# Symbolic Model Checking

*M. Pistore and M. Roveri*

IIT Delhi - India
Feb 6, 2004

# Acknowledgements


- These slides are the result of the work of the following members of the NuSMV team:
  - Alessandro Cimatti
  - Marco Pistore
  - Marco Roveri
  - Roberto Sebastiani


- For comments or questions, please contact:
  - Marco Pistore (`pistore@dit.unitn.it`)
  - Marco Roveri (`roveri@irst.itc.it`)
- For information on the NuSMV model checker:
  - `http://nusmv.irst.itc.it/`

# Introduction

*– Symbolic Model Checking–*

*M. Pistore and M. Roveri*

IIT Delhi - India, Feb 6, 2004

# Formal Verification

- The design and implementation **correct** software (and hardware) is a difficult task.
- In some domains, errors are both difficult to detect using standard testing techniques and very expensive:
  - Intel Pentium bug
  - long list of space missions failed due to software problems
  - ...
- In these domains, **Formal Verification** techniques are of help:
  - the correctness of the (software or hardware) system mathematically proven.
- We concentrate on a specific Formal Verification technique, namely **Model Checking**.

# Model Checking

Basic procedure:

- describe the system as Finite State Model (a Kripke model in our case).
- express properties in Temporal Logic.
- formal V&V by automatic exhaustive search over the state space.

Drawback:

- State space explosion.
- Expressiveness – hard to deal with parametrized systems.

Industrial Success:

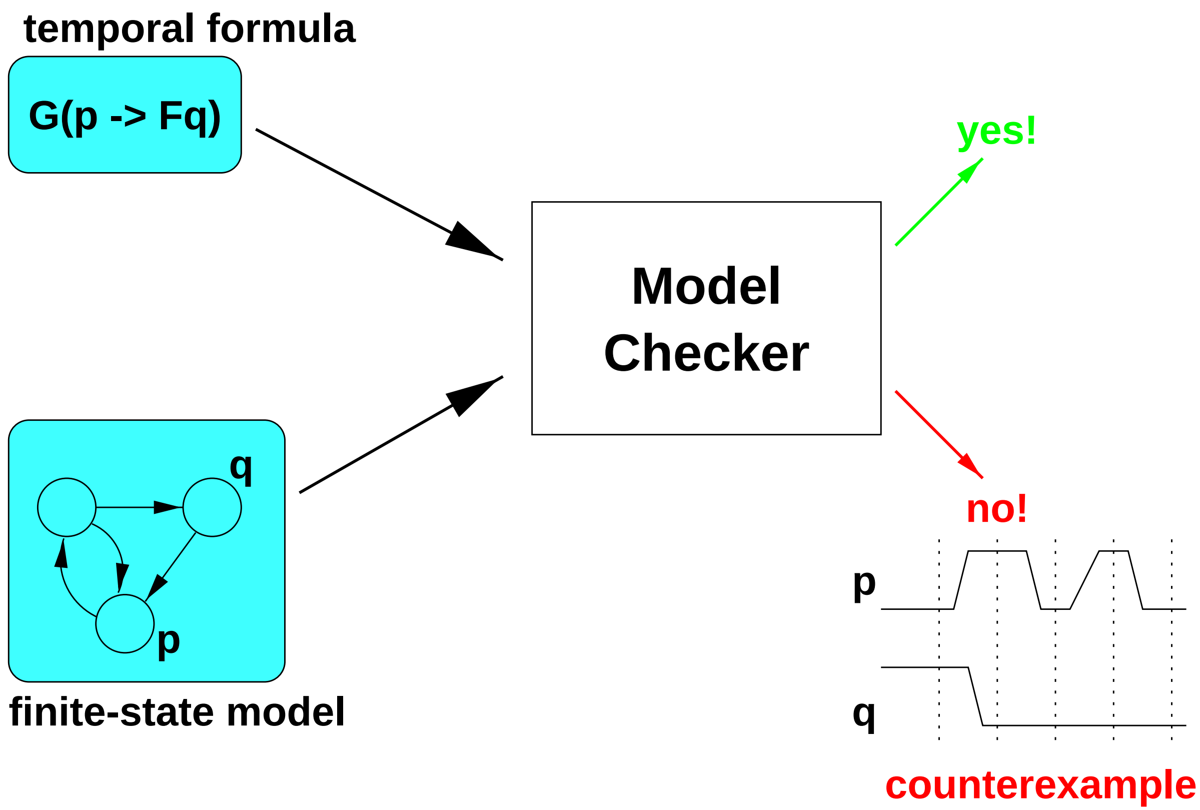
- From academics to industry in a decade.
- Powerful debugging capabilities.
- Easier to integrate within industrial development cycle.

# What is a Model Checker

A model checker is a software tool that

- given a description of a Kripke model $M$ ...
- ... and a property $\Phi$,
- decides whether $M \models \Phi$,
- returns “yes” if the property is satisfied,
- otherwise returns “no”, and provides a counterexample.

# What is a Model Checker



# Plan

- **Today: Symbolic Model Checking**
  - Models for Reactive Systems: Kripke Structures
  - Properties of Reactive Systems: CTL, LTL
  - Symbolic Model Checking Techniques: BDD-based and SAT-based techniques

- **Next Monday: The NuSMV Model Checker**
  - The NuSMV Open Source project
  - The SMV language

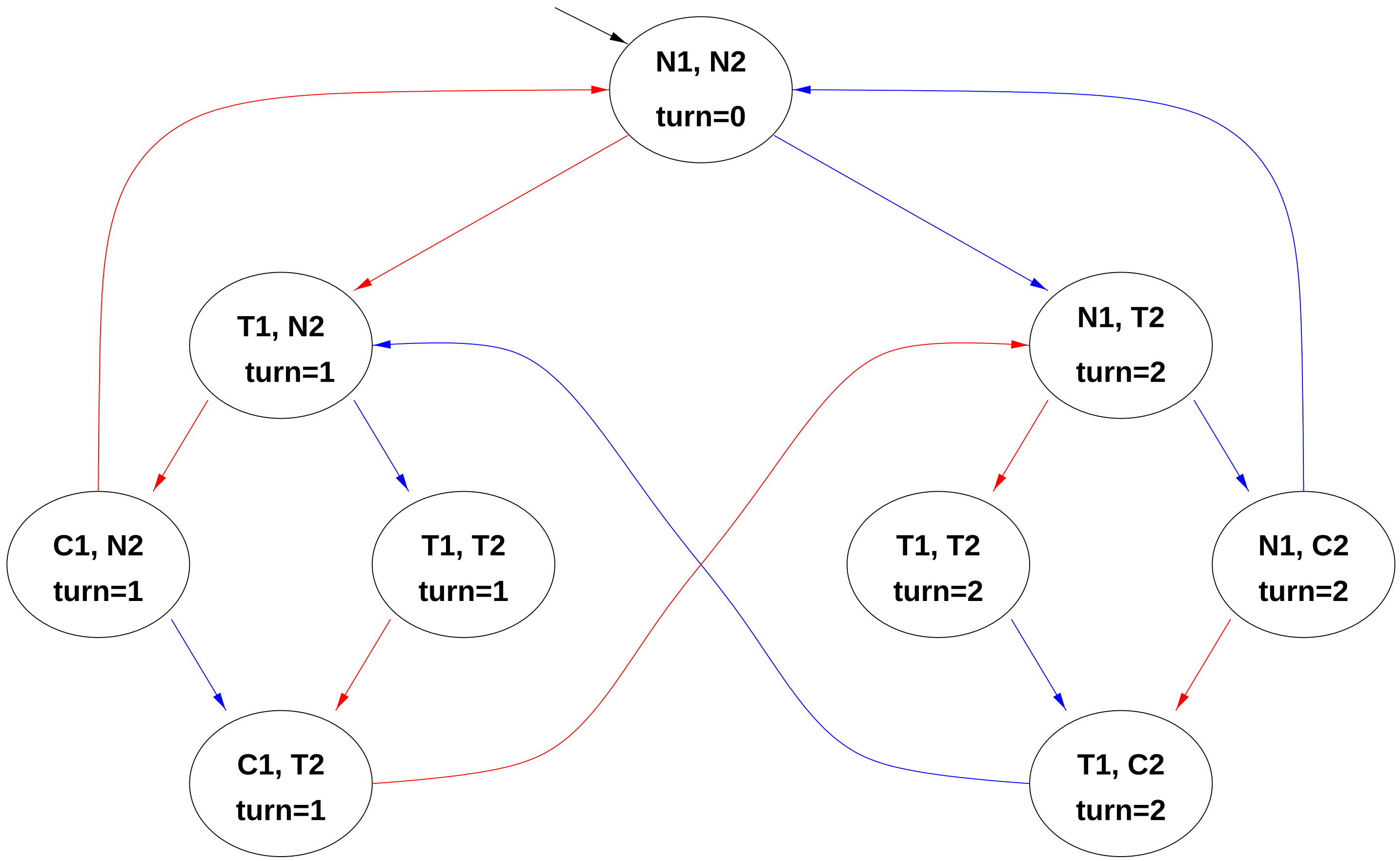
# Symbolic Model Checking

*– Symbolic Model Checking–*

*M. Pistore and M. Roveri*

IIT Delhi - India, Feb 6, 2004

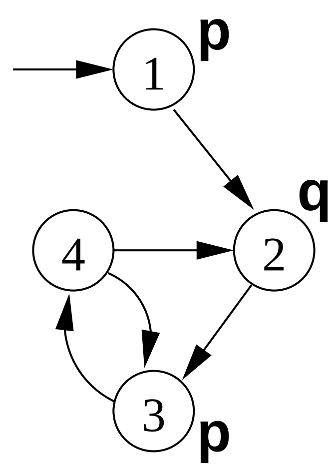
# A Kripke model for mutual exclusion




N = noncritical, T = trying, C = critical User 1 User 2

# Modeling the system: Kripke models

- Kripke models are used to describe reactive systems:
  - nonterminating systems with infinite behaviors,
  - e.g. communication protocols, operating systems, hardware circuits;
  - represent dynamic evolution of modeled systems;
  - values to state variables, program counters, content of communication channels.
- Formally, a Kripke model $(S, R, I, L)$ consists of
  - a set of **states** $S$;
  - a set of **initial states** $I \subseteq S$;
  - a set of **transitions** $R \subseteq S \times S$;
  - a **labeling** $L \subseteq S \times AP$.

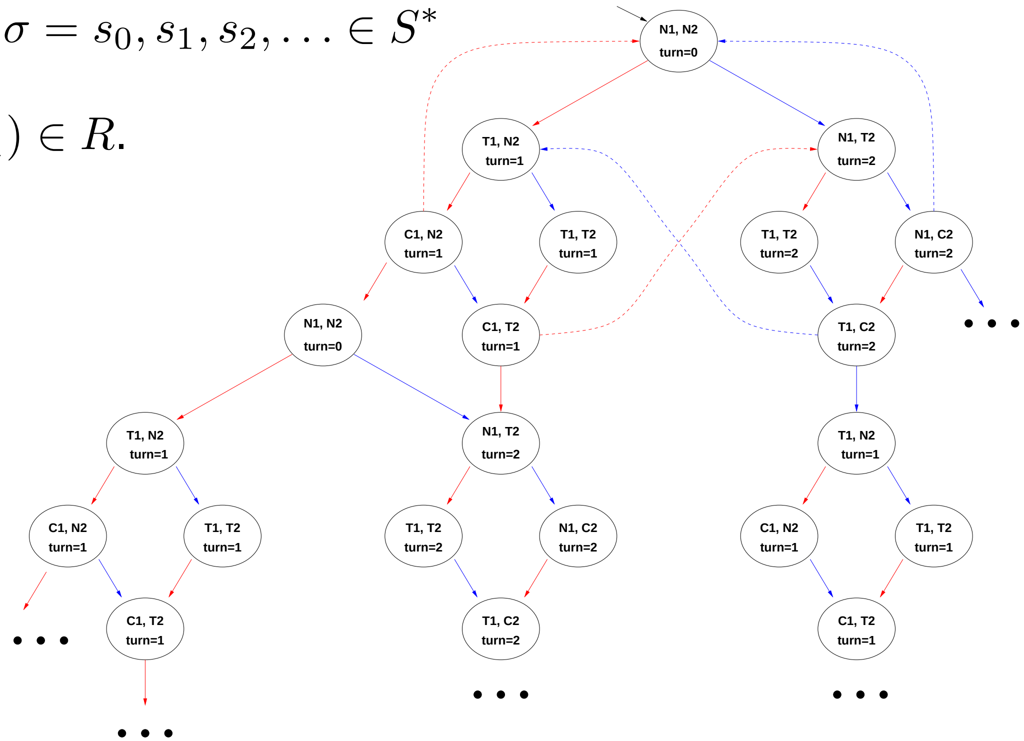


# Path in a Kripke Model

- A path in a Kripke model $M$ is an infinite sequence

$$\sigma = s_0, s_1, s_2, \ldots \in S^*$$

such that $s_0 \in I$ and $(s_i, s_{i+1}) \in R$.




- A state $s$ is reachable in $M$ if there is a path from the initial states to $s$.

# Description languages for Kripke Model

A Kripke model is usually presented using a **structured programming language**.

**Each component** is presented by specifying

- state variables: determine the state space $S$ and the labeling $L$.
- initial values for state variables: determine the set of initial states $I$.
- instructions: determine the transition relation $R$.

Components can be combined via

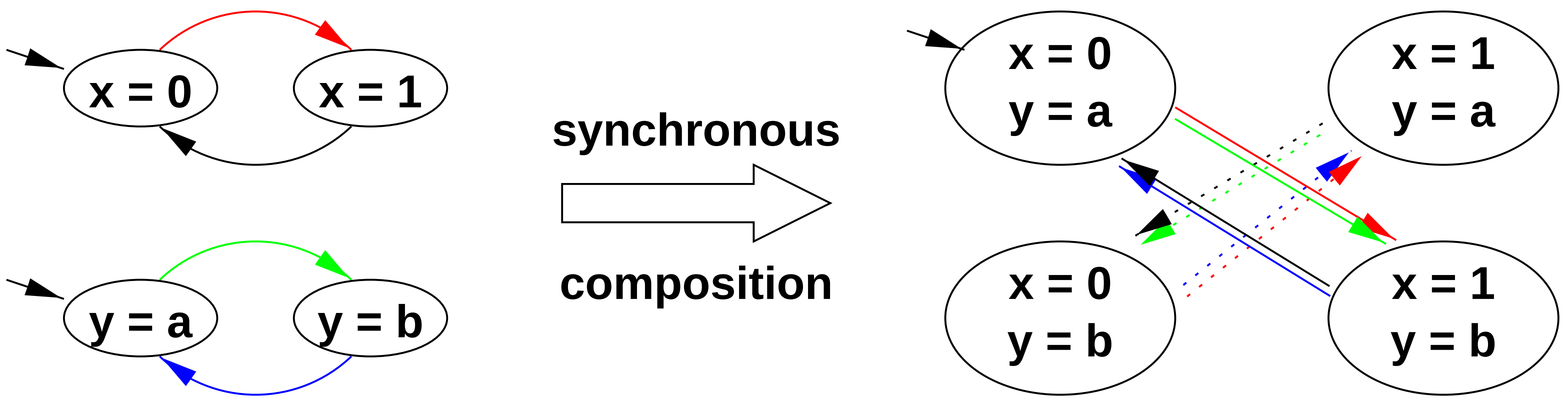
- **synchronous composition**,
- **asynchronous composition**.

**State explosion** problem in model checking:

- linear in model size, but model is exponential in number of components.

# Synchronous Composition

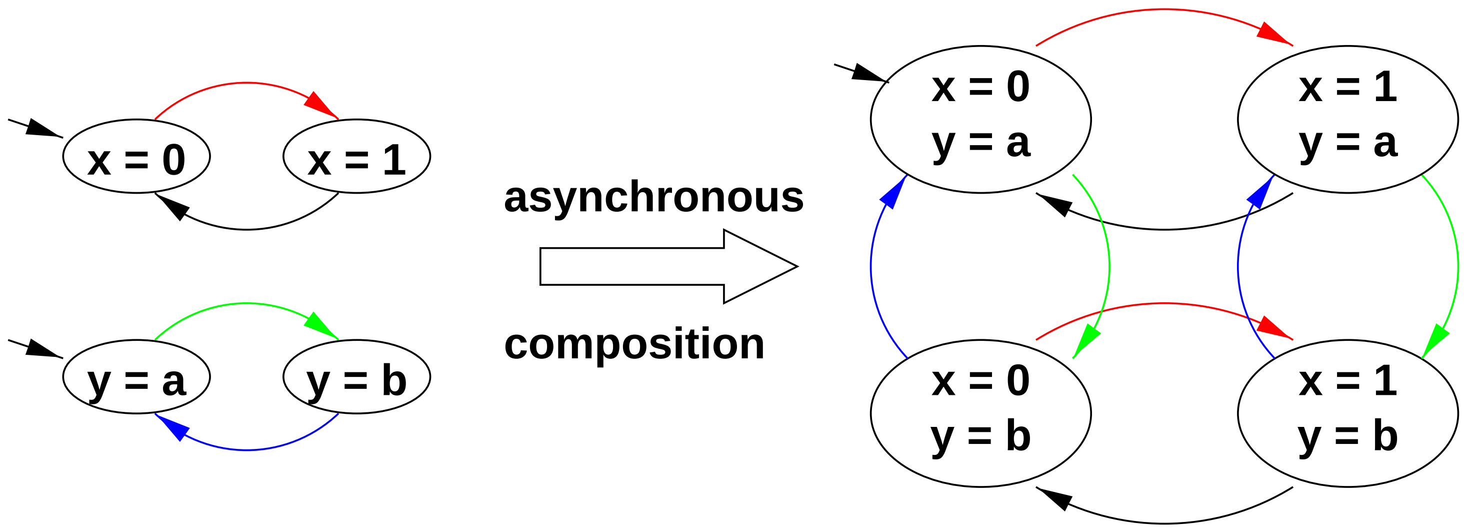
- Components evolve in parallel.
- At each time instant, every component performs a transition.




- Typical example: sequential hardware circuits.
- Synchronous composition is the default in NuSMV.

# Asynchronous Composition

- Interleaving of evolution of components.
- At each time instant, one component is selected to perform a transition.

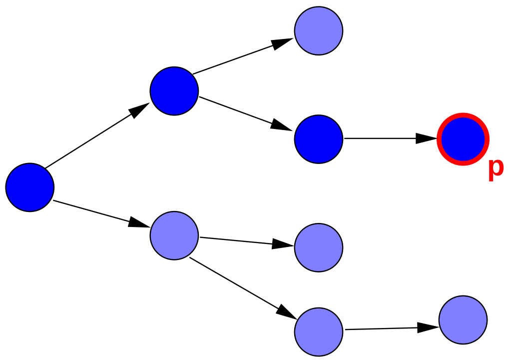



- Typical example: communication protocols.
- Asynchronous composition can be represented with NuSMV processes.

# Properties of Reactive Systems (I)

**Safety properties**:

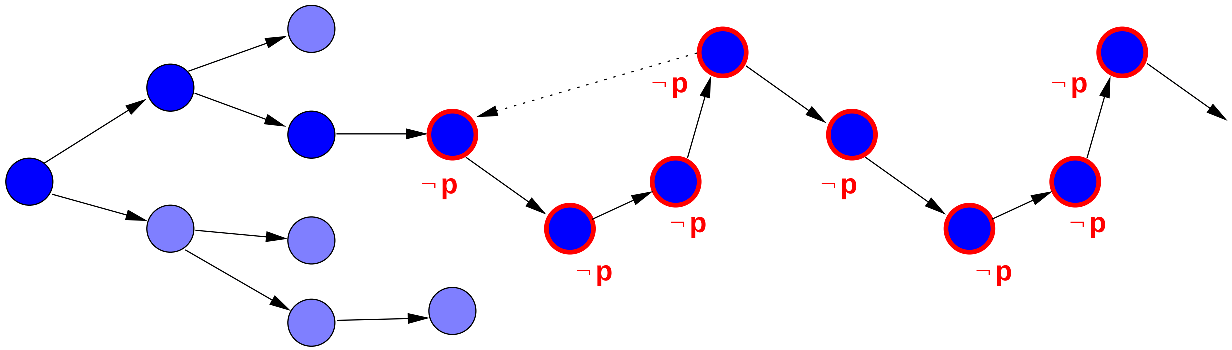
- nothing bad ever happens
  - deadlock: two processes waiting for input from each other, the system is unable to perform a transition.
  - a state is reached that satisfies a “bad” condition, e.g. two process in critical section at the same time
- can be refuted by a finite behaviour
- it is never the case that $p$.



# Properties of Reactive Systems (II)

**Liveness properties**:

- Something desirable will eventually happen
  - whenever a subroutine takes control, it will always return it (sooner or later)
- can be refuted by infinite behaviour
  - a subroutine takes control and never returns it



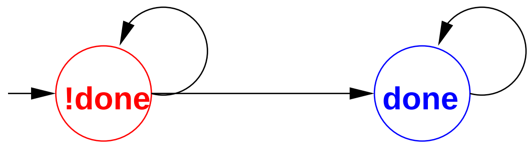

  - an infinite behaviour can be presented as a loop

# Temporal Logics

- Express properties of “Reactive Systems”
  - nonterminating behaviours,
  - without explicit reference to time.
- Linear Time Temporal Logic (LTL)
  - intepreted over each path of the Kripke structure
  - linear model of time
  - temporal operators
- Computation Tree Logic (CTL)
  - intepreted over computation tree of Kripke model
  - branching model of time
  - temporal operators plus path quantifiers

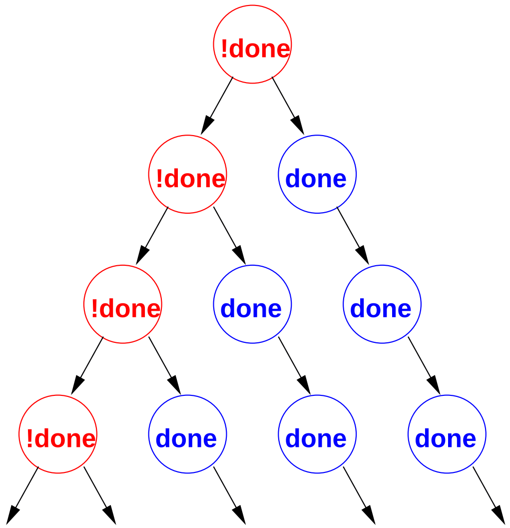
# Computation tree vs. computation paths

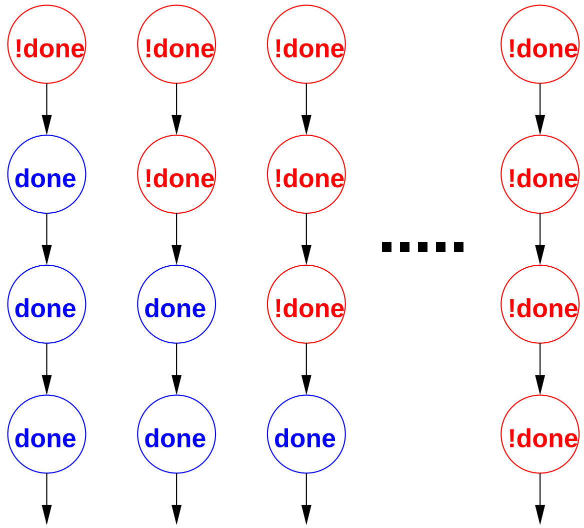
☞ Consider the following Kripke structure:




☞ Its execution can be seen as:

- an infinite **computation tree**
- a set of infinite **computation paths**



# Linear Time Temporal Logic (LTL)

LTL properties are evaluated over paths, i.e., over infinite, linear sequences of states:

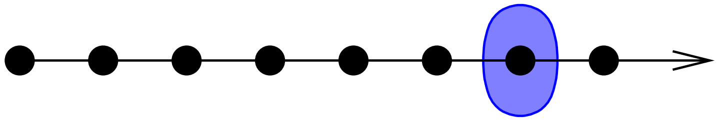
$$s[0] \rightarrow s[1] \rightarrow \cdots \rightarrow s[t] \rightarrow s[t+1] \rightarrow \cdots$$

LTL provides the following temporal operators:

- "**Finally**" (or "future"): $Fp$ is true in $s[t]$ iff $p$ is true in **some** $s[t']$ with $t' \geq t$
- "**Globally**" (or "always"): $Gp$ is true in $s[t]$ iff $p$ is true in **all** $s[t']$ with $t' \geq t$
- "**Next**": $Xp$ is true in $s[t]$ iff $p$ is true in $s[t+1]$
- "**Until**": $pUq$ is true in $s[t]$ iff
  - $q$ is true in some state $s[t']$ with $t' \geq t$
  - $p$ is true in all states $s[t'']$ with $t \leq t'' < t'$

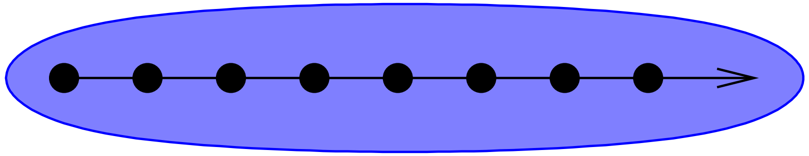
# LTL

finally p



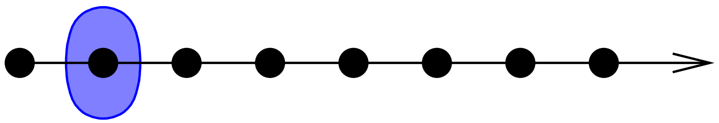
**F p**

globally p



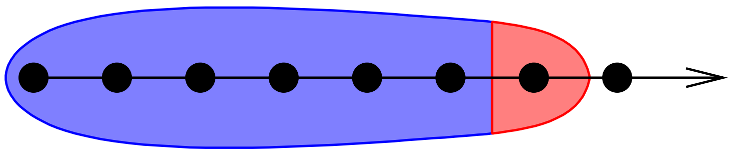
**G p**

next p



**X p**

p until q



**p U q**

# LTL: Examples

- **Safety**: “it never happens that a train arrives and the bar is up”

$$G\neg(\text{train-arrives} \wedge \text{bar-up})$$

- **Liveness**: “if input, then eventually output”

$$G(\text{input} \rightarrow F\,\text{output})$$

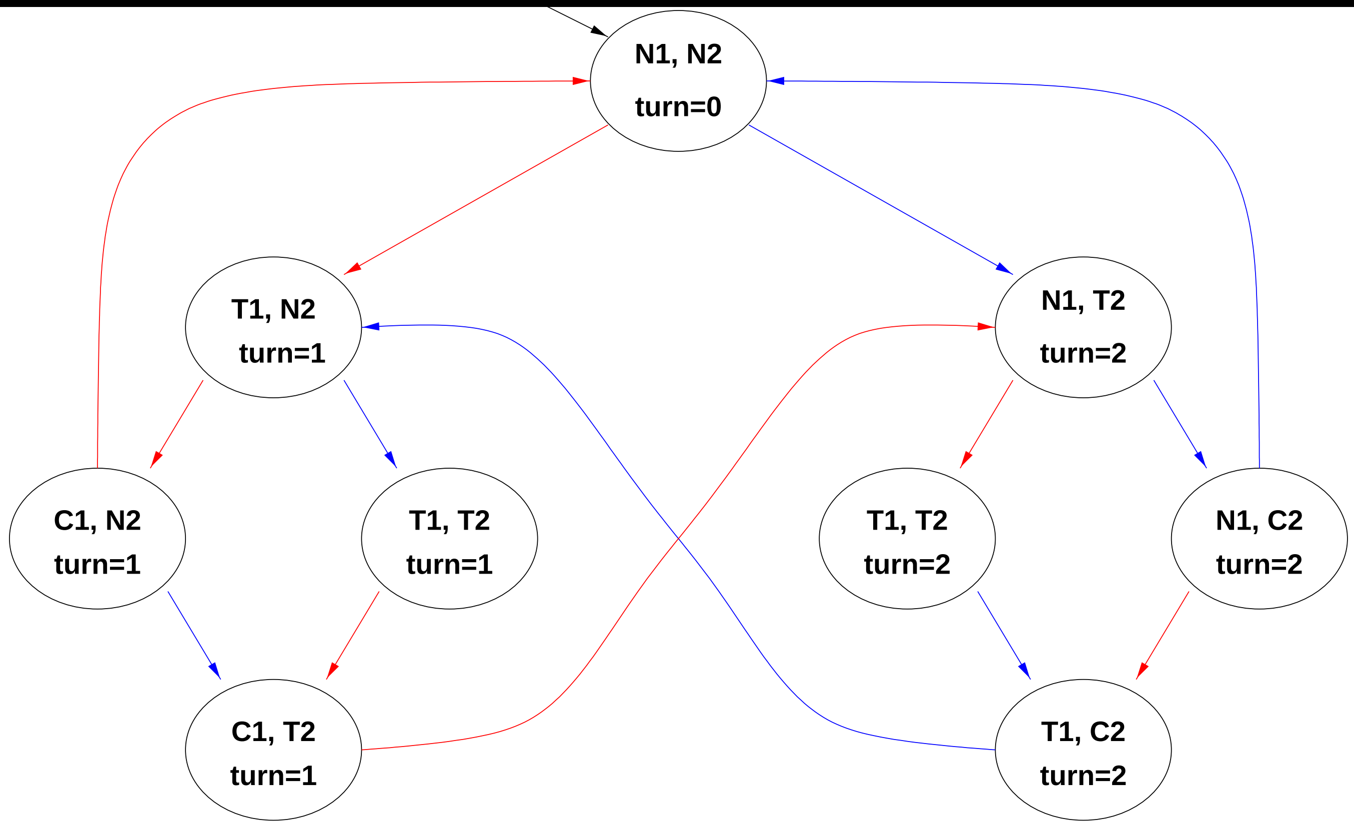
- **Fairness**: “infinitely often send”

$$G\,F\,\text{send}$$

- **Strong fairness**: “infinitely often send implies infinitely often recv.”

$$G\,F\,\text{send} \rightarrow G\,F\,\text{recv}$$

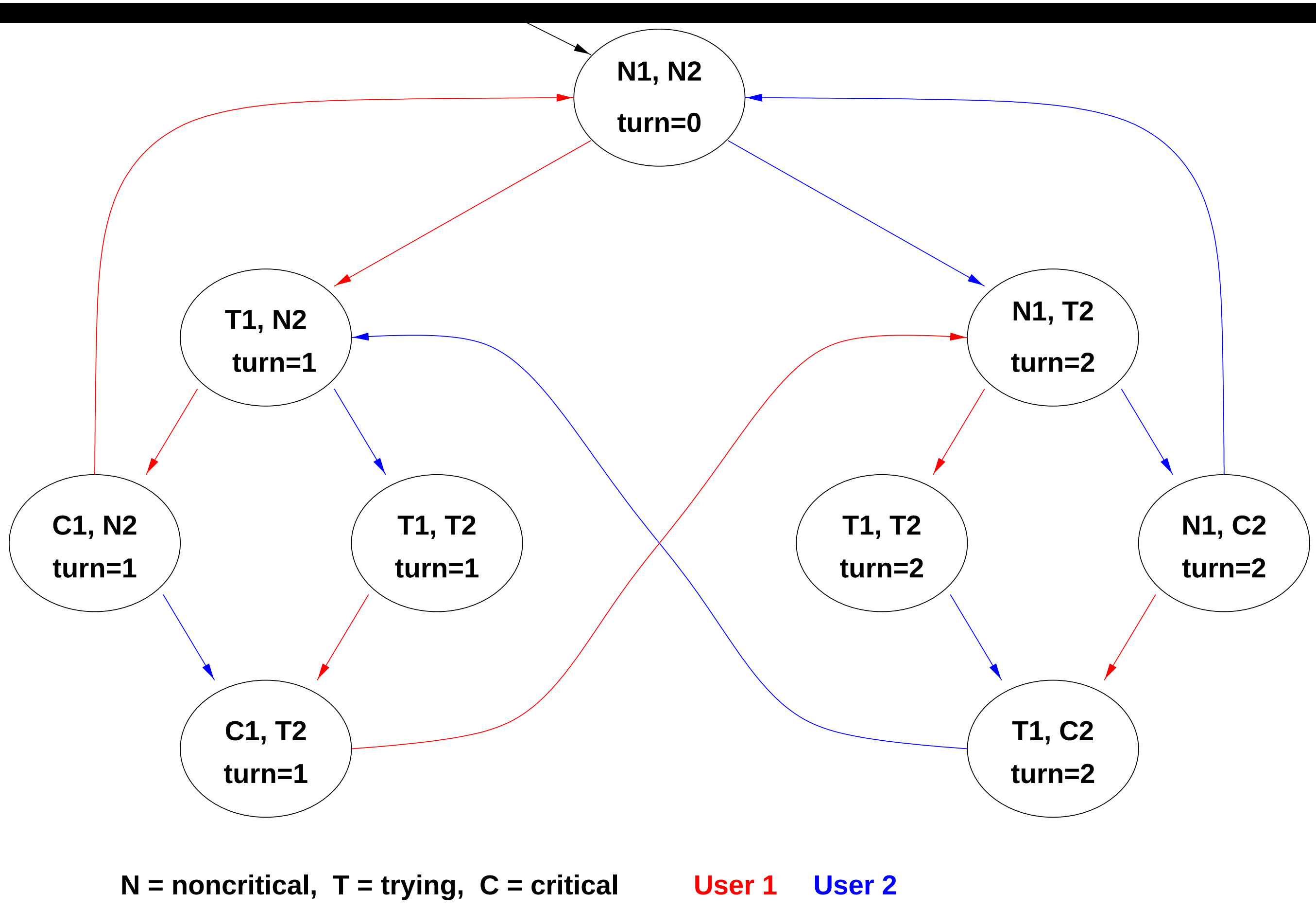
# Example: Safety




**N = noncritical, T = trying, C = critical** **User 1** **User 2**

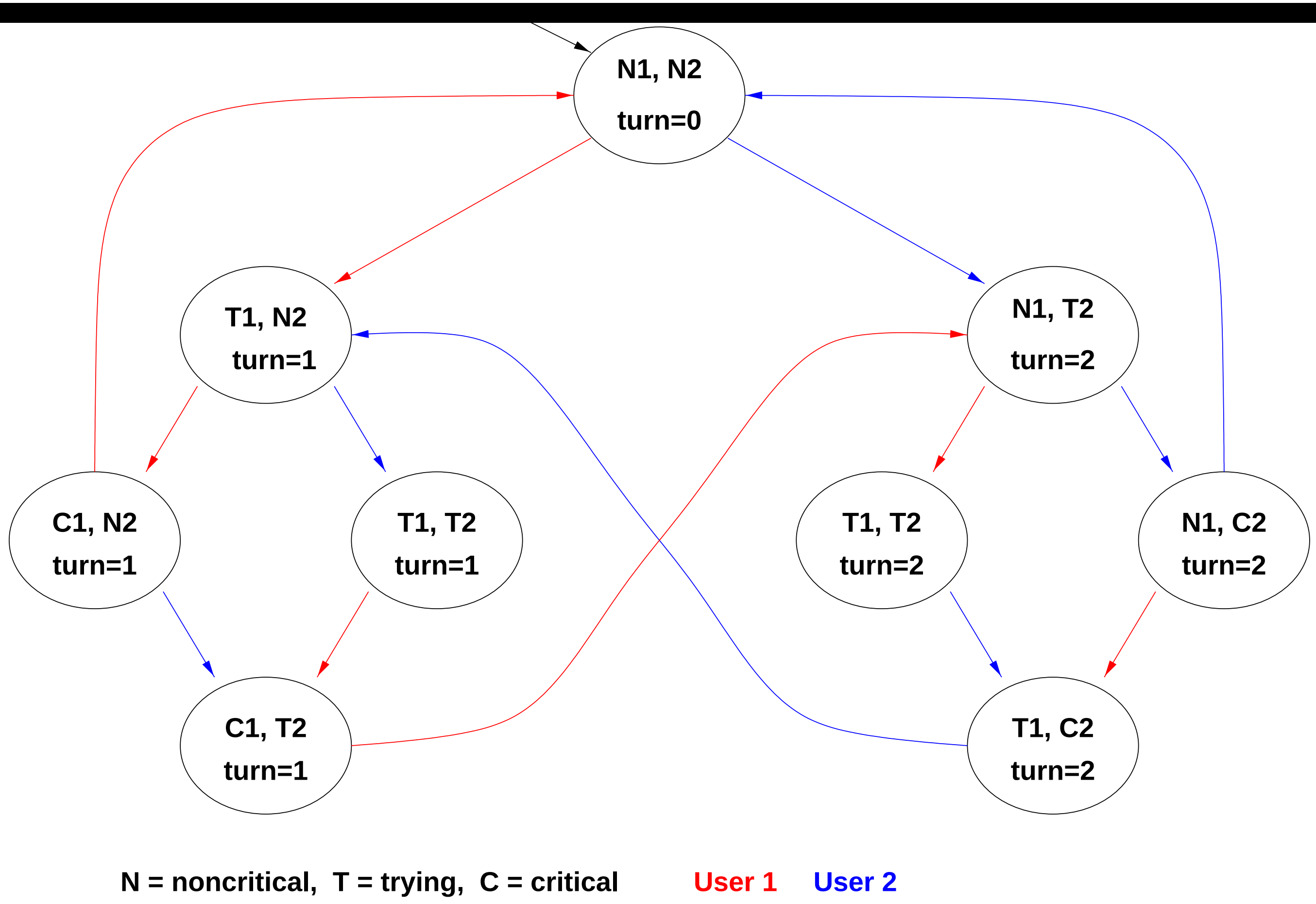
**Does $G\neg(C_1 \wedge C_2)$ hold?** **YES**

# Example: Liveness




**Does $F\,C_1$ hold? NO**

# Example: Liveness

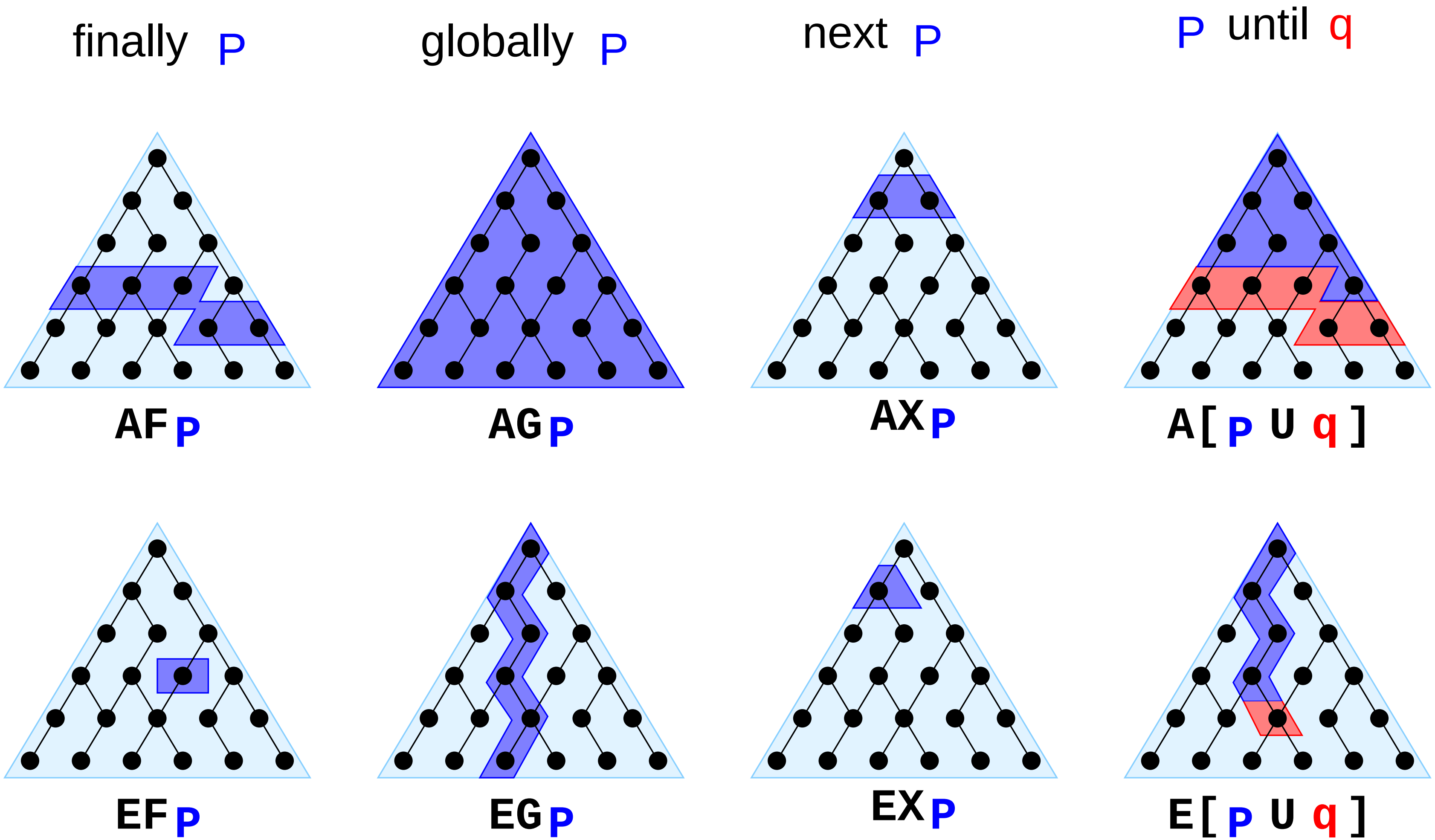



**Does** $G(T_1 \rightarrow F\, C_1)$ **hold?** **YES**

# Computation Tree Logic (CTL)

- **CTL properties are evaluated over trees.**
- Every **temporal operator** $(F, G, X, U)$ preceded by a **path quantifier** ($A$ or $E$).
- **Universal** (or necessity) modalities $(AF, AG, AX, AU)$: the temporal formula is true in **all paths** starting in the current state.
- **Existential** (or possibility) modalities $(EF, EG, EX, EU)$: the temporal formula is true in **some paths** starting in the current state.

# CTL


finally p
globally p
next p
p until q
AF p
AG p
AX p
A[ p U q ]
EF p
EG p
EX p
E[ p U q ]

# CTL

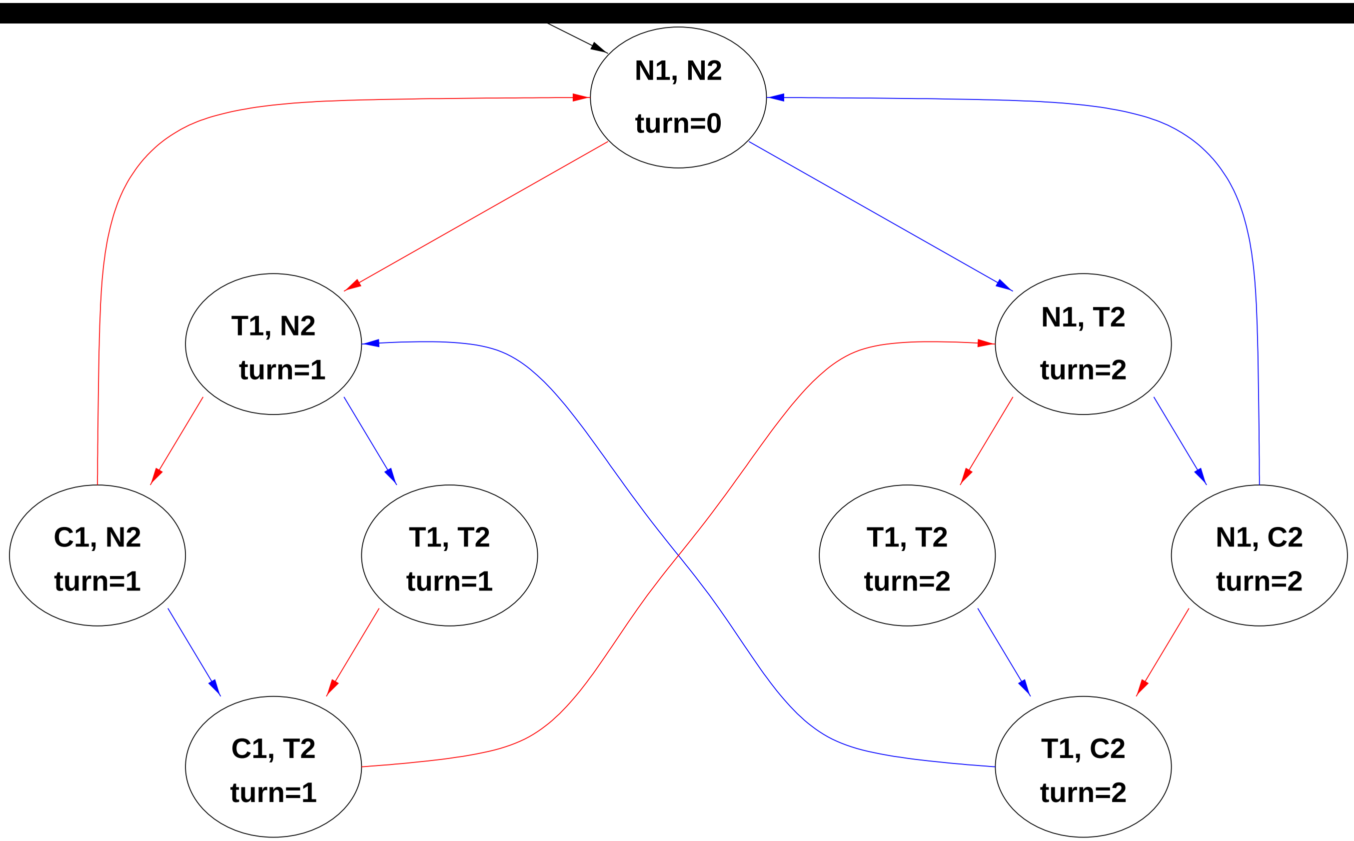
- **Dualities**:

$$
\begin{aligned}
AGp &\leftrightarrow \neg EF\neg p \\
AFp &\leftrightarrow \neg EG\neg p \\
AXp &\leftrightarrow \neg EX\neg p
\end{aligned}
$$

- **Progressions**:

$$
\begin{aligned}
AFp &\leftrightarrow p \vee AX\ AFp \\
EFp &\leftrightarrow p \vee EX\ EFp \\
AGp &\leftrightarrow p \wedge AX\ AGp \\
EGp &\leftrightarrow p \wedge EX\ EGp
\end{aligned}
$$

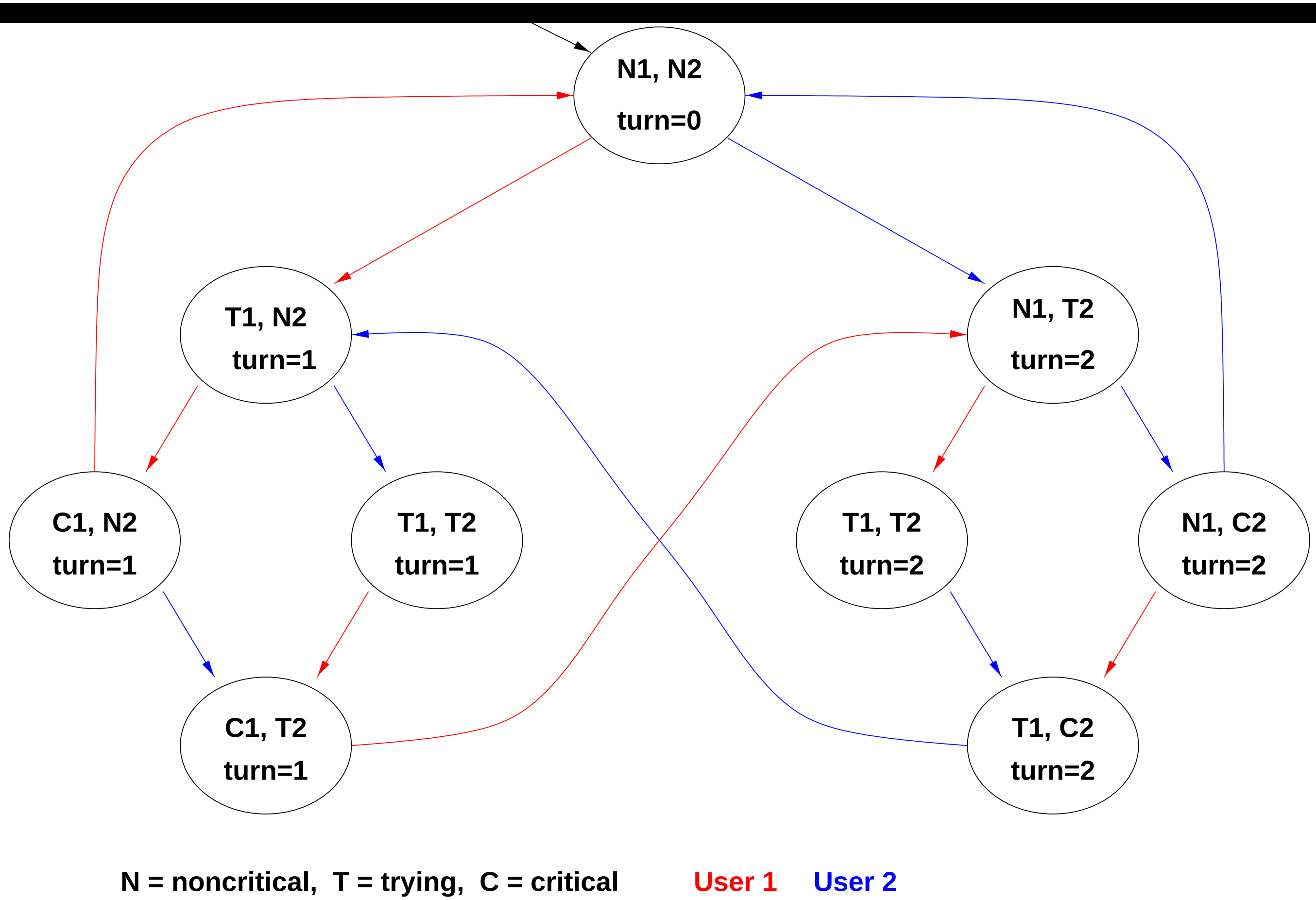
# Example: Safety




**N = noncritical, T = trying, C = critical** **User 1** **User 2**

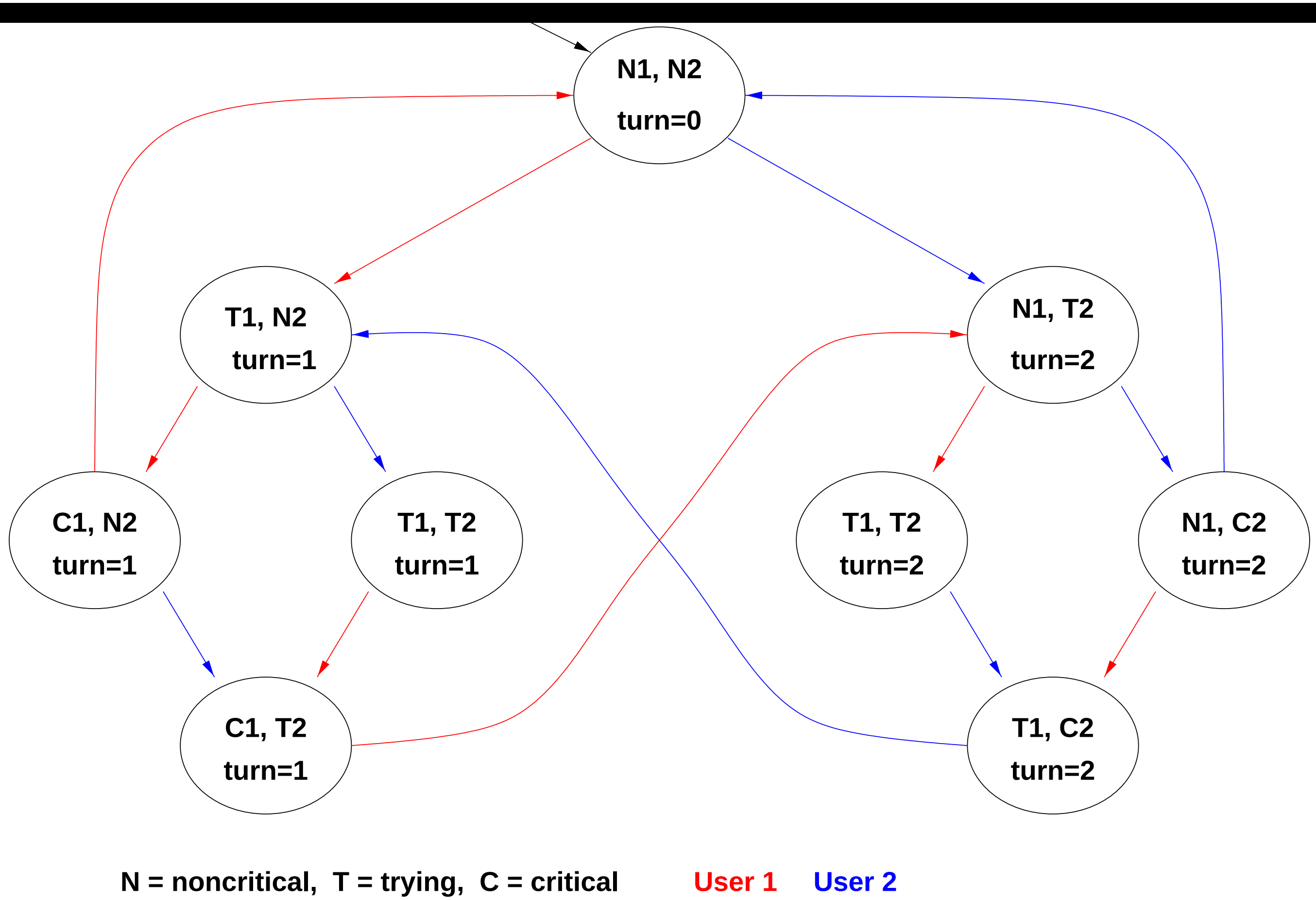
**Does $AG\neg(C_1 \wedge C_2)$ hold?** **YES**

# Example: Liveness



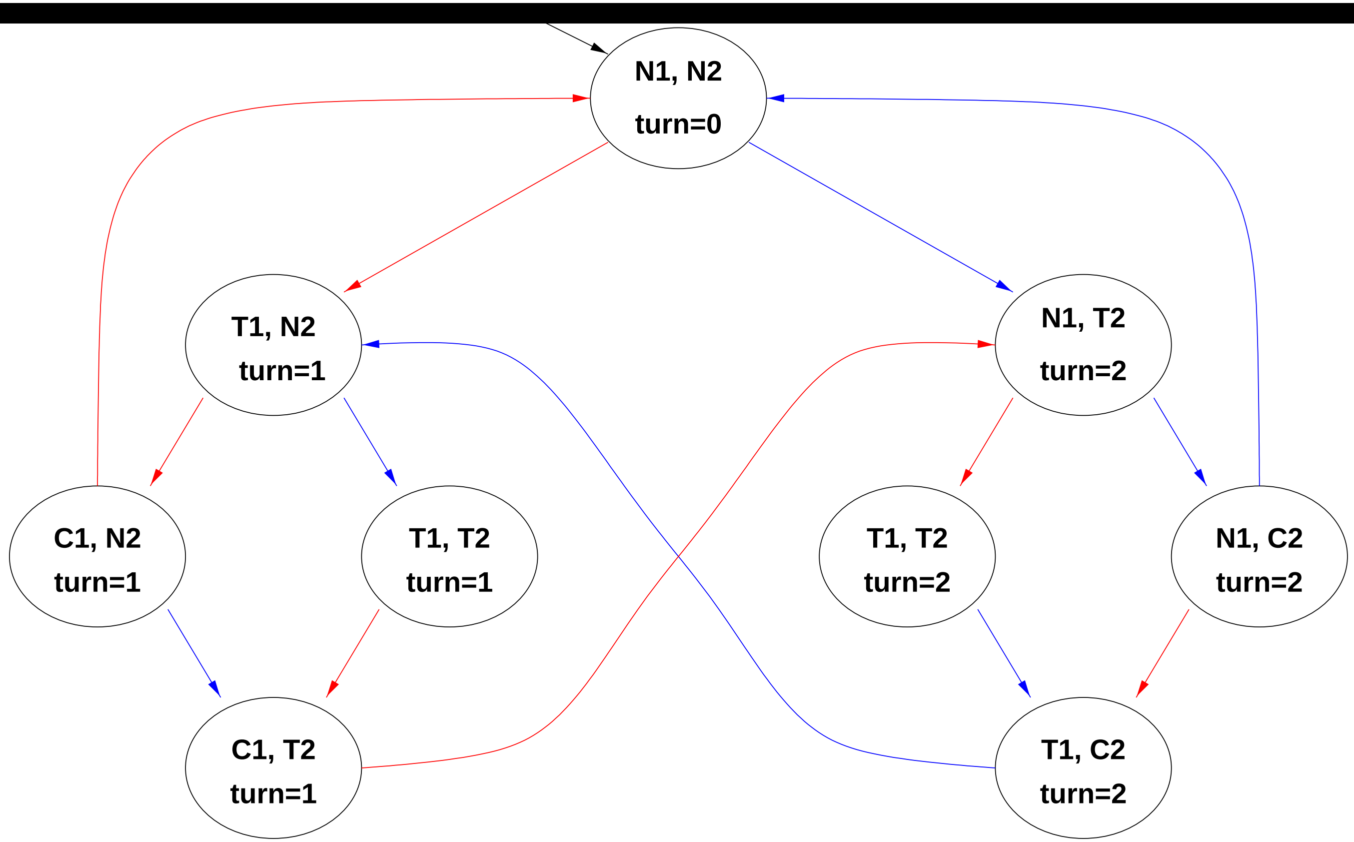

**Does $AG(T_1 \rightarrow AF\, C_1)$ hold?** **YES**

# Example: Liveness




**Does** $AG(N_1 \rightarrow AF\,T_1)$ **hold?** **NO**

# Example: Non-Blocking



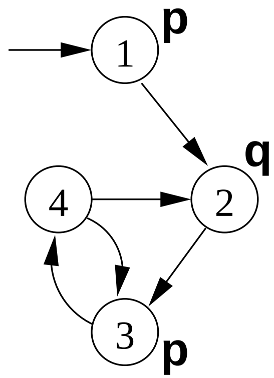

N = noncritical, T = trying, C = critical User 1 User 2

**Does** $AG(N_1 \rightarrow EF\,T_1)$ **hold?** **YES**

# Model Checking

Model Checking is a formal verification technique where...

- ...the system is represented as Finite State Machine




- ...the properties are expressed as temporal logic formulae

LTL: **G(p −> Fq)** CTL: **AG(p −> AFq)**

- ...the model checking algorithm checks whether all the executions of the model satisfy the formula.

# The Main Problem: State Space Explosion

The bottleneck:

- Exhaustive analysis may require to store all the states of the Kripke structure
- The state space may be exponential in the number of components
- State Space Explosion: too much memory required

Symbolic Model Checking:

- Symbolic representation
- Different search algorithms

# Symbolic Model Checking

Symbolic representation:

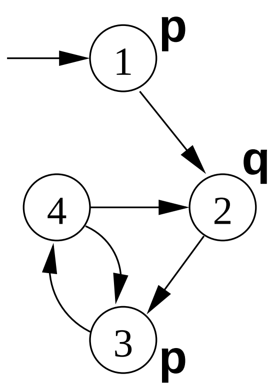
- manipulation of *sets of states* (rather than single states);
- sets of states represented by formulae in propositional logic;
  - set cardinality not directly correlated to size
- expansion of *sets of transitions* (rather than single transitions);
- two main symbolic techniques:
  - Binary Decision Diagrams (BDDs)
  - Propositional Satisfiability Checkers (SAT solvers)

Different model checking algorithms:

- Fix-point Model Checking (historically, for CTL)
- Bounded Model Checking (historically, for LTL)
- Invariant Checking, .... (not covered today)

# CTL Model Checking: Example

Consider a simple system and a specification:

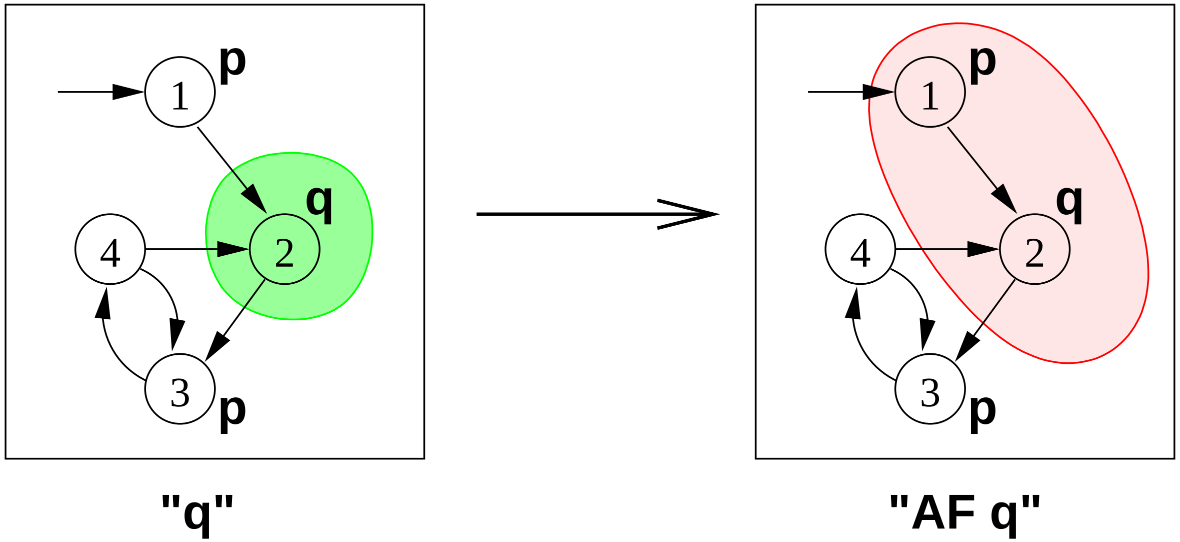



**AG(p −> AFq)**

Idea:

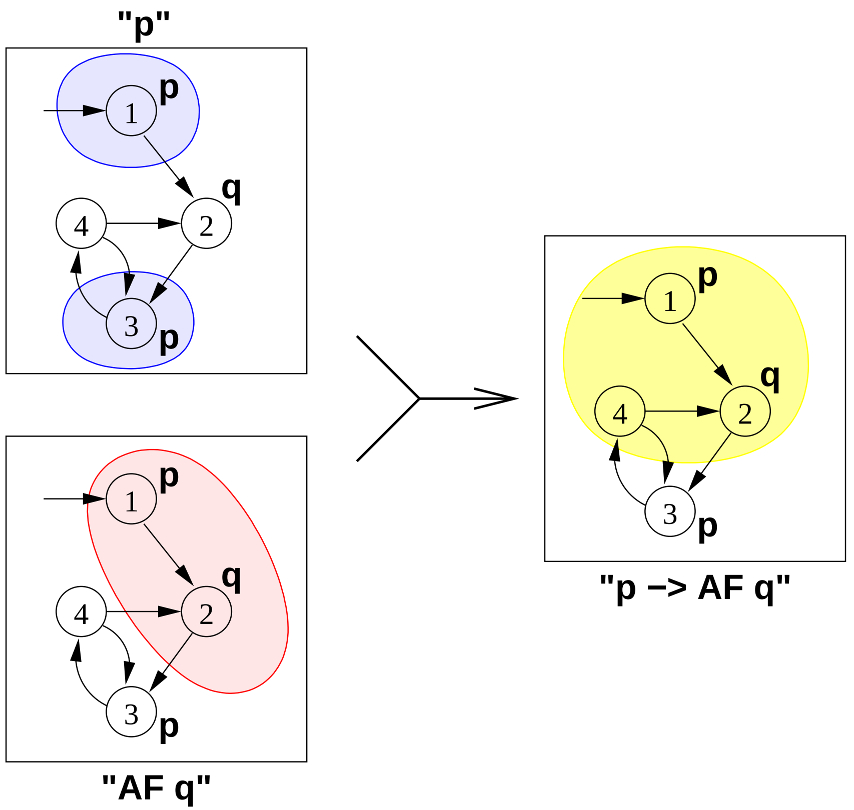
- construct the set of states where the formula holds
- proceeding “bottom-up” on the structure of the formula
- **q**, **AFq**, **p**, **p** $\rightarrow$ **AF q**, **AG(p** $\rightarrow$ **AF q)**

# CTL Model Checking: Example

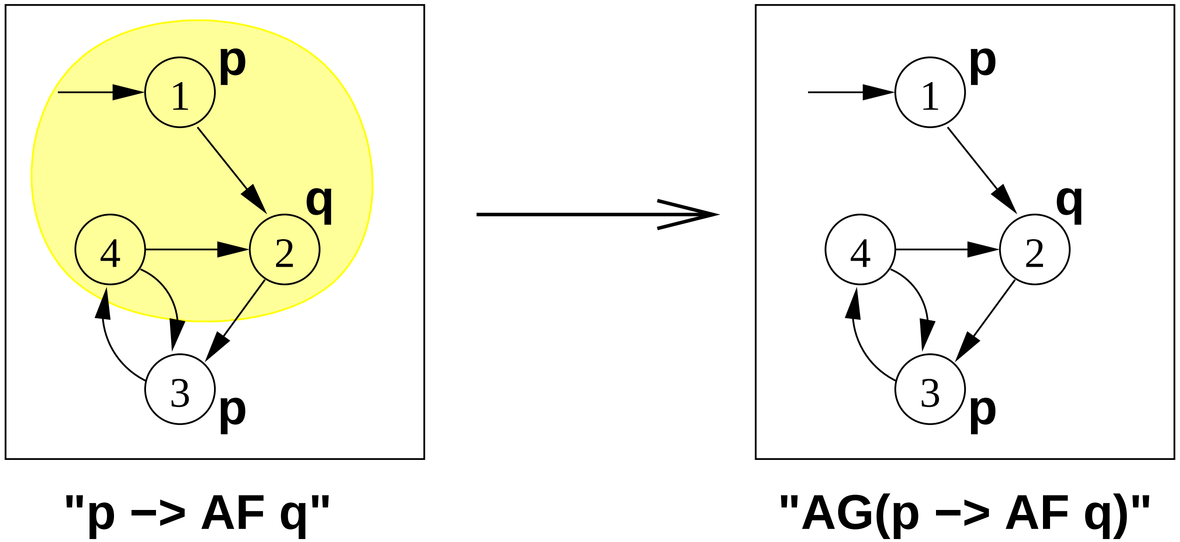



**AF q** is the union of **q**, **AX q**, **AX AX q**, ...

# CTL Model Checking: Example



# CTL Model Checking: Example




The set of states where the formula holds is empty!

Counterexample reconstruction is based on the intermediate sets.

# Fix-Point Symbolic Model Checking

Model Checking Algorithm for CTL formulae based on fix-point computation:

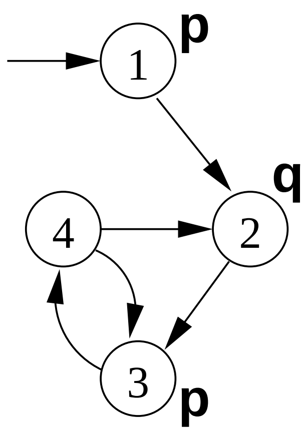
- traverse formula structure, for each subformula build set of satisfying states; compare result with initial set of states.
- boolean connectives: apply corresponding boolean operation;
- on $\mathrm{AX}\,\Phi$, apply preimage computation
  - $\forall \mathbf{s}'.(\mathcal{T}(\mathbf{s}, \mathbf{s}') \rightarrow \Phi(\mathbf{s}'))$
- on $\mathrm{AF}\,\Phi$, compute least fixpoint using
  - $\mathrm{AF}\,\Phi \leftrightarrow (\Phi \vee \mathrm{AX}\,\mathrm{AF}\,\Phi)$
- on $\mathrm{AG}\,\Phi$, compute greatest fixpoint using
  - $\mathrm{AG}\,\Phi \leftrightarrow (\Phi \wedge \mathrm{AX}\,\mathrm{AG}\,\Phi)$

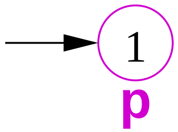
# Bounded Model Checking

Key ideas:

- looks for counter-example paths of increasing length $k$
  - oriented to finding bugs
- for each $k$, builds a boolean formula that is satisfiable iff there is a counter-example of length $k$
  - can be expressed using $k \cdot |\mathbf{s}|$ variables
  - formula construction is not subject to state explosion
- satisfiability of the boolean formulas is checked using a **SAT procedure**
  - can manage complex formulae on several 100K variables
  - returns satisfying assignment (i.e., a counter-example)

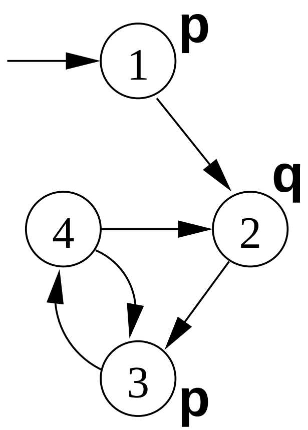
# Bounded Model Checking: Example



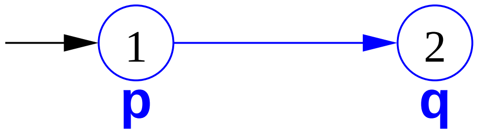

- Formula: **G(p −> Fq)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 0$: 

- No counter-example found.

# Bounded Model Checking: Example

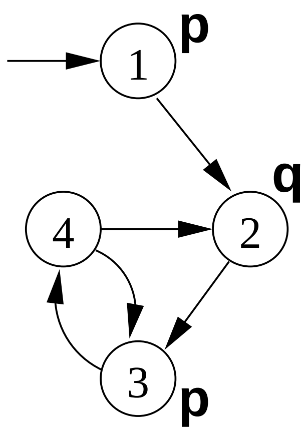


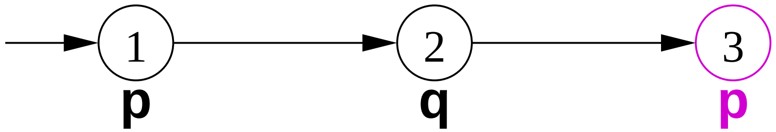

- Formula: **G(p −> Fq)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 1$:




- No counter-example found.

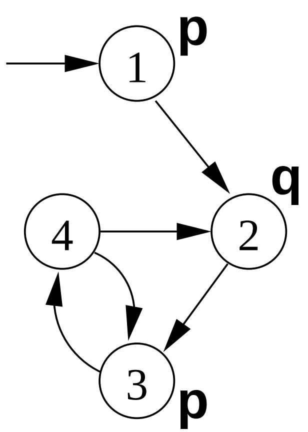
# Bounded Model Checking: Example



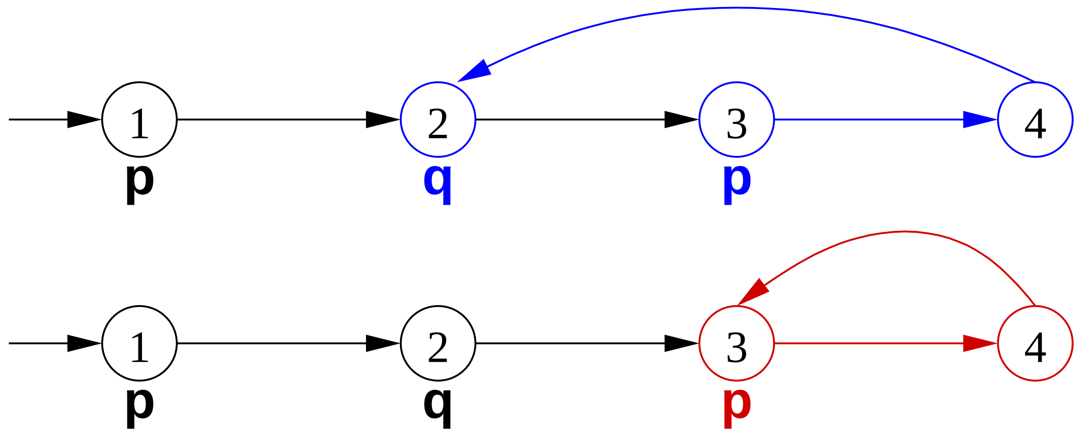

- Formula: **G(p −> Fq)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 2$: 

- No counter-example found.

# Bounded Model Checking: Example




- Formula: **G(p −> Fq)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 3$:




- The 2nd trace is a counter-example!

# Bounded Model Checking

- **Bounded Model Checking**:
  Given a FSM $\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{T} \rangle$, an LTL property $\phi$ and a bound $k \geq 0$:

$$\mathcal{M} \models_k \phi$$

- This is equivalent to the satisfiability problem on formula:

$$[\![\mathcal{M}, \phi]\!]_k \equiv [\![\mathcal{M}]\!]_k \wedge [\![\phi]\!]_k$$

  where:

  – $[\![\mathcal{M}]\!]_k$ is a $k$-path compatible with $\mathcal{I}$ and $\mathcal{T}$:

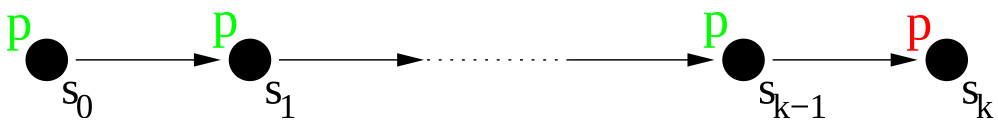
$$\mathcal{I}(\mathbf{s}_0) \wedge \mathcal{T}(\mathbf{s}_0, \mathbf{s}_1) \wedge \ldots \mathcal{T}(\mathbf{s}_{k-1}, \mathbf{s}_k)$$

  – $[\![\phi]\!]_k$ says that the $k$-path satisfies $\phi$

# Bounded Model Checking: Examples

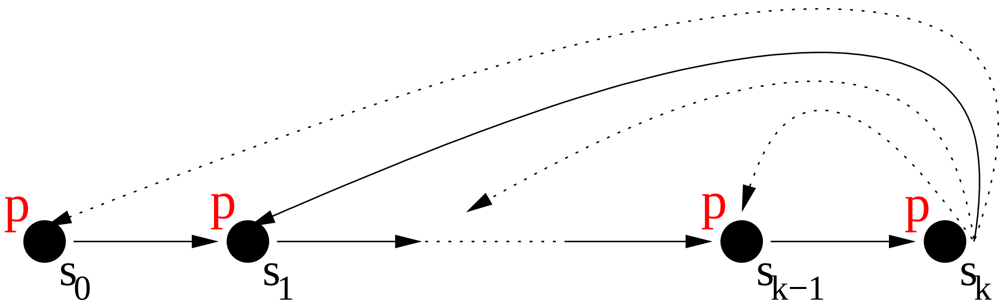
- $\phi = \mathrm{F}\, p$

$$[\![\mathrm{F}\, p]\!]_k = \bigvee_{i=0}^{k} p(\mathbf{s}_i)$$

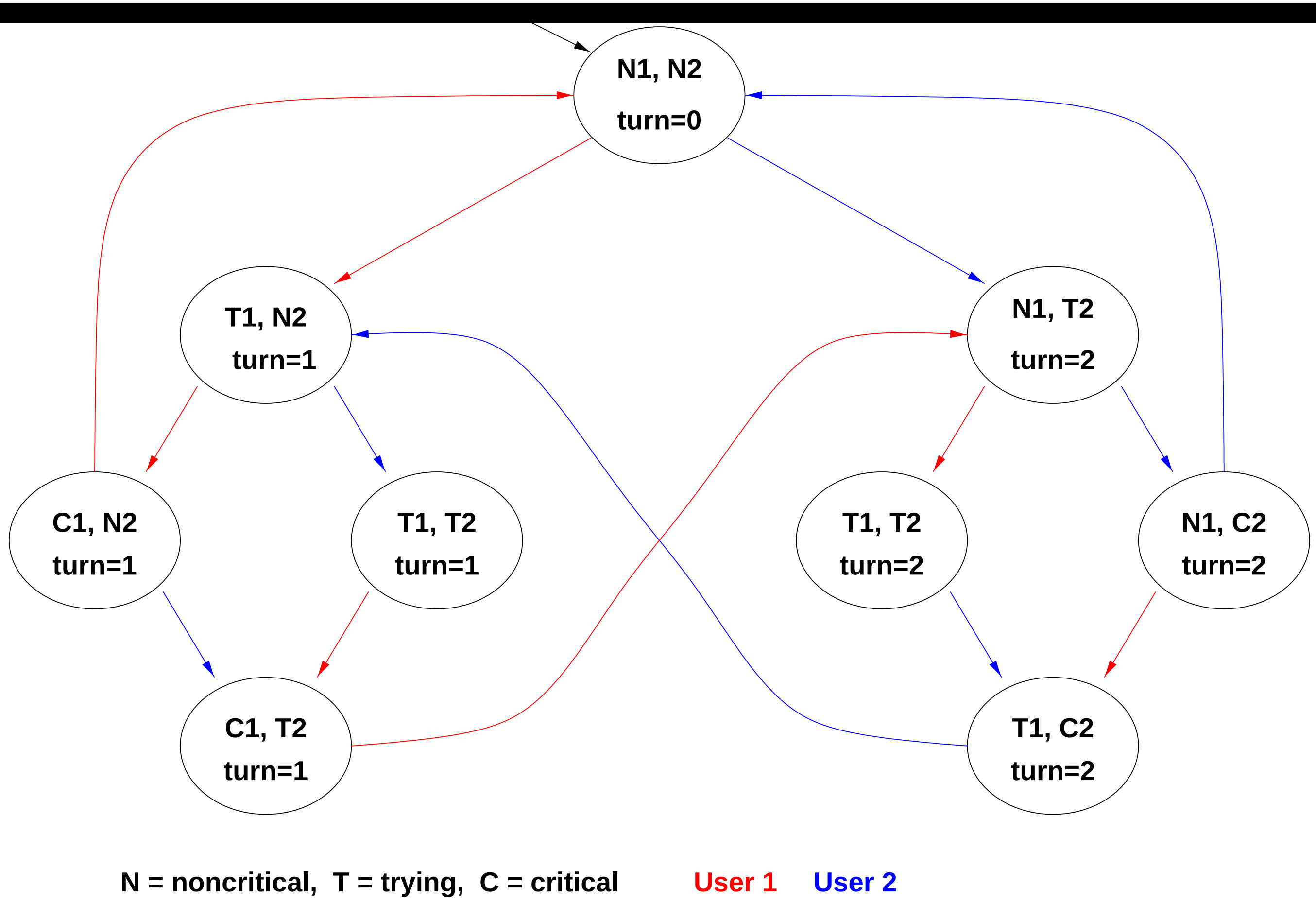



- $\phi = \mathrm{G}\, p$

$$[\![\mathrm{G}\, p]\!]_k = \bigvee_{i=0}^{k} \left( \mathcal{T}(\mathbf{s}_k, \mathbf{s}_i) \wedge \bigwedge_{i=0}^{k} p(\mathbf{s}_i) \right)$$

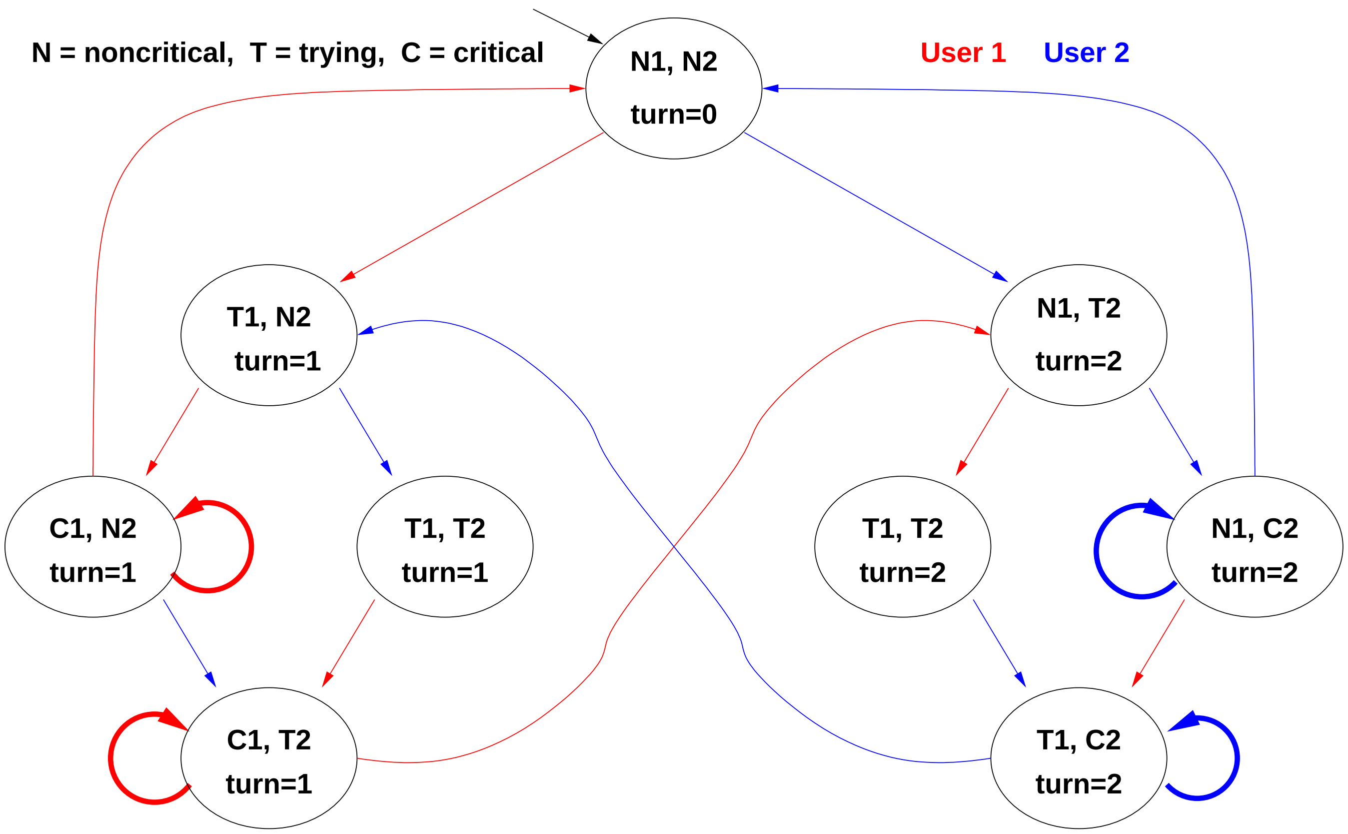


# The need for fairness conditions




**Does** $AG(T_1 \rightarrow AF\,C_1)$ **hold?** **YES**

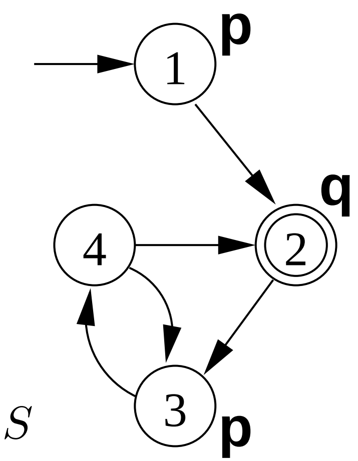
# The need for fairness conditions




**Does** $AG(T_1 \rightarrow AF\, C_1)$ **hold?** **NO**

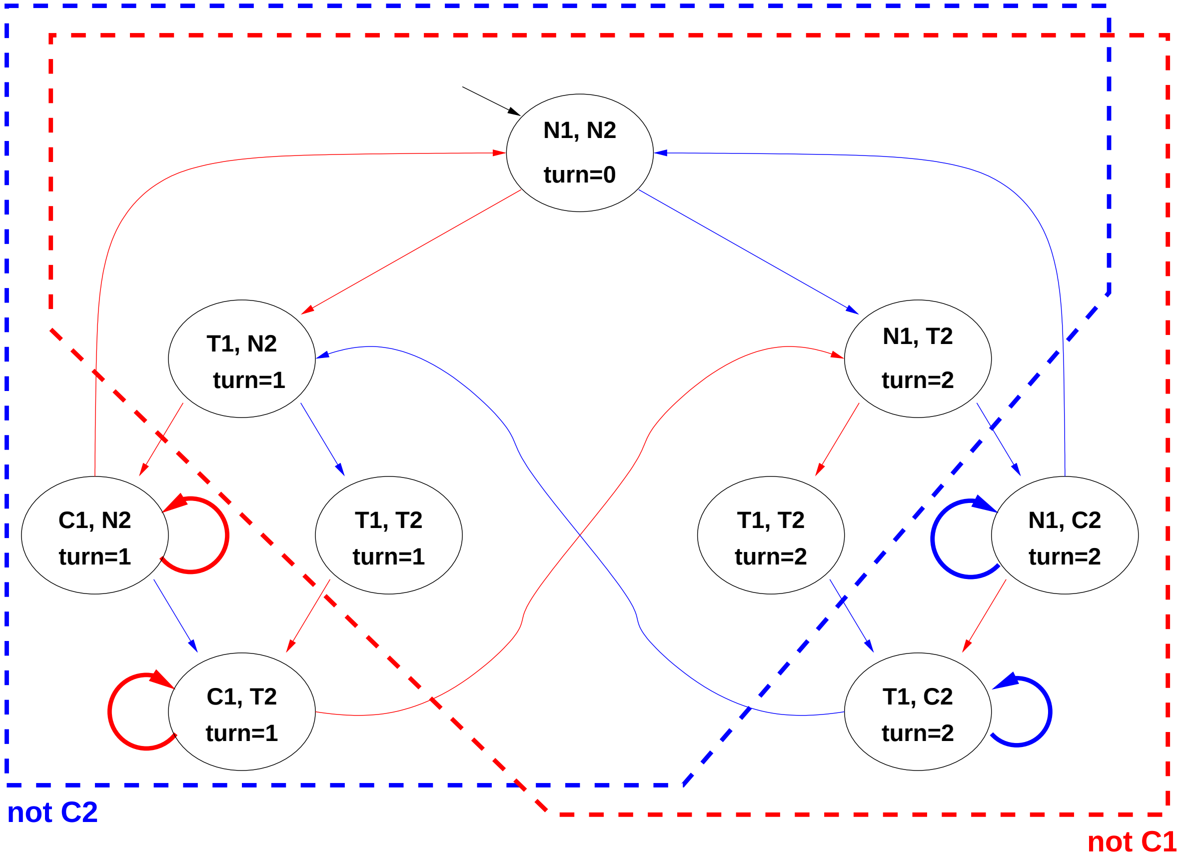
# Fair Kripke models

- Intuitively, fairness conditions are used to eliminate behaviours where a condition never holds
  - e.g. once a process is in critical section, it never exits
- Formally, a Kripke model $(S, R, I, L, F)$ consists of
  - a set of states $S$;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a labeling $L \subseteq S \times AP$.
  
  $\Rightarrow$ a set of fairness conditions $F = \{f_1, \ldots, f_n\}$, with $f_i \subseteq S$




- Fair path: at least one state for each $f_i$ occurs an infinite number of times
- Fair state: a state from which at least one fair path originates

# Fairness: {{ not C1},{not C2}}




**Does** $AG(T_1 \rightarrow AF\,C_1)$ **hold? YES**