Model Checking

Orna Grumberg
Technion
Haifa, Israel

Taiwan, October 8, 2009

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

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

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

- **communication protocols:**
  ($\neg$ get-message) **until** send-message

Model Checking [EC81,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, Cadence, …

- Recently applied successfully also for software verification: SLAM (Microsoft), Java PathFinder and SPIN (NASA), BLAST (EPFL), CBMC (Oxford),…

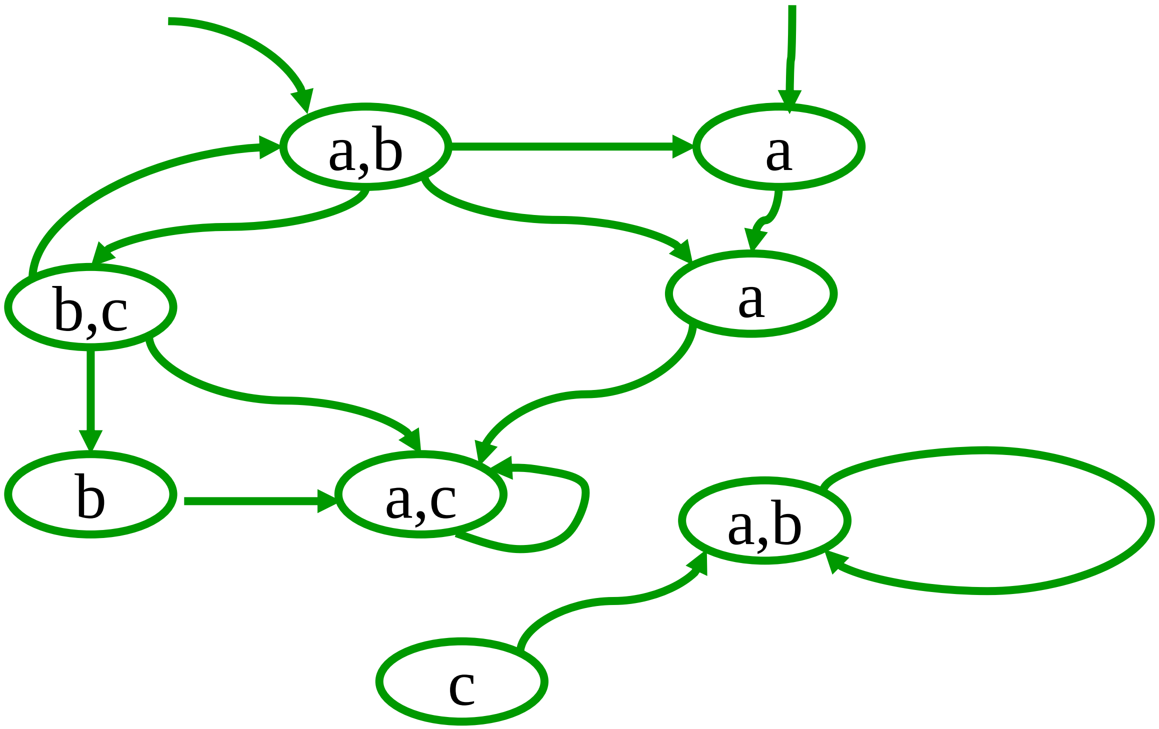
Clarke, Emerson, and Sifakis won the 2007 Turing award for their contribution to Model Checking

Overview

- Temporal logics
- Model Checking
- BDD-based (symbolic) model checking
- SAT-based (bounded) model checking

Model of a system

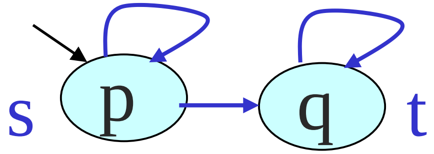
Kripke structure / transition system



Notation

- Kripke structure $M = (S, I, R, L)$
  - S : (finite) set of states
  - I : set of initial states
  - R : set of transitions
  - L: labeling function, associates each state with a subset of atomic propositions *AP*

M:



$AP = \{p, q\}$

$\pi = s_0 s_1 s_2 \ldots$ is a **path** in M from s iff

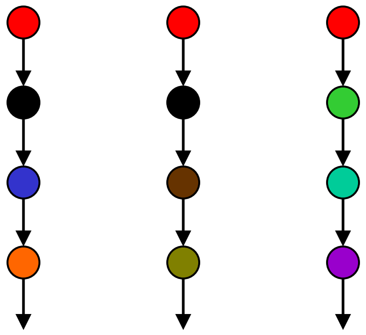
$s = s_0$ and

for every $i \geq 0$: $(s_i, s_{i+1}) \in R$

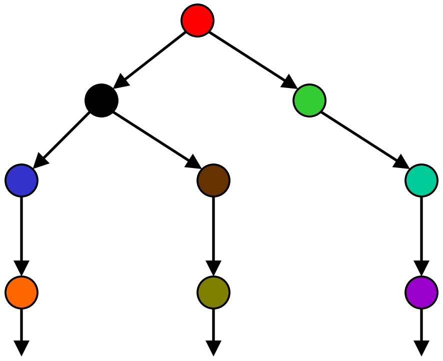
Temporal Logics

- Temporal Logics
  - Express properties of event orderings in time

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



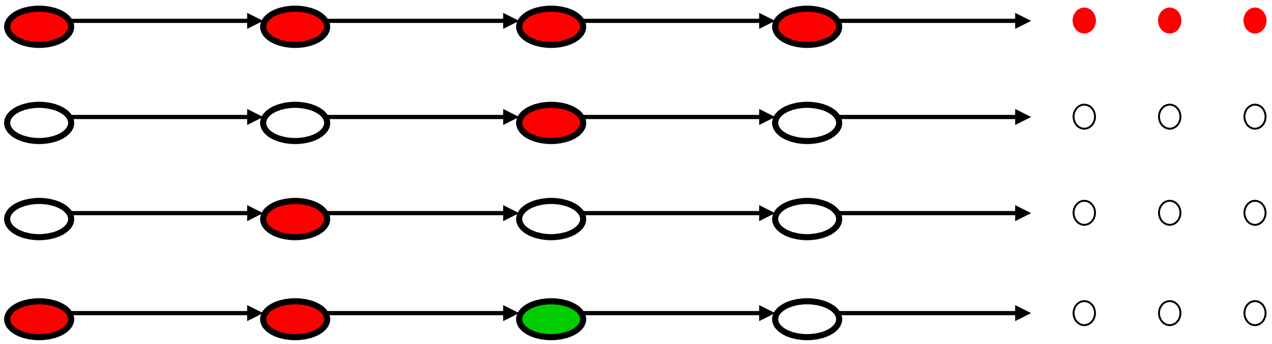
- **Branching Time**
  - Every moment has several successors
  - Infinite tree
  - Computation Tree Logic (CTL)



Propositional temporal logic

AP - a set of atomic propositions

Temporal operators:



Gp

Fp

Xp

pUq

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

E there **exists** a path

$M \models f \Leftrightarrow$ **for every initial state s,**
$s \models f$

Computation Tree Logic (CTL)

CTL operator:

path quantifier + temporal operator

Atomic propositions: **p** $\in$ ***AP***

Boolean operators: **f** $\wedge$ **g** , $\neg$**f**

CTL temporal operators: **EX f, E(fUg), EGf**

More CTL operators

Universal formulas:

- AX f, A(f U g), AG f , AF f

Existential formulas:

- EX f, E(fUg), EG f , EF f

Linear Temporal logic (LTL)

Formulas are of the form **A**f, where **f** can include any **nesting** of temporal operators but **no** path quantifiers

Example: LTL formula which is not CTL

A GF p

Meaning, along every path, infinitely often p

CTL*

Includes LTL and CTL and more

Example formulas

CTL formulas:

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

LTL formulas:

- **fairness:** $A(GF\ enabled \Rightarrow GF\ executed)$
- $A(x{=}a \wedge y{=}b \Rightarrow$ **XXXX** $z{=}a{+}b)$

Property types

	Universal	Existential
Safety	AGp	EGp
Liveness	AFp	EFp

Property types (cont.)

Combination of **universal safety** and **existential liveness**:

"along **every** possible execution, in **every state** **there is** a possible continuation that will **eventually** reach a reset state"

AG EF reset

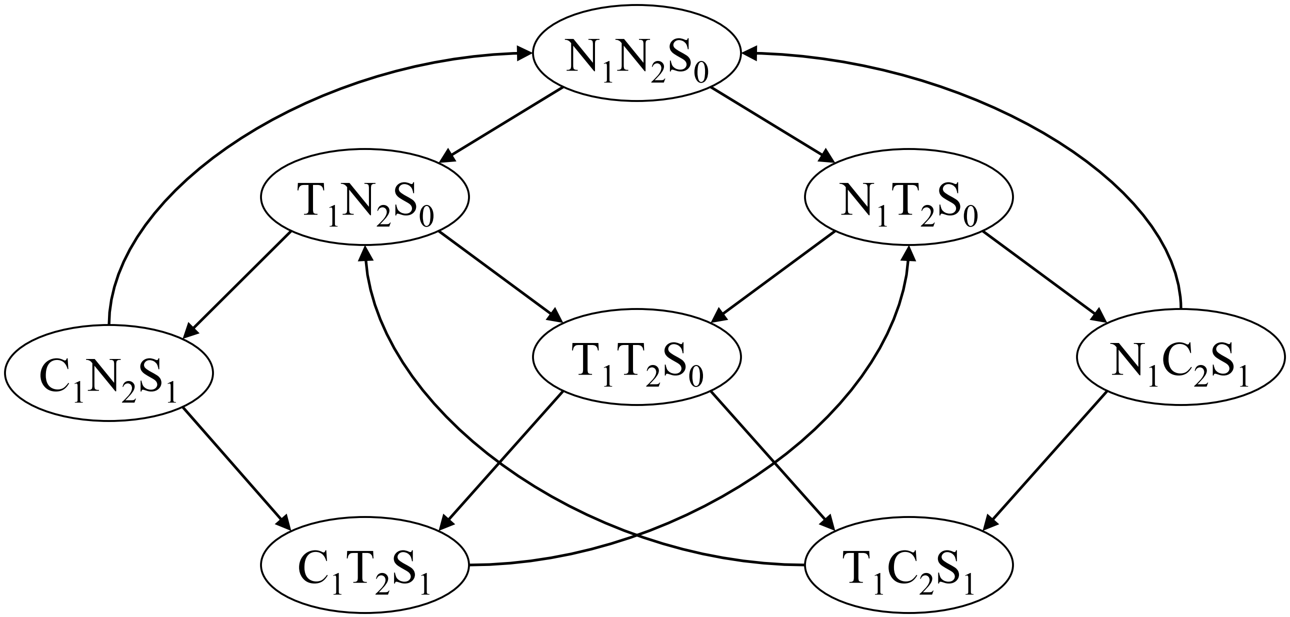
Mutual Exclusion Example

[by Willem Visser]

- 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 --- $N_1\ N_2\ S_0$

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

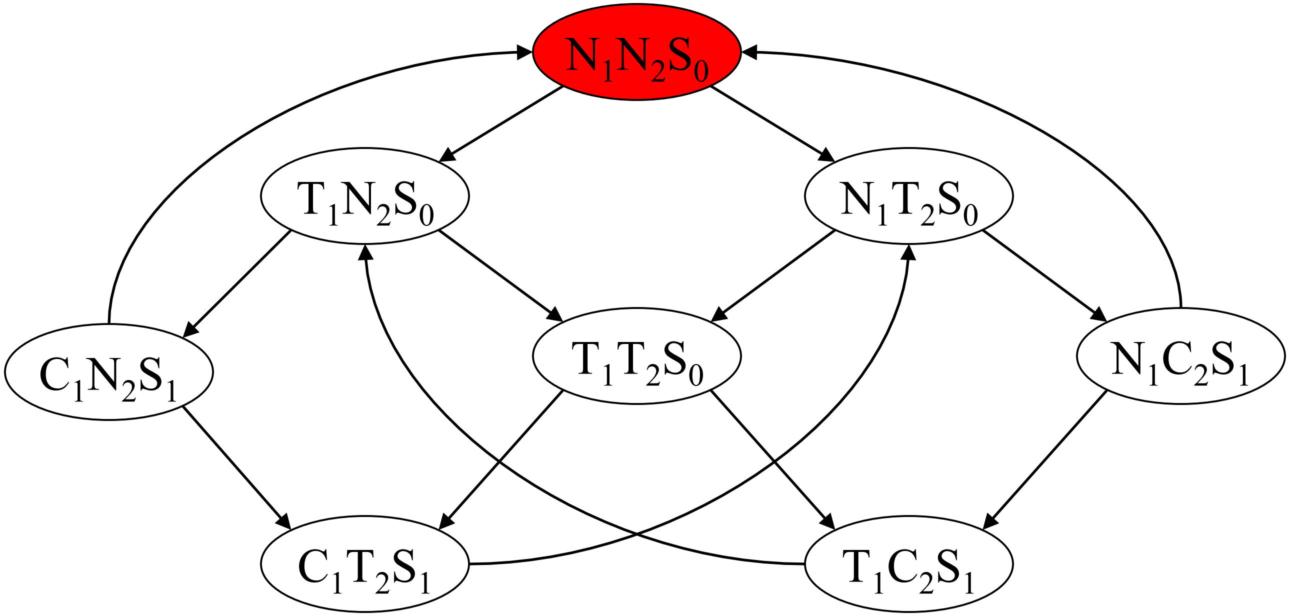
Model for Mutual Exclusion



Specification: $M \models$ **AG** EF $(N_1 \wedge N_2 \wedge S_0)$

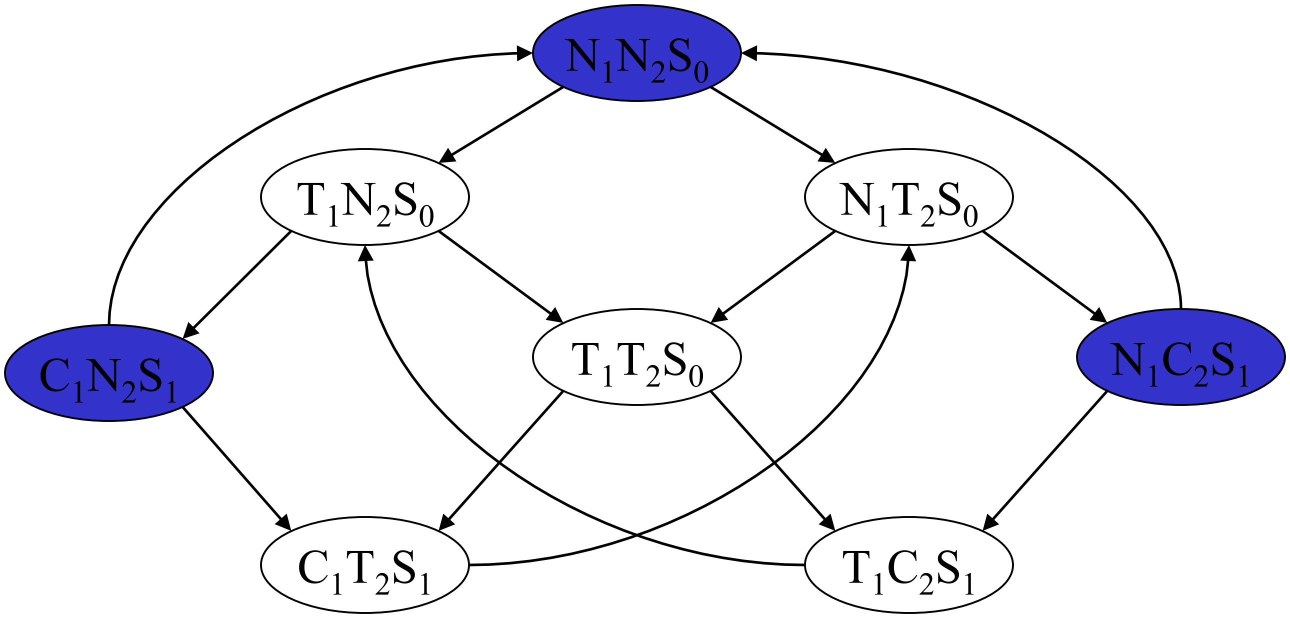
No matter where you are there is always a way to get to the initial state

Mutual Exclusion Example



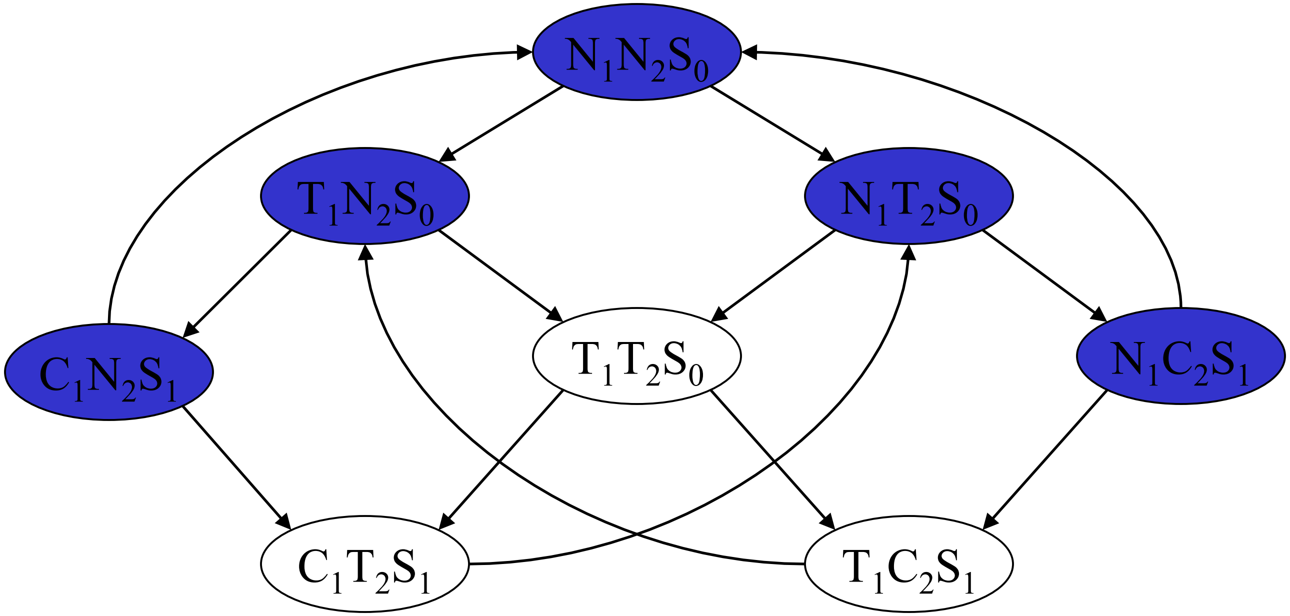
$$M \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$$

Mutual Exclusion Example



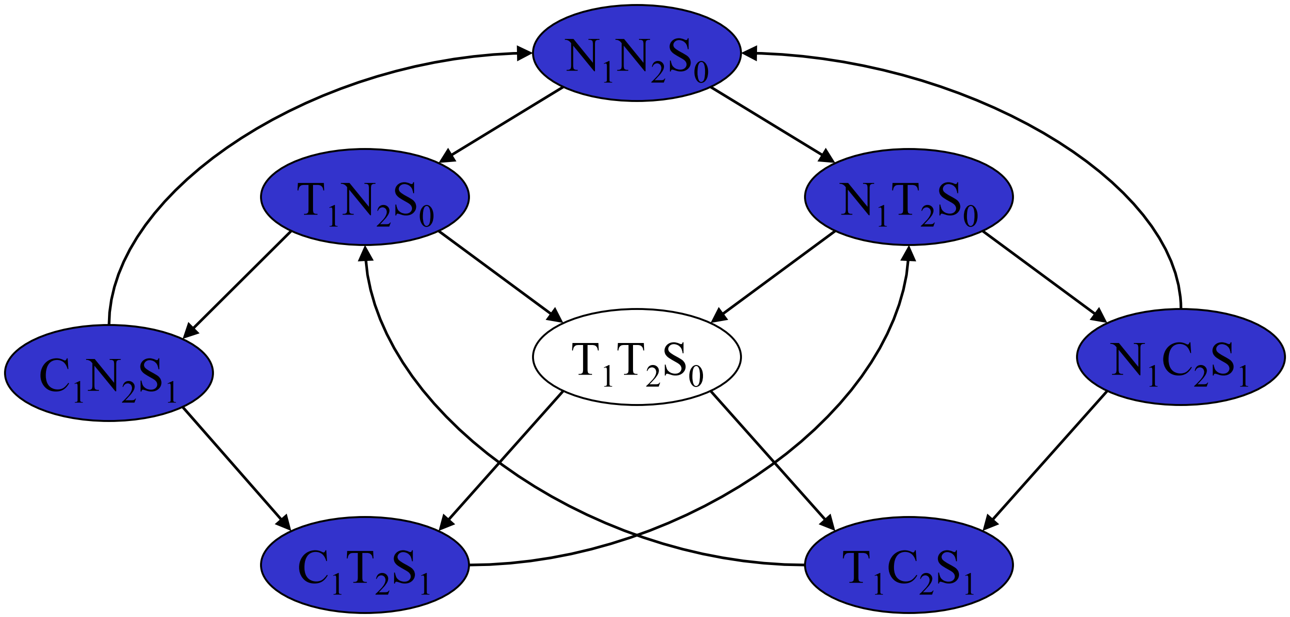
$$M \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$$

Mutual Exclusion Example



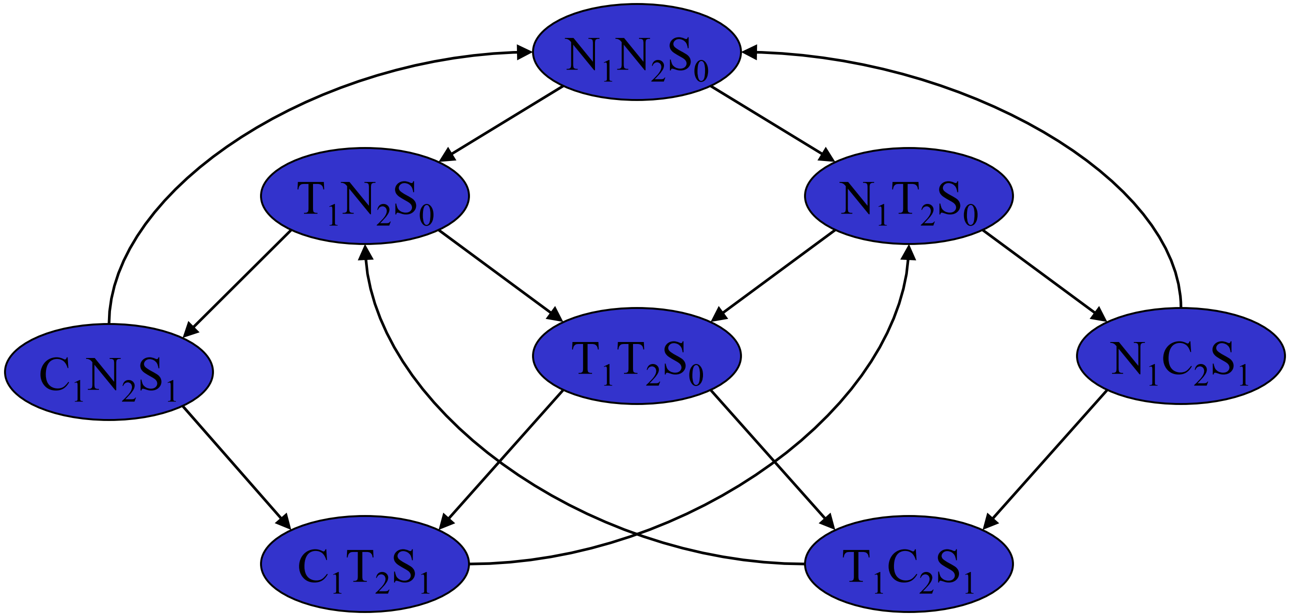
$$M \models AG\ EF\ (N_1 \land N_2 \land S_0)$$

Mutual Exclusion Example



$$M \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$$

Mutual Exclusion Example



$$M \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$$

No matter where you are there is always a way to get to the initial state

Model Checking M |= f

- The Model Checking algorithm works **iteratively** on subformulas of **f** , from **simpler** subformulas to more **complex** ones
- For checking **AG(request ⇒ AF grant)**
  - Check **grant, request**
  - Then check **AF grant**
  - Next check **request ⇒ AF grant**
  - Finally check **AG(request ⇒ AF grant)**

Model Checking M |= f (cont.)

- We check subformula **g** of **f** only after all subformulas of **g** have already been checked
- For subformula **g**, the algorithm returns the **set of states** that satisfy **g** (S_g)
- The algorithm has time complexity: O(|M| × |f|)

Model Checking M |= f (cont.)

- M |= f if and only if all initial states of M are contained in S_f .

Basic operations in model checking

Given a set of states Q:

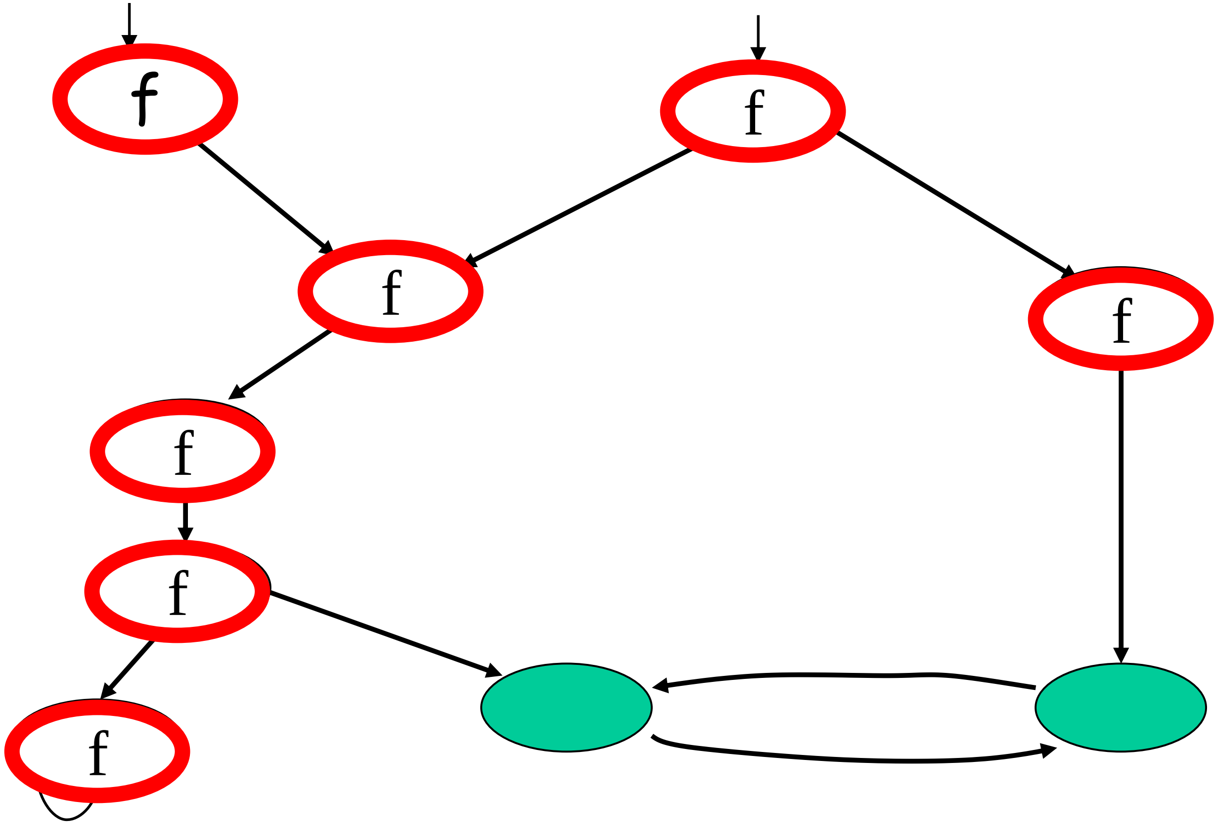
- **Succ(Q)** returns the set of successors of the states in Q
- **Pred(Q)** returns the set of states that have a successor in Q

Model checking **f = EF g**

Given: a model **M** and the set S_g of states satisfying **g** in M

```
procedure CheckEF ($S_g$)
Q := emptyset;  Q' := $S_g$ ;
while Q $\neq$ Q'  do
  Q := Q';
  Q' := Q $\cup$ Pred(Q)
end while
$S_f$ := Q ;   return($S_f$ )
```

Example: **f = EF g**

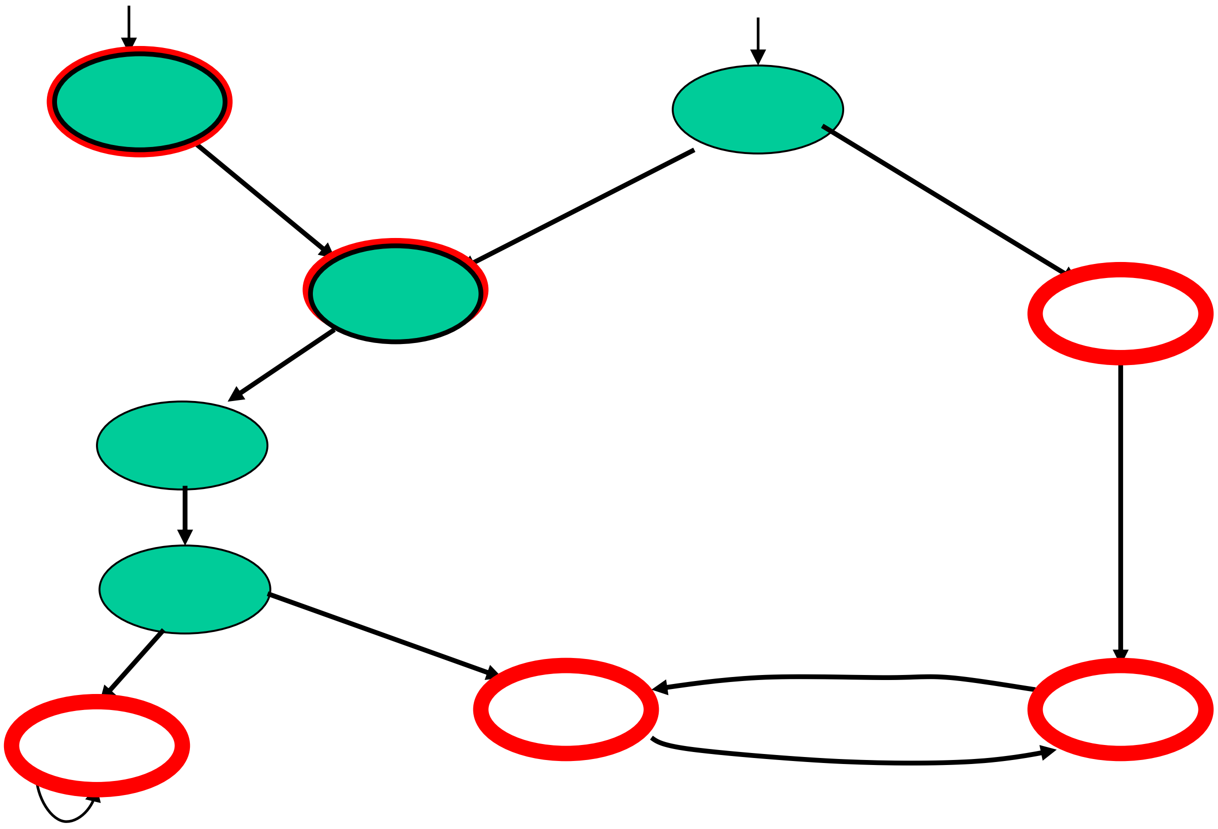


Model checking $f = EG\ g$

CheckEG gets **M** and S_g and returns S_f

```
procedure CheckEG ($S_g$)
Q := S ;  Q' := $S_g$ ;
while Q $\neq$ Q' do
  Q := Q';
  Q' := Q $\cap$ Pred(Q)
end while
$S_f$ := Q ;   return($S_f$ )
```

Example: f = EG g



Reachability + model checking f=AGp

- Starting from the initial states, iteratively computes the set of successors.

- At each iteration checks whether it reached a state which satisfies $\neg p$.
  - If yes, announces failure.

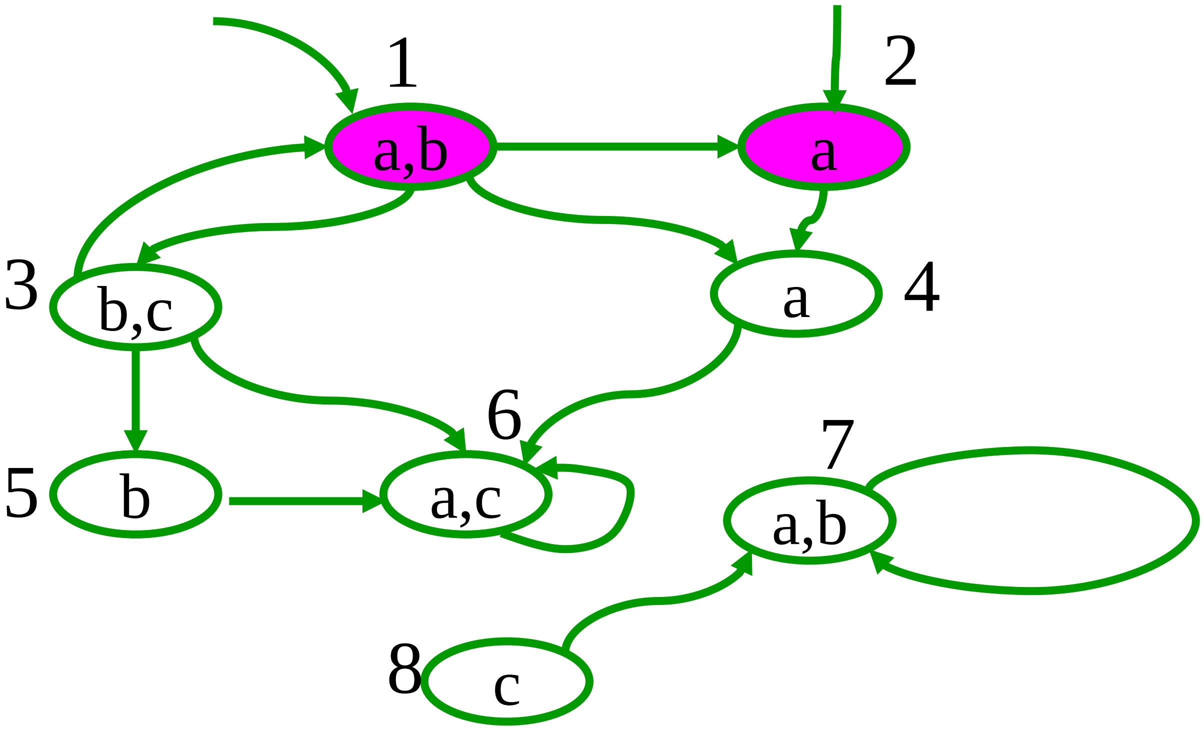
- Stops when no new states are found.
  - Result 1: the set of reachable states.
  - Result 2: **M |= AGp**

Model checking f = AG p

CheckAG gets **M, S_p** and returns **Reach**

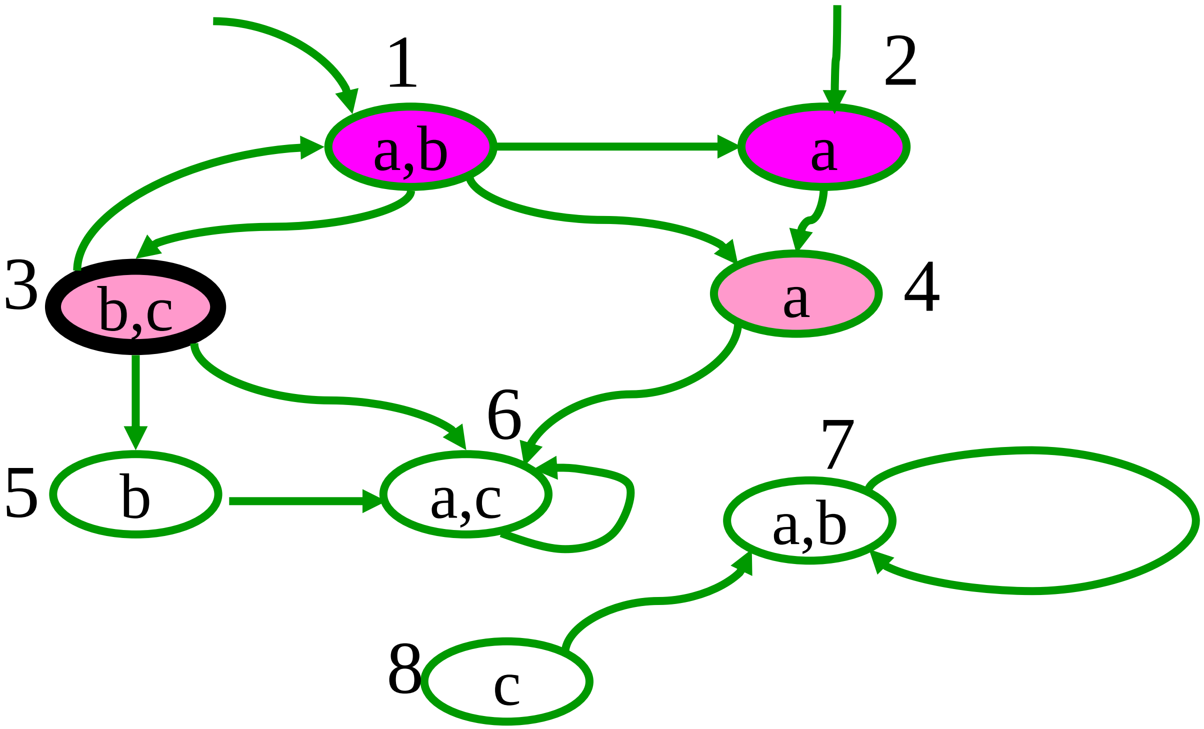
```
procedure CheckAG ($S_p$ )
Reach:= I ; New := I;
while New $\neq \emptyset$ do
    If New $\not\subset S_p$  return (M $\not\models$ AGp)
    New := Succ(New); New :=New\Reach;
    Reach := Reach $\cup$ New;
end while
return( Reach,  M |= AGp)
```

Reachability + checking AG a



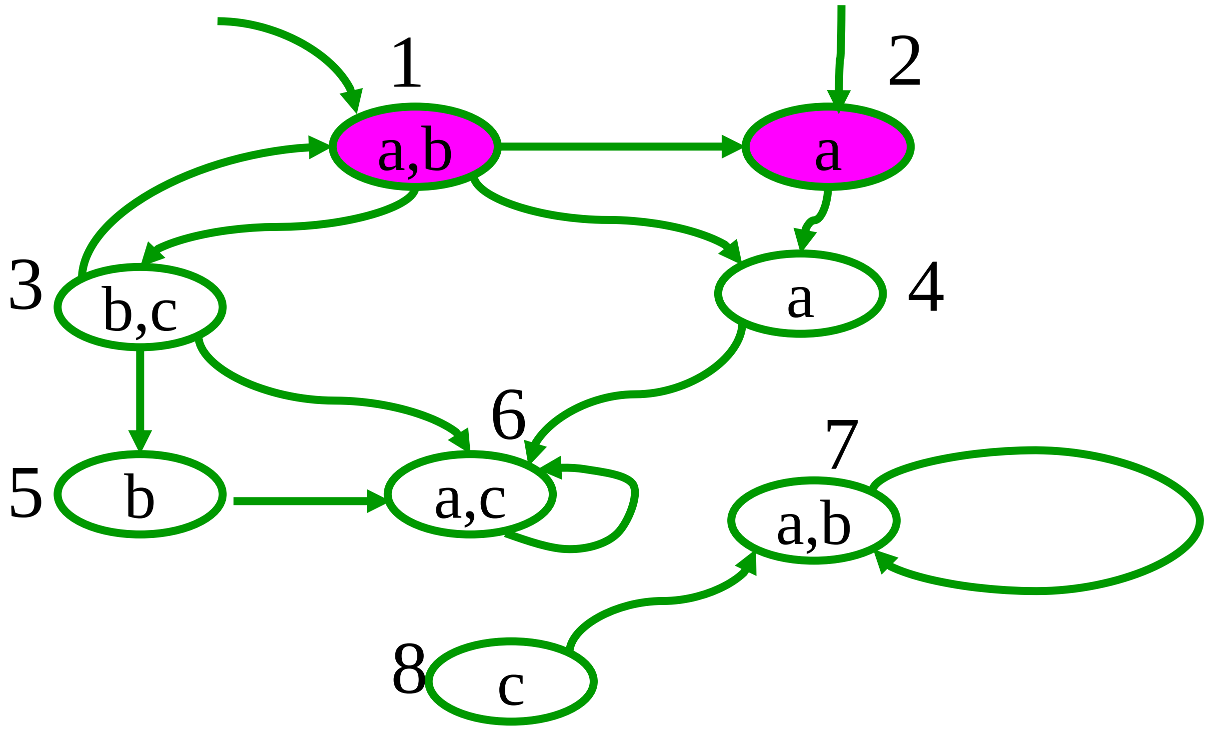
Reach = New = I = { 1, 2 }

Return: M |≠ AG a



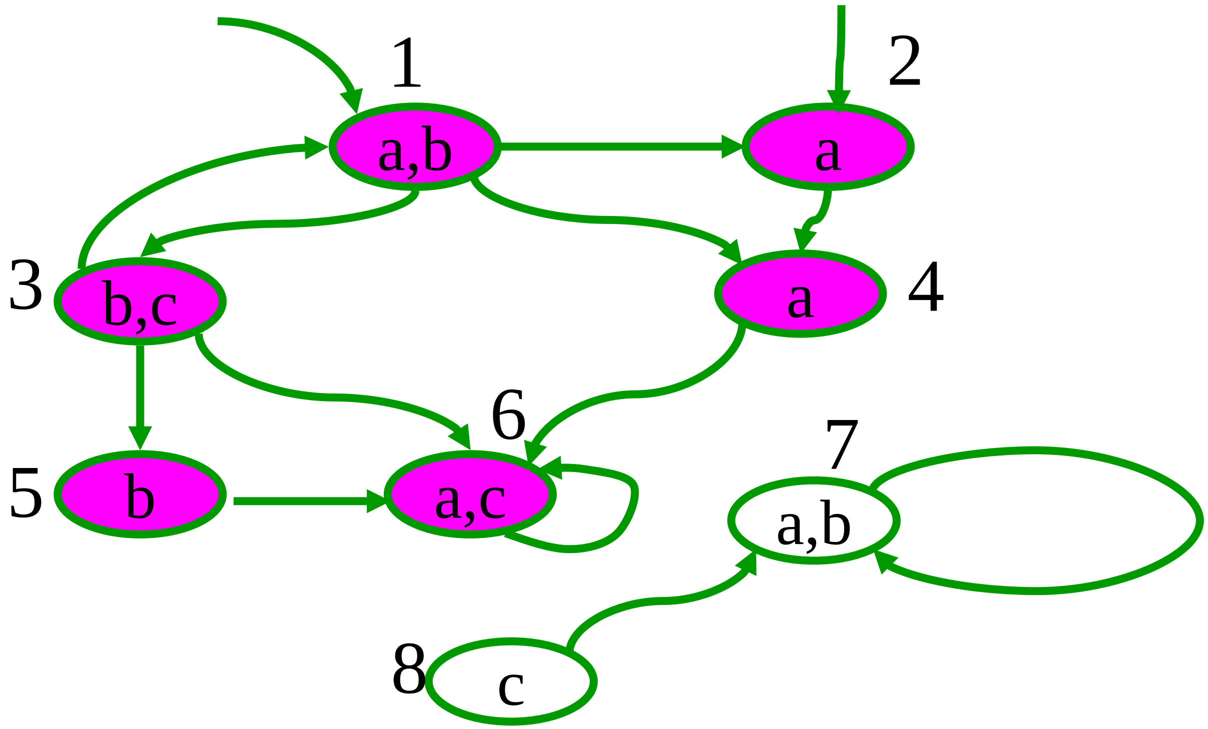
Failure: **New ⊄ S_a**

Reachability + checking AG (a∨b)



Reach = New = I = { 1, 2 }

Return: Reach, M |= AG (avb)



Reach = {1, 2, 3, 4, 5, 6} **New = emptyset**

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

If the model is given **explicitly** (e.g. by **adjacent matrix**) then only systems of restricted size can be handled.

- Strong reduction techniques are needed, e.g. Partial Order Reduction

Symbolic model checking

A solution to the state explosion problem:
BDD-based model checking

- **Binary Decision Diagrams (BDDs)** are used to represent the **model** and **sets of states**.

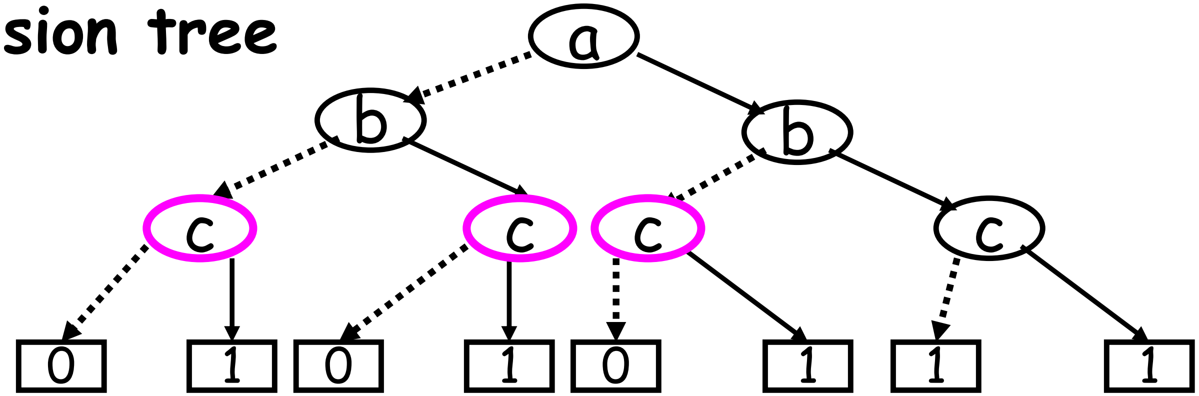
- It can handle systems with **hundreds** of Boolean variables.

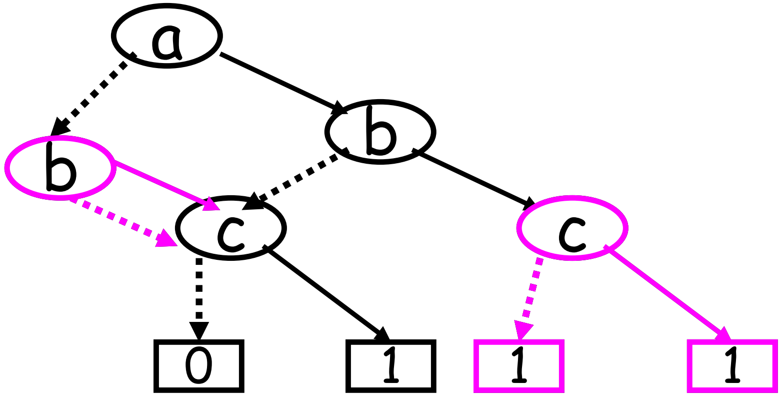
Binary decision diagrams (BDDs)

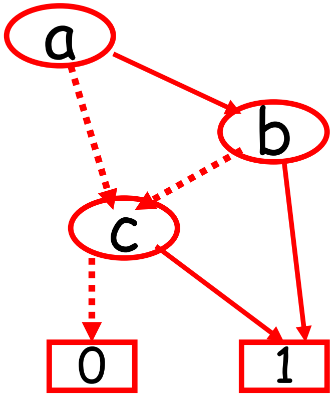
- Data structure for representing Boolean functions
- Often **concise** in memory
- **Canonical** representation
- Most **Boolean operations** can be performed on BDDs in **polynomial time** in the BDD size

BDD for f(a,b,c) = (a ∧ b) ∨ c

Decision tree







BDD

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 U 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 **$v_1 ... v_n$**
- **S_f** can be represented by a BDD over **$v_1 ... v_n$**
- **R** (a set of pairs of states **(s,s')**) can be represented by a BDD over **$v_1 ... v_n$ $v_1' ... v_n'$**

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**.

Symbolic model checking

- Same algorithm as before

- Succ(Q) and Pred(Q) use Boolean operations on BDDs R and Q

- Pred(Q)(s) = ∃**s' [R(s,s') ∧ Q(s')]**
  - **Boolean operations on BDDs R and Q**

Symbolic model checking (cont.)

- Most Boolean operations on BDDs are quadratic in the size of the BDDs
- BDDs are canonical
  - Easy to check Q = Q'

Symbolic Model checking for **f= EF g**

Given: BDDs **R** and **S_g** :

```
procedure CheckEF ($S_g$ )
$Q$ := emptyset;  $Q'$ := $S_g$ ;
While $Q \neq Q'$ do
    $Q$ := $Q'$;
    $Q'$ := $Q \vee$ Pred ( $Q$ )
end while
$S_f$ := $Q$ ;   return($S_f$ )
```

State explosion problem - revisited

- state of the art symbolic model checking can handle effectively designs 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 tools

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

GRASP (Silva, Sakallah)
Prover (Stalmark)
Chaff (Malik)
MiniSAT

The developers of GRASP and Chaff won the 2009 CAV award

- for their contribution to model checking

Bounded model checking for checking AGp

- Unwind the model for k levels, i.e., construct all computation of length k
- If a state satisfying ¬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 φ=**EF**$\neg$**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

$f_{M,2} = (x_0=0 \lor y_0=0 \lor z_0=0) \land$
$(x_1=y_0 \land y_1=z_0 \land z_1=1) \land$
$(x_2=y_1 \land y_2=z_1 \land z_2=1)$

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

Satisfying assignment: **101 011 111**

This is a counterexample!

A remark

In order to describe a computation of length k by a propositional formula we need k+1 copies of the state variables.

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

Bounded model checking

- Can handle all of **LTL** formulas
- Can be used for **verification** by choosing k which is large enough
- Using such k is often **not practical** due to the size of the model

SAT-based verification

- Induction
- interpolation

Other solutions to the state explosion problem

- Abstraction
- Modular verification
- Partial order reduction
- Symmetry
- Distributed model checking

References

Model Checking

- Model checking
  E. Clarke, O. Grumberg, D. Peled,
  MIT Press, 1999.

Temporal Logic

- The Temporal Logic of Programs
  A. Pnueli, FOCS 1977

- **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

Thank you!