Lecture 2

Model-Checking Finite-State Systems

(untimed systems)

Finite Automata, CTL, LTL and Model Checking

Finite state automata

- Finite graphs with labels on edges/nodes
  - a set of nodes (states)
  - a set of edges (transitions)
  - a set of labels (alphabet)

Complete Systems and Kripke Structure

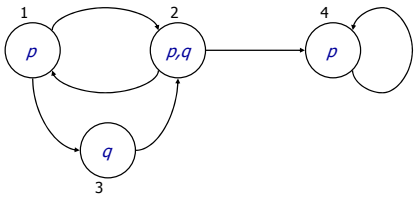
- From now on, we shall consider only Complete systems, that is, automata with labels on nodes.
  - There is no essential difference between models with labels on nodes or transitions
- This is the so called Kripke Structure, that is, automata with propositions labeled on states

CTL Models = *Kripke Structures*

A CTL-model is a triple $\mathcal{M} = (S, R, Label)$ where

- S is a non-empty set of states,
- $R \subseteq S \times S$ is a total relation on S, which relates to $s \in S$ its possible successor states,
- $Label : S \longrightarrow 2^{AP}$, assigns to each state $s \in S$ the atomic propositions $Label(s)$ that are valid in s.

Example



CTL: Computation Tree Logics

defined on Computation Trees of Kripke structures

Computation Tree Logic, CTL

Clarke & Emerson 1980

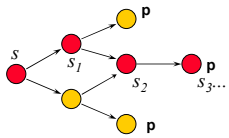
Syntax

$$\phi \;::=\; p \mid \neg\phi \mid \phi \vee \phi \mid \mathsf{EX}\,\phi \mid \mathsf{E}\,[\phi\,\mathsf{U}\,\phi] \mid \mathsf{A}\,[\phi\,\mathsf{U}\,\phi].$$

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

Path

Definition 20. (Path)
A *path* is an infinite sequence of states $s_0\, s_1\, s_2 \ldots$ such that $(s_i, s_{i+1}) \in R$ for all $i \geqslant 0$.



The set of path starting in s

$P_{\mathcal{M}}(s)$

Formal Semantics

(satisfaction relation $\models$)

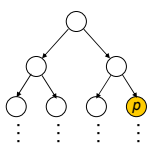
$s \models p$	iff $p \in Label(s)$
$s \models \neg\phi$	iff $\neg(s \models \phi)$
$s \models \phi \vee \psi$	iff $(s \models \phi) \vee (s \models \psi)$
$s \models \mathsf{EX}\,\phi$	iff $\exists\sigma \in P_{\mathcal{M}}(s).\,\sigma[1] \models \phi$
$s \models \mathsf{E}\,[\phi\,\mathsf{U}\,\psi]$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

CTL, Derived Operators

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

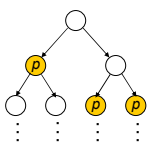
$\mathsf{AF}\,\phi \;\equiv\; \mathsf{A}\,[\mathrm{true}\,\mathsf{U}\,\phi].$ *inevitable*

EF p



E<> p in UPPAAL !

AF p



A<> p in UPPAAL

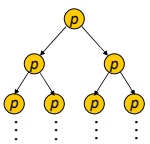
CTL, Derived Operators

$\mathsf{EG}\,\phi \;\equiv\; \neg\,\mathsf{AF}\,\neg\,\phi$ *potentially always*

$\mathsf{AG}\,\phi \;\equiv\; \neg\,\mathsf{EF}\,\neg\,\phi$ *always*

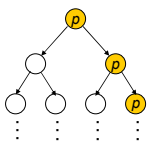
$\mathsf{AX}\,\phi \;\equiv\; \neg\,\mathsf{EX}\,\neg\,\phi.$

AG p



A[] p in UPPAAL

EG p



E[] p in UPPAAL

There are too many operators! But

We need to remember only the following:

(neXt time)
(Future, some time)
(Global)
(Until)

The most useful are EF, AG, EG and AF:

Theorem

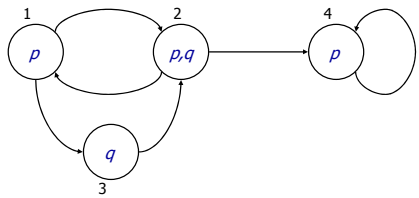
All operators are derivable from

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

and boolean connectives

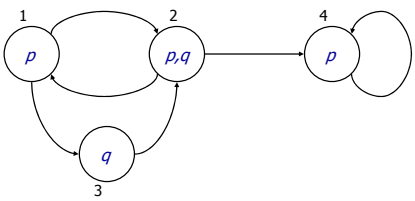
$$A[f \text{ U } g] \equiv \neg E[\neg g \text{ U} (\neg f \wedge \neg g)] \wedge \neg EG \neg g$$

Example



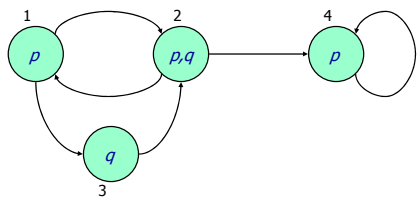
Example

EX p



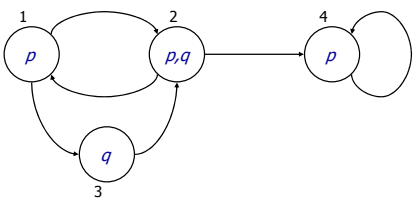
Example

EX p



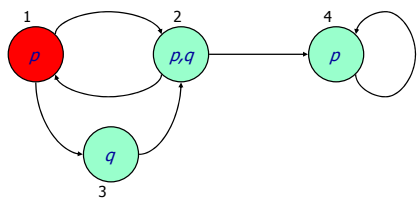
Example

AX p



Example

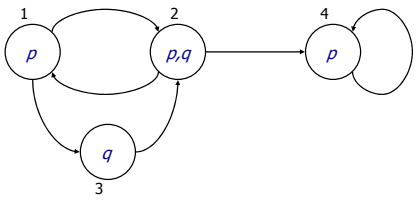
AX p



Note: state 1 doesn't satisfy AX p

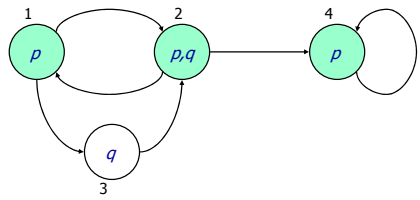
Example

EG *p*



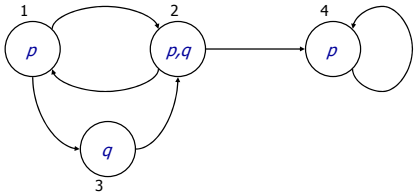
Example

EG *p*



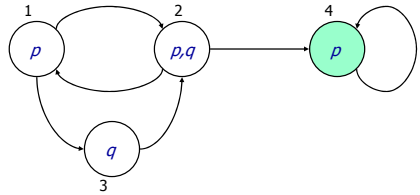
Example

AG *p*



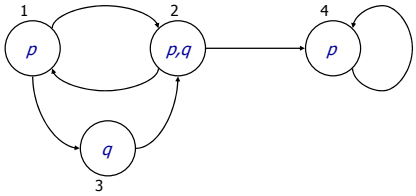
Example

AG *p*



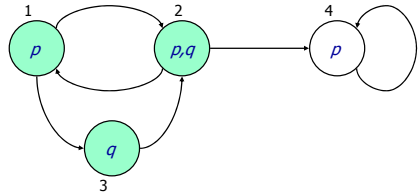
Example

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



Example

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



Properties of MUTEX example ?

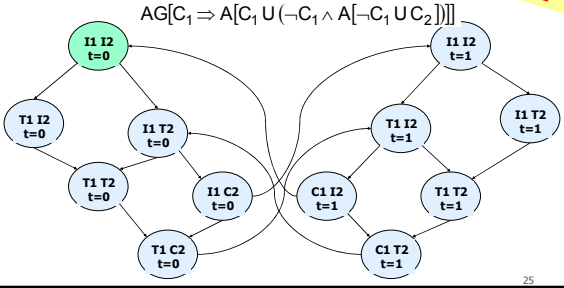
$AG\,\neg(C_1 \wedge C_2)$

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

$EG[\neg C_1]$

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

HOW to DECIDE IN GENERAL



CTL Model Checking Algorithms

Labeling Methods [Clarke et al 81]

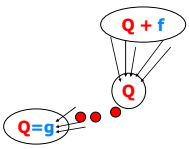
- Check all sub-formulas of F
- For each sub-formula f of F, label all nodes where f is true
- Check the composed formulas

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

Algorithm ideas for checking E(f U g)

- Mark all nodes where **f** is true and all nodes where **g** is true

- Start from all nodes where **g** is true and
- Perform backwards reachability analysis
- Each step backwards, store all nodes in Q where f is true
- Repeat the above step, until it converges
- Q contains all nodes satisfying E(f U g)

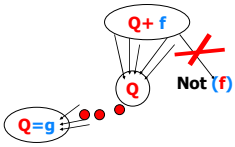


function $Sat_{EU}(\phi, \psi : Formula)$: **set of** $State$;
(* precondition: true *)
begin var Q, Q' : **set of** $State$;
  $Q, Q' := Sat(\psi), \varnothing$;
  do $Q \neq Q' \longrightarrow$
    $Q' := Q$;
    $Q := Q \cup (\{\, s \mid \exists\, s' \in Q.\,(s, s') \in R \,\} \cap Sat(\phi))$
  od;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

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

Algorithm ideas for checking A(f U g)

- Similar to the case for A(f U g)
- But each step backwards, store all nodes in Q where (f or g) is true, and the stored nodes do not lead to a node where (f or g) is false
- Repeat the above step, until it converges
- Q contains all nodes satisfying A(f U g)




```
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

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

or let A be the set of states satisfying EF p then

$$\mathsf{A} \quad \equiv \quad p \vee \mathsf{EX}\,\mathsf{A}$$

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

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

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

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

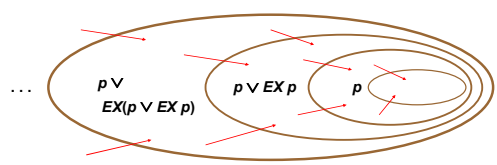
Note, since S is finite, convergence is finite

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...

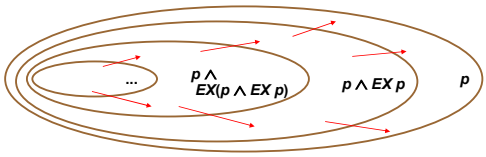


Example: *EG p*

- *EG p* is characterized by

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

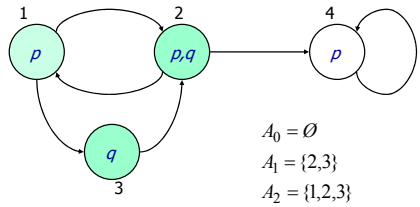
- Thus, it is the limit of the decreasing series...



Example, continued

EF *q*

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



$A_0 = \emptyset$

$A_1 = \{2,3\}$

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

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

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

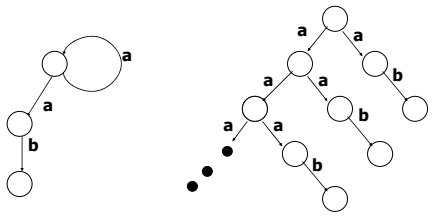
Something more about
Finite State Automata
and Temporal Logics

(Continuation of Lecture 2)

Branching time semantics

- Computation tree of an automaton is the unfolding of the automaton

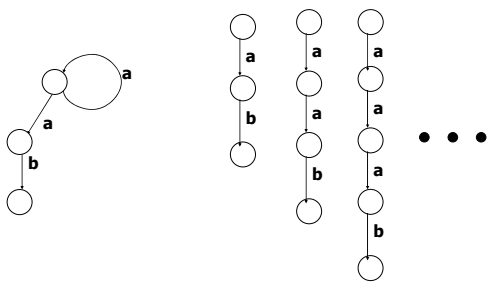
Example (Branching Time)



Linear Time Semantics

- Sequences of transitions (or states)
  - set of possible excecutions of a system
- Suite best for closed systems

Example (Linear Time)



Equivalences and Preorders

- A equivalent to B if the tree of A is identical to the tree of B (Too strong!)
- A is simulated by B if every transition of A is simulated by a transition of B (simulation [Milner78])
- A and B are bisimular if there is a symmetrical simulation between A's and B's states (bisimulation [Milner80])
- A and B are testing equivalent if they can pass the same set of tests (may and must testing [Nicola and Hennessy 84])
- A and B trace-equivalent if they provide the same set of sequences of transitions (trace equivalence [Hoare76])

LTL: Linear Time Logics

defined on infinite traces of Kripke structures with accepting conditions

Models: Infinite Sequences (ω-language accepted by automata)

- Automata with accepting conditions
  - Buchi, Muller automata
- Infininte accepted sequences of transitions as semantics of automata

LTL: Syntax

- P
- not F
- F1 and F2
- **O** F (next time)
- F1 **U** F2 (Until)

LTL: semantics

- assume an automaton M
  - a sequence of M: t=s(0)→s(1)→s(2)→...→s(i) ... ...
  - The set of sequences of M is Comp(M)

- s(i) sat p if p is a label of s(i)
- s(i) sat not F if not (s(i) sat F)
- s(i) sat F1 and F2 if s(i) sat F1 and s(i) sat F2
- s(i) sat **O** F if s(i+1) sat F
- s(i) sat F1 **U** F2 if s(k) sat F2 for some k=>i and s(j) sat F1 for all j such that i<=j<k

LTL: semantics (contn.)

- assume an automaton M
  - a sequence of M: t=s(0)→s(1)→s(2)→...→s(i) ... ...
  - The set of sequences of M is Comp(M)

- t sat F iff s(0) sat F
- M sat F iff t sat F for all sequences t of Comp(M)

Derived Operators

- <>F denotes (true **U** F)
- [ ]F denotes not (<> not F)
- F1 **W** F2 denotes (F1 **U** F2) or [ ]F1 (weak Until-operator)

Model Checking LTL [Wolper et al 1986]

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

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

Comparing CTL and LTL

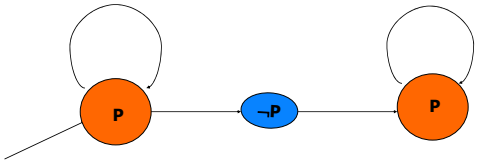
- <> P (LTL) similar AF p (CTL)
  [] p (LTL) similar AG p (CTL)

However,

- **LTL cannot express possibilities properties: EF P**
- **CTL cannot express <>[] p**

- **CTL* = LTL + CTL**

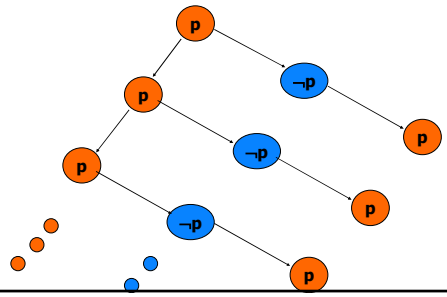
Comparing CTL and LTL (contn.)



Satisfies <>[] p
but it does not satisfy AF AG p

Why?

No subtree where p is true everywhere



END
(Finite State Untimed Systems)