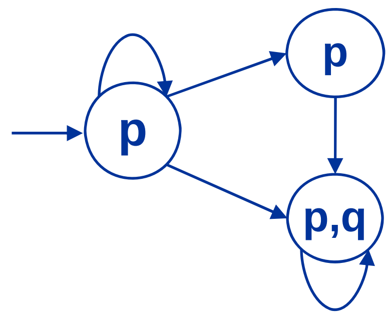
Linear Temporal Logic

Moonzoo Kim
CS Division of EECS Dept.
KAIST

KAIST

Review: Model checking

- Model checking
  - In a model-based approach, the system is represented by a model $\mathcal{M}$. The specification is again represented by a formula ϕ.
    - The verification consists of computing whether $\mathcal{M}$ satisfies ϕ $\mathcal{M} \vDash \phi$
      - Caution: $\mathcal{M} \vDash \phi$ represents satisfaction, not semantic entailment
- In model checking,
  - The model $\mathcal{M}$ is a transition systems and
  - the property ϕ is a formula in temporal logic
    - ex. □ p, □ q, ◇ q, □ ◇ q



Motivation for Temporal Logic

- So far, we have analyzed sequential programs only
  - `assert` is a convenient way of specify requirement properties
  - Safety properties are enough for sequential programs
    - “Bad thing never happens”
    - Ex. Mutual exclusion
- For concurrent programs, we need more than `assert` to specify important requirement properties conveniently
  - Liveness properties
    - “Good thing eventually happens”
    - Ex. Deadlock freedom
    - Ex. Starvation freedom
- Temporal logic is an adequate logic for describing requirement properties for concurrent system

Motivating Example (1/2)

Quoted from "The art of multiprocessor programing" by M.Herlihy et al, published by Morgan Kaufmann 2008

- Mutual exclusion protocol
  - Alice and Bob are neighbors, and they share a yard.
  - Alice owns a cat and Bob owns a dog.
  - Alice and Bob should coordinate that both pets are never in the yard at the same time.
- We would like to design a mutual exclusion protocol to satisfy
  1. Mutual exclusion
     - pets are excluded from being in the yard at the same time
  2. Deadlock-freedom
     - Both pets want to enter the yard, then eventually at leas one of them succeeds
  3. Starvation-freedom/lock-out freedom
     - If a pet wants to enter the yard, it will eventually succeed

KAIST

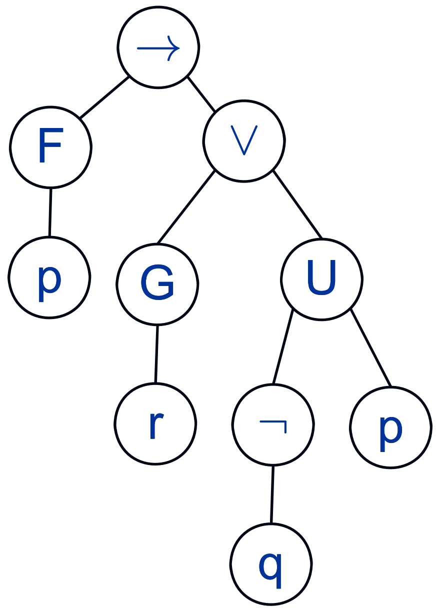
Motivating Example (2/2)

- One protocol design: Alice and Bob set up a flag pole at each house
  - Protocol @ Alice
    1. Alice raises her flag
    2. When Bob's flag is lowered, she unleashes her cat
    3. When her cat comes back, she lowers her flag
  - Protocol @ Bob
    1. He raises his flag
    2. While Alice's flag is raised
       1. Bob lowers his flag
       2. Bob waits until Alice's flag is lowered
       3. Bob raises his flag
    3. As soon as his flag is raised and hers is down, he unleashes his dog
    4. When his dog comes back, he lowers his flag

Linear time temporal logic (LTL)

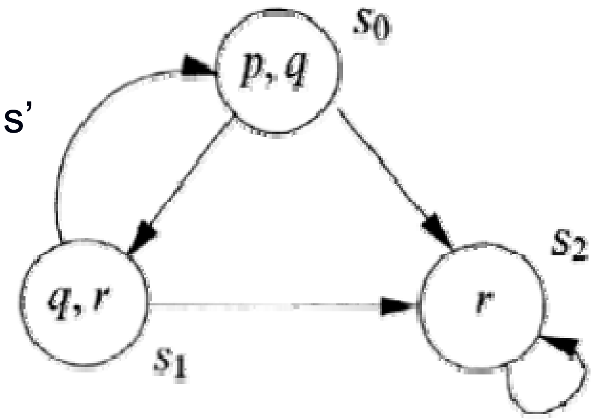
- LTL models time as a sequence of states, extending infinitely into the future
  - sometimes a sequence of states is called a computation path or an execution path, or simply a path
- Def 3.1 LTL has the following syntax
  - $\phi ::= \top \mid \bot \mid p \mid \neg \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi$
    $\mid X\, \phi \mid F\, \phi \mid G\, \phi \mid \phi\, U\, \phi \mid \phi\, W\, \phi \mid \phi\, R\, \phi$
    where p is any propositional atom from some set Atoms
  - Operator precedence
    - the unary connectives bind most tightly. Next in the order come U, R, W, $\wedge$, $\vee$, and $\rightarrow$

$\boxed{\mathbf{F\, p \rightarrow G\, r \vee \neg\, q\, U\, p}}$



Semantics of LTL (1/3)

- Def 3.4 A transition system (called model) $\mathcal{M}$ = (S, $\rightarrow$, L)
  - a set of states S
  - a transition relation $\rightarrow$ (a binary relation on S)
    - such that every s $\in$ S has some s' $\in$ S with s $\rightarrow$ s'
  - a labeling function L: S $\rightarrow \mathcal{P}$ (Atoms)

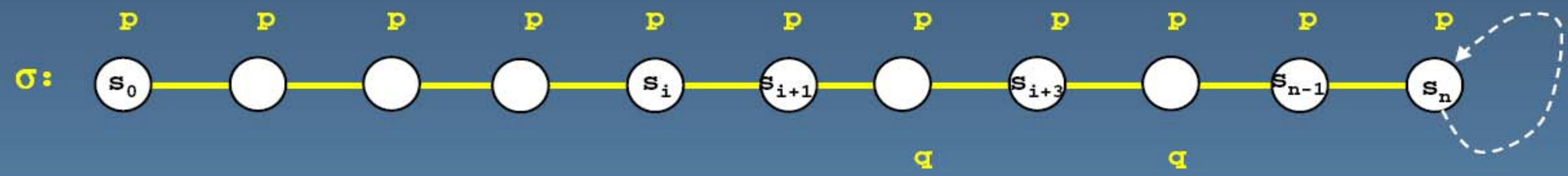


- Example
  - S={s_0,s_1,s_2}
  - $\rightarrow$={(s_0,s_1),(s_1,s_0),(s_1,s_2),(s_0,s_2),(s_2,s_2)}
  - L={(s_0,{p,q}),(s_1,{q,r}), (s_2,{r})}
- Def. 3.5 A path in a model $\mathcal{M}$ = (S, $\rightarrow$, L) is an infinite sequence of states s_{i_1}, s_{i_2}, s_{i_3},… in S s.t. for each j$\geq$ 1, $s_{i_j} \rightarrow s_{i_{j+1}}$. We write the path as $s_{i_1} \rightarrow s_{i_2} \rightarrow$ …
  - From now on if there is no confusion, we drop the subscript index i for the sake of simple description
- We write π^i for the suffix of a path starting at s_i.
  - ex. π^3 is $s_3 \rightarrow s_4 \rightarrow$ …

Semantics of LTL (2/3)

- Def 3.6 Let $\mathcal{M}$ = (S, $\rightarrow$, L) be a model and $\pi = s_1 \rightarrow \ldots$ be a path in $\mathcal{M}$. Whether π satisfies an LTL formula is defined by the satisfaction relation $\vDash$ as follows:
  - Basics: $\pi \vDash \top$, $\pi \nvDash \bot$, $\pi \vDash p$ iff $p \in L(s_1)$, $\pi \vDash \neg\phi$ iff $\pi \nvDash \phi$
  - Boolean operators: $\pi \vDash p \wedge q$ iff $\pi \vDash p$ and $\pi \vDash q$
    - similar for other boolean binary operators
  - $\pi \vDash$ X ϕ iff $\pi^2 \vDash \phi$ (next $\bigcirc$)
  - $\pi \vDash$ G ϕ iff for all i $\geq$ 1, $\pi^i \vDash \phi$ (always $\Box$)
  - $\pi \vDash$ F ϕ iff there is some i $\geq$ 1, $\pi^i \vDash \phi$ (eventually $\Diamond$)
  - $\pi \vDash \phi$ U ψ iff there is some i $\geq$ 1s.t. $\pi^i \vDash \psi$ and for all j=1,…,i-1 we have $\pi^j \vDash \phi$ (strong until)
  - $\pi \vDash \phi$ W ψ iff either (weak until)
    - either there is some i $\geq$ 1 s.t. $\pi^i \vDash \psi$ and for all j=1,…,i-1 we have $\pi^j \vDash \phi$
    - or for all k $\geq$ 1 we have $\pi^k \vDash \phi$
  - $\pi \vDash \phi$ R ψ iff either (release)
    - either there is some i $\geq$ 1 s.t. $\pi^i \vDash \phi$ and for all j=1,…,i we have $\pi^j \vDash \psi$
    - or for all k $\geq$ 1 we have $\pi^k \vDash \psi$

examples



[]p is satified at all locations in σ

<>p is satisfied at all locations in σ

[]<>p is satisfied at all locations in σ

<>q is satisfied at all locations except s_{n-1} and s_n

Xq is satisfied at s_{i+1} and at s_{i+3}

pUq (strong until) is satisfied at all locations except s_{n-1} and s_n

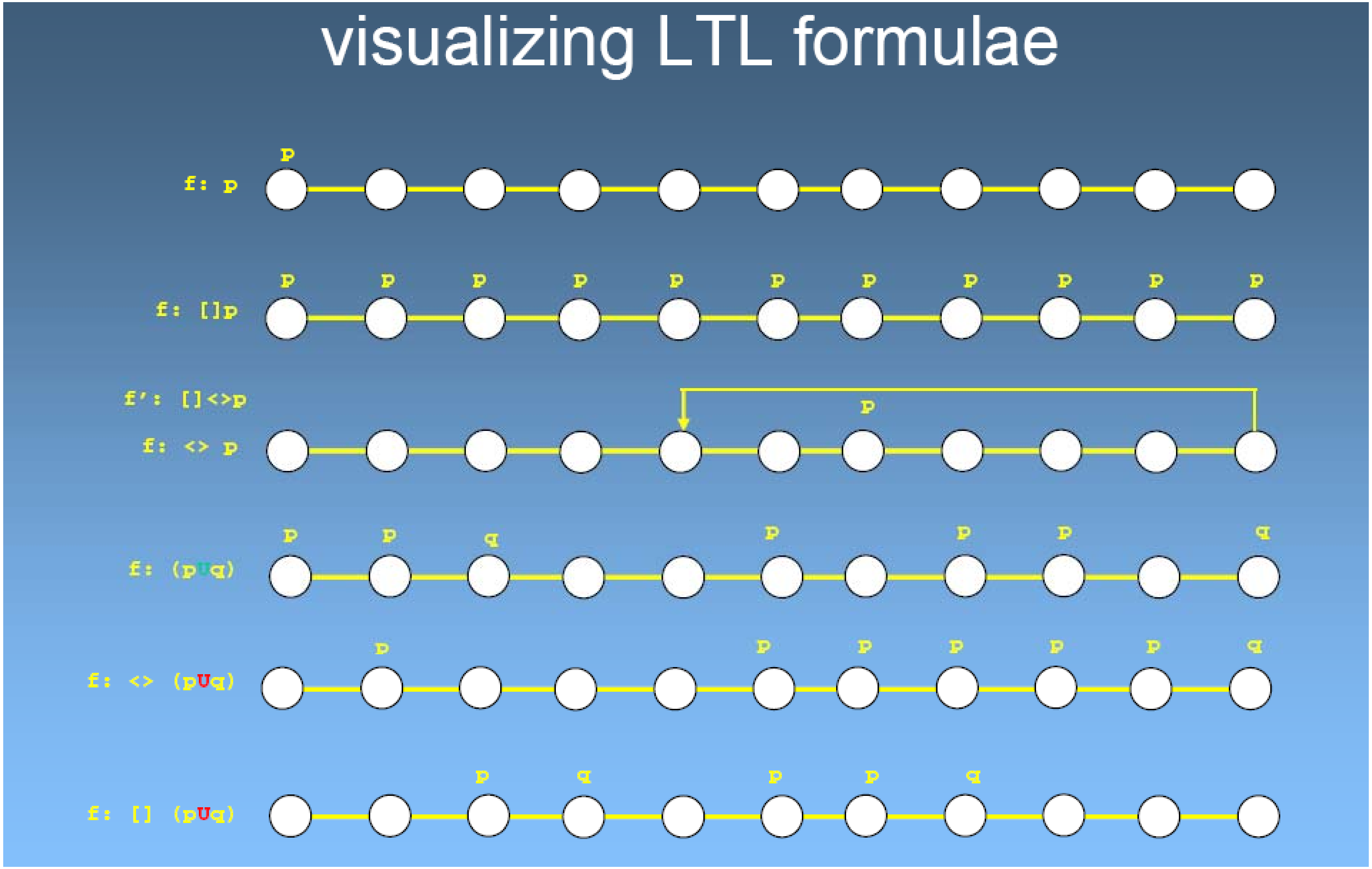
<>(pUq) (strong until) is satisfied at all locations except s_{n-1} and s_n

<>(pUq) (weak until) is satisfied at all locations

[]<>(pUq) (weak until) is satisfied at all locations

in model checking we are typically only interested in whether a temporal logic formula is satisfied for all runs of the system, starting in the initial system state (that is: at s_0)

slide quoted from Caltech 101b.2 “Logic Model Checking” by Dr.G.Holzmann



slide quoted from Caltech 101b.2 "Logic Model Checking" by Dr.G.Holzmann

interpreting formulae...

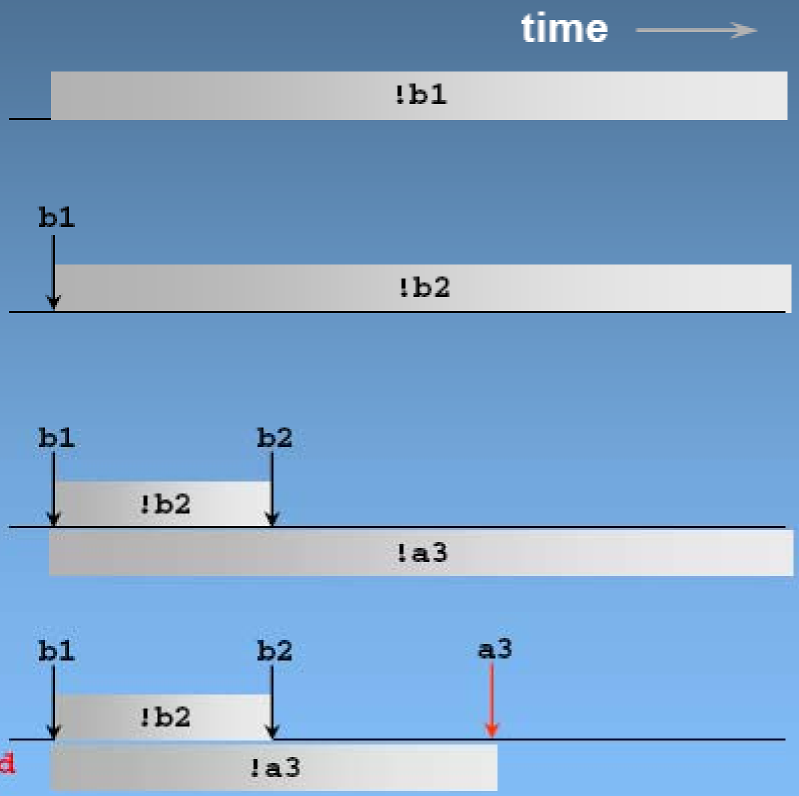
`LTL: (<>(b1 && (!b2 U b2))) -> []!a3`

1. suppose b1 never becomes true
   `(p->q) means (!p v q)`
   `the formula is` *`satisfied!`*

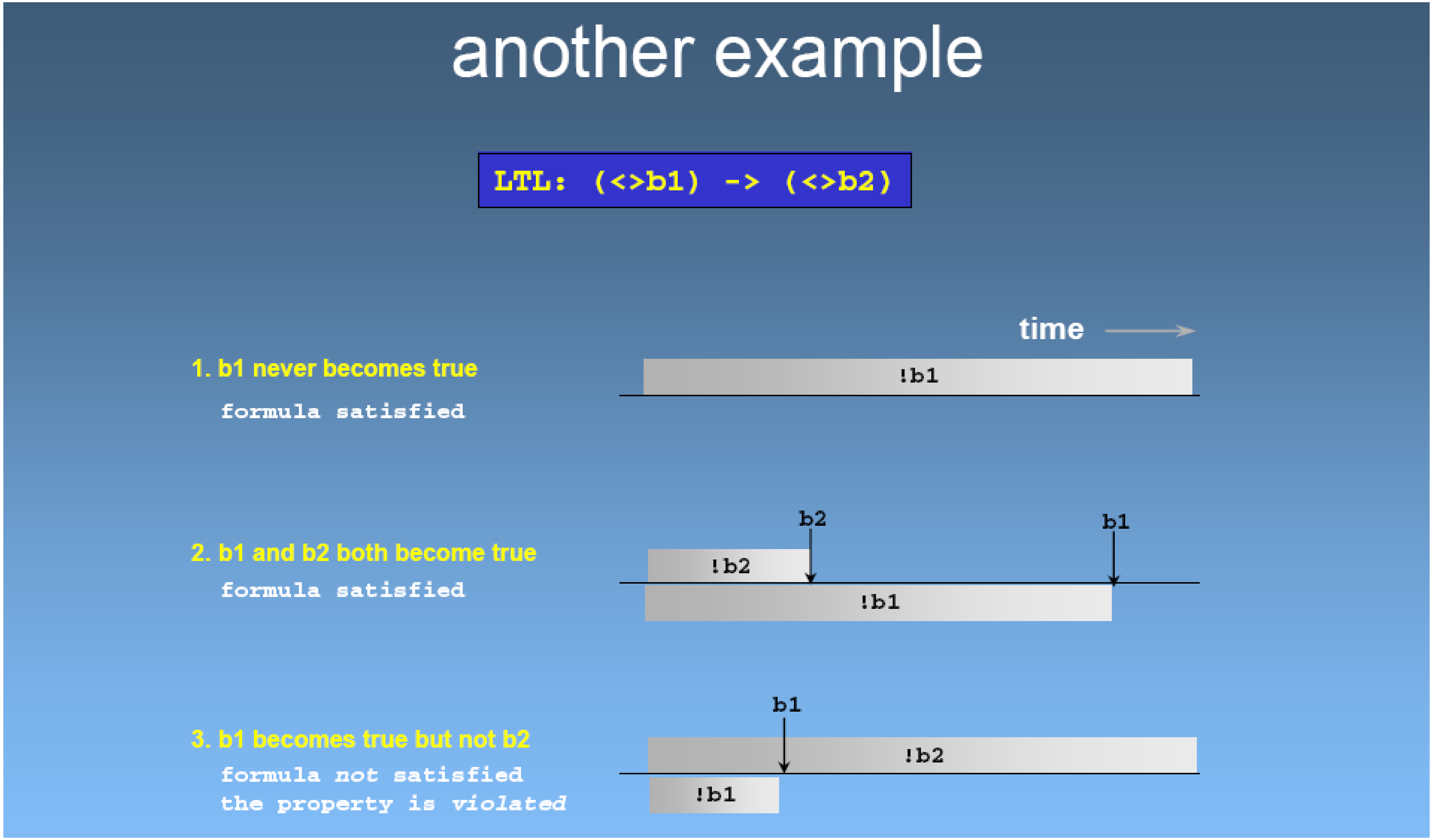
2. b1 becomes true, `but not b2`
   `the formula is` *`satisfied!`*

3. b1 becomes true, then b2
   `but not a3`
   `the formula is` *`satisfied`*

4. b1 becomes true, then b2, then a3
   `the formula is` *`not satisfied`*
   `i.e., the property is violated`



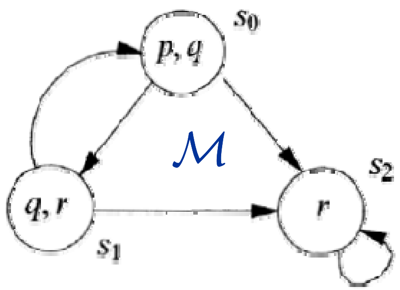
slide quoted from Caltech 101b.2 “Logic Model Checking” by Dr.G.Holzmann



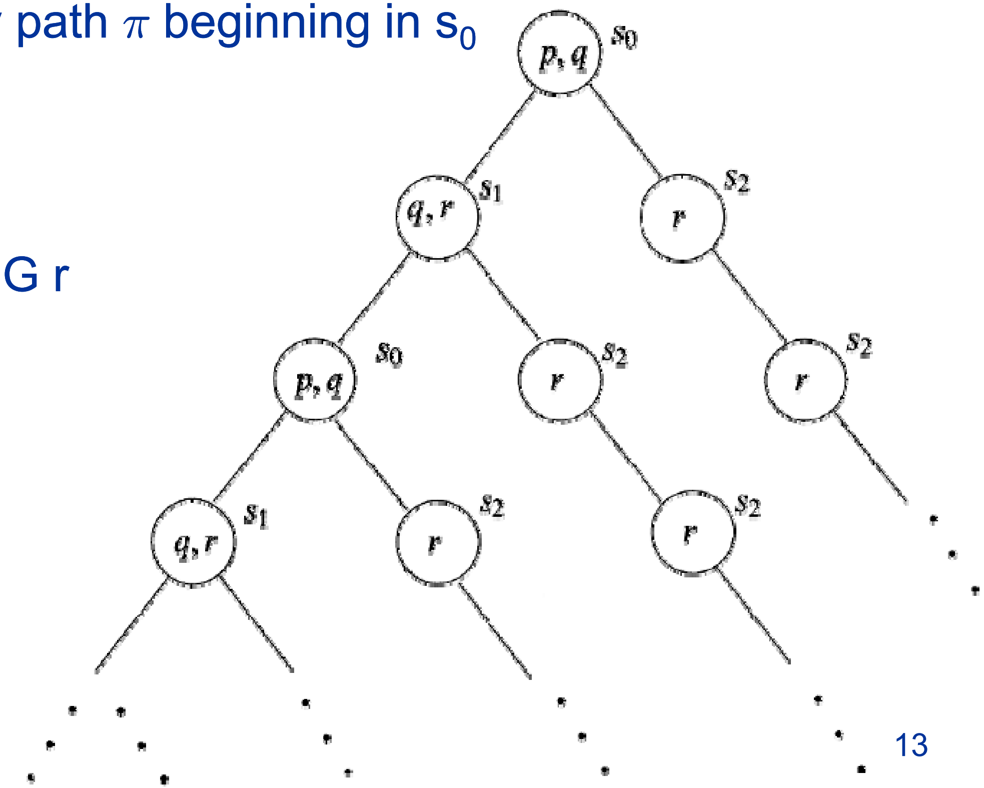
slide quoted from Caltech 101b.2 “Logic Model Checking” by Dr.G.Holzmann

Semantics of LTL (3/3)

- Def 3.8 Suppose $\mathcal{M}$ = (S, $\rightarrow$, L) is a model, s $\in$ S, and ϕ an LTL formula. We write $\mathcal{M}$,s $\vDash \phi$ if for every execution path π of $\mathcal{M}$ starting at s, we have $\pi \vDash \phi$
  - If $\mathcal{M}$ is clear from the context, we write s $\vDash \phi$



- Example
  - $s_0 \vDash p \wedge q$ since $\pi \vDash p \wedge q$ for every path π beginning in s_0
  - $s_0 \vDash \neg r$, $s_0 \vDash \top$
  - $s_0 \vDash X\, r$, $s_0 \nvDash X\, (q \wedge r)$
  - $s_0 \vDash G\, \neg(p \wedge r)$, $s_2 \vDash G\, r$
  - For any s of $\mathcal{M}$, $s \vDash F(\neg q \wedge r) \rightarrow F\, G\, r$
    - Note that s_2 satisfies $\neg q \wedge r$
  - $s_0 \nvDash G\, F\, p$
    - $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \ldots \vDash G\, F\, p$
    - $s_0 \rightarrow s_2 \rightarrow s_2 \rightarrow s_2 \ldots \nvDash G\, F\, p$
  - $s_0 \vDash G\, F\, p \rightarrow G\, F\, r$
  - $s_0 \nvDash G\, F\, r \rightarrow G\, F\, p$



Practical patterns of specification

- For any state, if a request occurs, then it will eventually be acknowledge
  - $G(\text{requested} \rightarrow F\ \text{acknowledged})$
- A certain process is enabled infinitely often on every computation path
  - G F enabled
- Whatever happens, a certain process will eventually be permanently deadlocked
  - F G deadlock
- If the process is enabled infinitely often, then it runs infinitely often
  - G F enabled $\rightarrow$ G F running
- An upwards traveling lift at the second floor does not change its direction when it has passengers wishing to go to the fifth floor
  - G (fllor2 $\wedge$ directionup $\wedge$ ButtonPressed5 $\rightarrow$ (directionup U floor5)
- It is impossible to get to a state where a system has started but is not ready
  - ϕ = G $\neg$(started $\wedge$ $\neg$ready)
  - What is the meaning of (intuitive) negation of ϕ ?
    - For every path, it is possible to get to such a state (started$\wedge\neg$ready).
    - There exists a such path that gets to such a state.
      - we cannot express this meaning directly
- LTL has limited expressive power
  - For example, LTL cannot express statements which assert the existence of a path
    - From any state s, there exists a path π starting from s to get to a restart state
    - The lift can remain idle on the third floor with its doors closed
  - Computation Tree Logic (CTL) has operators for quantifying over paths and can express these properties

KAIST

Summary of practical patterns

G p	always p	invariance
F p	eventually p	guarantee
$p \rightarrow (F\ q)$	p implies eventually q	response
$p \rightarrow (q\ U\ r)$	p implies q until r	precedence
G F p	always, eventually p	recurrence (progress)
F G p	eventually, always p	stability (non-progress)
$F\ p \rightarrow F\ q$	eventually p implies eventually q	correlation

Equivalences between LTL formulas

- Def 3.9 $\phi \equiv \psi$ if for all models $\mathcal{M}$ and all paths π in $\mathcal{M}$: $\pi \vDash \phi$ iff $\pi \vDash \psi$
- $\neg G\ \phi \equiv F\ \neg\phi$, $\neg F\ \phi \equiv G\ \neg\phi$, $\neg X\ \phi \equiv X\ \neg\phi$
- $\neg\ (\phi\ U\ \psi) \equiv \neg\phi\ R\ \neg\psi$, $\neg(\phi\ R\ \psi) \equiv \neg\phi\ U\ \neg\psi$
- $F\ (\phi \vee \psi) \equiv F\ \phi \vee F\ \psi$
- $G\ (\phi \wedge \psi) \equiv G\ \phi \wedge G\ \psi$
- $F\ \phi \equiv T\ U\ \phi$, $G\ \phi \equiv \perp R\ \phi$
- $\phi\ U\ \psi \equiv \phi\ W\ \psi \wedge F\ \psi$
- $\phi\ W\ \psi \equiv \phi\ U\ \psi \vee G\ \phi$
- $\phi\ W\ \psi \equiv \psi\ R\ (\phi \vee \psi)$
- $\phi\ R\ \psi \equiv \psi\ W\ (\phi \wedge \psi)$

KAIST

Adequate sets of connectives for LTL (1/2)

- X is completely orthogonal to the other connectives
  - X does not help in defining any of the other connectives.
  - The other way is neither possible
- Each of the sets {U,X}, {R,x}, {W,X} is adequate
  - {U,X}
    - ϕ R $\psi \equiv \neg$ ($\neg\, \phi$ U $\neg\, \psi$)
    - ϕ W $\psi \equiv \psi$ R ($\phi \vee \psi$) $\equiv \neg$ ($\neg\psi$ U $\neg$($\phi \vee \psi$))
  - {R,X}
    - ϕ U $\psi \equiv \neg$ ($\neg\phi$ R $\neg\psi$)
    - ϕ W $\psi \equiv \psi$ R ($\phi \vee \psi$)
  - {W,X}
    - ϕ U $\psi \equiv \neg$ ($\neg\, \phi$ R $\neg\, \psi$)
    - ϕ R $\psi \equiv \psi$ W ($\phi \wedge \psi$)

Adequate sets of connectives for LTL (2/2)

- Thm 4.10 $\phi \mathrm{U} \psi \equiv \neg(\neg\psi \mathrm{U} (\neg\phi \wedge \neg\psi)) \wedge \mathrm{F} \psi$
- Proof: take any path $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ in any model
  - Suppose $s_0 \vDash \phi \mathrm{U} \psi$
    - Let n be the smallest number s.t. $s_n \vDash \psi$
      - We know that such n exists from $\phi \mathrm{U} \psi$. Thus, $s_0 \vDash \mathrm{F} \psi$
      - For each $k < n$, $s_k \vDash \phi$ since $\phi \mathrm{U} \psi$
    - We need to show $s_0 \vDash \neg(\neg\psi \mathrm{U} (\neg\phi \wedge \neg\psi))$
      - case 1: for all i, $s_i \nvDash \neg\phi \wedge \neg\psi$. Then, $s_0 \vDash \neg(\neg\psi \mathrm{U} (\neg\phi \wedge \neg\psi))$
      - case 2: for some i, $s_i \vDash \neg\phi \wedge \neg\psi$. Then, we need to show
        - (*)for each $i > 0$, if $s_i \vDash \neg\phi \wedge \neg\psi$, then there is some $j < i$ with $s_j \nvDash \neg\psi$ (i.e. $s_j \vDash \psi$)
        - Take any $i > 0$ with $s_i \vDash \neg\phi \wedge \neg\psi$. We know that $i > n$ since $s_0 \vDash \phi \mathrm{U} \psi$. So we can take $j=n$ and have $s_j \vDash \psi$
  - Conversely, suppose $s_0 \vDash \neg(\neg\psi \mathrm{U} (\neg\phi \wedge \neg\psi)) \wedge \mathrm{F} \psi$
    - Since $s_0 \vDash \mathrm{F} \psi$, we have a minimal n as before s.t. $s_n \vDash \psi$
      - case 1: for all i, $s_i \nvDash \neg\phi \wedge \neg\psi$ (i.e. $s_i \vDash \phi \vee \psi$). Then $s_0 \vDash \phi \mathrm{U} \psi$
      - case 2: for some i, $s_i \vDash \neg\phi \wedge \neg\psi$. We need to prove for any $i < n$, $s_i \vDash \phi$
        - Suppose $s_i \nvDash \phi$ (i.e., $s_i \vDash \neg\phi$). Since n is minimal, we know $s_i \vDash \neg\psi$. So by (*) there is some j $<i<n$ with $s_j \vDash \psi$, contradicting the minimality of n. Contradiction

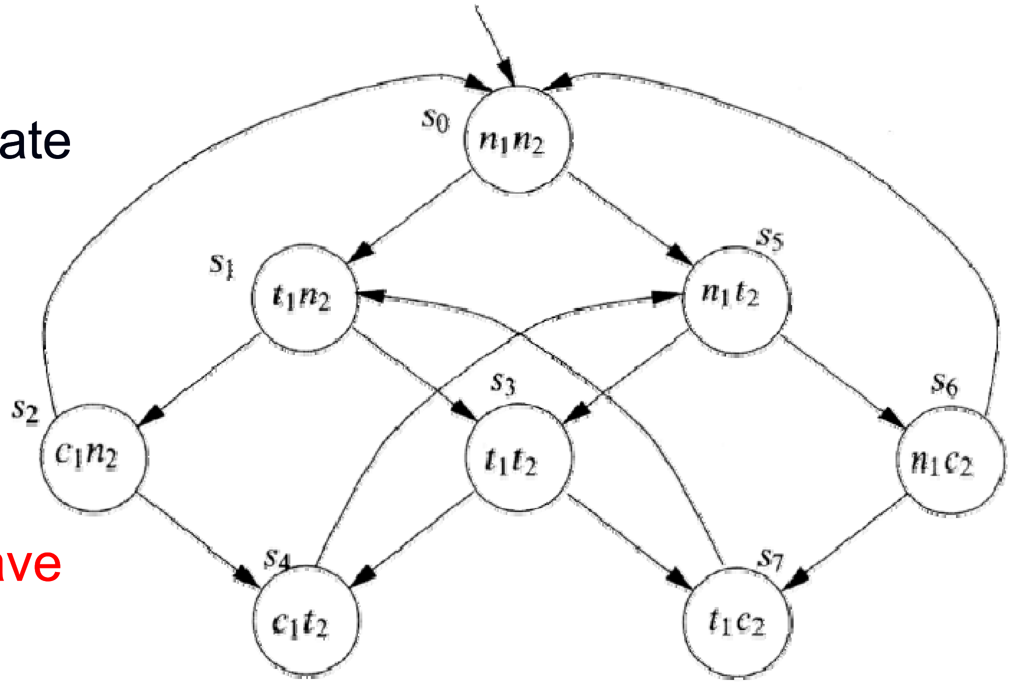
KAIST

Mutual exclusion example

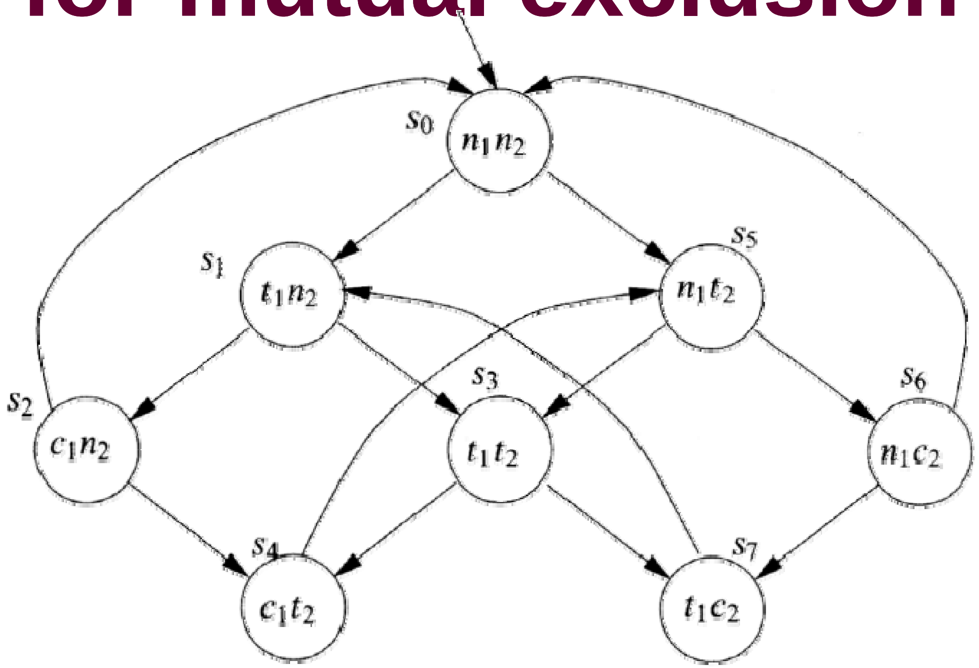
- When concurrent processes share a resource, it may be necessary to ensure that they do not have access to the common resource at the same time
  - We need to build a protocol which allows only one process to enter critical section
- Requirement properties
  - Safety:
    - Only one process is in its critical section at anytime
  - 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

1^{st} model

- We model two processes
  - each of which is in
    - non-critical state (n) or
    - trying to enter its critical state (t) or
    - critical section (c)
  - No self edges
- each process executes like n→ t → c → n → …
  - but the two processes interleave with each other
    - only one of the two processes can make a transition at a time (asynchronous interleaving)



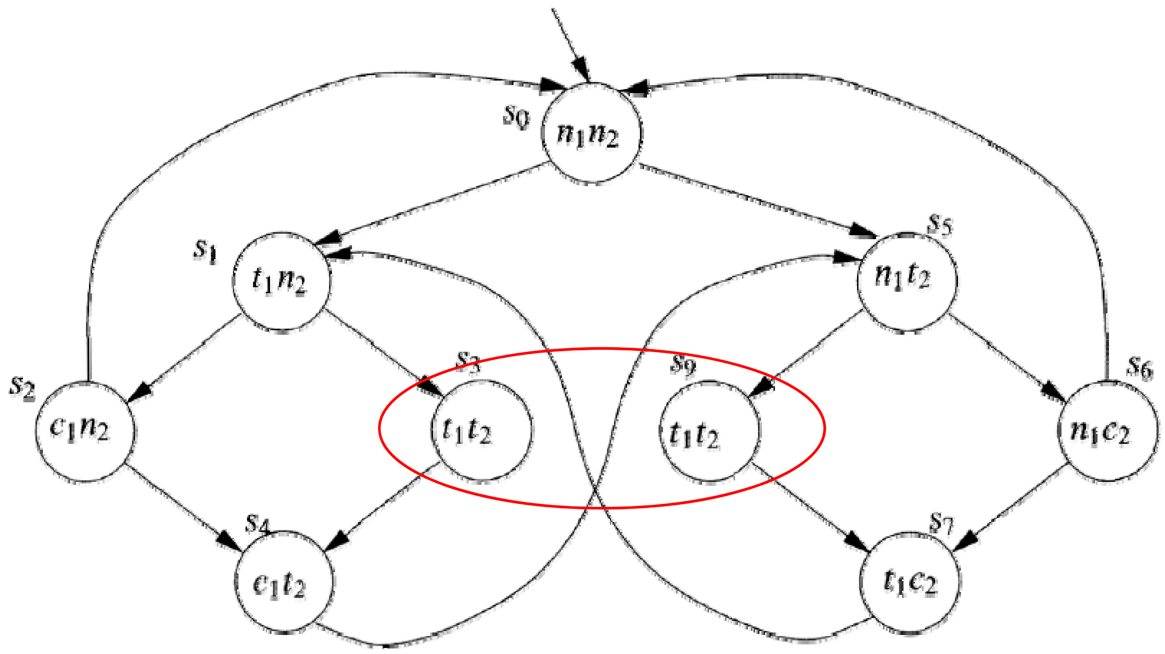
1^{st} model for mutual exclusion



- Safety: $s_0 \models G \neg (c_1 \wedge c_2)$
- Liveness $s_0 \not\models G(t_1 \rightarrow F\, c_1)$
  - see $s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \ldots$
- Non-blocking
  - for every state satisfying n_i, there is a successor satisfying t_i
    - s_0 satisfies this property
  - We cannot express this property in LTL but in CTL
    - Note that LTL specifies that ϕ is satisfied for all paths
- No strict ordering
  - there is a path where c_1 and c_2 do not occur in strict order
  - Complement of this is
    - $G(c_1 \rightarrow c_1\, W\, (\neg c_1 \wedge \underline{\neg c_1\, W\, c_2}))$
    - anytime we get into a c_1 state, either that condition persists indefinitely, or it ends with a non-c_1 state and in that case there is <u>no further c_1 state</u> unless and until we obtain a <u>c_2</u> state

2nd model for mutual exclusion

- All 4 properties are satisfied
  - Safety
  - Liveness
  - Non-blocking
  - No strict sequencing



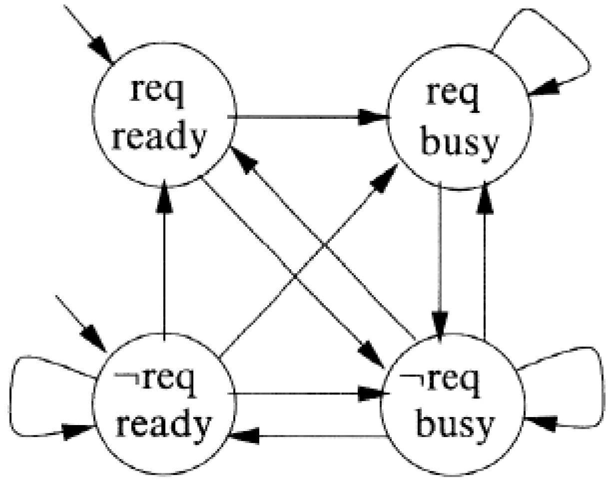
NuSMV model checker

- NuSMV programs consist of one or more modules.
  - one of the modules must be called main
- Modules can declare variables and assign to them.
- Assignments usually give the initial value of a variable x (init(x)) and its next value (next(x)) as an expression in terms of the current values of variables.
  - this expression can be non-deterministic
    - denoted by several expressions in braces, or no assignment at all

Example

```
MODULE main
VAR
  request: boolean;
  status: {ready,busy};
ASSIGN
  init(status) := ready;
  next(status) := case
                    request : busy;
                    1: {ready,busy};
                    esac;
LTLSPEC
  G(request -> F status=busy)
```

- request is under-specified, i.e., not controlled by the program
  - request is determined (randomly) by external environment
  - thus, whole program works non-deterministically
- Case statement is evaluated top-to-bottom



Modules in NuSMV

- A module is instantiated when a variable having that module name as its type is declared.
- A 3 bit counter increases from 000 to 111 repeatedly
  - Req. property
    - infinitely setting carry-out of most significant bit as 1
- By default, modules in NuSMV are composed synchronously
  - there is a global clock and, each time it ticks, each of the modules executes in parallel
  - By use of the 'process' keyword, it is possible to compose the modules asynchronously

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
   G  F bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := (value + carry_in) mod 2;
DEFINE
  carry_out := value & carry_in;
```

KAIST