Scuola Estiva di Logica, Gargnano, 31 agosto - 6 settembre 2008

LOGIC AT WORK

Formal Verification of Complex Systems via

Symbolic Model Checking

Roberto Sebastiani
Dip. Ingegneria e Scienza dell'Informazione, Università di Trento
roberto.sebastiani@disi.unitn.it
http://disi.unitn.it/~rseba

Content

Problems in Developing IT Systems

- ▷ increasing dependability
  - everything important depends on computers (industrial production, banking, stock market, transport,...)
    $\Longrightarrow$quality is essential
- ▷ systems increasingly complex
  - Moore law: exponential growth
    (10^{30} transistors/processor, multi million LOC's/OS)
    $\Longrightarrow$cost for testing is exploding
- ▷ time-to-market affects potential revenue dramatically:
  - 1 week delay for a microprocessor
    $\Longrightarrow$loss of more than $20.000.000$ US$ (year 2004)

Critical Systems

- Safety-critical:

  systems whose failure can cause life losses or serious environmental damage

  (e.g., trains & planes control, nuclear plants control, ...)

- Mission-critical:

  systems whose failure can cause the failure of the goals of important missions

  (e.g., space craft navigation)

- Business-critical:

  systems whose failure can cause the loss of big or huge amounts of money

  (e.g., bank management software, operating systems)

A paradigmatic catastrophe: the PENTIUM-IV Bug

- in 1994 Professor Thomas Nicely from Lynchburg College in Virginia discovered incorrect behaviors in the Pentium chip.
- Cause: a design error in the floating point division algorithm in the ALU.
- The chip was withdrawn and substituted by Intel.
- 450 US$ millions lost!
- Since 1994, Intel adopts formal methods!
- Others famous catastrophes due to bugged systems:
  - The Therac-25 case, 1985 (4 people killed, 2 ijured)
  - The Ariane-5 case, 1996 (500M US$ lost)
  - The Russian ATM case, 2006 (2Billion rubles wrongly credited)
  - ...
- Formal methods now mandatorily required by most international agencies for safety-critical applications.

Formal Verification

Problem: given a specification S, and a model M (system/program/circuit), check that M verifies S: $M \models S$

- ▷ Most important in HW and protocols (but increasingly used in SW)
- ▷ Exhaustive verification
- ▷ Approaches: theorem proving, equivalence checking, model checking

Model Checking

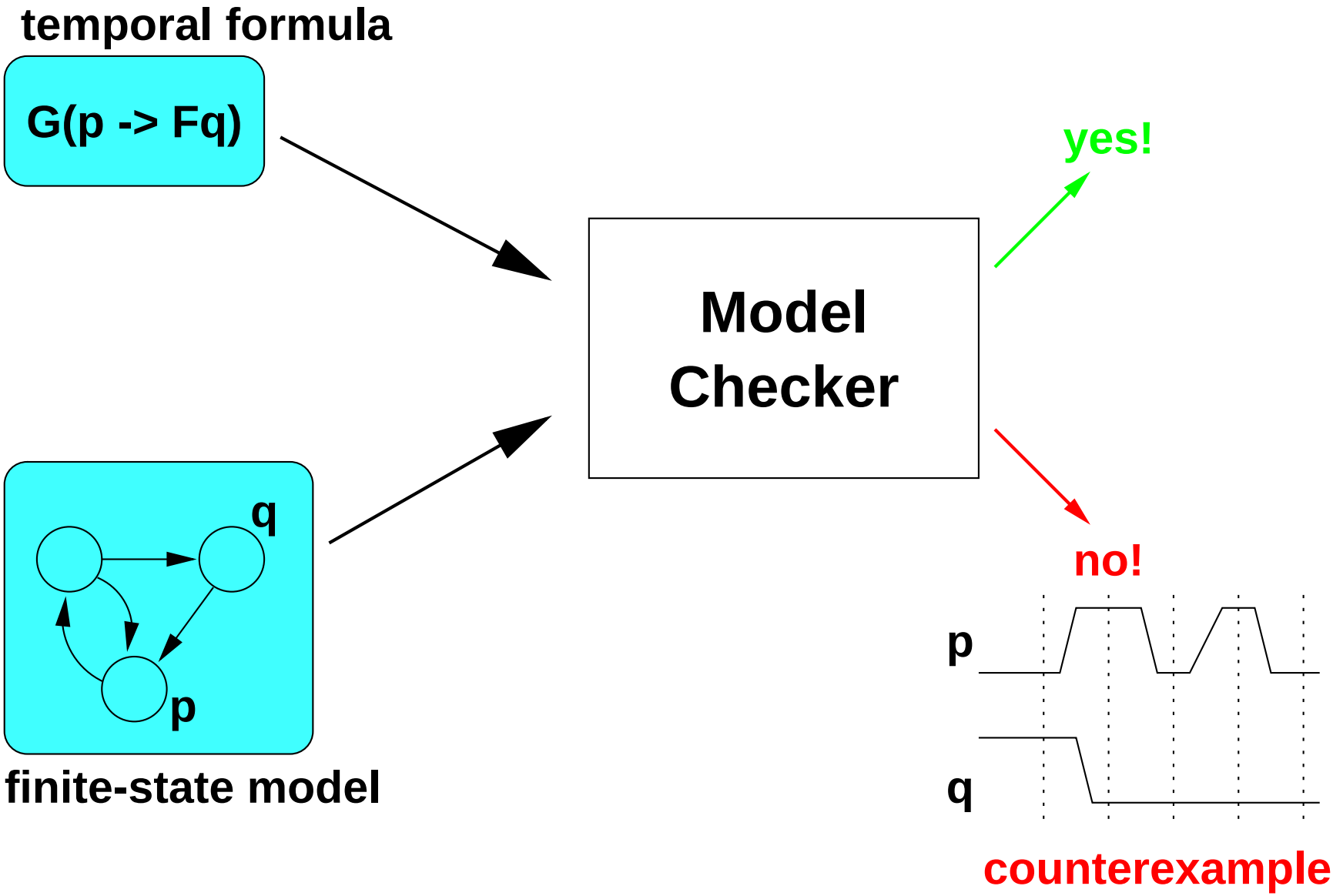
- ▷ F.V. by exhaustive search over the state space.
- ▷ Systems modeled as Finite State Machines M (Kripke models)
- ▷ Properties expressed with a formal representation Ψ
  (e.g Temporal Logic, Automata, MSCs, etc.).
  $\Longrightarrow$Precise, unambiguous semantics
- ▷ Verification via logical reasoning:

$$M \models \Psi$$

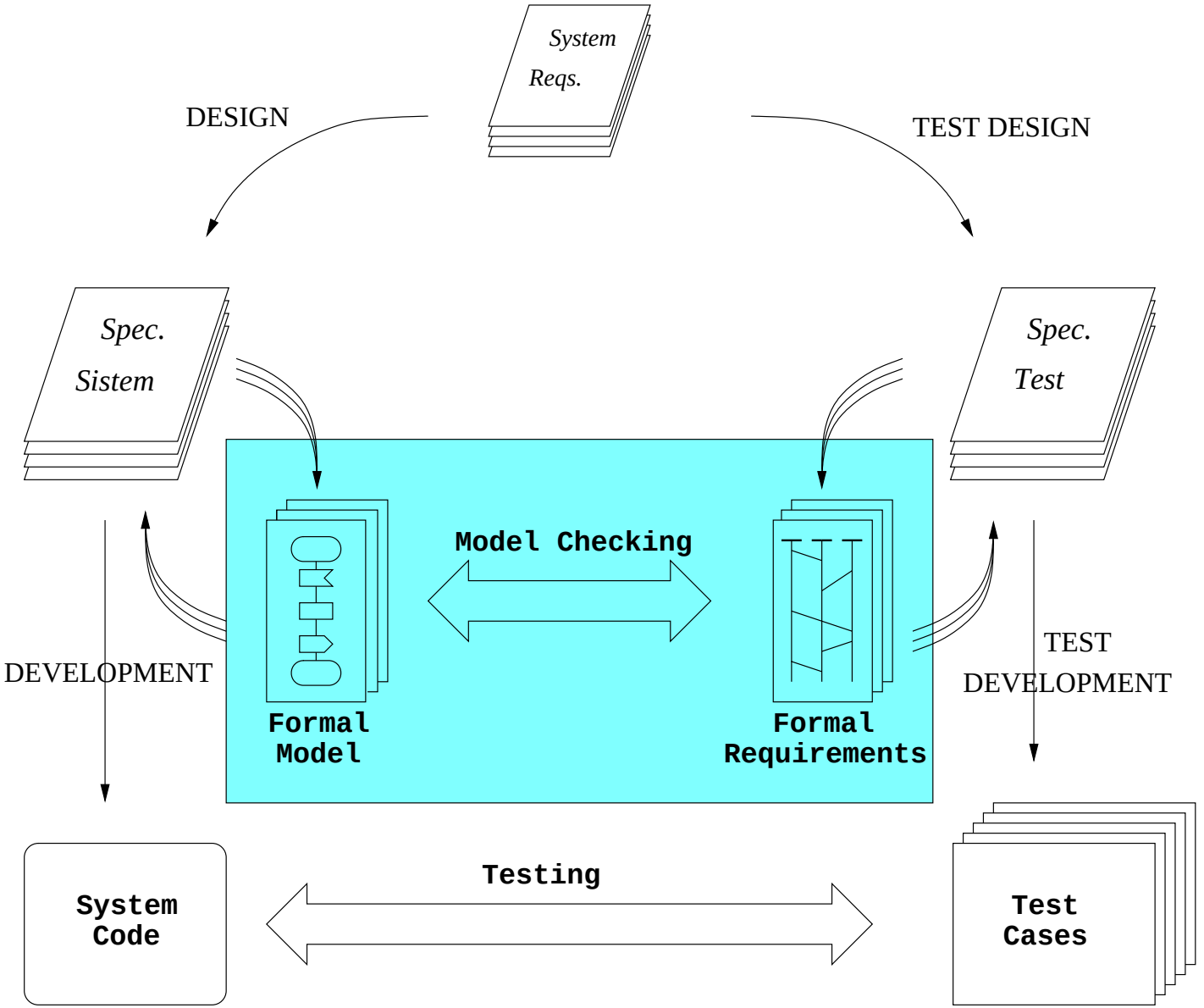
  Is M a logical model for Ψ?

  - Yes $\Longrightarrow$the system verifies the property
  - No $\Longrightarrow$a counter-example is returned
    (representing an execution leading to a bug).

Model Checking (cont.)



Extending the traditional development process with M.C.



Industrial Success of Model Checking

- ▷ Powerful debugging capabilities:
  - helps detecting problems in early stages of the development cycle
  - exhaustive, thus effective
  - provides counterexamples (directs the designer to the problem).
- ▷ can be integrated within industrial development cycle:
  - compilers for practical design languages
    (e.g., VHDL, VERILOG, Esterel, SDL, StateCharts, SMV, Promela,...);
- ▷ Who is using it?
  - Design teams: Intel, AMD, IBM, Microsoft, Lucent, ...
  - CAD tool vendors: Cadence, Synopsis,...
  - Commercial model checkers: FormalCheck (IBM), SLAM (Microsoft), ...
- ▷ Does not require deep training (“push-button” technology).

“... formal verification has now entered the critical path in the process of development of a microprocessor” [Bob Bentley, Intel, CAV’2005]

Model Checking: State-of-the-art

- Well-founded theory and algorithms
- Robust and well-established tools (e.g. VIS, SPIN, COSPAN, NuSMV, Uppaal)
- Very successful for verifying
  - medium-size “isolated” hardware
  - protocols
- increasingly used for verifying software (e.g., Microsoft)
- increasingly popular in industry

Model Checking: Awards

- Amir Pnueli: ACM Turing Award 1996

  "For his seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification."

- Gerard J. Holzmann, Robert P. Kurshan, Moshe Y. Vardi, and Pierre Wolper: ACM Kanellakis Award 2006

  "... demonstrated that checking the correctness of reactive systems can be achieved using a mathematical analysis of abstract machines."

- Edmund Clarke, E. Allen Emerson and Joseph Sifakis:
  ACM Turing Award 2007

  "... In recognition of their pioneering work on an automated method for finding design errors in computer hardware and software [Model Checking]"

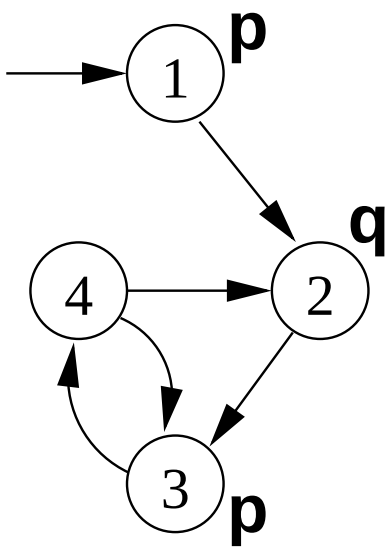
Content

Modeling the system: Kripke models

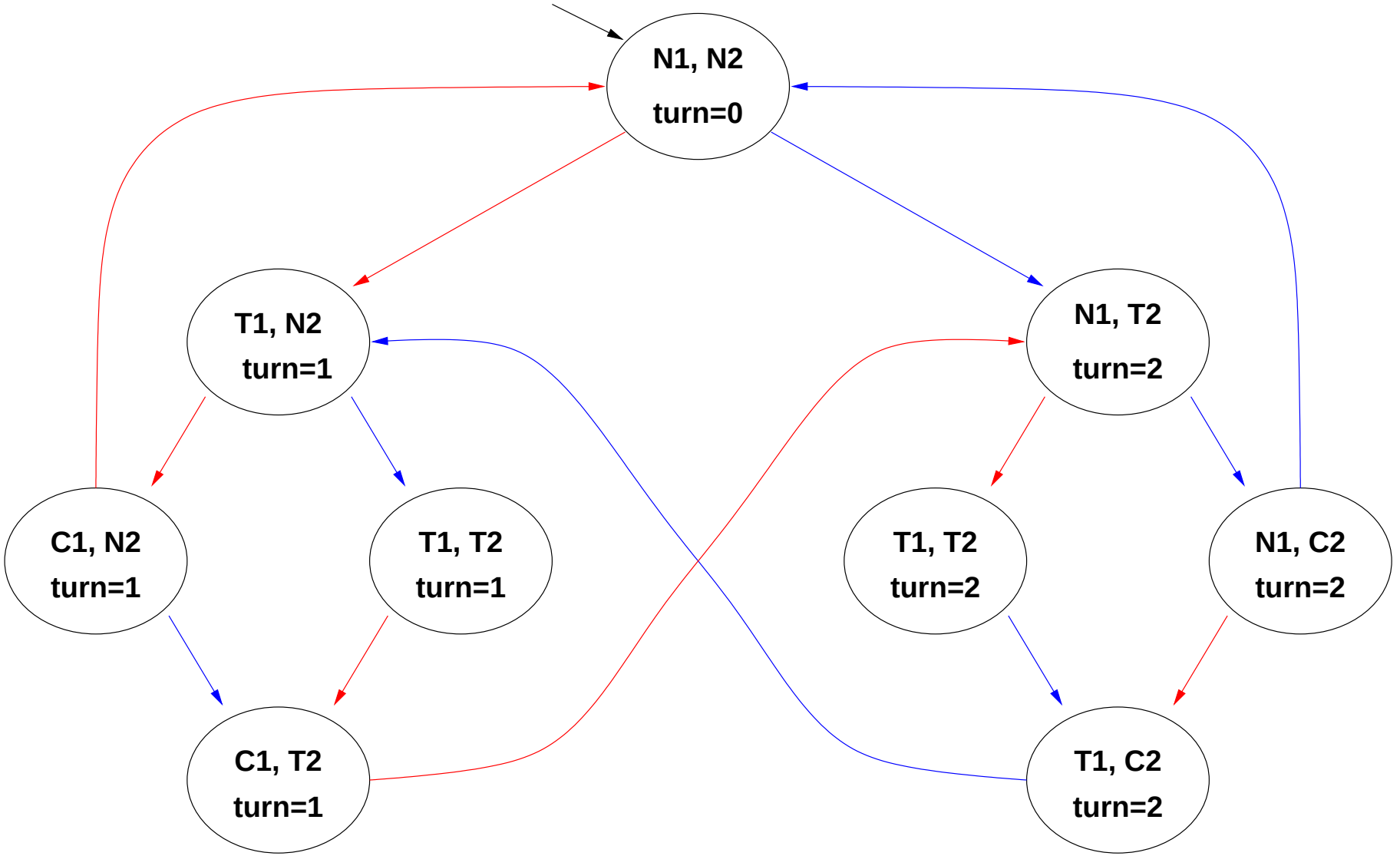
- Kripke models are used to describe reactive systems:
  - nonterminating systems with infinite behaviors (e.g. communication protocols, hardware circuits);
  - represent the dynamic evolution of modeled systems;
  - a state includes values to state variables, program counters, content of communication channels.
  - can be animated and validated before their actual implementation

Kripke model: definition

- Formally, a Kripke model $\langle S, I, R, AP, L \rangle$ 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 set of atomic propositions AP;
  - a labeling function $L \subseteq S \times AP$.



Example: a Kripke model for mutual exclusion



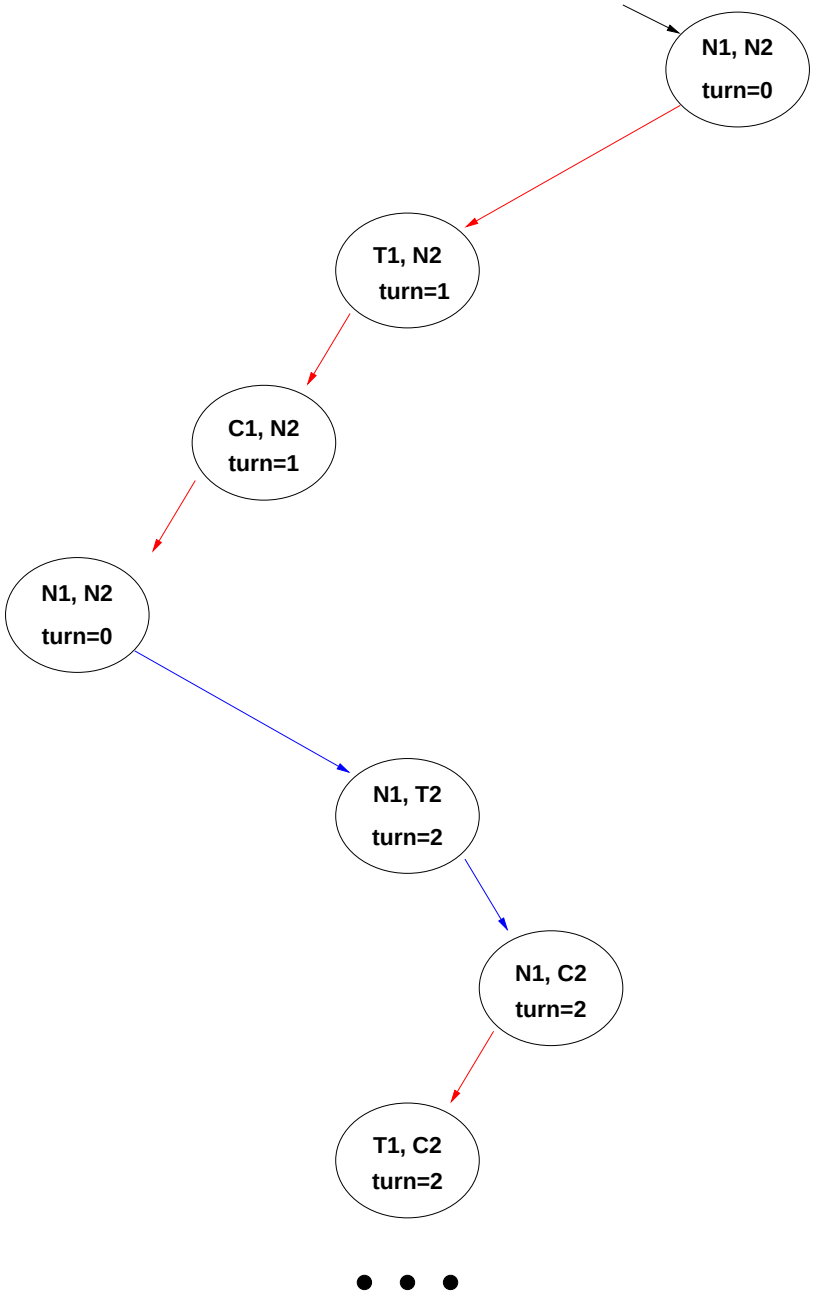
N = noncritical, T = trying, C = critical User 1 User 2

Path in a Kripke Model

- An path (execution) in a Kripke model M is an infinite sequence of states

$$\pi = s_0, s_1, s_2, \ldots$$

  such that $s_0 \in I$ and $(s_i, s_{i+1}) \in R$, $\forall i$.
- A state s is reachable in M if there is a path from the initial states to s.

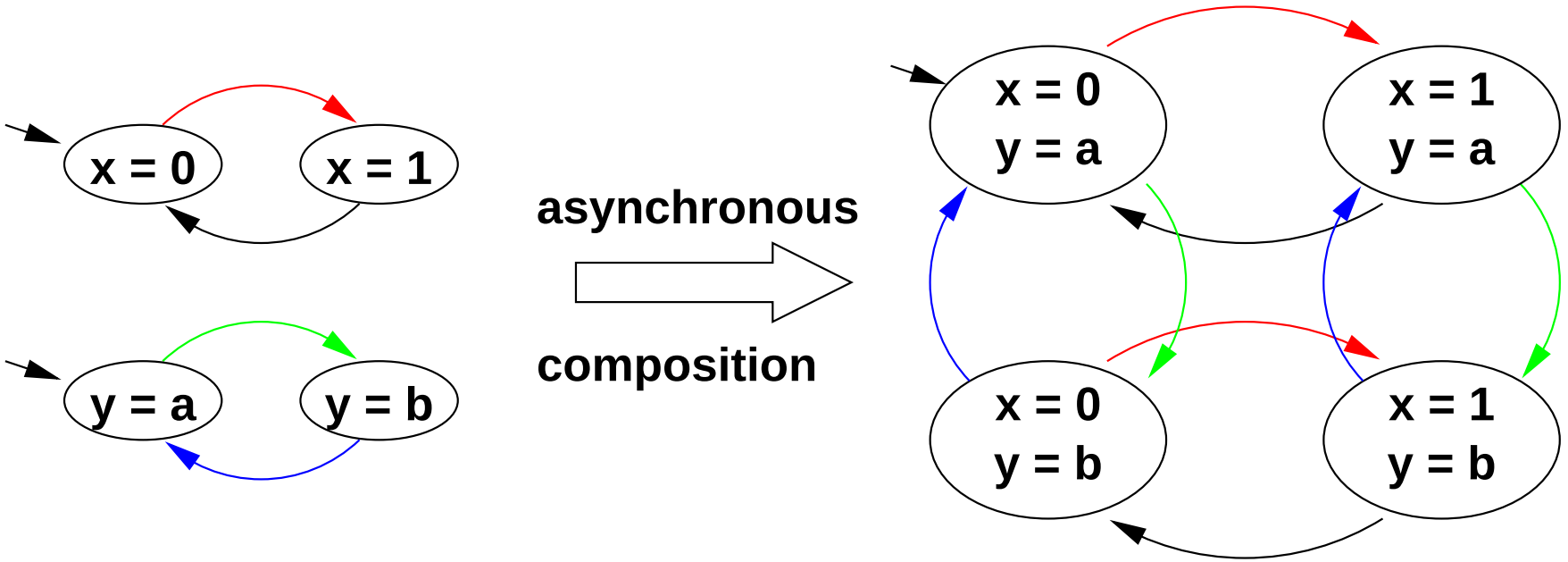


Composing Kripke Models

- Complex Kripke Models are tipically obtained by composition of smaller ones
- Components can be combined via
  - asynchronous composition.
  - synchronous composition,

Asynchronous Composition

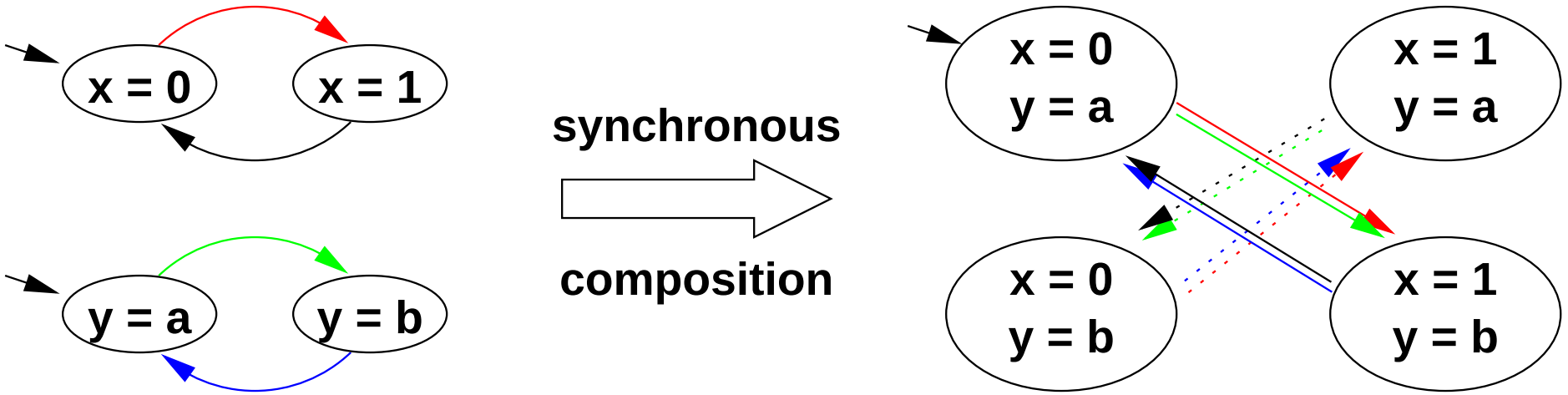
- Interleaving of evolution of components.
- At each time instant, one component is selected to perform a transition.



- Typical example: communication protocols.

Synchronous Composition

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



- Typical example: sequential hardware circuits.

Description languages for Kripke Model

- Tipically a Kripke model is not given explicitly, rather it is usually presented in a structured language (e.g., SMV, SDL, PROMELA, StateCharts, VHDL, ...)
- Each component is presented by specifying
  - state variables: determine the set of atomic propositions AP, 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.

Remark: tipically these description are much more compact (and intuitive) than the explicit representation of the Kripke model.

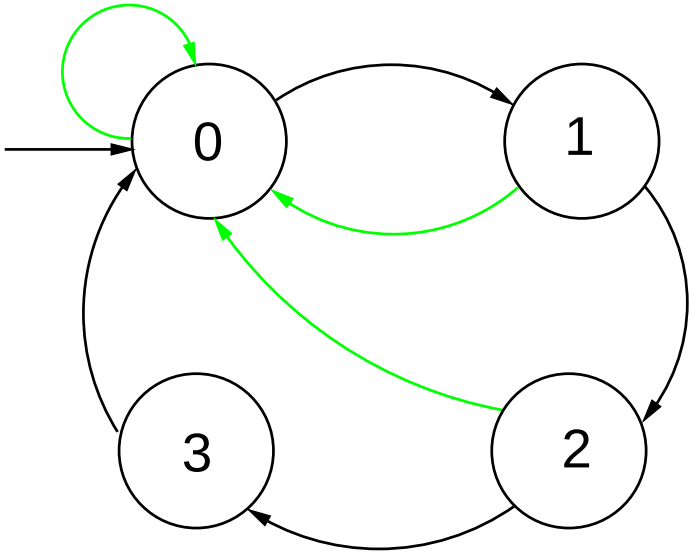
The SMV language

- The input language of the SMV M.C. (and NuSMV)
- Booleans, enumerative and bounded integers as data types
  (now enriched with other constructs)
- An SMV program consists of:
  - Declarations of the state variables (e.g., `b0`);
  - Assignments that define the valid initial states (e.g., `init(b0) := 0`).
  - Assignments that define the transition relation (e.g., `next(b0) := !b0`).
- Allows for both synchronous and asyncronous composition of modules (though synchronous interaction more natural)

The SMV language: example

Example: The modulo 4 counter with reset

```
MODULE main
 VAR
   b0    : boolean;
   b1    : boolean;
   reset : boolean;
   out   : 0..3;

 ASSIGN
   init(b0) := 0;
   next(b0) := case
                 reset = 1 : 0;
                 reset = 0 : !b0;
               esac;

   init(b1) := 0;
   next(b1) := case
                 reset = 1 : 0;
                 reset = 0 : (b0 xor b1);
               esac;
   out := b0 + 2*b1;
```



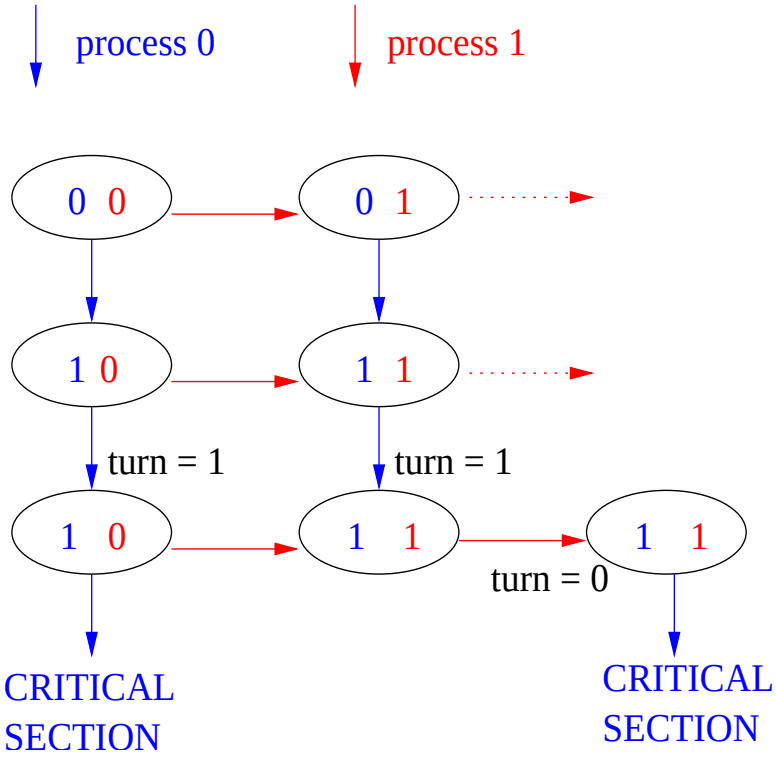
The PROMELA language

- PROMELA (Process Meta Language) is the modeling language of the M.C. SPIN
- The syntax is C-like
- A system in PROMELA consists of a set of *processes* that interact by means of:
  - *shared variables*
  - *communication channels*
    - rendez-vous communications
    - buffered communications
- Processes can be created dynamically
- Allows for both synchronous and asyncronous composition of processes (though asynchronous interaction more natural)

The PROMELA language: example

Example: A Mutual Exclusion Algorithm

```
bool turn;
bool flag[2];

proctype User(bool pid) {
  flag[pid] = 1;
  turn = 1-pid;
  (flag[1-pid] == 0 || turn == pid);
  /* Begin of critical section */
  ...
  /* End of critical section   */
  flag[pid] = 0;
}

init { run User(0); run User(1) }
```



Content

- $\surd$ Motivations and Goals . . . . . .
- $\surd$ Representing transition systems as Kripke Models . . . . . .
- $\Rightarrow$ Representing properties as temporal logic formulas . . . . . .
  - CTL Model Checking: general ideas . . . . . .
  - Symbolic CTL Model Checking . . . . . .
  - Conclusions, state of the art & research developments . . . .

Temporal Logics

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

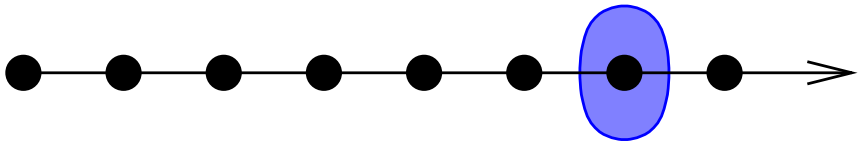
Linear Temporal Logic (LTL): intuitions

LTL is given by the standard boolean logic enhanced with the following temporal operators:

- "Next" $\mathbf{X}$: $\mathbf{X}\varphi$ is true in s_t iff φ is true in s_{t+1}
- "Finally" (or "eventually") $\mathbf{F}$: $\mathbf{F}\varphi$ is true in s_t iff φ is true in **some** $s_{t'}$ with $t' \geq t$
- "Globally" (or "henceforth") $\mathbf{G}$: $\mathbf{G}\varphi$ is true in s_t iff φ is true in **all** $s_{t'}$ with $t' \geq t$
- "Until" $\mathbf{U}$: $\varphi\mathbf{U}\psi$ is true in s_t iff
  - ψ is true in some state $s_{t'}$ with $t' \geq t$
  - φ is true in all states $s_{t''}$ with $t \leq t'' < t'$

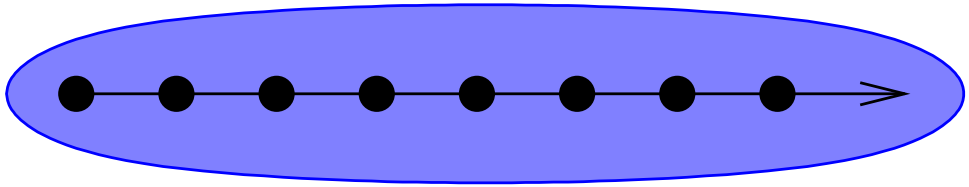
LTL: intuitions

finally P



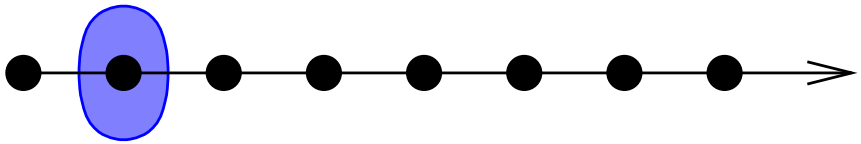
F P

globally P



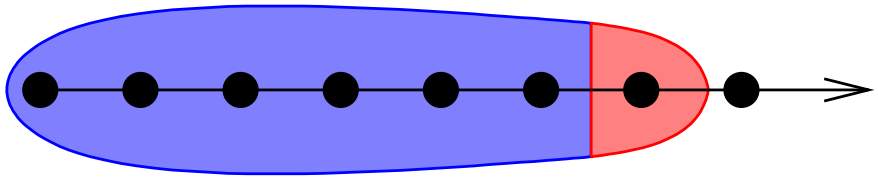
G P

next P



X P

P until q



P U q

Linear Time Temporal Logic (LTL): Syntax

- ▷ An atomic proposition is a LTL formula;
- ▷ if φ_1 and φ_2 are LTL formulae, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$ are LTL formulae;
- ▷ if φ_1 and φ_2 are LTL formulae, then $\mathbf{X}\varphi_1$, $\varphi_1\mathbf{U}\varphi_2$, $\mathbf{G}\varphi_1$, $\mathbf{F}\varphi_1$ are LTL formulae, where $\mathbf{X}$, $\mathbf{G}$, $\mathbf{F}$, $\mathbf{U}$ are the "next", "globally", "eventually","until" temporal operators respectively.

N.B: LTL can be defined in terms of $\wedge$, $\neg$, $\mathbf{X}$, $\mathbf{U}$ only:

- ▷ $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 := (\neg\varphi_1 \vee \varphi_2)$, $\varphi_1 \leftrightarrow \varphi_2 := (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$
- ▷ $\mathbf{F}\,\varphi_1 := \top\mathbf{U}\varphi_1$, $\mathbf{G}\varphi_1 := \neg\mathbf{F}\neg\varphi_1$, $\neg\mathbf{X}\varphi_1 := \mathbf{X}\neg\varphi_1$
- ▷ N.B. $\varphi_1\mathbf{R}\varphi_2 := \neg(\neg\varphi_1\mathbf{U}\neg\varphi_2)$, "Releases"

LTL: Formal Semantics

$$
\begin{array}{lcll}
\pi, s_i & \models & a & \text{iff} \quad a \in L(s_i) \\
\pi, s_i & \models & \neg\varphi & \text{iff} \quad \pi, s_i \not\models \varphi \\
\pi, s_i & \models & \varphi \wedge \psi & \text{iff} \quad \pi, s_i \models \varphi \textit{ and } \\
 & & & \qquad \pi, s_i \models \psi \\
\pi, s_i & \models & \mathbf{X}\varphi & \text{iff} \quad \pi, s_{i+1} \models \varphi \\
\pi, s_i & \models & \mathbf{F}\varphi & \text{iff} \quad \textit{for some } j \geq i : \pi, s_j \models \varphi \\
\pi, s_i & \models & \mathbf{G}\varphi & \text{iff} \quad \textit{for all } j \geq i : \pi, s_j \models \varphi \\
\pi, s_i & \models & \varphi \mathbf{U} \psi & \text{iff} \quad \textit{for some } j \geq i : \pi, s_j \models \psi \textit{ and} \\
 & & & \qquad \textit{for all } i \leq k < j : \pi, s_k \models \varphi
\end{array}
$$

LTL: Some Noteworthy Examples

- Safety: "it never happens that a train is arriving and the bar is up"

$$\mathbf{G}(\neg(\text{train\_arriving} \wedge \text{bar\_up}))$$

- Liveness: "if input, then eventually output"

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

- Fairness: "infinitely often send "

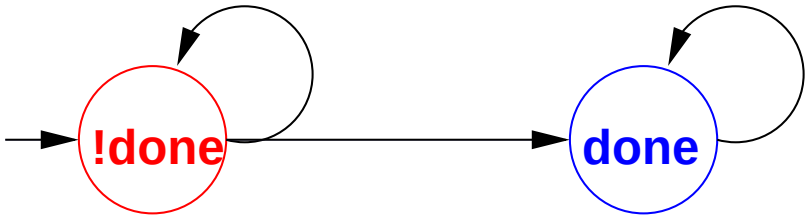
$$\mathbf{GF}\text{send}$$

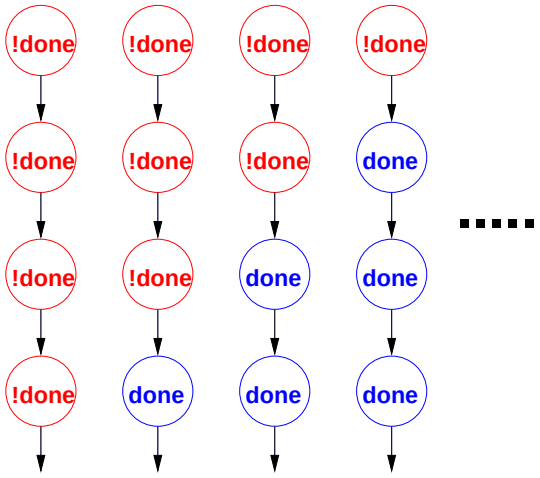
- Strong fairness: "infinitely often send implies infinitely often recv."

$$\mathbf{GF}\text{send} \rightarrow \mathbf{GF}\text{recv}$$

Model Checking in LTL

- LTL properties are evaluated over paths, i.e., over infinite, linear sequences of states:
  $\pi = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_t \rightarrow s_{t+1} \rightarrow \cdots$
- Given an infinite sequence $\pi = s_0, s_1, s_2, \ldots$
  - $\pi, s_i \models \phi$ if ϕ is true in state s_i of π.
  - $\pi \models \phi$ if ϕ is true in the initial state s_0 of π.
- The LTL model checking problem $\mathcal{M} \models \phi$
  - check if $\pi \models \phi$ for every path π of the Kripke structure $\mathcal{M}$ (e.g., $\phi = \mathbf{F}done$)





LTL tableaux rules

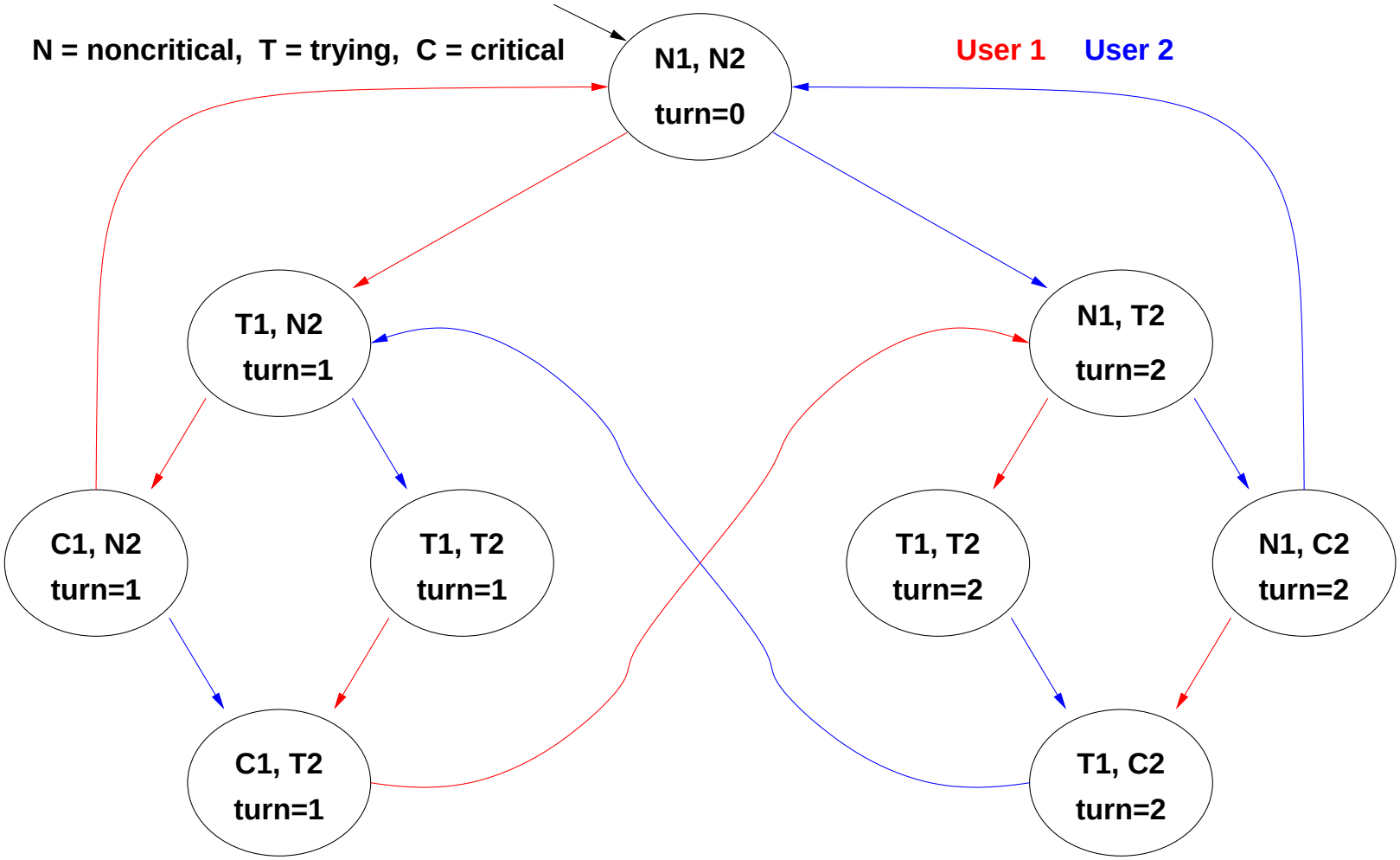
- Let φ_1 and φ_2 be LTL formulae:

$$\begin{aligned} \mathbf{F}\varphi_1 &\iff (\varphi_1 \vee \mathbf{XF}\varphi_1) \\ \mathbf{G}\varphi_1 &\iff (\varphi_1 \wedge \mathbf{XG}\varphi_1) \\ \varphi_1 \mathbf{U} \varphi_2 &\iff (\varphi_2 \vee (\varphi_1 \wedge \mathbf{X}(\varphi_1 \mathbf{U} \varphi_2))) \end{aligned}$$

- If applied recursively, rewrite an LTL formula in terms of atomic and $\mathbf{X}$-formulas:

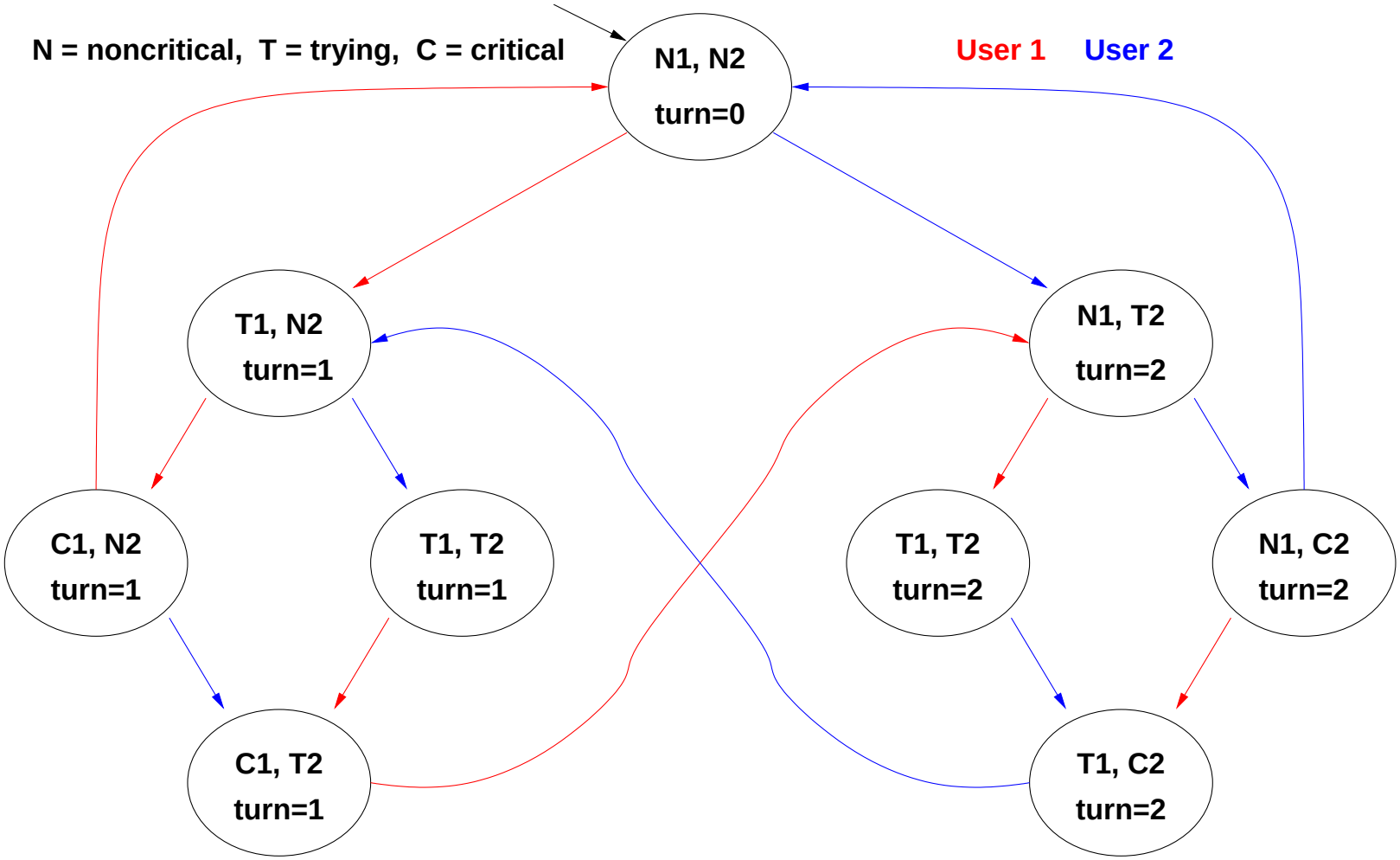
$$(p\mathbf{U}q) \wedge (\mathbf{G}\neg p) \Longrightarrow (q \vee (p \wedge \mathbf{X}(p\mathbf{U}q))) \wedge (\neg p \wedge \mathbf{XG}\neg p)$$

Example 1: mutual exclusion (safety)



$$M \models \mathbf{G}\neg(C_1 \wedge C_2) \ ?$$

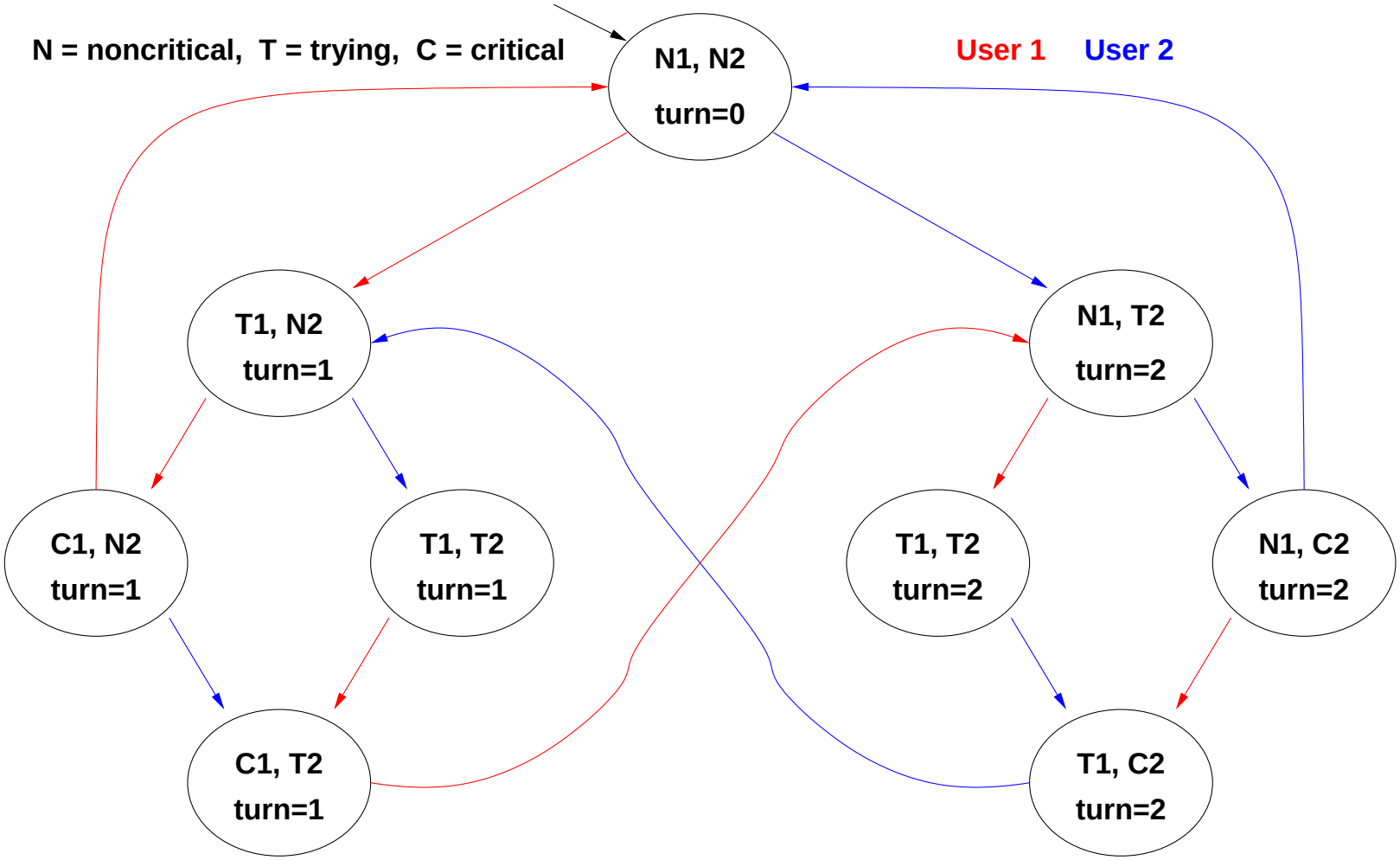
Example 1: mutual exclusion (safety) [cont.]



$$M \models \mathbf{G}\neg(C_1 \wedge C_2) \ ?$$

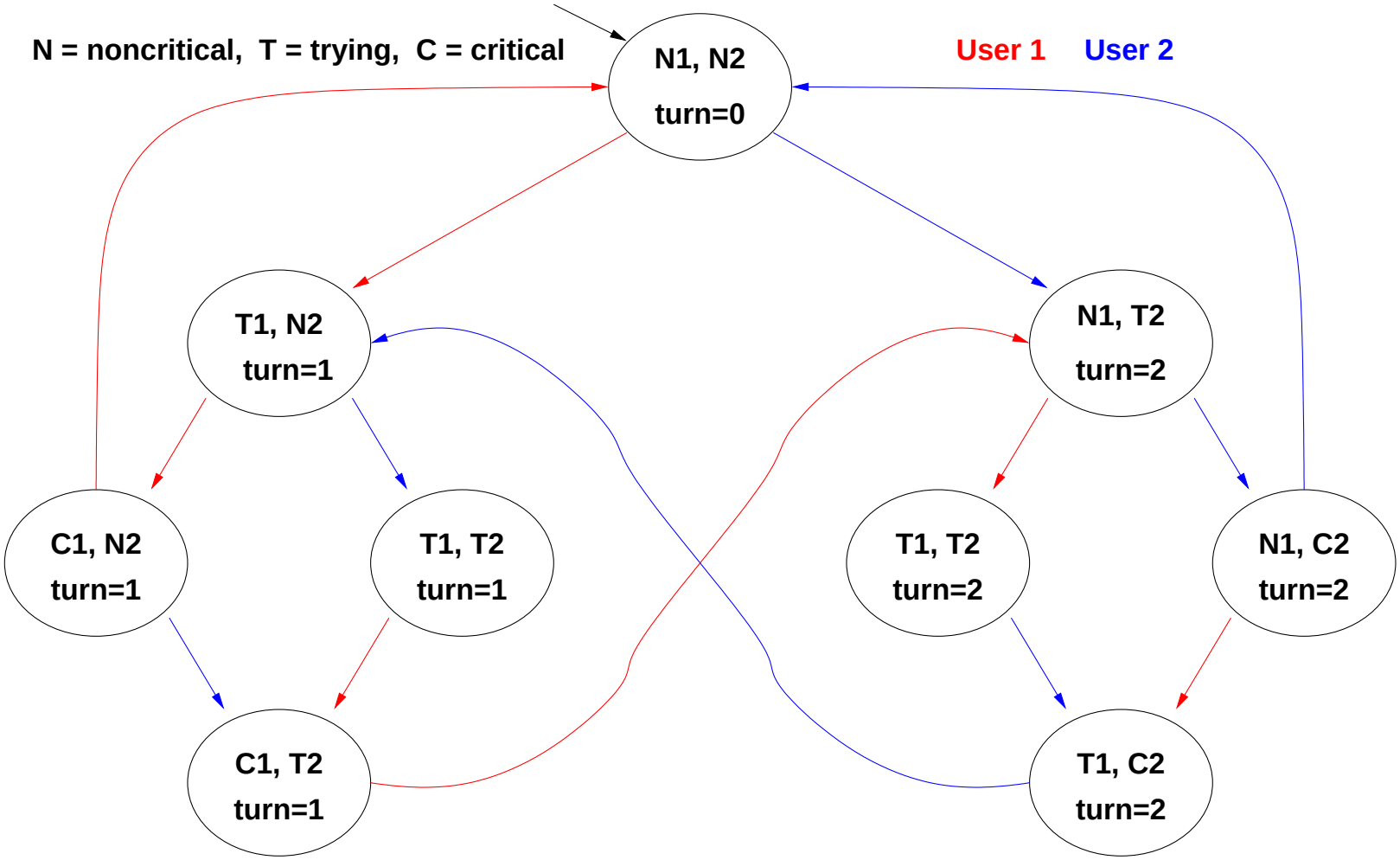
YES: There is no reachable state in which $(C_1 \wedge C_2)$ holds!

Example 2: liveness



$$M \models \mathbf{F}C_1 \ ?$$

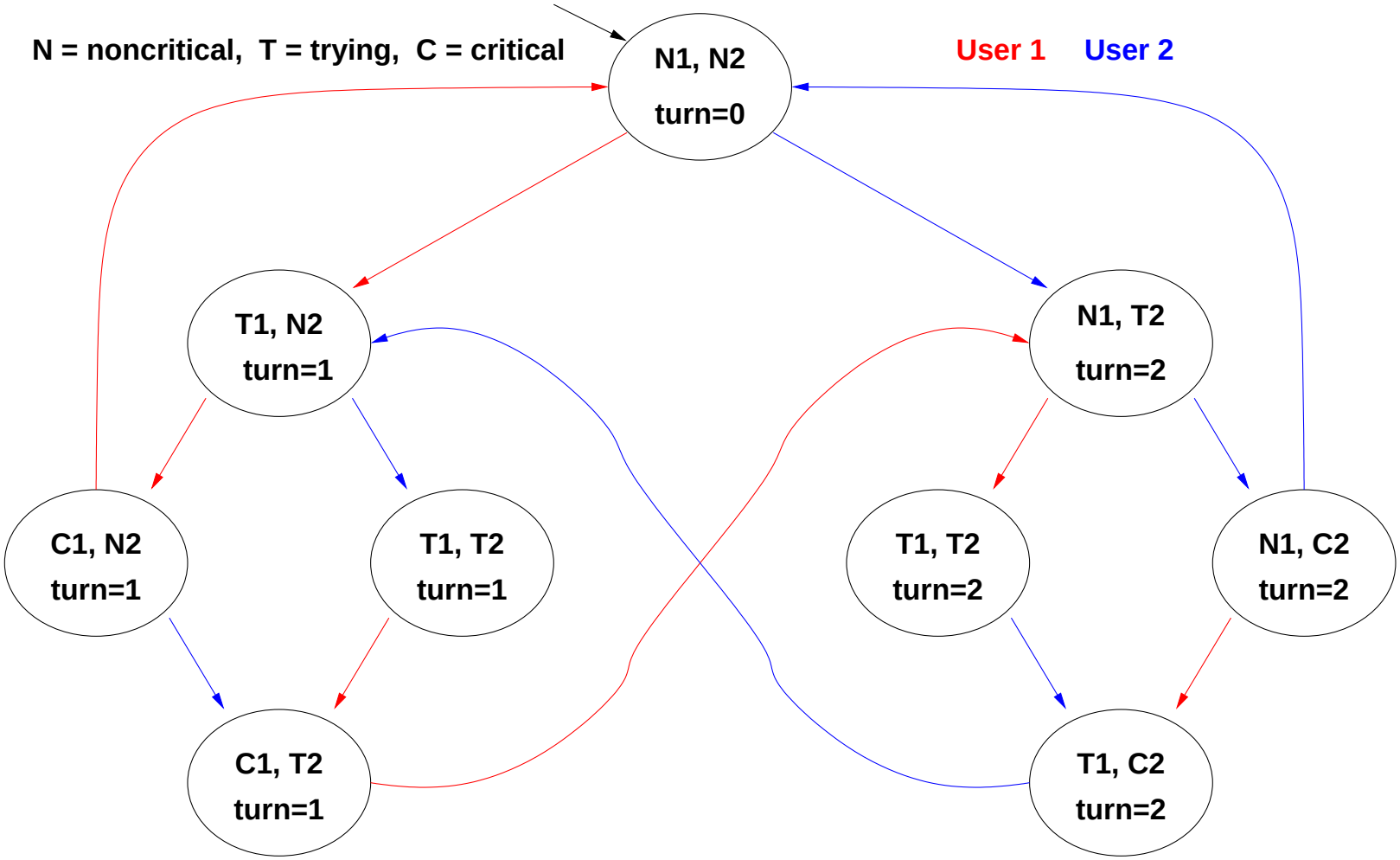
Example 2: liveness [cont.]



$$M \models \mathbf{F}C_1 \ ?$$

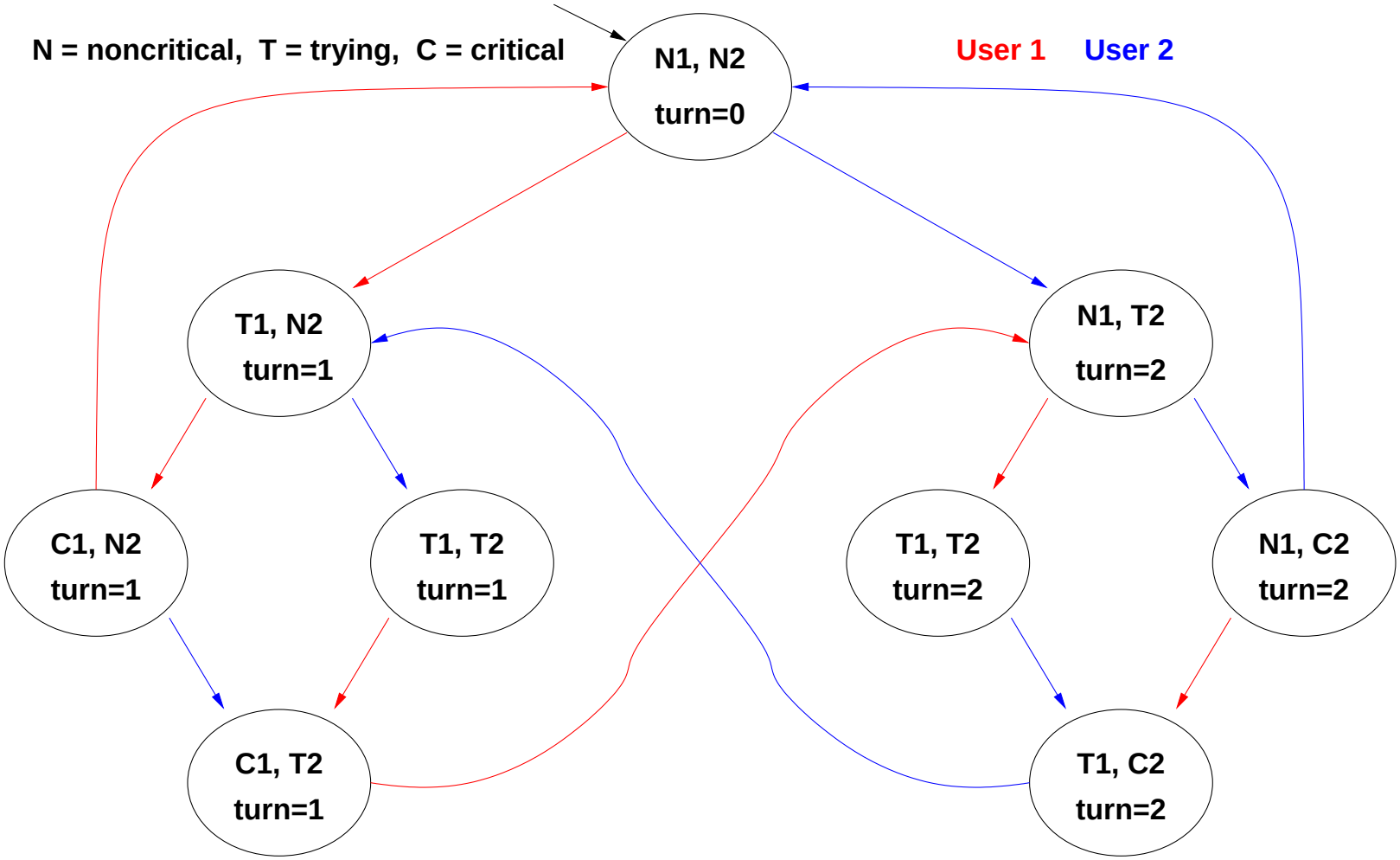
NO: there is an infinite cyclic solution in which C_1 never holds!

Example 3: liveness



$$M \models \mathbf{G}(T_1 \rightarrow \mathbf{F}C_1) \ ?$$

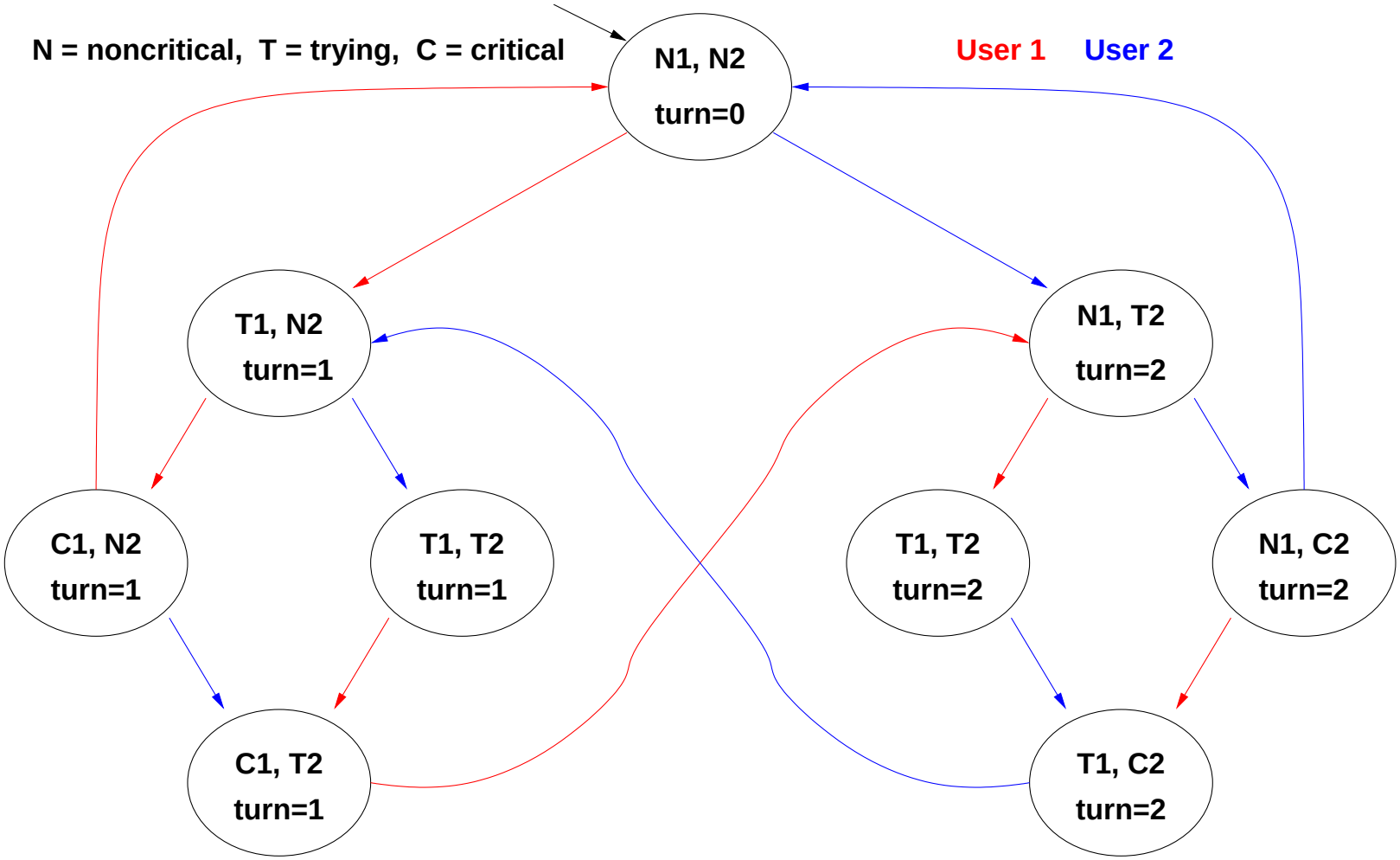
Example 3: liveness [cont.]



$$M \models \mathbf{G}(T_1 \rightarrow \mathbf{F}C_1) \ ?$$

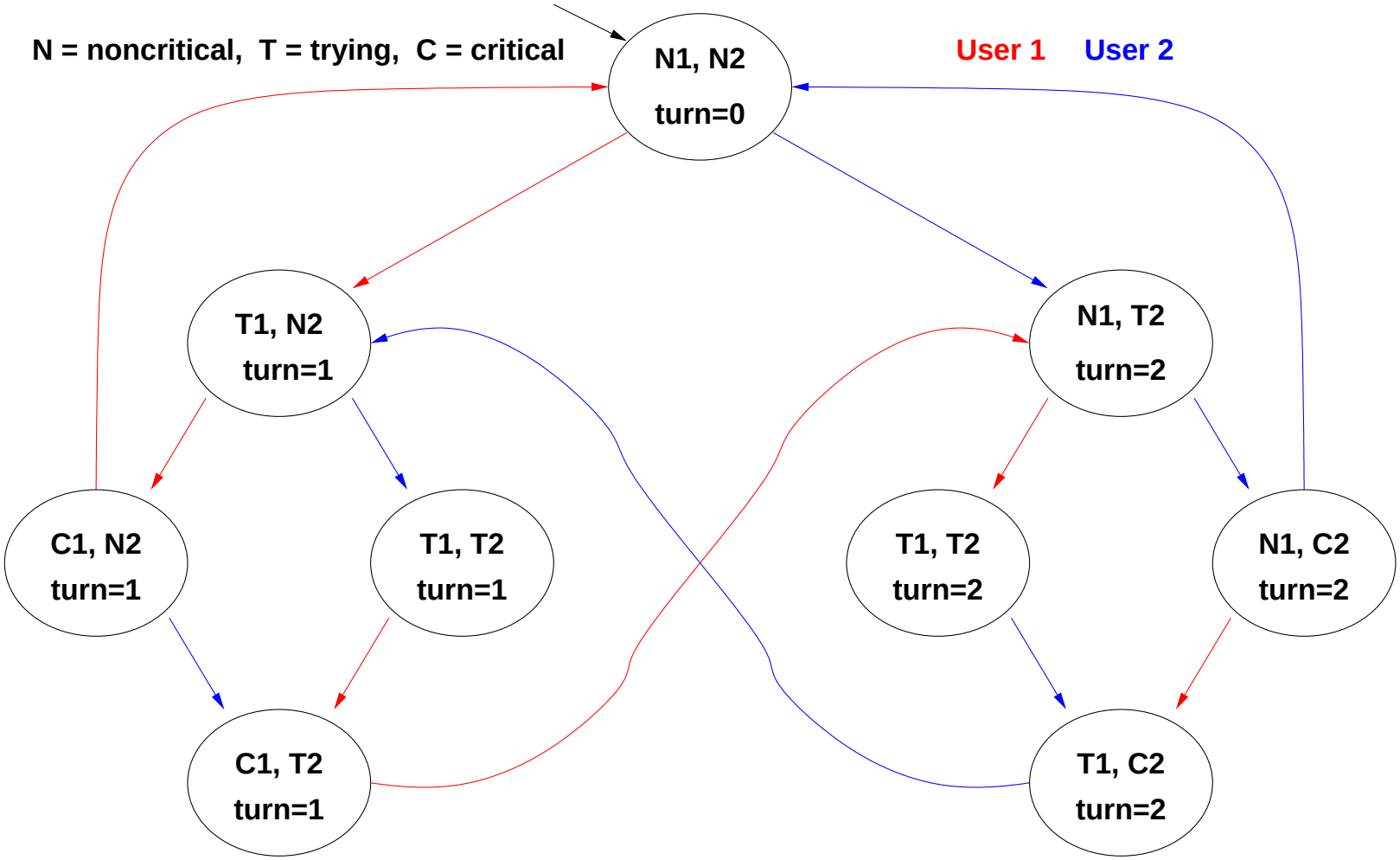
YES: every path starting from each state where T_1 holds passes through a state where C_1 holds

Example 4: fairness



$$M \models \mathbf{GF}C_1 \ ?$$

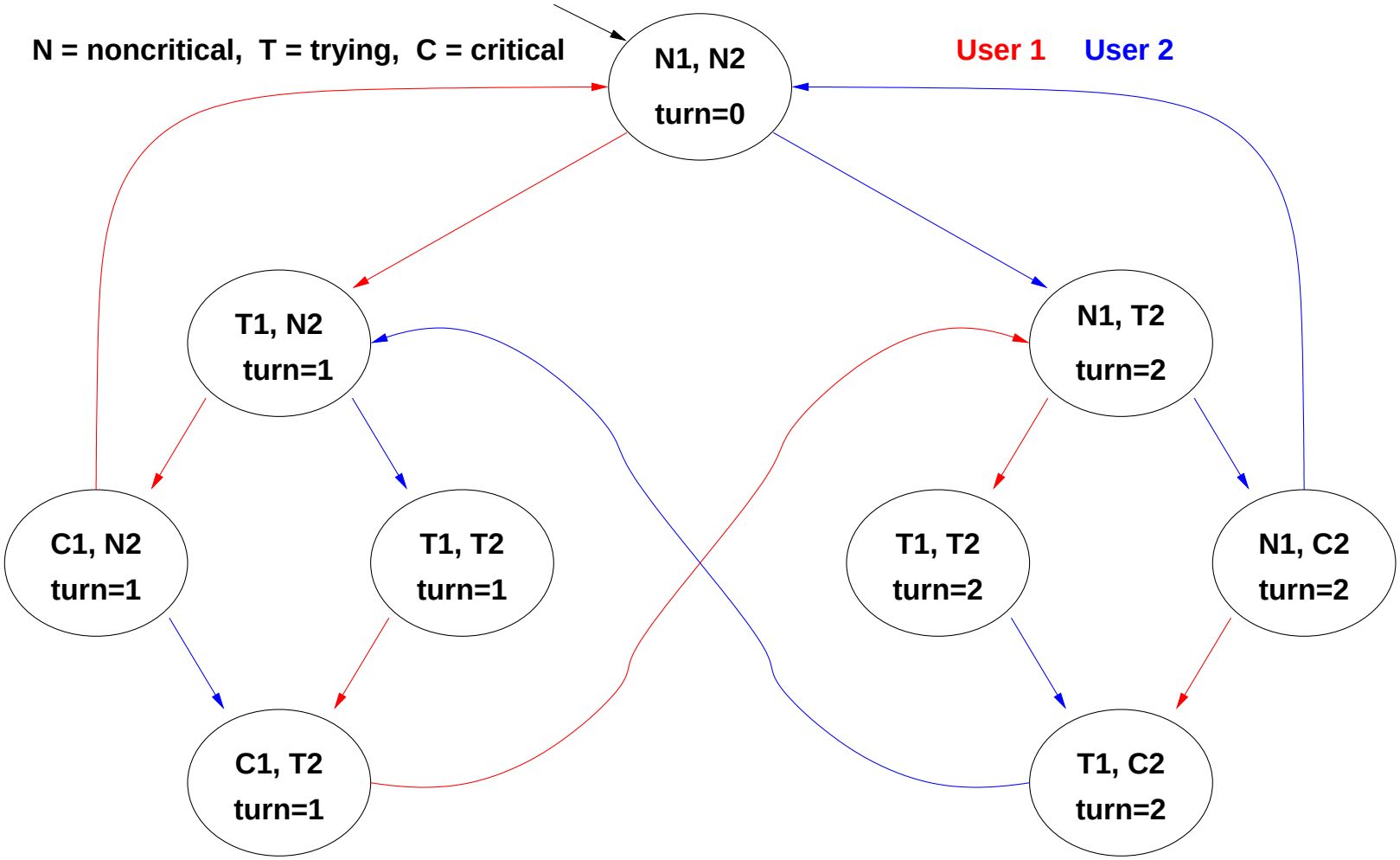
Example 4: fairness [cont.]



$$M \models \mathbf{GF}C_1 \ ?$$

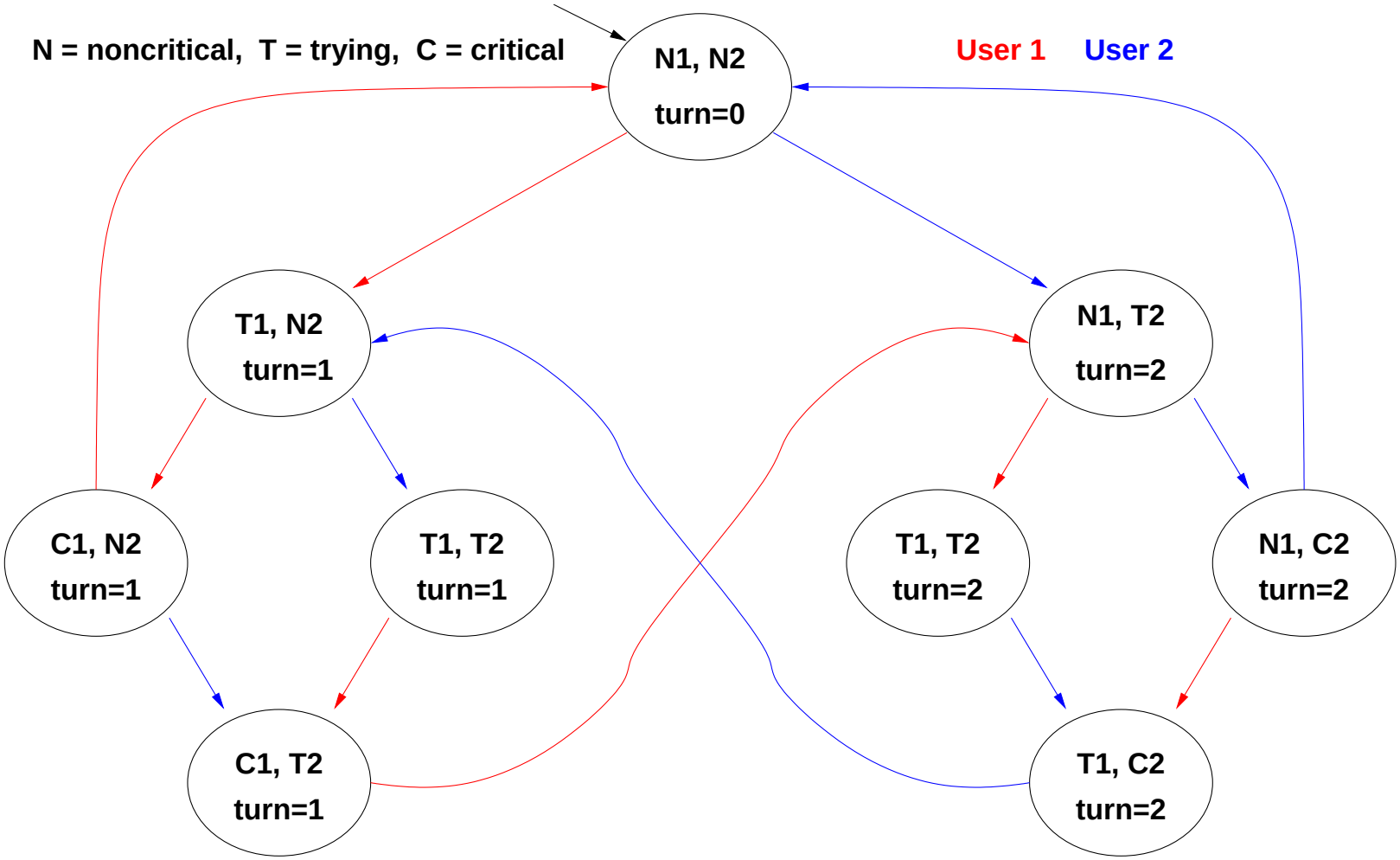
NO: e.g., in the initial state, there is an infinite cyclic solution in which C_1 never holds!

Example 5: strong fairness



$$M \models \mathbf{GF}T_1 \rightarrow \mathbf{GF}C_1 \ ?$$

Example 5: strong fairness [cont.]



$$M \models \mathbf{GF}T_1 \rightarrow \mathbf{GF}C_1 \ ?$$

YES: every path which visits T_1 infinitely often also visits C_1 infinitely often (see liveness property of previous example).

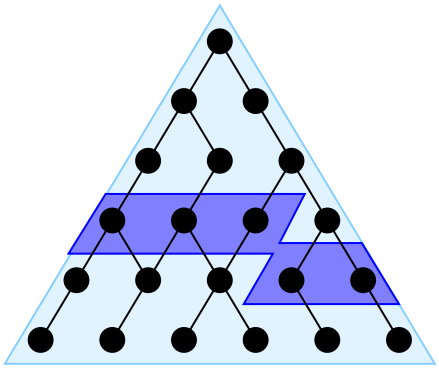
Computation Tree Logic (CTL): intuitions

CTL is given by the standard boolean logic enhanced with the operators **AX**, **AG**, **AF**, **AU**, **EX**, **EG**, **EF**, **EU**:

- "Necessarily Next" **AX**: $\mathbf{AX}\varphi$ is true in s_t iff φ is true in every successor state s_{t+1}
- "Possibly Next" **EX**: $\mathbf{EX}\varphi$ is true in s_t iff φ is true in one successor state s_{t+1}
- "Necessarily in the future" (or "Inevitably") **AF**: $\mathbf{AF}\varphi$ is true in s_t iff φ is inevitably true in **some** $s_{t'}$ with $t' \geq t$
- "Possibly in the future" (or "Possibly") **EF**: $\mathbf{EF}\varphi$ is true in s_t iff φ may be true in **some** $s_{t'}$ with $t' \geq t$
- "Globally" (or "always") **AG**: $\mathbf{AG}\varphi$ is true in s_t iff φ is true in **all** $s_{t'}$ with $t' \geq t$
- "Possibly henceforth" **EG**: $\mathbf{EG}\varphi$ is true in s_t iff φ is possibly true henceforth
- "Necessarily Until" **AU**: $\mathbf{A}(\varphi \mathbf{U} \psi)$ is true in s_t iff necessarily φ holds until ψ holds.
- "Possibly Until" **EU**: $\mathbf{E}(\varphi \mathbf{U} \psi)$ is true in s_t iff possibly φ holds until ψ holds.

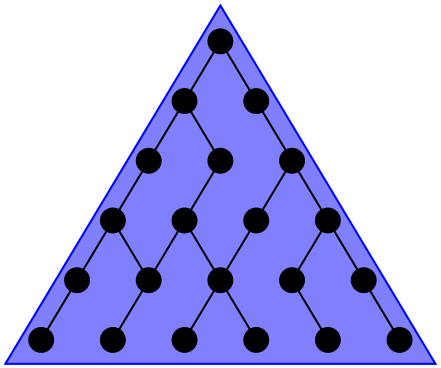
CTL: intuitions [cont.]

finally P



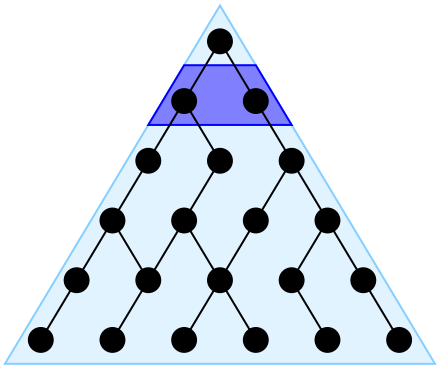
AF P

globally P



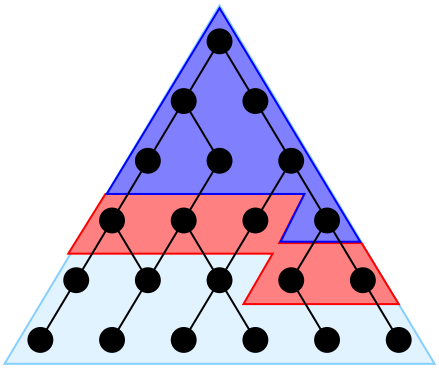
AG P

next P

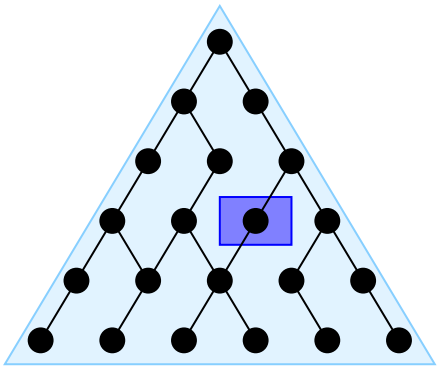


AX P

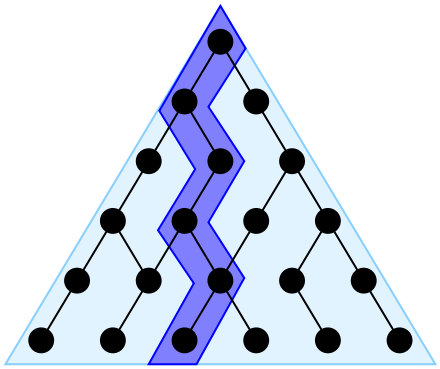
P until q



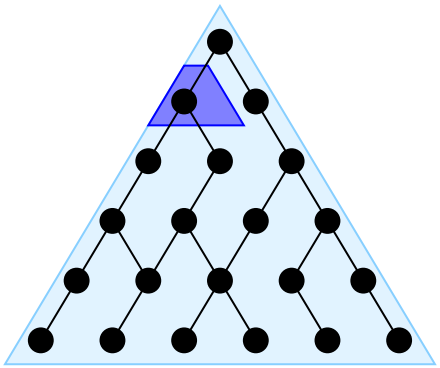
A[P U q]



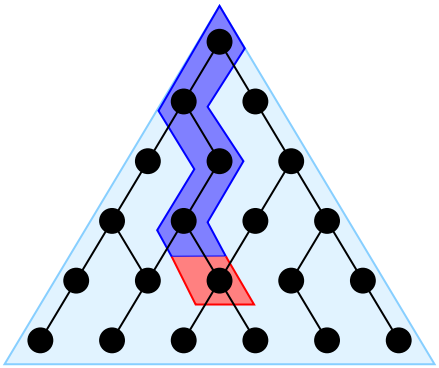
EF P



EG P



EX P



E[P U q]

Computation Tree Logic (CTL): Syntax

- An atomic proposition is a CTL formula;
- if φ_1 and φ_2 are CTL formulae, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$ are CTL formulae;
- if φ_1 and φ_2 are CTL formulae, then $\mathbf{AX}\varphi_1$, $\mathbf{A}(\varphi_1\mathbf{U}\varphi_2)$, $\mathbf{AG}\varphi_1$, $\mathbf{AF}\varphi_1$, $\mathbf{EX}\varphi_1$, $\mathbf{E}(\varphi_1\mathbf{U}\varphi_2)$, $\mathbf{EG}\varphi_1$, $\mathbf{EF}\varphi_1$,, are CTL formulae.

N.B: CTL can be defined in terms of $\wedge$, $\neg$, $\mathbf{EX}$, $\mathbf{EG}$, $\mathbf{EU}$ only:

- $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 := (\neg\varphi_1 \vee \varphi_2)$,
  $\varphi_1 \leftrightarrow \varphi_2 := (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$
- $\mathbf{A}(\varphi_1\mathbf{U}\varphi_2) := \neg\mathbf{E}(\neg\varphi_2\mathbf{U}(\neg\varphi_1 \wedge \neg\varphi_2)) \wedge \neg\mathbf{EG}\neg\varphi_2$, $\mathbf{EF}\ \varphi_1 := \mathbf{E}(\top\mathbf{U}\varphi_1)$,
  $\mathbf{AG}\varphi_1 := \neg\mathbf{EF}\neg\varphi_1$, $\mathbf{AF}\ \varphi_1 := \neg\mathbf{EG}\neg\varphi_1$, $\mathbf{AX}\varphi_1 := \neg\mathbf{EX}\neg\varphi_1$

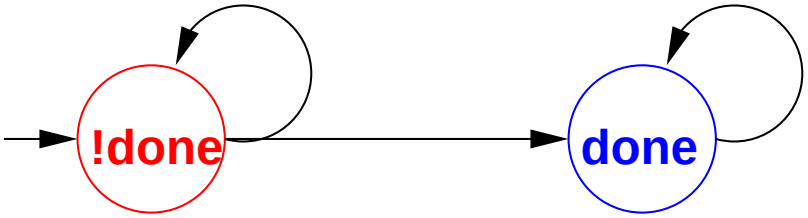
CTL Formal Semantics

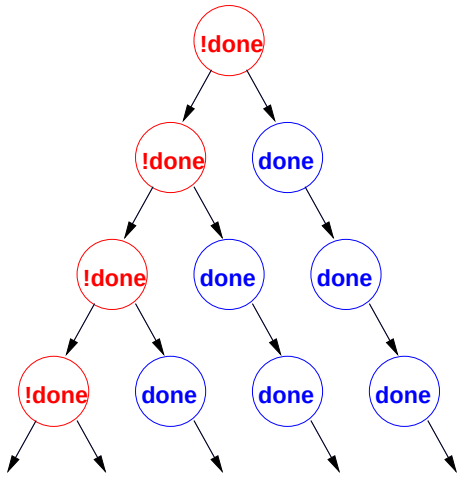
Let $(s_i, s_{i+1}, \ldots)$ be a path outgoing from state s_i in M

$$
\begin{array}{lcll}
M, s_i & \models & a & \text{iff } a \in L(s_i) \\
M, s_i & \models & \neg\varphi & \text{iff } M, s_i \not\models \varphi \\
M, s_i & \models & \varphi \vee \psi & \text{iff } M, s_i \models \varphi \text{ or } M, s_i \models \psi \\
M, s_i & \models & AX\varphi & \text{iff } for\ all\ (s_i, s_{i+1}, \ldots),\ s_{i+1} \models \varphi \\
M, s_i & \models & EX\varphi & \text{iff } for\ some\ (s_i, s_{i+1}, \ldots),\ s_{i+1} \models \varphi \\
M, s_i & \models & AG\varphi & \text{iff } for\ all\ (s_i, s_{i+1}, \ldots),\ for\ all\ j \geq i\ : M, s_j \models \varphi \\
M, s_i & \models & EG\varphi & \text{iff } for\ some\ (s_i, s_{i+1}, \ldots),\ for\ all\ j \geq i\ : M, s_j \models \varphi \\
M, s_i & \models & AF\varphi & \text{iff } for\ all\ (s_i, s_{i+1}, \ldots),\ for\ some\ j \geq i\ : M, s_j \models \varphi \\
M, s_i & \models & EF\varphi & \text{iff } for\ some\ (s_i, s_{i+1}, \ldots),\ for\ some\ j \geq i\ : M, s_j \models \varphi \\
M, s_i & \models & A(\varphi U \psi) & \text{iff } for\ all\ (s_i, s_{i+1}, \ldots),\ for\ some\ j \geq i\ : M, s_j \models \psi\ and \\
 & & & \quad for\ all\ i \leq k < j\ : M, s_k \models \varphi \\
M, s_i & \models & E(\varphi U \psi) & \text{iff } for\ some\ (s_i, s_{i+1}, \ldots),\ for\ some\ j \geq i\ : M, s_j \models \psi\ and \\
 & & & \quad for\ all\ i \leq k < j\ : M, s_k \models \varphi
\end{array}
$$

Model Checking in CTL

- CTL properties (e.g. $\mathbf{AF}done$) are evaluated over computation trees.





- Every temporal operator $(\mathbf{F}, \mathbf{G}, \mathbf{X}, \mathbf{U})$ preceded by a path quantifier ($\mathbf{A}$ or $\mathbf{E}$).
- Universal modalities $(\mathbf{AF}, \mathbf{AG}, \mathbf{AX}, \mathbf{AU})$: the temporal formula is true in **all** the paths starting in the current state.
- Existential modalities $(\mathbf{EF}, \mathbf{EG}, \mathbf{EX}, \mathbf{EU})$: the temporal formula is true in **some** path starting in the current state.

Model Checking in CTL, $\mathcal{M} \models \phi$:

Check if $\mathcal{M}, s \models \phi$ for every initial state $s \in I$ of the Kripke structure M

CTL tableaux rules

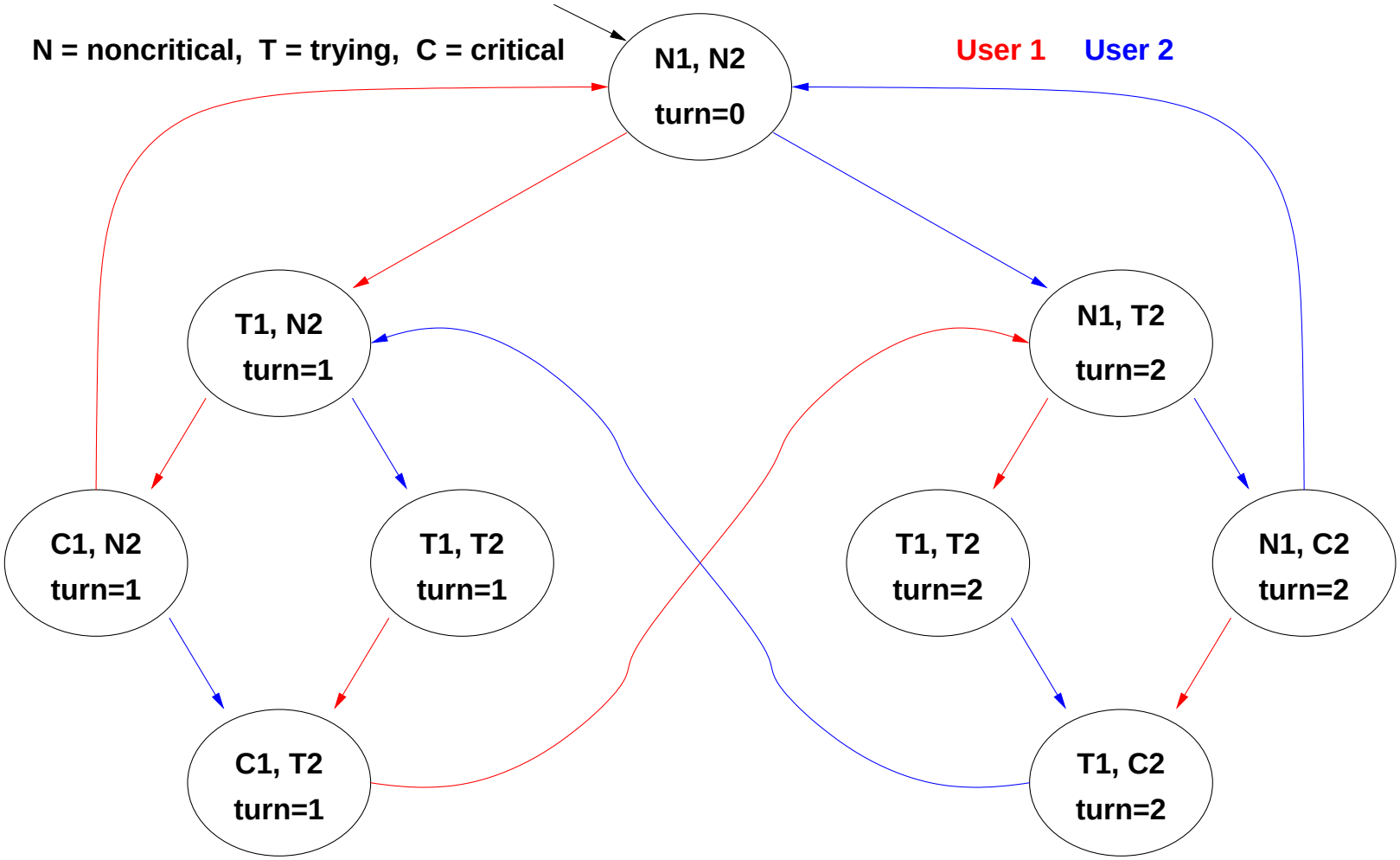
- Let φ_1 and φ_2 be CTL formulae:

$$\begin{aligned}
\mathbf{AF}\varphi_1 &\iff (\varphi_1 \vee \mathbf{AXAF}\varphi_1)\\
\mathbf{AG}\varphi_1 &\iff (\varphi_1 \wedge \mathbf{AXAG}\varphi_1)\\
\mathbf{A}(\varphi_1\mathbf{U}\varphi_2) &\iff (\varphi_2 \vee (\varphi_1 \wedge \mathbf{AXA}(\varphi_1\mathbf{U}\varphi_2)))\\
\mathbf{EF}\varphi_1 &\iff (\varphi_1 \vee \mathbf{EXEF}\varphi_1)\\
\mathbf{EG}\varphi_1 &\iff (\varphi_1 \wedge \mathbf{EXEG}\varphi_1)\\
\mathbf{E}(\varphi_1\mathbf{U}\varphi_2) &\iff (\varphi_2 \vee (\varphi_1 \wedge \mathbf{EXE}(\varphi_1\mathbf{U}\varphi_2)))
\end{aligned}$$

- Recursive definitions of $\mathbf{AF}$, $\mathbf{AG}$, $\mathbf{AU}$, $\mathbf{EF}$, $\mathbf{EG}$, $\mathbf{EU}$.
- If applied recursively, rewrite a CTL formula in terms of atomic, $\mathbf{AX}$- and $\mathbf{EX}$-formulas:

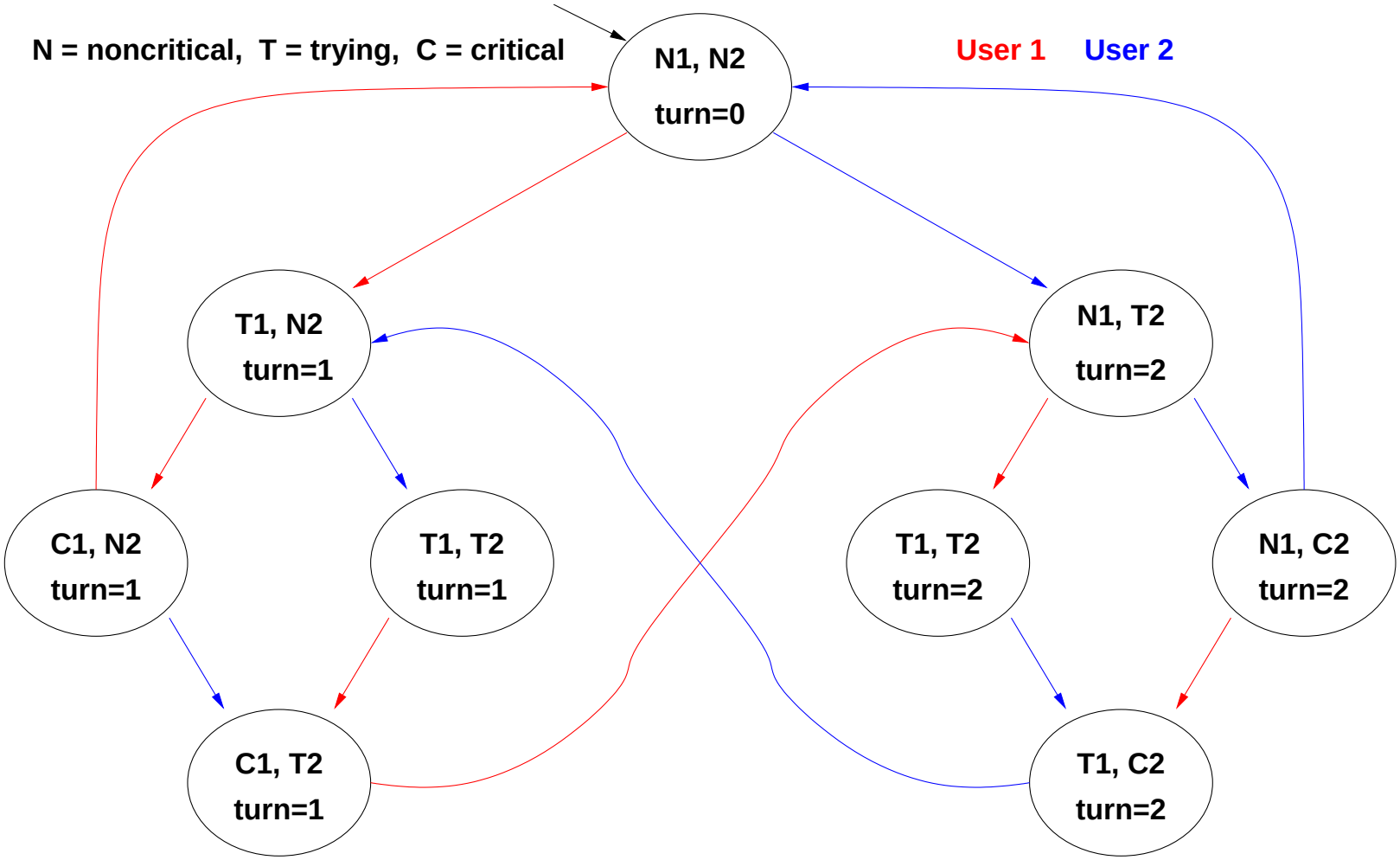
$$\mathbf{A}(p\mathbf{U}q) \wedge (\mathbf{EG}\neg p) \Longrightarrow (q \vee (p \wedge \mathbf{AXA}(p\mathbf{U}q))) \wedge (\neg p \wedge \mathbf{EXEG}\neg p)$$

Example 1: mutual exclusion (safety)



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

Example 1: mutual exclusion (safety) [cont.]

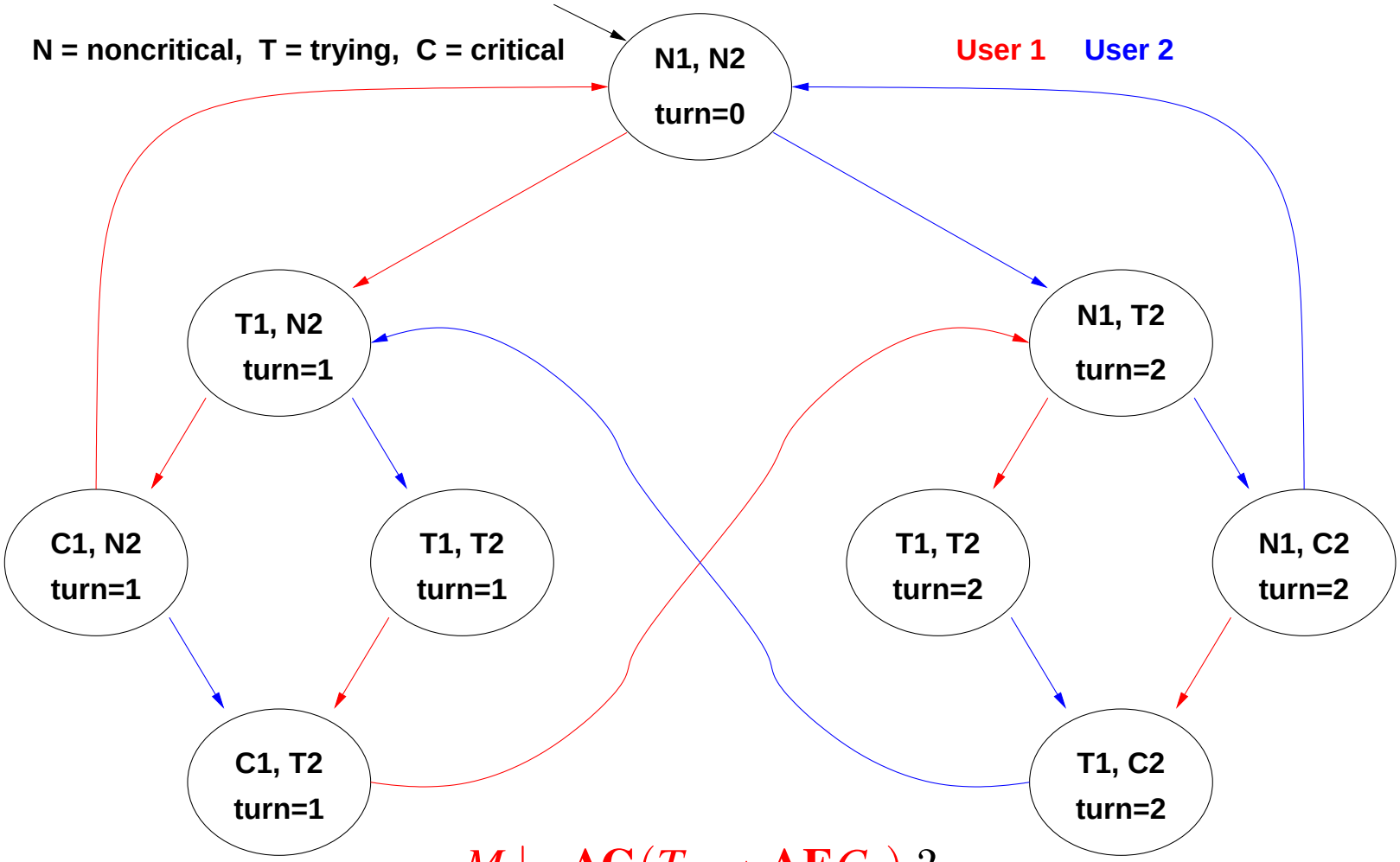


$$M \models \mathbf{AG}\neg(C_1 \wedge C_2)\ ?$$

YES: There is no reachable state in which $(C_1 \wedge C_2)$ holds!

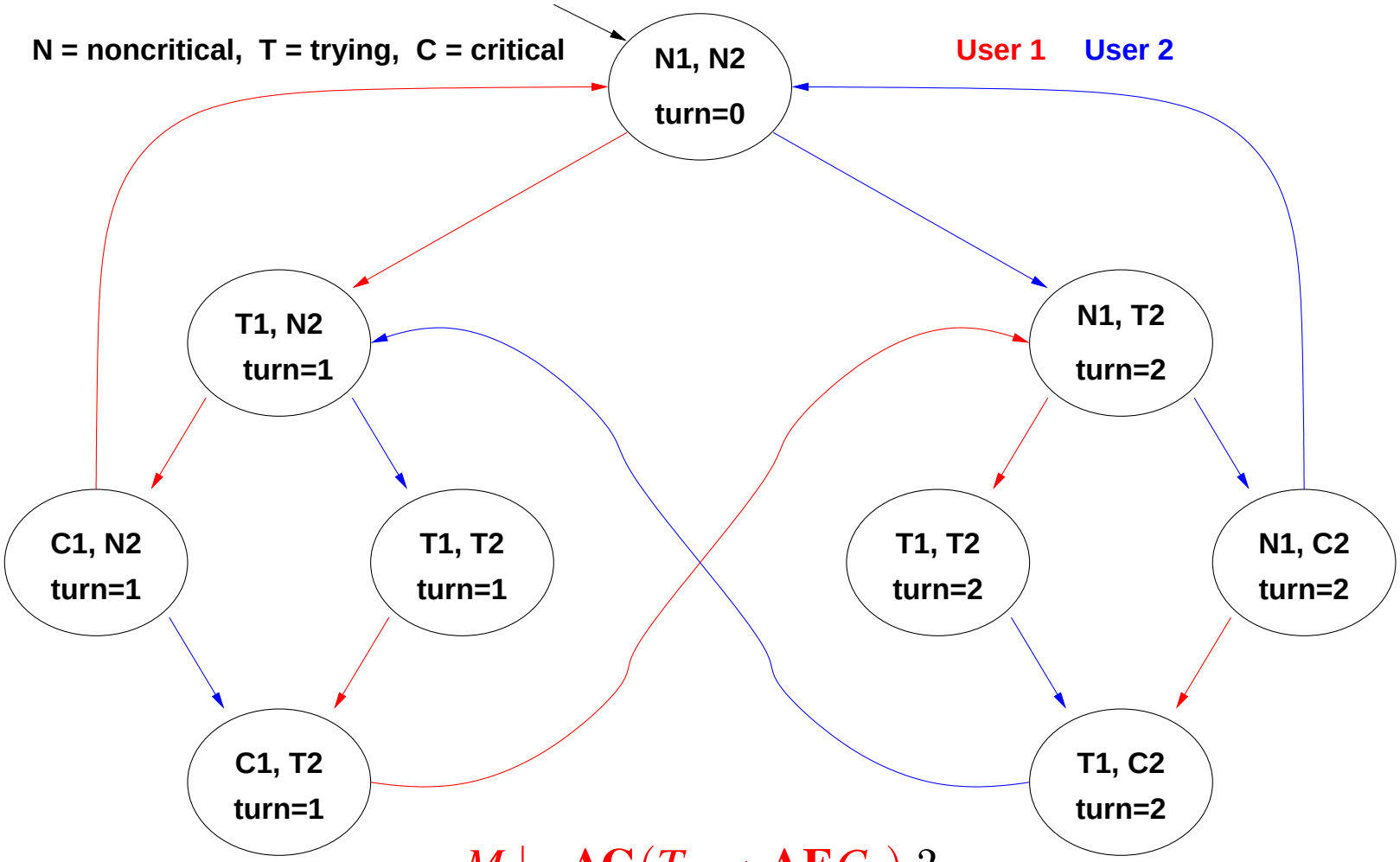
(Same as the $\mathbf{G}\neg(C_1 \wedge C_2)$ in LTL.)

Example 2: liveness



$$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}\, C_1)\ ?$$

Example 2: liveness [cont.]

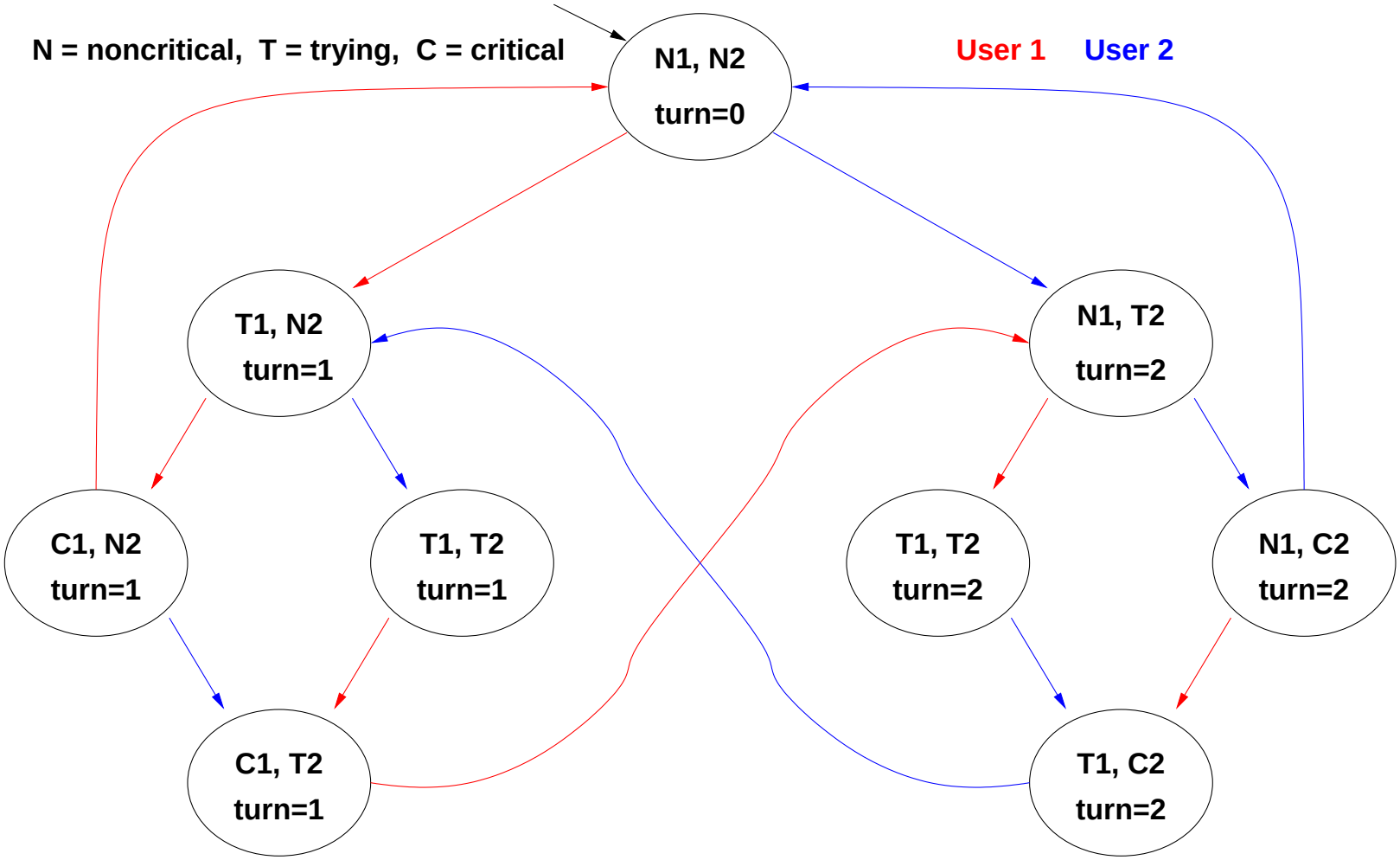


$$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}\, C_1)\ ?$$

YES: every path starting from each state where T_1 holds passes through a state where C_1 holds

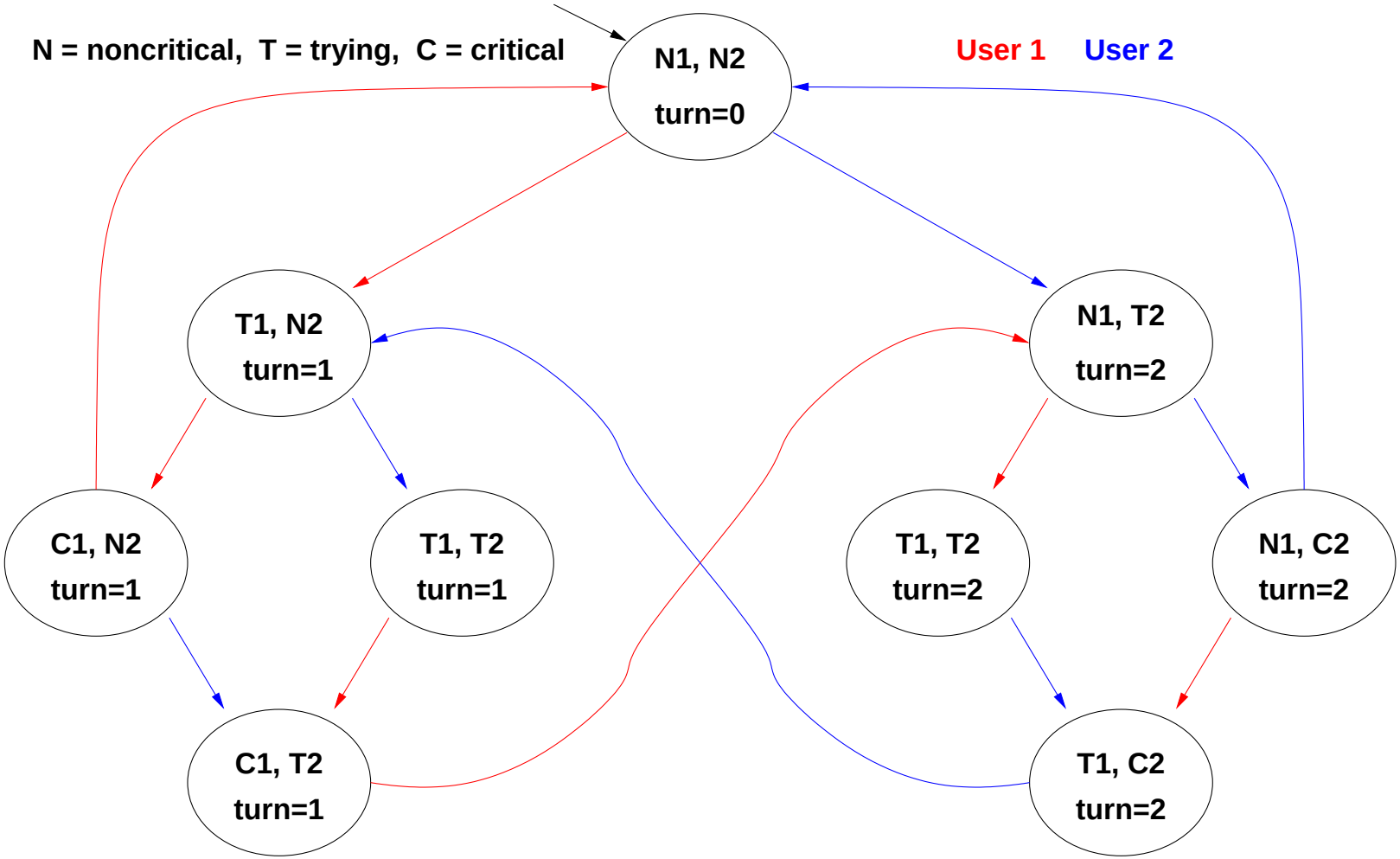
(Same as $\mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$ in LTL.)

Example 3: fairness



$$M \models \mathbf{AGAF}C_1 \ ?$$

Example 3: fairness [cont.]

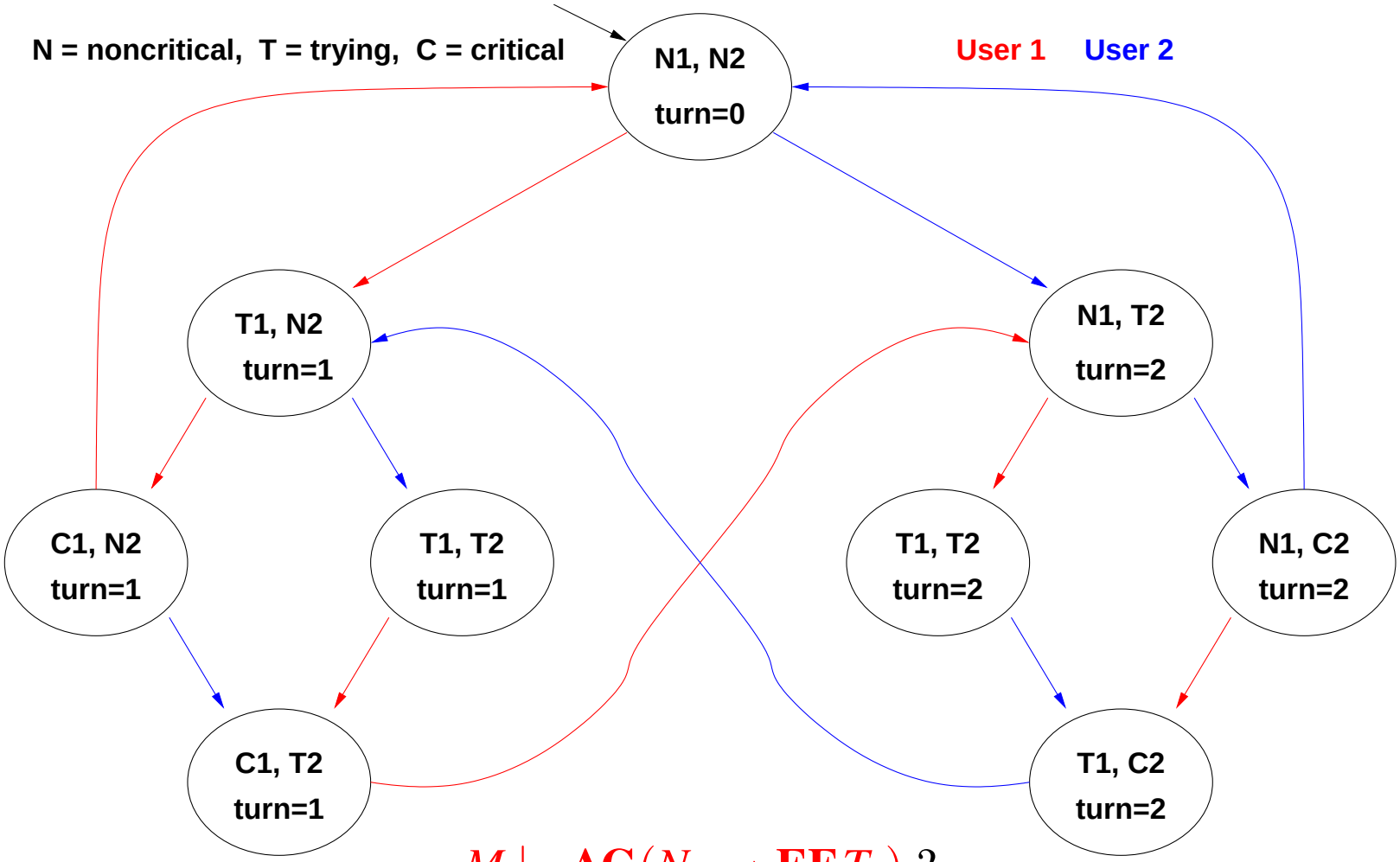


$$M \models \mathbf{AGAF}C_1 \ ?$$

NO: e.g., in the initial state, there is an infinite cyclic solution in which C_1 never holds!

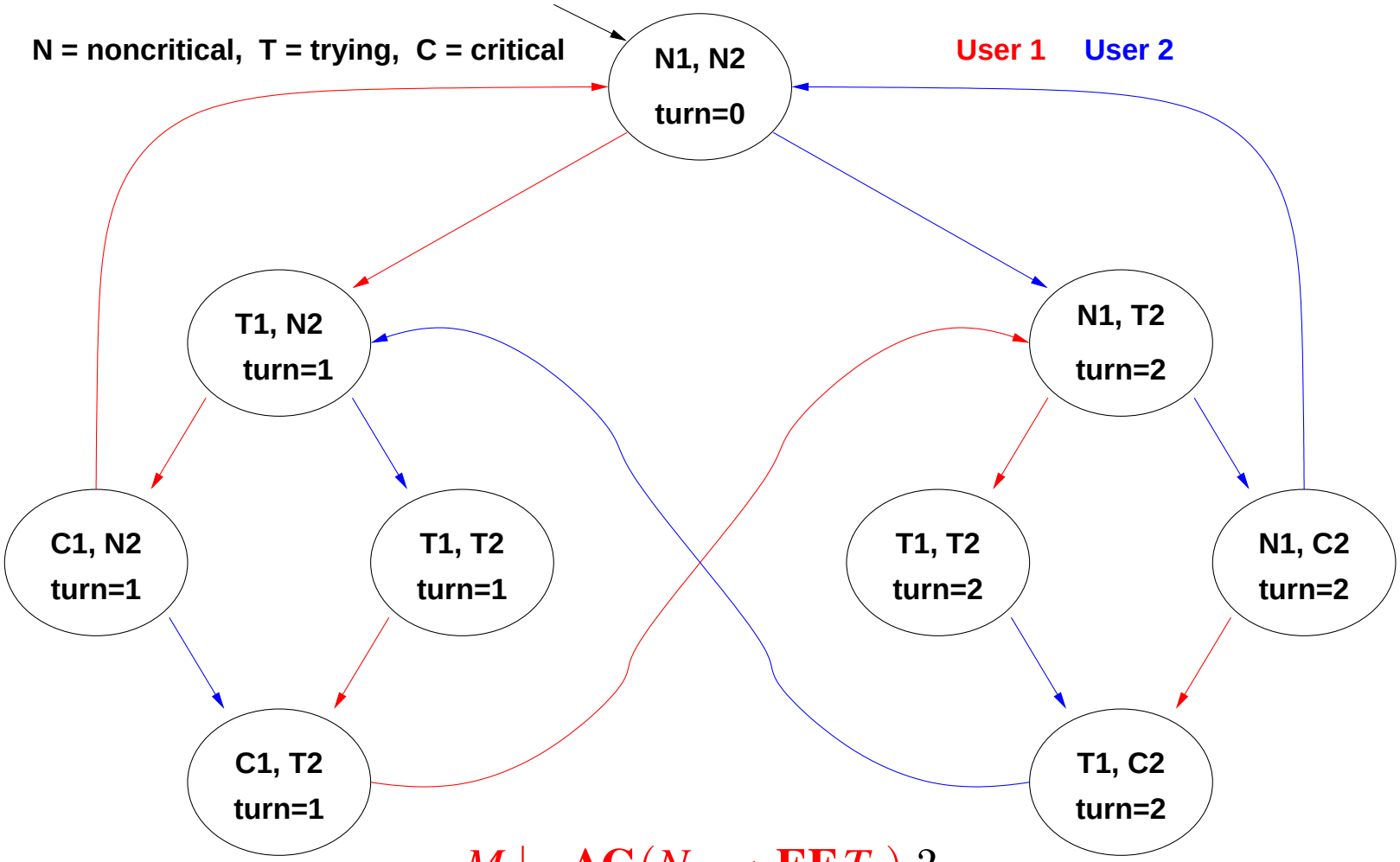
(Same as $\mathbf{GF}C_1$ in LTL.)

Example 4: blocking [cont.]



$$M \models \mathbf{AG}(N_1 \rightarrow \mathbf{EF}\, T_1)\ ?$$

Example 4: blocking [cont.]



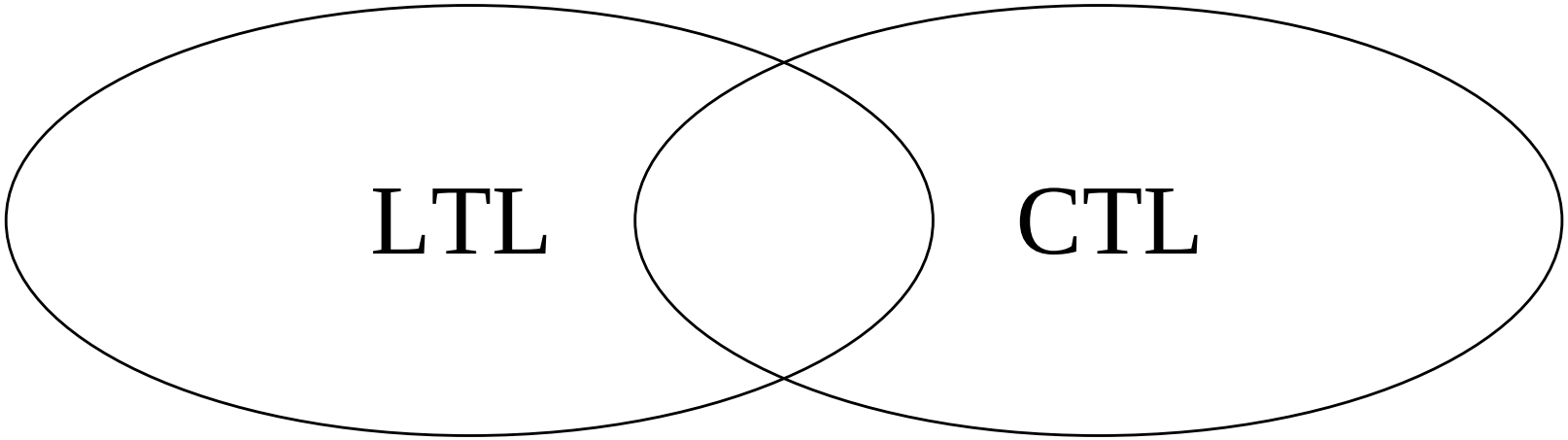
$$M \models \mathbf{AG}(N_1 \rightarrow \mathbf{EF}\, T_1)\ ?$$

YES: from each state where N_1 holds there is a path leading to a state where T_1 holds

(No corresponding LTL formula.)

LTL vs. CTL: expressiveness

- many CTL formulas cannot be expressed in LTL
  (e.g., those containing existentially quantified subformulas)
  E.g., $\mathbf{AG}(N_1 \rightarrow \mathbf{EF}T_1)$, $AFAG\varphi$
- many LTL formulas cannot be expressed in CTL
  (e.g. fairness LTL formulas)
  E.g., $\mathbf{GF}T_1 \rightarrow \mathbf{GF}C_1$, $FG\varphi$
- some formluas can be expressed both in LTL and in CTL (typically LTL formulas with operators of nesting depth 1, and/or with operators occurring positively)
  E.g., $\mathbf{G}\neg(C_1 \wedge C_2)$, $\mathbf{F}C_1$, $\mathbf{G}(T_1 \rightarrow \mathbf{F}C_1)$, $\mathbf{GF}C_1$



LTL vs. CTL: M.C. Algorithms

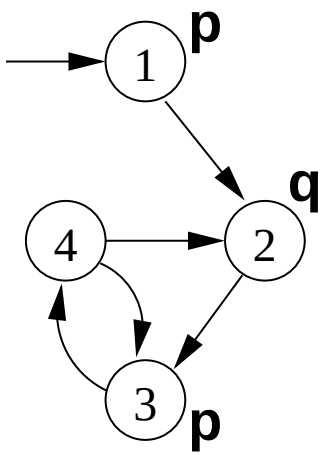
- LTL M.C. problems are typically handled with automata- based M.C. approaches (Wolper & Vardi)
- CTL M.C. problems are typically handled with symbolic M.C. approaches (Clarke & McMillan)
- LTL M.C. problems can be reduced to CTL M.C. problems under fairness constraints (Clarke et al.)

Content

CTL Model Checking

CTL Model Checking is a formal verification technique where...

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



- ...the property is expressed a CTL formula φ:

$$\mathbf{AG}(p \rightarrow \mathbf{AF}q)$$

- ...the model checking algorithm checks whether all the executions of the model satisfy the formula ($M \models \varphi$).

CTL Model Checking: General Idea

Two macro-steps:

1. construct the set of states where the formula holds:
   $[\varphi] := \{s \in S : M, s \models \varphi\}$
   ($[\varphi]$ is called the denotation of φ)
2. then compare with the set of initial states:
   $I \subseteq [\varphi]$?

CTL Model Checking: General Idea [cont.]

To compute $[\varphi]$:

- ▷ proceed "bottom-up" on the structure of the formula, computing $[\varphi_i]$ for each subformula φ_i of $\mathbf{AG}(p \rightarrow \mathbf{AF}q)$:
  - $[q]$,
  - $[\mathbf{AF}q]$,
  - $[p]$,
  - $[p \rightarrow \mathbf{AF}q]$,
  - $[\mathbf{AG}(p \rightarrow \mathbf{AF}q)]$

CTL Model Checking: General Idea [cont.]

▷ To compute each $[\varphi_i]$:

- handle boolean operators by standard set operations
- handle temporal operators $\mathbf{AX}$, $\mathbf{EX}$ by computing pre-images
- handle temporal operators $\mathbf{AG}$, $\mathbf{EG}$, $\mathbf{AF}$, $\mathbf{EF}$, $\mathbf{AU}$, $\mathbf{EU}$, by applying tableaux rules:

$$\mathbf{EG}\varphi_1 \iff (\varphi_1 \wedge \mathbf{EXEG}\varphi_1)$$
$$\mathbf{E}(\varphi_1 \mathbf{U} \varphi_2) \iff (\varphi_2 \vee (\varphi_1 \wedge \mathbf{EXE}(\varphi_1 \mathbf{U} \varphi_2)))$$

until a fixpoint is reached

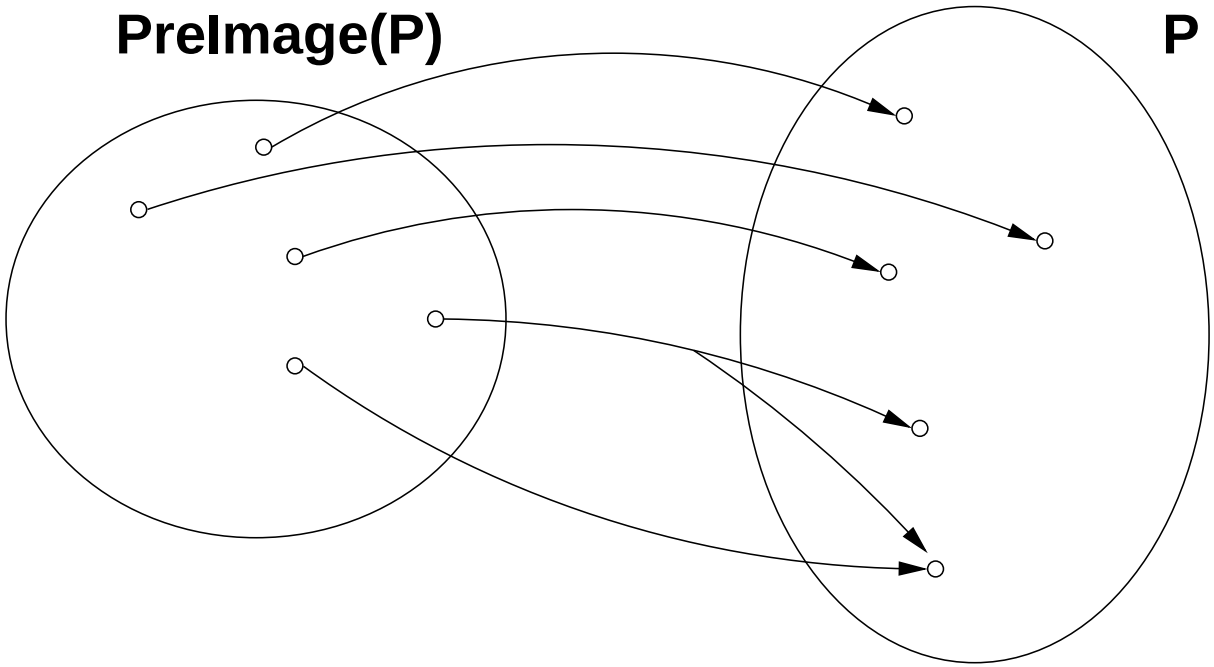
Denotation of a CTL formula φ: $[\varphi]$

$$[\varphi] := \{s \in S : M, s \models \varphi\}$$

$$\begin{aligned}
[false] &= \{\} \\
[true] &= S \\
[p] &= \{s | p \in L(s) \\
[\neg\varphi_1] &= S/[\varphi_1] \\
[\varphi_1 \wedge \varphi_2] &= [\varphi_1] \cap [\varphi_2] \\
[\mathbf{EX}\varphi] &= \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\} \\
[\mathbf{EG}\beta] &= \nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ ) \\
[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] &= \mu Z.(\ [\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z])\ )
\end{aligned}$$

Case EX

- $[\mathbf{EX}\varphi] = \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}$
- $[\mathbf{EX}\varphi]$ is said to be the Pre-image of $[\varphi]$ ($Preimage([\varphi])$)
- Key step of every CTL M.C. operation
- Note: Preimage() is monotonic: $X \subseteq X' \Longrightarrow Preimage(X) \subseteq Preimage(X')$



Case $\mathbf{EG}$

- $[\mathbf{EG}\beta] = \nu Z.(\ [\beta] \cap [\mathbf{EX}Z]\ )$
- Greatest fixed point $\nu x.F(x)$ of the function $F : 2^S \longmapsto 2^S$, s.t.

$$\begin{aligned} F([\varphi]) &= ([\beta] \cap Preimage([\varphi]) \\ &= ([\beta] \cap \{s \mid \exists s' \in [\varphi]\ s.t.\ \langle s, s' \rangle \in R\}) \end{aligned}$$

- F Monotonic: $a \subseteq a' \Longrightarrow F(a) \subseteq F(a')$
  - (Tarski's theorem): $\nu x.F(x)$ always exists
  - (Kleene's theorem): $\nu x.F(x)$ can be computed as the limit $S \supseteq F(S) \supseteq F(F(S)) \supseteq \ldots$, in a finite number of steps.

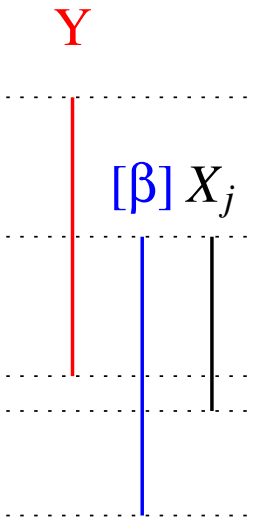
Case $\mathbf{EG}$ [cont.]

- We can compute $X := [\mathbf{EG}\beta]$ inductively as follows:

$$\begin{aligned} X_0 &:= S \\ X_1 &:= F(S) = [\beta] \\ X_2 &:= F(F(S)) = [\beta] \cap Preimage(X_1) \\ &\dots \\ X_{j+1} &:= F^{j+1}(S) = [\beta] \cap Preimage(X_j) \end{aligned}$$

- Noticing that $X_1 = [\beta]$ and $X_{j+1} \subseteq X_j$ for every $j \geq 0$, and that $([\beta] \cap Y) \subseteq X_j \subseteq [\beta] \Longrightarrow ([\beta] \cap Y) = (X_j \cap Y)$,
  we can use instead the following inductive schema:
  - $X_1 := [\beta]$
  - $X_{j+1} := X_j \cap Preimage(X_j)$

  until a fixpoint is reached.



Case $\mathbf{EU}$

- $[\mathbf{E}(\beta_1 \mathbf{U} \beta_2)] = \mu Z.([\beta_2] \cup ([\beta_1] \cap [\mathbf{EX}Z]))$
- Least fixed point $\mu x.F(x)$ of the function $F : 2^S \longmapsto 2^S$, s.t.

$$\begin{aligned} F([\varphi]) &= [\beta_2] \cup ([\beta_1] \cap Preimage([\varphi])) \\ &= [\beta_2] \cup ([\beta_1] \cap \{s \mid \exists s' \in [\varphi] \; s.t. \; \langle s, s' \rangle \in R\}) \end{aligned}$$

- F Monotonic: $a \subseteq a' \Longrightarrow F(a) \subseteq F(a')$
  - (Tarski's theorem): $\mu x.F(x)$ always exists
  - (Kleene's theorem): $\mu x.F(x)$ can be computed as the limit $\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \subseteq \ldots$, in a finite number of steps.

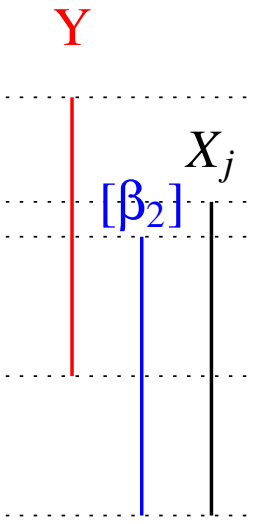
Case $\mathbf{EU}$ [cont.]

- We can compute $X := [\mathbf{E}(\beta_1 \mathbf{U} \beta_2)]$ inductively as follows:

$$
\begin{array}{lcll}
X_0 & := & \emptyset & \\
X_1 & := & F(\emptyset) & = [\beta_2] \\
X_2 & := & F(F(\emptyset)) & = [\beta_2] \cup ([\beta_1] \cap Preimage(X_1)) \\
\ldots & & & \\
X_{j+1} & := & F^{j+1}(\emptyset)) & = [\beta_2] \cup ([\beta_1] \cap Preimage(X_j))
\end{array}
$$

- Noticing that $X_1 = [\beta_2]$ and $X_{j+1} \supseteq X_j$ for every $j \geq 0$, and that $([\beta_2] \cup Y) \supseteq X_j \supseteq [\beta_2] \Longrightarrow ([\beta_2] \cup Y) = (X_j \cup Y)$,
  we can use instead the following inductive schema:
  - $X_1 := [\beta_2]$
  - $X_{j+1} := X_j \cup ([\beta_1] \cap Preimage(X_j))$

  until a fixpoint is reached.

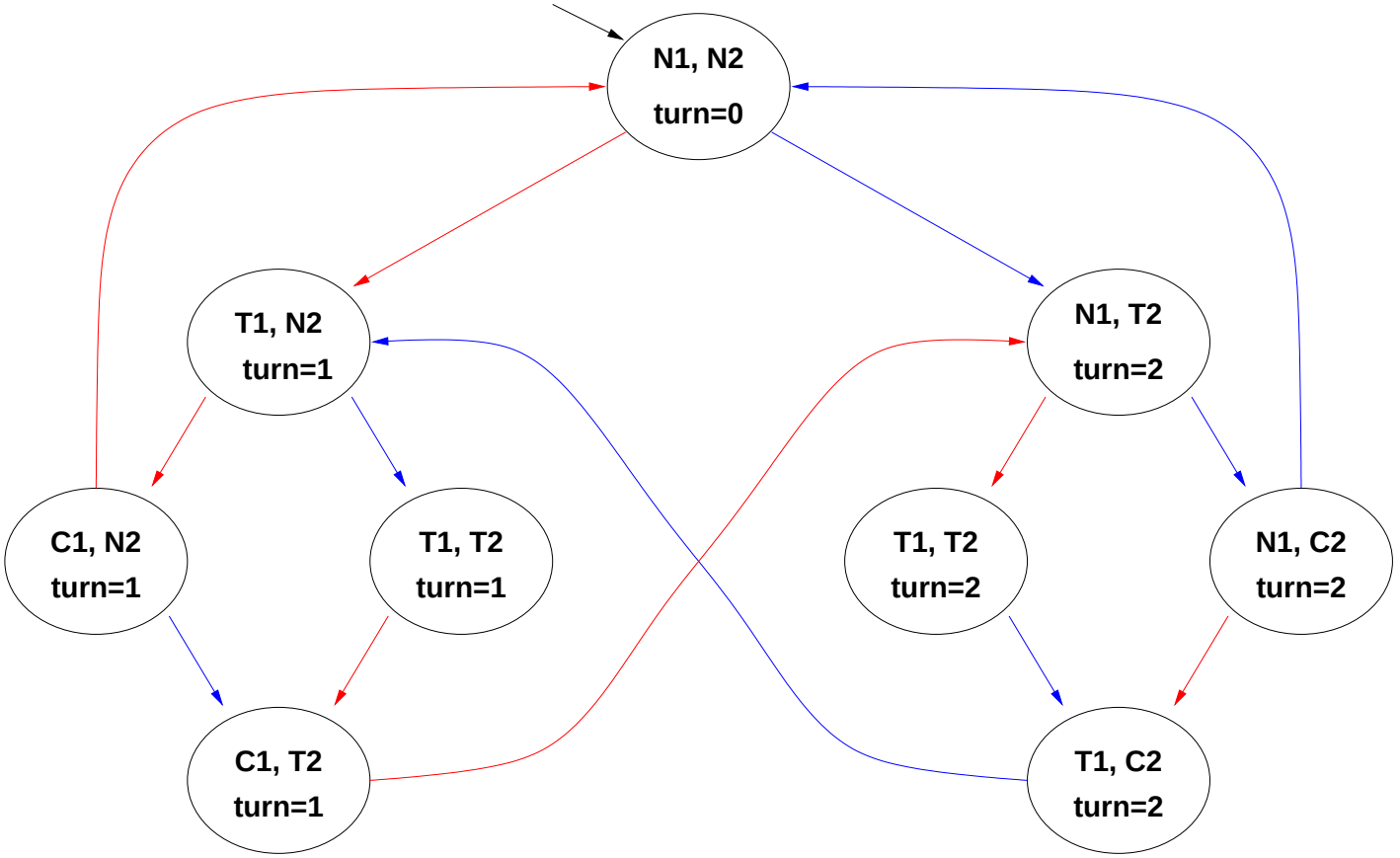


A relevant subcase: $\mathbf{EF}$

- $\mathbf{EF}\beta = \mathbf{E}(\top \mathbf{U} \beta)$
- $[\top] = S \Longrightarrow [\top] \cap Preimage(X_j) = Preimage(X_j)$
- We can compute $X := [\mathbf{EF}\beta]$ inductively as follows:
  - $X_1 := [\beta]$
  - $X_{j+1} := X_j \cup Preimage(X_j)$

  until a fixpoint is reached.

Example 1: fairness

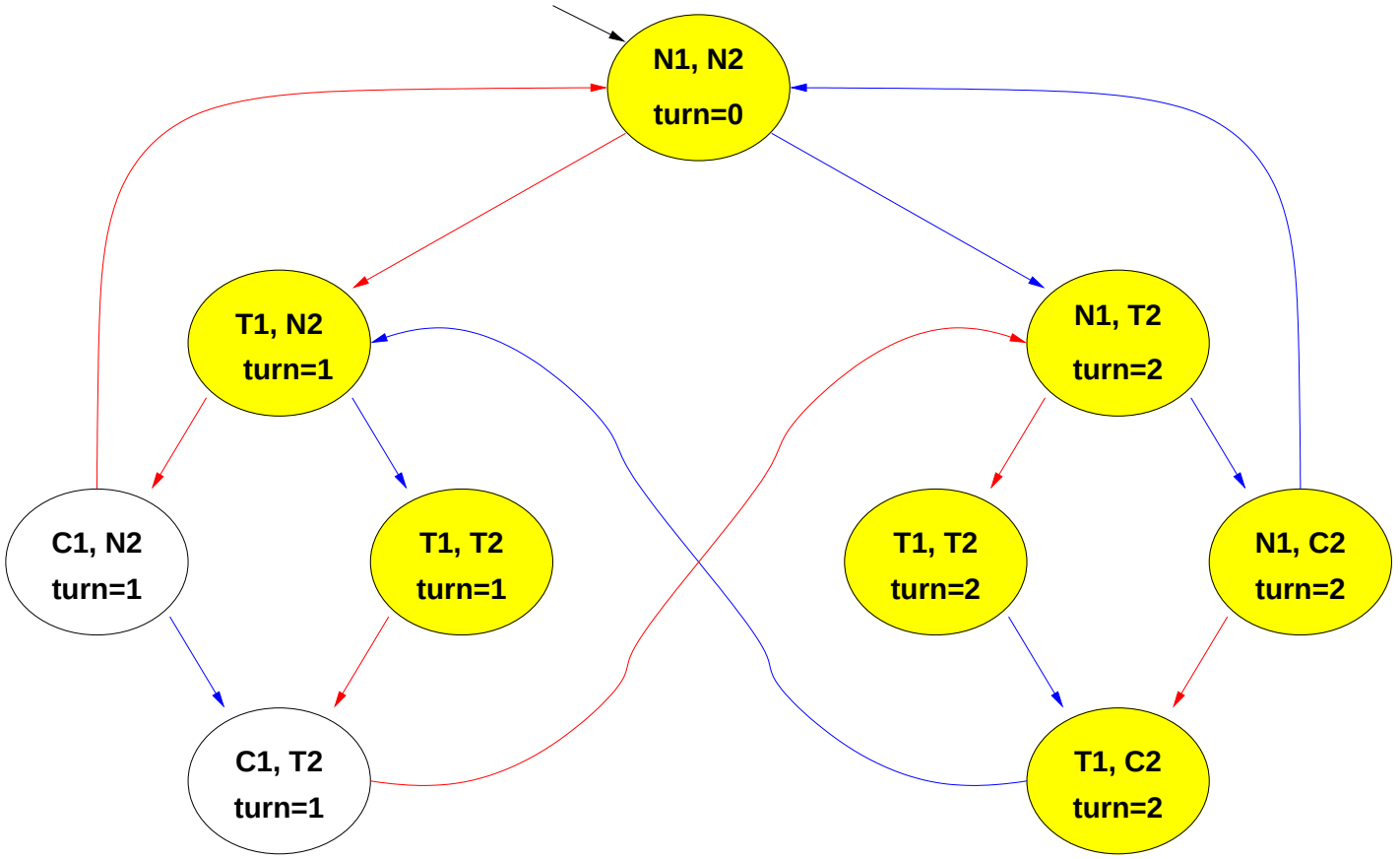


N = noncritical, T = trying, C = critical User 1 User 2

$$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$$

Example 1: fairness

$[\neg C_1]$

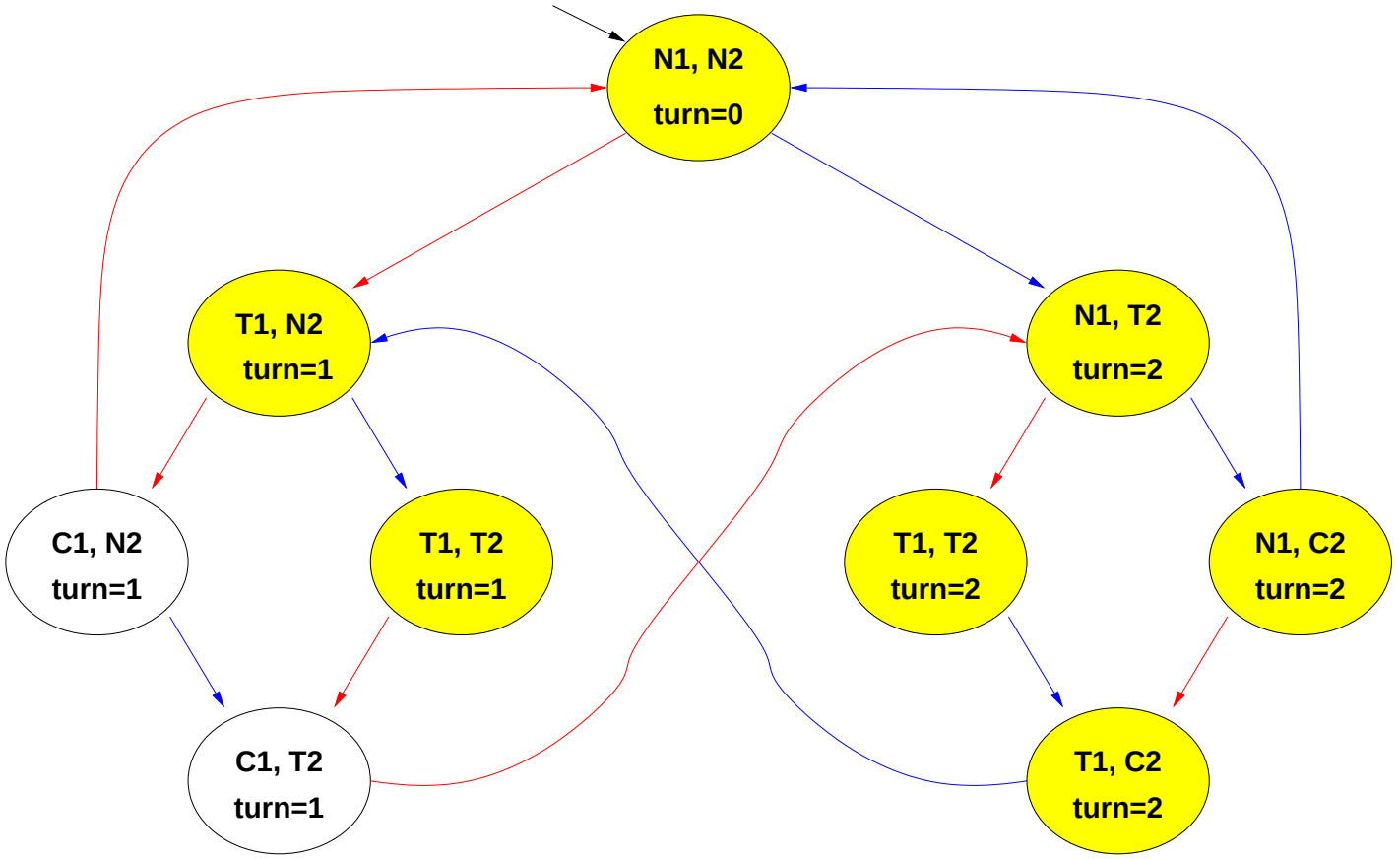


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$

Example 1: fairness

$[\mathbf{EG}\neg C_1]$, step 0:

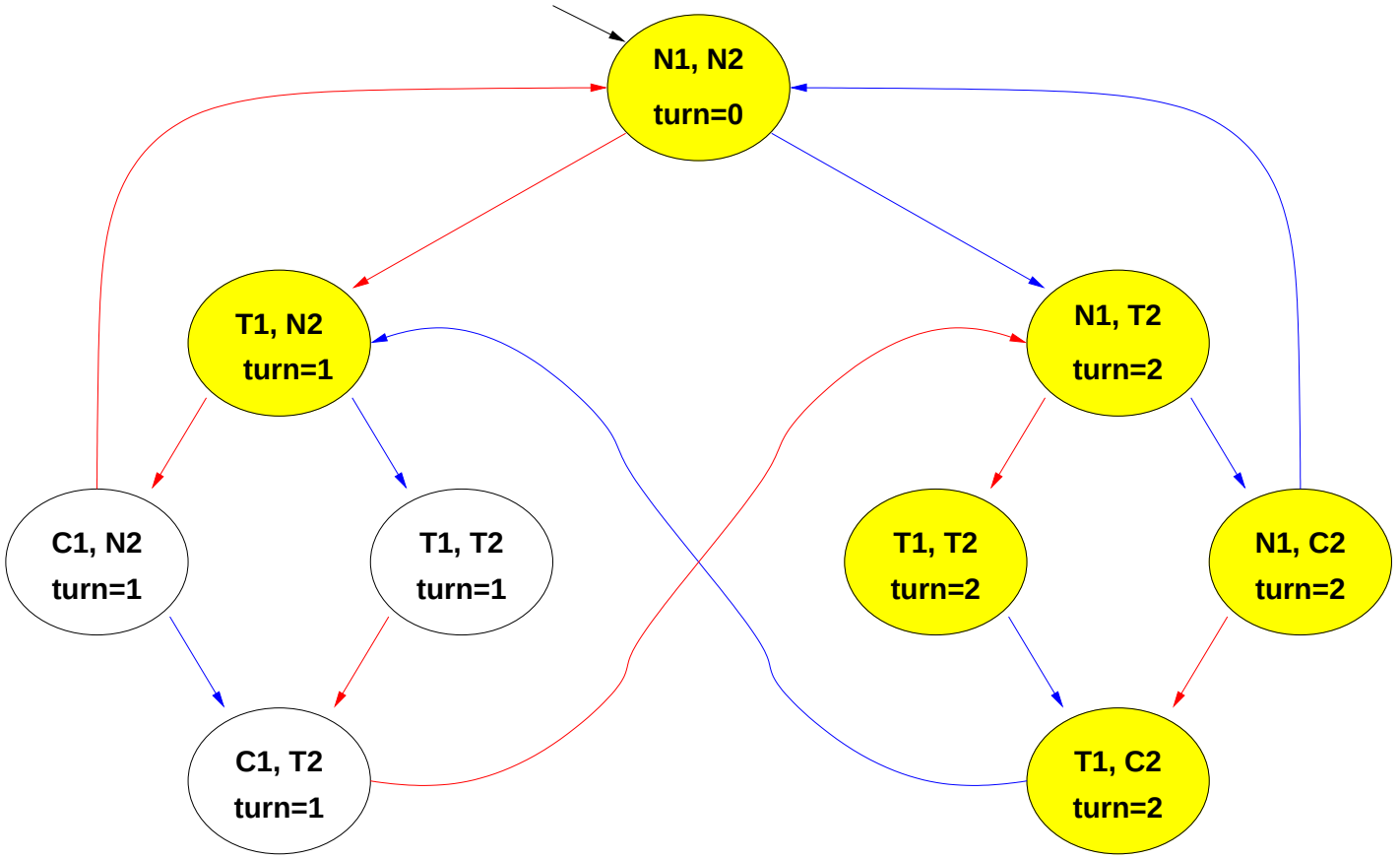


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$

Example 1: fairness

$[\mathbf{EG}\neg C_1]$, step 1:

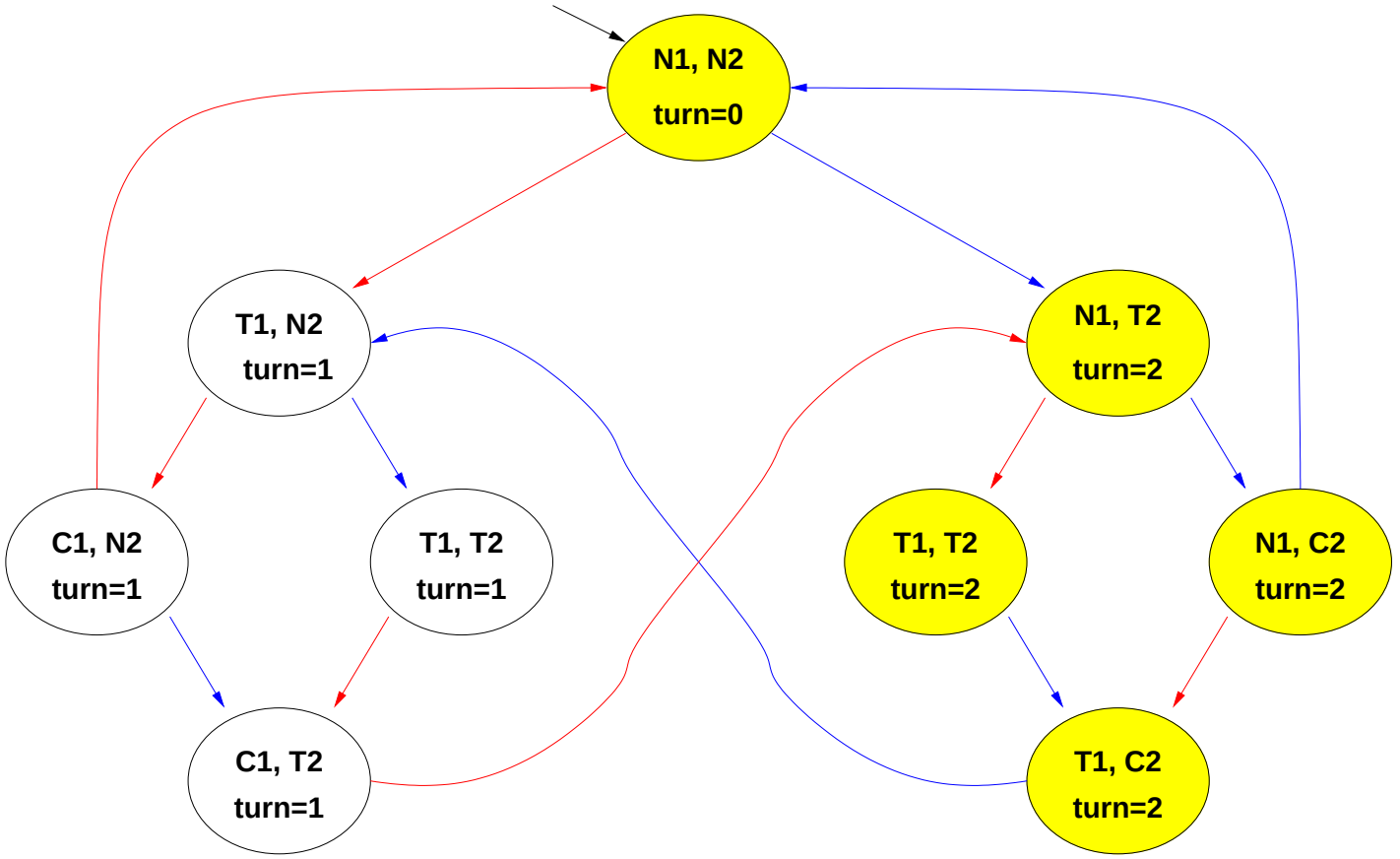


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \; ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \; ?$

Example 1: fairness

$[\mathbf{EG}\neg C_1]$, step 2:

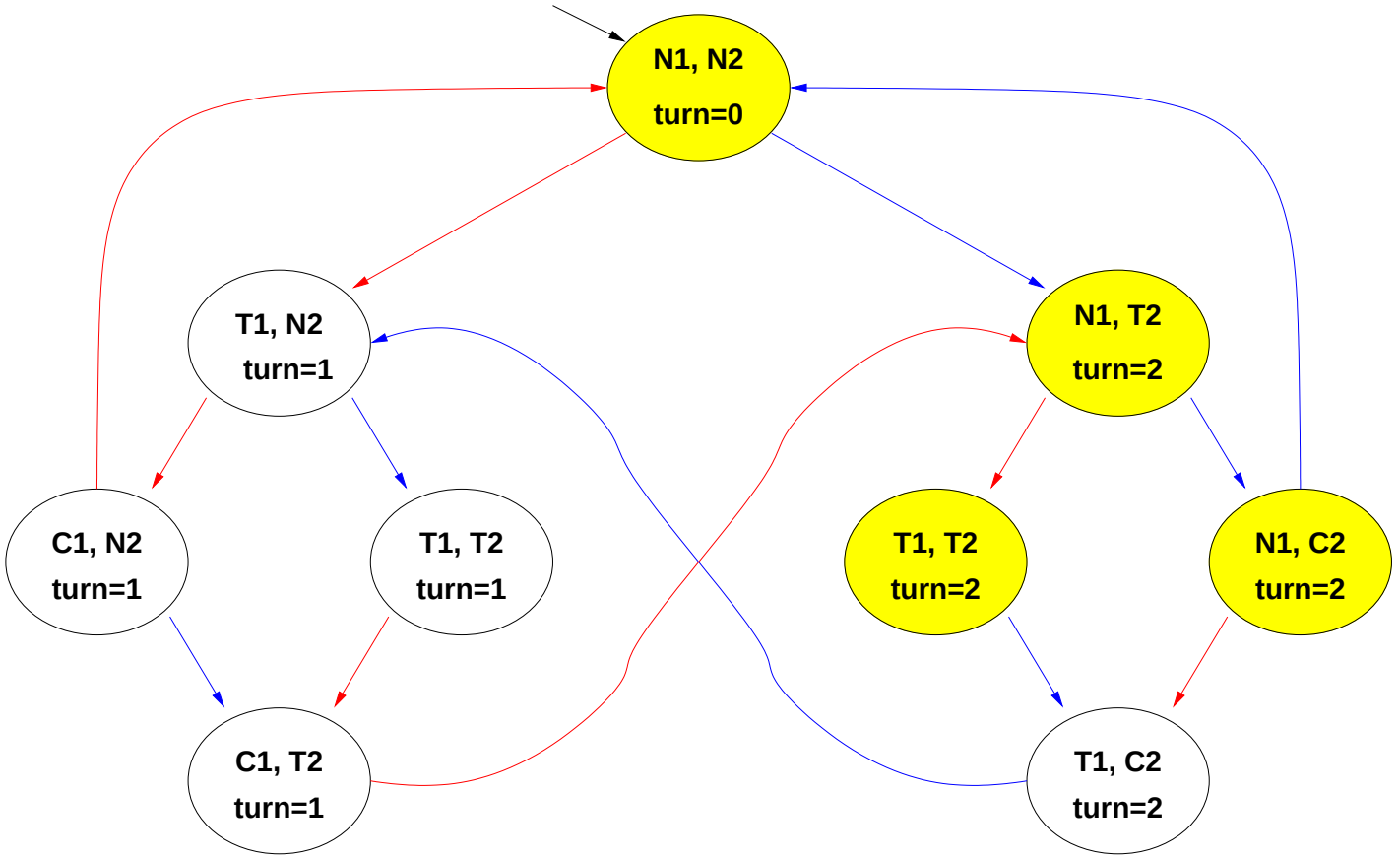


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \;?\Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \;?$

Example 1: fairness

$[\mathbf{EG}\neg C_1]$, step 3:

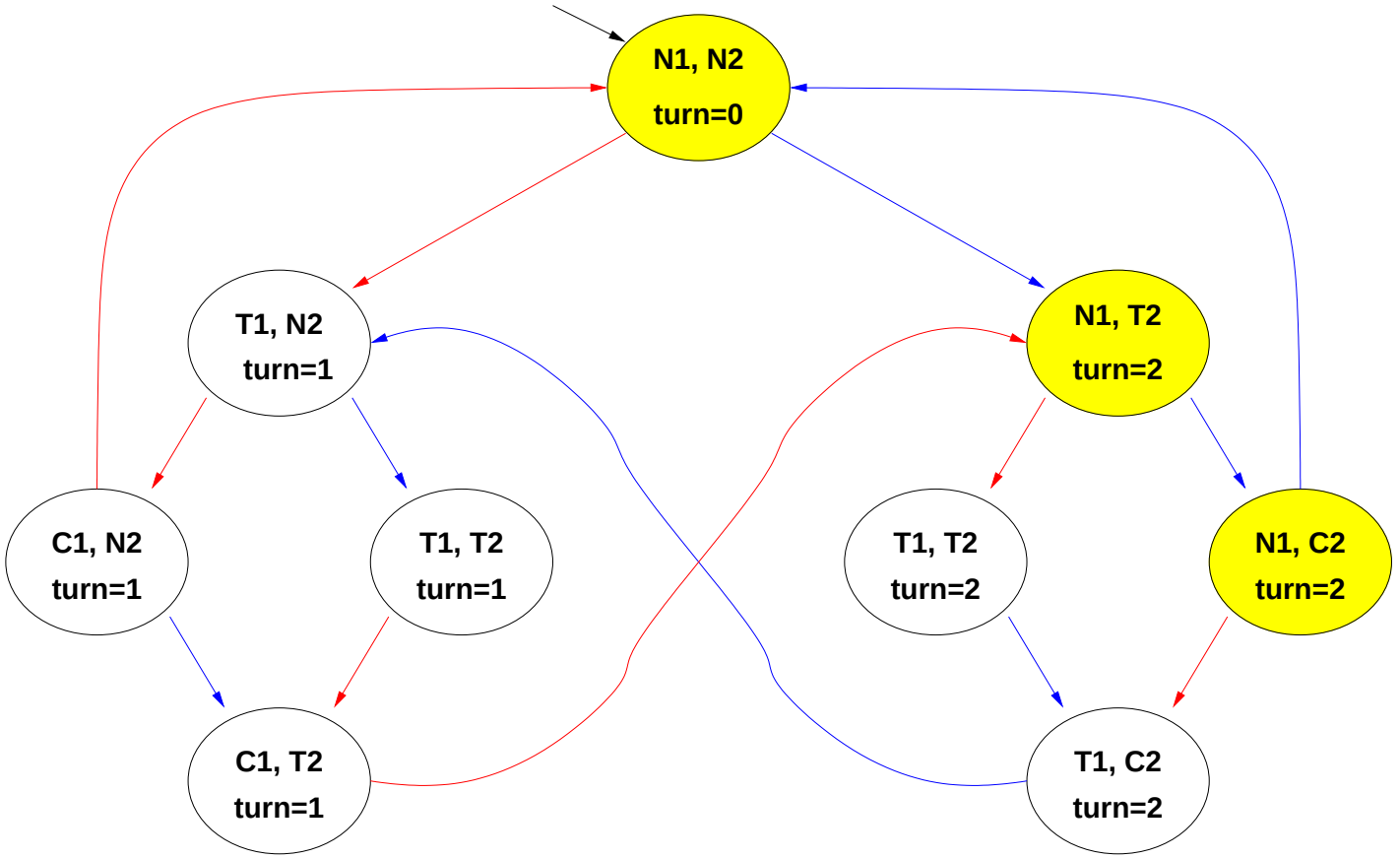


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \; ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \; ?$

Example 1: fairness

$[\mathbf{EG}\neg C_1]$, step 4:

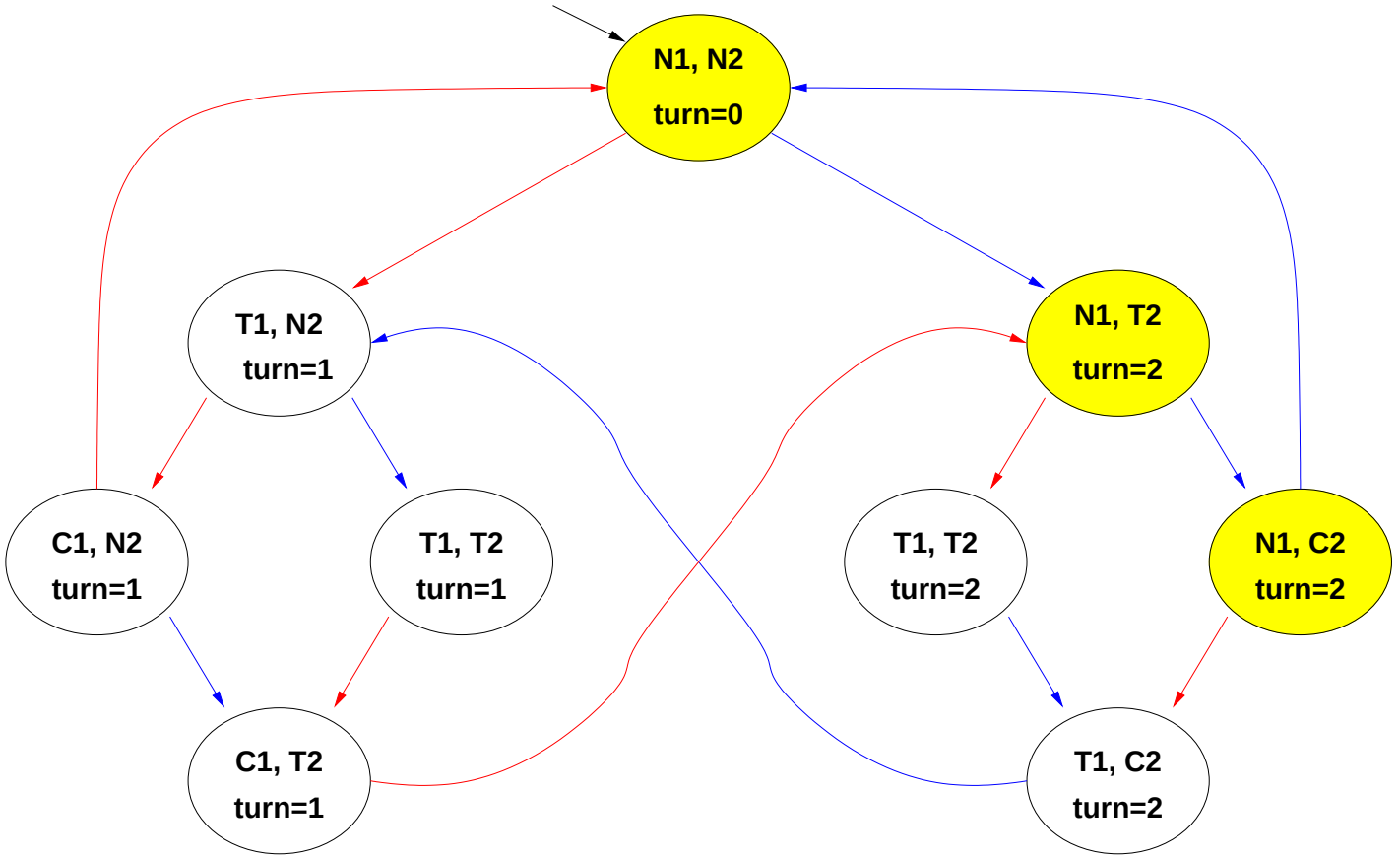


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \;?\Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \;?$

Example 1: fairness

$[\mathbf{EG}\neg C_1]$, FIXPOINT!

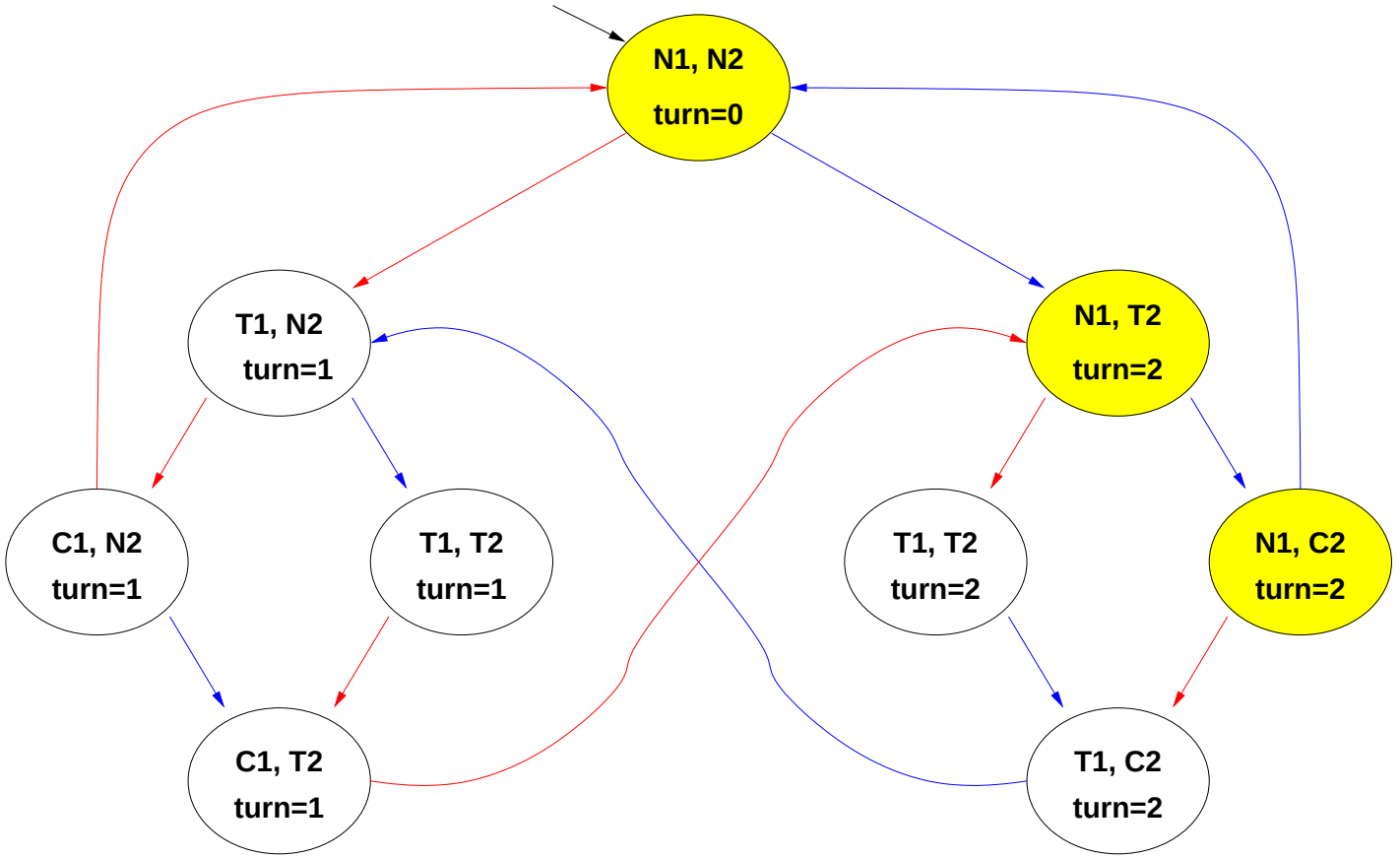


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$

Example 1: fairness

$[\mathbf{EFEG}\neg C_1]$, STEP 0

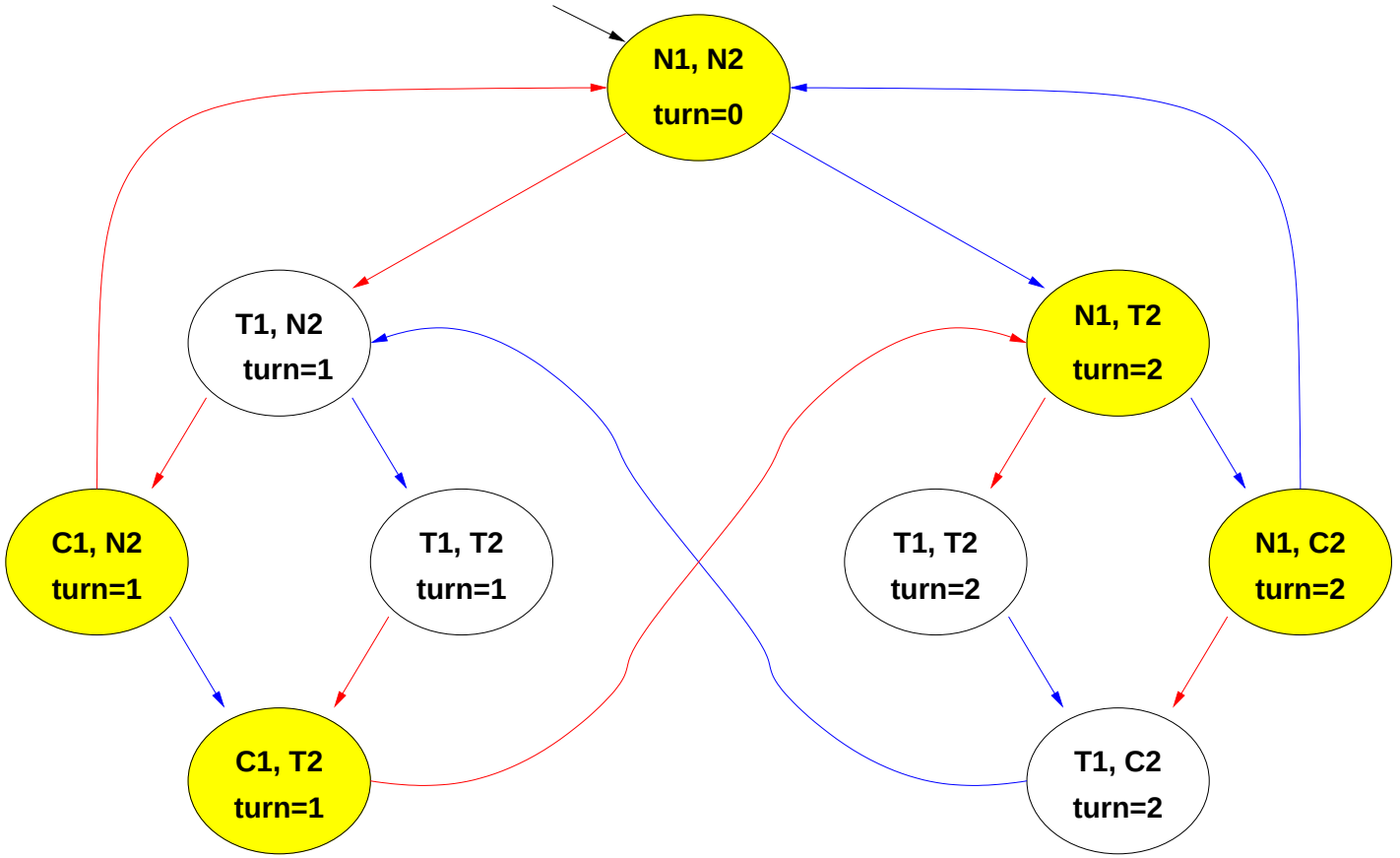


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$

Example 1: fairness

$[\mathbf{EFEG}\neg C_1]$, STEP 1

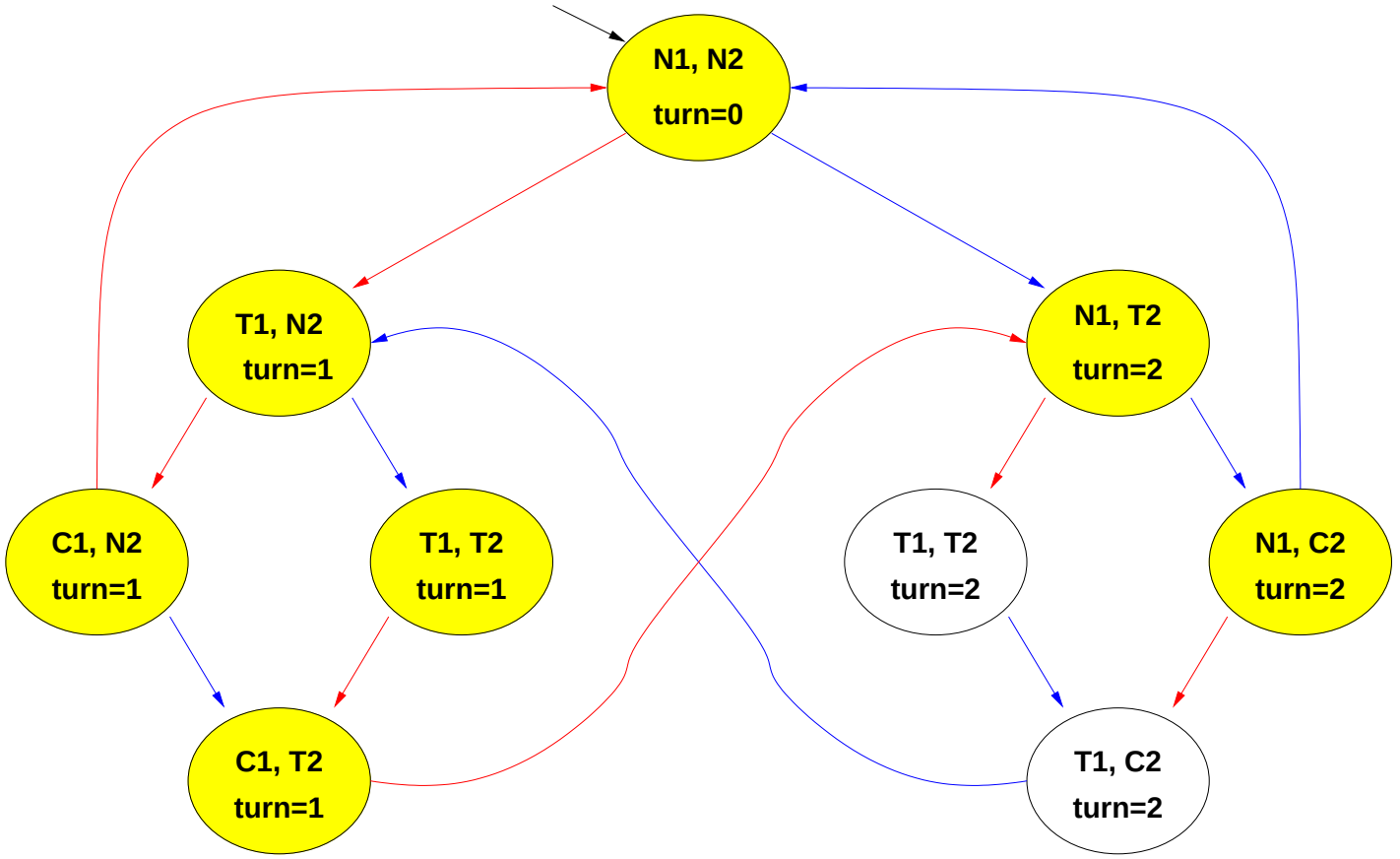


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$

Example 1: fairness

$[\mathbf{EFEG}\neg C_1]$, STEP 2

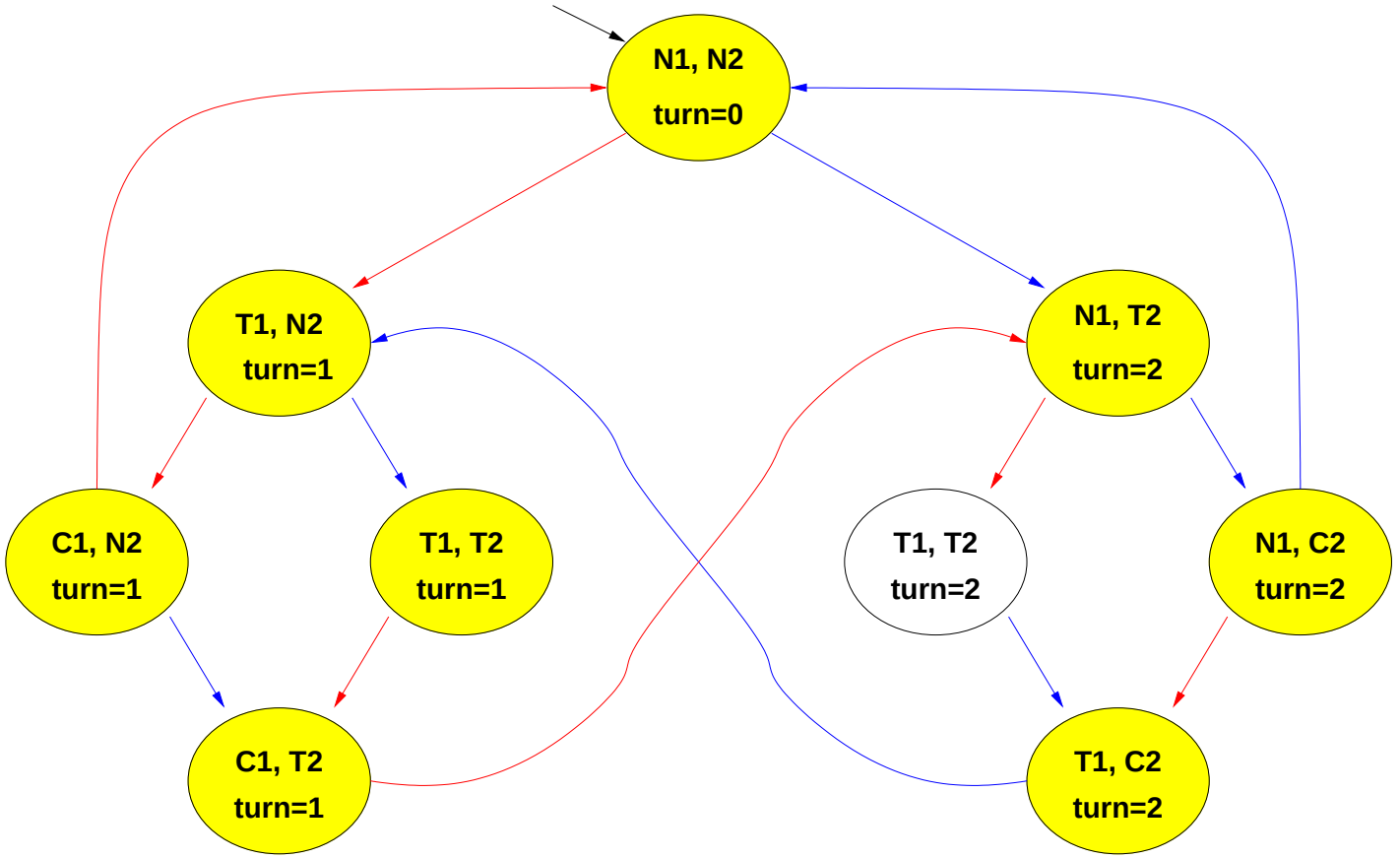


N = noncritical, T = trying, C = critical User 1 User 2

$$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$$

Example 1: fairness

$[\mathbf{EFEG}\neg C_1]$, STEP 3

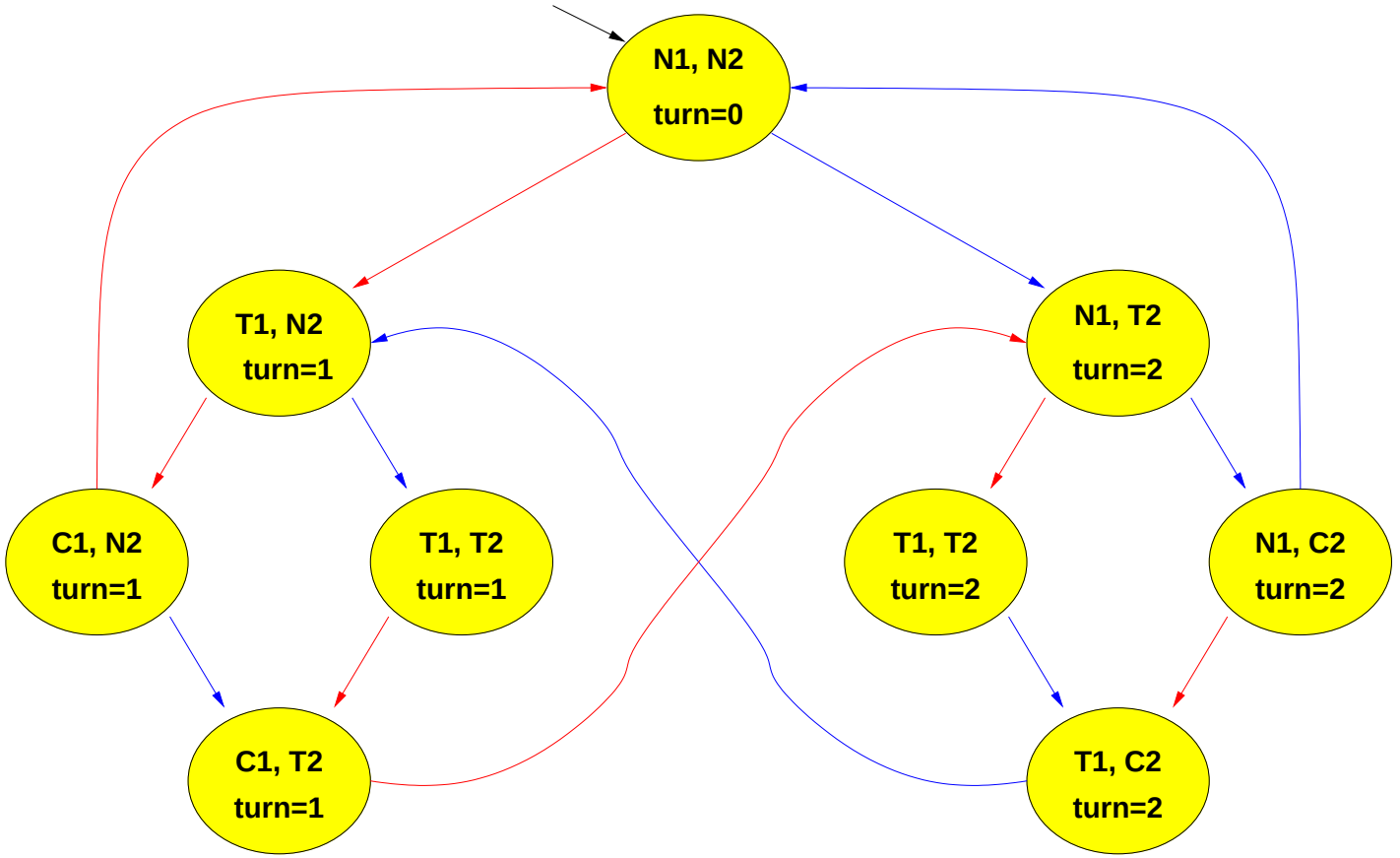


N = noncritical, T = trying, C = critical User 1 User 2

$$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$$

Example 1: fairness

$[\mathbf{EFEG}\neg C_1]$, STEP 4

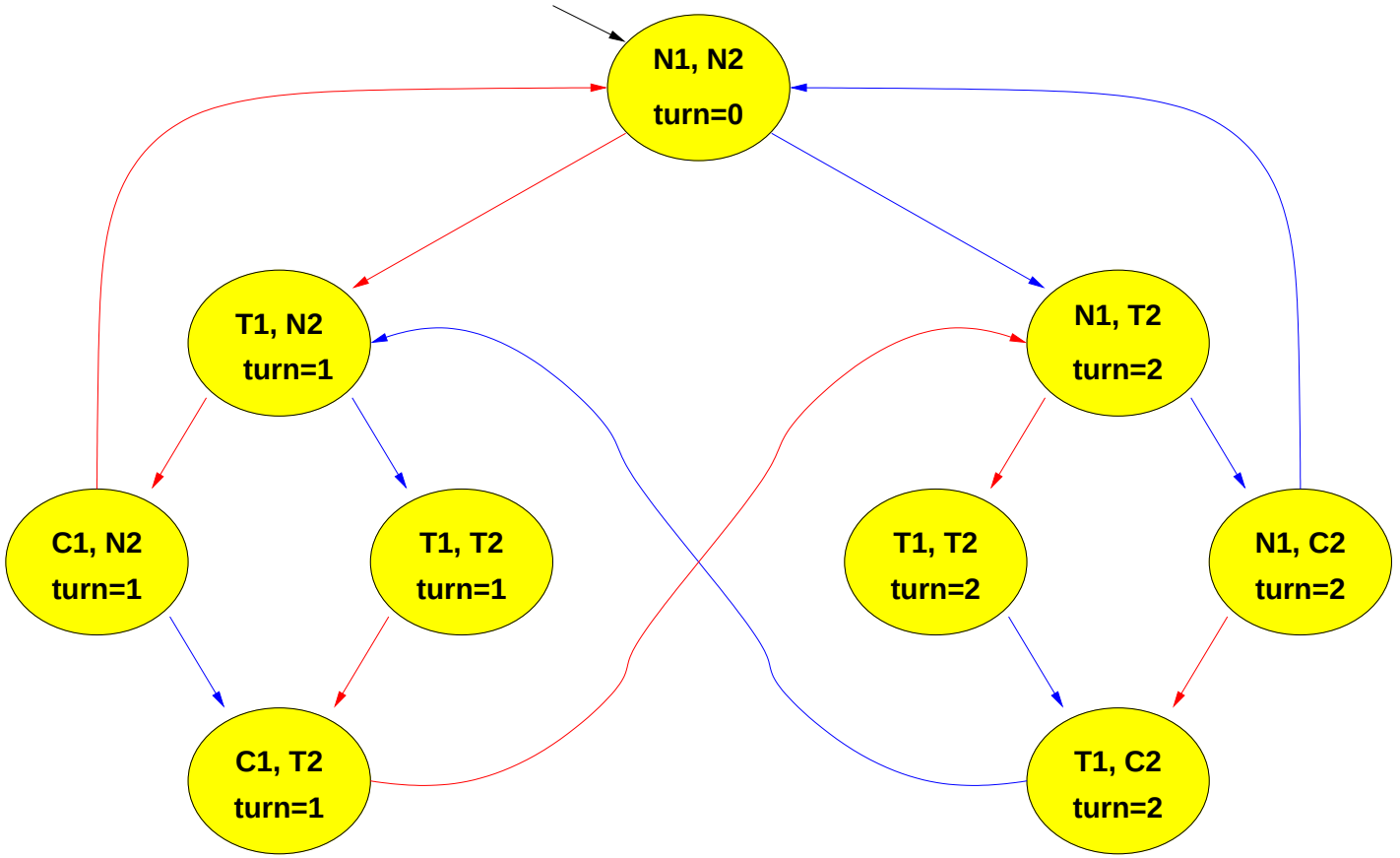


N = noncritical, T = trying, C = critical User 1 User 2

$$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ?$$

Example 1: fairness

$[\mathbf{EFEG}\neg C_1]$, FIXPOINT!

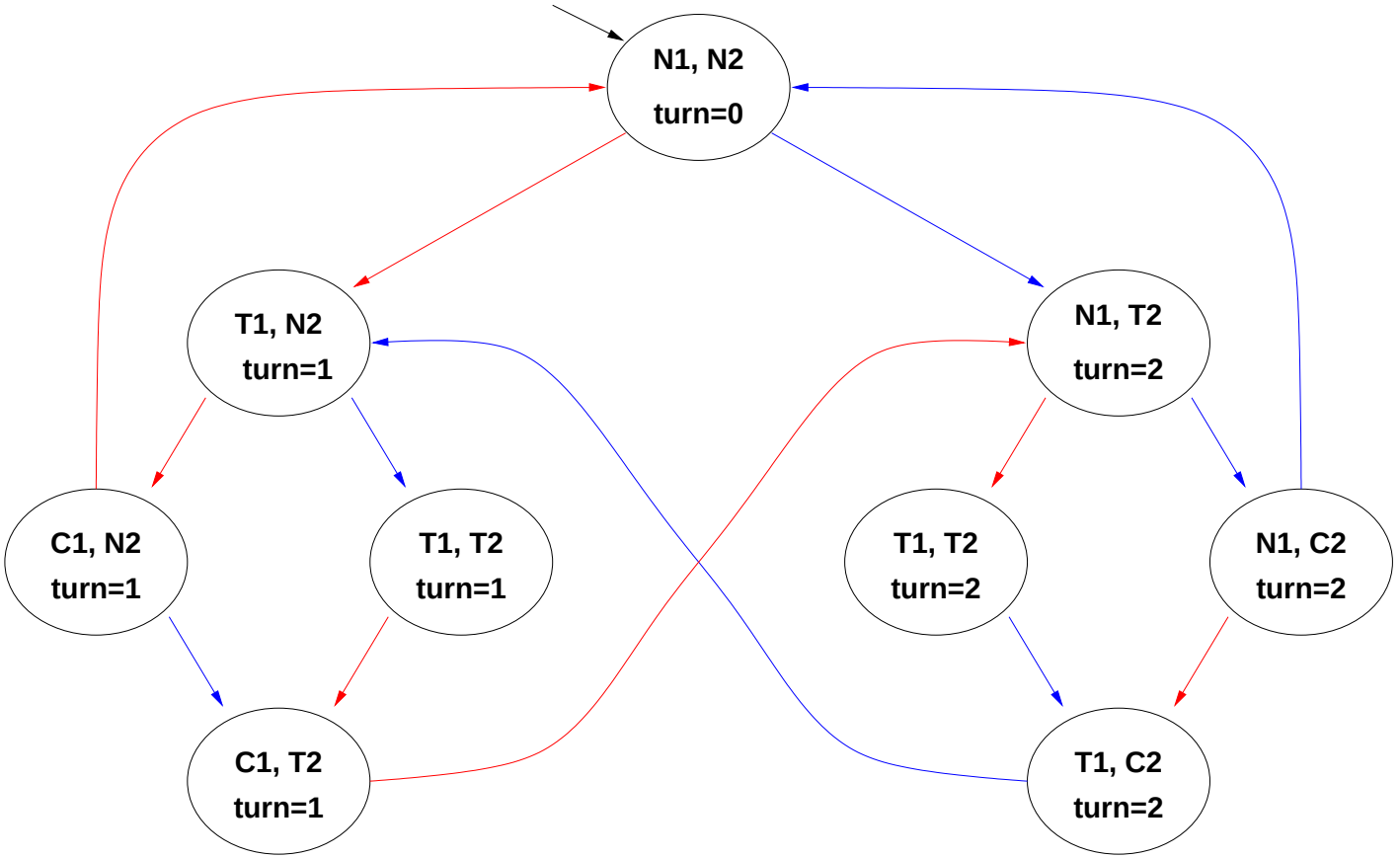


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AGAF}C_1 \; ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \; ?$

Example 1: fairness

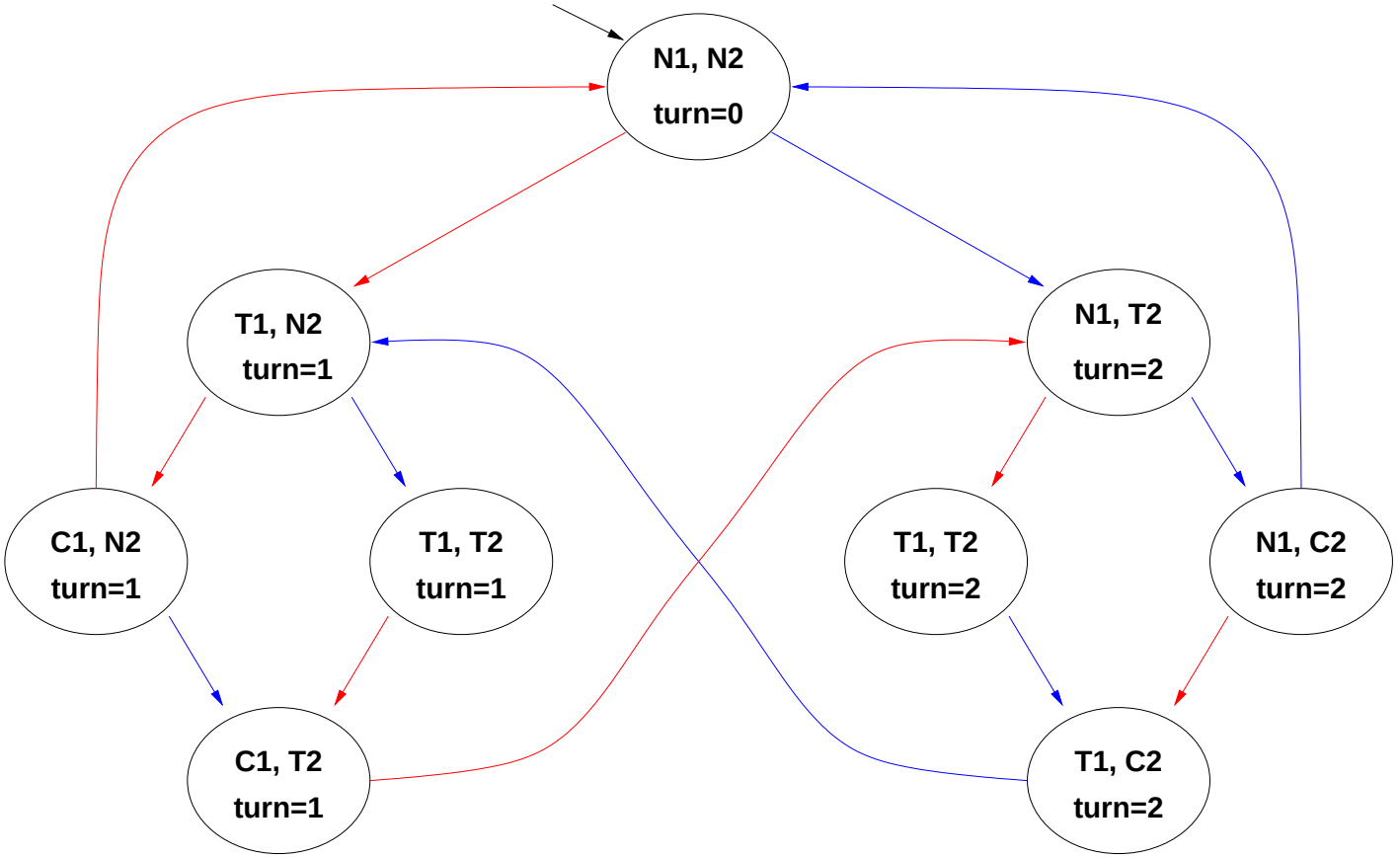
$[\neg\mathbf{EFEG}\neg C_1]$



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

$M \models \mathbf{AGAF}C_1 \ ? \Longrightarrow M \models \neg\mathbf{EFEG}\neg C_1 \ ? \Longrightarrow$ NO!

Example 2: liveness

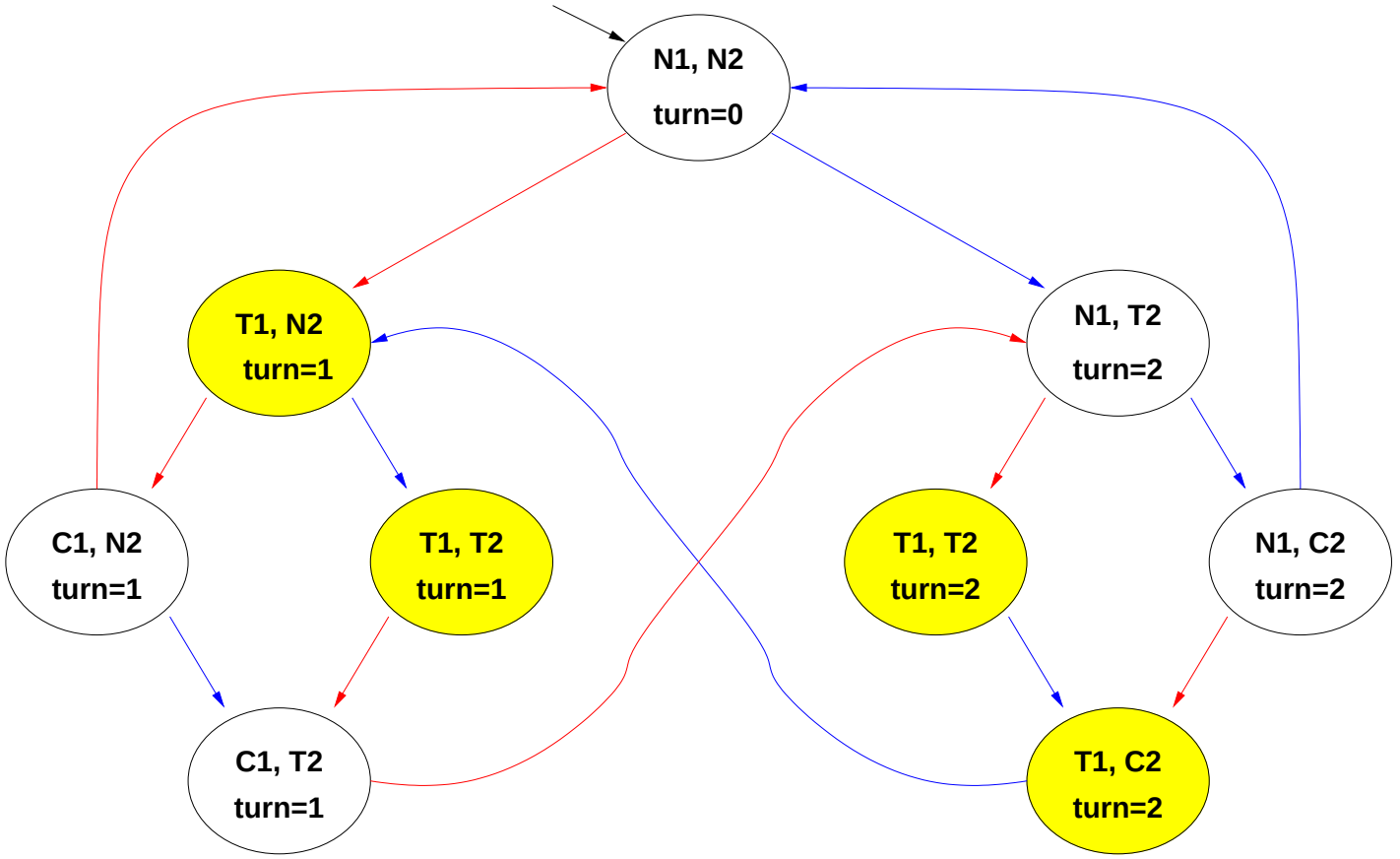


N = noncritical, T = trying, C = critical User 1 User 2

$$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)\ ? \Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)\ ?$$

Example 2: liveness

$[T_1]$:

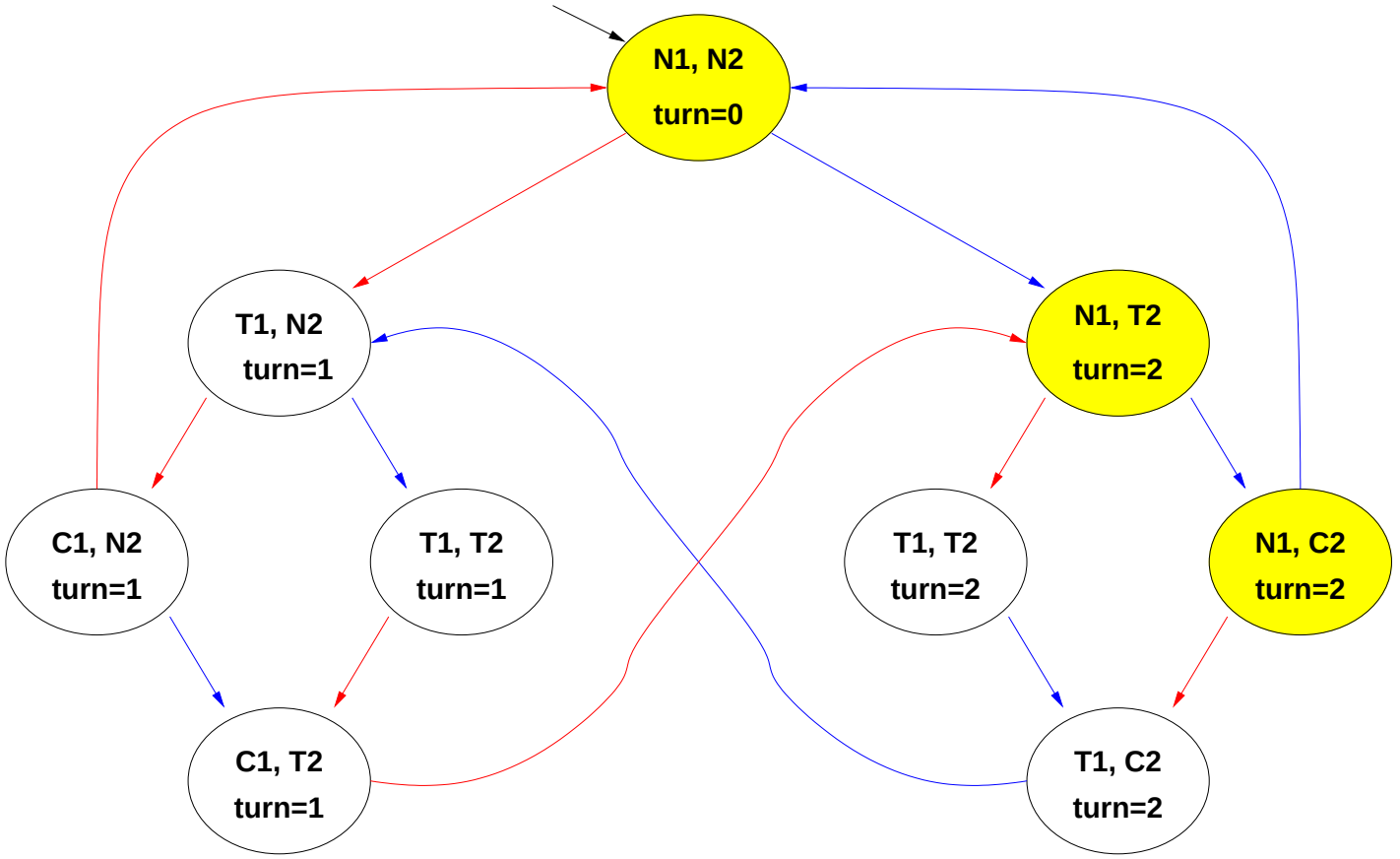


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)\ ? \Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)\ ?$

Example 2: liveness

$[\mathbf{EG}\neg C_1]$, STEPS 0-4: (see previous example)

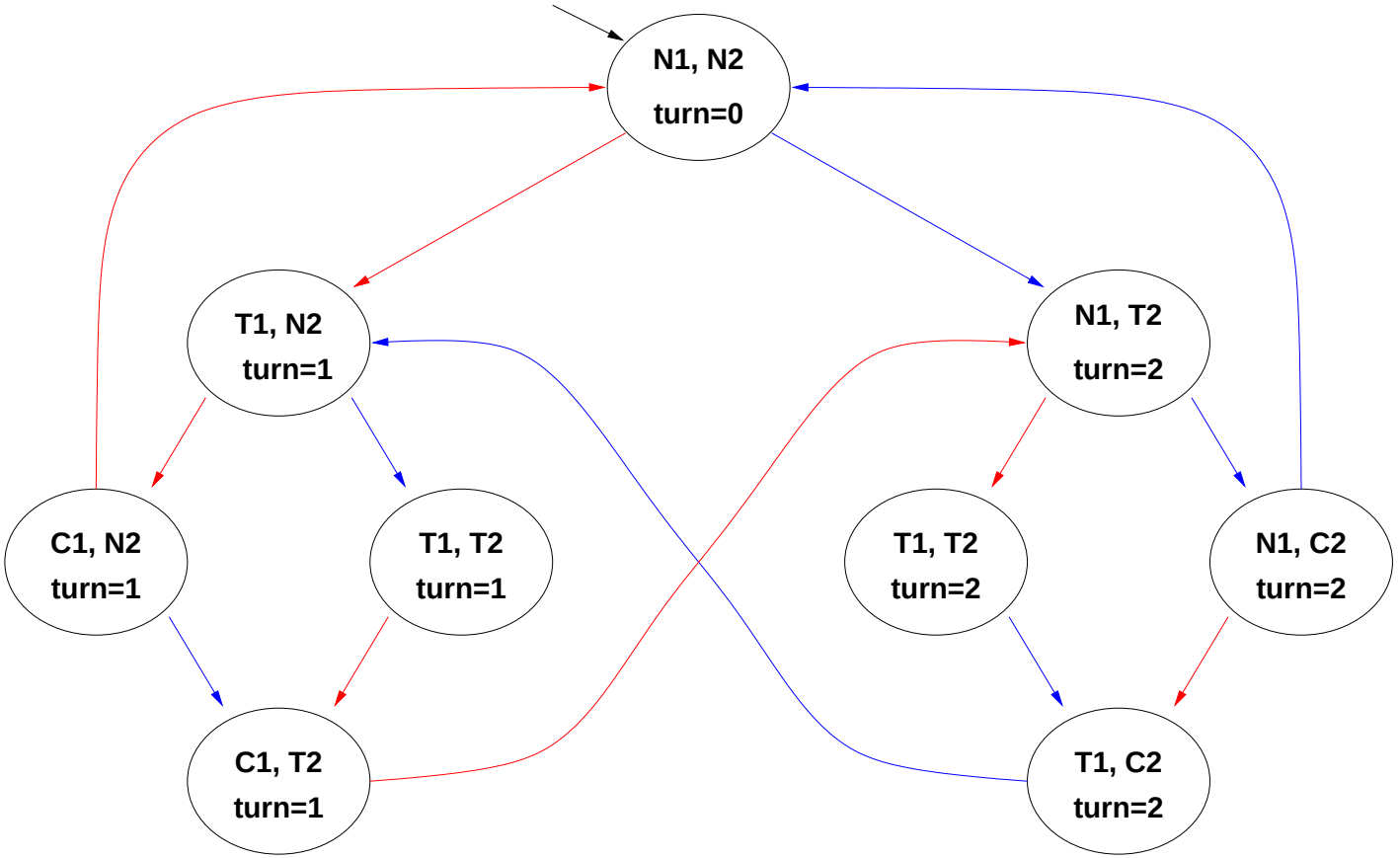


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)\ ? \Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)\ ?$

Example 2: liveness

$[T_1 \wedge \mathbf{EG}\neg C_1]$:

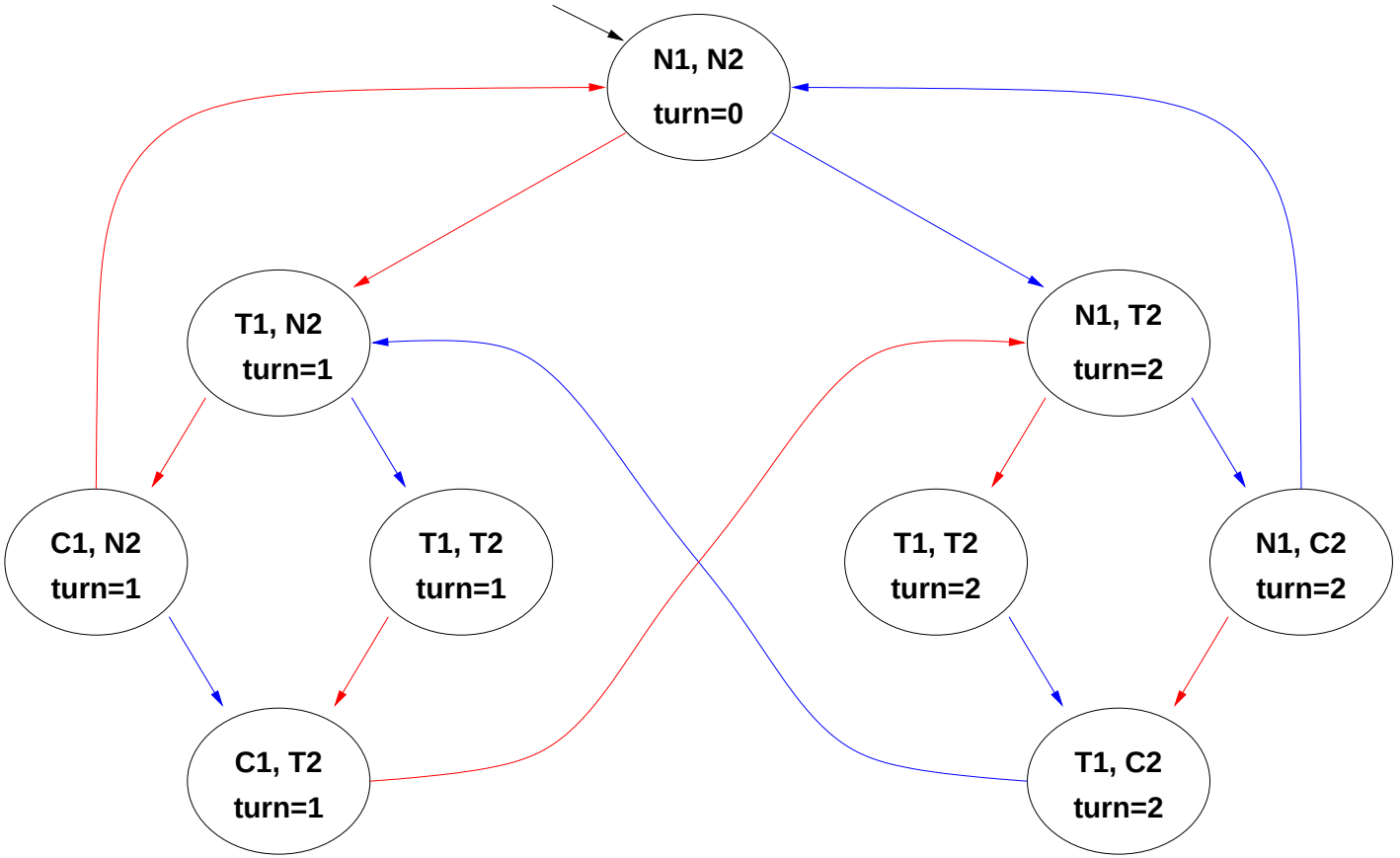


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)\ ? \Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)\ ?$

Example 2: liveness

$[\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)]$:

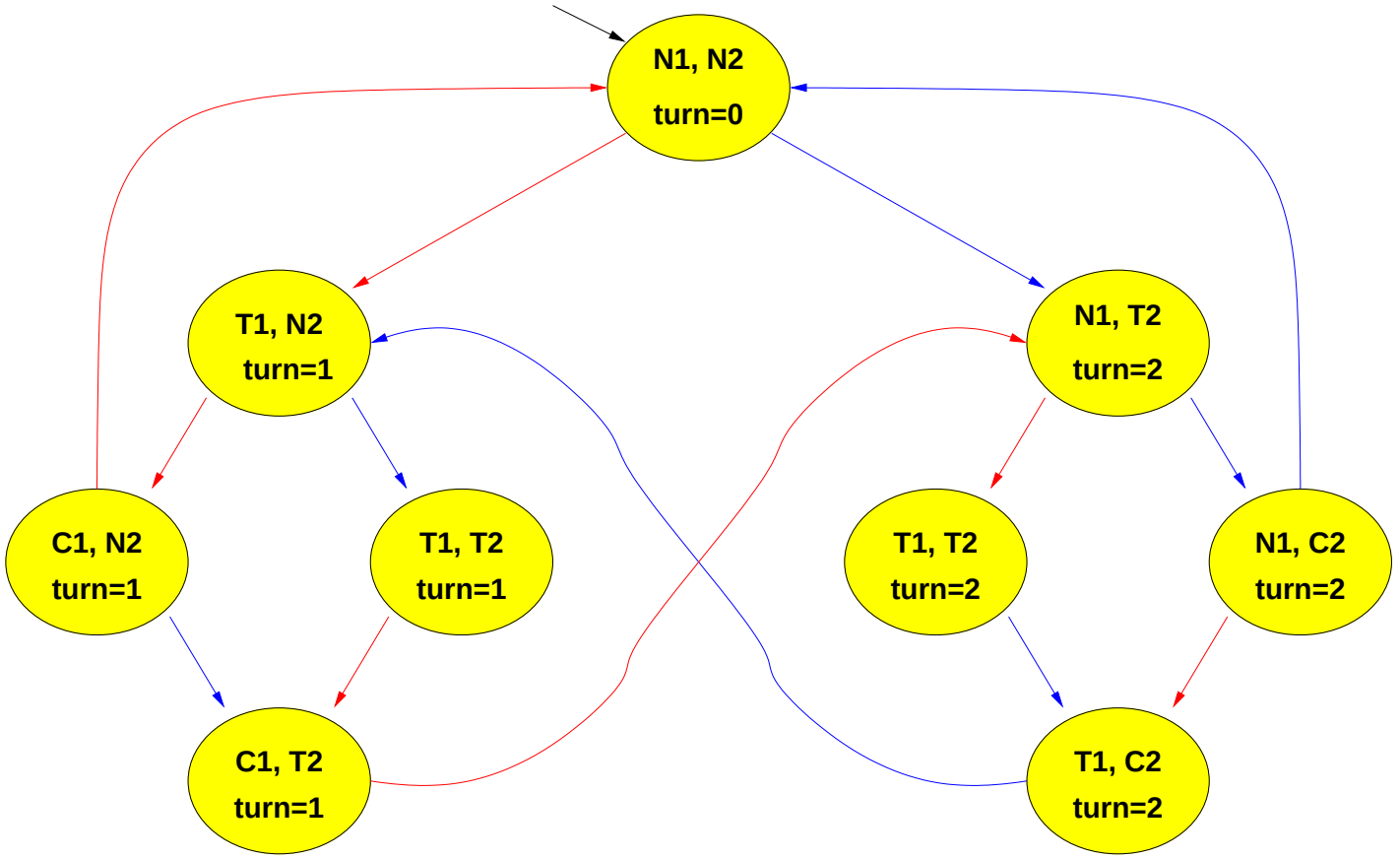


N = noncritical, T = trying, C = critical User 1 User 2

$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)\ ? \Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)\ ?$

Example 2: liveness

$[\neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)]$:



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

$M \models \mathbf{AG}(T_1 \rightarrow \mathbf{AF}C_1)$? $\Longrightarrow M \models \neg\mathbf{EF}(T_1 \wedge \mathbf{EG}\neg C_1)$? YES!

Content

The Main Problem of CTL M.C. State Space Explosion

- The bottleneck:
  - Exhaustive analysis may require to store all the states of the Kripke structure, and to explore them one-by-one
  - The state space may be exponential in the number of components and variables (E.g., 300 boolean vars $\Longrightarrow$ up to $2^{300} \approx 10^{100}$ states!)
  - State Space Explosion:
    - too much memory required
    - too much CPU time required to explore each state
- A solution: Symbolic Model Checking

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);

Symbolic Model Checking [cont.]

- two main symbolic techniques:
  - Ordered Binary Decision Diagrams (OBDDs)
  - Propositional Satisfiability Checkers (SAT solvers)
- Different model checking algorithms:
  - Fix-point Model Checking (for CTL)
  - Fix-point Model Checking for LTL (conversion to fair CTL MC)
  - Bounded Model Checking (for LTL)
  - Invariant Checking
  - ...

Symbolic Representation of Kripke Structures

- ▷ Symbolic representation:
  - sets of states as their characteristic function
  - provide logical representation and transformations of characteristic functions
- ▷ Example:
  - three state variables x_1, x_2, x_3:
    $\{$ 000, 001, 010, 011 $\}$ represented as "first bit false": $\neg x_1$
  - with five state variables x_1, x_2, x_3, x_4, x_5:
    $\{$ 00000, 00001, 00010, 00011, 00100, 00101, 00110, 00111,..., 01111 $\}$ still represented as "first bit false": $\neg x_1$

Kripke Structures in Propositional Logic

- Let $M = (S, I, R, L, AF)$ be a Kripke structure
- States $s \in S$ are described by means of an array V of boolean state variables.
- A state is an truth assignment to each atomic proposition in V.
  - 0100 is represented by the formula $(\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x_4)$
  - we call $\xi(s)$ the formula representing the state $s \in S$
    (Intuition: $\xi(s)$ holds iff the system is in the state s)
- A set of states $Q \subseteq S$ can be (naively) represented by the formula $\xi(Q)$

$$\bigvee_{s \in Q} \xi(s)$$

- Bijection between models of $\xi(Q)$ and states in Q

Remark

- any propositional formula is a (typically very compact) representation of the set of assignments satisfying it
- Any formula equivalent to $\xi(Q)$ is a representation of Q
  $\Longrightarrow$Typically Q is encoded by much smaller formulas than $\bigvee_{s\in Q}\xi(s)$!!!
- Example: $Q = \{$ 00000, 00001, 00010, 00011, 00100, 00101, 00110, 00111,. . . , 01111 $\}$ represented as "first bit false": $\neg x_1$

$$\begin{aligned}\bigvee_{s\in Q}\xi(s) \quad = \quad & (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5) \vee \\ & (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge x_5) \vee \\ & (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4 \wedge \neg x_5) \vee \\ & \ldots \\ & (\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5)\end{aligned} \Bigg\} 2^4 disjuncts$$

Symbolic Representation of Set Operators

- Set of all the states: $\xi(S) := \top$
- Empty set : $\xi(\emptyset) := \bot$
- Union represented by disjunction:
  $\xi(P \cup Q) := \xi(P) \vee \xi(Q)$
- Intersection represented by conjunction:
  $\xi(P \cap Q) := \xi(P) \wedge \xi(Q)$
- Complement represented by negation:
  $\xi(S/P) := \neg\xi(P)$
- Containment represented by implication:
  $\xi(P \subseteq Q) := \xi(P) \rightarrow \xi(Q)$
- Set equality represented by logical equivalence:
  $\xi(P = Q) := \xi(P) \leftrightarrow \xi(Q)$

Symbolic Representation of Transition Relations

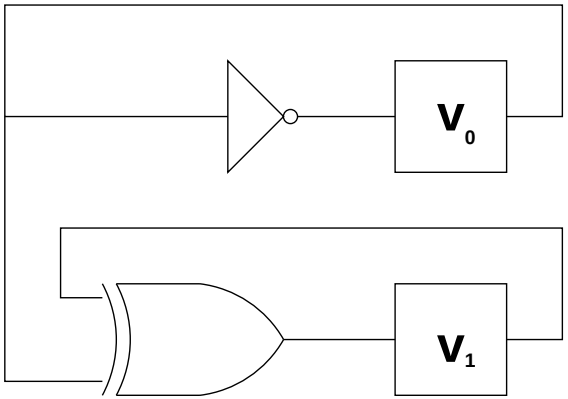
- The transition relation R is a set of pairs of states: $R \subseteq S \times S$
- A transition is a pair of states (s, s')
- A new vector of variables V' (the next state vector) represents the value of variables after the transition has occurred
- $\xi(s, s')$ defined as $\xi(s) \wedge \xi'(s')$
- The transition relation R can be (naively) represented by

$$\bigvee_{(s,s') \in R} \xi(s, s') = \bigvee_{(s,s') \in R} (\xi(s) \wedge \xi(s'))$$

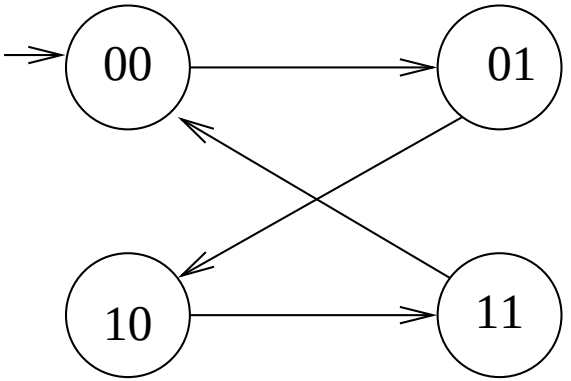
- Note: Each formula equivalent to $\xi(R)$ is a representation of R
  $\Longrightarrow$ Typically R is encoded by a much smaller formula than $\bigvee_{(s,s') \in R} (\xi(s) \wedge \xi(s'))$!!!

Example: a simple counter

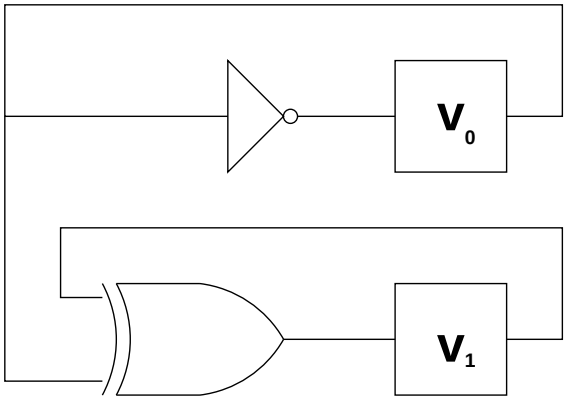
```
MODULE main
 VAR
   v0    : boolean;
   v1    : boolean;
   out   : 0..3;

 ASSIGN
   init(v0) := 0;
   next(v0) := !v0;

   init(v1) := 0;
   next(v1) := (v0 xor v1);

   out := v0 + 2*v1;
```



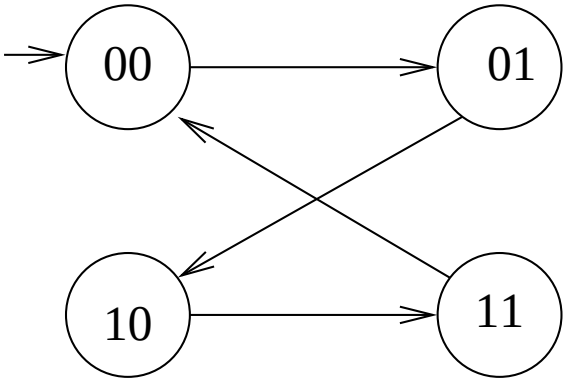
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



Example: a simple counter [cont.]



v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

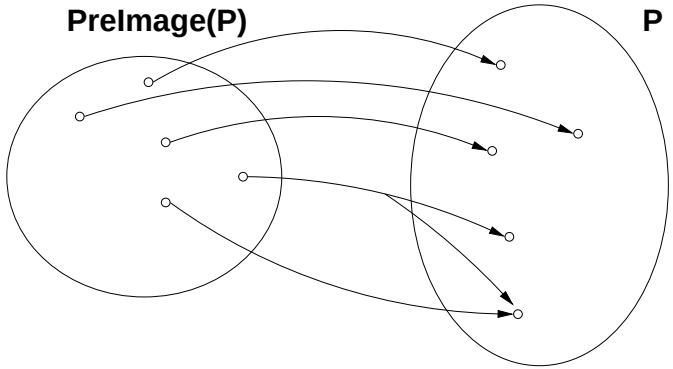


$$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$$

$$\bigvee_{(s,s')\in R} \xi(s) \wedge \xi(s') = (\neg v_1 \wedge \neg v_0 \wedge \neg v_1' \wedge v_0') \vee$$
$$(\neg v_1 \wedge v_0 \wedge v_1' \wedge \neg v_0') \vee$$
$$(v_1 \wedge \neg v_0 \wedge v_1' \wedge v_0') \vee$$
$$(v_1 \wedge v_0 \wedge \neg v_1' \wedge \neg v_0')$$

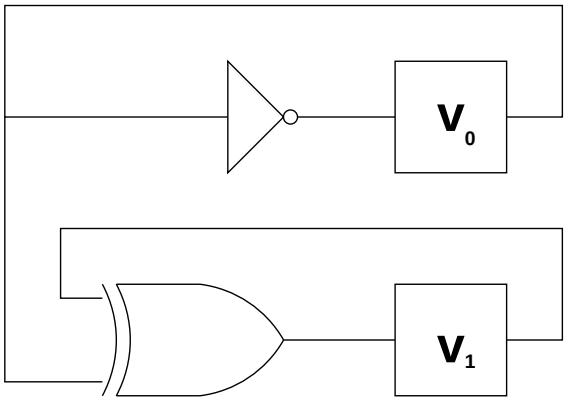
Preimmage

- (Backward) preimage of a set:

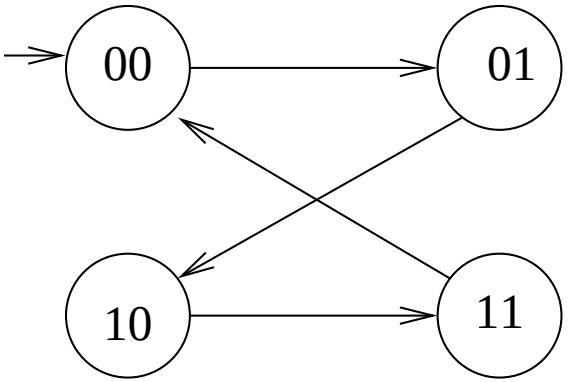


- Evaluate one-shot all transitions ending in the states of the set
- Set theoretic view: $Preimage(P,R) := \{s |$ for some $s' \in P, (s,s') \in R\}$
- Logical view: $\xi(Preimage(P,R)) := \exists V'.(\xi(P)[V'] \wedge \xi(R)[V,V'])$
  - $\exists v.\varphi \stackrel{\text{def}}{=} \varphi|_{v=0} \vee \varphi|_{v=1}$,
    $\exists v_1 \ldots v_k.\varphi \stackrel{\text{def}}{=} \varphi|_{v_1=0,\ldots,v_n=0} \vee \varphi|_{v_1=0,\ldots,v_n=1} \vee \ldots \vee \varphi|_{v_1=1,\ldots,v_n=1}$.
- μ over V is s.t $\mu \models \exists V'.(\xi(P)[V'] \wedge \xi(R)[V,V'])$ iff,
  for some μ' over V', we have: $\mu \cup \mu' \models (\xi(P)[V'] \wedge \xi(R)[V,V'])$,
  i.e., $\mu' \models \xi(P)[V']$ and $\mu \cup \mu' \models \xi(R)[V,V'])$
  - Intuition: $\mu \Longleftrightarrow s$, $\mu' \Longleftrightarrow s'$, $\mu \cup \mu' \Longleftrightarrow \langle s,s' \rangle$

Example: simple counter



v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$

$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

$$\xi(PreImage(P,R)) = \exists V'.(\xi(P)[V'] \wedge \xi(R)[V,V'])$$

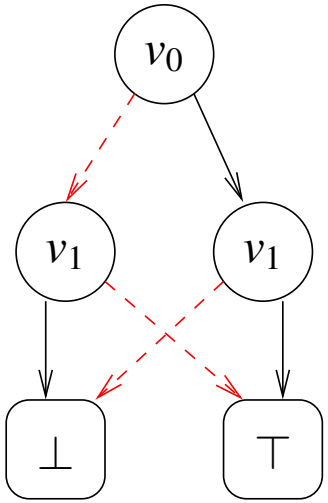
$$= \exists v_0' v_1'.((v_0' \leftrightarrow v_1') \wedge (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1))$$

$$= \underbrace{(\neg v_0 \wedge v_0 \bigoplus v_1)}_{v_0'=\top, v_1'=\top} \vee \underbrace{\bot}_{v_0'=\top, v_1'=\bot} \vee \underbrace{\bot}_{v_0'=\bot, v_1'=\top} \vee \underbrace{(v_0 \wedge \neg(v_0 \bigoplus v_1))}_{v_0'=\bot, v_1'=\bot}$$

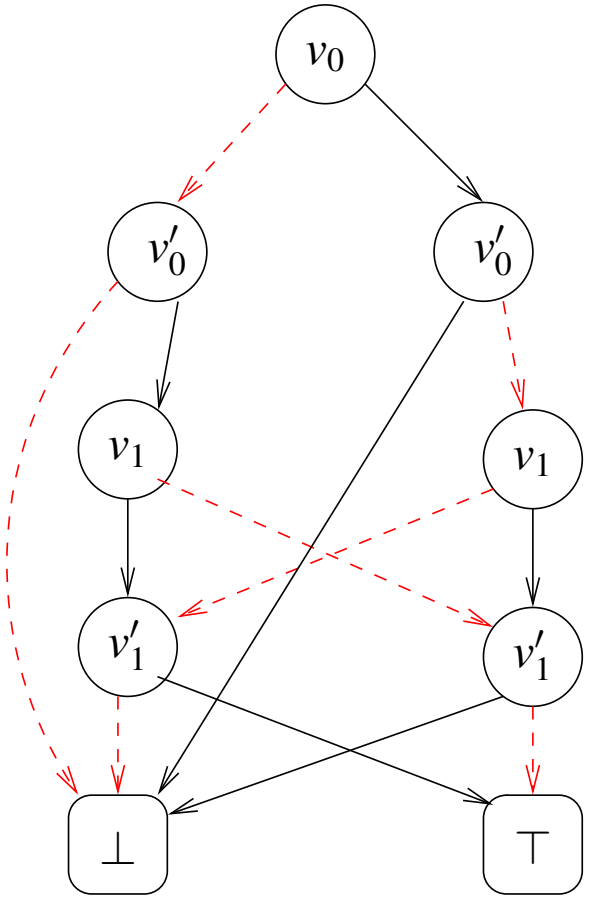
$$= v_1 \quad (i.e., \{10, 11\})$$

Preimage computation via OBDD's

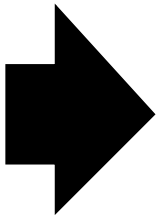
Symbolic M.C. use Ordered Binary Decision Diagramns (OBDDs) to represent boolean formulas $\Longrightarrow$allow for handling quantifiers very efficiently

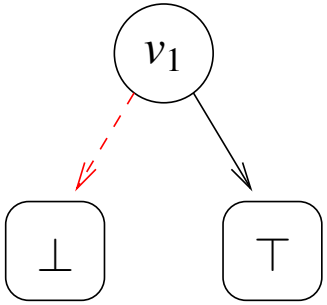


$\xi(P) = v_0 \leftrightarrow v_1$



$\xi(R) = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)$





$\xi(PreImage(P,R)) =$
$\exists V'.((v'_0 \leftrightarrow v'_1) \wedge (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)) =$
v_1

Application of the Transition Relation

- PreImage of a set of states S computed by means of quantified Boolean formulae
- The whole set of transitions can be fired (either forward or backward) in one logical operation
- The symbolic computation of PreImage provides the primitives for symbolic search of the state space of FSM's

Symbolic CTL model checking

- Problem: $M \models \varphi$?,
  - $M = \langle S, I, R, L, AP \rangle$ being a Kripke structure and
  - φ being a CTL formula
- Solution: represent I and R as boolean formulas $\xi(I), \xi(R)$ and encode them as OBDDs, and
- Apply fix-point CTL M.C. algorithm:
  - using OBDDs to represent sets of states and relations,
  - using OBDD operations to handle set operations
  - using OBDD quantification technique to compute PreImages
- $\xi([\varphi_i])$ computed directly, without computing $[\varphi_i]$ explicitly!!!
  - boolean operators handled directly by OBDDs
  - next temporal operators $\mathbf{EX}$: handled by symbolic PreImage computation
  - other temporal operators $\mathbf{EG}$, $\mathbf{EU}$: handled by fix-point symbolic computation

General M.C. Procedure

```
OBDD Check(CTL_formula β) {
    if (In_OBDD_Hash(β))
                    return OBDD_Get_From_Hash(β);
    case β of
    true:           return obdd_true;
    false:          return obdd_false;
    ¬β1:            return ¬ Check(β1);
    β1 ∧ β2:        return (Check(β1) ∧ Check(β2));
    EXβ1:           return PreImage(Check(β1));
    EGβ1:           return Check_EG(Check(β1));
    E(β1Uβ2):       return Check_EU(Check(β1),Check(β2));
}
```

PreImage

OBDD PreImage(**OBDD** X) $\{$

return $\exists V'.(\, X[V'] \wedge \xi(R)[V,V'])$;

$\}$

Check_EG

OBDD Check_EG(**OBDD** X) $\{$
    $Y' := X$;
    repeat
        $Y := Y'$;
        $Y' := Y \wedge PreImage(Y)$);
    until $(Y' \leftrightarrow Y)$;
return Y;
$\}$

Check_EU

OBDD Check_EU(**OBDD** X_1,X_2) $\{$

$Y' := X_2$;

repeat

$Y := Y'$;

$Y' := Y \vee (X_1 \wedge PreImage(Y))$;

until $(Y' \leftrightarrow Y)$;

return Y;

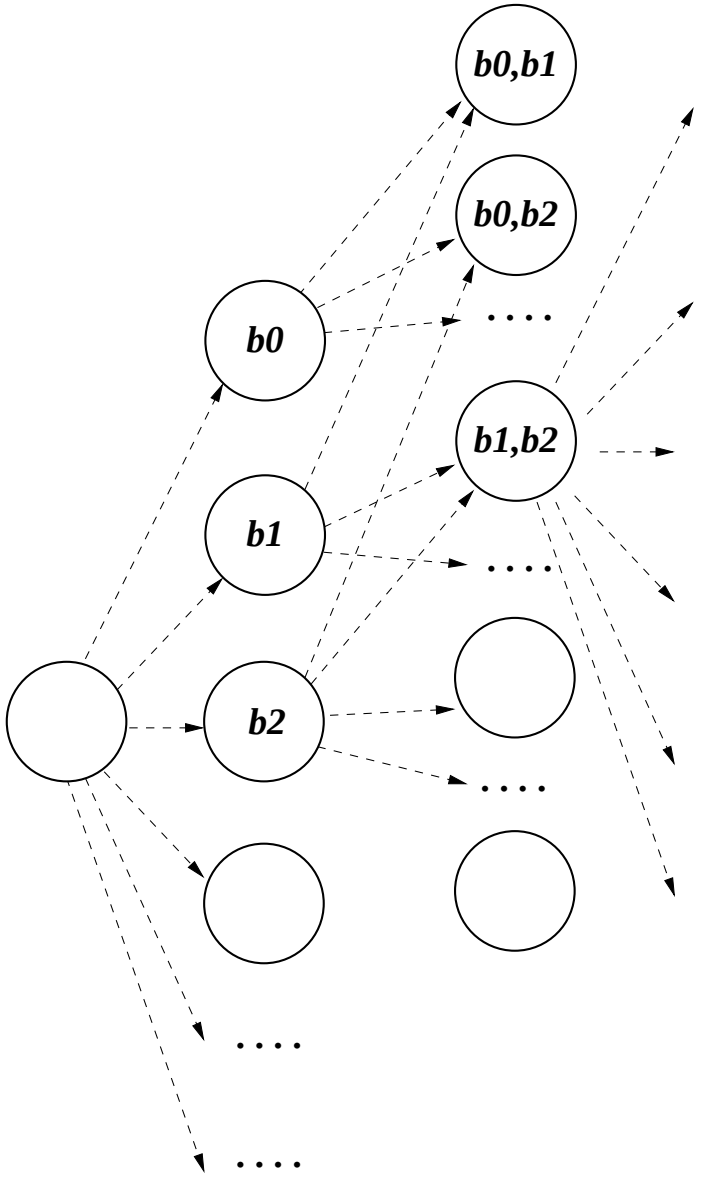
$\}$

A simple example

- N boolean variables $b0, b1, ...$
- Initially, all variables set to 0
- Each variable can pass from 0 to 1, but not vice-versa
- 2^N states, all reachable
- (Simplified) model of a student career behaviour.

```
MODULE main
VAR
  b0 : boolean;
  b1 : boolean;
  ...
ASSIGN
  init(b0) := 0;
  next(b0) := case
    b0  : 1;
    !b0 : {0,1};
  esac;
  init(b1) := 0;
  next(b1) := case
    b1  : 1;
    !b1 : {0,1};
  esac;
  ...
```

A simple example: explicit representation of the Kripke model

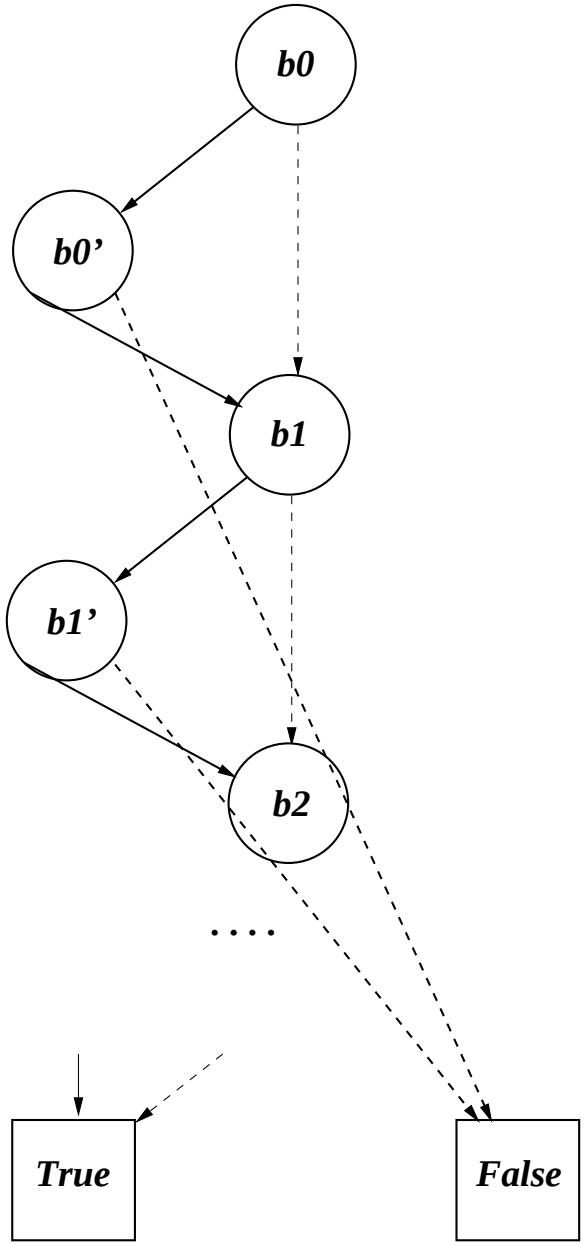


(transitive trans. omitted)

2^N STATES

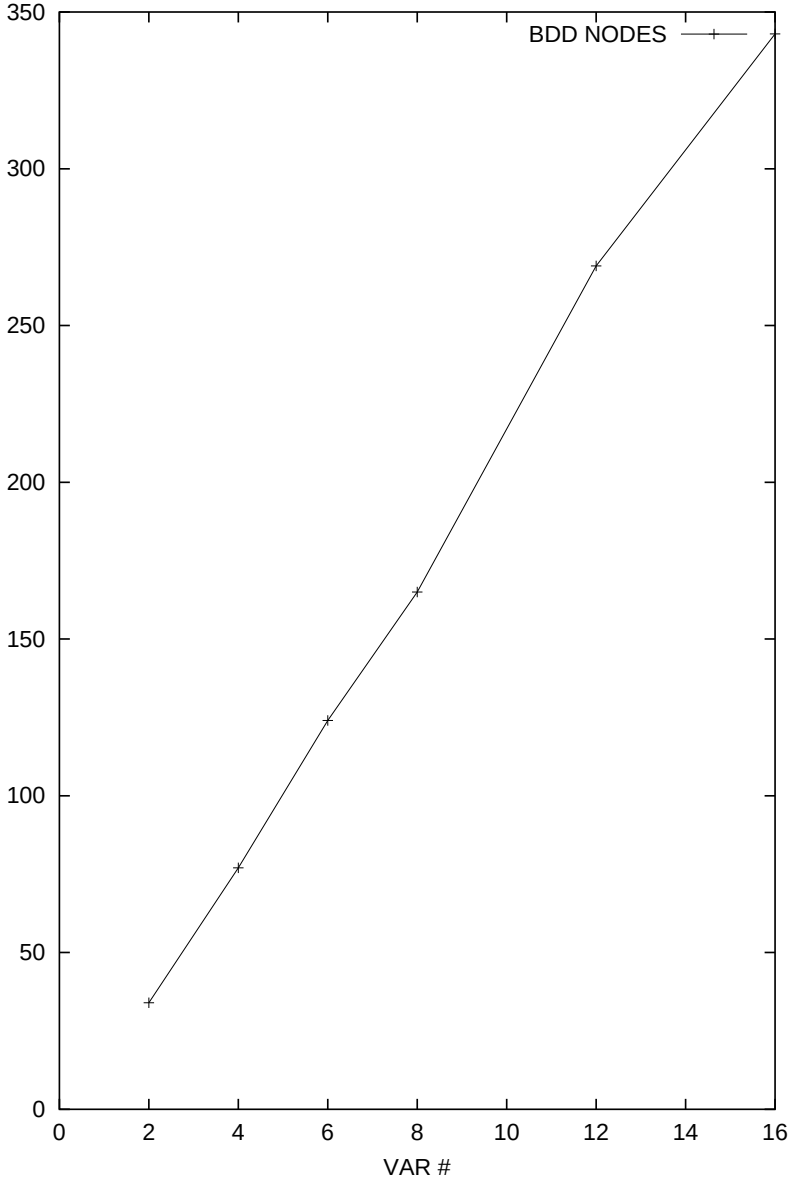
$O(2^N)$ TRANSITIONS

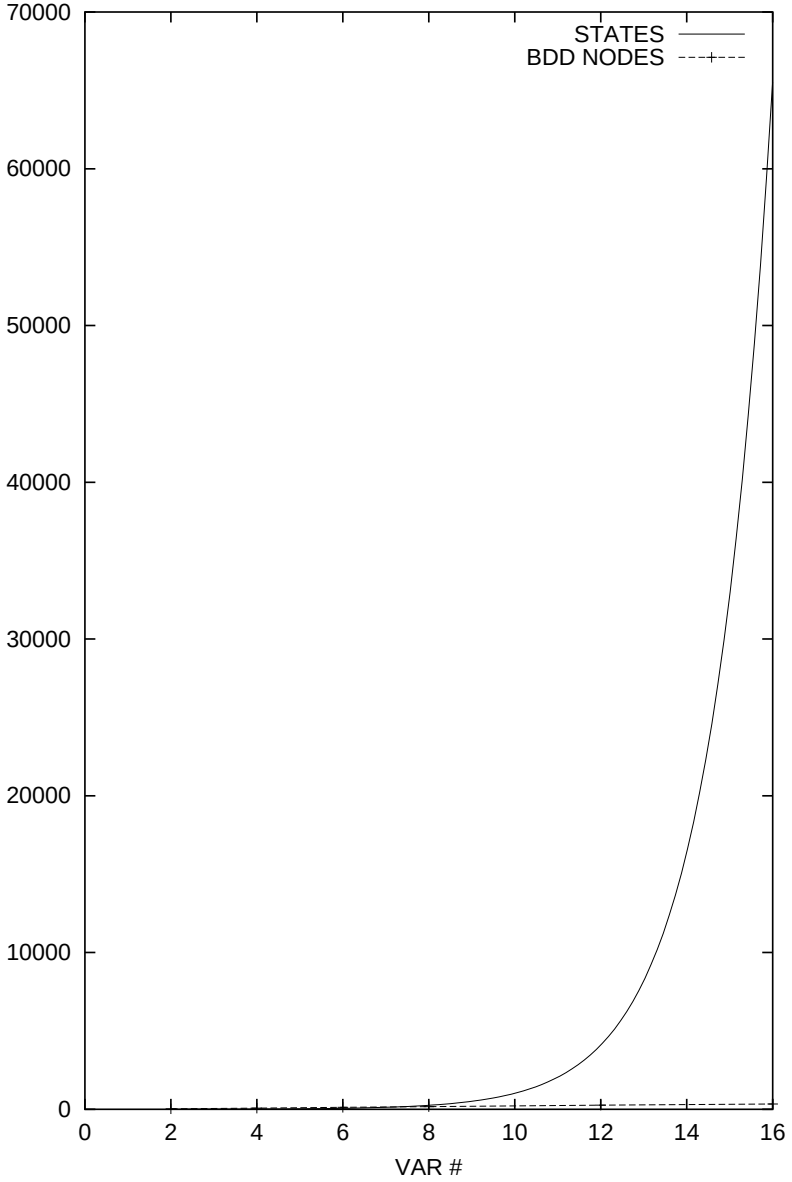
A simple example: $OBDD(\xi(R))$



$2N + 2$ NODES

A simple example: states vs. OBDD nodes [NuSMV.2]

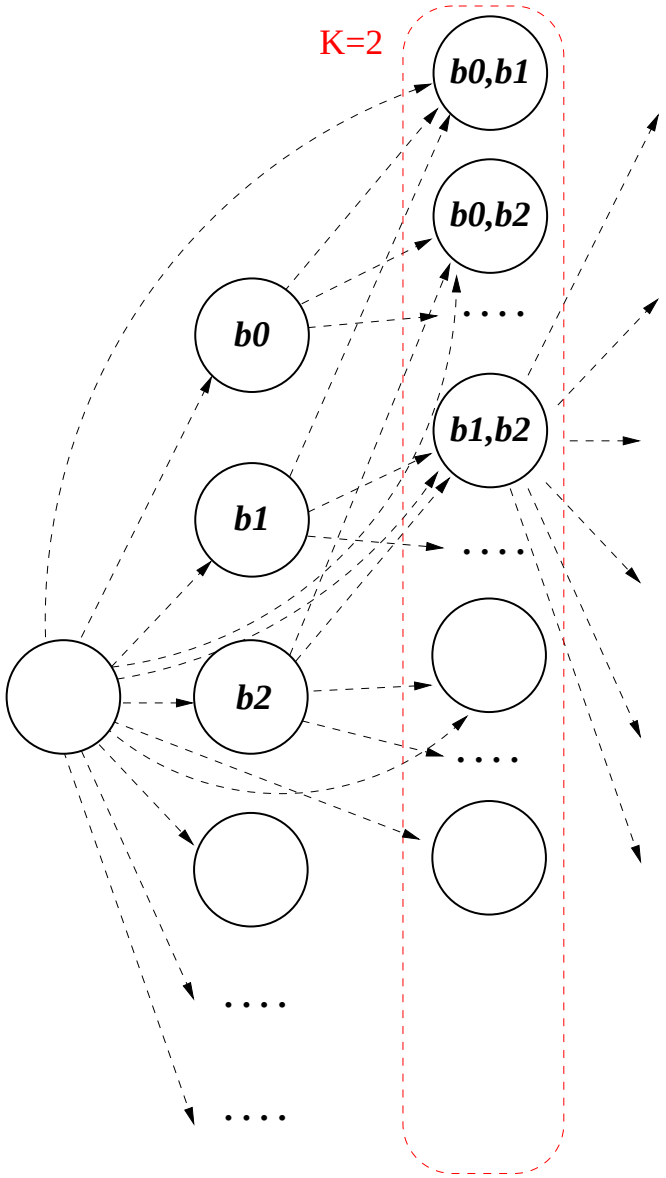




A simple example: reaching K bits true

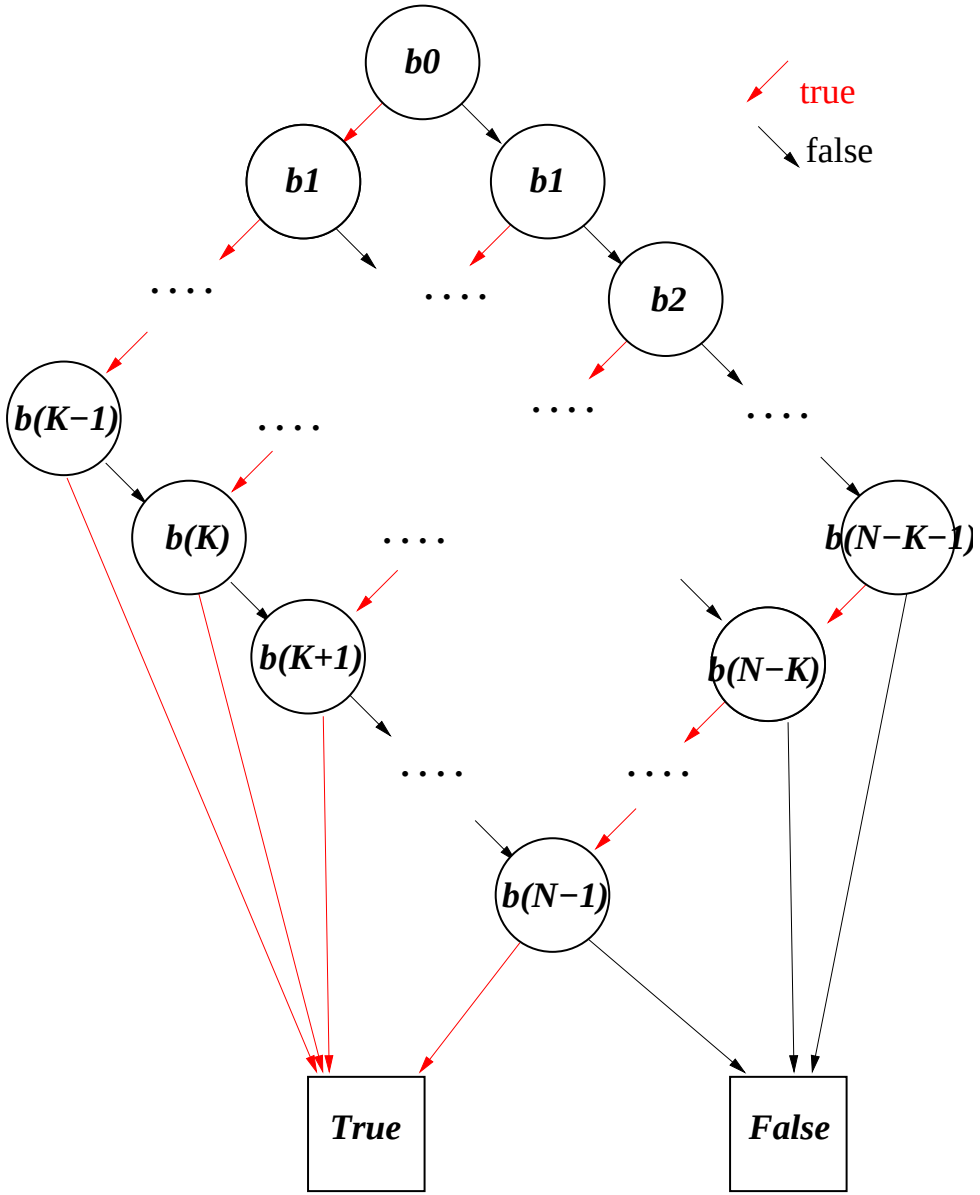
- Property $\mathbf{EF}(b0 + b1 + \ldots + b(N-1) = K)$ ($K \leq N$)
  (it may be reached a state in which K bits are true)
- E.g.: “it is reachable a state where K exams are passed”

A simple example: FSM



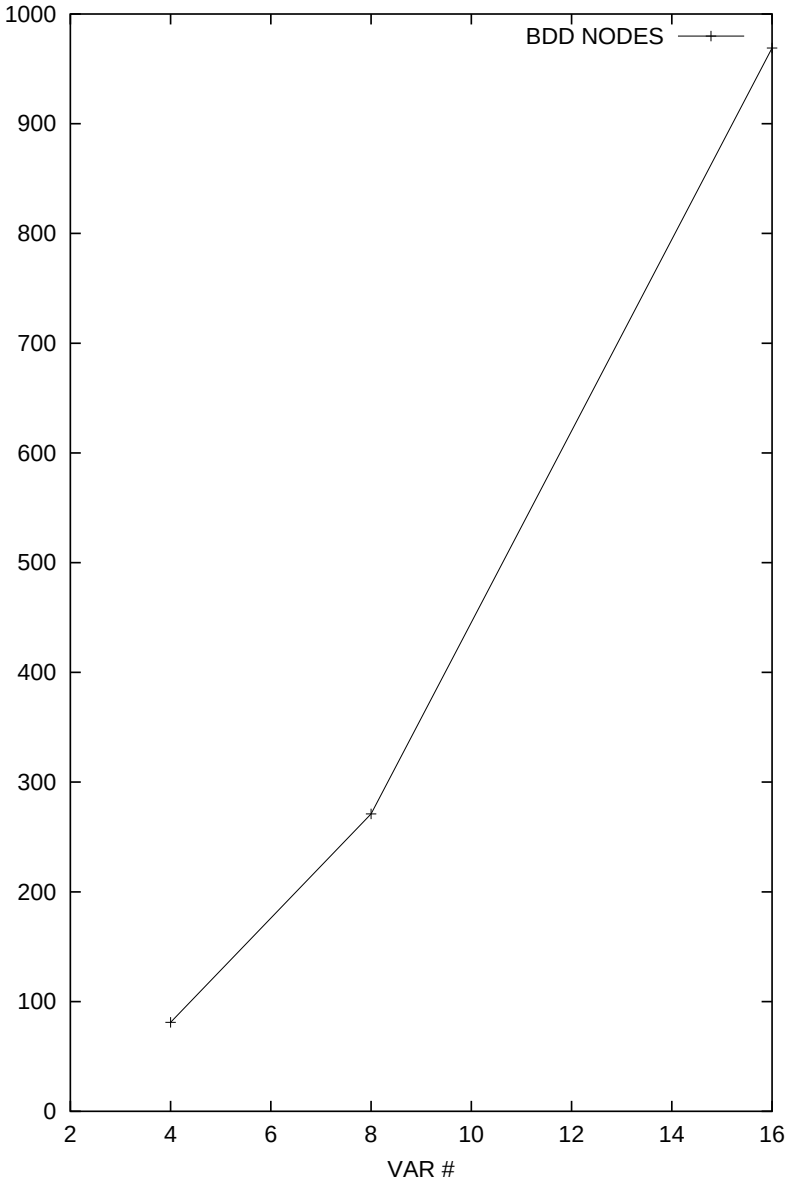
$$\binom{N}{K}$$ STATES

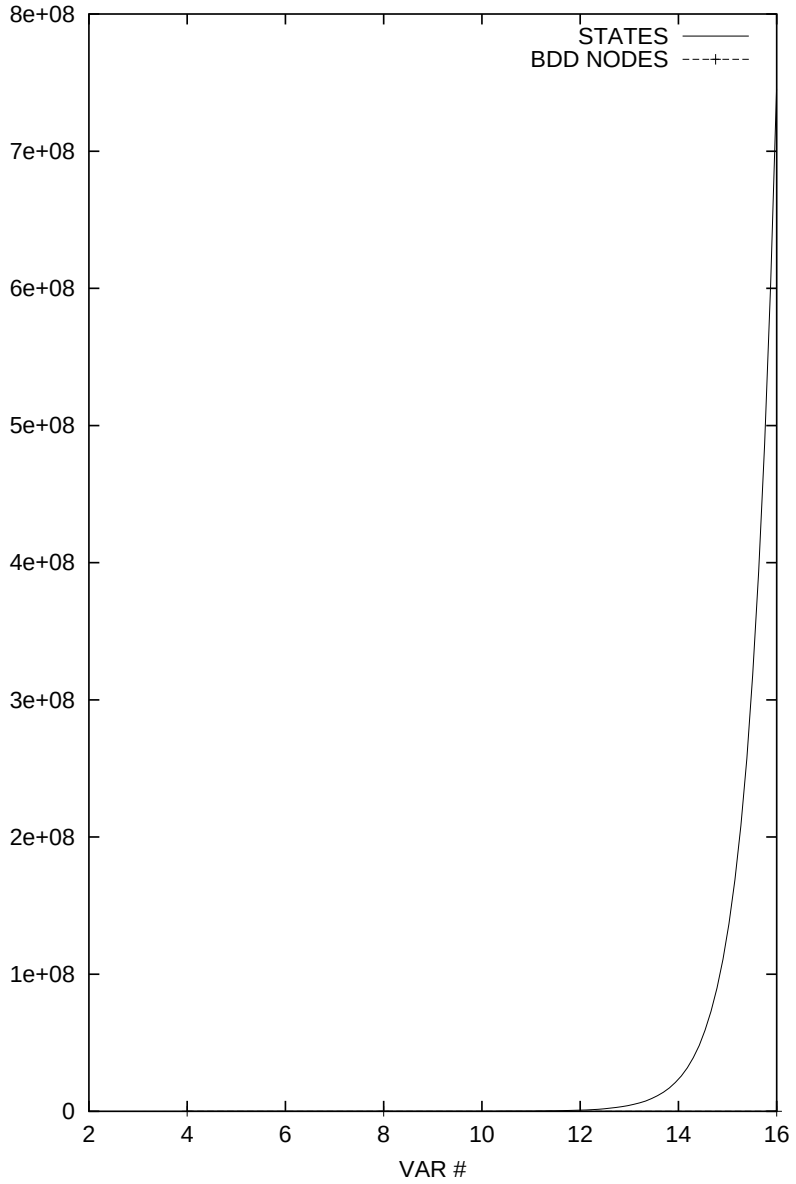
A simple example: $OBDD(\xi(\varphi))$



$(N-K)\cdot K+2$ NODES

A simple example: states vs. OBDD nodes [NuSMV.2]





Content

CTL Symbolic Model Checking – Summary

- Based on fixed point CTL M.C. algorithms
- Kripke structure encoded as boolean formulas (OBDDs)
- All operations handled as (quantified) boolean operations
- Avoids building the state graph explicitly
- reduces dramatically the state explosion problem

  $\Longrightarrow$ PROBLEMS OF UP TO 10^{120} STATES HANDLED !

Symbolic Model Checkers

- Most hardware design companies have their own Symbolic Model Checker(s)
  - Intel, IBM, Motorola, Siemens, ST, Cadence, ...
  - very advanced tools
  - proprietary technology
- On the academic side
  - CMU SMV [McMillan]
  - VIS [Berkeley, Colorado]
  - Bwolen Yang's SMV [CMU]
  - NuSMV [CMU, IRST, UNITN, UNIGE]
  - ...

Other Symbolic M.C. Techniques

- Symbolic Model Checking for LTL
  - based on conversion to fair CTL MC
  - same fixpoint techniques and OBDD technology as with CTL
- Bounded Model Checking for LTL
  - based on SAT technology
  - incomplete, though extremely efficient
- Counter-example-guided-abstraction-refinement
  - based on abstraction-refinement paradighm
  - based on a mixed SAT & OBDD technology
  - incomplete, though extremely efficient
- verification of “timed” and “hybrid” systems
  - based on timed & Hybrid automata
  - combine logical with mathematical reasoning
- ...

Lots of ongoing research!

To learn more...

Books:

- An essential book:

  E. Clarke, O. Grunberg, D. Peled "Model Checking" – MIT Press.

Slides:

- A full 100-hour course on formal verification & model checking:

  R. Sebastiani "Introduction to Formal Methods"
  slides available at `http://disi.unitn.it/~rseba/DIDATTICA/fm2008/`

To learn more...

Books:

- An essential book:

  E. Clarke, O. Grunberg, D. Peled "Model Checking" – MIT Press.

Slides:

- A full 100-hour course on formal verification & model checking:

  R. Sebastiani "Introduction to Formal Methods"
  slides available at `http://disi.unitn.it/~rseba/DIDATTICA/fm2008/`

THANK YOU FOR YOUR ATTENTION