Introduction to Model Checking

Debdeep Mukhopadhyay
IIT Madras

How good can you fight bugs?

Comprising of three parts

- Formal Verification techniques consist of three parts:

1. A framework for modeling systems
   - some kind of specification language
2. A specification language
   - for describing the properties to be verified
3. A verification method
   - for establishing if the description of the system satisfies the specification

Proof-based verification

- The system description is a set of formula Γ in a suitable logic
- The specification is another formula φ
- The verification method is finding a proof that $\Gamma \vdash \varphi$
  - $\vdash$ means deduction
- It typically needs the user guidance and expertise

Model-based verification

- The system is represented by a model $\mathcal{M}$ for an appropriate logic
- The specification is again represented by a formula φ
- The verification method consist of **computing** whether a model $\mathcal{M}$ satisfies φ
  - $\mathcal{M}$ satisfies φ : $\mathcal{M} \models \varphi$
- The computation is usually **automatic** for finite models

Degree of automation

- From fully automated to fully manual

Full- vs. property-verification

- The specification may describe a single property of the system, or it may describe its full behavior (expensive).

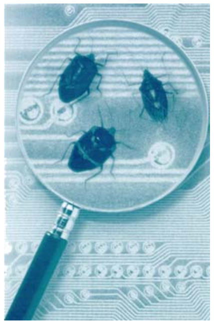
Intended domain of application

- Hardware, software
- Sequential, concurrent
- Reactive , terminating
  - Reactive: reacts to its environment, and is not meant to terminate (e.g. operating systems, embedded systems, computer hardware)

Pre- vs. post-development

- Verification is of greater advantage if introduced early in system development

Model checking

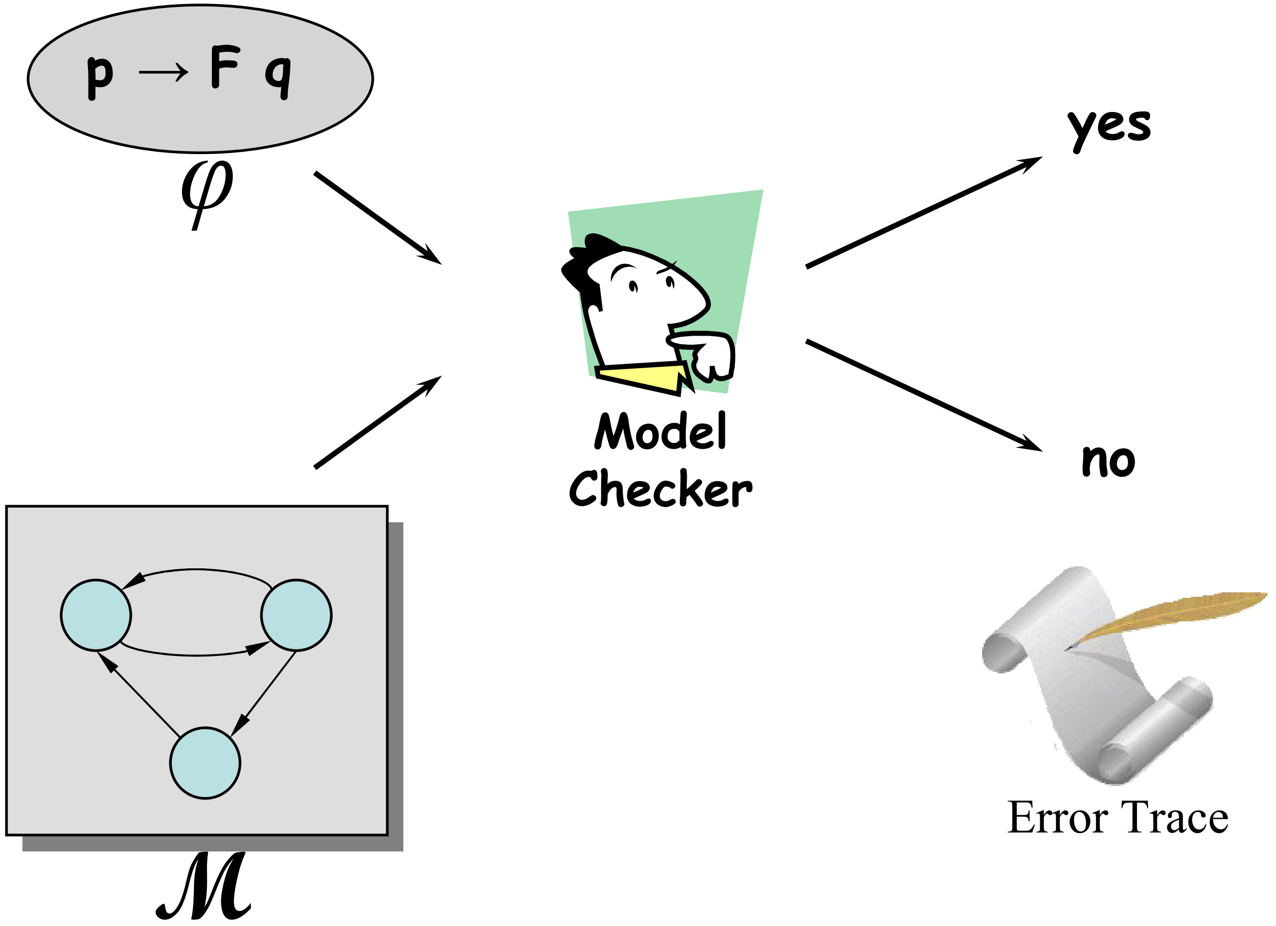


- Model checking is an *automatic*, *model-based*, *property-verification* approach
- It is intended to be used for *concurrent* and *reactive* systems
  - The purpose of a reactive system is not necessarily to obtain a final result, but to maintain some interaction with its environment

Temporal Logic (cont.)

- In model checking:
  - The models $\mathcal{M}$ are transition systems
  - The properties φ are formulas in temporal logic
- Model checking steps:
  1. Model the system using the description language of a model checker : $\mathcal{M}$
  2. Code the property using the specification language of the model checker : φ
  3. Run the model checker with the inputs $\mathcal{M}$ and φ

Model checker based on satisfaction



$p \rightarrow F\,q$

φ

$\mathcal{M}$

Model Checker

yes

no

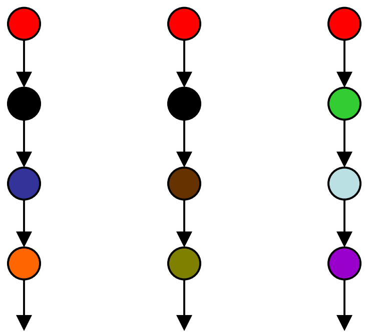
Error Trace

Linear vs. Branching

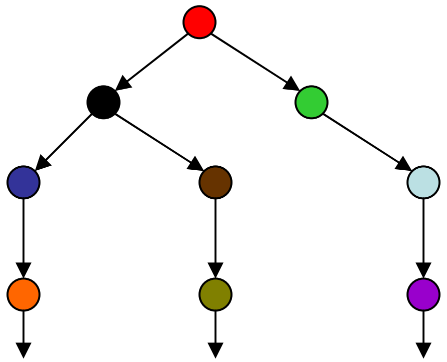
- Linear-time logics think of time as a **set of paths**
  - path is a sequence of time instances
- Branching time logics represent time as a **tree**
  - it is rooted at the present moment and branches out into the future
- Many logics were suggested during last years that fit into one of above categories
- We study LTL in linear time logics and CTL in branching time logics

Linear vs. Branching (cont.)

- 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)

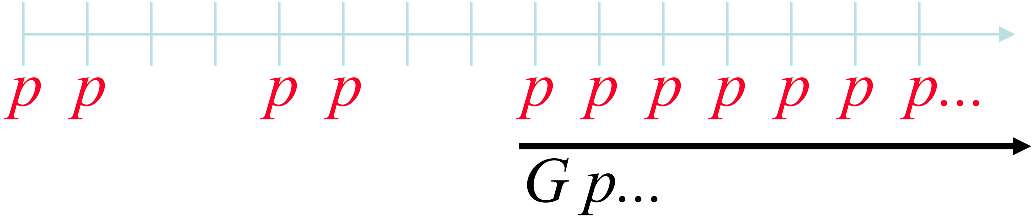


Propositional Linear Temporal Logic

- Express properties of “Reactive Systems”
  - interactive, nonterminating
- For PLTL, a *model* is an infinite state sequence

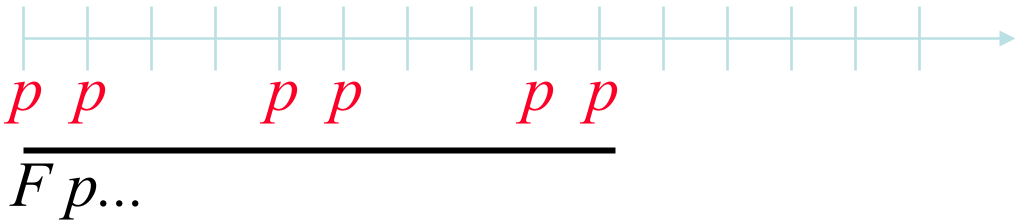
$$\sigma = s_0, s_1, s_2 \ldots$$

- Temporal operators
  - “Globally”: *G p* at *t* iff *p* for all *t’* ≥ *t.*

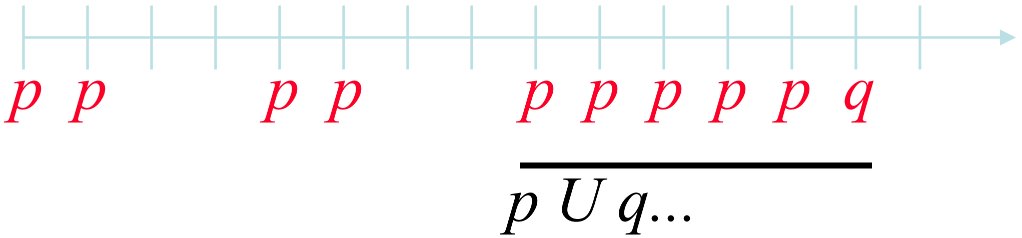


Temporal operators...

- "Future": $F\ p$ at t iff p for some $t' \geq t$.



- "Until": $p\ U\ q$ at t iff
  - q for some $t' \geq t$ and
  - p in the range $[\, t, t' \,)$



- "Next-time": $X\ p$ at t iff p at $t+1$

Examples

- Liveness: “if input, then eventually output”

$$G\ (\text{input} \Rightarrow F\ \text{output})$$

atomic props

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

$$\text{GF send} \;\Rightarrow\; \text{GF recv}$$

infinitely often

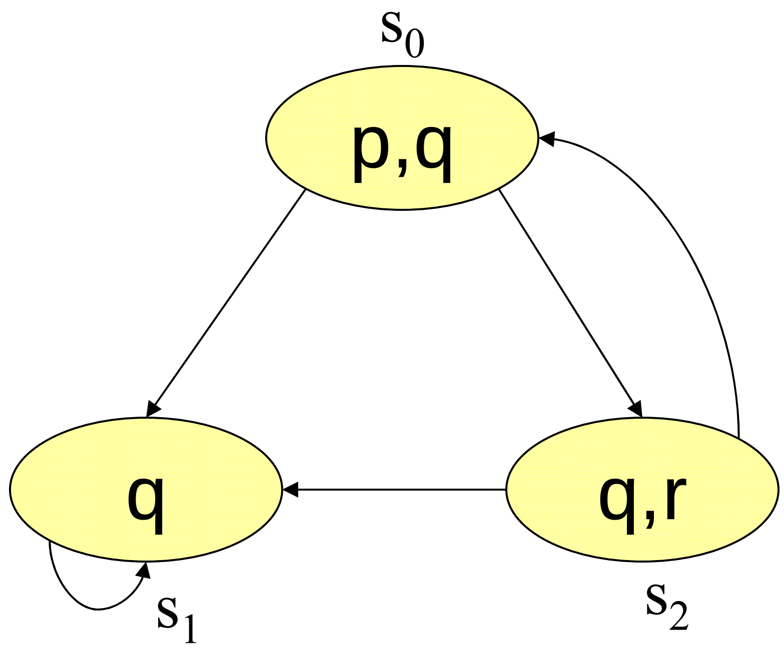
Recap: What is a model?

- Atoms: Atomic formulas (such as p. q, r, …).
- These atoms stand for atomic facts which may be true for a system.
- e,g
  - Printer crypto-6 is working
  - Process encipher is suspended
  - Content of the register ‘key’ is the integer value 6

Model

- A Model is a transition system.
- A transition system M=(S,→,L) is a set of states S endowed with a transition relation → (a binary relation on S), such that every state s from S, has some successor state s’ which is also from S. Thus s→s’
- Also associated with each state is a set of atomic propositions which are true at that state, described by a labeling function, L

Example



$S = \{s_0, s_1, s_2\}$

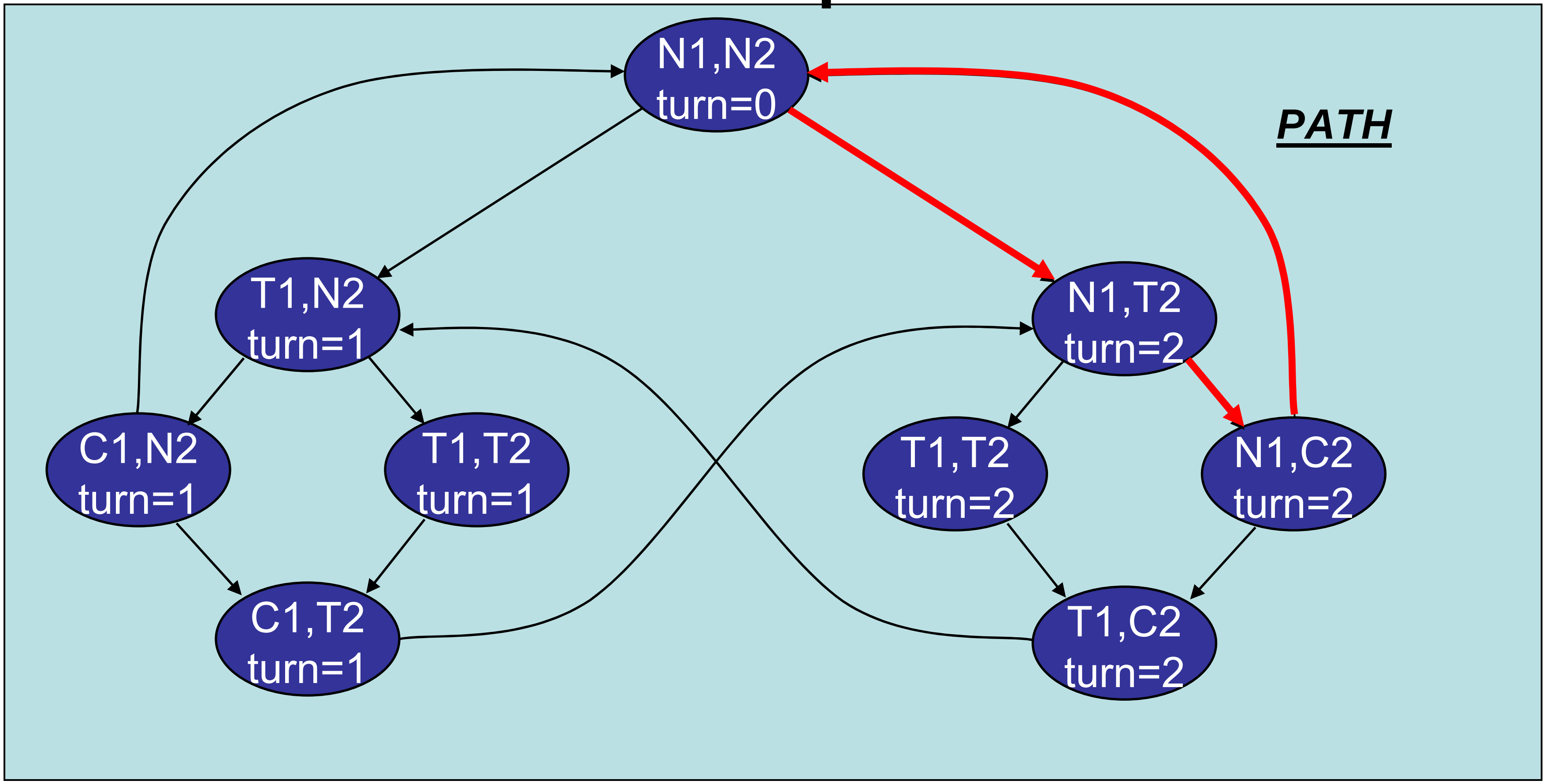
transitions = $s_0 \rightarrow s_1$, $s_1 \rightarrow s_1$, $s_2 \rightarrow s_1$, $s_2 \rightarrow s_0$, $s_0 \rightarrow s_2$

$L(s_0) = \{p,q\}$

$L(s_1) = \{q\}$

$L(s_2) = \{q,r\}$

Example



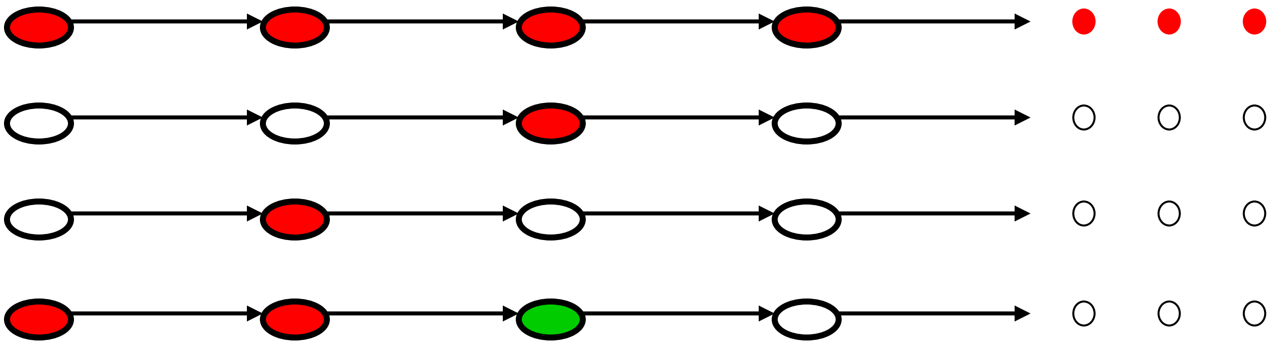
N = noncritical, T = trying, C = critical

Propositional temporal logic

In **Negation Normal Form**

AP – a set of atomic propositions

Temporal operators:



Gp

Fp

Xp

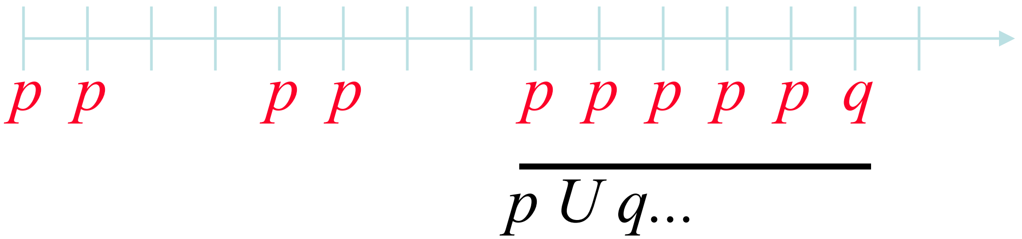
pUq

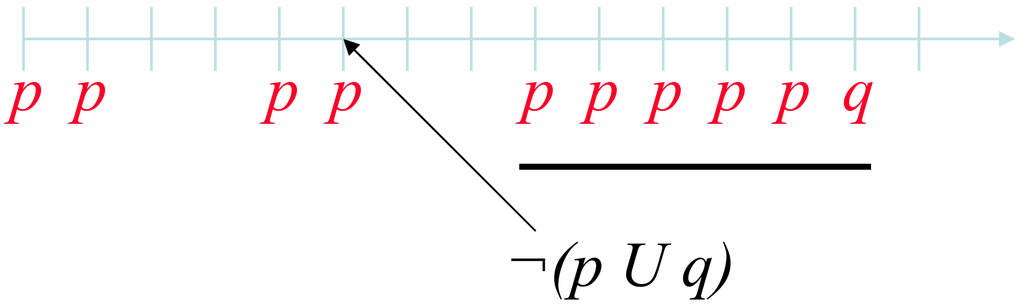
Path quantifiers: A for **all** path

E there **exists** a path

Not Until ¬(**p**U**q**)

- Whenever q occurs there must be a non-occurrence of p before.





Explanation

$$p \cup q := \exists i[(\Pi^i \models q) \wedge (\forall j < i, \Pi^j \models p)]$$

$$\neg(p \cup q) := \forall i[\neg(\Pi^i \models q) \vee (\exists j < i, \Pi^j \models \neg p)]$$

$$:= \forall i[(\Pi^i \models q) \Rightarrow (\exists j < i, \Pi^j \models \neg p)]$$

Some Finer Points on p U q

- Until demands that q does hold in some future state i,e Fq
- It does not say anything about what happens after q occurs
  - contrary to English Language: “*I smoked until 22’*
  - *Means p=‘I smoke’ was true till q=‘I am 22’ became true.*
  - *Also after q=‘I am 22’, p=‘I smoke’ does not occur*
  - *In LTL, means p U (G¬p Λ q)*

Two more terms

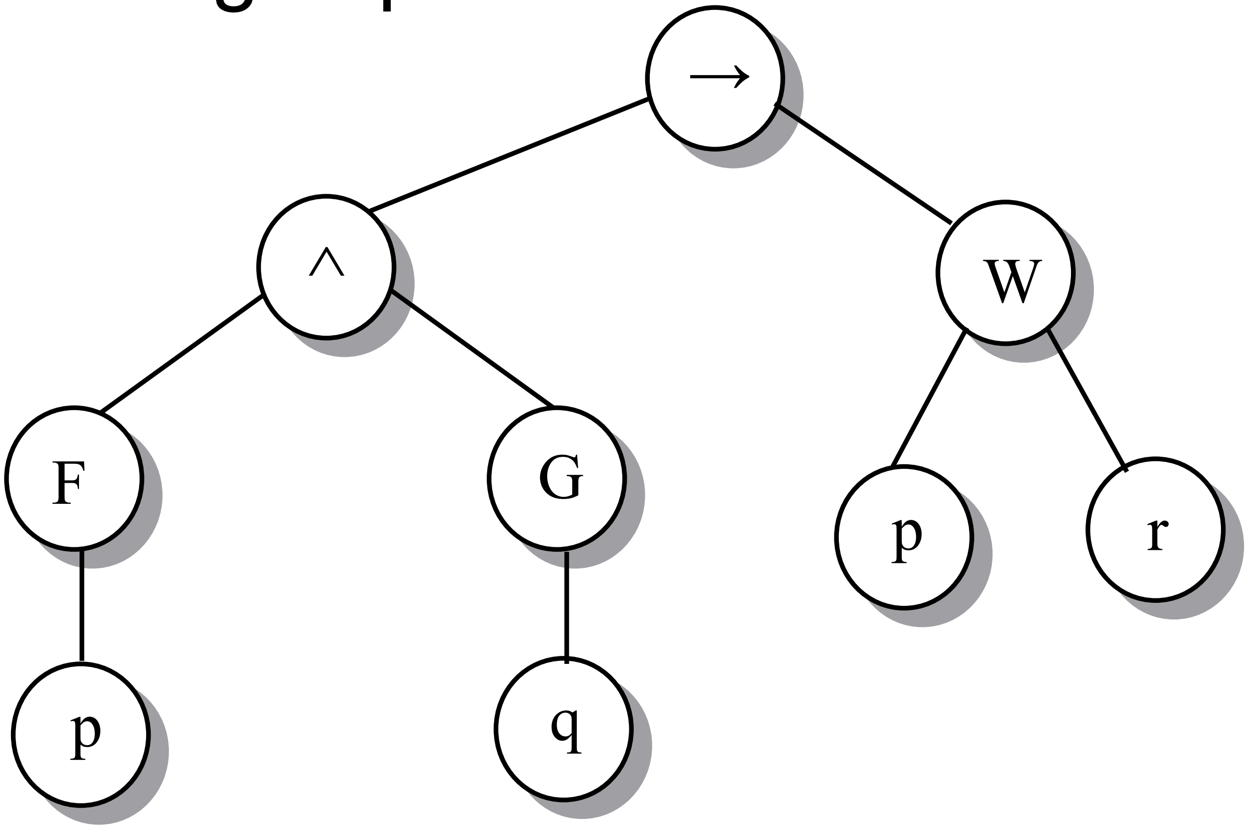
- **Weak Until** (pWq): Like pUq except q need not occur.
- **Release** (pRq): p is released by q. It means that q occurs entirely or it occurs till p occurs. Note than unlike until q occurs also at the time instant when p is asserted.

Operator precedence

- Unary operators including negation have strongest precedence
  - $\neg p\ U\ q$ is parsed as $(\neg p)\ U\ q$ rather than $\neg(p\ U\ q)$
- Temporal binary operators have stronger precedence than non-temporal binary operators
  - $p \wedge q\ U\ r$ is parsed as: $p \wedge (q\ U\ r)$
- The precedence over propositional logic is as usual
  - First do the AND
  - then the ORs and XORs
  - finally the IMPLIES and EQUIVALENCEs.

Example

- The parse tree of $Fp \wedge Gq \rightarrow p\ W\ r$ according to precedence rules



More of Until

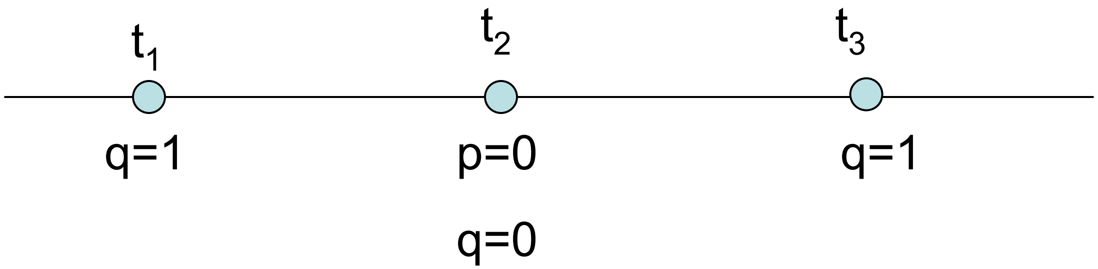
- What is not pUq?
- We have seen that.
- Here is another expression for that.

$$\boxed{\neg(p \cup q) = \neg q \cup (\neg p \wedge \neg q) \vee G \neg q}$$

Intuitive Explanation

$$\boxed{(p \cup q) = \neg(\neg q \cup (\neg p \wedge \neg q)) \wedge Fq}$$

- Fq is straight-forward
- Let q occur => Fq



Let t_3 be the first time interval when q is true.

Let us contradict the equation, that is pUq does not hold.

Then, there is a time instant $t=t_2$, when p=0. Obviously q=0, as $t_2<t_3$

But by RHS, if $\neg(\neg q \cup (\neg p \wedge \neg q))$ then at time $t=t_1$, ¬q=0 => q=1

But, $t_1<t_3$ and hence we have a violation that t_3 is the first time when q=1. Thus, there is a contradiction and pUq does hold. The equivalence follows.

Release

- Release R is dual of U; that is:

  $p \text{ R } q \equiv \neg (\neg p \text{ U } \neg q)$

  p must remain true up to and including the moment when *q* becomes true (if there is one); *p* releases *q*

Thus, $pRq = Gq \lor [q \text{ U } (p \land q)]$

$= \neg[F \neg q \land \neg(q \text{ U } (p \land q)]$

$= \neg[\neg p \text{ U } \neg q]$

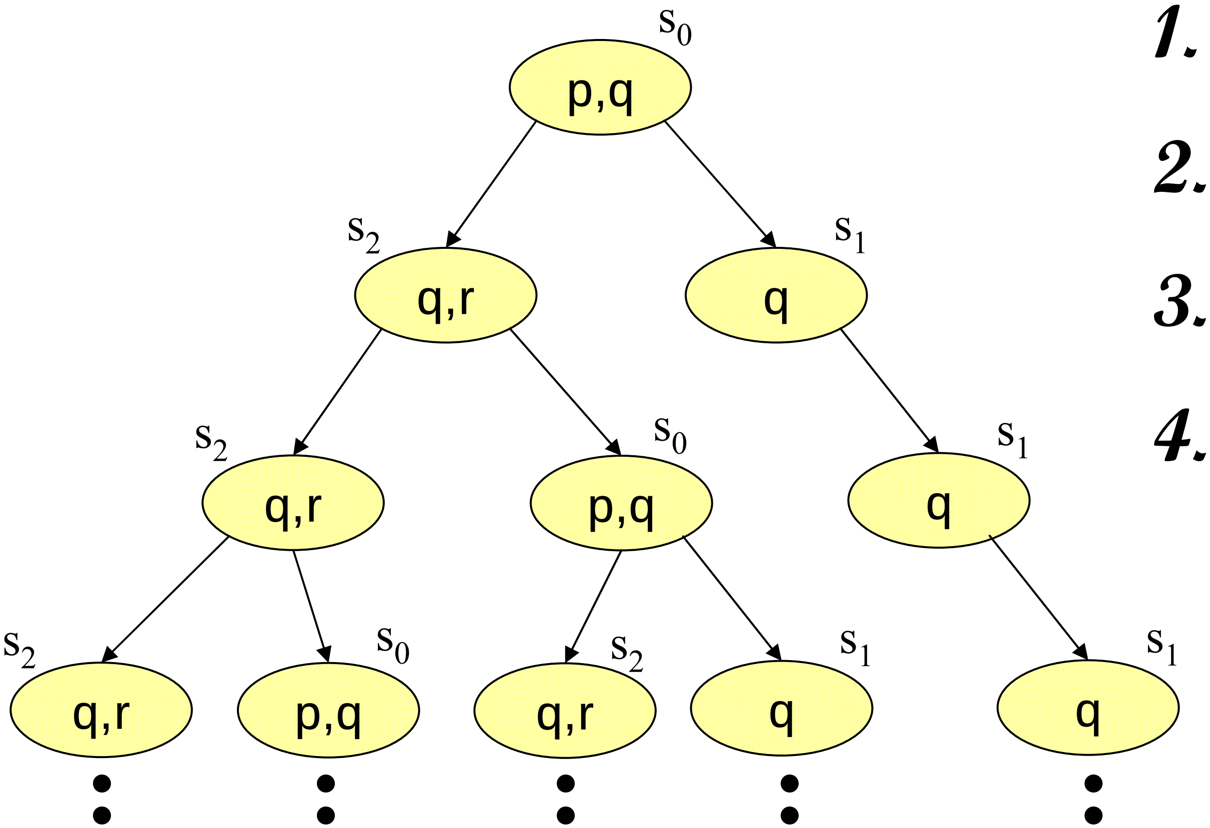
Weak Until

- φ W ψ : Weak Until is related to the Until with the difference that it does not require that ψ is eventually hold
- Essentially φ W ψ is a short form for writing φ U $\psi \vee$ Gφ

LTL satisfaction by a system

- Suppose $\mathcal{M} = (S, \to, L)$ is a model, $s \in S$, and φ an LTL formula
- We write $\mathcal{M}, s \models \varphi$ if for every execution path π of $\mathcal{M}$ starting at s, we have $\pi \models \varphi$
- Sometimes $\mathcal{M}, s \models \varphi$ is abbreviated as $s \models \varphi$

Example



1. $\mathcal{M}, s_0 \models \mathrm{X}\, q$
2. $\mathcal{M}, s_0 \models \mathrm{G}\, \neg(p \wedge r)$
3. $\mathcal{M}, s_1 \models \mathrm{G}\, q$
4. $\mathcal{M}, s_0 \models p\, \mathrm{U}\, q$

Practical patterns of specifications

- It is impossible to get to a state where *started* holds, but *ready* does not hold
  - $G \neg(\mathit{started} \wedge \neg \mathit{ready})$
- For any state, if a request occurs, then it will eventually be acknowledged
  - $G\ (\mathit{requested} \rightarrow F\ \mathit{acknowledged})$
- Whatever happens, a certain process will eventually be permanently *deadlocked*
  - $F\ G\ \mathit{deadlock}$

Some practical patterns (cont.)

- A certain process is *enabled* <u>infinitely often</u> on every computation path
  - G F *enabled*
  - In other words, in a path of the system there must never be a point at which the condition *enabled* becomes false and stays false forever
- If a process is enabled infinitely often, then it runs infinitely often
  - G F *enabled* → G F *running*

Practical patterns(contd.)

- An upwards travelling lift at the 2nd floor does not change its direction when it has passengers wishing to go to the 5th floor:

G(floor2 Λ directionup Λ ButtonPressed5 →(directionup U floor5)

LTL weakness

- The features which assert the existence of a path are **not** (directly) expressible in LTL
- This problem can be solved by: checking whether all paths satisfy the negation of the required property
- A positive answer to this is a negative answer to our original question and vice versa.
- *But properties which mix universal and existential path quantifiers cannot in general be expressed in LTL*

LTL Weakness: Examples

- LTL cannot express these features:
  - From any state it is *possible* to get to a restart state (i.e., there is a path from all states to a state satisfying restart)
  - The lift *can* remain idle on the third floor with its door closed (i.e., from ***all*** states if there is path to a state in which it is on the third floor, ***there is*** a path along which it stays there)
- LTL cannot assert these because existential and universal logics are mixed.
- However, CTL can express these properties

Model checking example: Mutual exclusion

- The mutual exclusion problem (mutex)
  - Avoiding the simultaneous access to some kind of resources by the *critical sections* of concurrent processes
- The problem is to find a *protocol* for determining which process is allowed to enter its critical section

Expected Properties

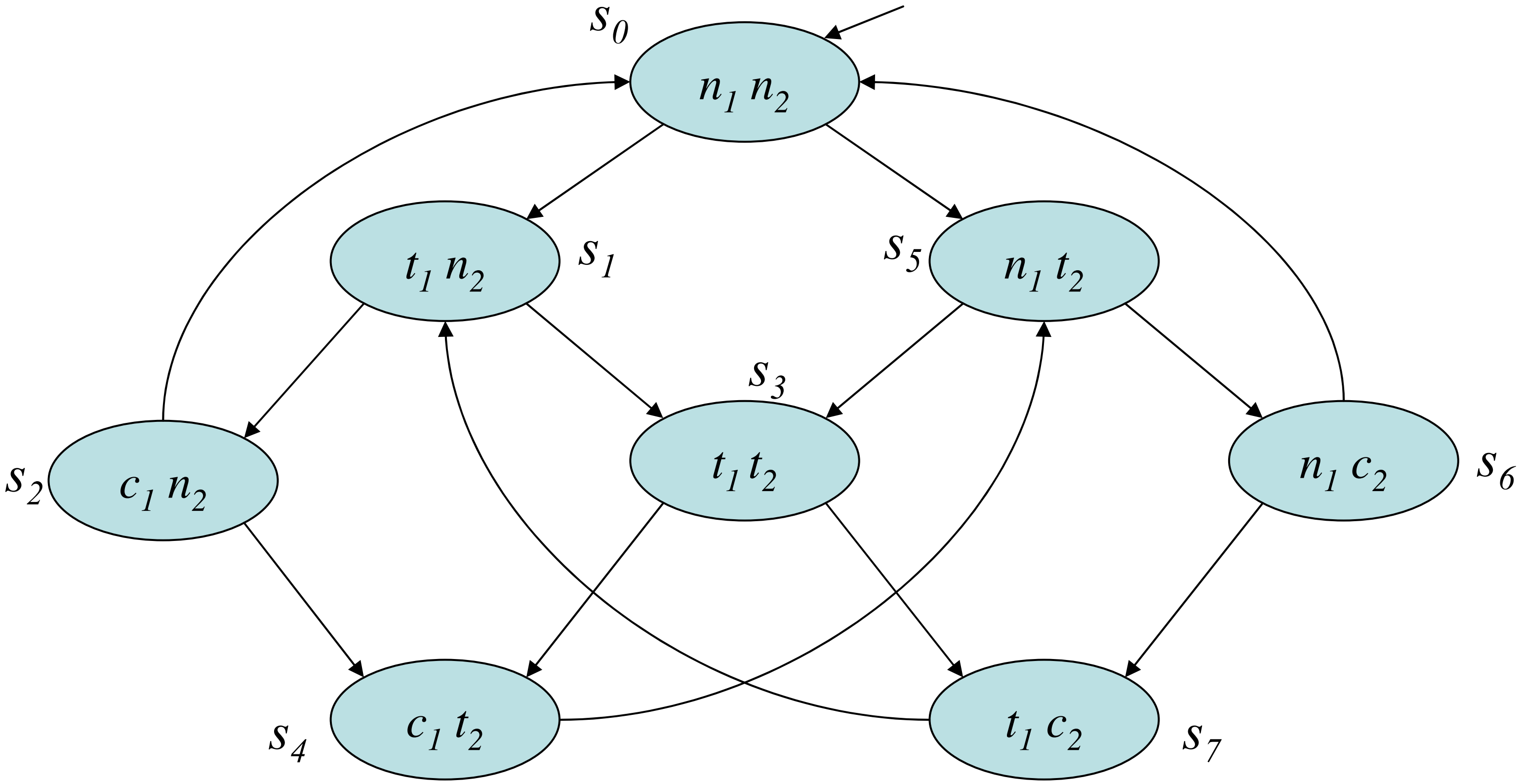
- **Safety:** Only one process is in its critical section at any time.
- **Liveness:** Whenever any process requests to enter its critical section, it will eventually be permitted to do so.
- **Non-blocking:** A process can always request to enter its critical section.
- **No strict sequencing**: Processes need not enter their critical section in strict sequence.

Modeling mutex

- Consider each process to be either:
  - in its non-critical state *n*
  - trying to enter the critical section *t*
  - or in critical section *c*
- Each individual process has this cycle:
  - $n \rightarrow t \rightarrow c \rightarrow n \rightarrow t \rightarrow c \rightarrow n$ ...
- The processes phases are interleaved

2 process mutex

- The processes are *asynchronous interleaved*
  - – one of the processes makes a transition while the other remains in its current state



Checking the properties

- **Safety**: $G\ \neg(c_1 \wedge c_2)$
  - This formula is satisfied in all states
- **Liveness**: $G\ (t_1 \rightarrow F\ c_1)$
  - This formula is not satisfied in the initial state!
  - $s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow \ldots$

Checking the properties

- Non-blocking:
  - – Consider process 1.
  - – We wish to check the following property:
    - *for every states satisfying* n_1 *there exists a state which satisfies* t_1
  - – This property cannot be expressed in LTL

Checking the properties

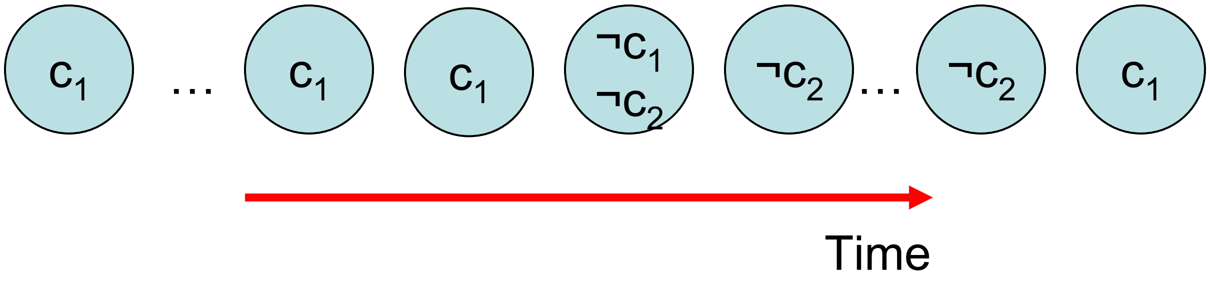
- No strict sequencing:
  - Processes should not enter their critical section in a strict sequence.
  - There should be at least one path where strict sequencing does not hold
  - But LTL cannot express the logic there exists.
  - Instead not of there exists is for all.
  - Thus we can say that the following property s:
    - in all paths there is a strict sequencing
  - If the answer is no there is no strict sequence.

No Strict Sequencing

- c_1 and c_2 need not alternate
- Desired scenario:
  - Process 1 acquires critical section (c_1)
  - Process 1 releases the critical section ($\neg c_1$)
  - Process 2 does not enter the critical section ($\neg c_2$)
  - Process 1 regains access to the critical section (c_1)

No Strict Sequencing

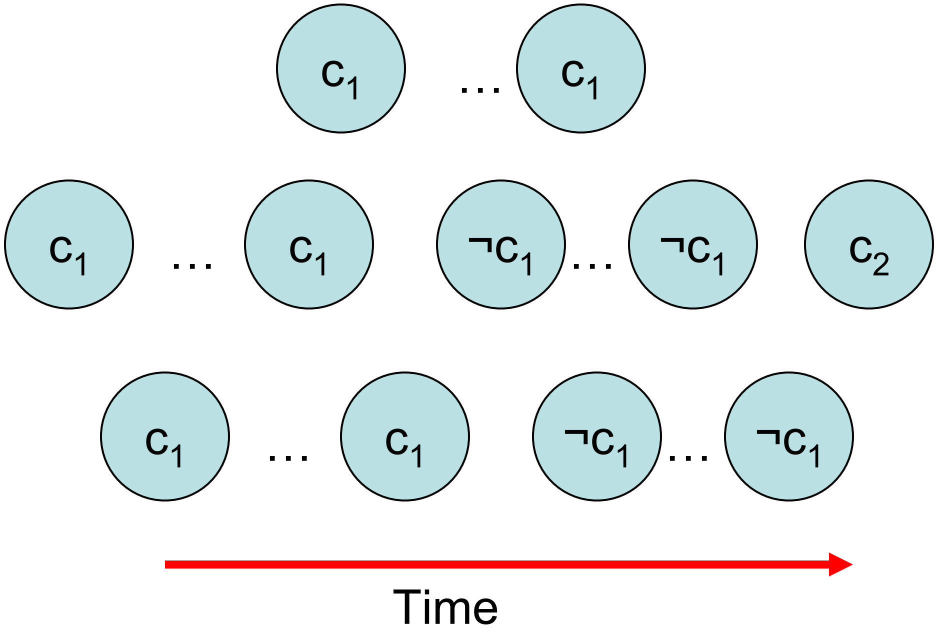
There exists at least one path with no strict sequencing:



Or, in all paths there is strict sequencing:

Anytime we have c_1 state, the condn persists, or it ends with a non-c_1 state and in that case there is no further c_1 unless and until we obtain a c_2 state.

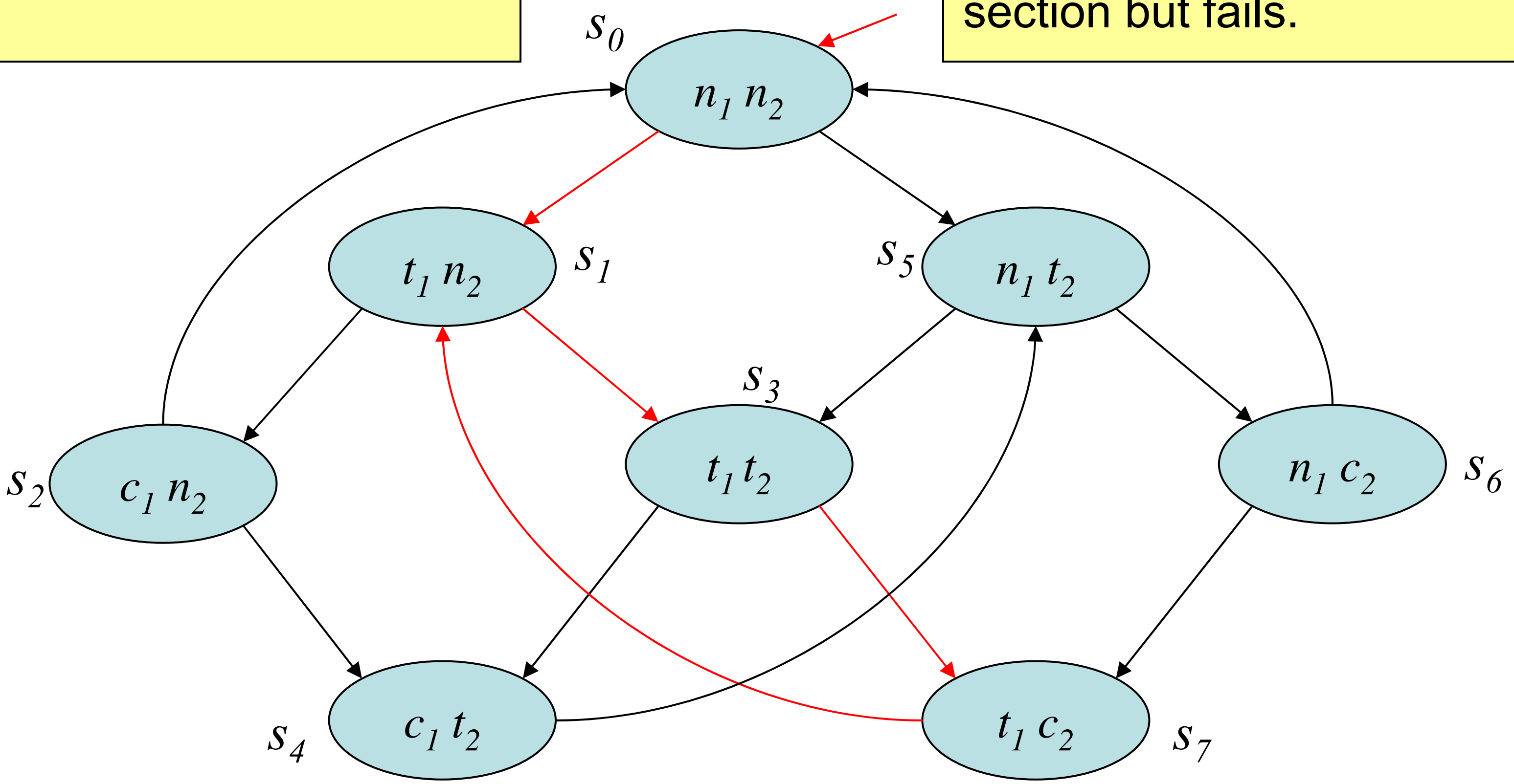
$$G[c_1 \rightarrow c_1 W(\neg c_1 \wedge \neg c_1 W c_2)]$$



Evaluation of the Protocol

Safety property is satisfied. c_1 and c_2 do not become one at the same time in any state.

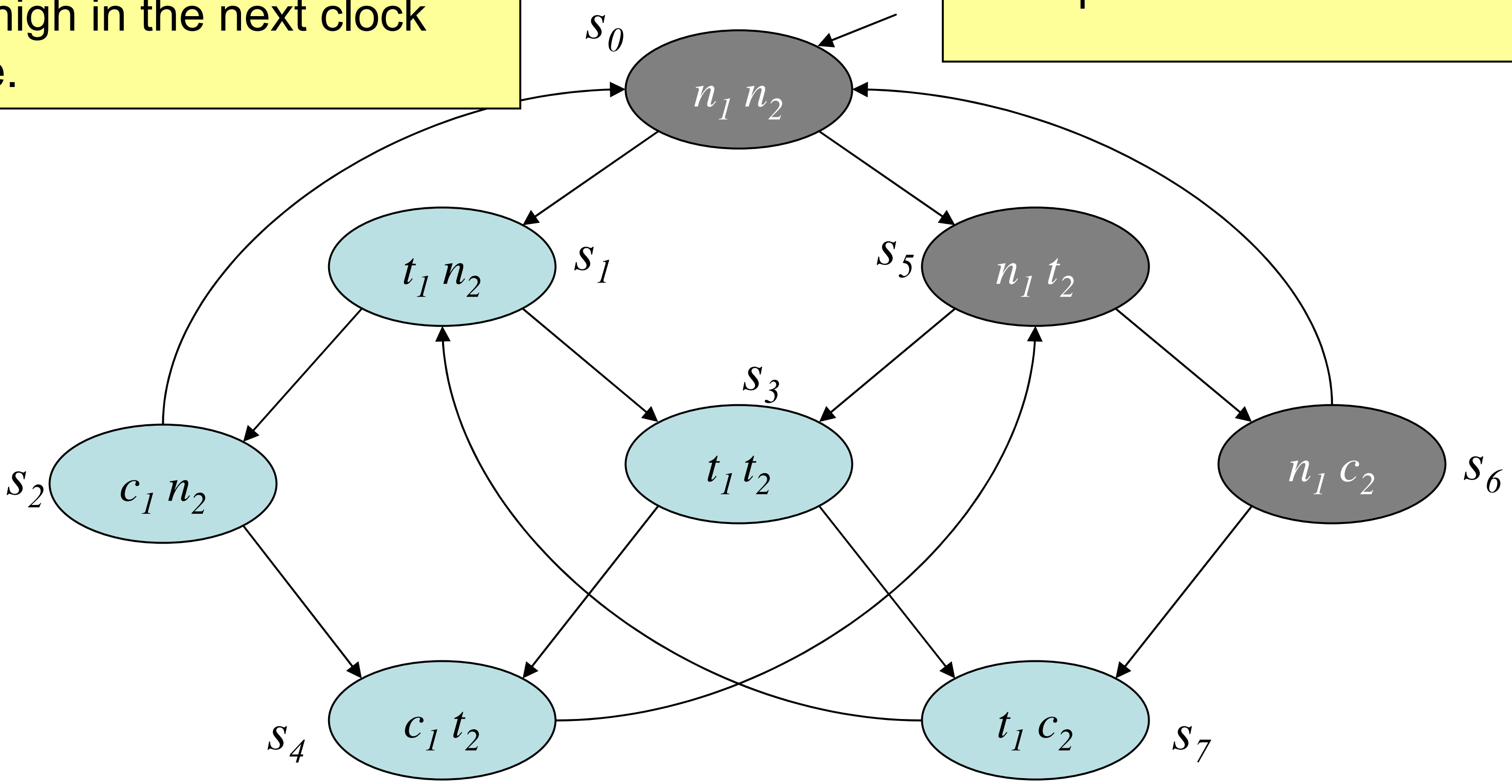
Live-ness property is violated. Follow the path marked in red. Processor 1 tries to enter the critical section but fails.



Evaluation of the Protocol

Non-blocking: Observe all states where n_1 is high.
All of them should
have at least one path where t_1 is high in the next clock cycle.

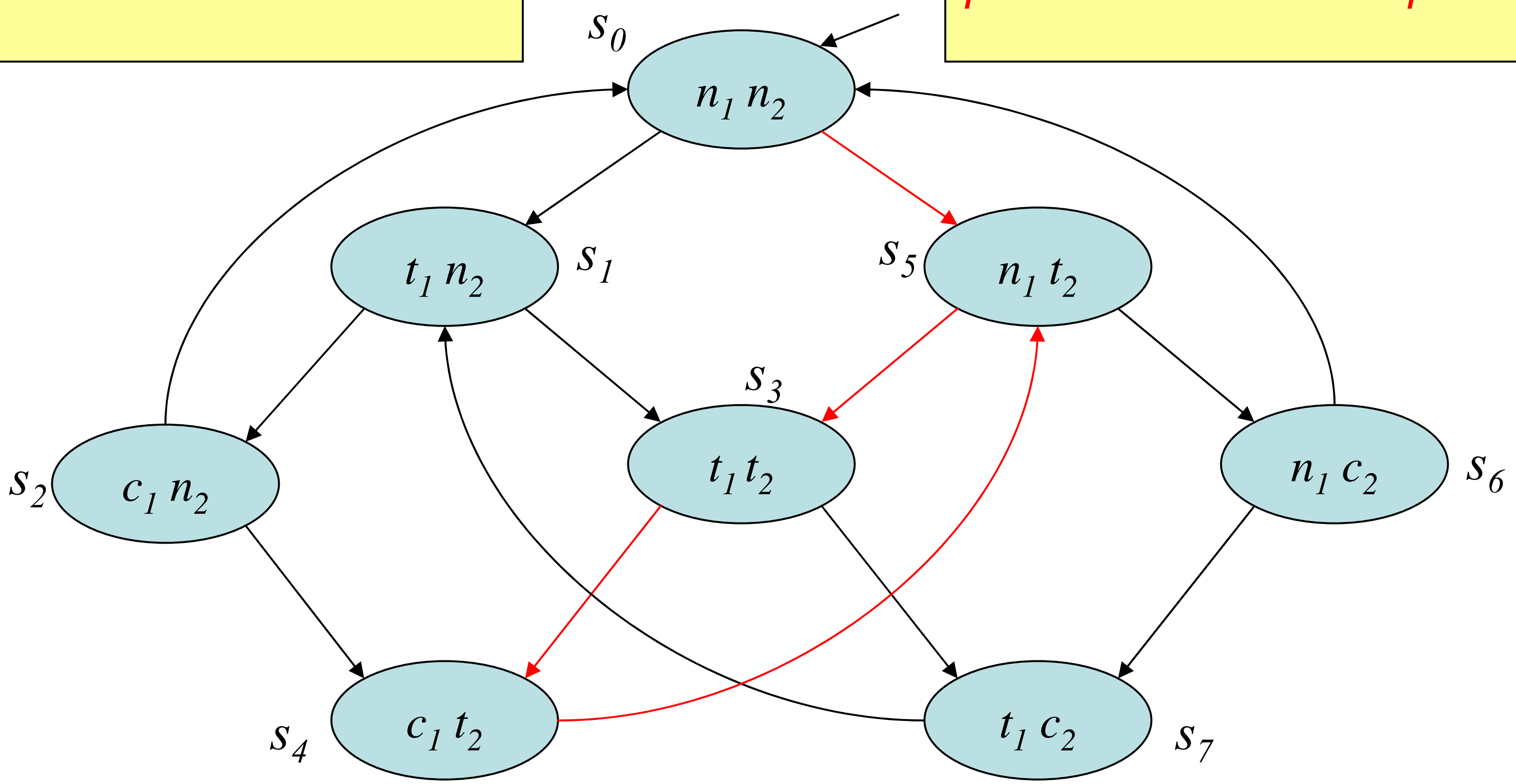
The property thus have to look for both for all and there exists logic and thus cannot be expressed in LTL.



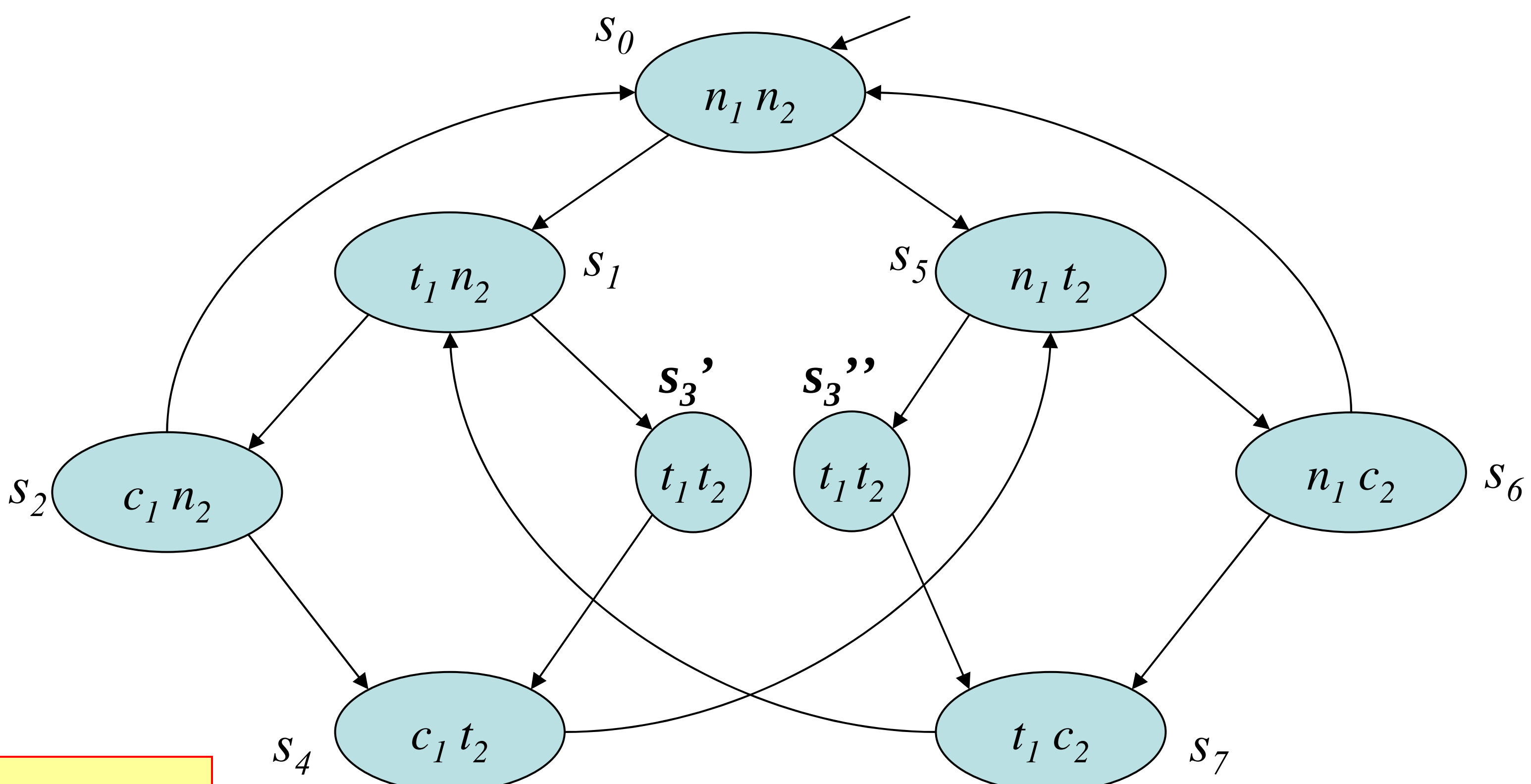
No-strict sequencing

Path marked in red shows that *“all paths are sequencing”* is false. Thus, no strict sequencing is maintained.

Note that since we are using LTL, we have negated the property: *“there exists a path with no strict sequencing”*



Solution



All the four properties are satisfied

s_3' and s_3'' now expresses which process was requesting for the critical section early.
Thus the live-ness problem is solved.

The SMV model checker

- New Symbolic Model Verifier
- Provides a language for describing the models.
- The properties are written as LTL (or CTL) formulas.
- It produces an output whether the specifications hold ‘true’, or a trace to show why the specification is false.