Formal Methods: Model Checking and Other Applications

Orna Grumberg
Technion, Israel

Marktoberdorf 2017

Outline

- Model checking of finite-state systems

- Assisting in program development
  - Program repair
  - Program differencing

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 propositional 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

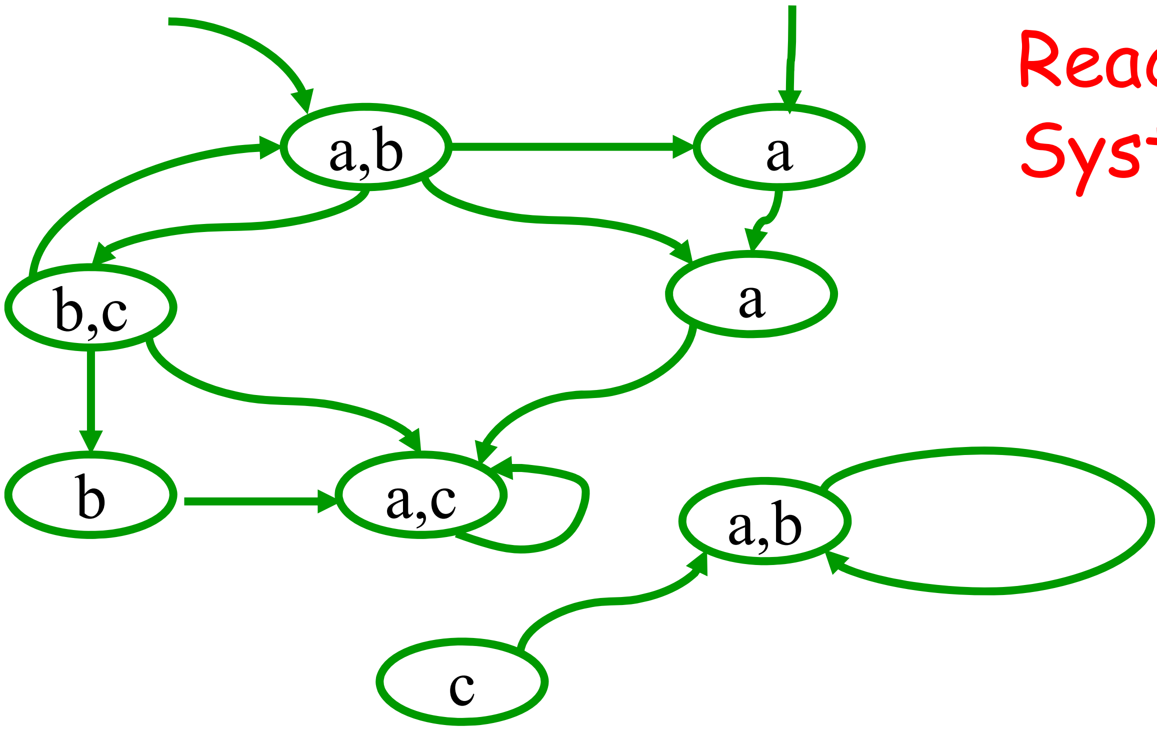
It returns

yes, if the system has the property

no + Counterexample, otherwise

Model of a system

Kripke structure / transition system



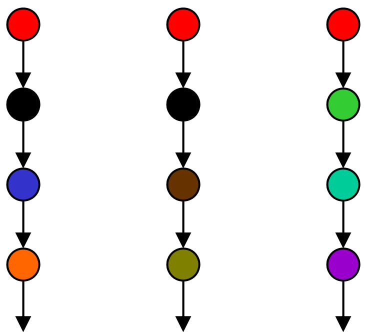
Reactive Systems

Labeled by **atomic propositions** ***AP***
(critical section, variable value…)

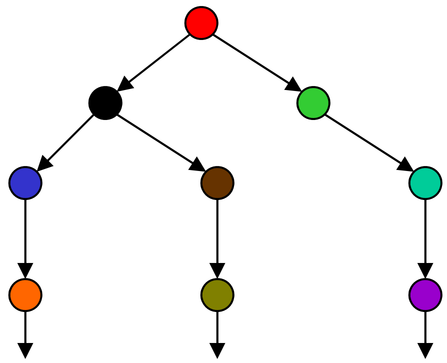
Temporal Logics

- Temporal Logics
  - Express properties of event orderings in time

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



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



Propositional temporal logic

AP – a set of atomic propositions

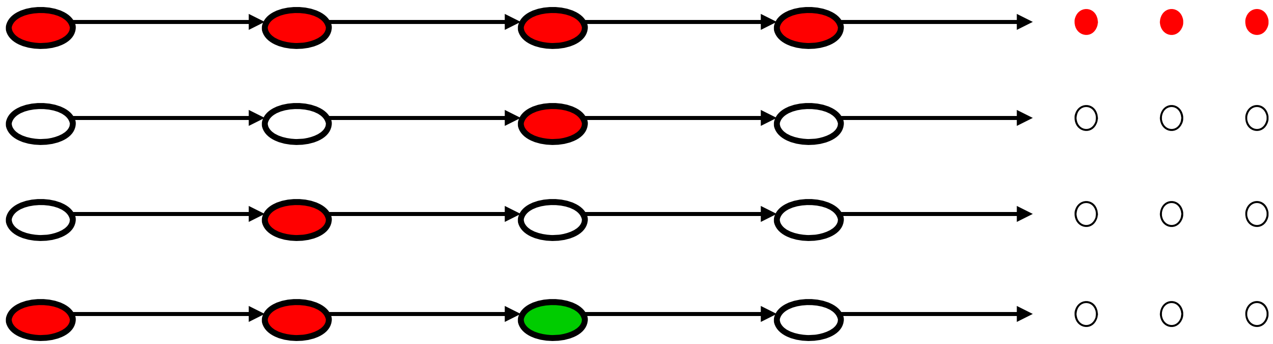
Temporal operators:

Gp

Fp

Xp

pUq



Path quantifiers: A for **all** path

E there **exists** a path

Model checking AG p on M

- Iteratively compute the sets S_j of states reachable from an initial state in j steps

- At each iteration check whether S_j contains a state satisfying $\neg p$
  - If so, declare a failure

- Terminate when all states were found

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

  - A fixpoint has been reached

Mutual Exclusion Example

- Two processes with a joint Boolean signal sem
- Each process P_i has a variable v_i describing its state:
  - $v_i = N$ Non critical
  - $v_i = T$ Trying
  - $v_i = C$ Critical

Mutual Exclusion Example

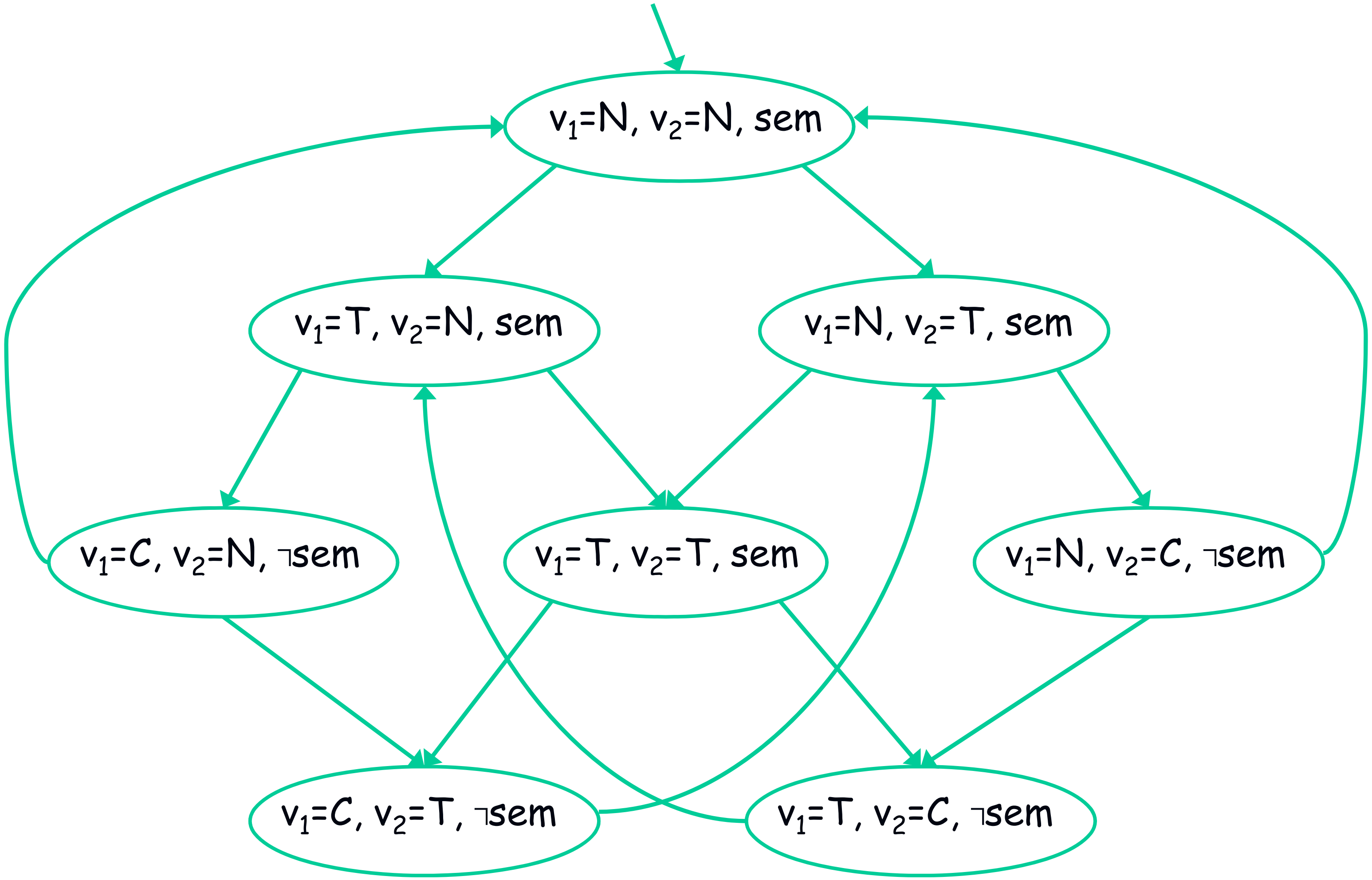
- Each process runs the following program:

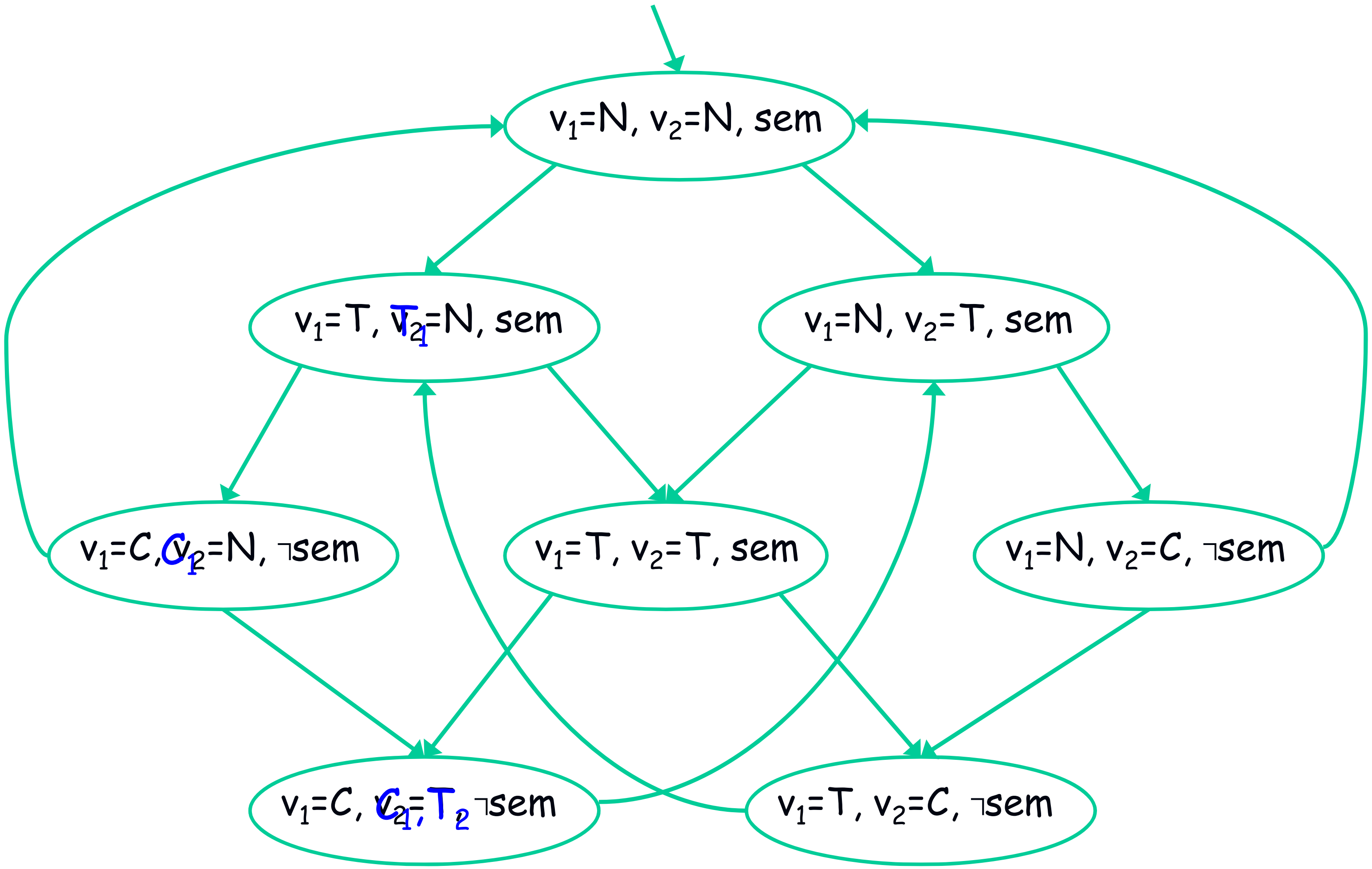
```
P_i ::  while (true) {
              if (v_i == N)  v_i = T;
              else if (v_i == T && sem)  { v_i = C; sem = 0;
        }
              else if (v_i == C)  {v_i = N; sem = 1; }
        }
```

Atomic action

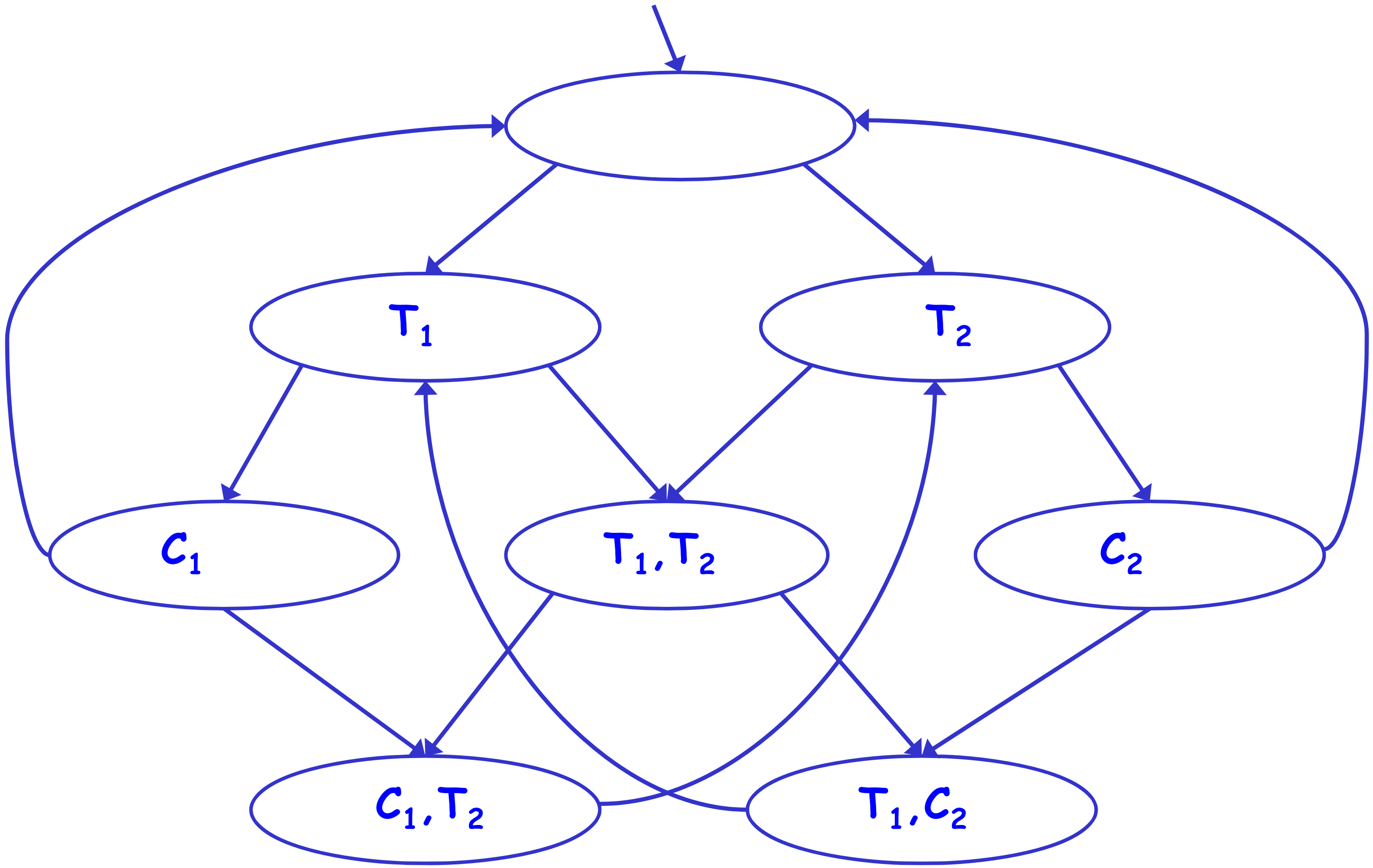
- The full program is: $P_1||P_2$
- Initial state: (v_1=N, v_2=N, sem)
- The execution is interleaving

Mutual Exclusion Example

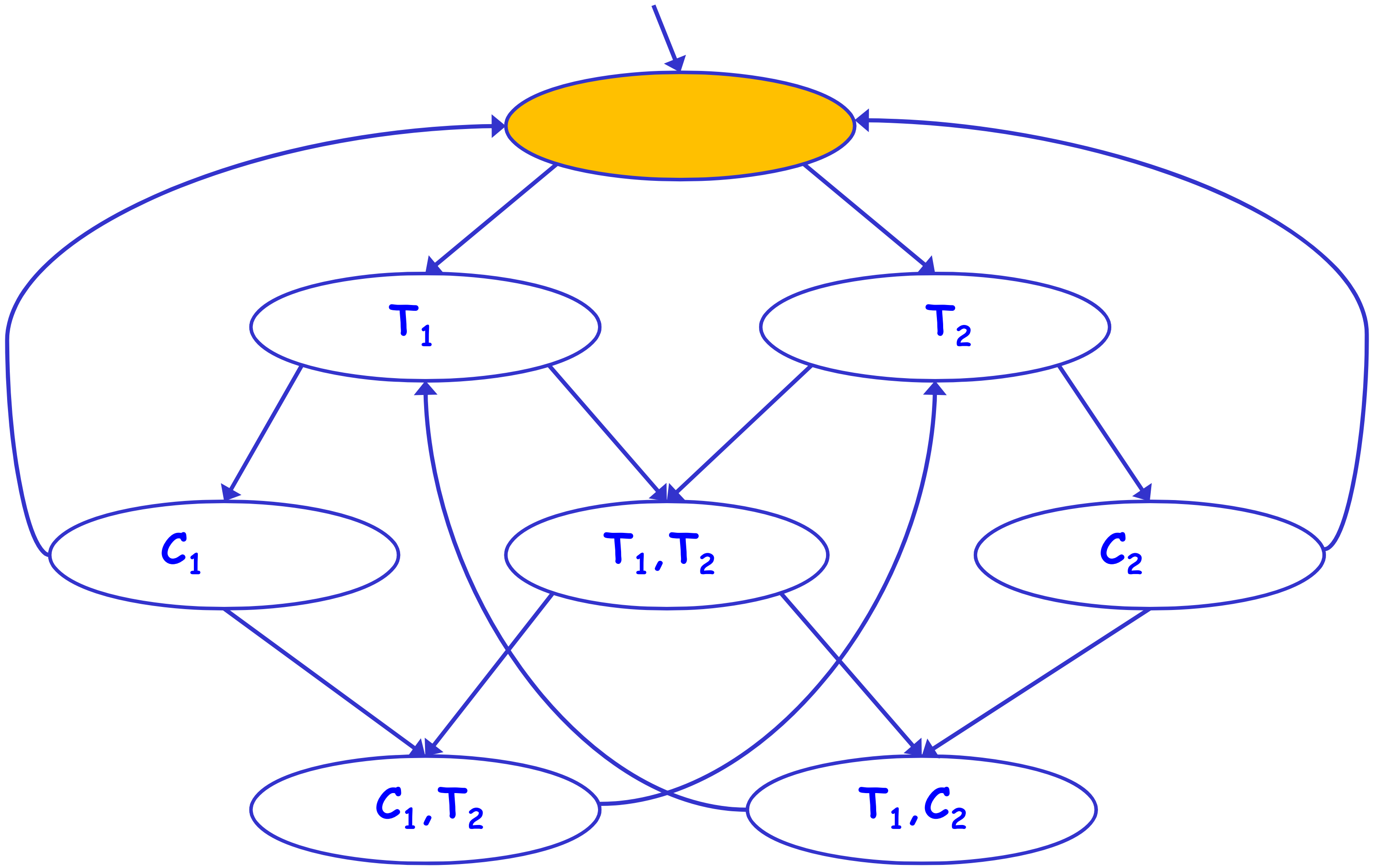




- We define atomic propositions: $AP=\{C_1, C_2, T_1, T_2)$
- A state is marked with T_i if $v_i=T$
- A state is marked with C_i if $v_i=C$

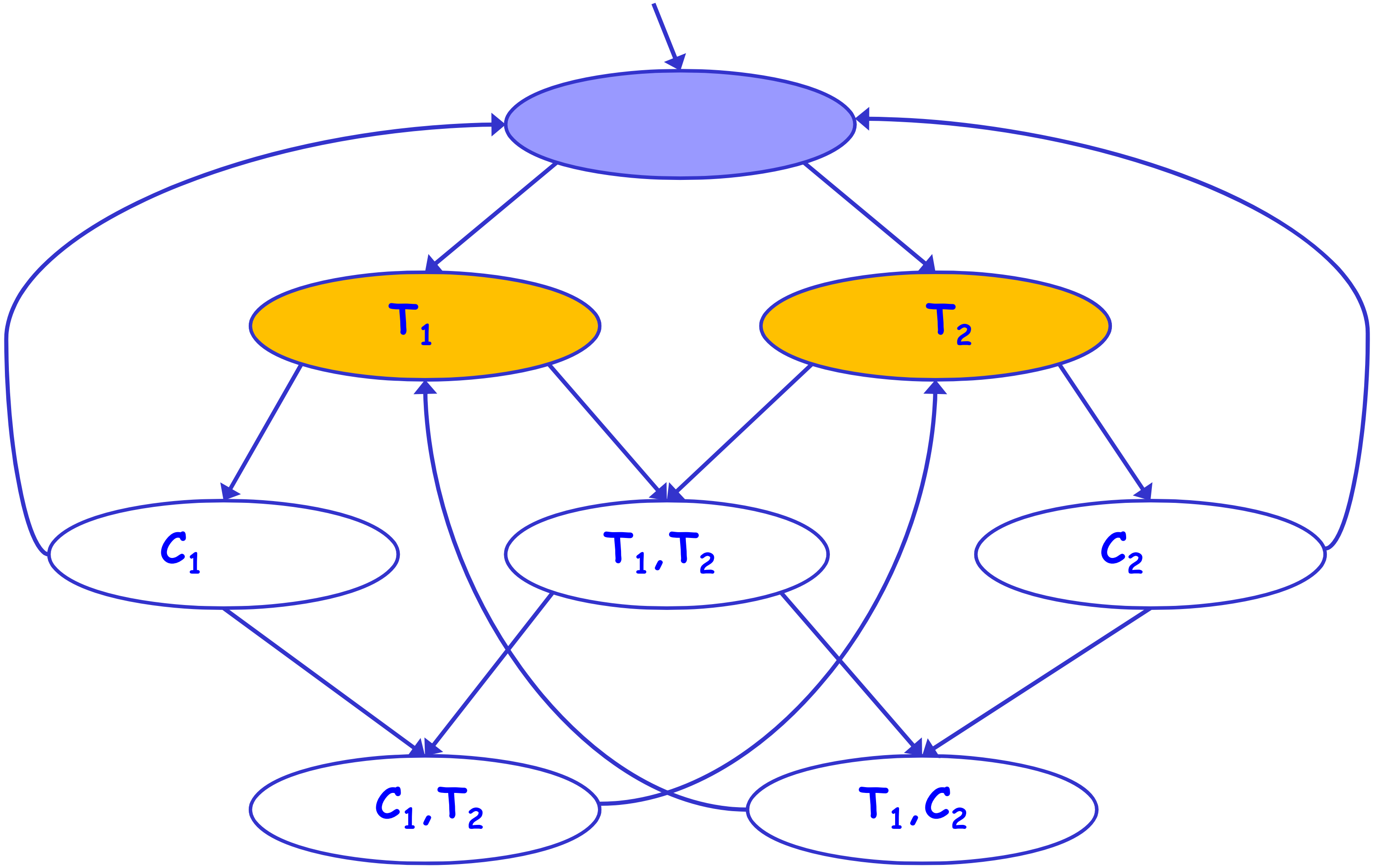


- Property 1: $\boldsymbol{AG}\neg(C_1 \land C_2)$



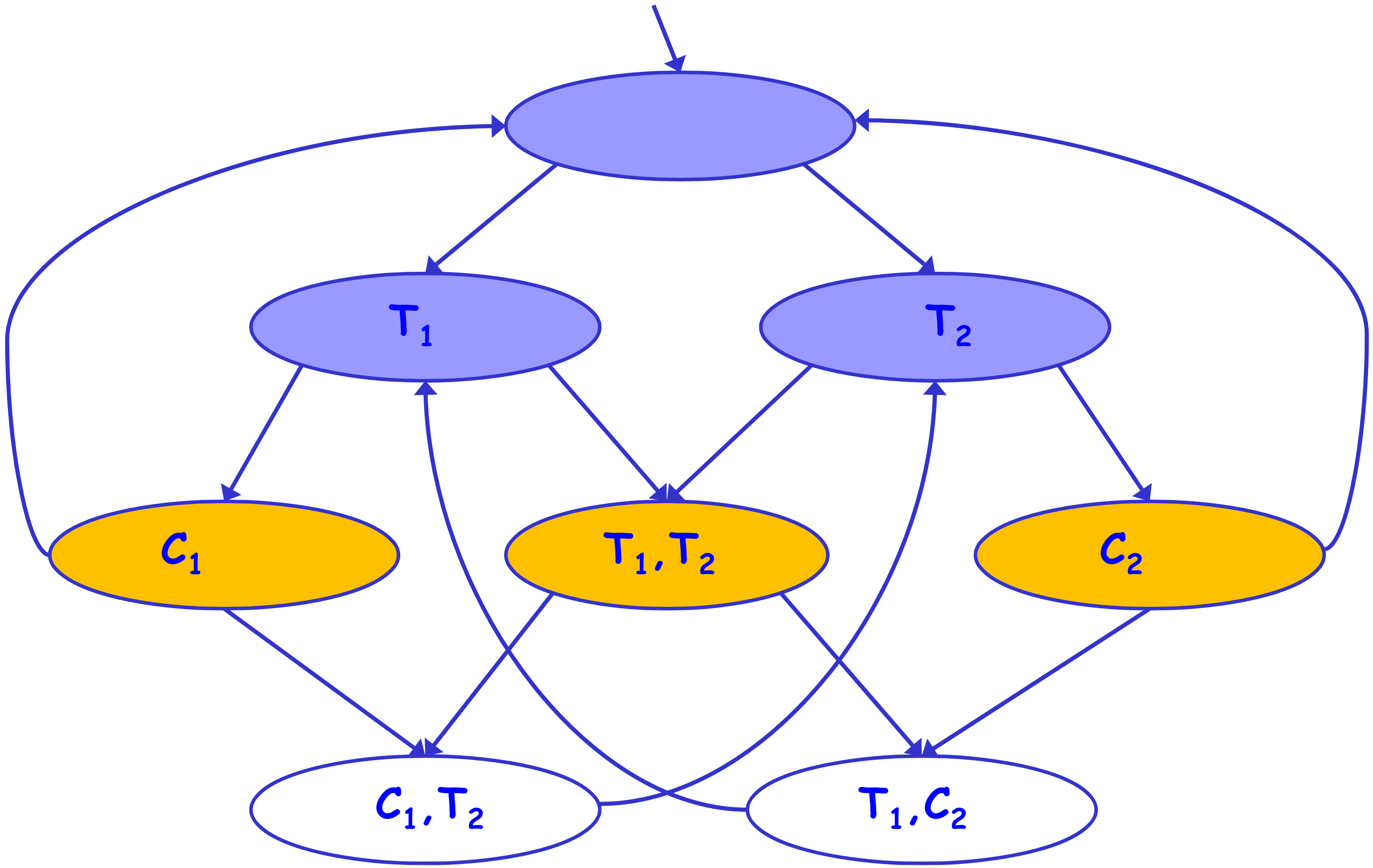
- Property 1: $\mathbf{AG}\neg(C_1 \wedge C_2)$

S_0



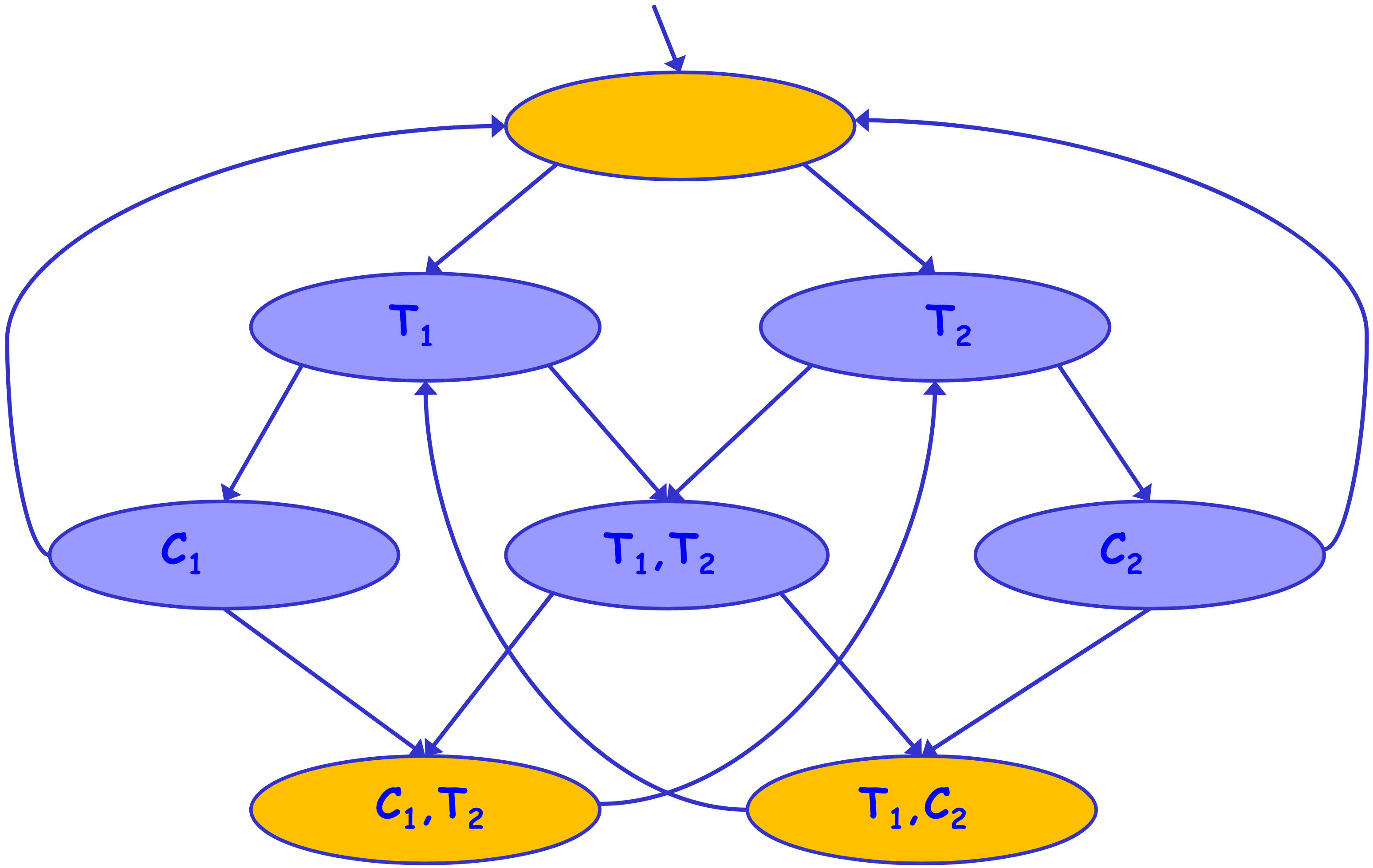
- Property 1: $\boldsymbol{AG}\neg(C_1 \wedge C_2)$

S_1



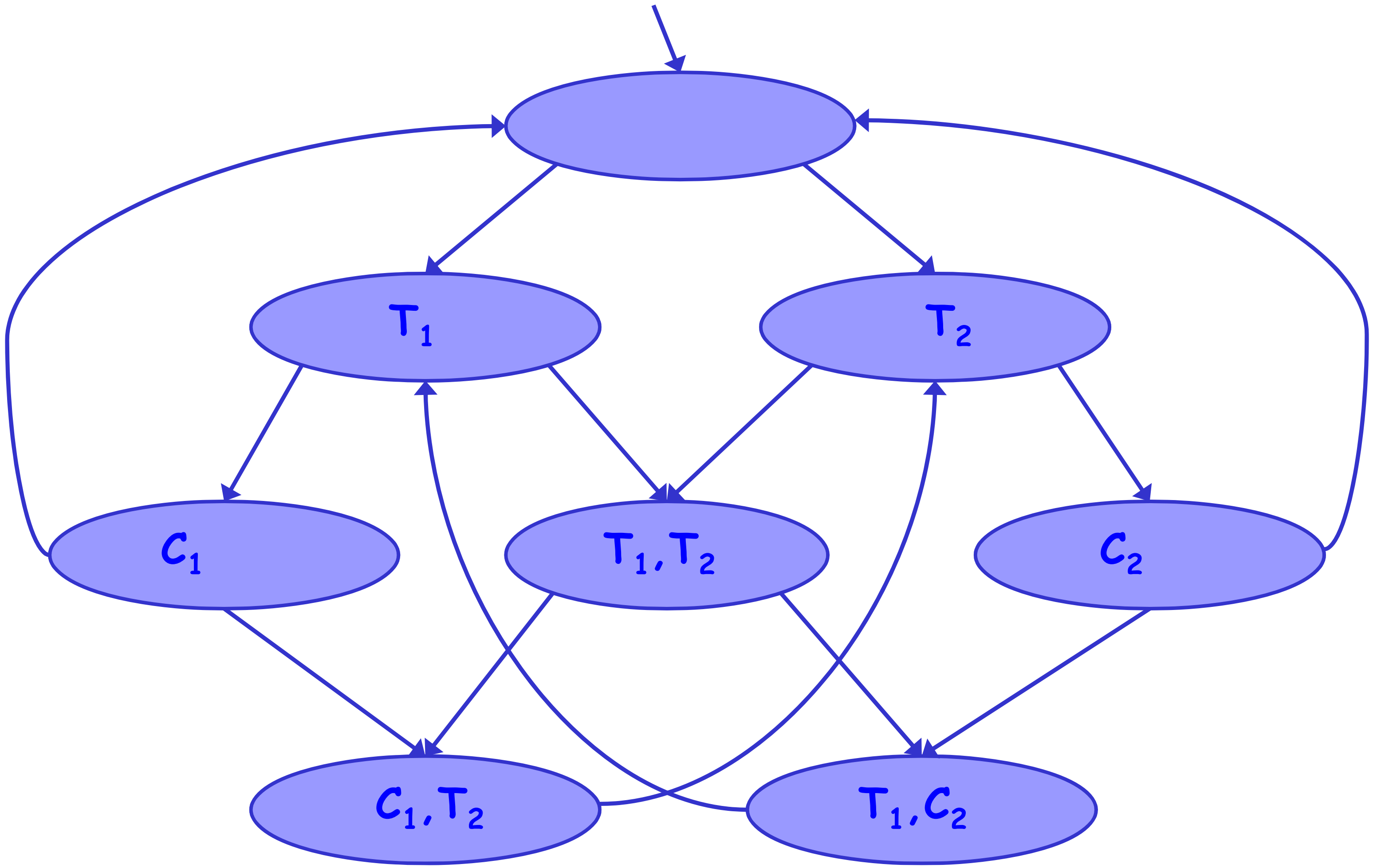
- Property 1: $\boldsymbol{AG}\neg(C_1 \wedge C_2)$

S_2



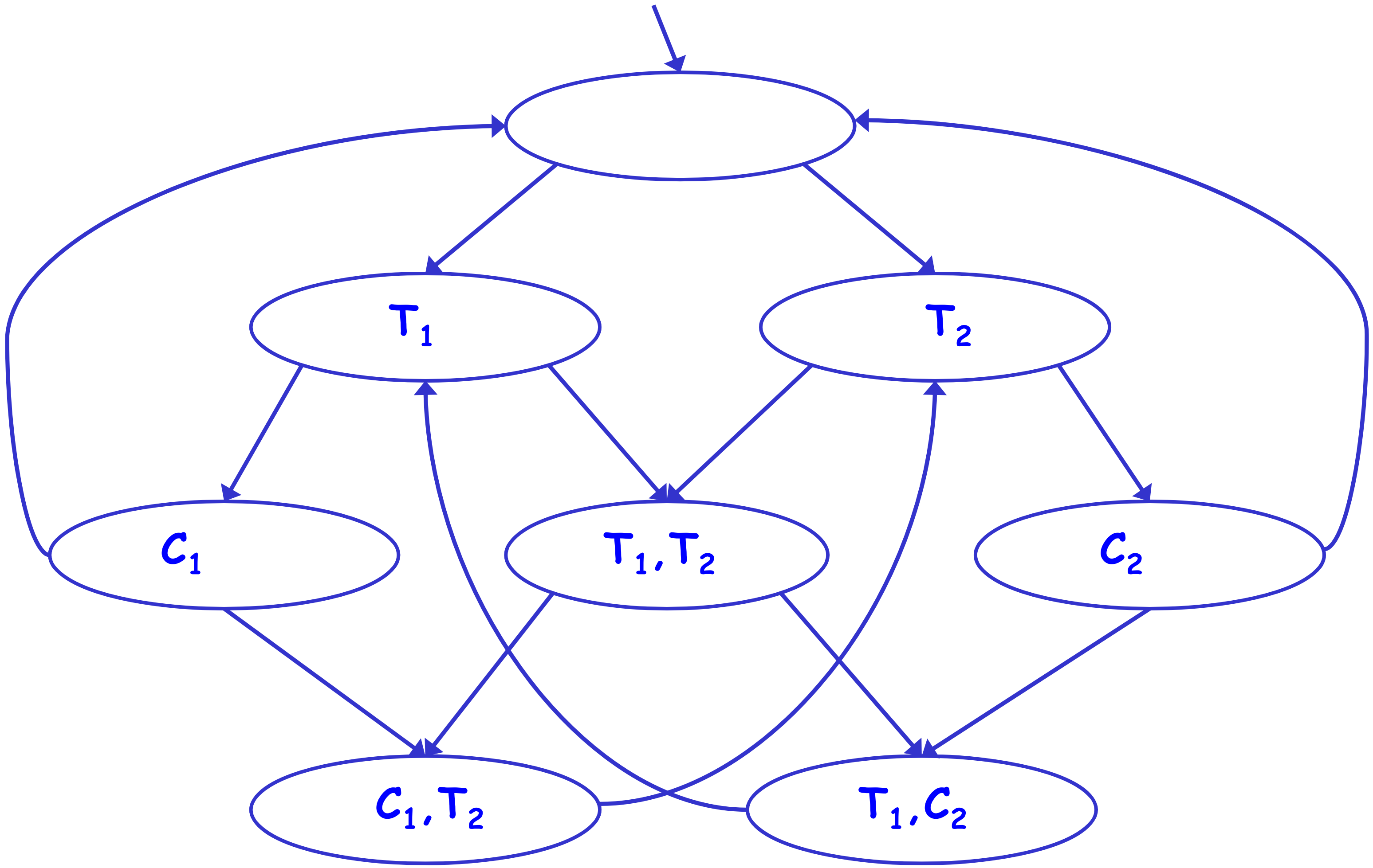
- Property 1: $\boldsymbol{AG}\neg(C_1 \wedge C_2)$

S_3

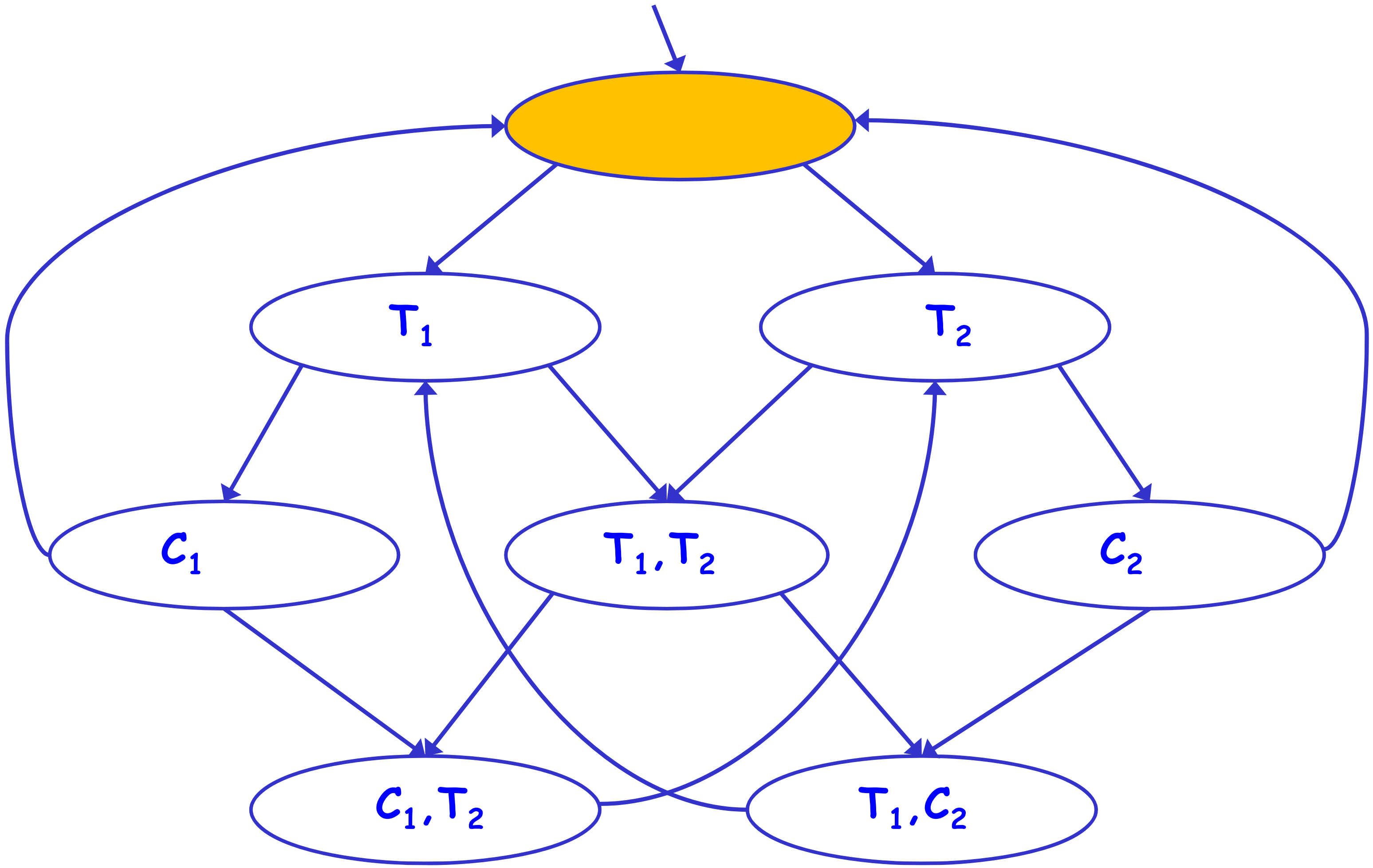


- $M \models AG \neg (C_1 \wedge C_2)$ ✓

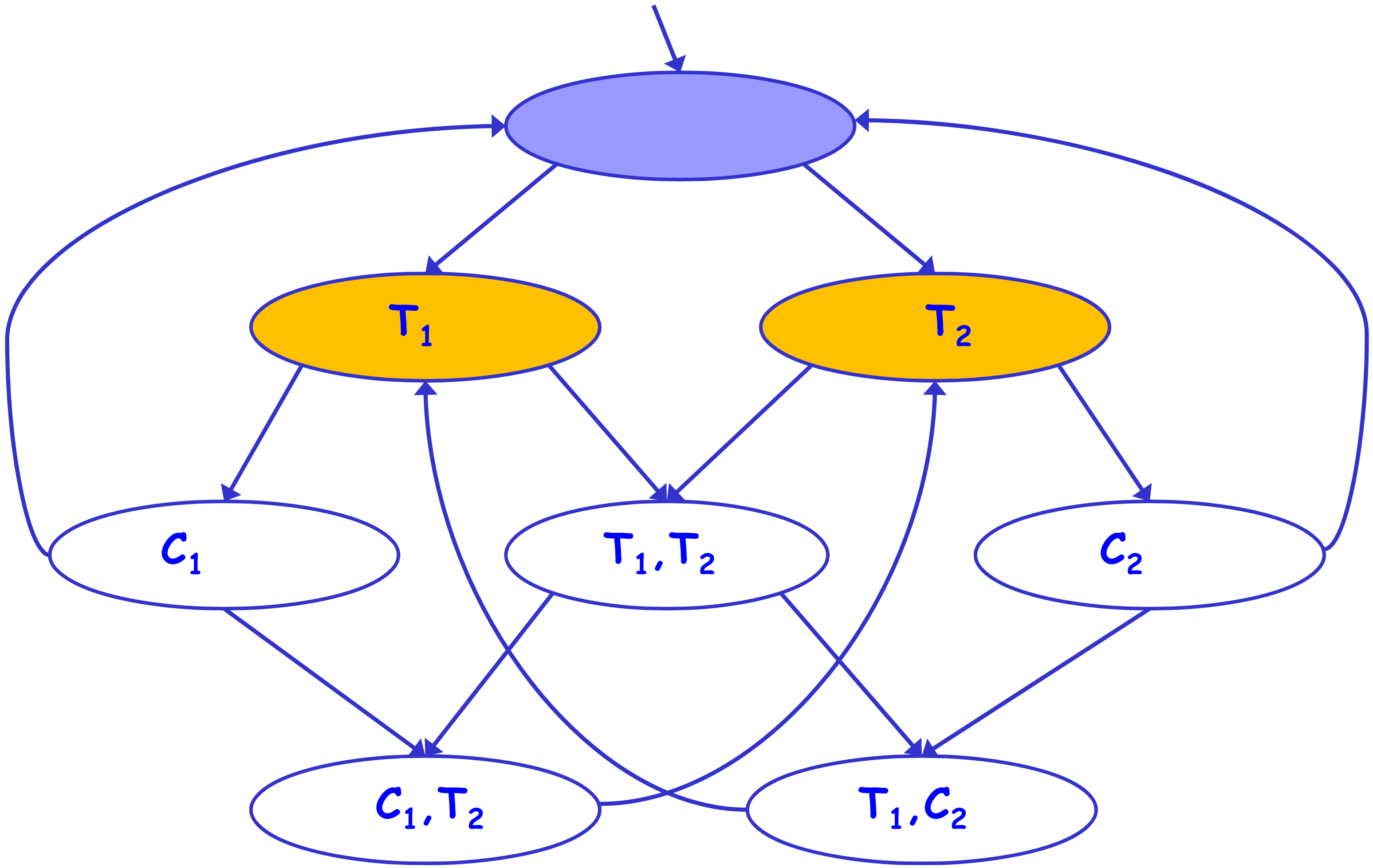
$S_4 \subseteq S_0 \cup S_1 \cup S_2 \cup S_3$



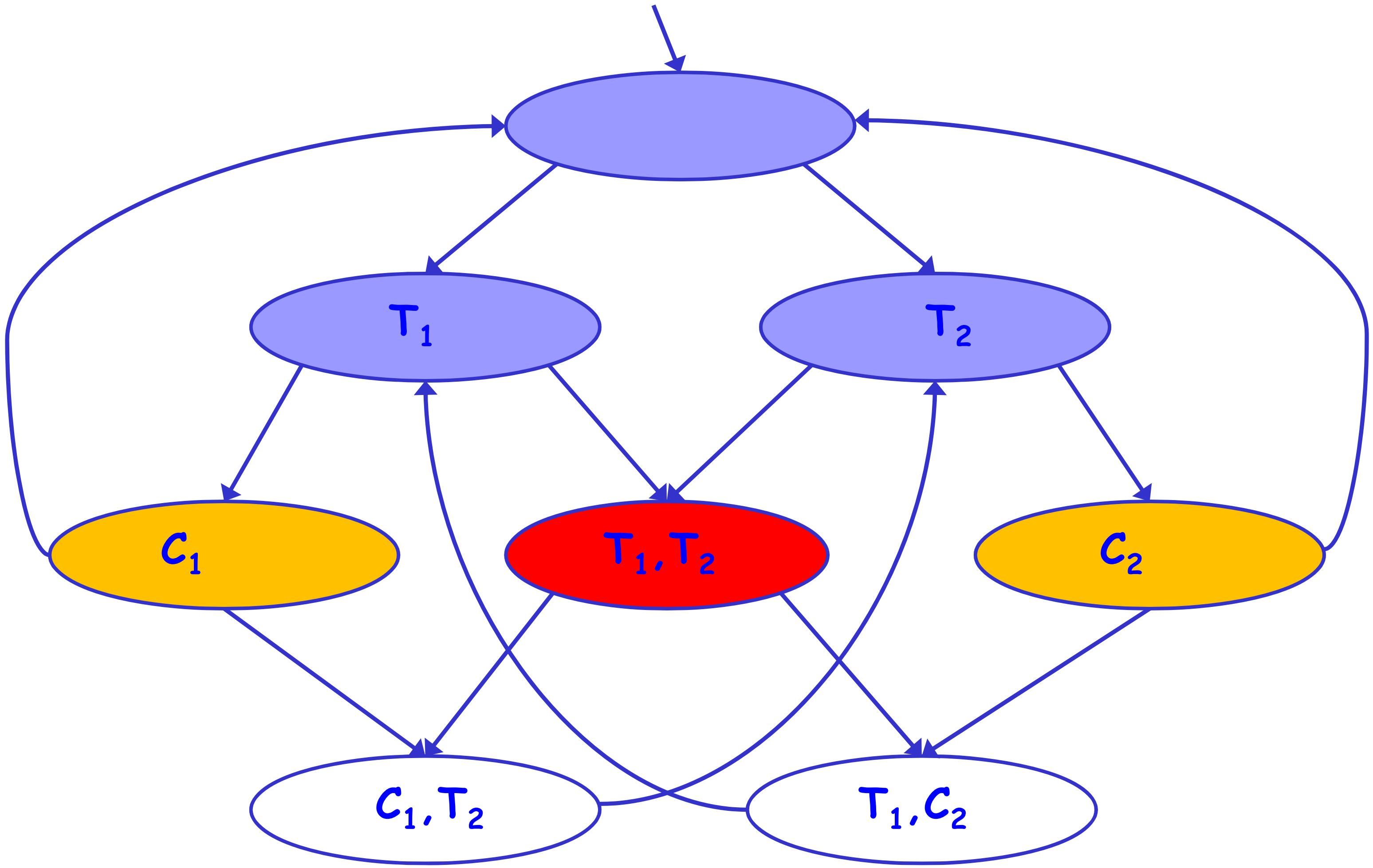
- Property 2: $AG\neg(T_1 \wedge T_2)$



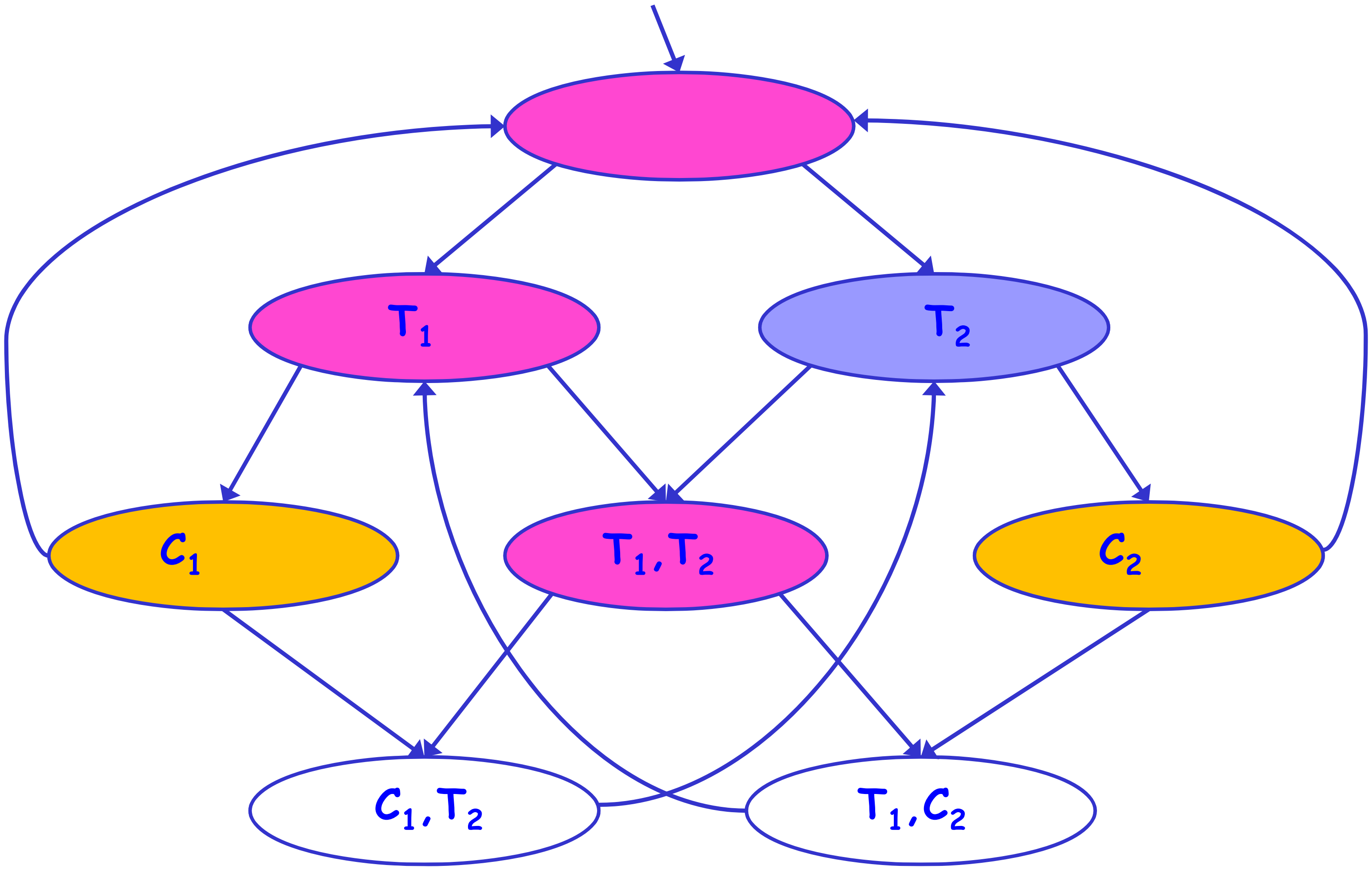
- Property 2: $AG\neg(T_1 \wedge T_2)$



- Property 2: $AG\neg(T_1 \land T_2)$



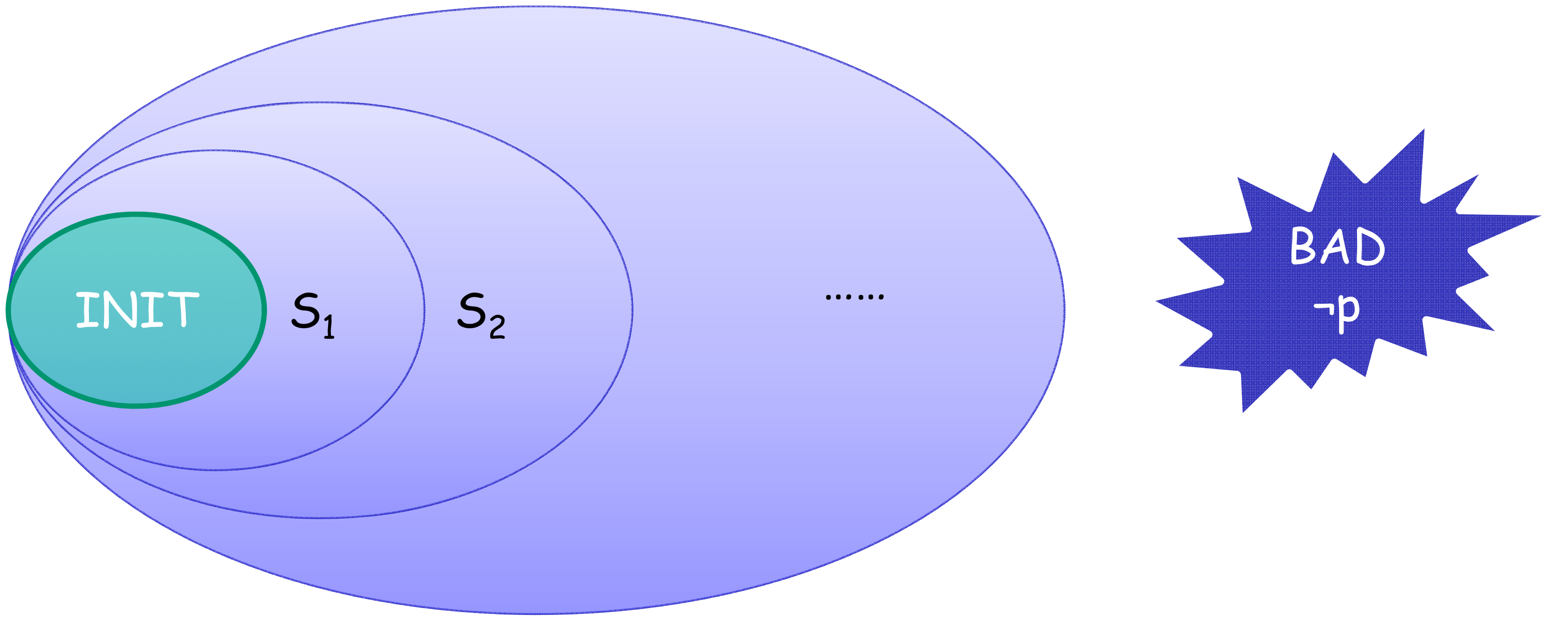
- $M \not\models AG\, \neg\, (T_1 \wedge T_2)$
- *A violating state has been found*



- $M \not\models AG\, \neg\, (T_1 \wedge T_2)$

Model checker returns a counterexample

Forward Reachability Analysis



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

Main limitation

The state explosion problem:

Space and time requirements grow with the size of the model

SAT-based model checking

- 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**.

Bounded model checking (BMC) for checking AGp

- Given
  - A finite system M
  - A safety property AGp
  - A bound k

- Determine
  - Does M contain a counterexample to AGp of *k transitions (or fewer)*?

Bounded Model Checking (BMC) for checking AGp

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

[BCCZ 99]

Bounded Model Checking

Terminates

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

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

BMC for checking AGp (EF$\neg$p)

Input to BMC:

A system over variables $V = \{v_1,\ldots,v_n\}$, where

- INIT(V) is a propositional formula representing the set of initial states
- R(V,V') is a propositional formula representing the transition relation

A specification:

- $\neg$p(V) is a propositional formula representing the set of states satisfying $\neg$p

- If $(f_M^k \wedge f_\varphi^k)$ is unsatisfiable:
  M has no counterexample of length k

- If $k = 2^{|V|}$ then we can conclude $M \models AGp$
  - Too big - not practical

- The method is suitable for refutation
  - Bug finding

BMC for checking $\varphi = \neg AGp \equiv EF\neg p$

- $f_M^k \; (V_0,\ldots,V_k) =$
  $INIT(V_0) \wedge R(V_0,V_1) \wedge \ldots \wedge R(V_{k-1},V_k)$

- Uses $k+1$ copies of $V = \{ v_1, \ldots, v_n \}$
- V_i represents the state after i transitions

BMC for checking $\varphi = EF\neg p$

- To check if p is violated within k steps:

$$f_{\varphi}^{k}(V_0,\ldots,V_k) =$$
$$\neg p(V_0) \vee \ldots \vee \neg p(V_k) = \bigvee_{i=0\cdots k} \neg p(V_i)$$

BMC for checking φ=EF¬p

- The iterative algorithm:

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

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

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

.

.

.

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

Example - shift register of $\langle x,y,z \rangle$

The set of states: all valuations of $\langle x,y,z \rangle$

Transition relation:

$$T(x,y,z,x',y',z') = \quad x'=y \ \wedge \ y'=z \ \wedge \ \underbrace{z'=1}_{\text{error}}$$

Initial condition:

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

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

Propositional formula for k=2

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

INIT = $x{=}0 \vee y{=}0 \vee z{=}0$
R = $x'{=}y \;\wedge\; y'{=}z \;\wedge\; z'{=}1$

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

p = $x{=}0 \vee y{=}0 \vee z{=}0$

Satisfying assignment: **101 011 111**

This is a counterexample!

Verification with SAT solvers

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

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