Logic in Computer Science

Lecture 06

Verification by Model Checking

时态 logic
霍尔 logic
} 验证程序正确性

Li Qin

Associate Professor

School of Software Engineering

★ Motivation for verification

★ Computation Tree Logic —syntax and semantics

★ Example: mutual exclusion

★ A model checking algorithm

Verification methods may be classified according to the following main criteria:

- ***Proof-based vs. model-based*** - if a soundness and completeness theorem holds, then:

    – proof = valid formula = true in all models;

    – model-based = check satisfiability in one model

- ***Degree of automation*** - fully automated, partially automated, or manual

- ***Full- vs. property-verification*** - a single property vs. full behavior

- ***Domain of application*** - hardware or software; sequential or concurrent; reactive or terminating; etc.

- ***Pre- vs. post-development***

(handwritten annotations:)
霍尔逻辑
proof = ...
model-based 证明全自动
但建立 model 和 语句书写不自动
霍尔逻辑
model-based
霍尔logic   model - based
model-based 和霍尔逻辑都可以
在这些 domain 下证明

model checking

$$M \models \phi$$

基于语义

Model Checking is a verification method that is:

- model-based, automated, using a property-verification approach, mainly useful to verifying concurrent programs and reactive systems, typically in a post-development stage.

霍尔逻辑

Program Verification (to be studied later), is:

- proof based, computer-assisted (partially-automated), mainly used for sequential, terminating programs

Model: 论域D, 函数F: Dⁿ→D, 谓词p: Dⁿ

- Classical propositional and predicate calculi use a *unique* universe for interpreting formulas.

- In the 1950s Kripke introduced a type of semantic models where more (local) universes are possible

- There is a relation of *accessibility* between these universes and operators to express relationships between such universes, leading to various kinds of *modalities*.

- When such operators are added, one gets *modal logics*. When *time* is the parameter that causes the passing from one universe to another, one speaks about *temporal logics*.

Programs (software) fit well in this framework:

- a universe corresponds to a state;

- the accessibility relation is given by the transition from one state to another;

- classic predicate logic may be used to specify relationships between variables in a state.

At this point we are lacking a mechanism to relate these universes (states). A variety of such mechanisms shall be introduced throughout this course.

We have the following characterizations of time:

- *linear* —a chain of time instances, or

- *branching* —several alternative future worlds may be possible at a given point in time ;

or

- *discrete* —the time is represented by the set of integers, or

- *continuous* —time is represented by the set of real numbers.

Next, we shall study *Computation Tree Logic (CTL)* which is a type of temporal logic using branching and discrete time.

LTL : linear temporal logic.

For any ~~model checking~~ problem (based on CTL, or other logic), we are required to answer the question of whether

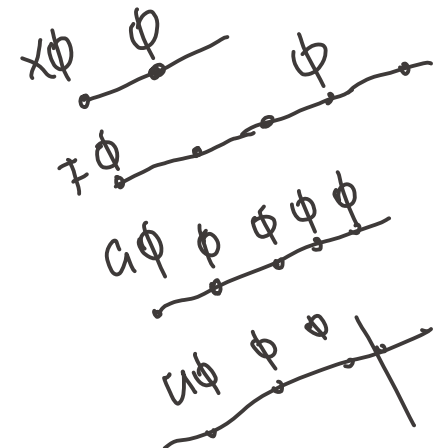$$\mathcal{M}, s \models \Phi \; ?$$

where

- $\mathcal{M}$ is an appropriate model for the given system, and s is a state of the model;

- Φ is a CTL formula intended to be satisfied by the system.

BNF definition of CTL:

$$\phi ::= \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi)$$
$$\mid \mathrm{AX}\phi \mid \mathrm{EX}\phi \mid \mathrm{A}[\phi\mathrm{U}\phi] \mid \mathrm{E}[\phi\mathrm{U}\phi] \mid \mathrm{AG}\phi \mid \mathrm{EG}\phi \mid \mathrm{AF}\phi \mid \mathrm{EF}\phi$$

The new connectives AX, EX, AU, EU, AG, EG, AF, and EF are called *temporal connectives*.

Depth and Breadth Quantification

The temporal connectives use two letters:

- A and E to quantify over the breadth in a branching point:

  – *All alternatives* in a branching point;

  – there *Exists at least one alternative* in a branching point

- G and F to quantify along the paths:

  – all future states on a path, *Globally*;

  – there exists at least one *Future state* along the path

Two more operators expressing properties along the paths are used:

- X to refer to the *neXt state* in the path (this leads to the discrete feature of the time), and

- U —the *Until* operator.

Convention:

- the unary connectives (including AX, EX, AG, EG, AF, and EF) bind most tightly;

- next come ∧ and ∨;

- lowest priority →, AU and EU.

1 $\text{EG } r$

2 $\text{AG}\,(q \to \text{EG}\, r)$

3 $\text{A}\,[r \text{ U } q]$

4 $\text{EF E}[r \text{ U } q]$

5 $\text{A}\,[p \text{ U EF } r]$

6 $\text{EF EG}p \to \text{AF } r$
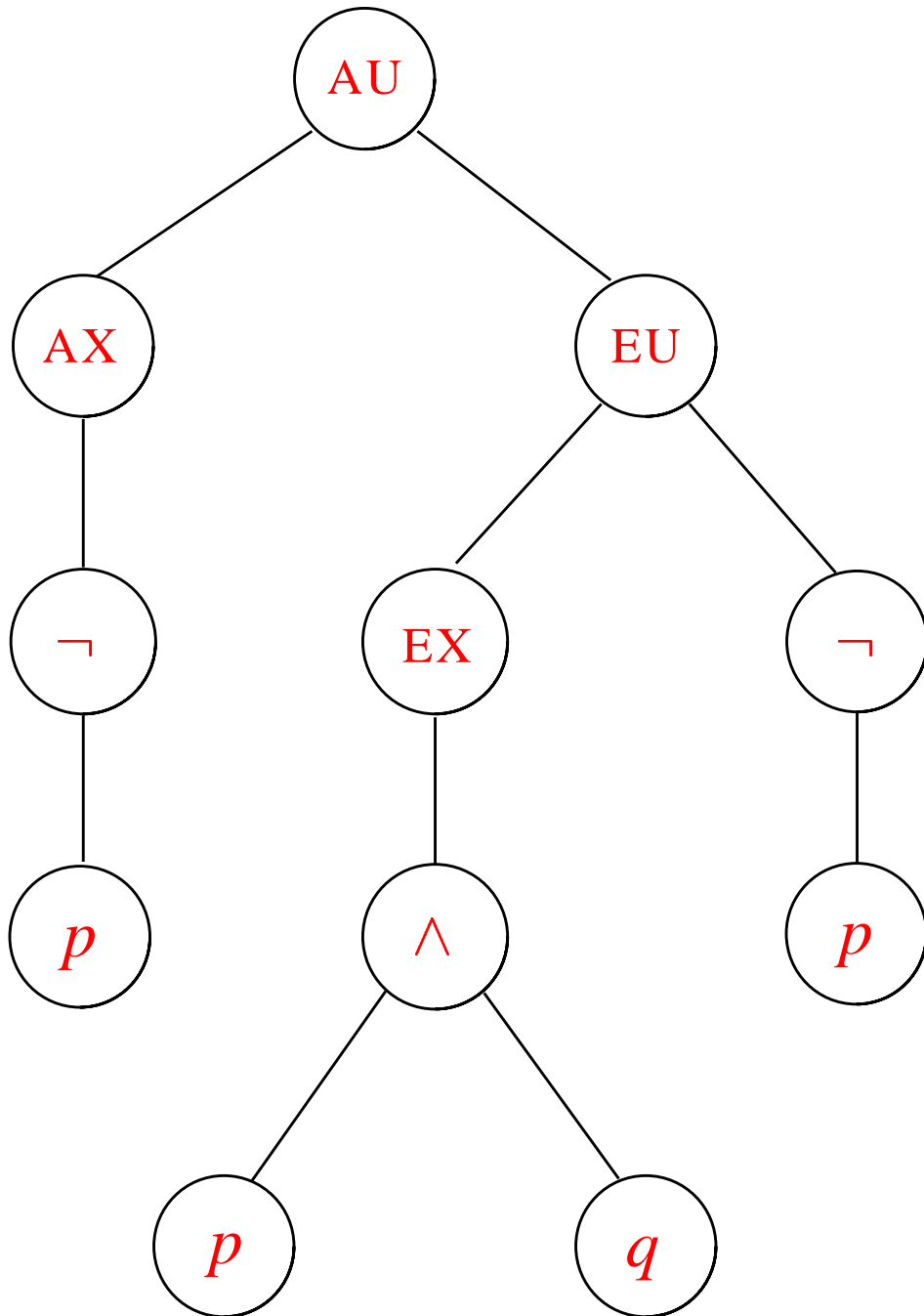
7 $\text{AG AF } r$

8 $\text{A}\,[p_1 \text{ U A}[p_2 \text{ U } p_3]]$

9 $\text{E}\,[\text{A}[p_1 \text{ U } p_2] \text{ U } p_3]$

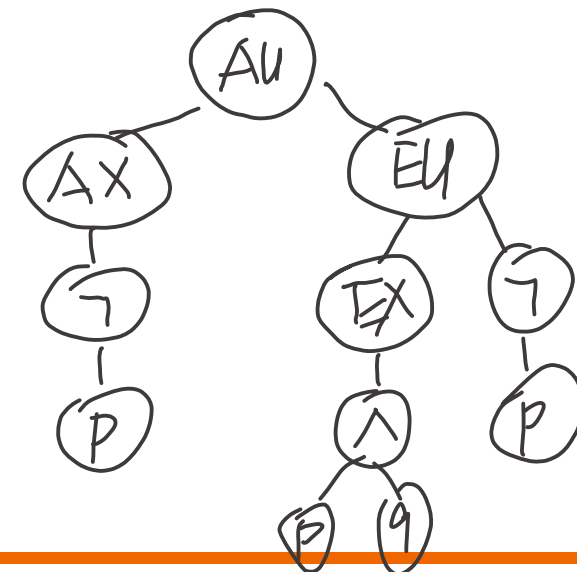10 $\text{AG}(p \to \text{A}\,[p \text{ U }(\neg p \wedge \text{A}\,[\neg p \text{ U } q])])$

The parse tree of the formula

$$\mathrm{A}\big[\mathrm{AX}\,\neg p\;\mathrm{U}\;\mathrm{E}[\mathrm{EX}(p \wedge q)\;\mathrm{U}\,\neg p]\big]$$

状态迁移系统 Model (state, →)

predicate logic model 的子集

A **_model_** $\mathcal{M} = (S, \to, L)$ for CTL consists of

$M \models \phi$ 在谓词logic中不可判定
在时态logic中可判定. NP

- a set of states S

- a binary relation $\to$ on S such that for every $s \in S$ there exists $s' \in S$ with $s \to s'$
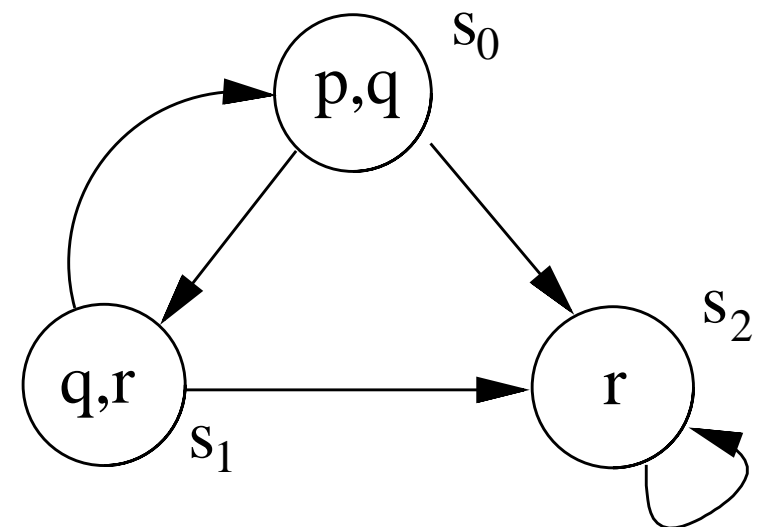
- a labeling function $L : S \to \mathcal{P}(Atoms)$

The intuition is that L says which atoms are true in a state and $\to$ describes how the systems move from state to state.

Graphical description:

$S = \{s_0, s_1, s_2\}$
$\to = \{(s_0, s_1), (s_0, s_2), (s_1, s_0), (s_1, s_2), (s_2, s_2)\}$
$L(s_0) = \{p, q\}, L(s_1) = \{q, r\}, L(s_2) = \{r\}$

Let $\mathcal{M} = (S, \rightarrow, L)$ be a model for CTL, $s \in S$, and ϕ a CTL formula. The *satisfaction relation*

$$\mathcal{M}, s \models \phi$$

is inductively defined by

1. $\mathcal{M}, s \models \top$ and $\mathcal{M}, s \not\models \bot$ for all $s \in S$;

2. $\mathcal{M}, s \models p$ iff $p \in L(s)$;

3. $\mathcal{M}, s \models \neg\phi$ iff $\mathcal{M}, s \not\models \phi$;

4. $\mathcal{M}, s \models \phi \wedge \psi$ iff $\mathcal{M}, s \models \phi$ and $\mathcal{M}, s \models \psi$;

5. $\mathcal{M}, s \models \phi \vee \psi$ iff $\mathcal{M}, s \models \phi$ or $\mathcal{M}, s \models \psi$;

6. $\mathcal{M}, s \models \phi \rightarrow \psi$ iff $\mathcal{M}, s \not\models \phi$ or $\mathcal{M}, s \models \psi$;

7. $\mathcal{M}, s \models \mathrm{AX}\, \phi$ iff for all s' such that $s \rightarrow s'$ we have $\mathcal{M}, s' \models \phi$;

8. $\mathcal{M}, s \models \mathrm{EX}\, \phi$ iff for some s' such that $s \rightarrow s'$ we have $\mathcal{M}, s' \models \phi$;

$$s_i = S$$

9. $\mathcal{M}, s \models AG\ \phi$ iff for **all paths** $p = s_1 \to s_2 \to \dots$ we have $\mathcal{M}, s_i \models \phi$, for **all** i;

10. $\mathcal{M}, s \models EG\ \phi$ iff **there exists a path** $p = s_1 \to s_2 \to \dots$ such that $\mathcal{M}, s_i \models \phi$, for **all** i;

11. $\mathcal{M}, s \models AF\ \phi$ iff for **all paths** $p = s_1 \to s_2 \to \dots$ we have $\mathcal{M}, s_i \models \phi$, for **some** i;

12. $\mathcal{M}, s \models EF\ \phi$ iff **there exists a path** $p = s_1 \to s_2 \to \dots$ such that $\mathcal{M}, s_i \models \phi$, for **some** i;

13. $\mathcal{M}, s \models A[\phi\ U\ \psi]$ iff for **all paths** $p = s_1 \to s_2 \to \dots$ there exists an i such that $\mathcal{M}, s_i \models \psi$ and $\mathcal{M}, s_j \models \phi$ for all $j < i$;

14. $\mathcal{M}, s \models E[\phi\ U\ \psi]$ iff **there exists a path** $p = s_1 \to s_2 \to \dots$ and an i such that $\mathcal{M}, s_i \models \psi$ and $\mathcal{M}, s_j \models \phi$ for all $j < i$;
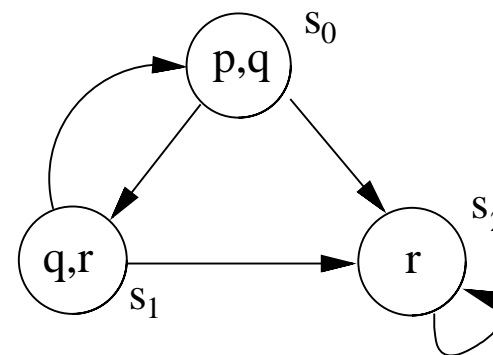
- Notice that 'the future' is the reflexive-transitive closure $\rightarrow^*$ of the (direct) accessibility relation $\rightarrow$.

- In common words:

  $\bigvee$ – the *future contains the present* and

      – *a future of a future of t is a future of t*.

- By unfolding [unwinding] the graph of a CTL model one gets an *infinite tree*, hence *'computation tree logic'*.

kripke图可以转换成树
更接近 CTL
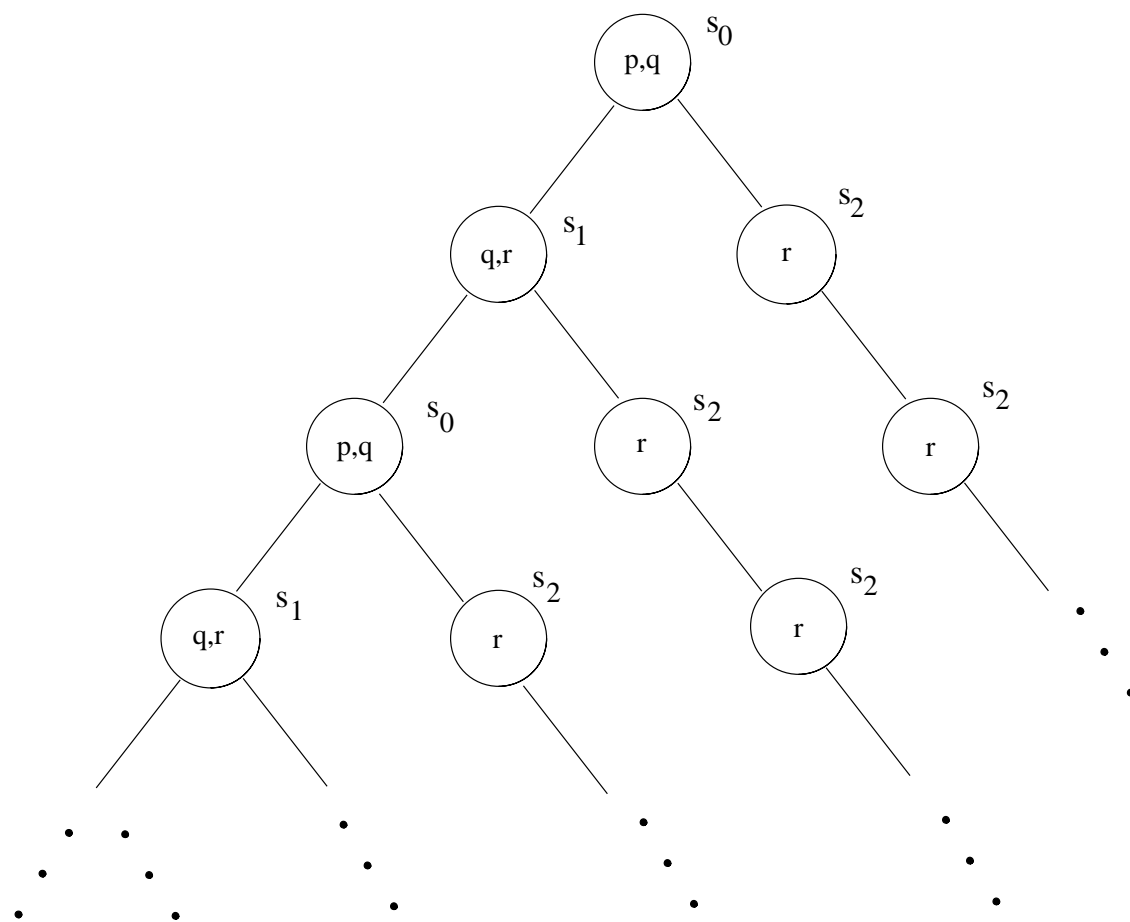
无穷,如何实现算法?

A CTL graph and its unfolding.

The Meaning of EF, EG, AG, and AF

s0   s1   s2   s3   s4   s5   s6   s7   s8   s9   s10
●————●————●————●————●————●————●————●————●————●————●———— ...

      p   p   p   p   p   p   q
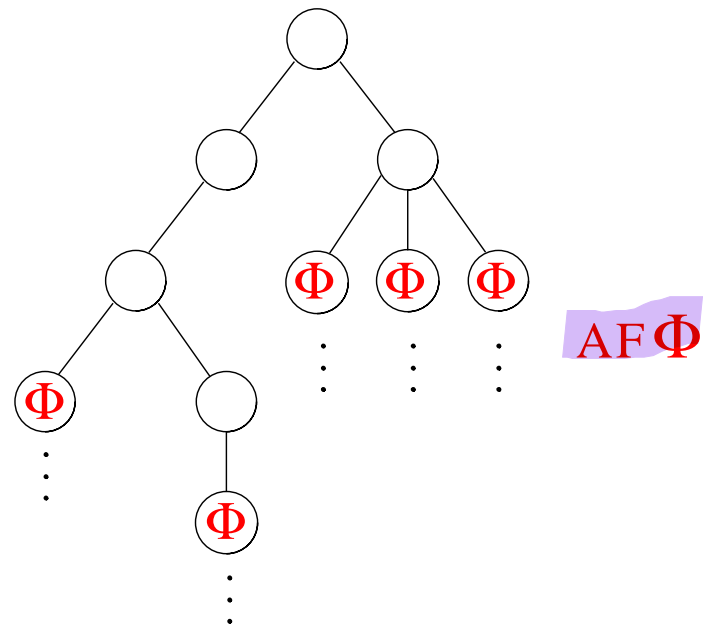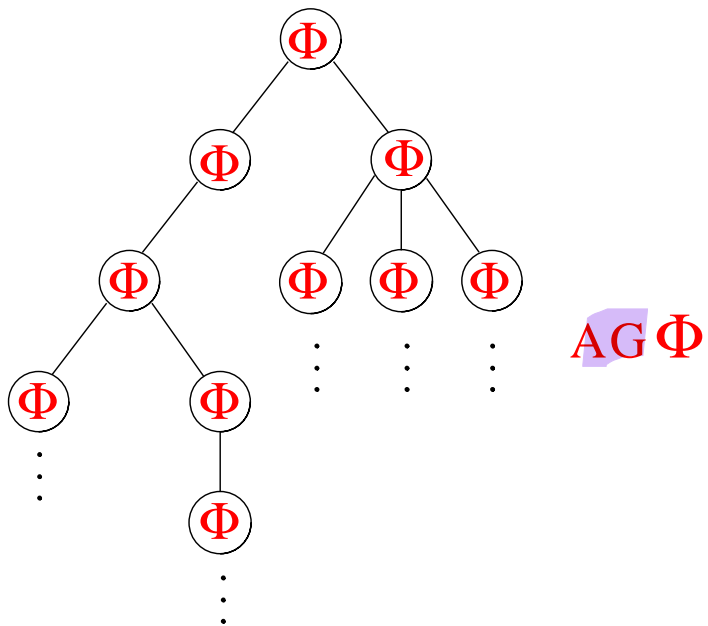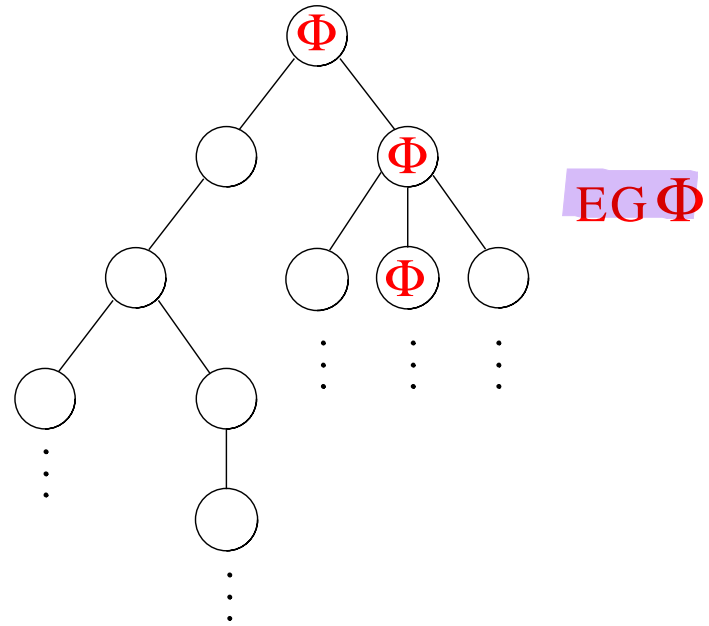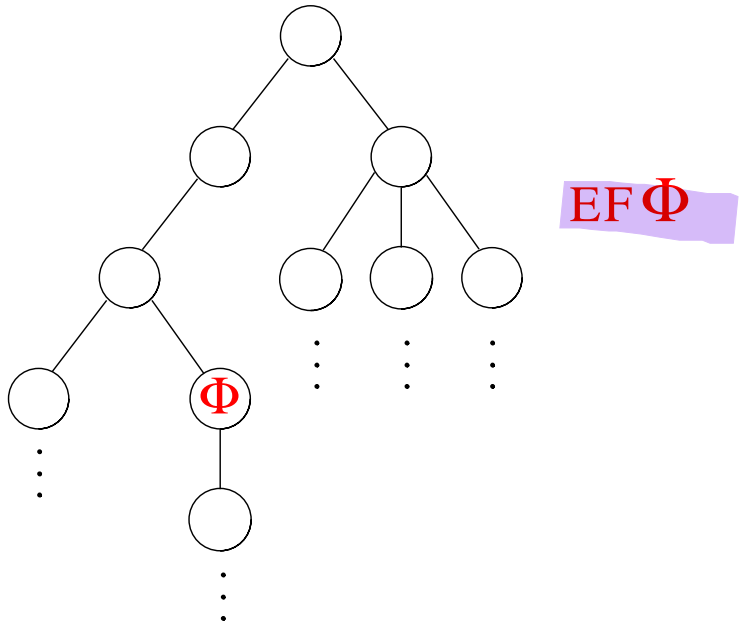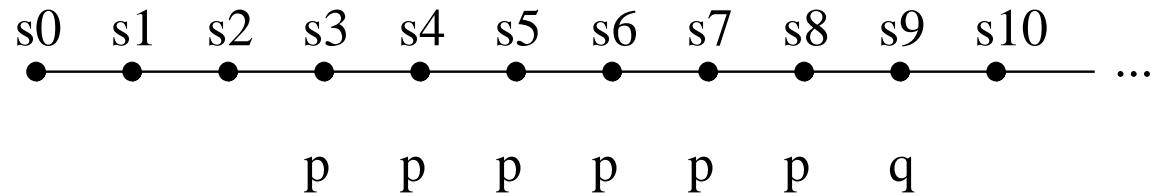
Until *p* U *q* in a linear time model [or on a path in CTL].

The formula *p* U *q* holds in *s3*, but not in *s0* (we suppose *p* holds only in the indicated states)

Examples
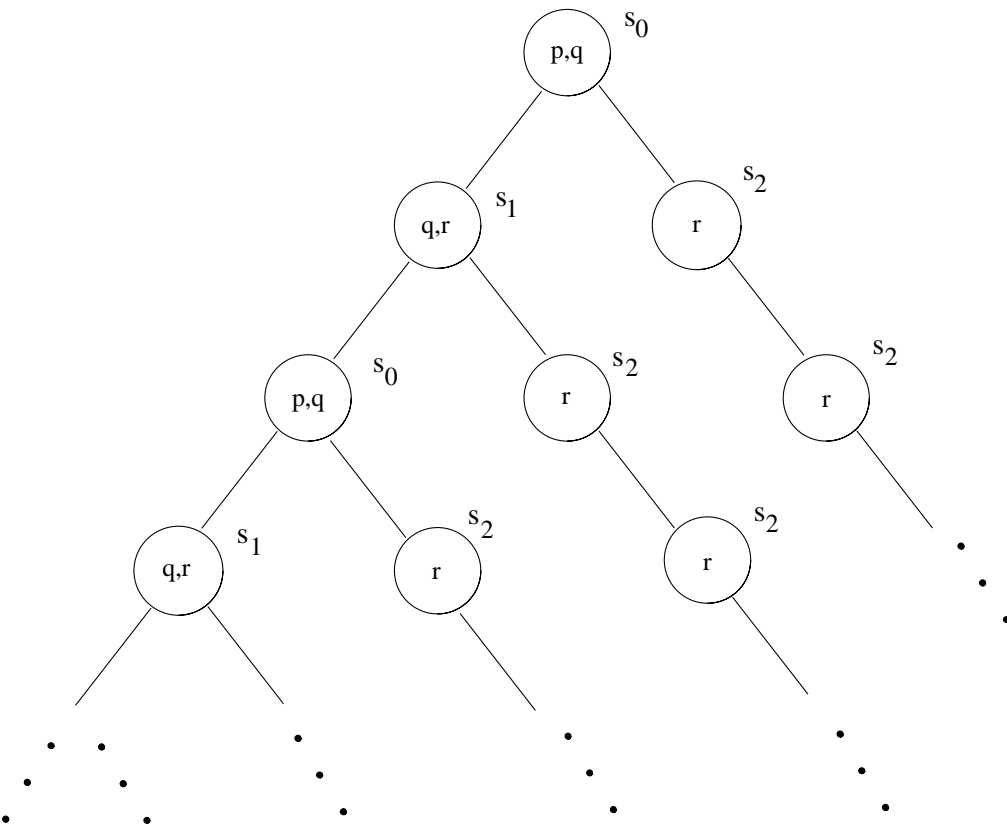
$$\mathcal{M}, s0 \models p \wedge q$$

$$\mathcal{M}, s0 \models \neg r$$

$$\mathcal{M}, s0 \models \top$$

$$\mathcal{M}, s0 \models \text{EX} (q \wedge r)$$

$$\mathcal{M}, s0 \models \neg \text{AX} (q \wedge r)$$

$$\mathcal{M}, s0 \models \neg \text{EF} (p \wedge r)$$

$$\mathcal{M}, s0 \models \text{EG} \; r$$

$$\mathcal{M}, s2 \models \text{EG} \; r$$

$$\mathcal{M}, s2 \models \text{AG} \; r$$

$$\mathcal{M}, s0 \models \text{AF} \; r$$

$$\mathcal{M}, s0 \models \text{E}[(p \wedge q) \; \text{U} \; r]$$

$$\mathcal{M}, s0 \models \text{A}[p \; \text{U} \; r]$$
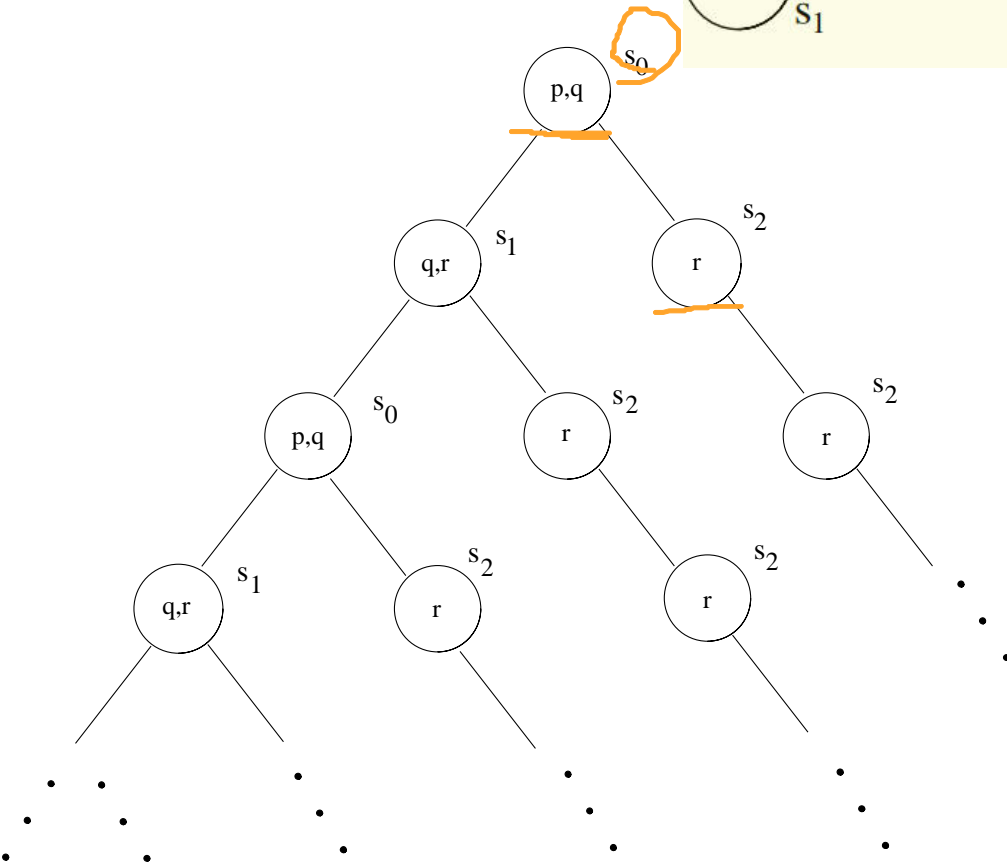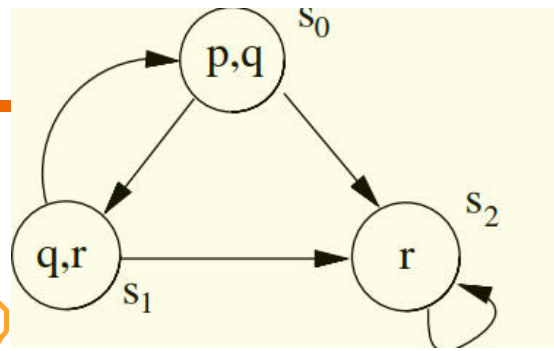
包含自身

$\mathcal{M}, s0 \models p \wedge q$ - Yes ✓

$\mathcal{M}, s0 \models \neg r$ - Yes ✓

$\mathcal{M}, s0 \models \top$ - Yes ✓

$\mathcal{M}, s0 \models \mathrm{EX}\,(q \wedge r)$ - Yes ✓

$\mathcal{M}, s0 \models \neg \mathrm{AX}\,(q \wedge r)$ - Yes

$\mathcal{M}, s0 \models \neg \mathrm{EF}\,(p \wedge r)$ - Yes

$\mathcal{M}, s0 \models \mathrm{EG}\,r$ - No   S0不满足

$\mathcal{M}, s2 \models \mathrm{EG}\,r$ - Yes

$\mathcal{M}, s2 \models \mathrm{AG}\,r$ - Yes

$\mathcal{M}, s0 \models \mathrm{AF}\,r$ - Yes

$\mathcal{M}, s0 \models \mathrm{E}[(p \wedge q)\ \mathrm{U}\ r]$ - Yes ✓

$\mathcal{M}, s0 \models \mathrm{A}[p\ \mathrm{U}\ r]$ - Yes

Practical Patterns of Specifications

Examples of practically relevant properties that may be checked:

- it is possible to reach a state where *started* holds, but *ready* not:

$$\text{进程创建} \quad \text{EF}(started \wedge \neg ready)$$

- for any state, if a *request* occurs, then it will eventually be *acknowledged*:

$$web\ 请求$$
$$\text{AG}(request \rightarrow \text{AF } acknowledged)$$

- a certain process is *enabled* infinitely often on every computation path:

$$敏活$$
$$\text{AG}(\text{AF } enabled)$$

- whatever happens, a certain process will eventually be permanently *deadlocked*:

$$系统最终会$$
$$停止$$
$$\text{AF}(\text{AG } deadlocked)$$

- from any state it is possible to get to a *restart* state:

$$\text{AG}(\text{EF } \textit{restart})$$

- an upwards traveling elevator at the 2nd floor does not change its direction when has passengers going to the 5th floor:

即使有人按了-楼的按钮也不会立即改变方向

$$\text{AG}(floor = 2 \wedge direction = up \wedge ButtonPressed5$$
$$\rightarrow \text{A}[direction = up \text{ U } floor = 5])$$

- the elevator can remain idle on the third floor with its doors closed

$$\text{AG}(floor = 3 \wedge idle \wedge door = closed$$
$$\rightarrow \text{EG}(floor = 3 \wedge idle \wedge door = closed))$$

Definition: Two CTL formulas ϕ and ψ are ***semantically equivalent***, denoted $\phi \equiv \psi$ if any state in any model that satisfies one of them also satisfies the other.

Useful equivalences:

用定义证明

1 $\neg AF\,\phi \equiv EG\,\neg\phi$ 2 $\neg EF\,\phi \equiv AG\,\neg\phi$

3 $\neg AX\,\phi \equiv EX\,\neg\phi$

4 $AF\,\phi \equiv A[\top\ U\ \phi]$ 5 $EF\,\phi \equiv E[\top\ U\ \phi]$

Corollary (adequate sets of temporal connectives): The following sets of connectives are adequate for CTL (in the sense that each CTL formula may be transformed into an equivalent one using only those connectives):

用3个谓词可以表达整个CTL系统，只需要3个验证算法

- AU, EU *and* EX;

- EG, EU *and* EX; (hint for proof: $A[\phi \cup \psi] \equiv \neg(E[\neg\psi \cup (\neg\phi \wedge \neg\psi)] \vee EG \neg\psi)$)

- AG, AU *and* AX;

- AF, EU *and* AX

无限问题、递归、归纳

More useful equivalences (fixed-point definitions)

递归形式

6 $\text{AG } \phi \equiv \phi \wedge \text{AX AG } \phi$

7 $\text{EG } \phi \equiv \phi \wedge \text{EX EG } \phi$

8 $\text{AF } \phi \equiv \phi \vee \text{AX AF } \phi$

9 $\text{EF } \phi \equiv \phi \vee \text{EX EF } \phi$

10 $\text{A}[\phi \text{ U } \psi] \equiv \psi \vee (\phi \wedge \text{AX A}[\phi \text{ U } \psi])$

11 $\text{E}[\phi \text{ U } \psi] \equiv \psi \vee (\phi \wedge \text{EX E}[\phi \text{ U } \psi])$

A mechnism for solving such ***fixed-point*** equations $Y = \phi \wedge \text{AX } Y$ and the next operators AX and EX are sufficient to represent all temporal logic operators.

Goal: to develop protocols for accessing some critical sections such that only one process can be in its critical section at a time. A collection of desirable properties is:

Safety: the protocol allows only one process to be in its critical section at any time

Liveness: whenever any process wants 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

A simple model MUT1:
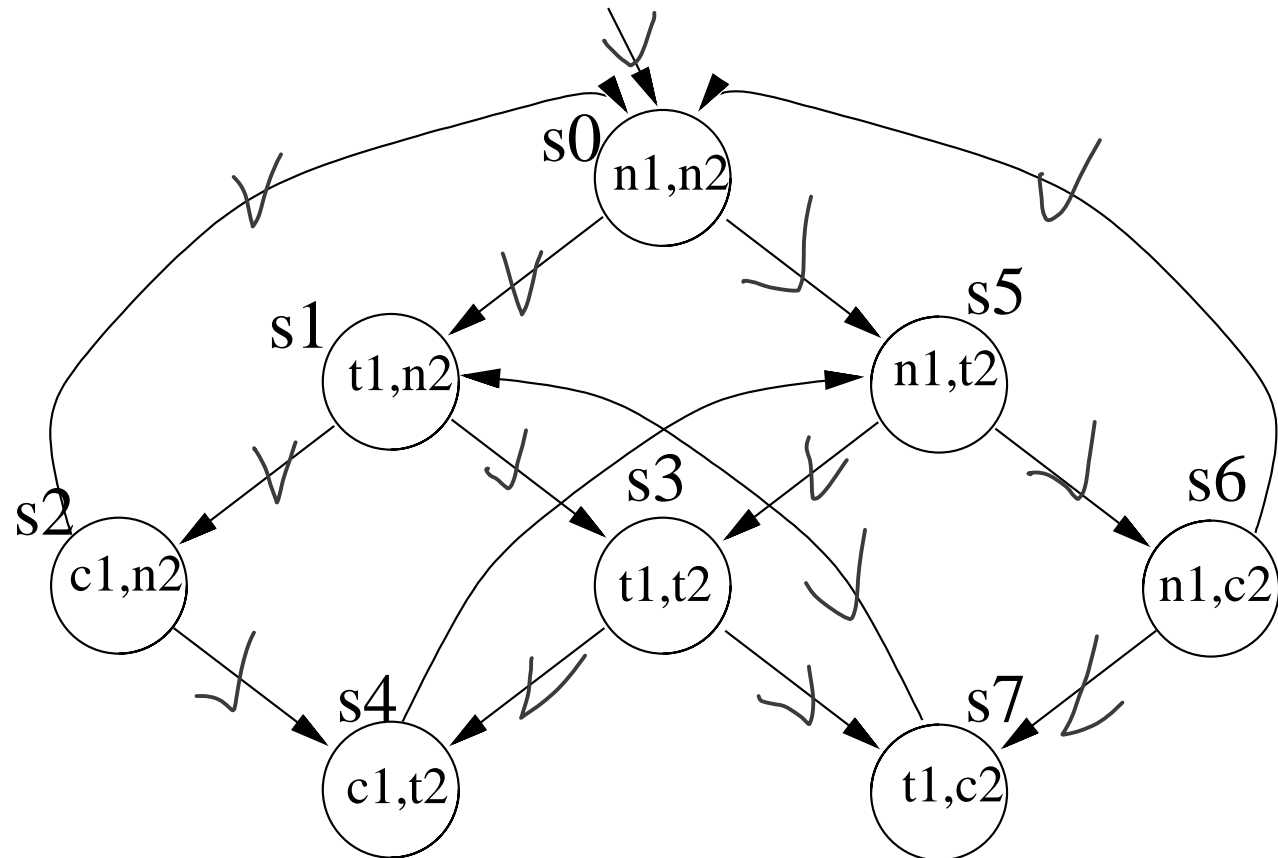
The system consists of two processes $P1$ and $P2$, each making a loop $n \to t \to c \to \ldots$ (noncritical $\to$ trying $\to$ critical $\to \ldots$). The system's behaviour is the product (interleaving) of the behaviours of $P1$ and $P2$, but the state $(c1, c2)$ is excluded.

Safety: $\phi_1 =_{def} \text{AG} \neg (c1 \wedge c2)$ —satisfied in each state;

Liveness: $\phi_2 =_{def} \text{AG}(t1 \rightarrow \text{AF } c1)$ —not satisfied in the initial state $s0$; e.g., $s1$ is accessible, $t1$ is true, but there is a path $s1 \rightarrow s3 \rightarrow s7 \rightarrow s1 \rightarrow \ldots$ where $c1$ is always false;

Non-blocking: $\phi_3 =_{def} \text{AG}(n1 \rightarrow \text{EX } t1)$ —true

No strict sequencing:

$$\phi_4 =_{def} \text{EF}(c1 \wedge \text{E}[c1 \text{ U } (\neg c1 \wedge \text{E}[\neg c2 \text{ U } c1])]) \text{ —true}$$

C_1 连续 两次进入 critical section

A second model MUT2:

谁先请求 谁先进入 Critical Section
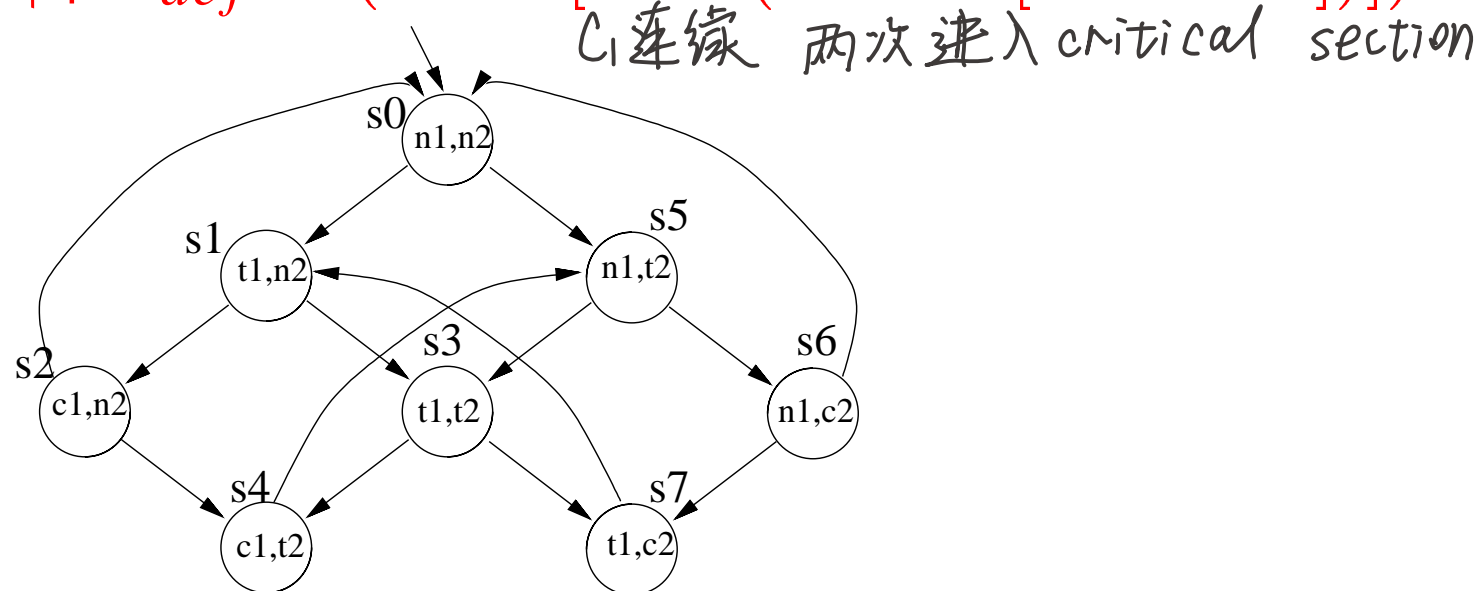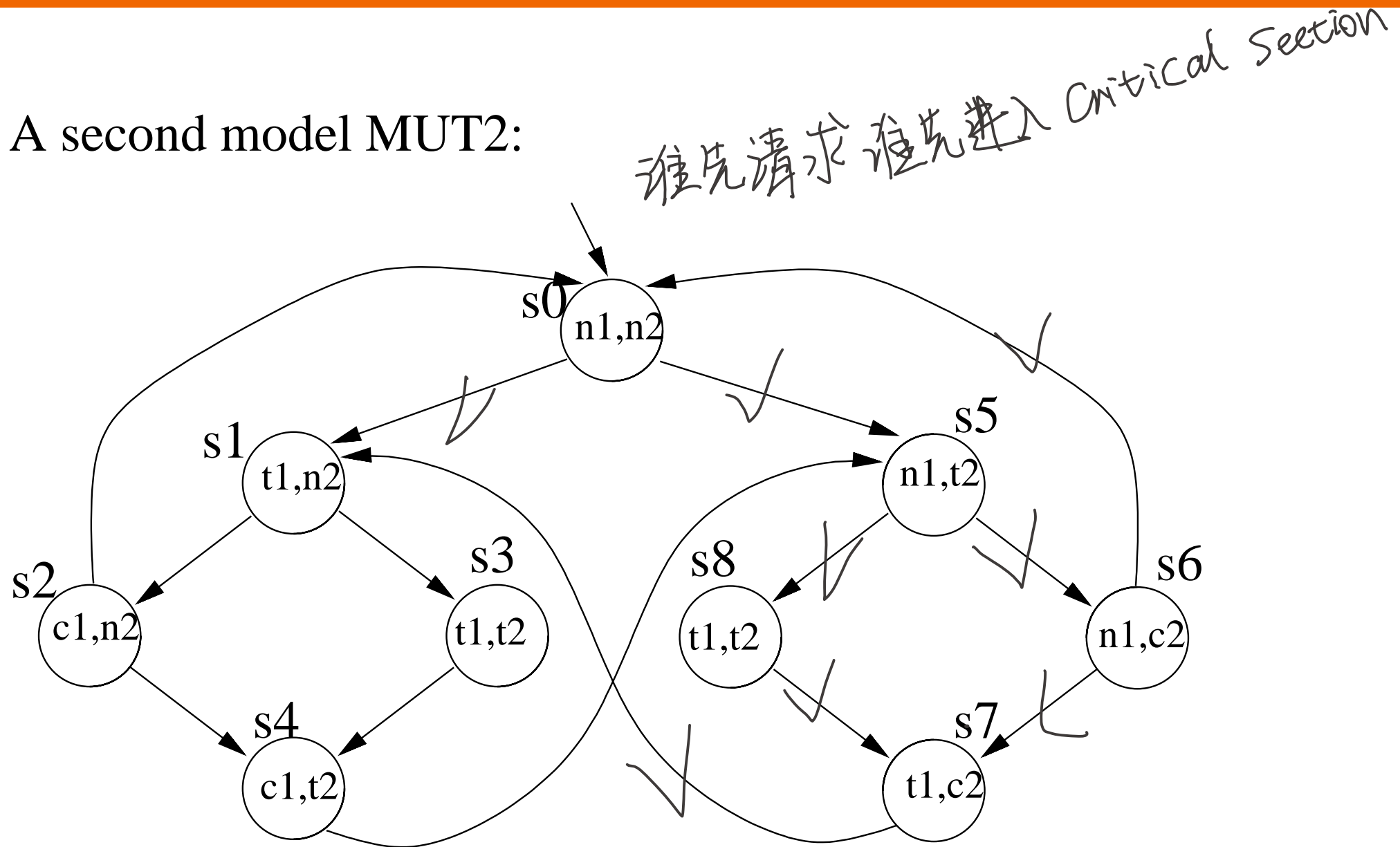
- This model is obtained by splitting state $s3$ of MUT1 in two different states $s3$ and $s8$.

- By spliting $s3$ into two states we are able to identify which process was the first asking to access its critical section: if $P1$ was the first, the the resulting state is $s3$, otherwise $s8$.

Note: Are all four properties (i.e.,formulas ϕ_1 to ϕ_4) valid in MUT2?

Recall that the problem to be solved by model checking is

(A) "Given a model $\mathcal{M}$, a CTL formula ϕ, and *a state s*, does $\mathcal{M}, s \models \phi$ hold?", where

      – $\mathcal{M}$ is a model of the system and s is a state of the model;

      – ϕ is a CTL formula intended to be satisfied by the system

What is the model used for the system? Typically:

> *"A system is represented by a finite transition system (usually, a huge labeled directed graph, often with milions of states)."*

The infinite trees obtained by unfolding such graphs are useful to develop an intuition of the reasoning process, but not to be used on our finite computers.

Given $\mathcal{M}, s$, and ϕ, the result returned by a model checker is either

(1) yes: $\mathcal{M}, s \models \phi$ or

(2) no: $\mathcal{M}, s \not\models \phi$

but, quite useful, in the latter case most of model checkers return a trace/path which invalidates ϕ, as well (a counterexample).

Alternative problem:

(B) Given a model $\mathcal{M}$ and a CTL formula ϕ find **all states** *s* of the model which satisfy ϕ

These two problems are obviously equivalent: once one is able to develop algorithms to solve one of them, the other is solved, as well. We will be mainly concerned with the *latter* problem B.

We are starting with a version using the following reduced set of CTL connectives

$$\Gamma = \{\perp, \neg, \wedge, \text{AF}, \text{EU}, \text{EX}\}$$

where:

- $\perp, \neg$, and $\wedge$ are used for the propositional part

- AF, EU, and EX are used for the temporal part

Hence, there is a ***preprocessing*** procedure to:

1. check the CTL syntax correctness of the given formula ϕ and

2. translate it in a formula TRANSLATE(ϕ) written with connectives in Γ, only.

In the sequel, we suppose ϕ to be in CTL Γ-format.

The idea of this algorithm is to:

- decompose formula ϕ in pieces (sub-formulas) and apply a structural induction to label the graph with sub-formulas of ϕ (the intuition is that *a formula that labels a state is true in that state*)

- for each such sub-formula, parse the graph to infer the truth in a state according to the meaning of the connectives and the truth values of its sub-formulas

In 2, one may need to know the values of sub-formulas in possibly many different states; this is the case for temporal operators, but not for the propositional ones.

Input: a CTL model $\mathcal{M} = (S, \rightarrow, L)$ and a CTL formula ϕ
(in Γ-format)

Output: the set of states of $\mathcal{M}$ which satisfy ϕ

1. $\bot$: no states are labeled with $\bot$

2. p: label with p all states s such that $p \in L(s)$

3. $\neg \phi_1$: label s with $\neg \phi_1$ if s is not already labeled with ϕ_1

4. $\phi_1 \wedge \phi_2$: label s with $\phi_1 \wedge \phi_2$ if s is already labeled both with ϕ_1 and ϕ_2

5. EX ϕ_1: label s with EX ϕ_1 if one of its successors is already labeled with ϕ_1

6. AF ϕ_1 :

  a. (initial marking) label any *s* with AF ϕ_1 if *s* is already labeled with ϕ_1

  b. (repeated marking) label any *s* with AF ϕ_1 if all successor states of *s* are already labeled with AF ϕ_1

  c. repeat (2) until there are no change

7. E$[\phi_1$ U $\phi_2]$:

  a. (initial marking) label any *s* with E$[\phi_1$ U $\phi_2]$ if *s* is already labeled with ϕ_2

  b. (repeated marking) label any *s* with E$[\phi_1$ U $\phi_2]$ if *s* is already labeled with ϕ_1 and at least one of its successor states is already labeled with E$[\phi_1$ U $\phi_2]$

  c. repeat (2) until there is no change

A rough analysis of the algorithm shows that it has the worse time complexity

$$O(k \cdot m \cdot (m + n))$$

where k is the number of connectives of the formula, m is the number of the states of the model, and n is the number of the transitions of the model.

Checking $E[\neg c_2 \ U \ c_1]$ in the second mutual exclusion model MUT2:
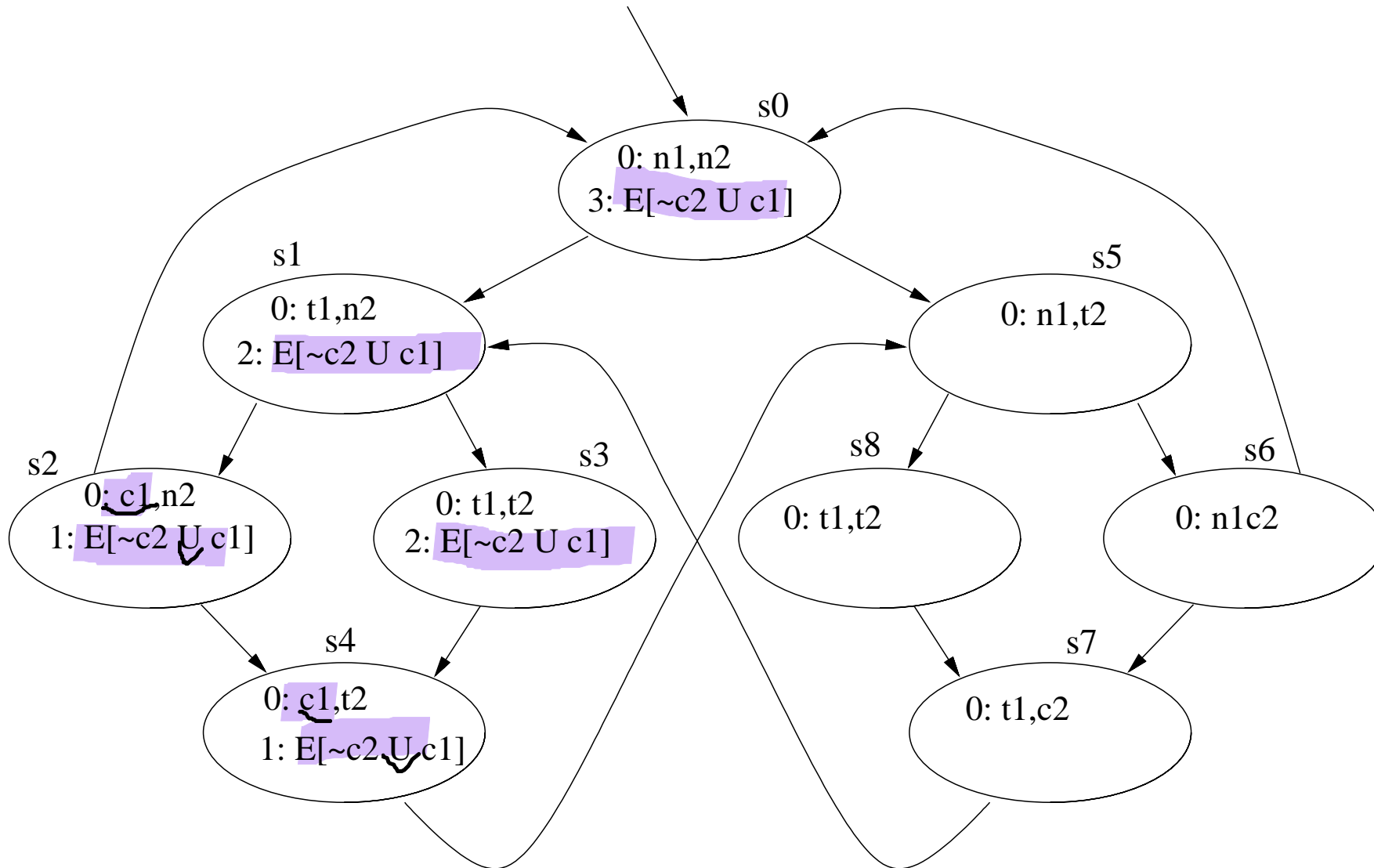
EG may be handled directly as follows:

6' EG ϕ_1:

   0. label *all* states *s* with EG ϕ_1

   1. (initial de-marking) if ϕ_1 does not hold in *s* then *delete* the label EG ϕ_1

   2. (repeated de-marking) *delete* the label EG ϕ_1 from any state *s* if none of its successor states is labeled with EG ϕ_1

   3. repeat (2) until there are no change

This different approach is based on the following greatest fixed-point characterization of EG

$$\text{EG } \phi \equiv \phi \wedge \text{EX EG } \phi$$

An Improved Variant

- Use EX, EU, and EG instead of EX, EU, and AF

- handle EX and EU as before (using backwards breadth-first search)

- for $\mathrm{EG}\ \phi$

  - restrict to states satisfying ϕ

  - find SCCs (maximal strongly connected components; these are maximal regions such that any vertex is connected to any other vertex in the region)

  - use backwards breadth-first searching on the restricted graph to find any state that can reach an SCC

Complexity is reduced to $O(k \cdot (m+n))$ (k, m, n as before).

function SAT(ϕ):

/* precondition: ϕ is an arbitrary CTL formula */

/* postcondition: SAT(ϕ) returns the set of states satisfying ϕ */

begin function

    case

        ϕ is $\top$: **return** S

        ϕ is $\bot$: **return** $\emptyset$

        ϕ is atomic formula: **return** $\{s \in S \mid \phi \in L(s)\}$

        ϕ is $\neg\phi_1$: **return** $S \setminus$ SAT(ϕ_1)

        ϕ is $\phi_1 \wedge \phi_2$: **return** SAT$(\phi_1) \cap$ SAT(ϕ_2)

        ϕ is $\phi_1 \vee \phi_2$: **return** SAT$(\phi_1) \cup$ SAT(ϕ_2)

        ϕ is $\phi_1 \rightarrow \phi_2$: **return** SAT$(\neg\phi_1 \vee \phi_2)$

(...cont.)

ϕ is $\text{AX}\phi_1$: **return** $\text{SAT}(\neg\text{EX}\neg\phi_1)$

ϕ is $\text{EX}\phi_1$: **return** $\text{SAT}_{EX}(\phi_1)$

ϕ is $\text{A}[\phi_1\text{U}\phi_2]$:

   return $\text{SAT}(\neg(\text{E}[\neg\phi_1\text{U}(\neg\phi_1 \wedge \neg\phi_2)] \vee \text{EG}\neg\phi_2))$

ϕ is $\text{E}[\phi_1\text{U}\phi_2]$: **return** $\text{SAT}_{EU}(\phi_1, \phi_2)$

ϕ is $\text{EF}\phi_1$: **return** $\text{SAT}(\text{E}[\top \text{ U } \phi_1])$

ϕ is $\text{EG}\phi_1$: **return** $\text{SAT}(\neg\text{AF}\neg\phi_1])$

ϕ is $\text{AF}\phi_1$: **return** $\text{SAT}_{AF}(\phi_1)$

ϕ is $\text{AG}\phi_1$: **return** $\text{SAT}(\neg\text{EF}\neg\phi_1)$

   end case

end function

function $\text{SAT}_{EX}(\phi)$:

/* pre: ϕ is an arbitrary CTL formula */

/* post: $\text{SAT}_{EX}(\phi)$ returns the set of states satisfying $\text{EX } \phi$ */

local var X, Y

begin

$\quad X := \text{SAT}(\phi)$;

$\quad Y := \{ s_0 \in S \mid s_0 \to s_1 \text{ for some } s_1 \in X \}$;

$\quad$ **return** Y

end

function $\text{SAT}_{AF}(\phi)$:

/* pre: ϕ is an arbitrary CTL formula */

/* post: $\text{SAT}_{AF}(\phi)$ returns the set of states satisfying AF ϕ */

local var X, Y

begin

$\quad X := S$;

$\quad Y := \text{SAT}(\phi)$;

$\quad$ **repeat until** $X = Y$

$\qquad$ **begin**

$\qquad\quad X := Y$;

$\qquad\quad Y := Y \cup \{s \in S \mid \text{ for all } s' \text{ with } s \to s' \text{ we have } s' \in Y\}$;

$\qquad$ **end**

$\quad$ **return** Y

end

function $\text{SAT}_{EU}(\phi, \psi)$:

/* pre: ϕ is an arbitrary CTL formula */

/* post: $\text{SAT}_{EU}(\phi, \psi)$ returns the set of states satisfying $\text{E}[\phi \text{U} \psi]$ */

local var W, X, Y

begin

    $W := \text{SAT}(\phi)$;

    $X := S$;

    $Y := \text{SAT}(\psi)$;

    repeat until $X = Y$

        begin

            $X := Y$;

            $Y := Y \cup (W \cap \{s \in S \mid \text{exists } s' \text{ such that } s \to s' \text{ and } s' \in Y\}$;

        end

    return Y

end

- the labeling algorithm is quite efficient [linear in the size of the model]

- ... but the model itself may be large, exponential in the number of the components (running in parallel 10 threads each of them having 10 states results in a systems with $10^{10} = 10,000,000,000$ states!)

- the tendency of the state space to become very large is commonly referred to as the ***state explosion*** problem

- the state explosion problem is mainly ***unsolved*** - no general solution is known at the moment

The problem is general unsolved. The following techniques were developed to overcome it in certain particular cases:

1. *efficient data structures* - e.g., *ordered binary decision diagrams OBDDs* (OBDDs are used to represents sets of states, not individual states)

2. *abstraction* - one may abstract away variables in the model that are not relevant for the formula being checked

3. *partial order reduction* - different runnings may be equivalent as far as the formula to be checked is concerned; partial order reduction check one trace from such a class only

4. *induction* - this technique is used when a large number of processes is considered

5. *composition* - try to split the problem in small parts to be separately checked