Model-Checking Untimed Systems

Transition Systems, Temporal Logics and
Basic Verification Algorithms

OUTLINE

- A Brief Introduction
  - Motivation ... what are the problems to solve
  - CTL, LTL and basic model-checking algorithms
- Timed Systems
  - Timed automata, TCTL and verification problems
  - UPPAAL tutorial: data stuctures & algorithms
  - TIMES: schedulability analysis using timed automata
- Recent Work
  - The multicore timing analysis problems
  - Some solutions: WCET analysis and multiprocessor scheduling

Transition Systems

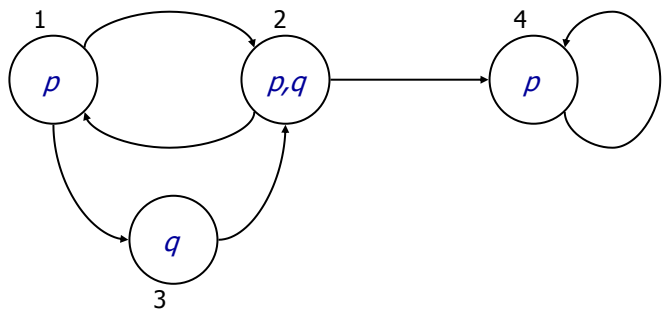
A transition system is a graph with

- a set of nodes (states)
- a set of edges (transitions)

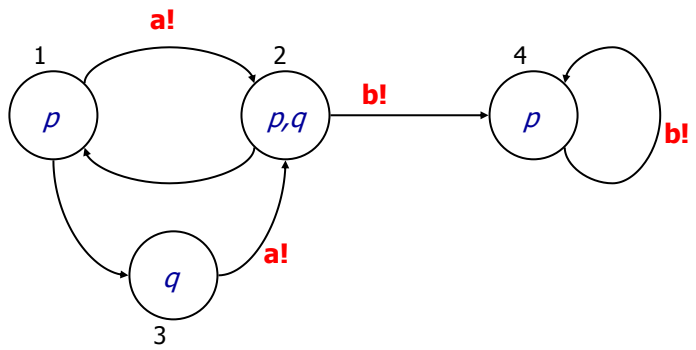
where

- nodes may be labeled with propositions (state properties)
- edges may be labeled with action names (synchronization)

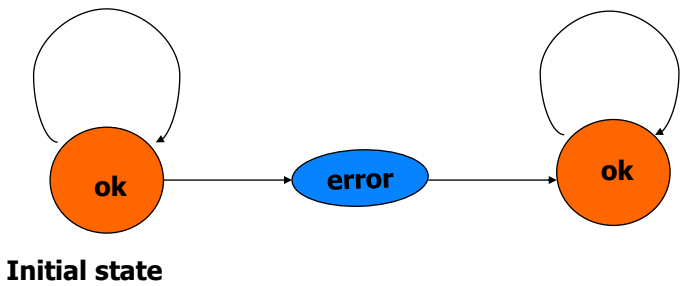
Example (with labeled nodes)



EXAMPLE (with labeled nodes, and labeled edges)

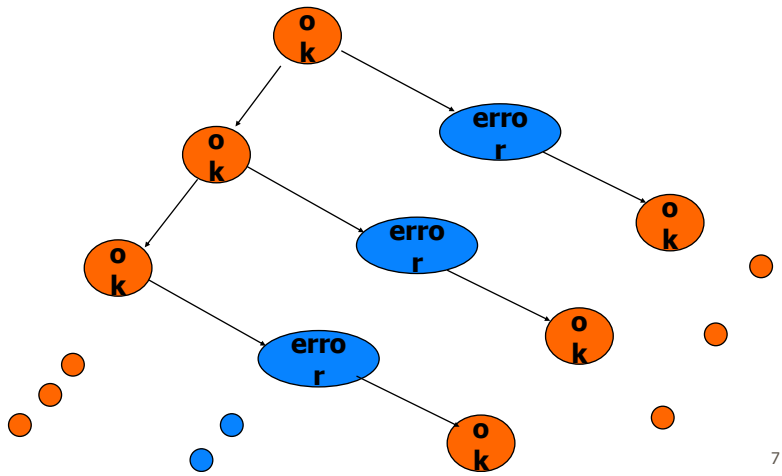


EXAMPLE: a BUGGY machine



The (branching-time) semantic of BUGGY

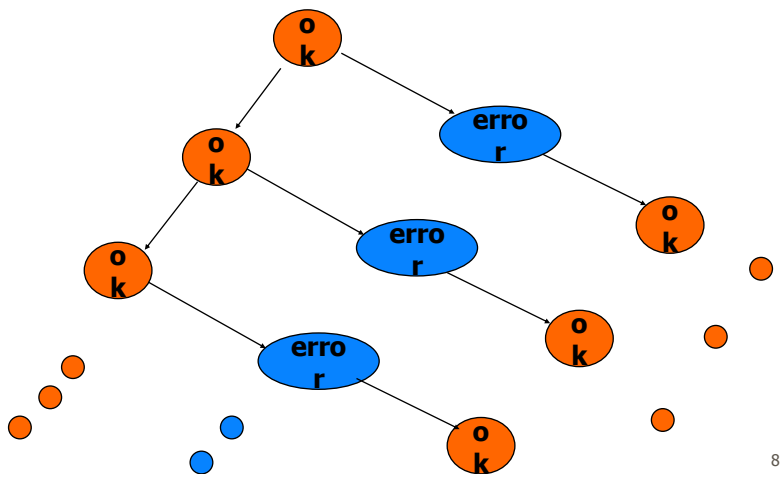
Computation Tree



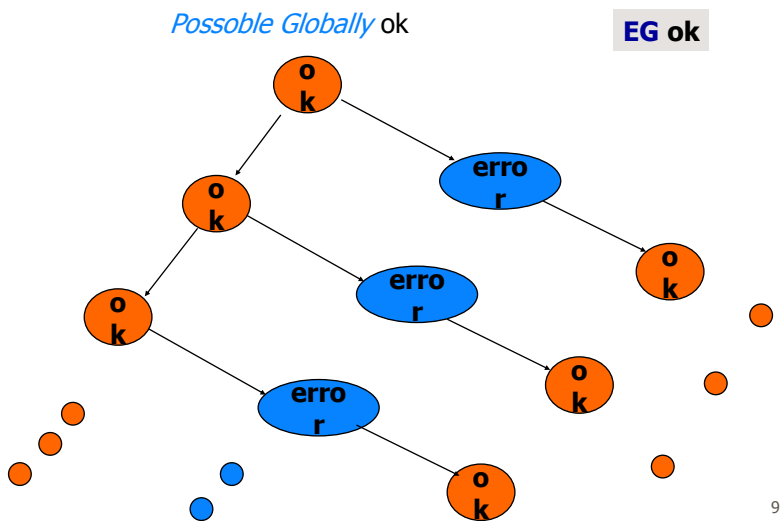
”Properties” of BUGGY

Possible error

EF error

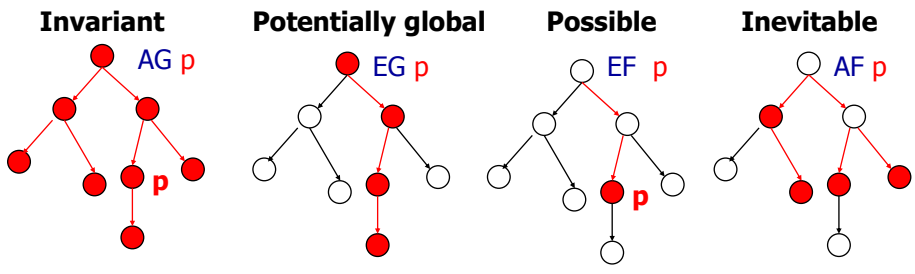


”Properties” of BUGGY



9

Properties of Computation Trees



10

CTL: Computation Tree Logics

defined on Computation Trees of Kripke structures

Computation Tree Logic, CTL

Clarke & Emerson 1980

Syntax

ϕ **::=** **P** | $\neg\,\phi$ | $\phi \vee \phi$ | **EX** ϕ | **E[**ϕ **U** ϕ**]** | **A[**ϕ **U** ϕ**]**

where **P** $\in$ AP (atomic propositions)

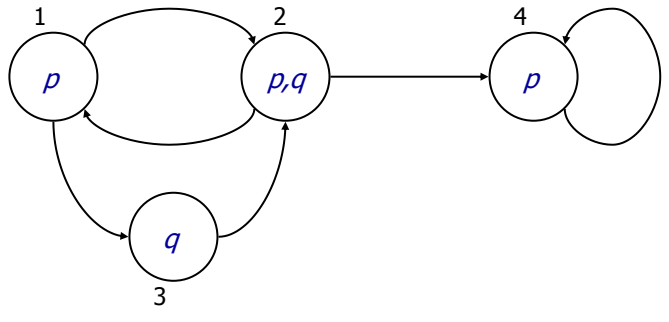
- EX (pronounced "for some path next")
- E (pronounced "for some path")
- A (pronounced "for all paths") and
- U (pronounced "until").

A CTL-model is a Kripke Structure (=transition systems with labeled nodes)

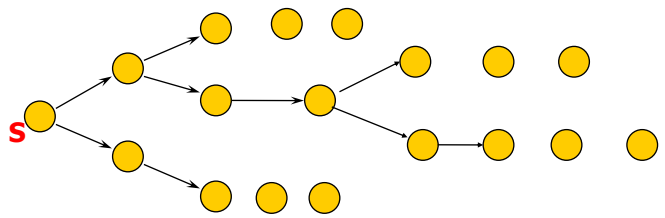
M = <S, E, Label> where

- S is a non-empty set of states
- $E \subseteq S \times S$ is a transition relation
- Label: $S \rightarrow 2^{AP}$ is a labeling function

Example

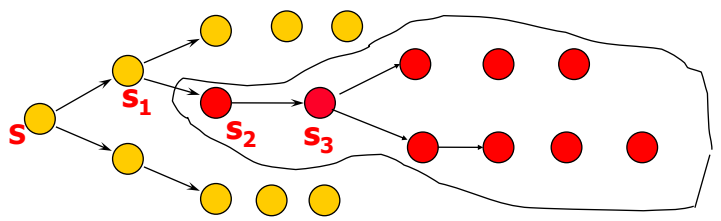


Computation Trees vs. STATES



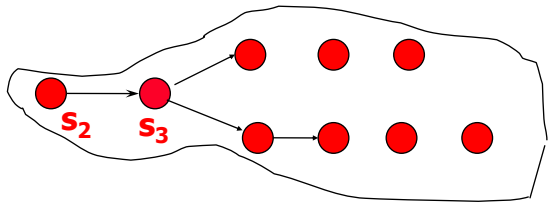
The computation tree of state s

Computation Trees vs. STATES



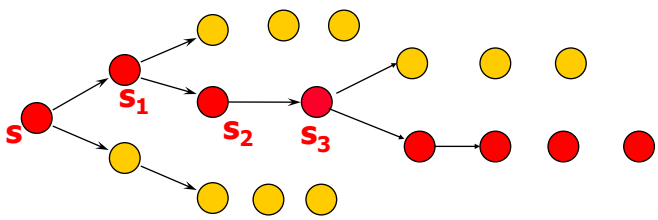
The computation tree of state s_2

Computation trees of STATES



The computation tree of state s_2

Path (of computation tree)



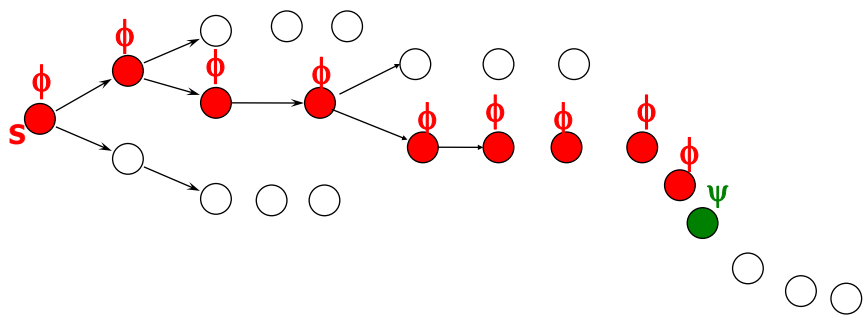
A path is an infinite sequence of states
e.g. $\sigma = s \; s_1 \, s_2 \, s_3 \; \ldots \ldots$

Formal Semantics of CTL

$$
\begin{array}{ll}
s \models p & \text{iff } p \in Label(s) \\
s \models \neg\phi & \text{iff } \neg(s \models \phi) \\
s \models \phi \vee \psi & \text{iff } (s \models \phi) \vee (s \models \psi) \\
s \models \mathsf{EX}\,\phi & \text{iff } \exists \sigma \in P_{\mathcal{M}}(s).\, \sigma[1] \models \phi \\
s \models \mathsf{E}\,[\phi\,\mathsf{U}\,\psi] & \text{iff } \exists \sigma \in P_{\mathcal{M}}(s).(\exists j \geqslant 0.\, \sigma[j] \models \psi \ \wedge\ (\forall 0 \leqslant k < j.\, \sigma[k] \models \phi)) \\
s \models \mathsf{A}\,[\phi\,\mathsf{U}\,\psi] & \text{iff } \forall \sigma \in P_{\mathcal{M}}(s).(\exists j \geqslant 0.\, \sigma[j] \models \psi \ \wedge\ (\forall 0 \leqslant k < j.\, \sigma[k] \models \phi)).
\end{array}
$$

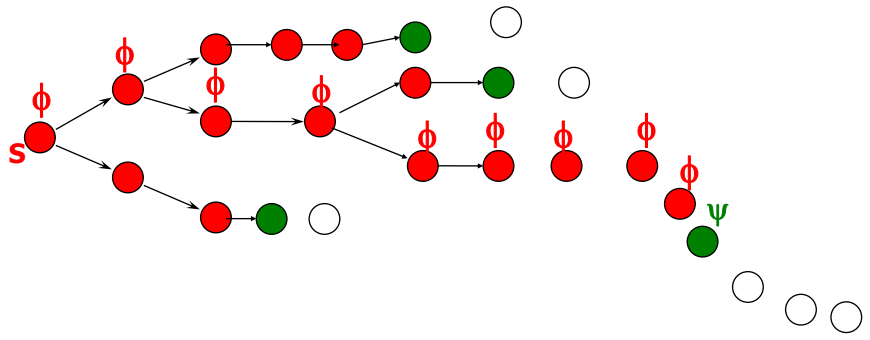
Where $P_M(s)$ denotes the set of pathes starting from s and σ[i] denotes i'th element of σ

E[ϕ U ψ]



E[ϕ U ψ] is valid in **s** if <u>some path</u> from **s** satisfies the above

A[ϕ U ψ]

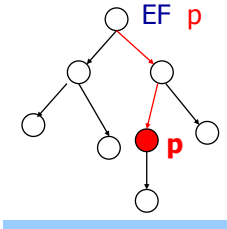


A[ϕ U ψ] is valid in **s** if <u>all pathes</u> from **s** satisfy the above

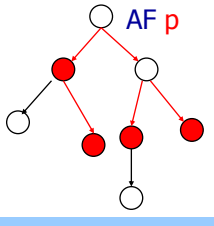
CTL, Derived Operators

$$\mathsf{EF}\,\phi \equiv \mathsf{E}\,[\mathrm{true}\,\mathsf{U}\,\phi]$$ *possible*

$$\mathsf{AF}\,\phi \equiv \mathsf{A}\,[\mathrm{true}\,\mathsf{U}\,\phi].$$ *inevitable*



E<> P in UPPAAL

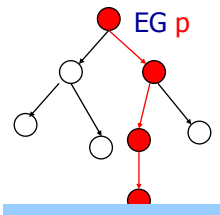


A<> P in UPPAAL

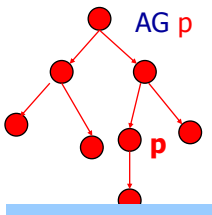
CTL, Derived Operators

$$EG\,\phi \equiv \neg AF\,\neg\phi$$ *potentially always*

$$AG\,\phi \equiv \neg EF\,\neg\phi$$ *always*



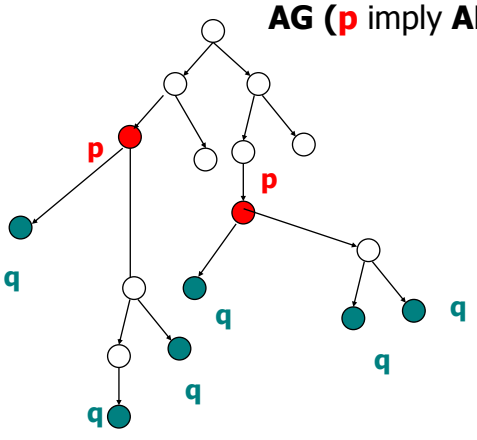
E[] P in UPPAAL



A[] P in UPPAAL

CTL, Derived Operators (cont.)

AG (p imply **AF q)**



p - -> q in **UPPAAL**

Theorem

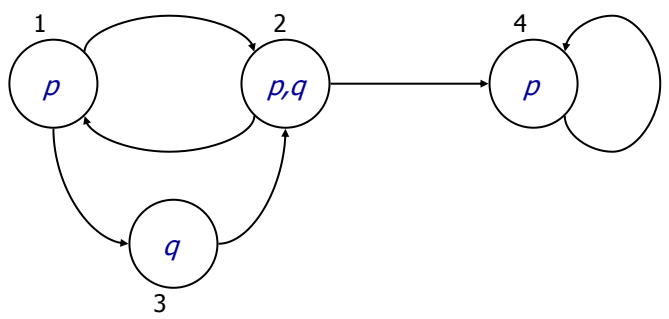
All operators are derivable from

- EX f
- EG f
- E[f U g]

and boolean connectives

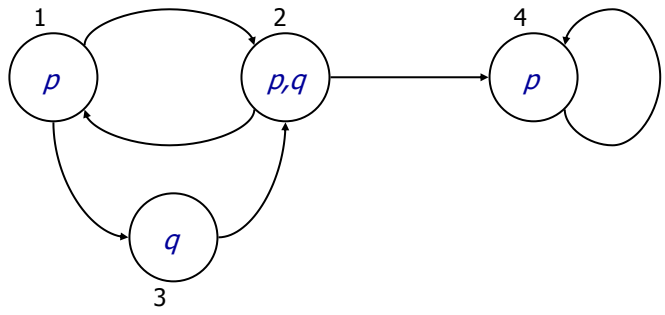
$$A[f \text{ U } g] \equiv \neg E[\neg g \text{ U} (\neg f \wedge \neg g)] \wedge \neg EG \neg g$$

Example



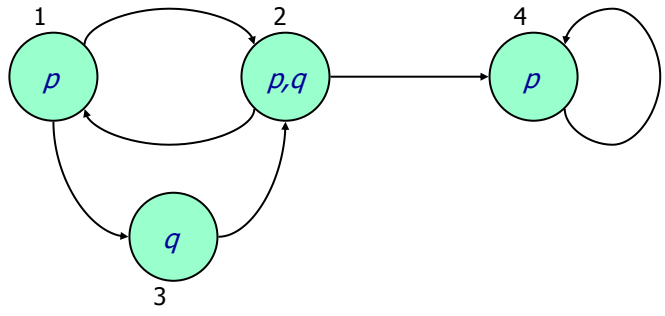
Example

EX *p*



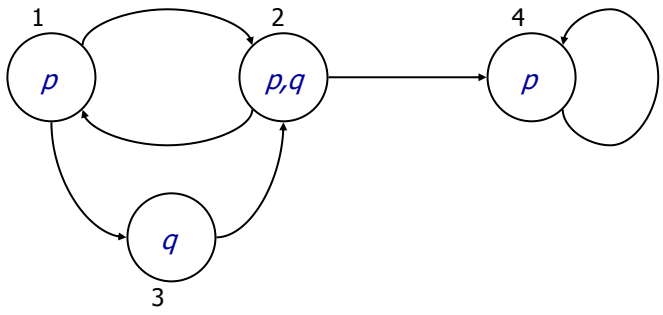
Example

EX *p*



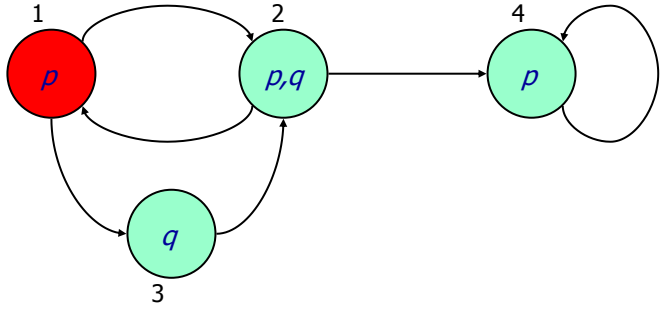
Example

AX *p*



Example

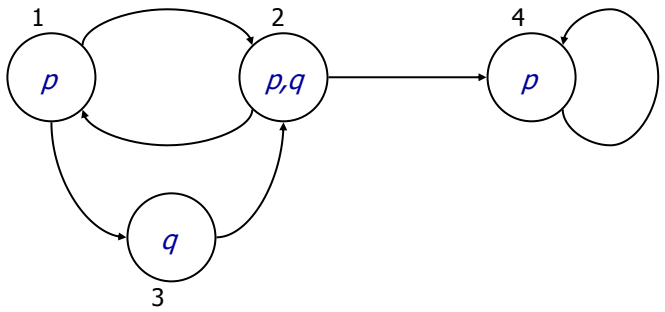
AX *p*



Note: state 1 doesn't satisfy AX *p*

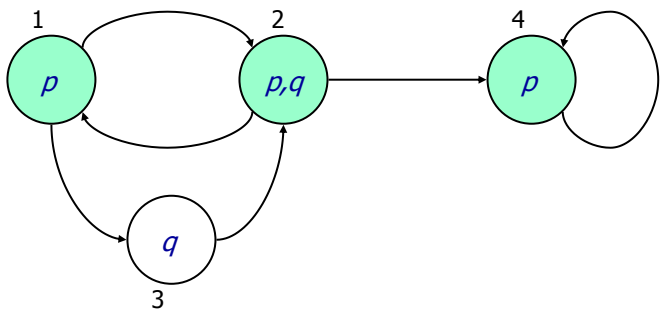
Example

EG *p*



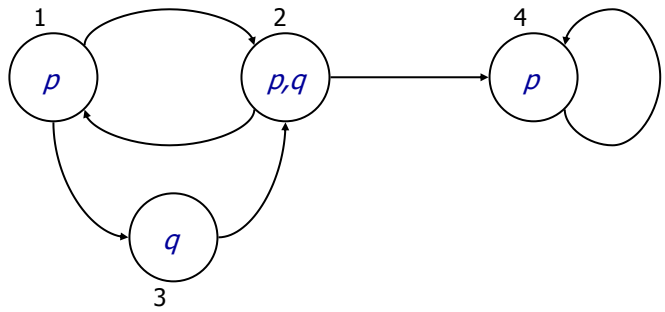
Example

EG *p*



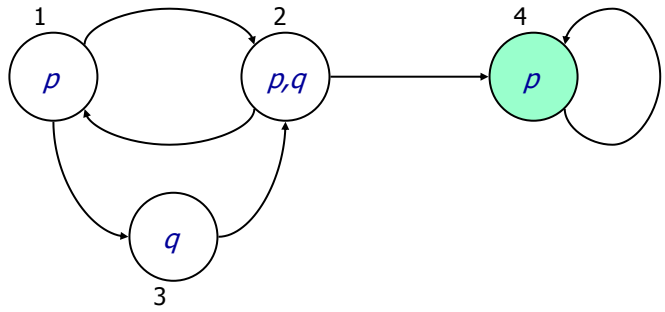
Example

AG *p*



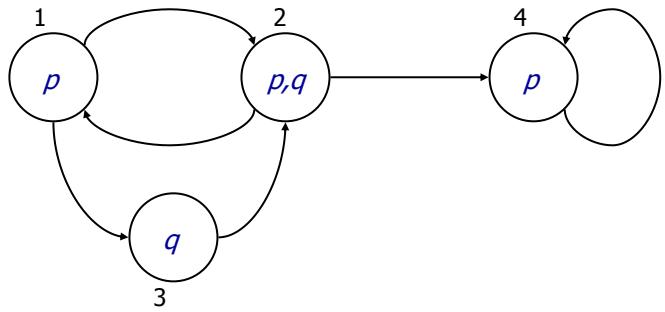
Example

AG *p*



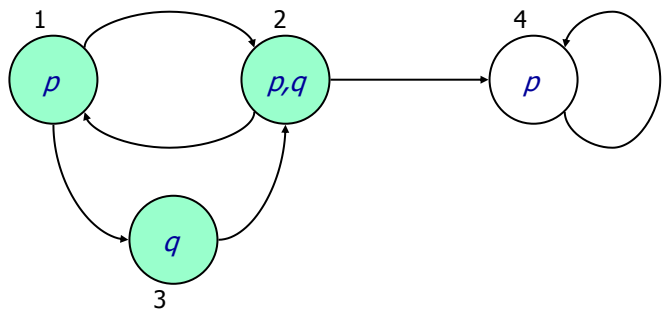
Example

A[*p* **U** *q* **]**



Example

A[*p* **U** *q* **]**



Properties of MUTEX example ?

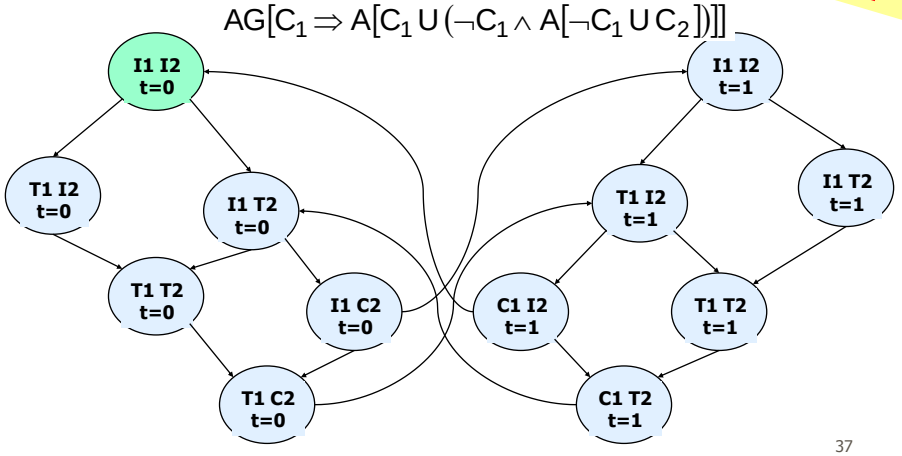
$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$

HOW to DECIDE IN GENERAL



CTL Model-Checking Algorithms

Labeling Methods [Clarke et al 81]

- Sat(ϕ) = all states where ϕ is true
- Compute Sat(ϕ) recursively as follows:
  - For each sub-formula ϕ_i of ϕ, compute Sat(ϕ_i)
  - This is easier: e.g. Sat(P) = {s | P$\in$Label(s)}
- Compose Sat(ϕ_i) to get Sat(ϕ)

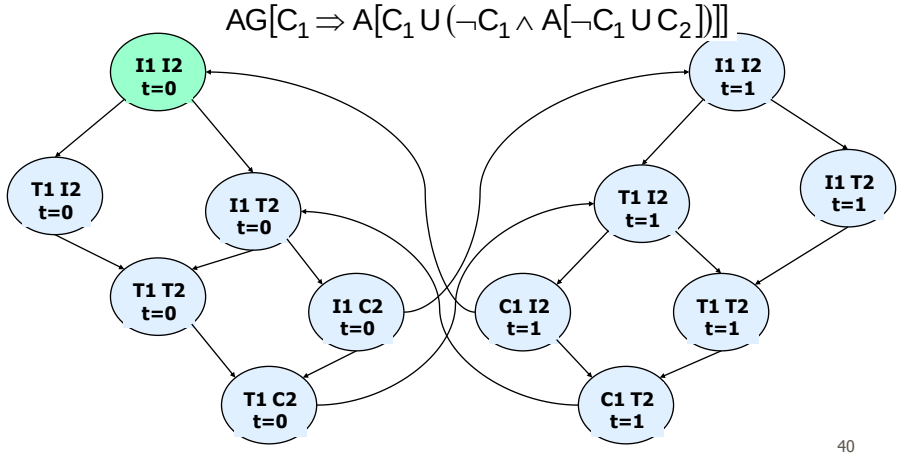
Properties of MUTEX example ?

$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$



Properties of MUTEX example ?

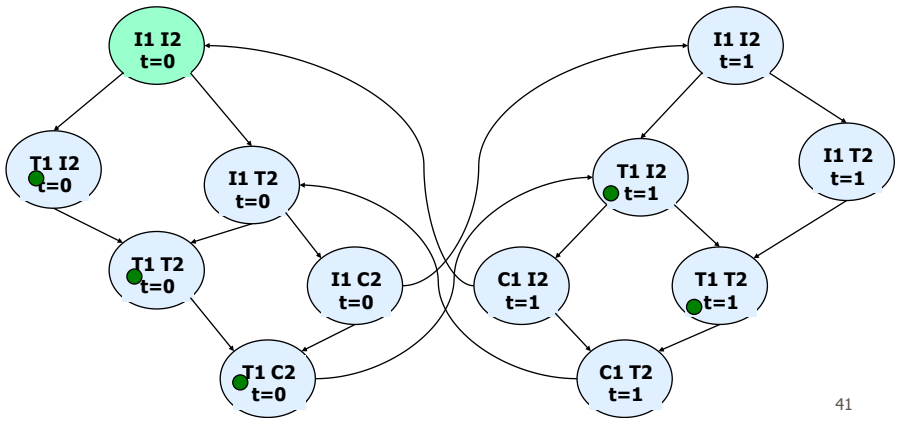
Compute Sat(T_1)

$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$



41

Properties of MUTEX example ?

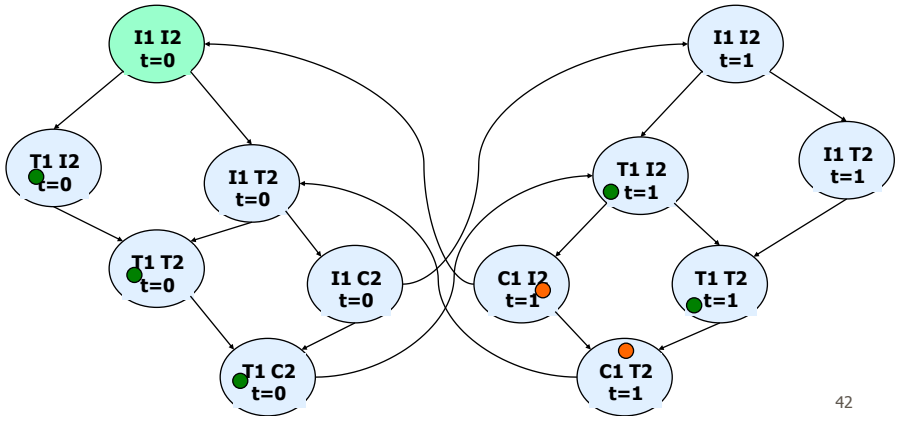
Compute Sat(C_1)

$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$



42

Properties of MUTEX example ?

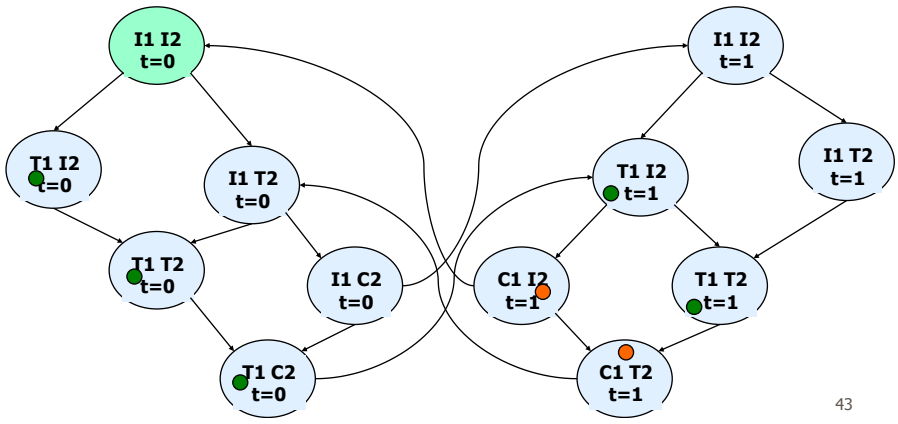
Compute Sat(AF C_1)

$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$



Properties of MUTEX example ?

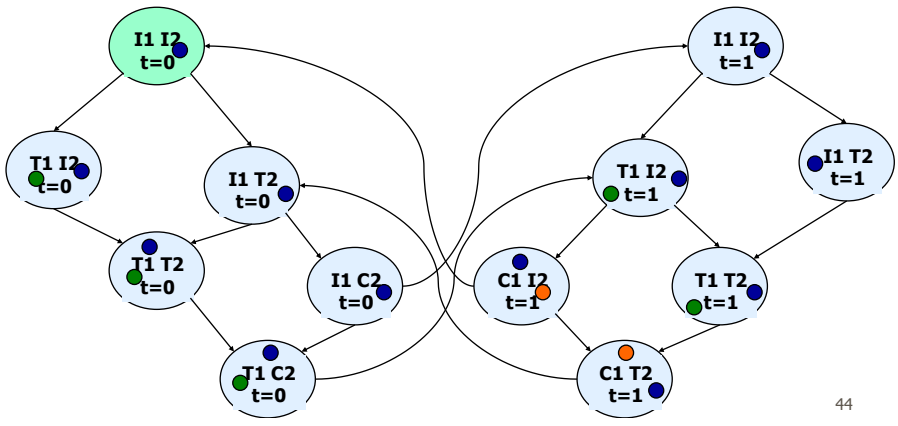
Compute Sat(AF C_1)

$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$



Properties of MUTEX example ?

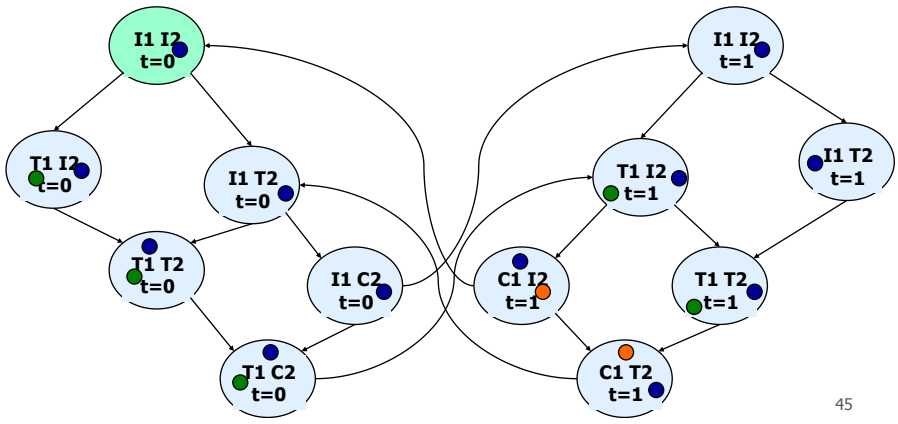
Sat(T_1 => AF C_1) ?

$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$



45

Properties of MUTEX example ?

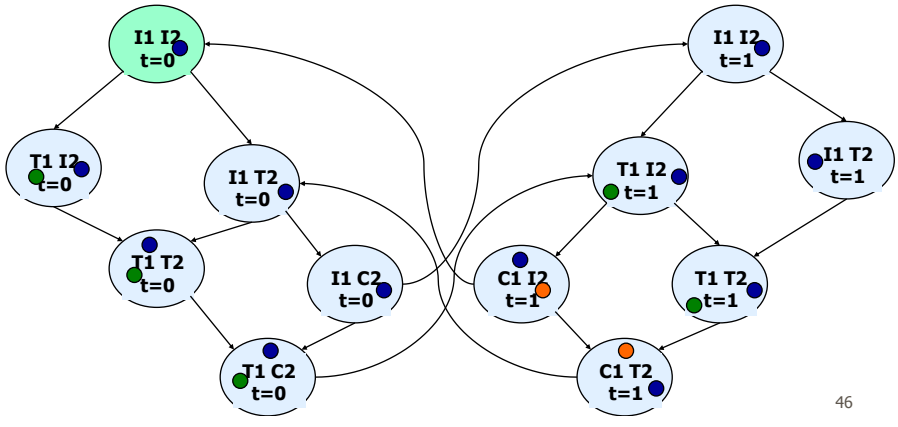
Sat(T_1 => AF C_1) = all states !!

$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$



46

Properties of MUTEX example ?

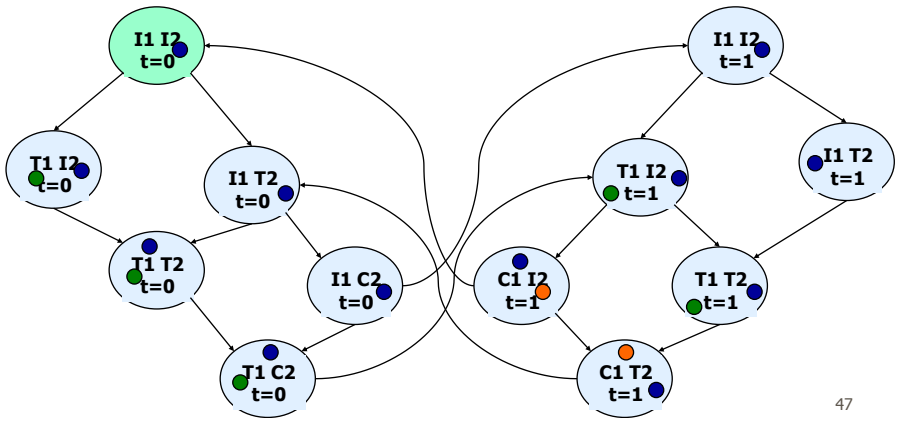
Sat(AG[T_1 => AF C_1]) = all states !!

$AG\ \neg(C_1 \wedge C_2)$

$AG[\ T_1 \Rightarrow AF(C_1)]$

$EG[\neg C_1]$

$AG[C_1 \Rightarrow A[C_1\ U\ (\neg C_1 \wedge A[\neg C_1\ U\ C_2])]]$

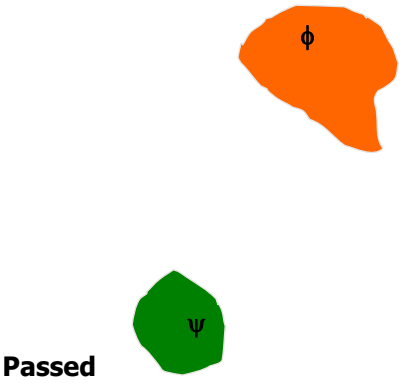


function $Sat(\phi : Formula)$: **set of** $State$;
(* precondition: true *)
begin
  if $\phi = \text{true} \longrightarrow$ **return** S
  $[]\ \phi = \text{false} \longrightarrow$ **return** $\varnothing$
  $[]\ \phi \in AP \longrightarrow$ **return** $\{\, s \mid \phi \in Label(s) \,\}$
  $[]\ \phi = \neg\phi_1 \longrightarrow$ **return** $S - Sat(\phi_1)$
  $[]\ \phi = \phi_1 \vee \phi_2 \longrightarrow$ **return** $(Sat(\phi_1) \cup Sat(\phi_2))$
  $[]\ \phi = \mathsf{EX}\,\phi_1 \longrightarrow$ **return** $\{\, s \in S \mid (s, s') \in R \ \wedge\ s' \in Sat(\phi_1) \,\}$
  $[]\ \phi = \mathsf{E}\,[\phi_1\ \mathsf{U}\ \phi_2] \longrightarrow$ **return** $Sat_{EU}(\phi_1, \phi_2)$
  $[]\ \phi = \mathsf{A}\,[\phi_1\ \mathsf{U}\ \phi_2] \longrightarrow$ **return** $Sat_{AU}(\phi_1, \phi_2)$
  fi
(* postcondition: $Sat(\phi) = \{\, s \mid \mathcal{M}, s \models \phi \,\}$ *)
end

How to Compute Sat(E[ϕ U ψ])

```
function Sat_EU (ϕ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
      Q, Q' := Sat(ψ), ∅;
      do Q ≠ Q' ⟶
         Q' := Q;
         Q := Q ∪ ({ s | ∃ s' ∈ Q. (s, s') ∈ R } ∩ Sat(ϕ))
      od;
      return Q
(* postcondition: Sat_EU(ϕ, ψ) = { s ∈ S | M, s ⊨ E[ϕ U ψ] } *)
end
```

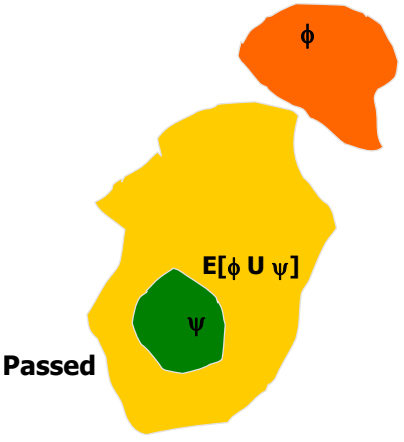
Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$



How to Compute Sat(E[ϕ U ψ])

```
function Sat_EU (ϕ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
      Q, Q' := Sat(ψ), ∅;
      do Q ≠ Q' ⟶
         Q' := Q;
         Q := Q ∪ ({ s | ∃ s' ∈ Q. (s, s') ∈ R } ∩ Sat(ϕ))
      od;
      return Q
(* postcondition: Sat_EU(ϕ, ψ) = { s ∈ S | M, s ⊨ E[ϕ U ψ] } *)
end
```

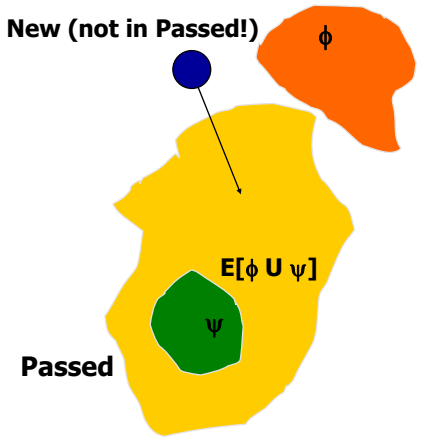
Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$



How to Compute Sat(E[ϕ U ψ])

```
function Sat_EU (ϕ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
      Q, Q' := Sat(ψ), ∅;
      do Q ≠ Q' →
         Q' := Q;
         Q := Q ∪ ({ s | ∃ s' ∈ Q. (s, s') ∈ R } ∩ Sat(ϕ))
      od;
      return Q
(* postcondition: Sat_EU(ϕ, ψ) = { s ∈ S | M, s ⊨ E[ϕ U ψ] } *)
end
```

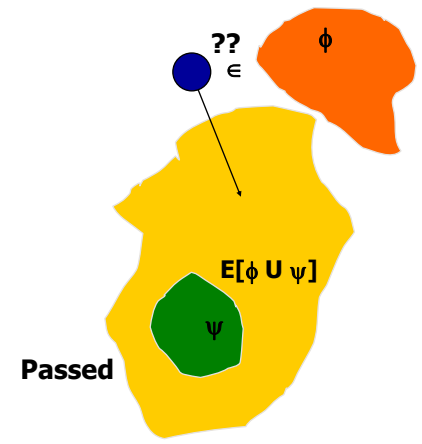
Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$



How to Compute Sat(E[ϕ U ψ])

```
function Sat_EU (ϕ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
      Q, Q' := Sat(ψ), ∅;
      do Q ≠ Q' →
         Q' := Q;
         Q := Q ∪ ({ s | ∃ s' ∈ Q. (s, s') ∈ R } ∩ Sat(ϕ))
      od;
      return Q
(* postcondition: Sat_EU(ϕ, ψ) = { s ∈ S | M, s ⊨ E[ϕ U ψ] } *)
end
```

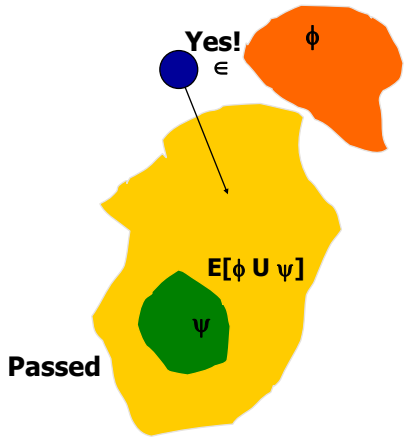
Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$



How to Compute Sat(E[ϕ U ψ])

```
function Sat_EU (φ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
      Q, Q' := Sat(ψ), ∅;
      do Q ≠ Q' →
         Q' := Q;
         Q := Q ∪ ({ s | ∃ s' ∈ Q. (s, s') ∈ R } ∩ Sat(φ))
      od;
      return Q
(* postcondition: Sat_EU(φ, ψ) = { s ∈ S | M, s ⊨ E[φ U ψ] } *)
end
```

Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$

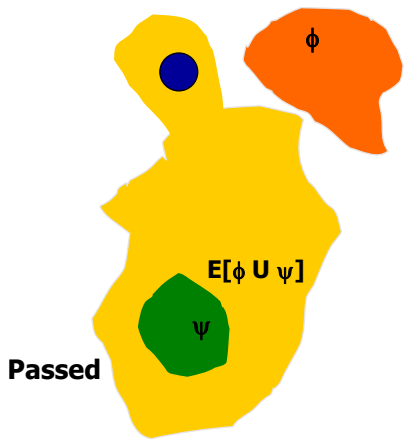


53

How to Compute Sat(E[ϕ U ψ])

```
function Sat_EU (φ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
      Q, Q' := Sat(ψ), ∅;
      do Q ≠ Q' →
         Q' := Q;
         Q := Q ∪ ({ s | ∃ s' ∈ Q. (s, s') ∈ R } ∩ Sat(φ))
      od;
      return Q
(* postcondition: Sat_EU(φ, ψ) = { s ∈ S | M, s ⊨ E[φ U ψ] } *)
end
```

Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$

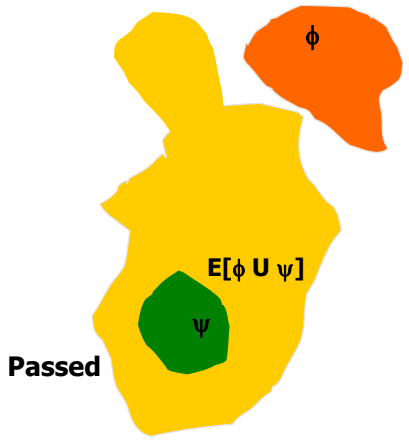


54

How to Compute Sat(E[ϕ U ψ])

```
function Sat_EU (ϕ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
      Q, Q' := Sat(ψ), ∅;
      do Q ≠ Q' →
            Q' := Q;
            Q := Q ∪ ({ s | ∃ s' ∈ Q. (s, s') ∈ R } ∩ Sat(ϕ))
      od;
      return Q
(* postcondition: Sat_EU(ϕ, ψ) = { s ∈ S | M, s ⊨ E[ϕ U ψ] } *)
end
```

Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$

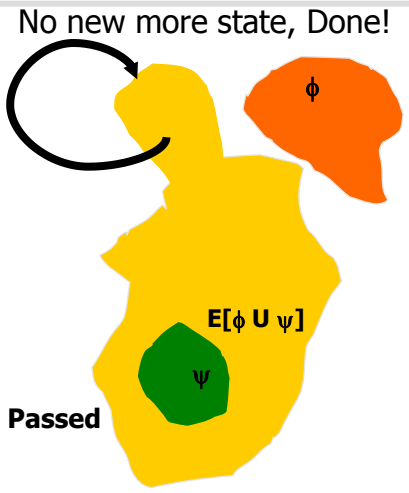


55

How to Compute Sat(E[ϕ U ψ])

```
function Sat_EU (ϕ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
      Q, Q' := Sat(ψ), ∅;
      do Q ≠ Q' →
            Q' := Q;
            Q := Q ∪ ({ s | ∃ s' ∈ Q. (s, s') ∈ R } ∩ Sat(ϕ))
      od;
      return Q
(* postcondition: Sat_EU(ϕ, ψ) = { s ∈ S | M, s ⊨ E[ϕ U ψ] } *)
end
```

Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$

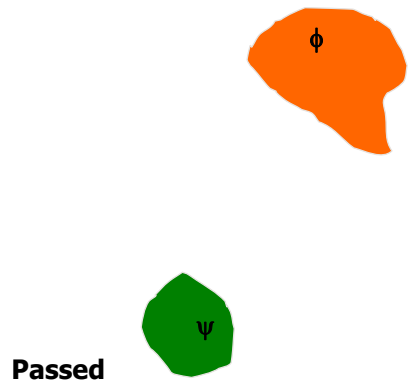


56

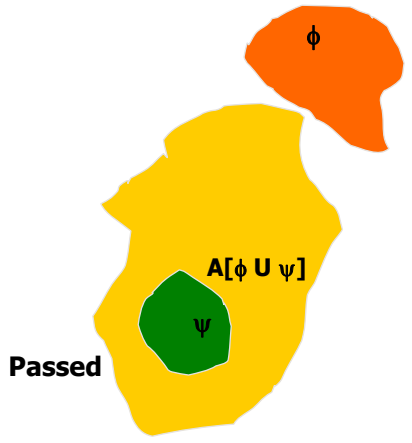
How to Compute Sat(A[ϕ U ψ])

?

How to Compute Sat(A[ϕ U ψ])

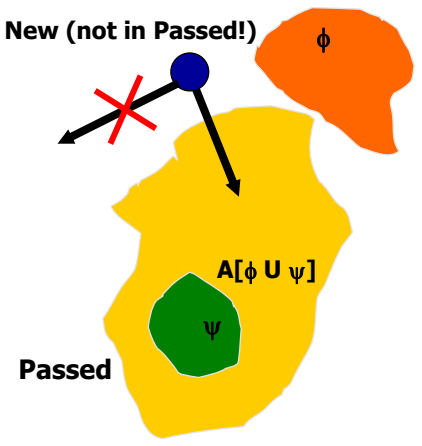


How to Compute Sat(A[ϕ U ψ])



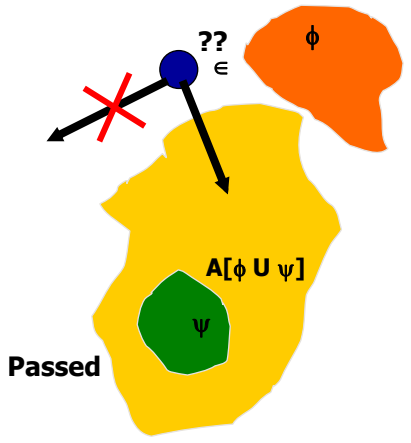
59

How to Compute Sat(A[ϕ U ψ])



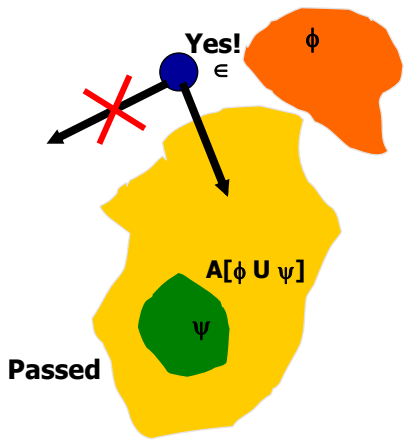
60

How to Compute Sat(A[ϕ U ψ])



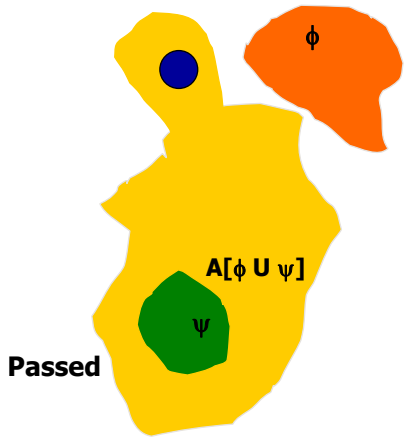
61

How to Compute Sat(A[ϕ U ψ])



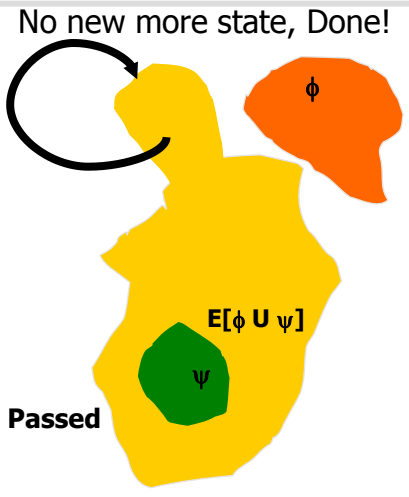
62

How to Compute Sat(A[ϕ U ψ])



63

How to Compute Sat(A[ϕ U ψ])



64

```
function Sat_AU (φ, ψ : Formula) : set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
        Q, Q' := Sat(ψ), ∅;
        do Q ≠ Q' ⟶
            Q' := Q;
            Q := Q ∪ ({s | ∀s'.(s, s') ∈ R ⇒ s' ∈ Q} ∩ Sat(φ))
        od;
        return Q
(* postcondition: Sat_AU(φ, ψ) = { s ∈ S | M, s ⊨ A[φ U ψ] } *)
end
```

Table 3.5: Labelling procedure for $\mathsf{A}[\phi \,\mathsf{U}\, \psi]$

Fixpoint Characterizations (SMV)

$$\mathsf{EF}\, p \quad \equiv \quad p \vee \mathsf{EXEF}\, p$$

Let A **be the set of states satisfying** EF p **then**

$$\mathsf{A} \quad \equiv \quad p \vee \mathsf{EXA}$$

in fact A **is the smallest one of sets satisfying the equations (the least fixpoint)**

Fixed points of monotonic functions

- Let τ be a function $S \to S$
- Say τ is *monotonic* when

$$x \subseteq y \text{ implies } \tau(x) \subseteq \tau(y)$$

- Fixed point of τ is y such that

$$\tau(y) = y$$

- If τ monotonic, then it has
  - least fixed point $\mu y.\ \tau(y)$
  - greatest fixed point $\nu y.\ \tau(y)$

Iteratively computing fixed points

- Suppose S is finite
  - The least fixed point $\mu y.\ \tau(y)$ is the limit of

$$\text{false} \subseteq \tau(\text{false}) \subseteq \tau(\tau(\text{false})) \subseteq \cdots$$

  - The greatest fixed point $\nu y.\ \tau(y)$ is the limit of

$$\text{true} \supseteq \tau(\text{true}) \supseteq \tau(\tau(\text{true})) \supseteq \cdots$$

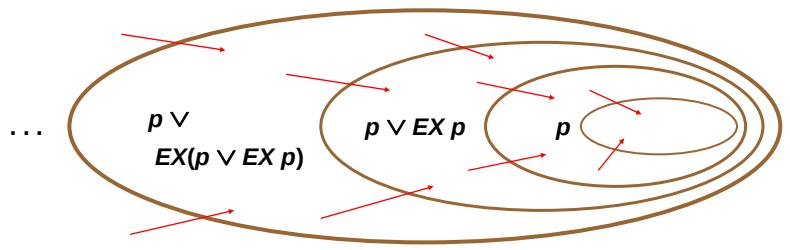
Note, since S is finite, convergence is finite

Example: *EF p*

- *EF p* is characterized by

$$EF\ p = \mu y.(p \vee EX\ y)$$

- Thus, it is the limit of the increasing series...



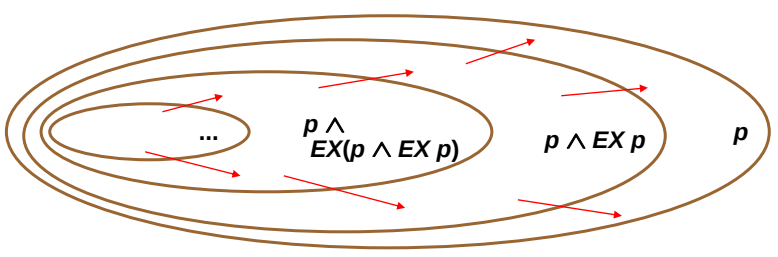
69

Example: *EG p*

- *EG p* is characterized by

$$EG\ p = \nu y.\ (p \wedge EX\ y)$$

- Thus, it is the limit of the decreasing series...

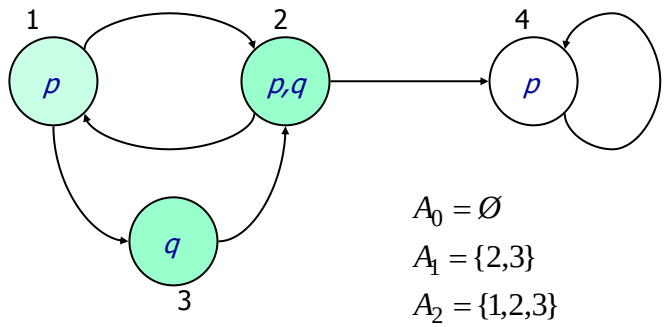


70

Example, continued

EF *q*

$$EF\, q = \mu y.(q \vee EX\ y)$$



$A_0 = \emptyset$

$A_1 = \{2,3\}$

$A_2 = \{1,2,3\}$

$A_3 = \{1,2,3\}$

Remaining operators

$$\begin{aligned} AF\ p &= \mu y.(p \vee AX\ y) \\ AG\ p &= \nu y.(p \wedge AX\ y) \\ E(p\,U\,q) &= \mu y.(q \vee (p \wedge EX\ y)) \\ A(p\,U\,q) &= \mu y.(q \vee (p \wedge AX\ y)) \end{aligned}$$

Complexity

The worst-case time complexity of checking whether system-model sys satisfies the CTL-*formula* ϕ *is* $\mathcal{O}(|\, S_{sys} \,|^2 \times |\, \phi \,|)$

However S_{sys} may be EXPONENTIAL in number of parallel components!
--
FIXPOINT COMPUTATIONS may be carried out using
ROBDD's
(Reduced Ordered Binary Decision Diagrams)
Bryant, 86

LTL: Linear Time Logics

defined on infinite traces (of transition systems with Buchi accepting conditions)

LTL, Linear-Time Logic

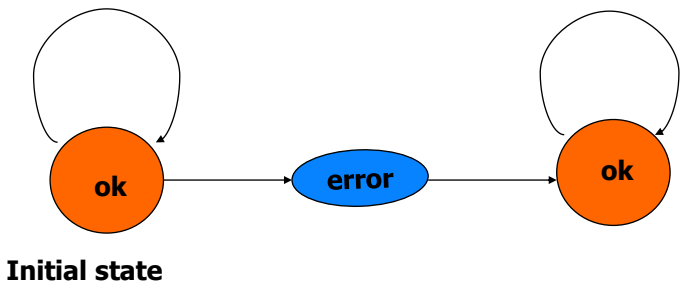
Syntax

ϕ **::=** **P** | $\neg\,\phi$ | $\phi \vee \phi$ | **EX** ϕ | ϕ **U** ϕ

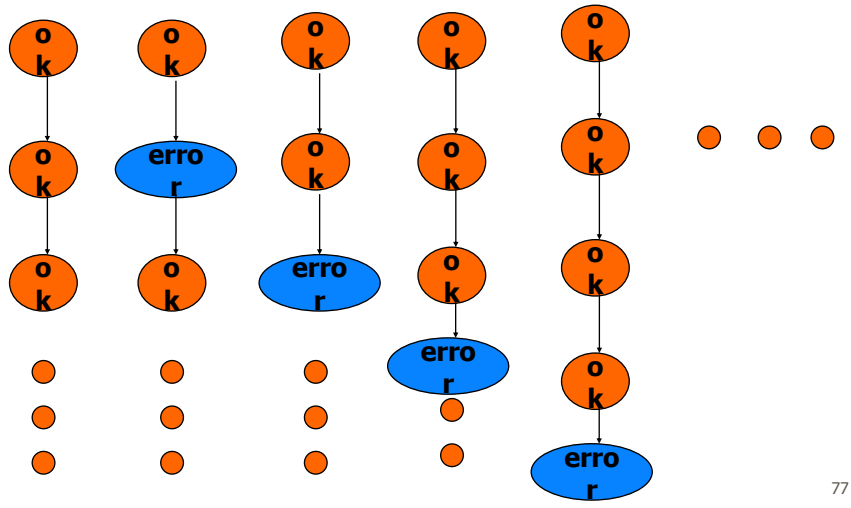
where **P** $\in$ AP (atomic propositions)

EX pronounced "n**EX**t state"
U pronounced "**U**ntil"

EXAMPLE: a BUGGY machine

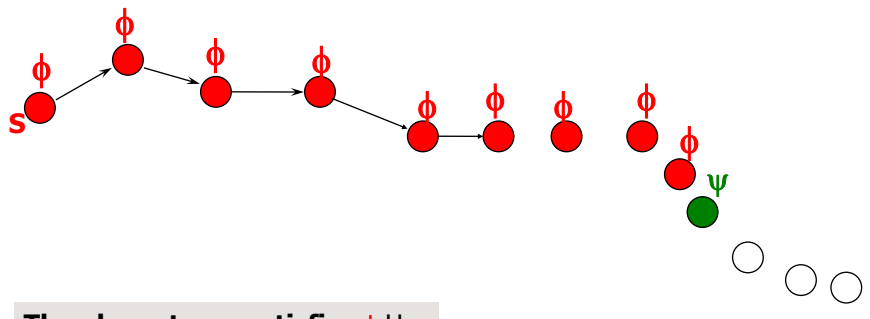


The linear-time behaviour of BUGGY



77

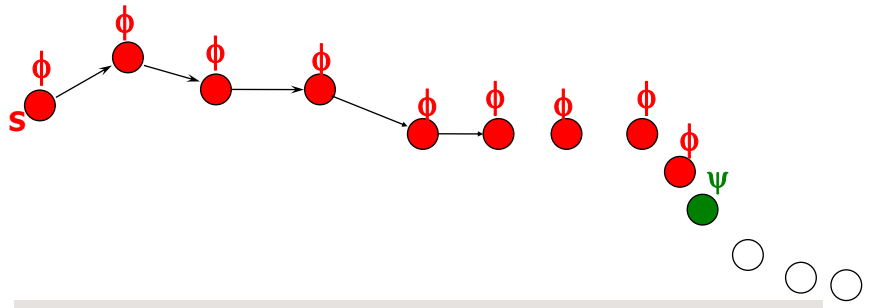
ϕ U ψ satisfied by a trace



The above trace satisfies ϕ U ψ

78

ϕ U ψ satisfied by a system (def.)



If all traces from initial state S of the system satisfies ϕ U ψ

Derived Operators

- $\Diamond\phi$ denotes (true U ϕ) *inevitablly*
- $\Box\phi$ denotes $\neg(\Diamond \neg \phi)$ *invariantly*/globally

Comparing CTL and LTL

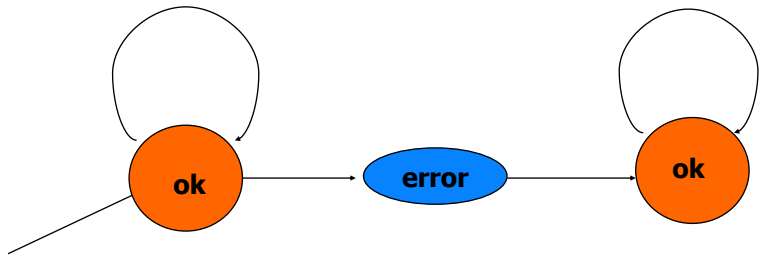
◊P (LTL) similar to AF p (CTL)
□p (LTL) similar to AG p (CTL)

However,

- LTL can't express reachability properties: EF P in CTL
- CTL can't express ◊□ **p** in LTL

- **CTL* = LTL + CTL**

Comparing CTL and LTL (contn.)

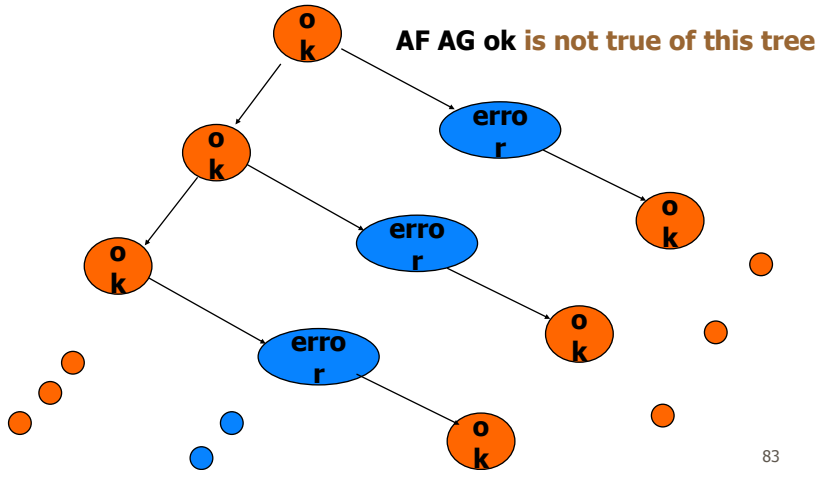


Satisfies ◊□ ok
but it does not satisfy AF AG ok

Why?

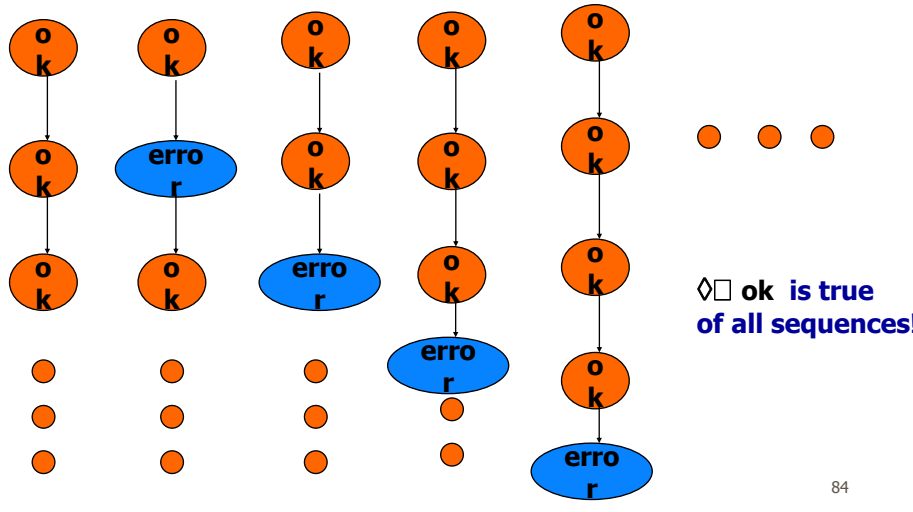
No subtree where ok is true everywhere

AF AG ok is not true of this tree



83

The linear-time behaviour of BUGGY



◊□ ok is true of all sequences!

84

Model Checking LTL [Wolper et al 1986]

- Given an automata M and a formula ϕ, to check M sat ϕ
  - Construct the formula automaton: $A(\neg \phi)$
  - Construct the product automaton $M \mid\mid A(\neg \phi)$ (on-the-fly)
- If $M \mid\mid A(\neg \phi)$ is empty then M sat ϕ otherwise NO
- Time-Complexity = $|M| * 2^{O(|\phi|)}$

The same idea can be used for CTL model checking using Tree-automata

END
(of Untimed Systems)