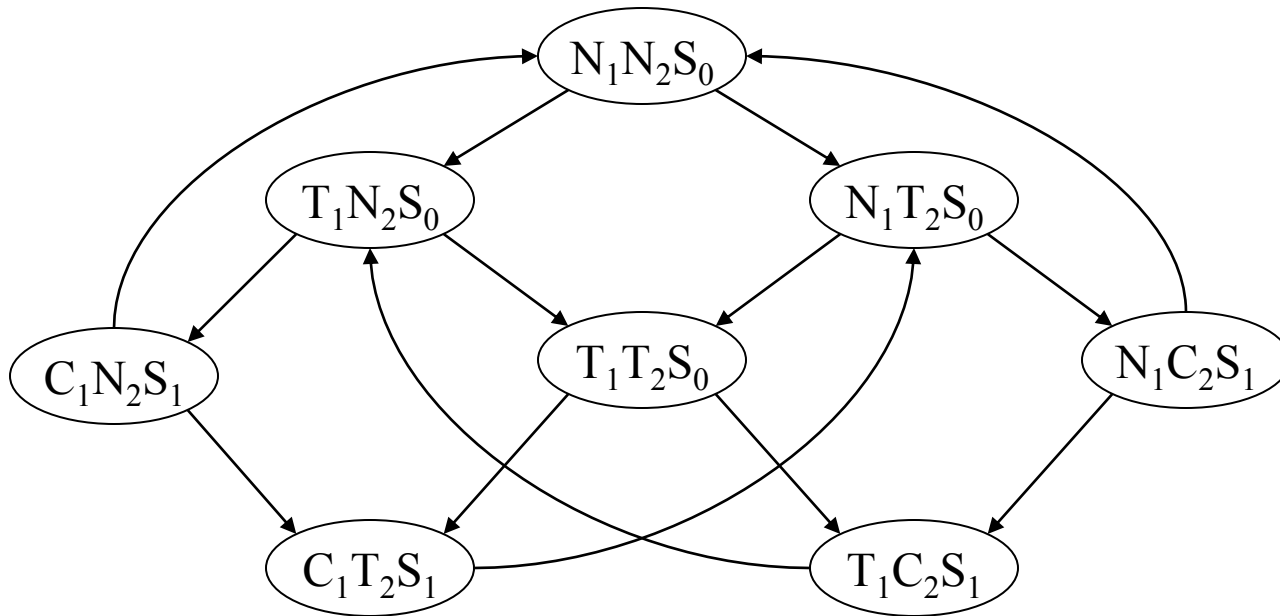
- Also called forward reachability analysis

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 --- $N_1 N_2 S_0$

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

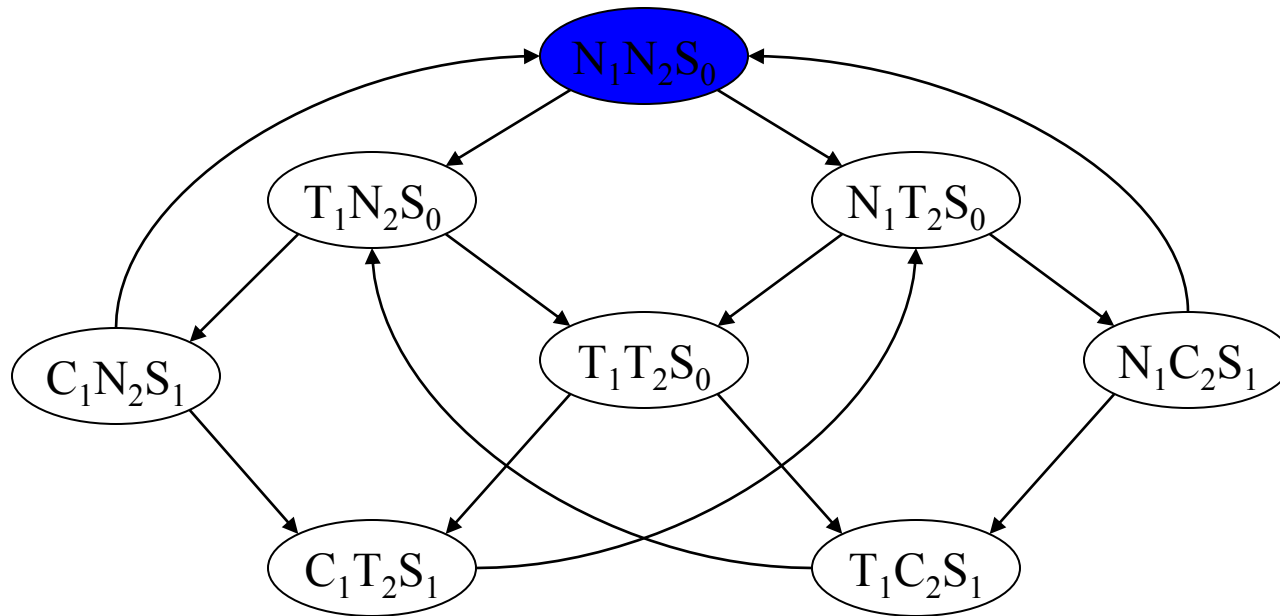
Mutual Exclusion Example



$$M \models \text{AG } \neg (C_1 \wedge C_2)$$

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

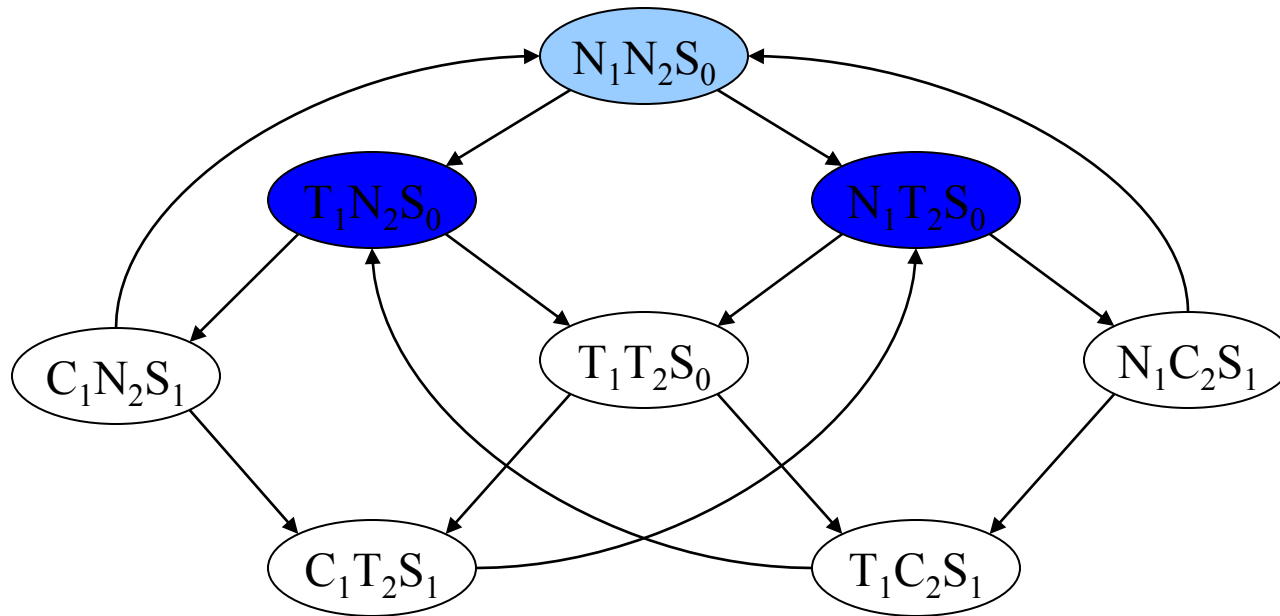
Mutual Exclusion Example



$$M \models \text{AG } \neg (C1 \wedge C2)$$

S_0

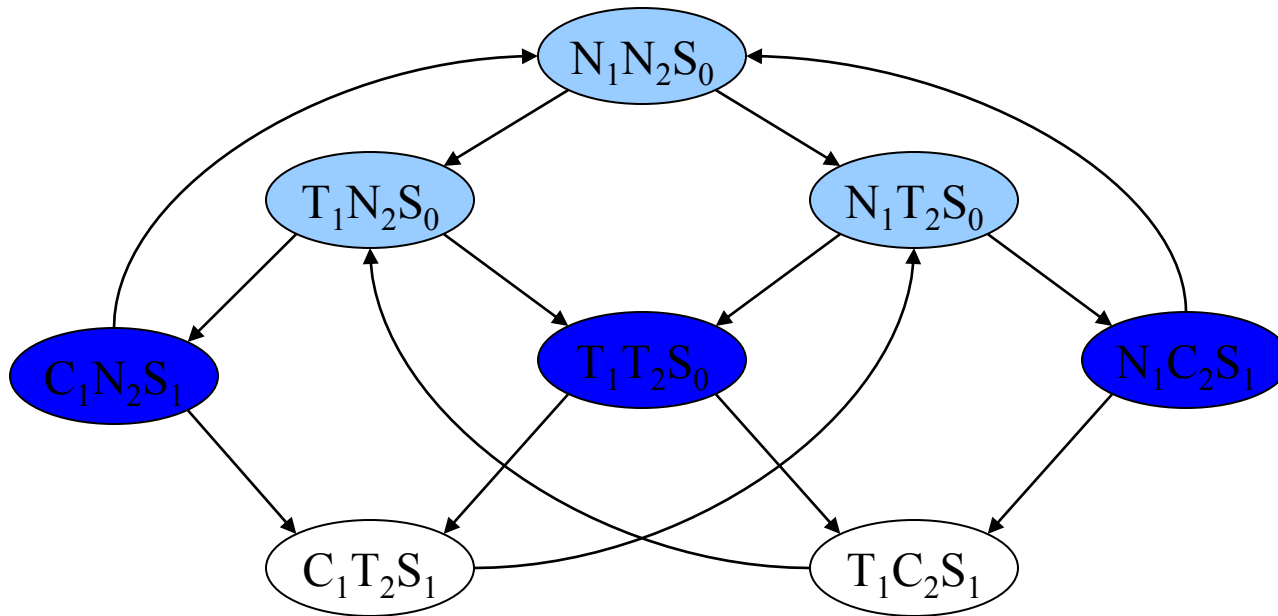
Mutual Exclusion Example



$$M \not\models \text{AG } \neg (C1 \wedge C2)$$

S_1

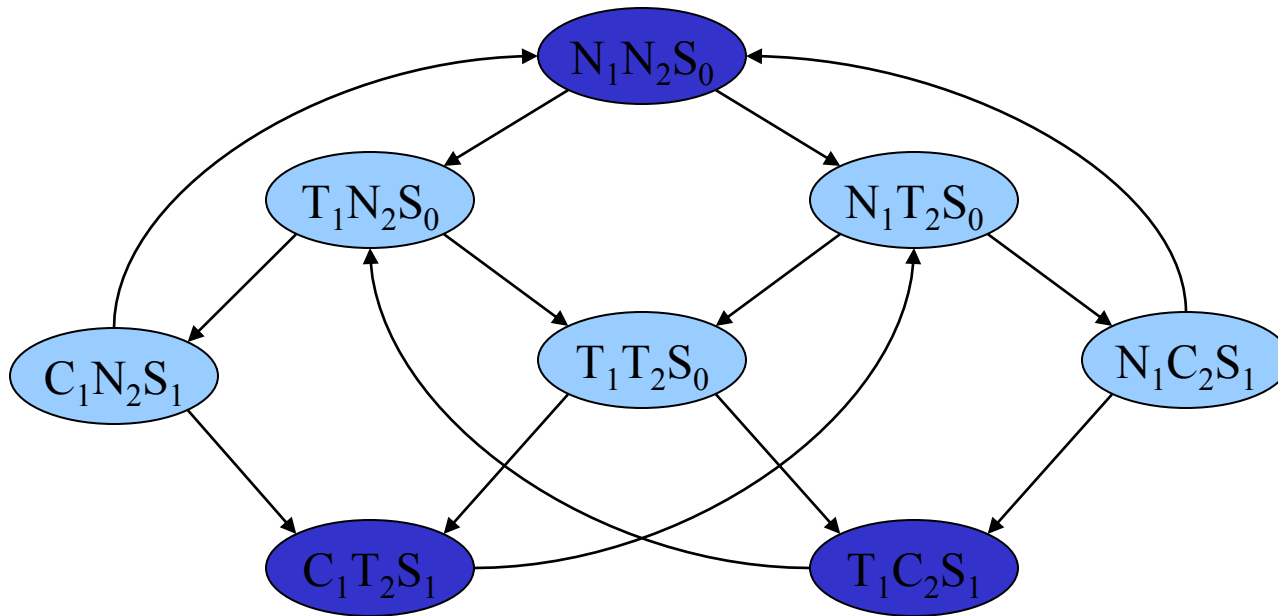
Mutual Exclusion Example



$$M \models \text{AG } \neg (C1 \wedge C2)$$

S_2

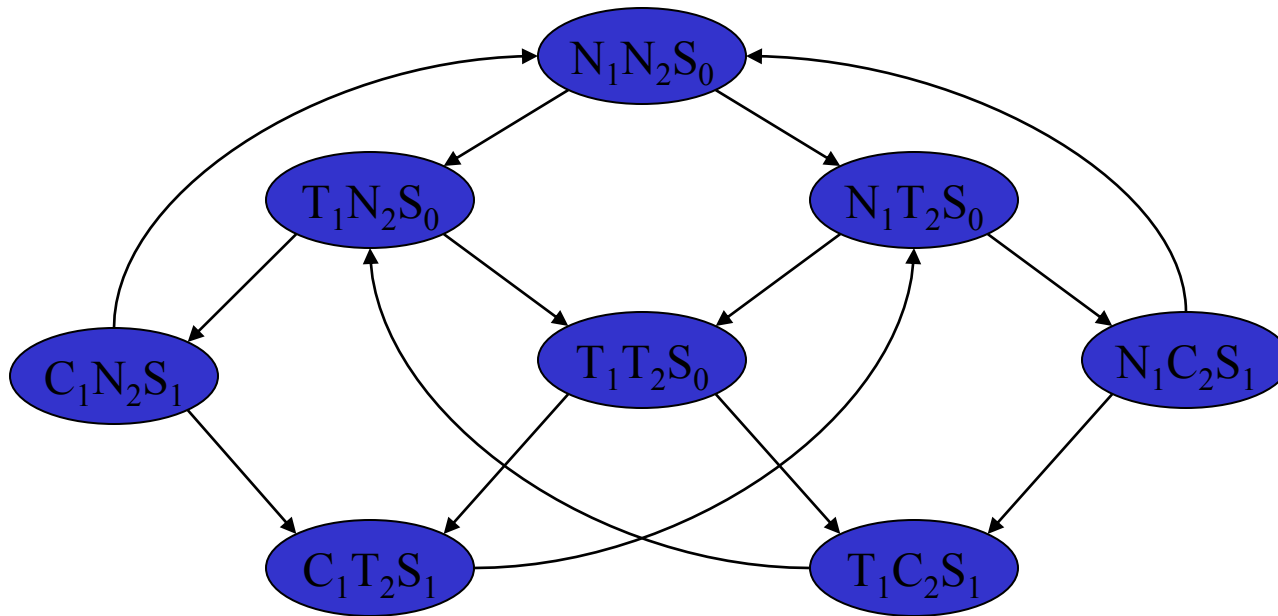
Mutual Exclusion Example



$$M \models \text{AG } \neg (C1 \wedge C2)$$

S_3

Mutual Exclusion Example



$$M \not\models \text{AG } \neg (C1 \wedge C2)$$

$$S_4 \subseteq S_0 \cup \dots \cup S_3$$

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 $v_1 \dots v_n$
- **Reach**, S_k can be represented by BDDs over $v_1 \dots v_n$
- R (a set of pairs of states **(s, s')**) can be represented by a BDD over $v_1 \dots v_n \ v_1' \dots v_n'$

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:

$$s_1: v_1v_2=00 \quad s_2: v_1v_2=01 \quad s_3: v_1v_2=11$$

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

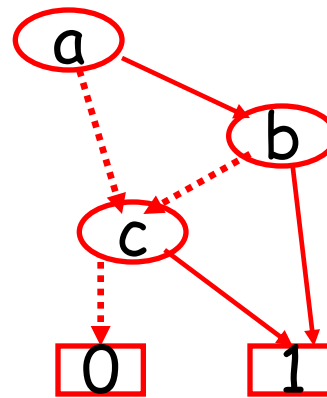
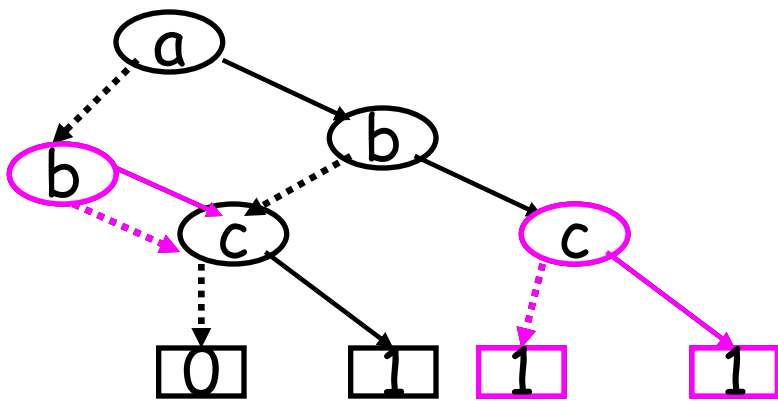
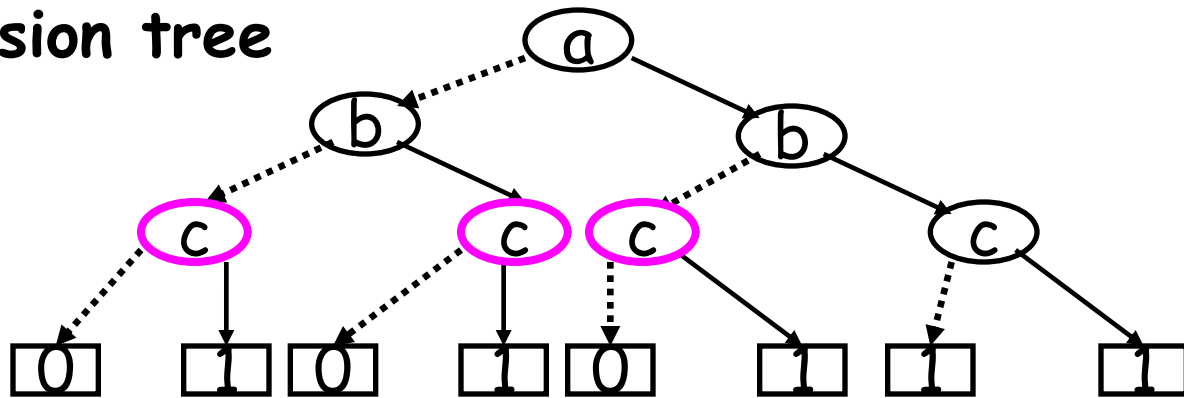
$$f_A(v_1, v_2) = (\neg v_1 \wedge \neg v_2) \vee (\neg v_1 \wedge v_2) = \neg v_1$$

$$\begin{aligned} f_R(v_1, v_2, v'_1, v'_2) = & \\ & (\neg v_1 \wedge \neg v_2 \wedge \neg v'_1 \wedge v'_2) \vee \\ & (\neg v_1 \wedge v_2 \wedge \neg v'_1 \wedge v'_2) \vee \\ & (v_1 \wedge v_2 \wedge \neg v'_1 \wedge \neg v'_2) \end{aligned}$$

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 represented by formulas with a few millions 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 $f_{M,k}$ describing all possible computations of M of length k
- Construct a formula $f_{\varphi,k}$ expressing that $\varphi = \text{EF}\neg p$ holds within k computation steps
- Check whether $f = f_{M,k} \wedge f_{\varphi,k}$ is satisfiable


If f is satisfiable then $M \not\models \text{AG}p$

The satisfying assignment is a counterexample

Example - shift register

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

Transition relation:

$$R(x, y, z, x', y', z') = x' = y \wedge y' = z \wedge z' = 1$$


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

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

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

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

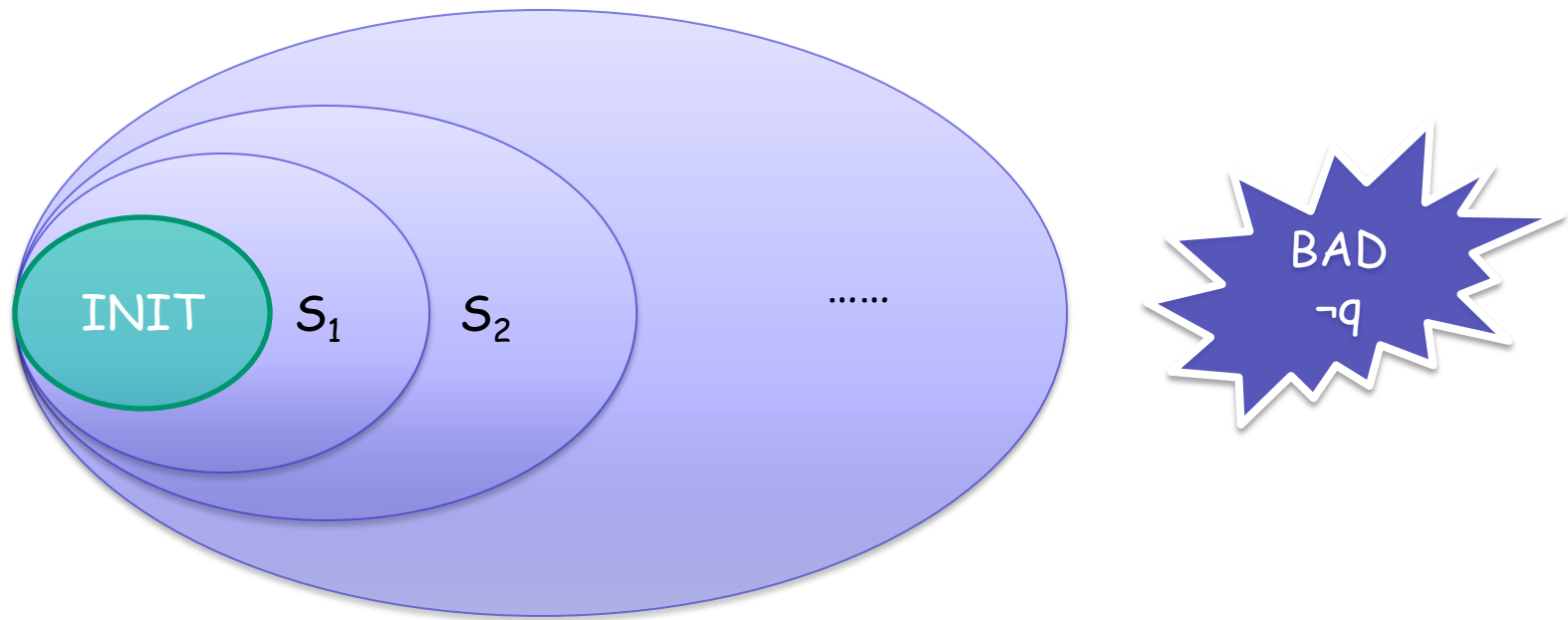
Combines:

- 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:

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

Bounded Model Checking

- Does the system have a counterexample of length k ?

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

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

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

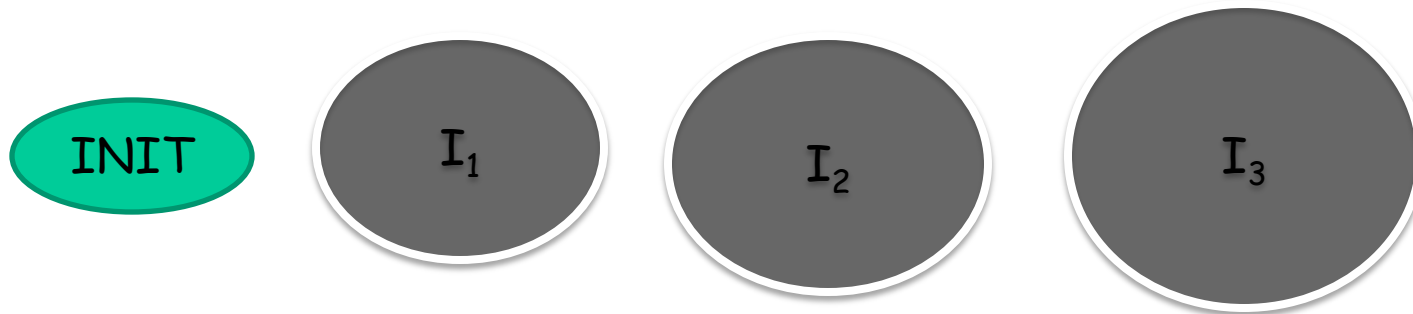
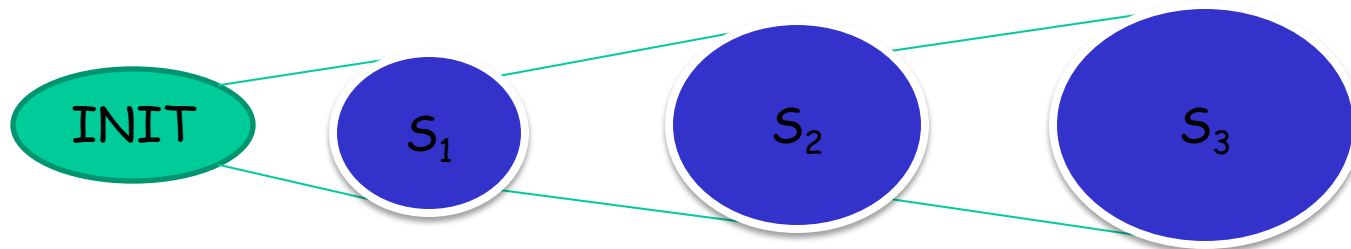
▪

▪

▪

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

A Bit of Intuition



Interpolation

(Craig,57)

- If $A \wedge B = \text{false}$, 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

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



I

$$A \Rightarrow I$$

$$I \wedge B \equiv F$$

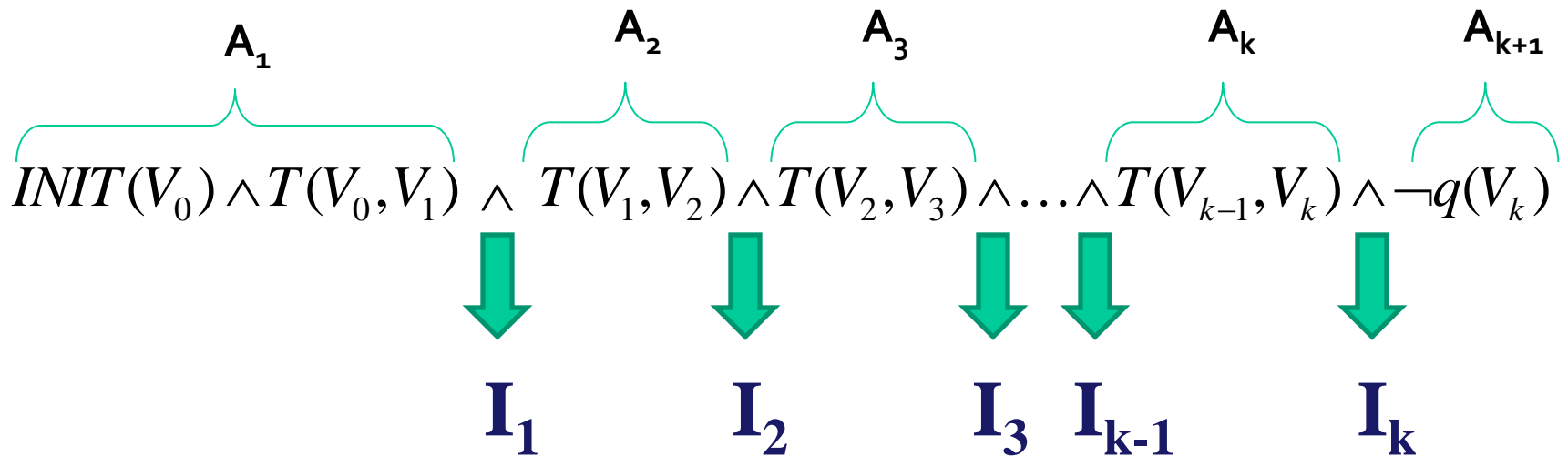
I is over the common variables of A and B, i.e V_1

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:



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

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

I_j is over the common variables of A_1, \dots, A_j and A_{j+1}, \dots, A_{k+1} , i.e. V_j

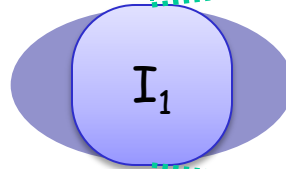
Interpolation-Sequence (2)

- 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)

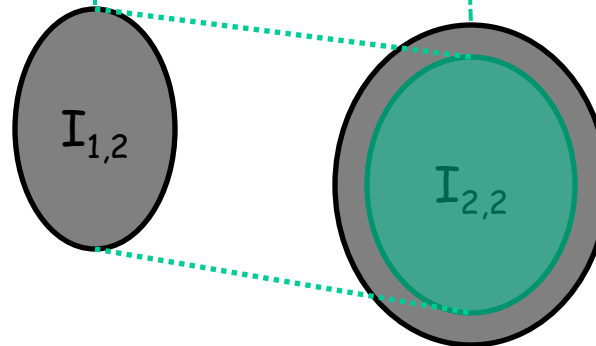
Interpolation-Sequence Based Model Checking

Using Interpolation-Sequence

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



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

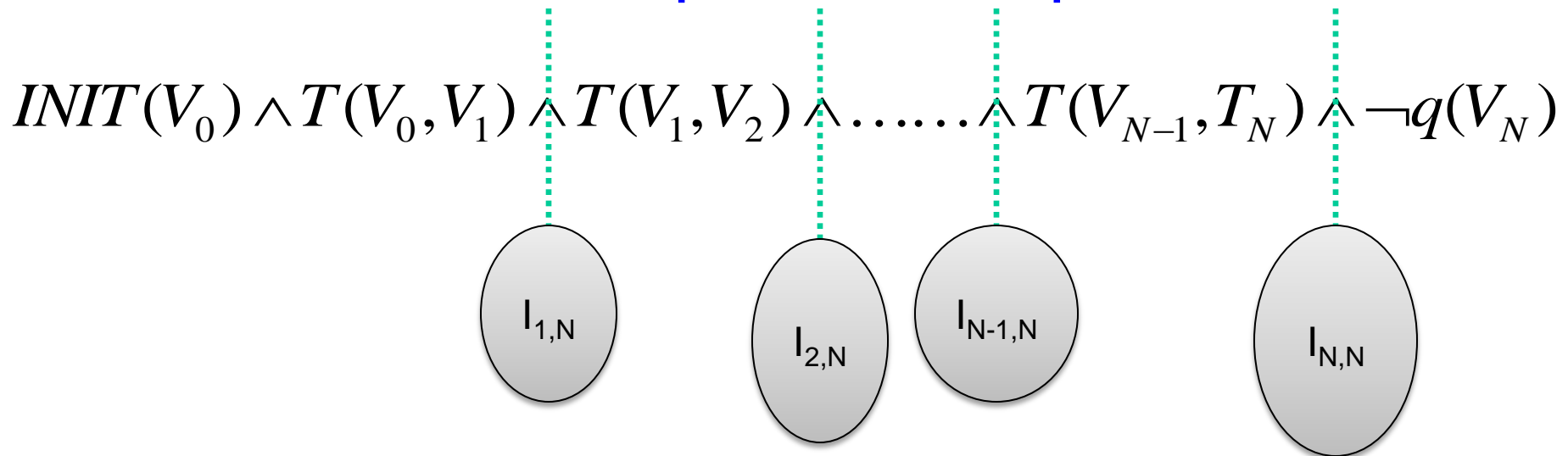


Combining Interpolation-Sequence and BMC

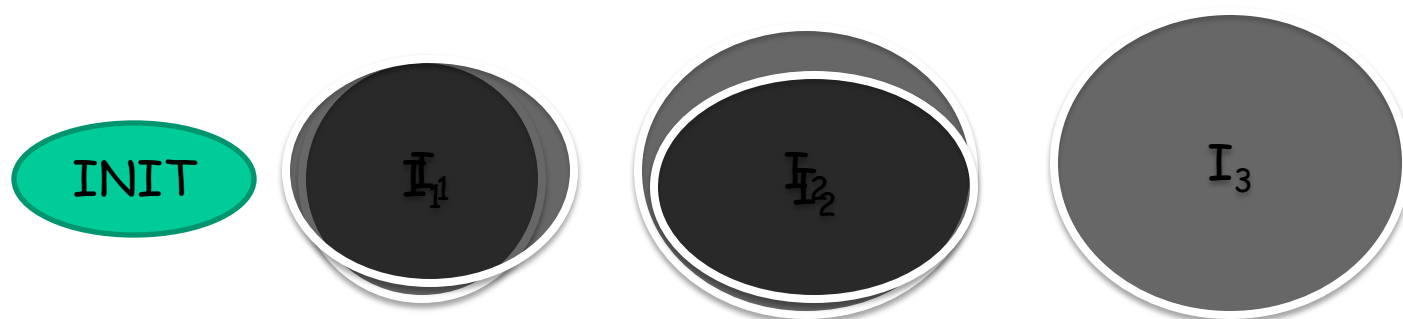
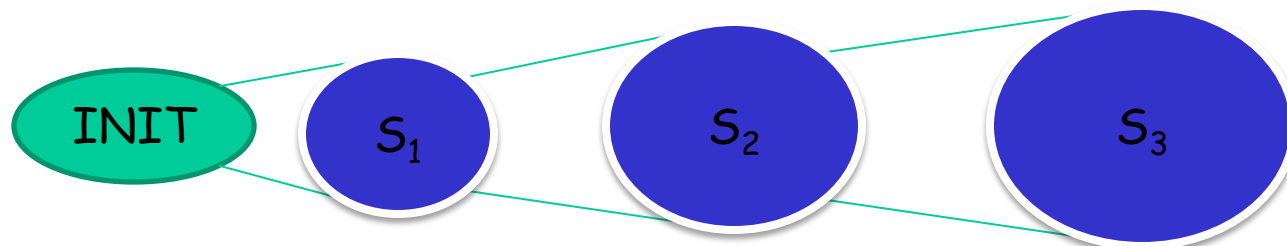
- A way to do reachability analysis using a SAT solver.
- Uses the original BMC loop and adds an inclusion check for full verification.
- Similar sets to those computed by Forward Reachability Analysis but over-approximated.

Computing Reachable States with a SAT Solver

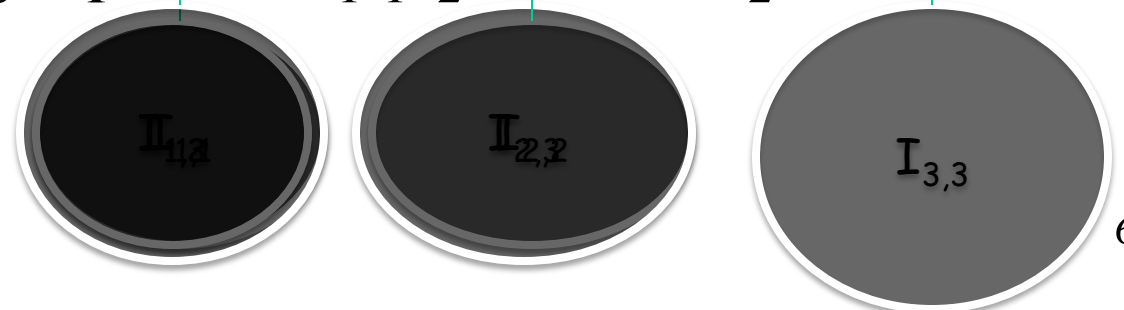
- Use BMC to search for bugs.
- Partition the checked BMC formula and extract the interpolation sequence



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

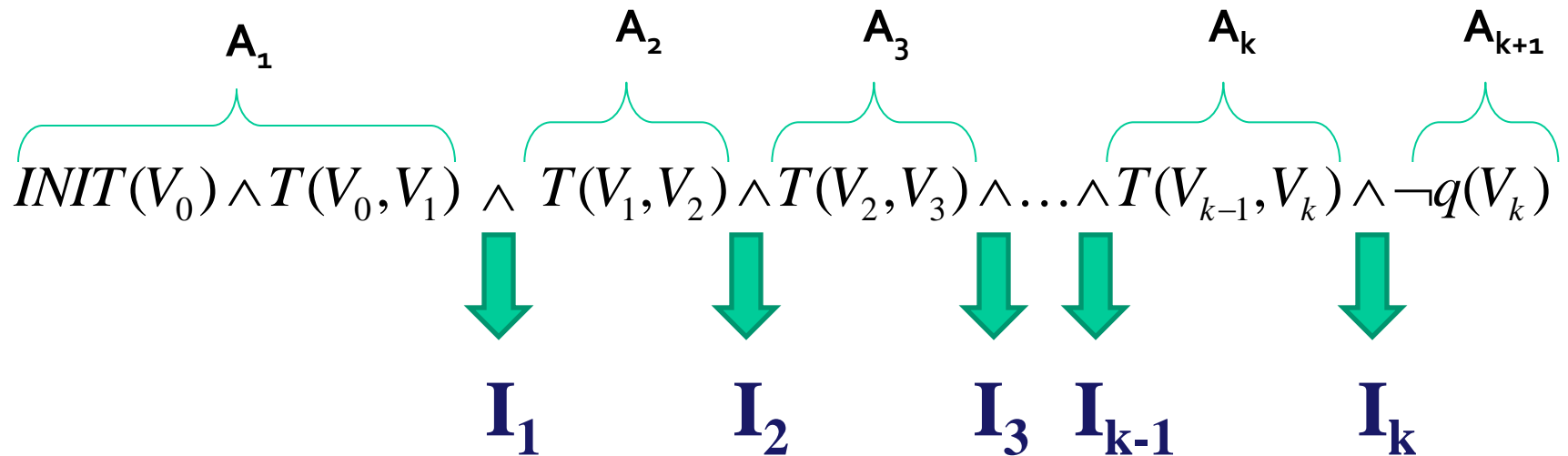
Combining Interpolation-Sequence and BMC

Always terminates

- either when BMC finds a bug:
 $M \not\models AGq$
- or when all reachable states has been found:
 $M \models AGq$

Interpolation-Sequence

- The same BMC formula partitioned in a different manner:



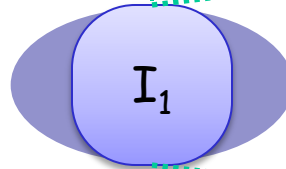
$$I_0 = T, I_{k+1} = F$$

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

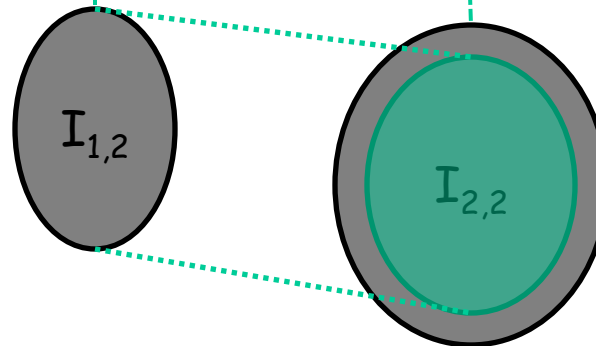
I_j is over the common variables of A_1, \dots, A_j and A_{j+1}, \dots, A_{k+1} , i.e. V_j

Using Interpolation-Sequence

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



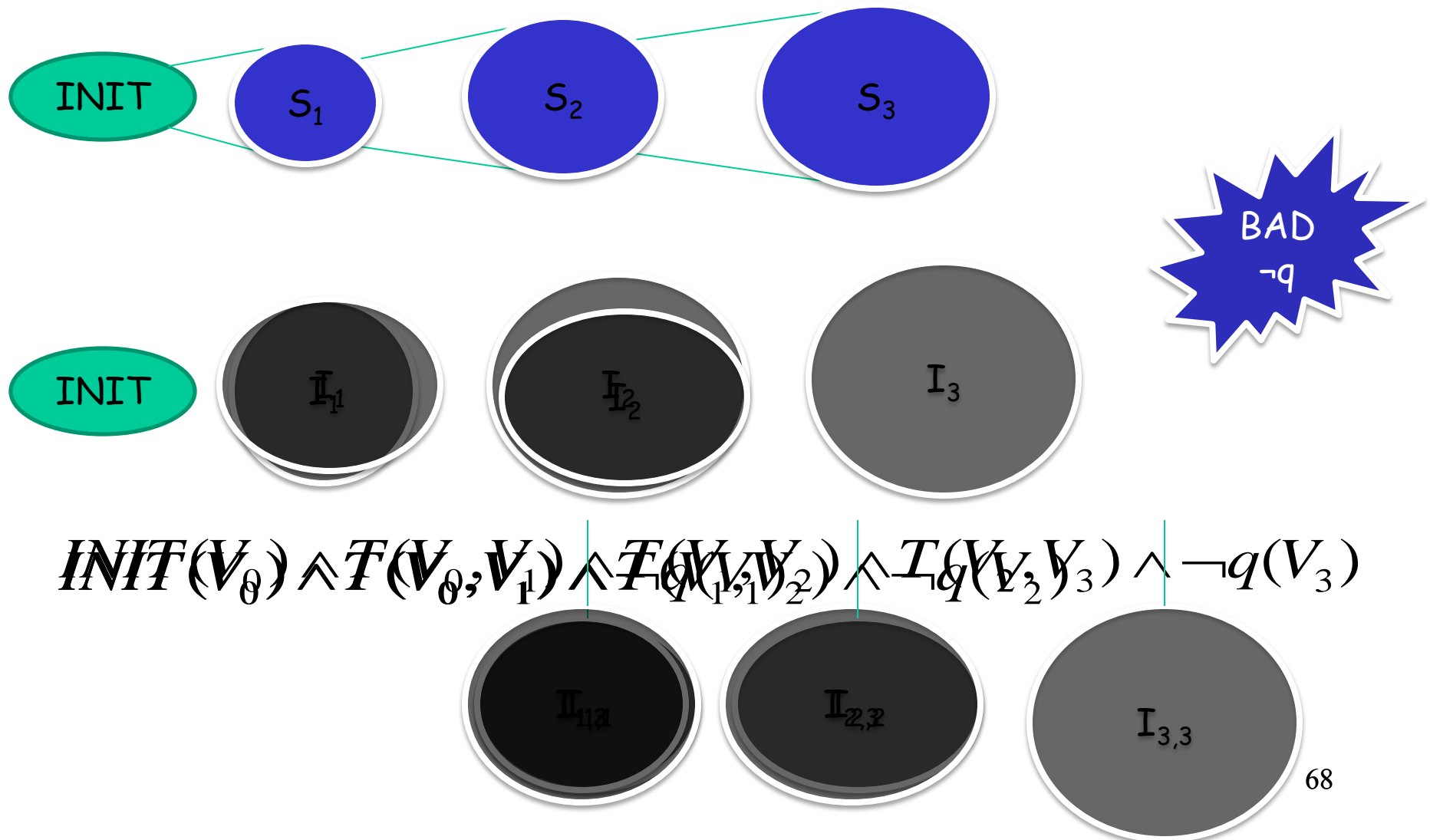
$$INIT(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge \neg q(V_2)$$



Checking if a “fixpoint” has been reached

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



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

```

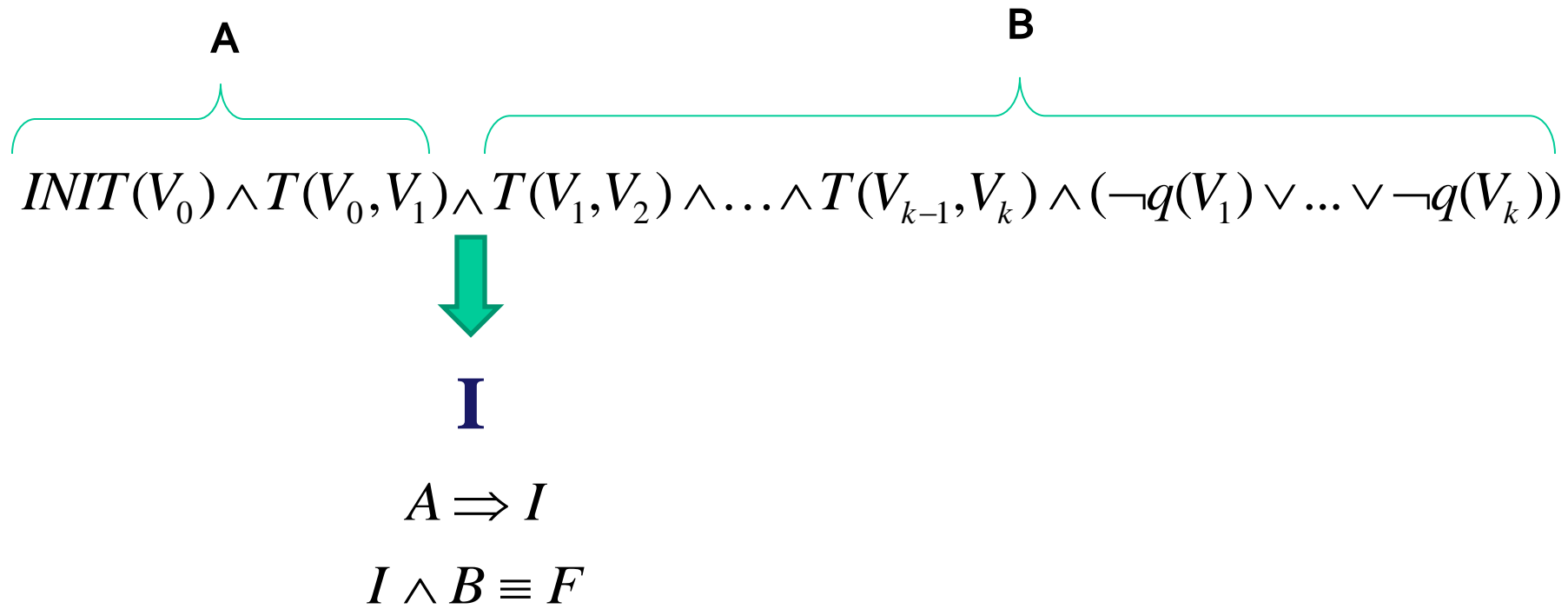
function FixpointReached ( $\hat{I}$ ) // check  $I_j \Rightarrow V_{k=1,j-1} I_k$ 
  j=2
  while (j  $\leq$   $\hat{I}$ .length) do
    R =  $V_{k=1,j-1} I_k$ 
     $\alpha = I_j \wedge \neg R$  // negation of  $I_j \Rightarrow R$ 
    if (SAT( $\alpha$ )==false) then return true
  end if
  j = j+1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

```

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**



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

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



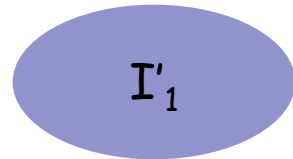
I_2

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



Using Interpolation (2)

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

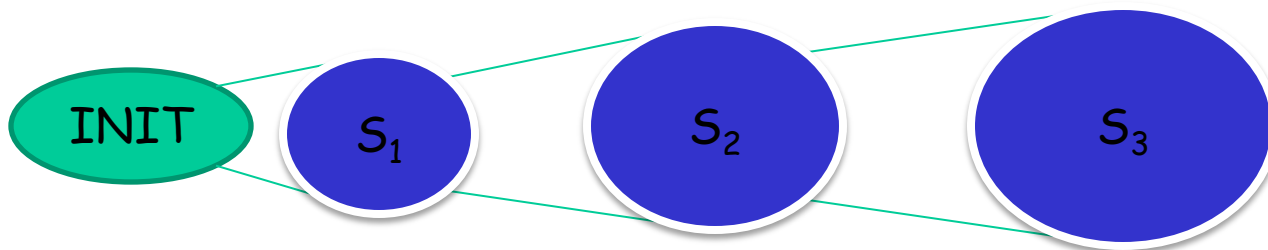


$$I'_1(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge (\neg q(V_1) \vee \neg q(V_2))$$

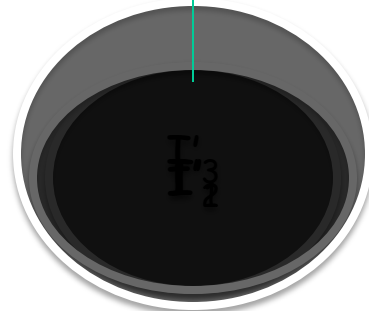
·
·
·

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

The Analogy to Forward Reachability Analysis



$$INIT_1(V_0) \wedge T(V_0, V_1) \wedge T(V_1, V_2) \wedge (\neg q(W_{11}) \wedge \neg q(W_{22}))$$

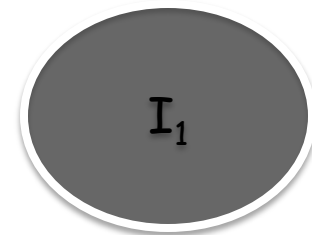
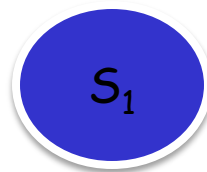
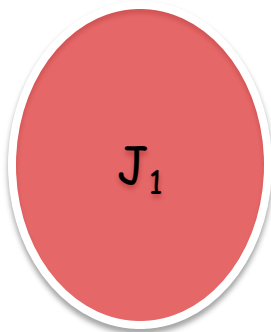


Characteristics

- 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 $k-1+i$ steps or less.
 - It is similar to I_i calculated in ISB after $k+i$ iterations.

McMillan's Method

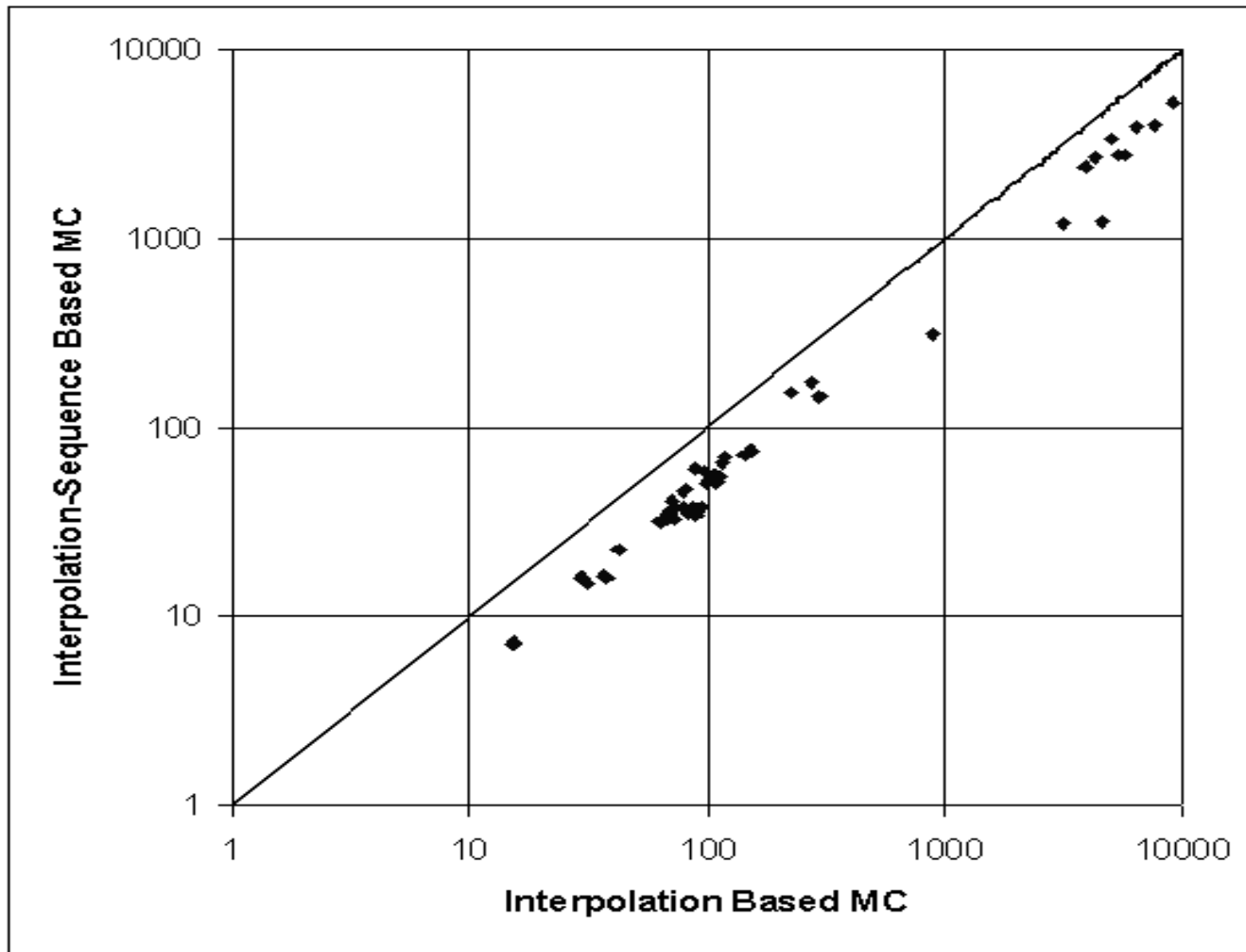
- The computation itself is different.
 - Uses basic interpolation.
 - Successive calls to BMC for the same bound.
 - Not incremental.
- The sets computed are different.



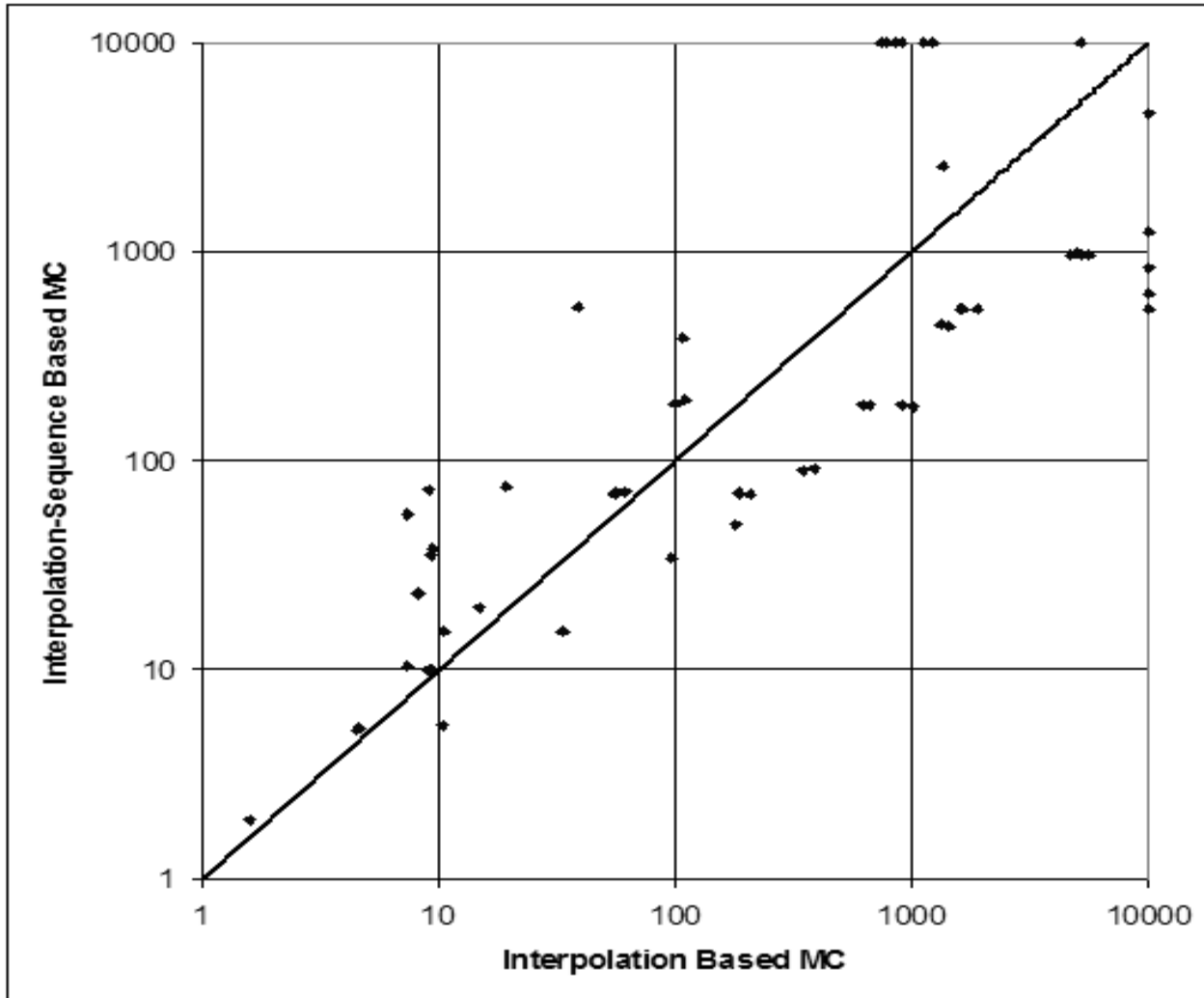
Experimental Results

- 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.

Thank You

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

- **Existential abstraction + data abstraction:**

E. M. Clarke, O. Grumberg, D. E. Long,
Model Checking and Abstraction,
TOPLAS, 1994.

- **Localization reduction:**

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

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

- **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

Model Checking: From BDDs to Interpolation

Lecture 4

Orna Grumberg
Technion
Haifa, Israel

Summer school at Bayrischzell 2011

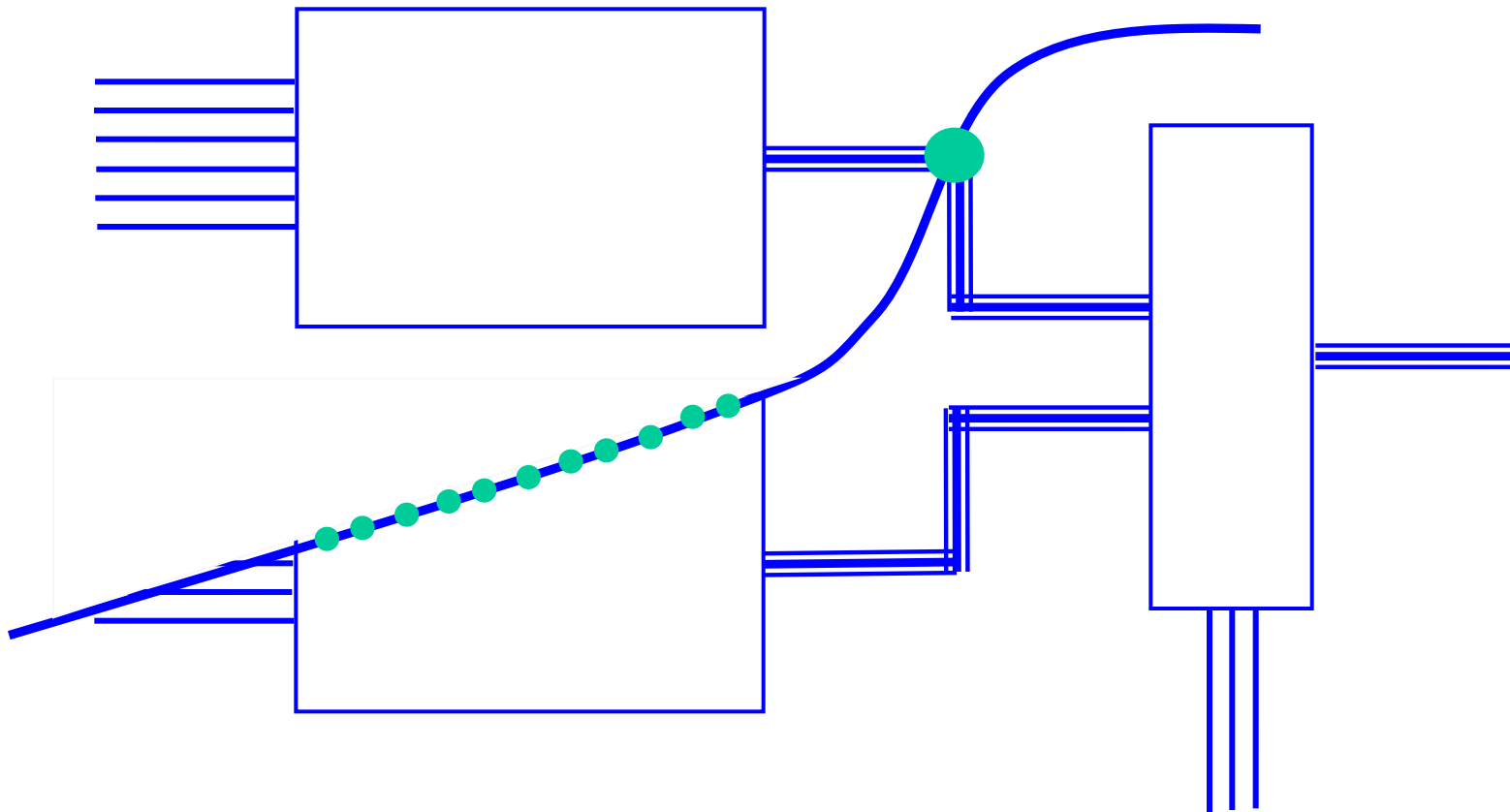
3-Valued Abstraction in (Bounded) Model Checking for Hardware

[Yadgar, Ph.d. thesis]

Motivation

- Increase capacity of (Bounded) Model Checking
 - By abstracting out parts of the model
- "Smart" abstraction
 - Automatic or manual
- "Easy" abstraction
 - Abstract out inputs or critical nodes
- Holy Grail: Change the level of BMC

Abstraction in Model Checking



Localization reduction

Over-approximating abstraction:

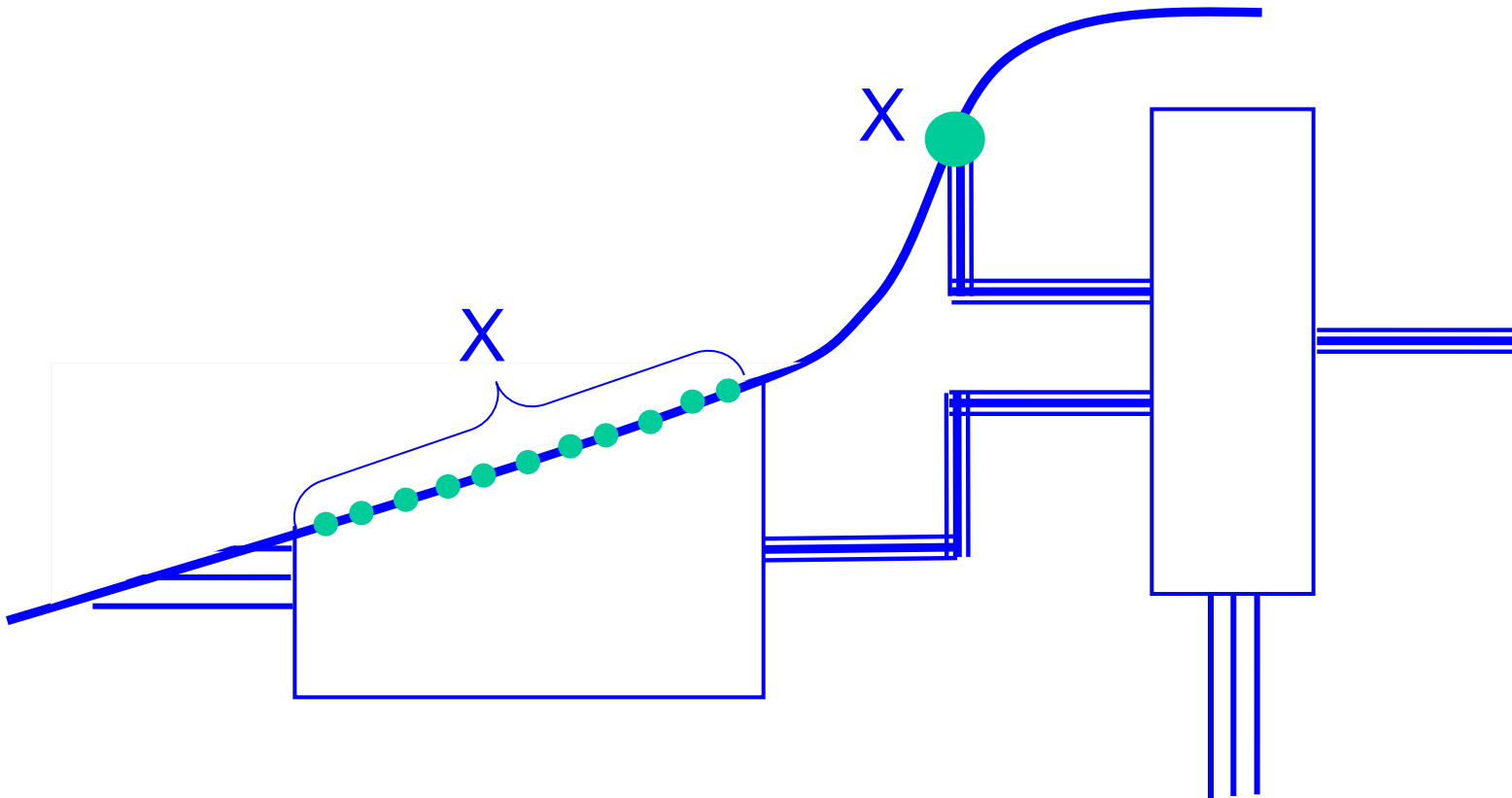
Abstract model contains more behaviors

- Property is **true** on abstract model \Rightarrow
Property is **true** on the concrete model
- Property is **false**: counterexample might be spurious
- Refinement is needed (CEGAR)

- Finding cutpoints:
computationally expensive or needs human expertise
- False negative results:
overhead in checking if counterexample is spurious

3-Valued Abstraction

- Add a third value "X" ("Unknown")

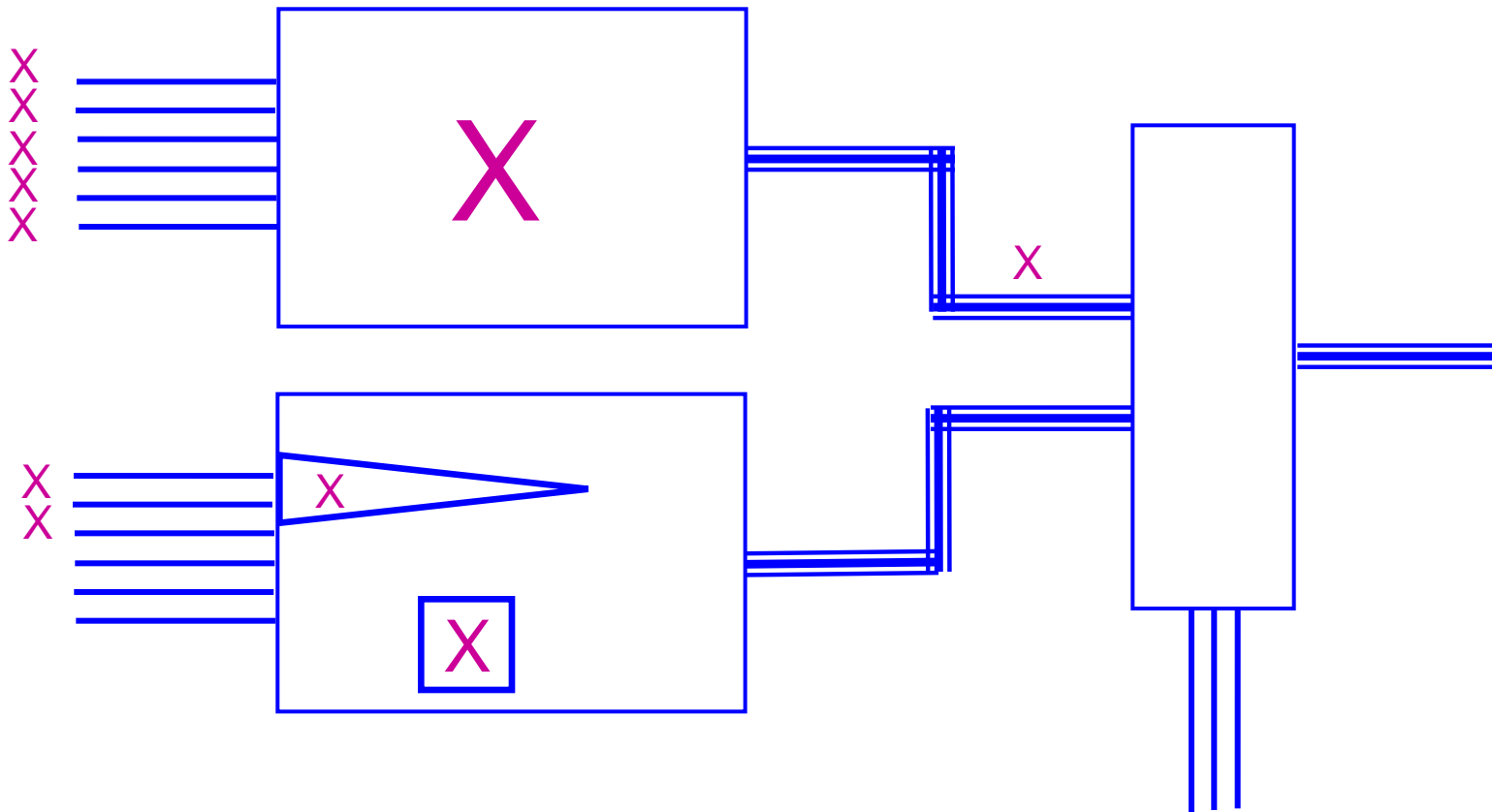


Introducing X (“Unknown”)

- Property is **true** on abstract model \Rightarrow
Property is **true** on the concrete model
- Property is **false** on abstract model \Rightarrow
Property is **false** on the concrete model
- Property is **X** \Rightarrow needs refinement

3-Valued Abstraction

- Add a third value "X"

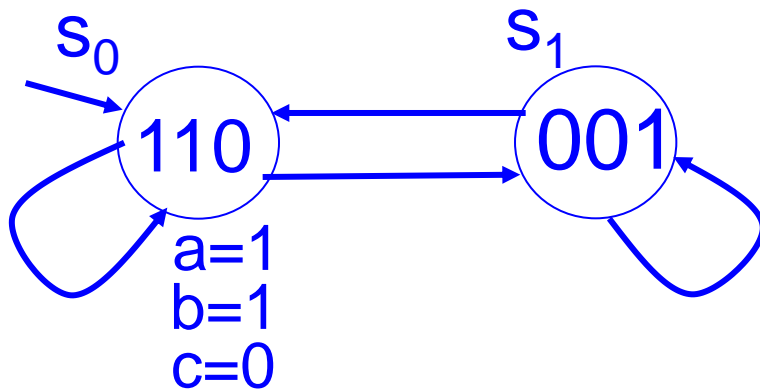


Outline

- LTL Model Checking - Automata Approach
 - Kripke Structures and LTL
 - Büchi Automata
 - BMC
- 3-Valued Abstraction
- 3-Valued BMC (X-BMC)

Kripke Structure

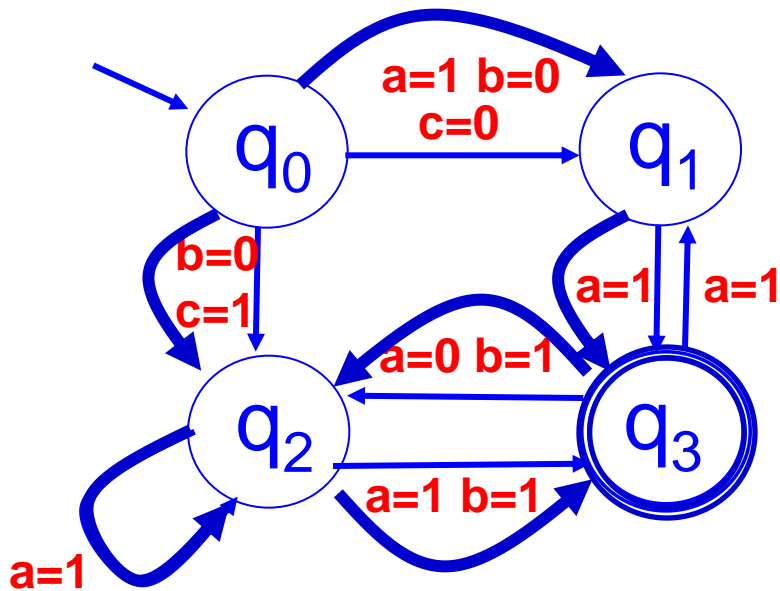
- $M = (S, s_0, R, L)$ over AP
- $L: S \rightarrow (AP \rightarrow \{0,1\})$ $L: S \rightarrow \{0,1\}^{AP}$
- Can describe hardware circuits



$$AP = \{a, b, c\}$$

Büchi Automata

- $B = (\Sigma, Q, q_0, \rho, \alpha)$ $\rho: Q \times \Sigma \rightarrow 2^Q$ $w \in \Sigma^\omega$
- Accepts w iff there is an accepting run for w
 - Such that α is met infinitely often

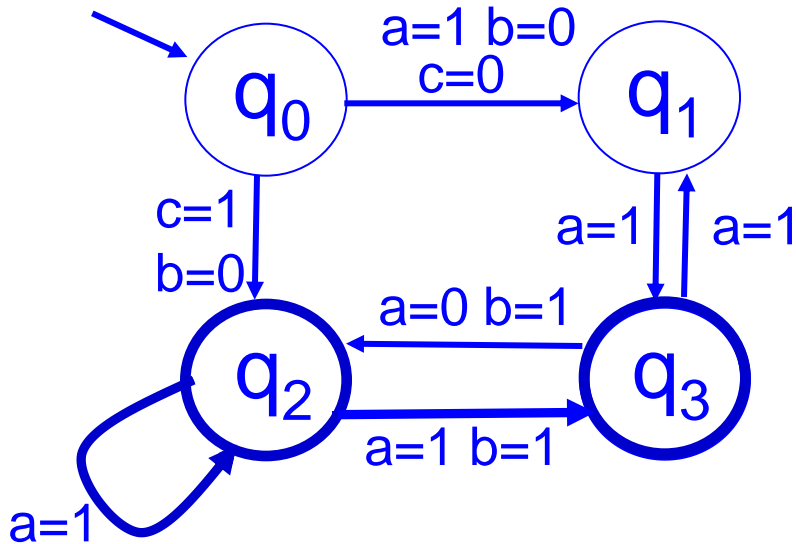


$$\Sigma = \{0,1\}^{\{a,b,c\}}, \quad \alpha = \{q_3\}$$

- 100,100,010,110,010,110,010,110...
- ✗ 010,010,010,010...
- ✗ 001,100,100,100...

Büchi Automata

- ρ can be represented as a function $F:Q \times \Sigma \times N \rightarrow Q$
 - $q' = F(q, \sigma, nd)$



$$\rho(q_2, 110) = \{q_2, q_3\}$$

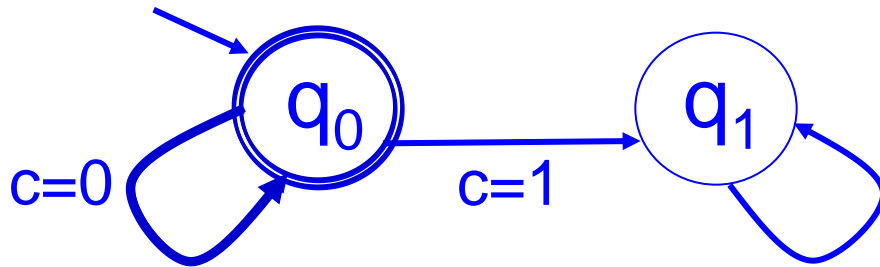
$$F(q_2, 110, 0) = q_2$$

$$F(q_2, 110, 1) = q_3$$

Büchi for LTL

- Given $\varphi = A\psi$, build an automaton $B_{\neg\psi}$ for $\neg\psi$
- $\Sigma = \{0,1\}^{AP}$

$$P = AFc$$

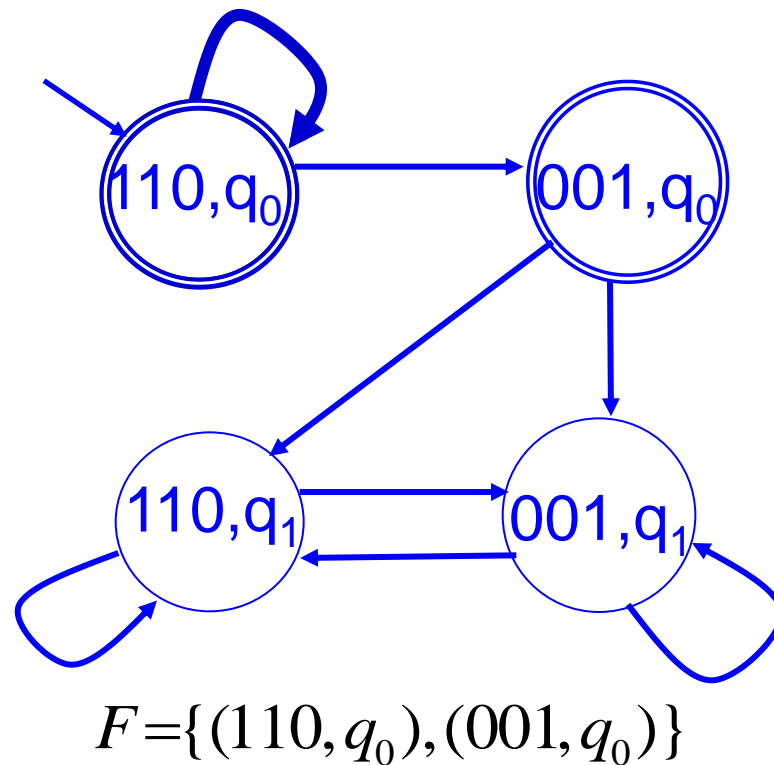
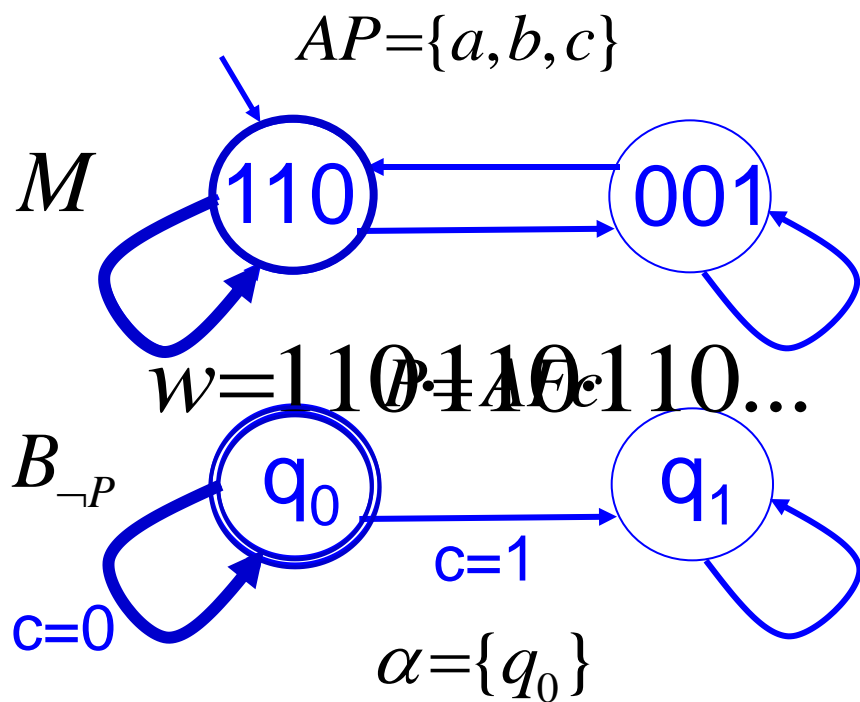


$$\alpha = \{q_0\}$$

$$\pi = q_0, q_0, q_0, q_0 \dots$$

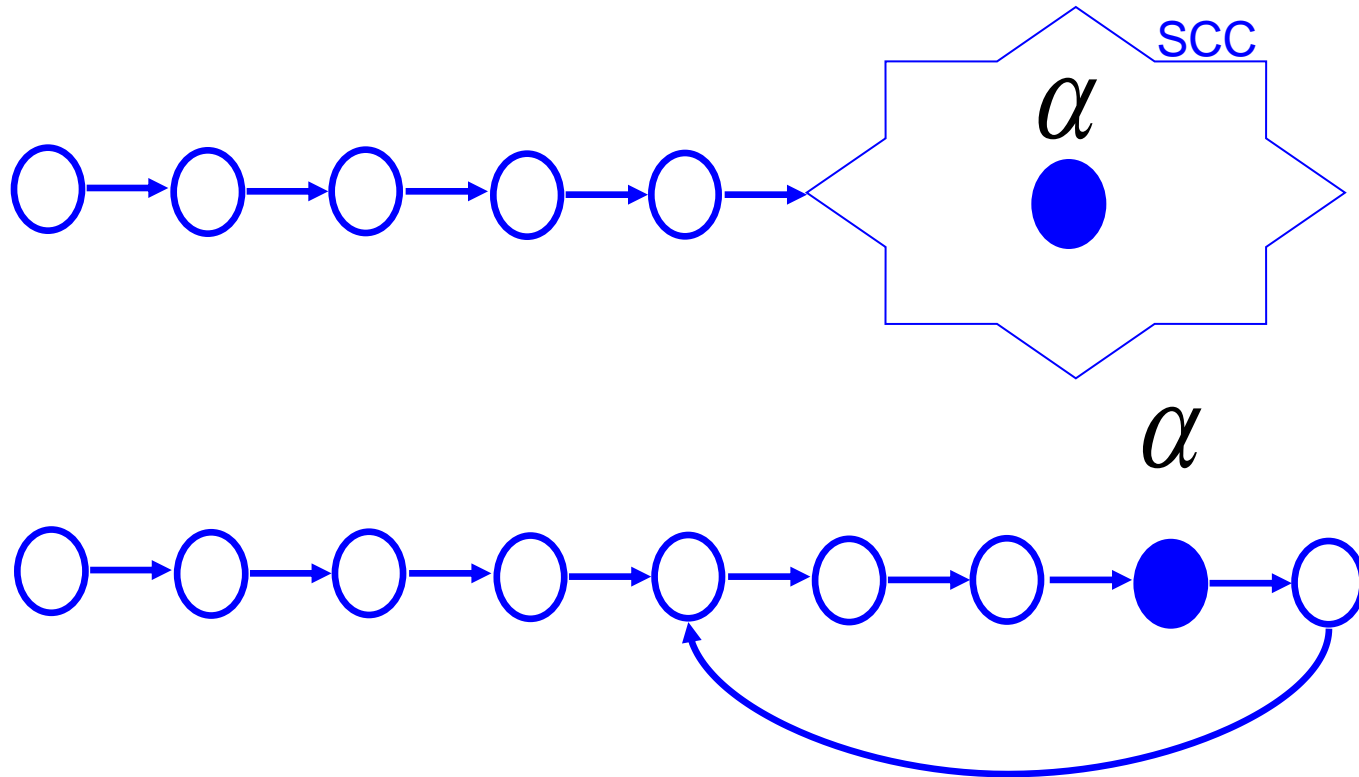
Model Checking

- Let $E = M \times B$ $F = S \times \alpha$
- Reduce Model Checking to Emptiness of E



Model Checking

- Fair Paths in E



Bounded Model Checking (BMC)

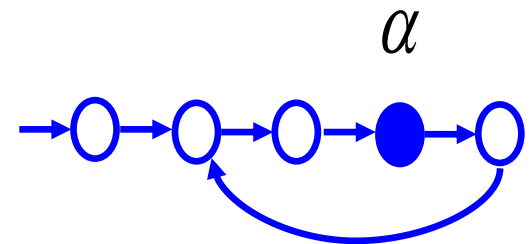
- Build a propositional representation of E
 - Describe paths of bounded length

$$\varphi_M^i(\bar{v}_0 \dots \bar{v}_i) = I_0^M(\bar{v}_0) \wedge \bigwedge_{0 \leq j < i} R_M(\bar{v}_j, \bar{v}_{j+1})$$

$$\varphi_B^i(\bar{v}_0 \dots \bar{v}_i) = I_0^B(\bar{v}_0) \wedge \bigwedge_{0 \leq j < i} R_B(\bar{v}_j, \bar{v}_{j+1}) \wedge fair_i$$

$$fair_i(\bar{v}_0 \dots \bar{v}_i) = \bigvee_{0 \leq l < i} ((\bar{v}_l = v_i) \wedge \bigvee_{l \leq j \leq i} \alpha_E(\bar{v}_j))$$

$$\varphi_i(\bar{v}_0 \dots \bar{v}_i) = \varphi_M^i \wedge \varphi_B^i$$



BMC

- Check finite paths in E

$BMC(M, P)$

$i \leftarrow 0$

while(*true*) {

 if SAT (φ_i) return *false*

inc(i)

}

3-Valued logic

- Ternary domain $D = \{0, 1, X\}$
 - X is "unknown" (not "don't care")

	0	1	X
\neg	1	0	X

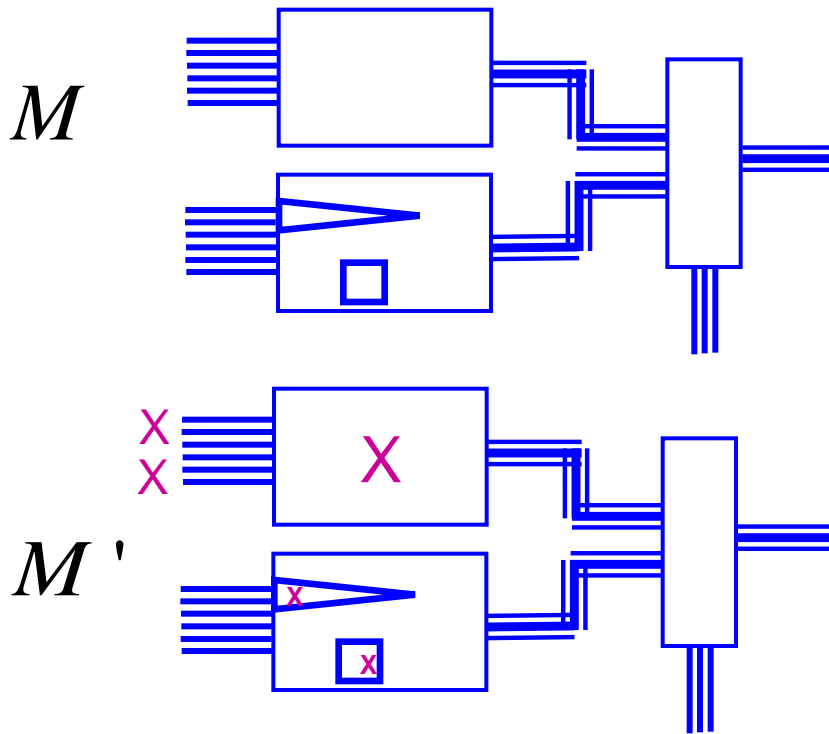
\wedge	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

\vee	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

- Ternary operators agree with Boolean operators on Boolean values

3-Valued Abstraction

- Ternary domain $D = \{0, 1, X\}$
 - X is "unknown" (not "don't care")

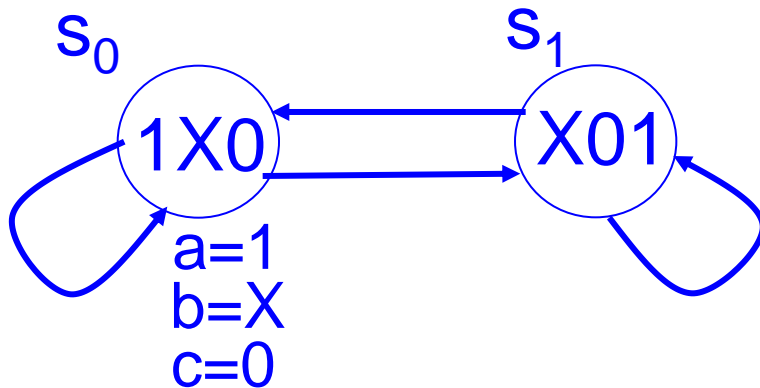


$$[M' \models P] = 1 \Rightarrow [M \models P] = 1$$
$$[M' \models P] = 0 \Rightarrow [M \models P] = 0$$



3-Valued Kripke Structure

- $M' = (S', s'_0, R', L')$ over AP
- $L': S' \rightarrow \{0, 1, X\}^{AP}$



$$AP = \{a, b, c\}$$

3-Valued LTL

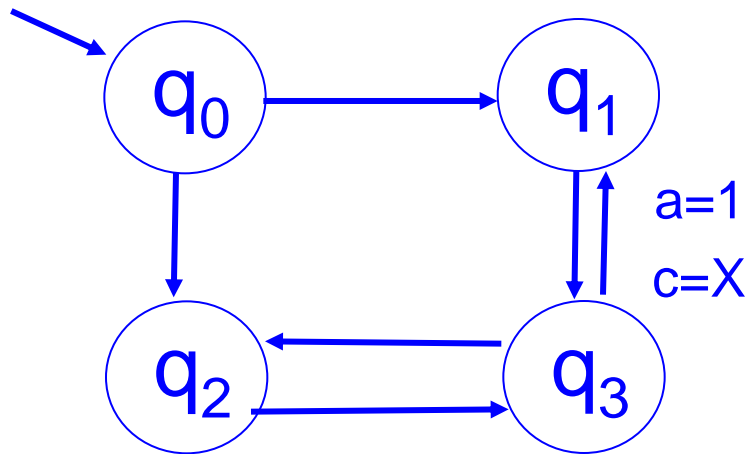
- Over AP
- $P = A\psi$

$$\pi \models \psi \in \{0, 1, X\}$$

$$[M \models P] = \begin{cases} 1 & \forall \pi, [\pi \models \psi] = 1 \\ 0 & \exists \pi, [\pi \models \psi] = 0 \\ X & \text{otherwise} \end{cases}$$

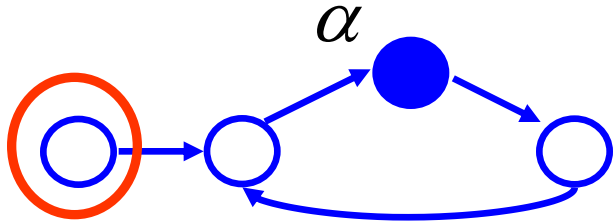
3-Valued Büchi

- $\Sigma = \{0, 1, X\}^{AP}$
- 3-Valued transition function F' for ρ
 - $F': Q \times \Sigma \times N \rightarrow Q$
 - Ternary variables and operators

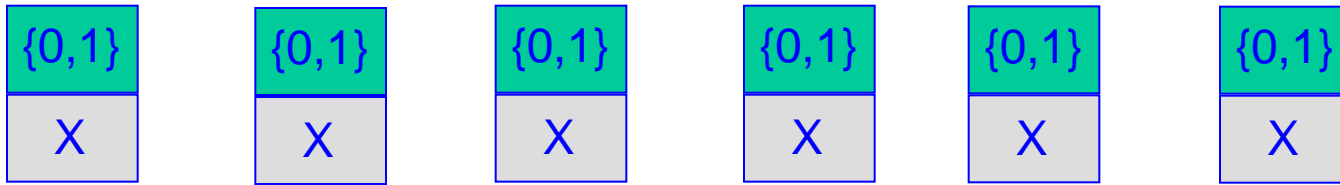


$$F'(q_3, 11X, 0) = q_1$$

3-Valued Model Checking

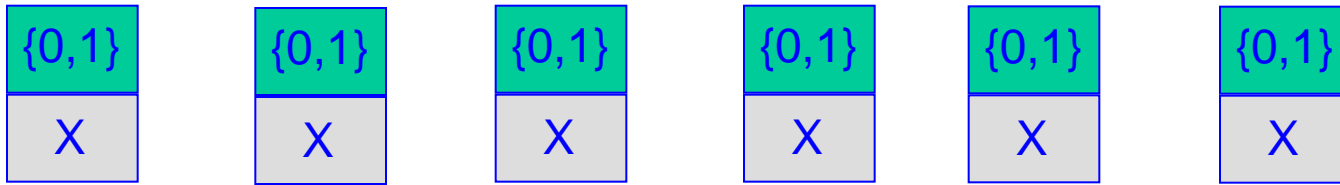
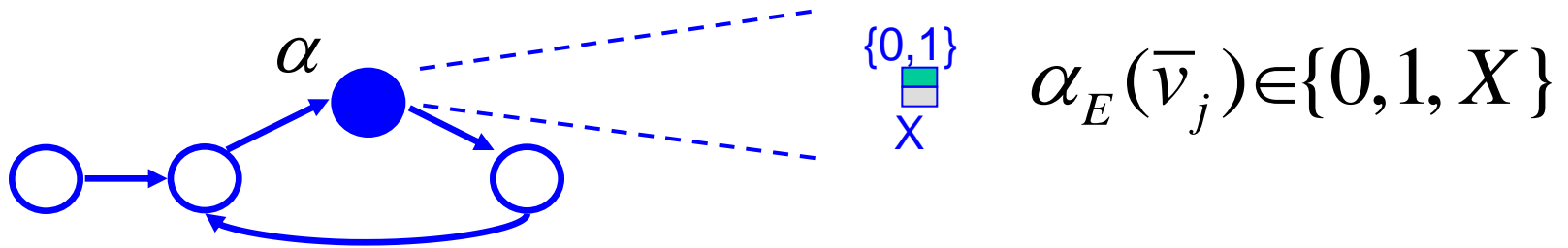


$$E' = M' \times B'$$



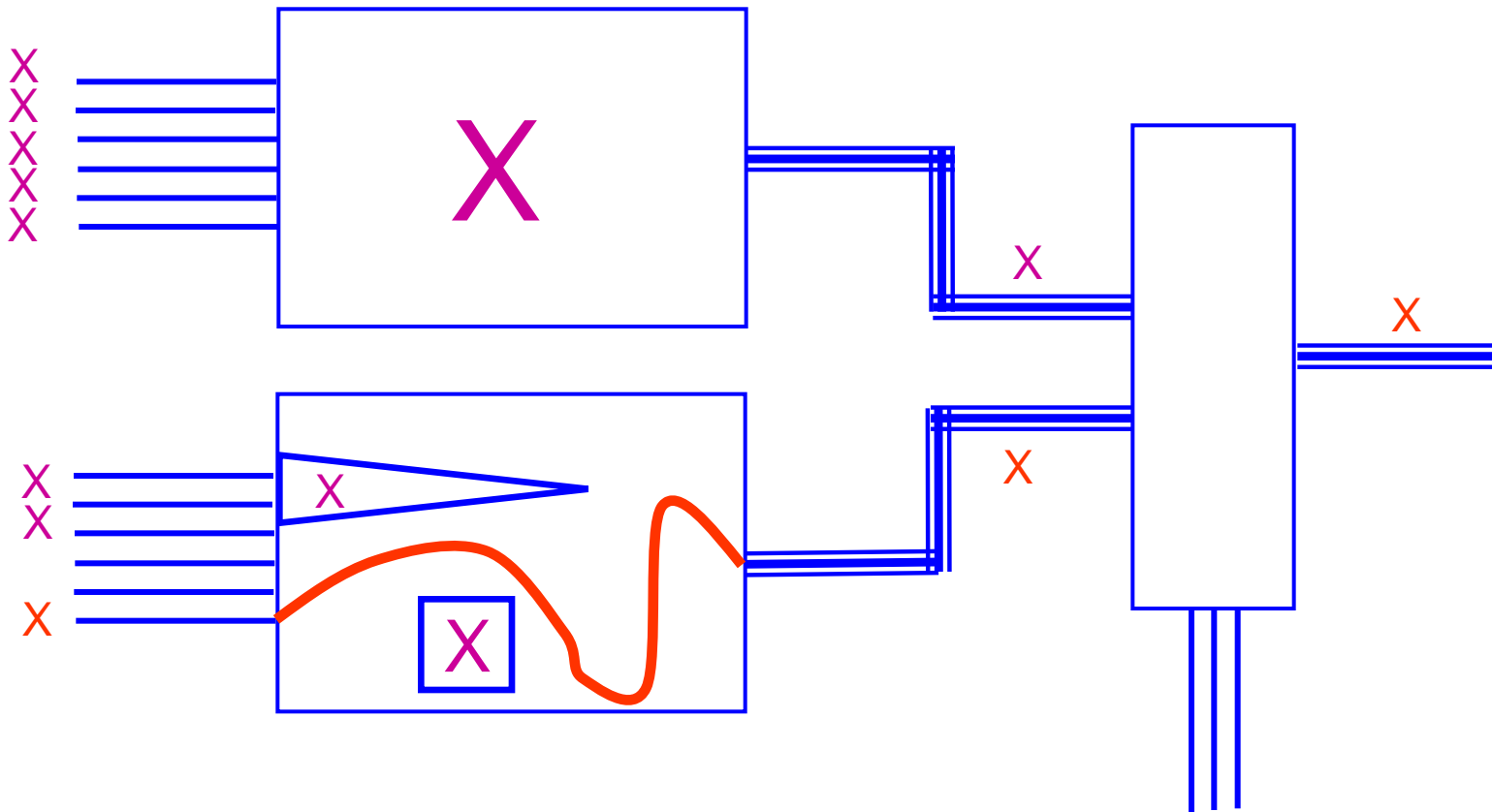
- A short loop is a witness for a long concrete loop
 - Lower the bound required for finding bugs

3-Valued Model Checking



- Checking might yield an "unknown" result.

X-BMC



BMC - Reminder

$$\varphi_M^i(\bar{v}_0 \dots \bar{v}_i) = I_0^M(\bar{v}_0) \wedge \bigwedge_{0 \leq j < i} R_M(\bar{v}_j, \bar{v}_{j+1})$$

$$\varphi_B^i(\bar{v}_0 \dots \bar{v}_i) = I_0^B(\bar{v}_0) \wedge \bigwedge_{0 \leq j < i} R_B(\bar{v}_j, \bar{v}_{j+1}) \wedge \text{fair}_i$$

$$\text{fair}_i(\bar{v}_0 \dots \bar{v}_i) = \bigvee_{0 \leq l < i} ((\bar{v}_l = \bar{v}_j) \wedge \bigvee_{l \leq j \leq i} \alpha_E(\bar{v}_j))$$

X-BMC

- Create 3-Valued propositional formulae (dual rail)

$BMC(M', \psi) \{$

$i \leftarrow 0$

$while(true) \{$

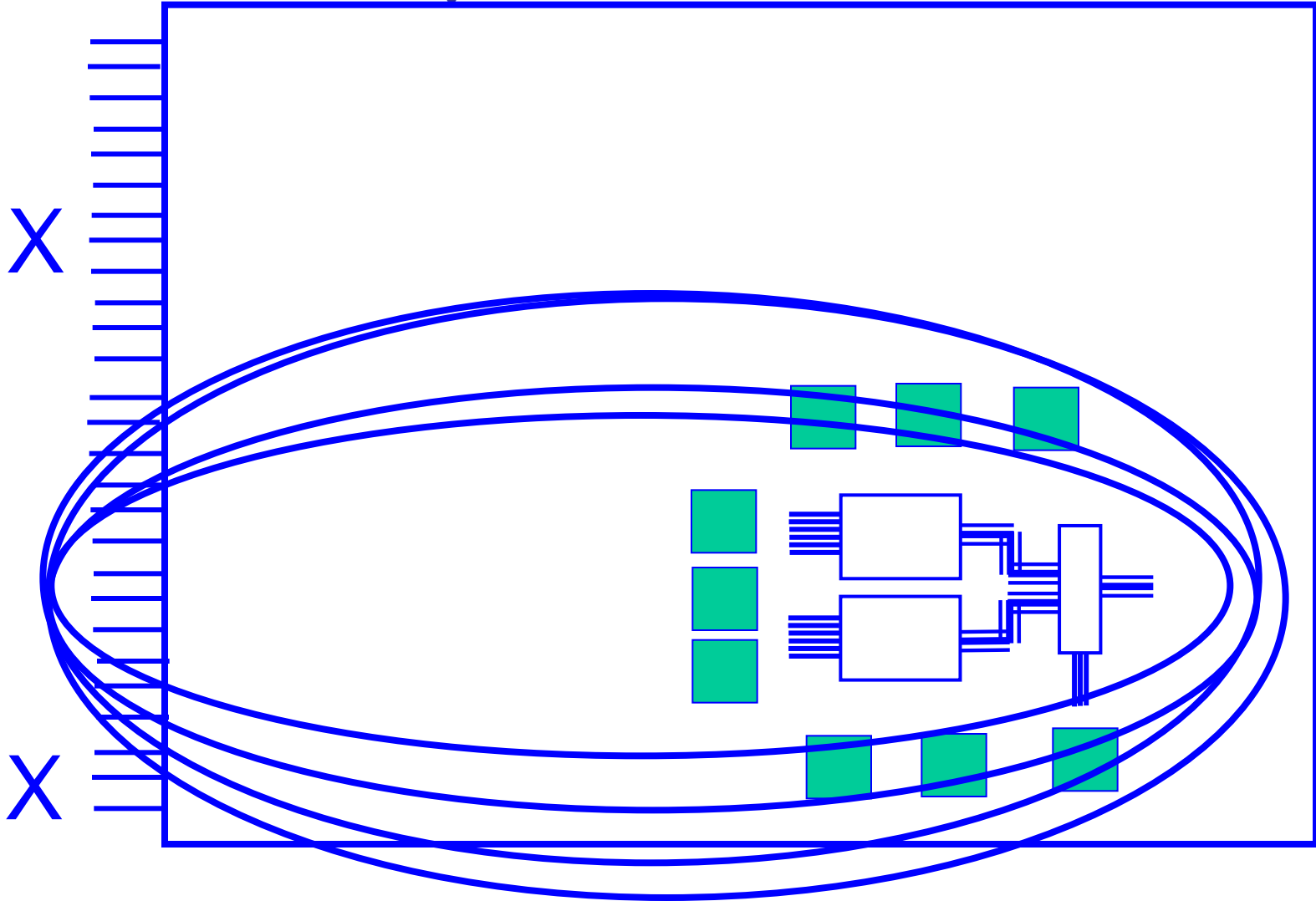
if $SAT(\varphi_{M'}^i = 1 \wedge \varphi_B^i = 1)$ return *false*

if $SAT(\varphi_{M'}^i = 1 \wedge \varphi_B^i = X)$ return X

$inc(i)$

$\}$

Holy Grail - Revisited



Experimental Results (EXE Cluster)

		Model	EXE	Abs 1	Abs 2	Abs 3	Abs 4	Abs 5
		# Latches	133K	132K	115K	108K	74K	71K
		# Gates	6.1M	6.0M	5.9M	5.8M	0.6M	0.5M
Property	Result	Run Time (s)						
XBMC	P1	fail	266	281	270	254	103	105
	P2	pass	262	271	265	244	212	205
	P3	fail	264	280	249	282	285	103
	P4	pass	412	365	342	323	X	X
	P5	fail	278	267	252	264	110	108
	P6	pass	654	640	631	615	587	552
BMC	P1	fail	M/O	M/O	M/O	12280	525	168
	P2	pass	M/O	M/O	M/O	479	411	235
	P3	fail	M/O	M/O	M/O	M/O	M/O	408
	P4	F/N	M/O	M/O	M/O	M/O	F/N	F/N
	P5	fail	M/O	M/O	M/O	M/O	908	632
	P6	pass	M/O	M/O	M/O	M/O	2241	199

Conclusion

- 3-Valued Abstraction
 - Models, specification and automata
 - Automatic or manual abstraction
 - Abstraction of inputs to the model
- 3-Valued Bounded Model Checking
 - Enhanced performance
 - Increased capacity
 - Reduced counterexample lengths
 - Insensitive to size of irrelevant parts of the model
 - Allows checking higher level models
 - Change in methodology (!)
- Unbounded Model Checking (Induction)
- Automatic Refinement

Thank You

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

- **Existential abstraction + data abstraction:**

E. M. Clarke, O. Grumberg, D. E. Long,
Model Checking and Abstraction, TOPLAS,
1994.

- **Localization reduction:**

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

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

- **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