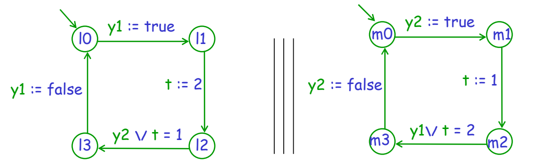
Lecture 1
Branching time logics and CTL

Peterson-Fischer: Possible Specifications

Variables: y1, y2: boolean, t: {1,2}
Initially y1 = y2 = false, t = 1

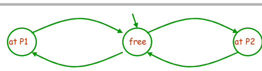


Mutual Exclusion: the two processes never simultaneously reach l3,m3

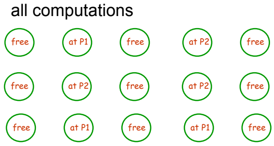
Absence of Starvation: If the left process is at l1, it will later reach l3

Bounded Overtaking: If the left process is at l1, the other process will reach m3 at most once (twice?) before the left process reaches l3

Two Classes of Formalizations

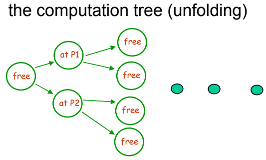


Linear Time Properties:
Specify properties that hold for all computations



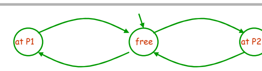
Free in every 2nd state ;
at P1 always followed by free

Branching Time Properties:
Specify properties that hold for the computation tree (unfolding)



always possible to reach free ;
always possible to reach at P1 ;
free is always eventually reached

Two Classes of Formalizations

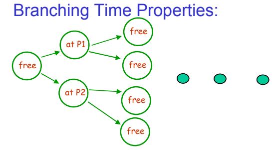


Linear Time Properties:

Specification says:
In all computations,
Free in every 2nd state ;

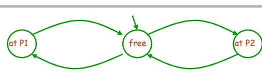
In all computations,
at P1 always followed by free

Branching Time Properties:



In all computations,
always
in some computation
possible to reach free

Two Classes of Formalizations

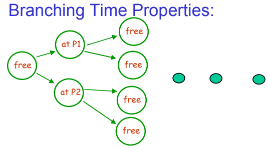


Linear Time Properties:

Specification says:
In all computations,
Free in every 2nd state ;

In all computations,
at P1 always followed by free

Branching Time Properties:



In all computations (paths),
always
in some computation (path)
possible to reach free

Structure of CTL Formulas

Operators in branching time logic typically have two components:
<in all/some (future) path(s)> <temporal property>

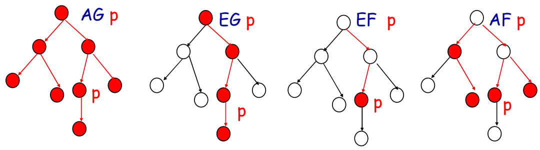
CTL syntax:
A/E G/F/U φ

Structure of CTL Formulas

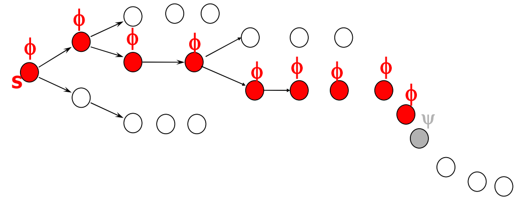
Operators in branching time logic typically have two components:
<in all/some (future) path(s)> <temporal property>

CTL syntax:
A/E G/F/U φ

Invariant **Potentially globally** **Possible** **Inevitable**

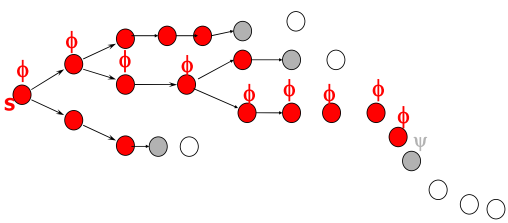


E[ϕ U ψ]



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

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]**

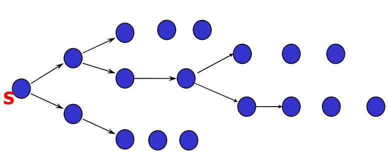
where **P** $\in$ AP (atomic propositions)

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

Computations vs. States

Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

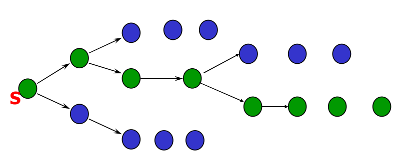
$\Diamond\Box$ p



Computations vs. States

Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

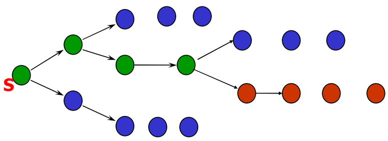
$\Diamond\Box$ p



Computations vs. States

Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

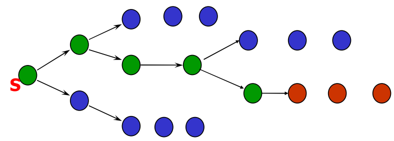
◇□ p



Computations vs. States

Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

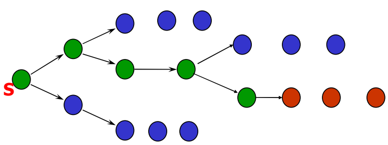
In branching time AF (? G p)



Computations vs. States

Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

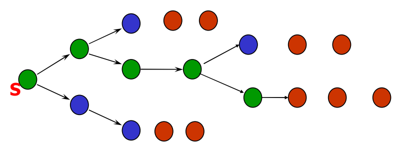
In branching time AF (A G p)



Computations vs. States

Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

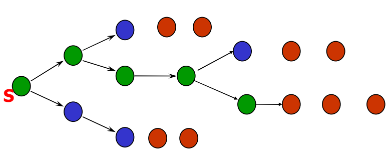
In branching time AF (A G p)



Computations vs. States

Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

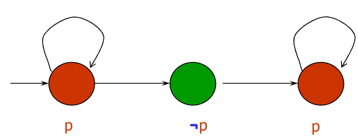
In branching time AF (A G p)
But then, we cannot let each computation "decide" when to start G p



Computations vs. States

Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

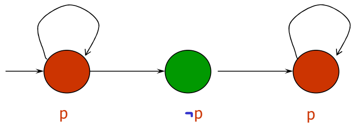
In branching time AF (A G p)
But then, we cannot let each computation "decide" when to start G p



Computations vs. States

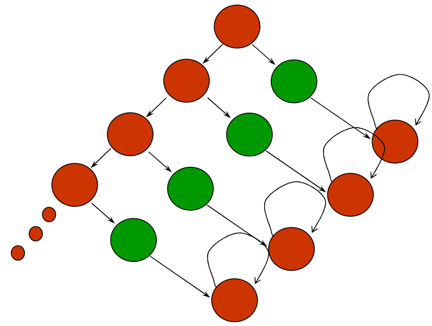
Linear time formula satisfied by state in certain computation
Branching time formula satisfied by state in the computation tree.
Meaning of nested formula different

In branching time AF (A G p)
But then, we cannot let each computation "decide" when to start G p



This Transition system satisfies ◇□ p
But not AF (A G p)

Why?



Formal Semantics of CTL

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

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

Theorem

All operators are derivable from

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

and boolean connectives

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

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)

Labeling Methods [Clarke et al 81]

- $Sat(\phi)$ = all states where ϕ is true

- Compute $Sat(\phi)$ recursively as follows:
  - For each sub-formula ϕ_i of ϕ, compute $Sat(\phi_i)$
  - This is easier: e.g. $Sat(p) = \{s \mid p \in Label(s)\}$
- Compose $Sat(\phi_i)$ to get $Sat(\phi)$

This is a **GLOBAL** model checking algorithm:

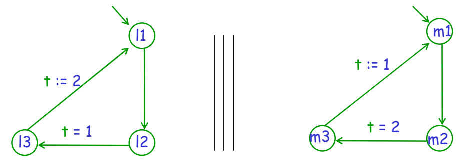
Compute all states that satisfy ϕ

Compare with **LOCAL** model checking algorithm:

Check whether some (e.g., initial) state satisfies ϕ

Simpler mutex

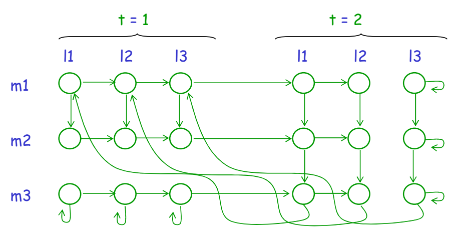
Variables: t: {1,2}
Initially t = 1



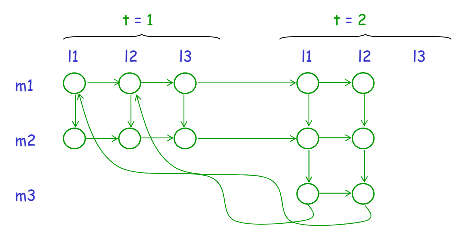
Mutual Exclusion: AG ¬(at l2 /\ at m3)

Absence of Starvation: AG (at l2 -> AF at l3)
Bounded Overtaking: AG(at l2 -> A[¬at m3 U A[at m3 U A[¬at m3 U at l3]]])

Simpler state space

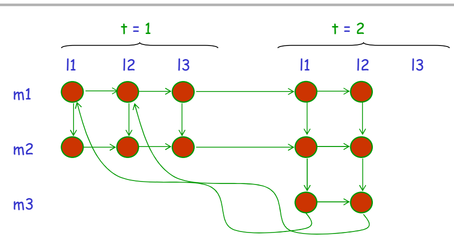


Simpler state space



Absence of Starvation: AG (at l2 -> AF at l3)

Simpler state space



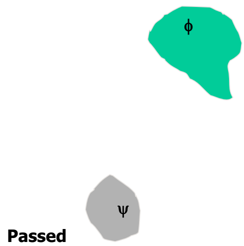
Absence of Starvation: AG (at l2 -> AF at l3)

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** $S - Sat(\phi_1)$
  [] $\phi = \phi_1 \lor \phi_2 \longrightarrow$ **return** $(Sat(\phi_1) \cup Sat(\phi_2))$
  [] $\phi = \mathsf{EX}\,\phi_1 \longrightarrow$ **return** $\{ s \in S \mid (s, s') \in R \ \land \ s' \in Sat(\phi_1) \}$
  [] $\phi = \mathsf{E}\,[\phi_1 \,\mathsf{U}\, \phi_2] \longrightarrow$ **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 $Sat_{EU}(\phi, \psi : Formula)$: set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
$Q, Q' := Sat(\psi), \varnothing$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \exists s' \in Q.(s,s') \in R\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{s \in S \mid \mathcal{M}, s \models E[\phi \, U \, \psi]\}$ *)
end

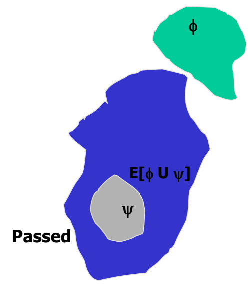
Table 3.4: Labelling procedure for E[ϕ U ψ]



How to Compute Sat(E[ϕ U ψ])

function $Sat_{EU}(\phi, \psi : Formula)$: set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
$Q, Q' := Sat(\psi), \varnothing$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \exists s' \in Q.(s,s') \in R\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{s \in S \mid \mathcal{M}, s \models E[\phi \, U \, \psi]\}$ *)
end

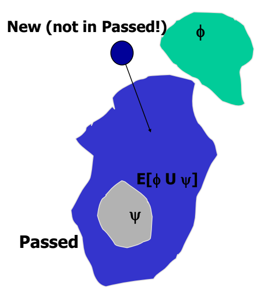
Table 3.4: Labelling procedure for E[ϕ U ψ]



How to Compute Sat(E[ϕ U ψ])

function $Sat_{EU}(\phi, \psi : Formula)$: set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
$Q, Q' := Sat(\psi), \varnothing$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \exists s' \in Q.(s,s') \in R\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{s \in S \mid \mathcal{M}, s \models E[\phi \, U \, \psi]\}$ *)
end

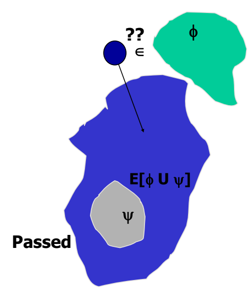
Table 3.4: Labelling procedure for E[ϕ U ψ]



How to Compute Sat(E[ϕ U ψ])

function $Sat_{EU}(\phi, \psi : Formula)$: set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
$Q, Q' := Sat(\psi), \varnothing$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \exists s' \in Q.(s,s') \in R\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{s \in S \mid \mathcal{M}, s \models E[\phi \, U \, \psi]\}$ *)
end

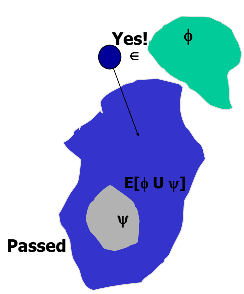
Table 3.4: Labelling procedure for E[ϕ U ψ]



How to Compute Sat(E[ϕ U ψ])

function $Sat_{EU}(\phi, \psi : Formula)$: set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
$Q, Q' := Sat(\psi), \varnothing$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \exists s' \in Q.(s,s') \in R\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{s \in S \mid \mathcal{M}, s \models E[\phi \, U \, \psi]\}$ *)
end

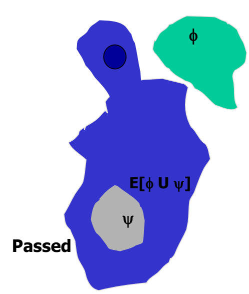
Table 3.4: Labelling procedure for E[ϕ U ψ]



How to Compute Sat(E[ϕ U ψ])

function $Sat_{EU}(\phi, \psi : Formula)$: set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
$Q, Q' := Sat(\psi), \varnothing$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \exists s' \in Q.(s,s') \in R\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{s \in S \mid \mathcal{M}, s \models E[\phi \, U \, \psi]\}$ *)
end

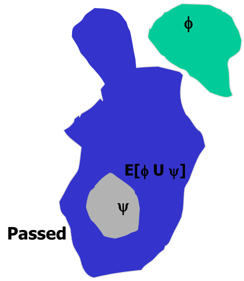
Table 3.4: Labelling procedure for E[ϕ U ψ]



How to Compute Sat(E[ϕ U ψ])

function $Sat_{EU}(\phi, \psi : Formula)$: set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
$Q, Q' := Sat(\psi), \emptyset$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \exists s' \in Q.(s, s') \in R\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{s \in S \mid \mathcal{M}, s \models \mathsf{E}[\phi \mathsf{U} \psi]\}$ *)
end

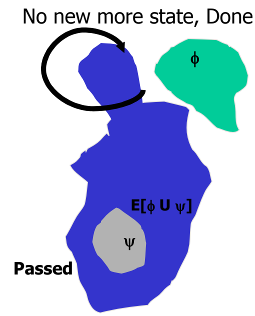
Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$



How to Compute Sat(E[ϕ U ψ])

function $Sat_{EU}(\phi, \psi : Formula)$: set of State;
(* precondition: true *)
begin var Q, Q' : set of State;
$Q, Q' := Sat(\psi), \emptyset$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \exists s' \in Q.(s, s') \in R\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{s \in S \mid \mathcal{M}, s \models \mathsf{E}[\phi \mathsf{U} \psi]\}$ *)
end

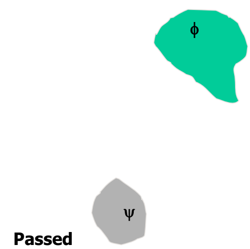
Table 3.4: Labelling procedure for $\mathsf{E}[\phi \mathsf{U} \psi]$



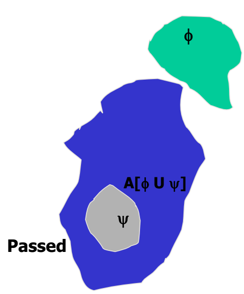
How to Compute Sat(A[ϕ U ψ])

?

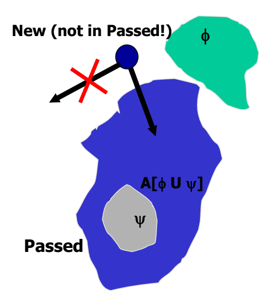
How to Compute Sat(A[ϕ U ψ])



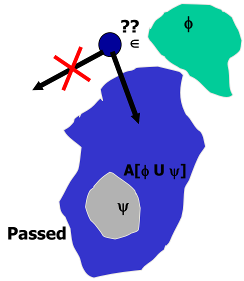
How to Compute Sat(A[ϕ U ψ])



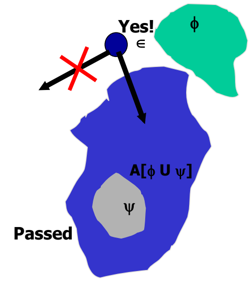
How to Compute Sat(A[ϕ U ψ])



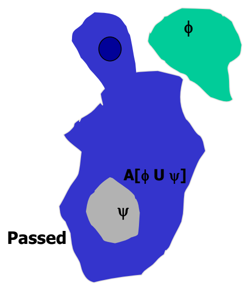
How to Compute Sat(A[ϕ U ψ])



How to Compute Sat(A[ϕ U ψ])

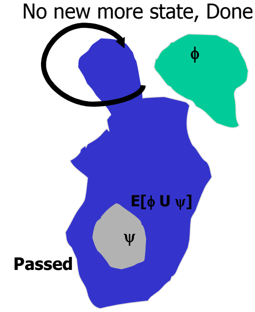


How to Compute Sat(A[ϕ U ψ])



How to Compute Sat(A[ϕ U ψ])

No new more state, Done!



function $Sat_{AU}\,(\phi, \psi : Formula)$: **set of** $State$;
(* precondition: true *)
begin var Q, Q' : **set of** $State$;
$\quad Q, Q' := Sat(\psi), \varnothing$;
$\quad$ **do** $Q \neq Q' \longrightarrow$
$\quad\quad Q' := Q$;
$\quad\quad Q := Q \cup (\{s \mid \forall s'.(s,s') \in R \Rightarrow s' \in Q\} \cap Sat(\phi))$
$\quad$ **od**;
$\quad$ **return** Q
(* postcondition: $Sat_{AU}(\phi, \psi) = \{ s \in S \mid \mathcal{M}, s \models \mathsf{A}[\phi \,\mathsf{U}\, \psi] \}$ *)
end

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

More efficient algorithm for A[φ U ψ]

The preceding algorithm has quadratic complexity.
We can do better by excluding states that satisfy

E[¬ψ W (¬φ /\ ¬ψ)]

1. Include states that satisfy ψ
2. Exclude states that satisfy ¬φ /\ ¬ψ
3. Exclude states that satisfy φ /\ ¬ψ and either
   1. can reach a state that satisfies ¬ φ /\ ¬ψ
   2. or are blocked
   3. or are in a non-trivial (with at least 2 states) strongly connected component in states that satisfy ¬ φ /\ ¬ψ

Fixpoint Characterizations (SMV)

$$\text{EF}\,p \;\equiv\; p \vee \text{EXEF}\,p$$

Let A be the set of states satisfying EF p then

$$\text{A} \;\equiv\; p \vee \text{EXA}$$

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 $S \rightarrow S$
- 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

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

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

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

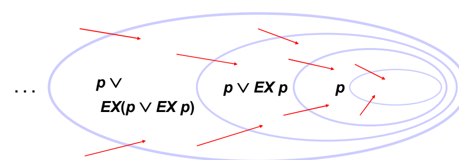
Note, since S is finite, convergence is finite

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...

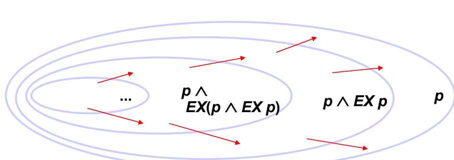


Example: *EG p*

- *EG p* is characterized by

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

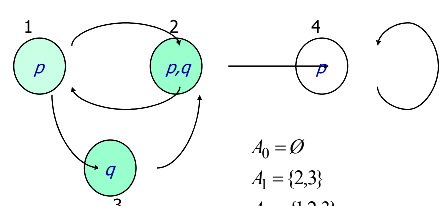
- Thus, it is the limit of the decreasing series...



Example, continued

EF *q*

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



$$A_0 = \varnothing$$
$$A_1 = \{2,3\}$$
$$A_2 = \{1,2,3\}$$
$$A_3 = \{1,2,3\}$$

Remaining operators

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

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

Course Material

You will **need**

- Lecture Handouts
- A few papers (will be distributed)
- SPIN documentation

Reference texts: You can from

- SPIN MODEL CHECKER Primer and Reference Manual by Gerard Holzmann, good textbook for the course.
- *Design and Validation of Computer Protocols*, G.J. Holzmann, Prentice Hall 1991, older book, can be downloaded from the net.
- Old notes prepared by me some years ago.

I will borrow a few slides by

- G. Holzmann
- J.-P. Katoen
- T. Ruys

You can inspect these slides yourself (see the course page).

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