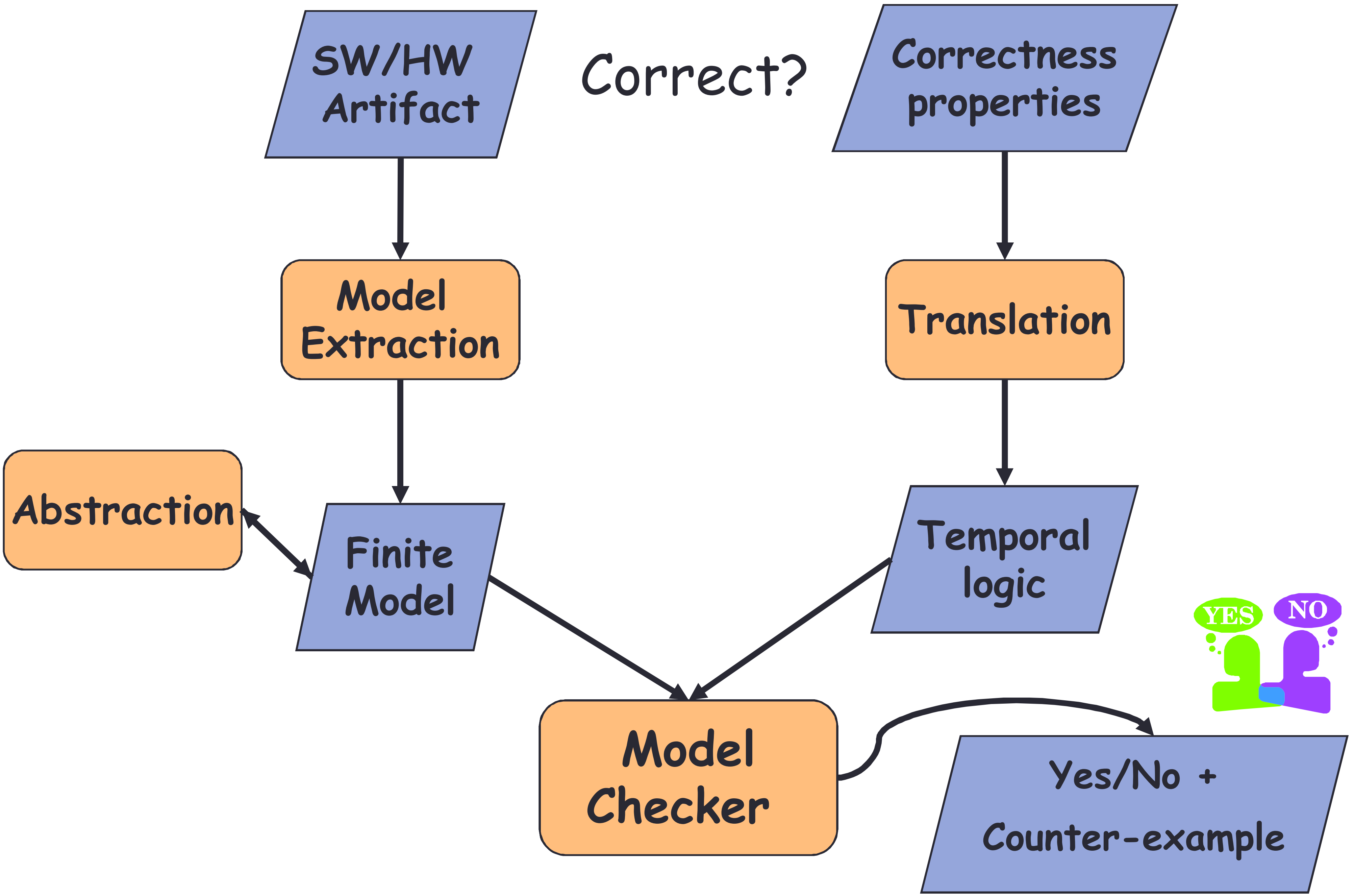
MODEL CHECKING

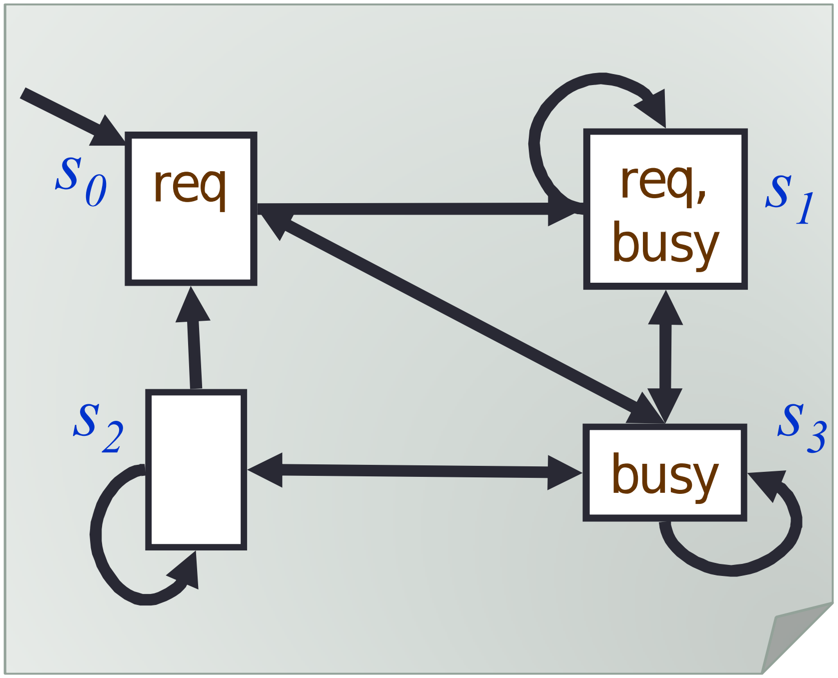
Arie Gurfinkel

Overview

- Kripke structures as models of computation
- CTL, LTL and property patterns
- CTL model-checking and counterexample generation
- State of the Art Model-Checkers


SW/HW Artifact
Correct?
Correctness properties
Model Extraction
Translation
Abstraction
Finite Model
Temporal logic
YES
NO
Model Checker
Yes/No +
Counter-example

Models: Kripke Structures



Conventional state machines

- $K = (V, S, s_0, I, R)$
- V is a (finite) set of atomic propositions
- S is a (finite) set of states
- $s_0 \in S$ is a start state
- $I: S \rightarrow 2^V$ is a labelling function that maps each state to the set of propositional variables that hold in it
  - That is, $I(S)$ is a set of interpretations specifying which propositions are true in each state
- $R \subseteq S \times S$ is a transition relation

Propositional Variables

Fixed set of atomic propositions, e.g, *{p, q, r}*

Atomic descriptions of a system

“Printer is busy”

“There are currently no requested jobs for the printer”

“Conveyer belt is stopped”

Do not involve time!

Modal Logic

Extends *propositional logic* with modalities to qualify propositions

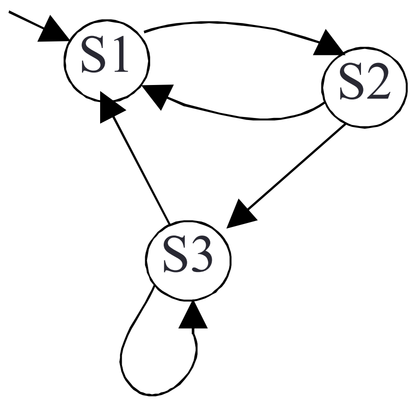
- *"it is raining" –* *rain*
- "it will rain tomorrow" – $\square$*rain*
  - it is raining in all possible futures
- "it might rain tomorrow" – $\Diamond$*rain*
  - it is raining in some possible futures

Modal logic formulas are interpreted over a collection of *possible worlds* connected by an *accessibility relation*

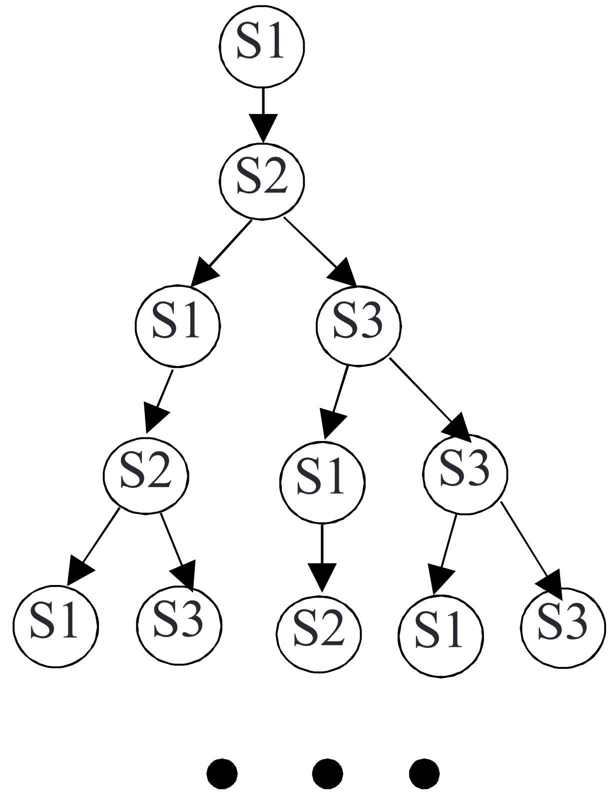
Temporal logic is a modal logic that adds temporal modalities: next, always, eventually, and until

Computation Tree Logic (CTL)

CTL: Branching-time propositional temporal logic
Model - a tree of computation paths



Kripke Structure



Tree of computation

CTL: Computation Tree Logic

Propositional temporal logic with explicit quantification over possible futures

Syntax:

True and *False* are CTL formulas;
propositional variables are CTL formulas;

If φ and ψ are CTL formulae, then so are: $\neg \varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$

EX φ : φ holds in some next state

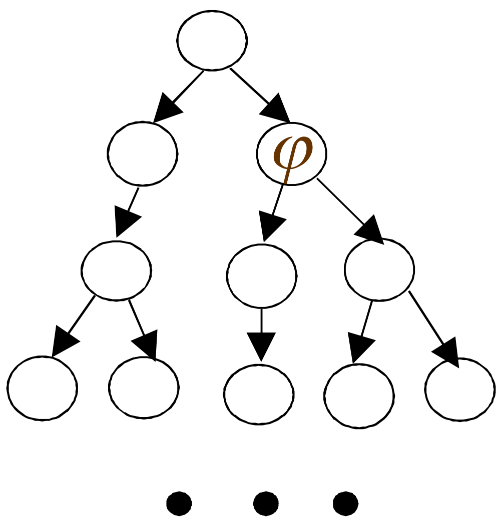
EF φ : along some path, φ holds in a future state

E[φ U ψ] : along some path, φ holds until ψ holds

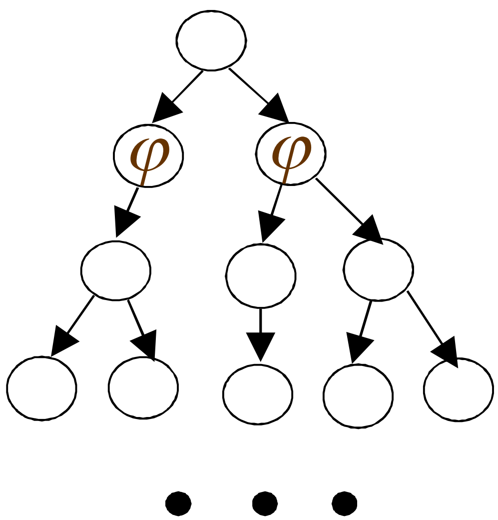
EG φ : along some path, φ holds in every state

- Universal quantification: AX φ , AF φ , A[φ U ψ], AG φ

Examples: EX and AX

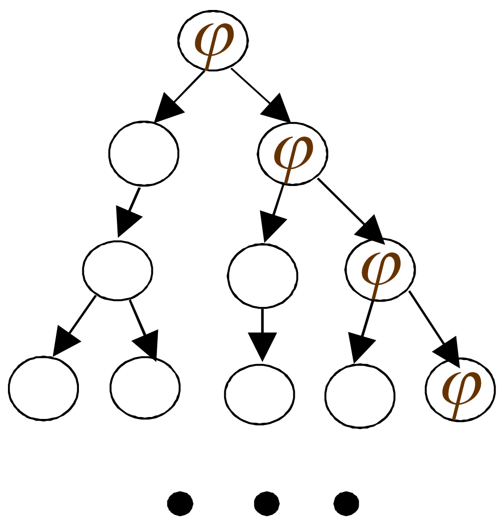


EX φ (exists next)

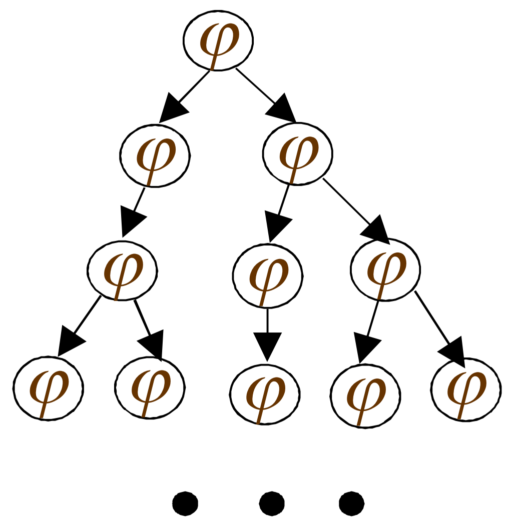


AX φ (all next)

Examples: EG and AG

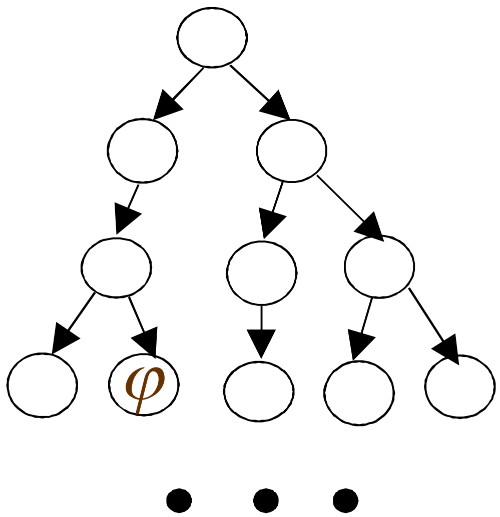


EG φ (exists global)

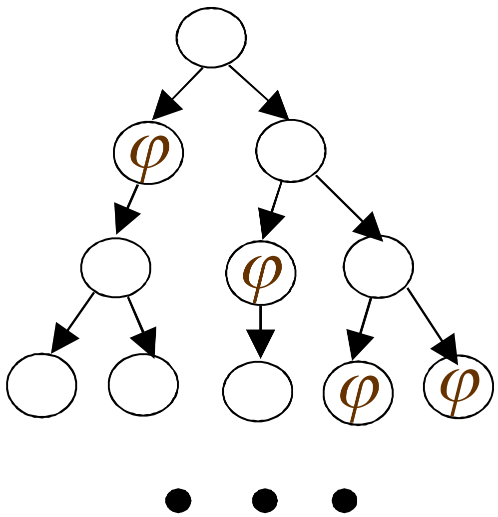


AG φ (all global)

Examples: EF and AF

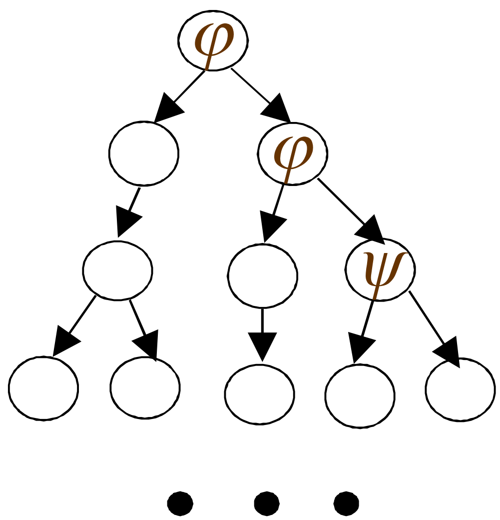


EF φ (exists future)

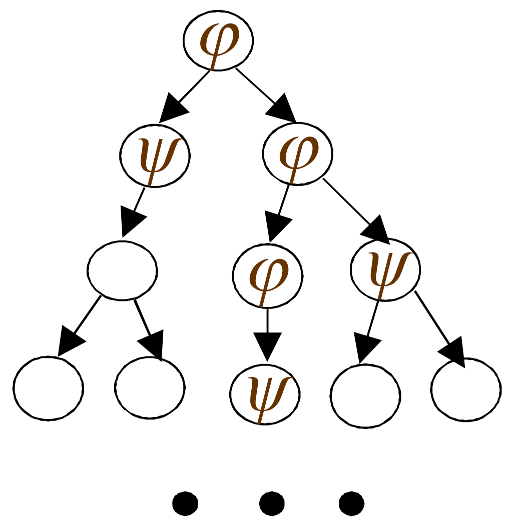


AF φ (all future)

Examples: EU and AU



E[φ U ψ] (exists until)



A[φ U ψ] (all until)

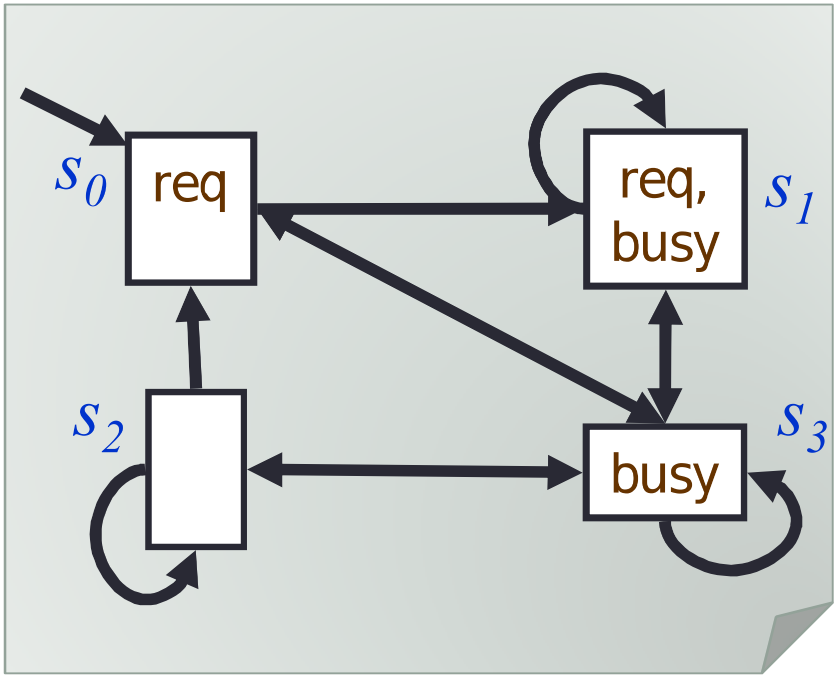
CTL Examples

Properties that hold:

- (AX busy)(s_0)
- (EG busy)(s_3)
- A (req U busy) (s_0)
- E (¬req U busy) (s_1)
- AG (req ⇒ AF busy) (s_0)

Properties that fail:

- (AX (req ∨ busy))(s_3)



Some Statements To Express

- An elevator can remain idle on the third floor with its doors closed
- When a request occurs, it will eventually be acknowledged
- A process is enabled infinitely often on every computation path
- A process will eventually be permanently deadlocked
- Action *s* precedes *p* after *q*
  - Note: hard to do correctly. See later on helpful techniques

Semantics of CTL

$K, s \vDash \varphi$ – means that formula φ is true in state s. K is often omitted since we always talk about the same Kripke structure

- E.g., $s \vDash p \wedge \neg q$

$\pi = \pi^0 \pi^1 \ldots$ is a path

π^0 is the current state (root)

π^{i+1} is a successor state of π^i. Then,

$\text{AX}\ \varphi = \forall \pi \cdot \pi^1 \vDash \varphi$

$\text{EX}\ \varphi = \exists \pi \cdot \pi^1 \vDash \varphi$

$\text{AG}\ \varphi = \forall \pi \cdot \forall i \cdot \pi^i \vDash \varphi$

$\text{EG}\ \varphi = \exists \pi \cdot \forall i \cdot \pi^i \vDash \varphi$

$\text{AF}\ \varphi = \forall \pi \cdot \exists i \cdot \pi^i \vDash \varphi$

$\text{EF}\ \varphi = \exists \pi \cdot \exists i \cdot \pi^i \vDash \varphi$

$\text{A}[\varphi\ \text{U}\ \psi] = \forall \pi \cdot \exists i \cdot \pi^i \vDash \psi \wedge \forall j \cdot 0 \leq j < i \Rightarrow \pi^j \vDash \varphi$

$\text{E}[\varphi\ \text{U}\ \psi] = \exists \pi \cdot \exists i \cdot \pi^i \vDash \psi \wedge \forall j \cdot 0 \leq j < i \Rightarrow \pi^j \vDash \varphi$

Relationship Between CTL Operators

$\neg AX\varphi = EX\, \neg\varphi$

$\neg AF\varphi = EG\, \neg\varphi$ $\qquad$ $\neg EF\varphi = AG\, \neg\varphi$

$AF\varphi = A[\text{true}\ U\ \varphi]$ $\qquad$ $EF\varphi = E[\text{true}\ U\ \varphi]$

$AG\ \varphi = \varphi \land AX\ AG\ \varphi$ $\qquad$ $EG\ \varphi = \varphi \land EX\ EG\ \varphi$

$AF\ \varphi = \varphi \lor AX\ AF\ \varphi$ $\qquad$ $EF\ \varphi = \varphi \lor EX\ EF\ \varphi$

$A\ [\text{false}\ U\ \varphi] = E[\text{false}\ U\ \varphi] = \varphi$

$A[\varphi\ U\ \psi] = \neg\ E[\neg\psi\ U\ (\neg\varphi \land \neg\psi)] \ \land\ \neg EG\, \neg\psi$

$A[\varphi\ U\ \psi] = \psi \lor (\varphi \land AX\ A[\varphi\ U\ \psi])$

$E[\varphi\ U\ \psi] = \psi \lor (\varphi \land EX\ E[\varphi\ U\ \psi])$

$A[\varphi\ W\ \psi] = \neg\ E[\neg\psi\ U\ (\neg\varphi \land \neg\psi)]$ (weak until)

$E[\varphi\ U\ \psi] = \neg\ A[\neg\psi\ W\ (\neg\varphi \land \neg\psi)]$

Adequate Sets

Def. A set of connectives is adequate if all connectives can be expressed using it.

- e.g., {¬,∧} is adequate for propositional logic:
  - $a \lor b = \neg (\neg a \land \neg b)$

Theorem. The set of operators {false,¬, ∧} together with EX, EG, and EU is adequate for CTL

  - e.g., AF (a ∨ AX b) = ¬ EG ¬ (a ∨ AX b) = ¬ EG (¬a ∧ EX ¬b)
- EU describes reachability
- EG – non-termination (presence of infinite behaviours)

Universal and Existential CTL

- A CTL formula is in ACTL if it uses only universal temporal connectives (AX, AF, AU, AG) with negation applied to the level of atomic propositions
  - Also called "*universal*" CTL formulas
  - e.g., A [*p* U AX ¬*q*]
- ECTL: uses only existential temporal connectives (EX, EF, EU, EG) with negation applied to the level of atomic propositions
  - Also called "*existential*" CTL formulas
  - e.g., E [*p* U EX ¬*q*]
- CTL formulas not in ECTL ∪ ACTL are called "mixed"
  - e.g., E [*p* U AX ¬*q*] and A [*p* U EX ¬*q*]

Safety and Liveness

Safety: Something "bad" will never happen

- AG ¬bad
- e.g., mutual exclusion: no two processes are in their critical section at once
- Safety = if false then there is a finite counterexample

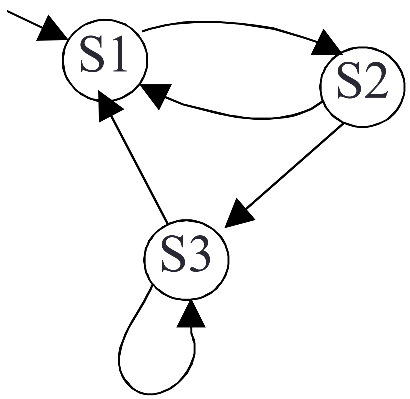
Liveness: Something "good" will always happen

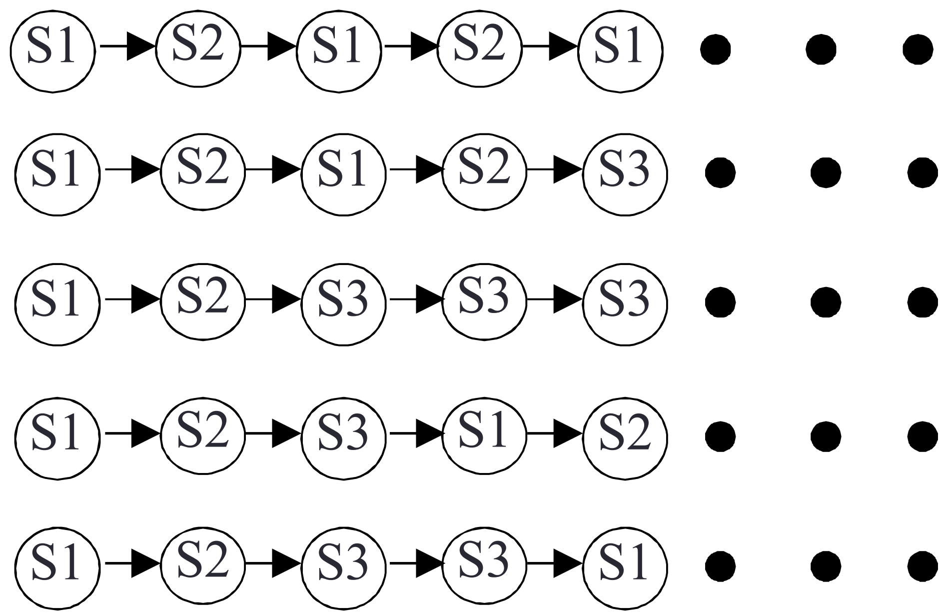
- AG AF good
- e.g., every request is eventually serviced
- Liveness = if false then there is an infinite counterexample

Every universal temporal logic formula can be decomposed into a conjunction of safety and liveness

Linear Temporal Logic (LTL)

For reasoning about complete traces through the system





Allows to make statements about a trace

LTL Syntax

- If φ is an atomic propositional formula, it is a formula in LTL
- If φ and ψ are LTL formulas, so are $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg \varphi$, φ U ψ (until), X φ (next), Fφ (eventually), G φ (always)
- Interpretation: over computations $\pi: \omega \Rightarrow 2^V$ which assigns truth values to the elements of V at each time instant

$\pi \vDash \mathrm{X}\,\varphi$ iff $\pi^{1} \vDash \varphi$

$\pi \vDash \mathrm{G}\,\varphi$ iff $\forall i \cdot \pi^{i} \vDash \varphi$

$\pi \vDash \mathrm{F}\varphi$ iff $\exists i \cdot \pi^{i} \vDash \varphi$

$\pi \vDash \varphi\,\mathrm{U}\,\psi$ iff $\exists i \cdot \pi^{i} \vDash \psi \wedge \forall j \cdot 0 \leq j < i \Rightarrow \pi^{j} \vDash \varphi$

Here, π^{i} is the i ’th state on a path

Properties of LTL

$$\neg X \varphi = X \neg \varphi$$

$$F \varphi = \text{true } U \varphi$$

$$G \varphi = \neg F \neg \varphi$$

$$G \varphi = \varphi \wedge X G \varphi$$

$$F \varphi = \varphi \vee X F \varphi$$

$$\varphi W \psi = G \varphi \vee (\varphi U \psi) \quad \text{(weak until)}$$

A property holds in a model if it holds on every path starting from the initial state

Expressing Properties in LTL

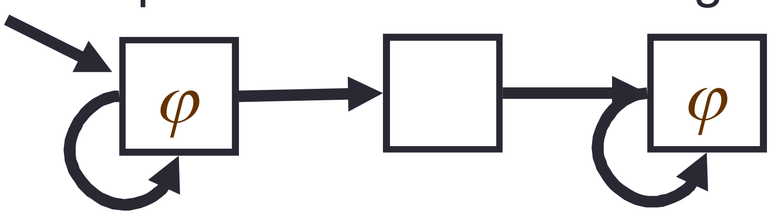
- Good for safety (G ¬) and liveness (F) properties
- Express:
  - When a request occurs, it will eventually be acknowledged
  - Each path contains infinitely many *q*'s
  - At most a finite number of states in each path satisfy ¬*q* (or property *q* eventually stabilizes)
  - Action *s* precedes *p* after *q*
    - Note: Hard to do correctly. See later on helpful techniques

Comparison between LTL and CTL

Syntactically: LTL is simpler than CTL

Semantically: incomparable!

- CTL formula AG EF φ (always can reach) is not expressible in LTL
- LTL formula F G φ (eventually always) is not expressible in CTL
  - What about AF AG φ?
  - Has different interpretation on the following state machine:



  - AF AG φ is false
  - F G φ is true

The logic CTL* is a super-set of both CTL and LTL

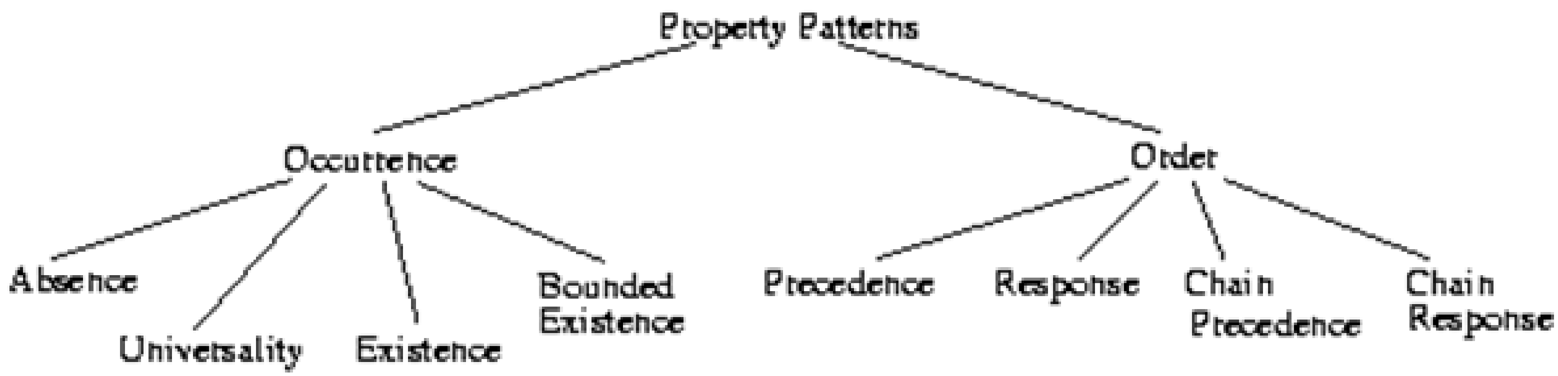
LTL and CTL coincide if the model has only one path!

Property Patterns: Motivation

- Temporal properties are not always easy to write or read
  - e.g., $G\ ((q \wedge \neg r \wedge F\ r) \Rightarrow (p \Rightarrow (\neg r\ \ U\ (s \wedge \neg r))\ U\ r)$
  - Meaning:
    - *p* triggers *s* between *q* (e.g., end of system initialization) and *r* (start of system shutdown)
- Many properties are specifiable in both CTL and LTL
  - e.g., Action *q* must respond to action *p*:
    - CTL: $AG\ (p \Rightarrow AF\ q)$
    - LTL: $G\ (p \Rightarrow F\ q)$
  - e.g., Action *s* precedes *p* after *q*
    - CTL: $A[\neg q\ U\ (q \wedge A[\neg p\ U\ s])]$
    - LTL: $[\neg q\ U\ (q \wedge [\neg p\ U\ s])]$

Pattern Hierarchy

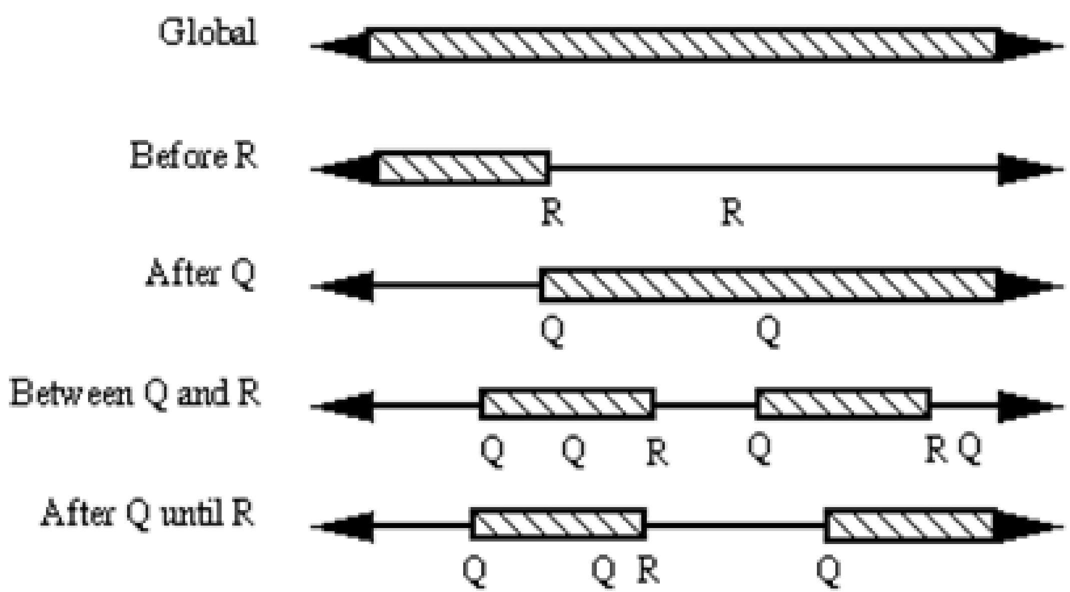
http://patterns.projects.cis.ksu.edu/
Specifying and reusing property specifications



- Absence: A condition does not occur within a scope
- Existence: A condition must occur within a scope
- Universality: A condition occurs throughout a scope
- Response: A condition must always be followed by another within a scope
- Precedence: A condition must always be preceded by another within a scope

Pattern Hierarchy: Scopes

Scopes of interest over which the condition is evaluated



Using the System: Example

- Property
  - There should be a dequeue() between an enqueue() and an empty()
  - Propositions: deq, enq, em
- Pattern: “existence” (of deq)
  - Scope: “between” (events: enq, em)
  - Look up (*S* exists between *Q* and *R*)
    - CTL: $AG\ (Q \land \neg R \Rightarrow A[\neg R\ W\ (S \land \neg R)])$
    - LTL: $G\ (Q \land \neg R \Rightarrow (\neg R\ W\ (S \land \neg R)))$
- Result
    - CTL: AG (enq ∧ ¬ em ⇒ A[¬ em W (deq ∧ ¬ em)])
    - LTL: G (enq ∧ ¬ em ⇒ (¬ em W (deq ∧ ¬ em)))

CTL Model-Checking

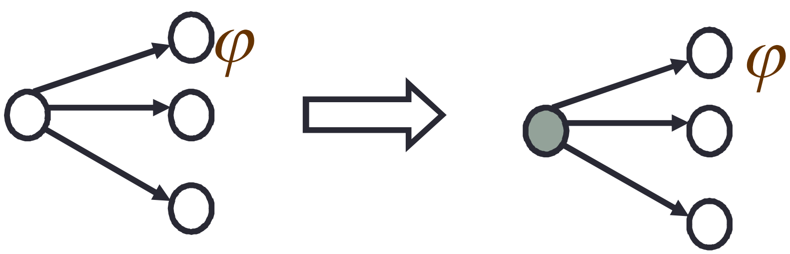
- Inputs:
  - Kripke structure *K*
  - CTL formula φ
- Assumptions:
  - The Kripke structure is finite
  - Finite length of a CTL formula
- Algorithm:
  - Working outwards towards φ
  - Label states of *K* with sub-formulas of φ that are satisfied these states
  - Output states labeled with φ

Example: EX EG ($p \Rightarrow$ E[p U q])

CTL Model-Checking (EX, EU)

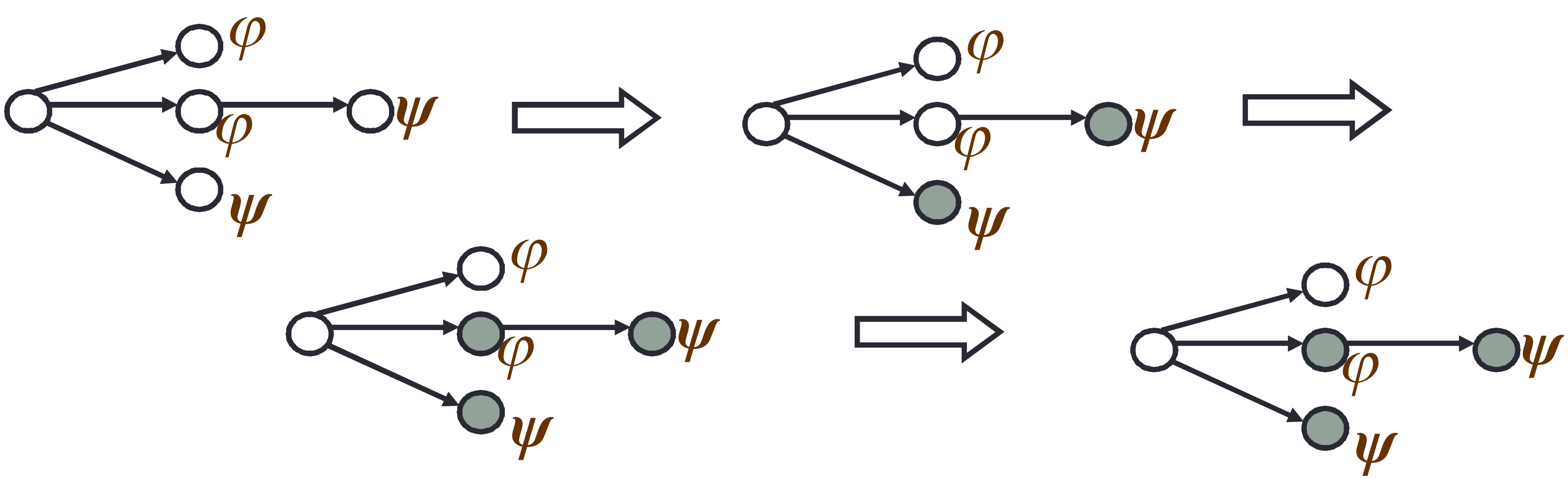
EX φ

- Label a state EX φ if any of its successors is labeled with φ



E [φ U ψ]

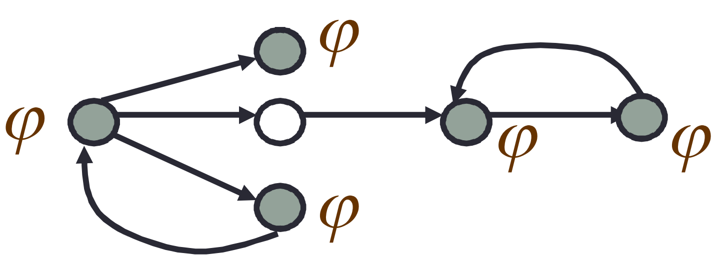
- Label a state E[φ U ψ] if it is labeled with ψ
- Until there is no change
  - label a state with E[φ U ψ] if it is labeled with φ and has a successor labeled with E[φ U ψ]

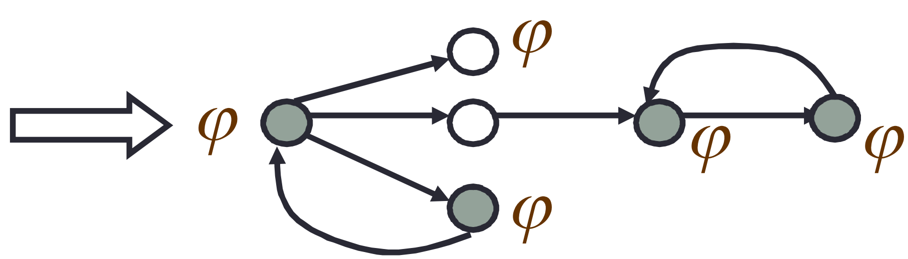


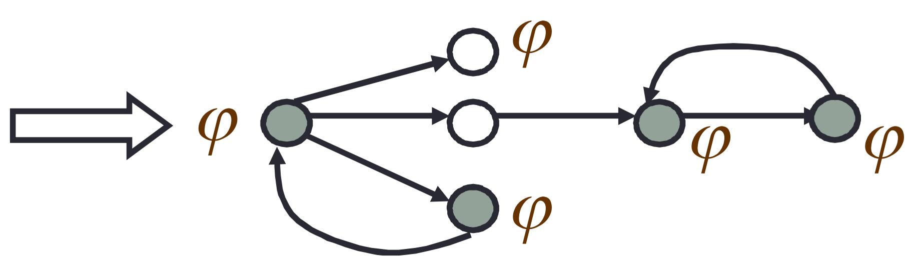
CTL Model-Checking (EG)

EG φ

- Label every node labeled with φ by EG φ
- Until there is no change
  - remove label EG φ from any state that does not have successors labeled by EG φ



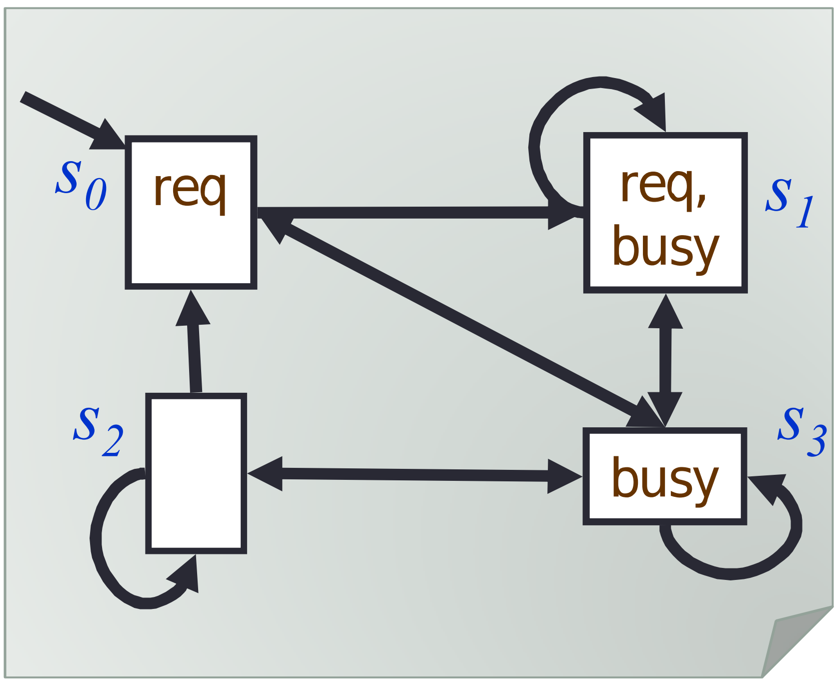




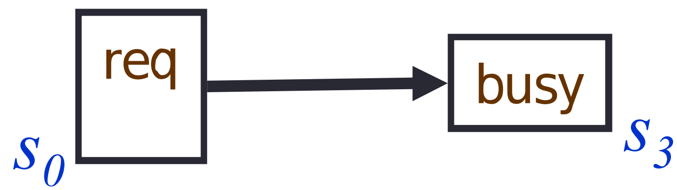
Counterexamples

Explain why the property fails to hold

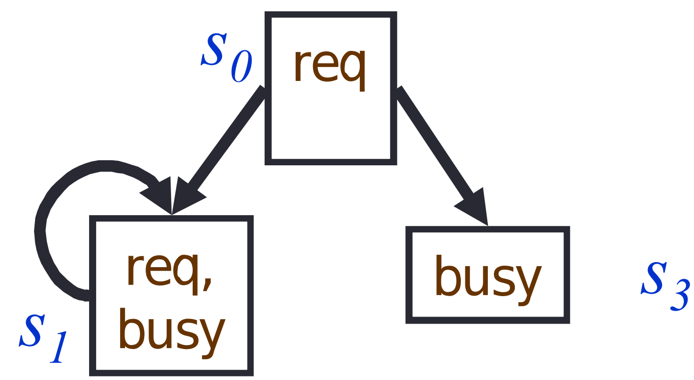
- to disprove that ϕ holds on all elements of S, produce a single element $s \in S$ s.t. $\neg\phi$ holds on s.
  - counterexamples are restricted to universally-quantified formulas
  - counterexamples are paths (trees) from initial state illustrating the failure of property



AG req



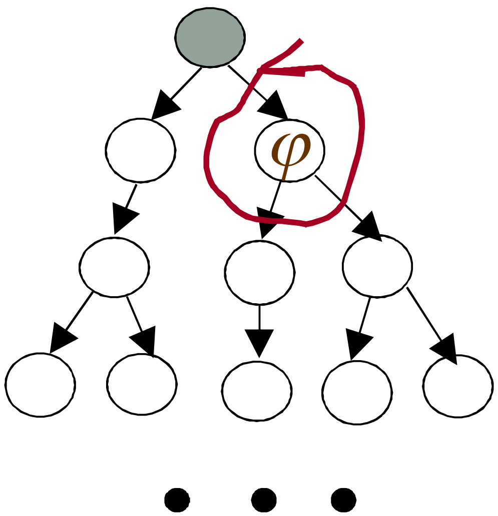
AF ¬req ∨ AX req



Generating Counterexamples (EX, EG)

Negate the prop. and express using EX, EU, EG

- e.g., AG ($\varphi \Rightarrow$ AF ψ) becomes EF($\varphi \wedge$ EG $\neg\ \psi$)



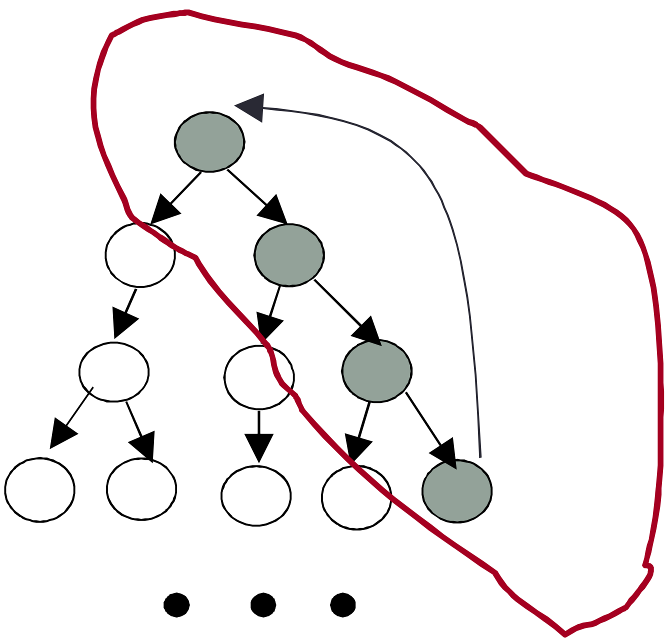
EX φ :

find a successor state labeled with φ

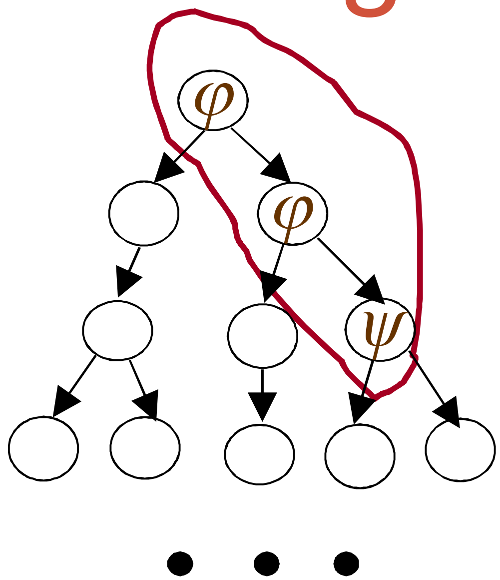
EG φ:

follow successors labeled

with EG φ until a loop is found



Generating Counterexamples (EU)



E[φ U ψ]:
remove all states that are not labeled with either φ or ψ. Then, find a path to ψ

This procedure works only for universal properties

- AX φ
- AG ($\varphi \Rightarrow$ AF ψ)
- etc.

State Explosion

- How fast do Kripke structures grow?

  Composing linear number of structures yields exponential growth!
- How to deal with this problem?
  - Symbolic model checking with efficient data structures (BDDs, SAT).
    - Do not need to represent and manipulate the model explicitly
  - Abstraction
    - Abstract away variables in the model which are not relevant to the formula being checked
  - Partial order reduction (for asynchronous systems)
    - Several interleavings of component traces may be equivalent as far as satisfaction of the formula to be checked is concerned
  - Composition
    - Break the verification problem down into several simpler verification problems

Model-Checking Techniques (Symbolic)

- BDD
  - Express transition relation by a formula, represented as BDD. Manipulate these to compute logical operations and fixpoints
  - Based on very fast decision diagram packages (e.g., CUDD)
- SAT
  - Expand transition relation a fixed number of steps (e.g., loop unrolling), resulting in a formula
  - For this unrolling, check whether the property holds
  - Continue increasing the unrolling until error is found, resources are exhausted, or diameter of the problem is reached
  - Based on very fast SAT solvers

Model-Checking Techniques (Explicit State)

- Model checking as partial graph exploration
- In practice:
  - Compute part of the reachable state-space, with clever techniques for state storage (e.g., Bit-state hashing) and path pruning (partial-order reduction)
  - Check *reachability* (X, U) properties "on-the-fly", as state-space is being computed
  - Check *non-termination* (G) properties by finding an accepting cycle in the graph

Pros and Cons of Model-Checking

- Often cannot express full requirements
  - Instead check several smaller properties
- Few systems can be checked directly
  - Must generally abstract
- Works better for certain types of problems
  - Very useful for control-centered concurrent systems
    - Avionics software
    - Hardware
    - Communication protocols
  - Not very good at data-centered systems
    - User interfaces, databases

Pros and Cons (Cont'd)

- Largely automatic and fast
- Better suited for debugging
  - … rather than assurance
- Testing vs model-checking
  - Usually, find more problems by

    exploring all behaviours of a downscaled system

    than by

    testing some behaviours of the full system

Some State of the Art Model-Checkers

- SMV, NuSMV, Cadence SMV
  - CTL and LTL model-checkers
  - Based on symbolic decision diagrams or SAT solvers
  - Mostly for hardware and other models
- Spin
  - LTL model-checker
  - Explicit state exploration
  - Mostly for communication protocols
- CBMC, SatAbs, CPAChecker, UFO
  - Combine Model Checking and Abstraction
  - Work directly on the source code (mostly C)
  - Control-dependent properties of programs (buffer overflow, API usage, etc.)