Temporal Logics

- CTL: definition, relationship between operators, adequate sets, specifying properties, safety/liveness/fairness
- Modeling: sequential, concurrent systems; maximum parallelism/interleaving
- LTL: definition, relationship between operators, CTL vs. LTL
- CTL*
- Property patterns

Computation Tree Logic

Computational tree logic - propositional branching time logic, permitting explicit quantification over all possible futures.

Fixed set of atomic formulas (p, q, r), standing for atomic descriptions of a system:
The printer is busy
There are currently no requested jobs for the printer
Conveyer belt is stopped

Choice of atomic propositions depends on our intension (but usually does not involve time)

Computation Tree Logic (Cont'd)

- Syntax:
  1. $\perp$, $\top$, and every atomic proposition is a CTL formula
  2. If f and g are CTL formulae, then so are $\neg f$, $f \wedge g$, $f \vee g$, AXf, EXf, A[$f\mathrm{U}g$], E[$f\mathrm{U}g$], AFf, EFf, AGf, EGf, A[$f\mathrm{R}g$], E[$f\mathrm{R}g$], A[$f\mathrm{W}g$], E[$f\mathrm{W}g$]
- Temporal operators - quantifier (A or E) followed by F (future), G (global), U (until), R (release), W (weak until) or X (next).

CTL Syntax - Cont'd

Which of these are well-formed and which are not:

EG r

FG r

AG AF r

E [A [p_1 U p_2] U p_3]

AF [(r U q) $\wedge$ (p U r)]

EF E[r U q] AF[(r U q) $\wedge$ (p U r)]

Formulas, Subformulas and Parse Trees

Draw parse tree for E[A[p U q] U r]

Draw parse tree for AG($p \rightarrow$ A[p U ($\neg p \wedge$ A[$\neg p$ U q])])

Definition: A subformula of a CTL formula ϕ is any formula ψ whose parse tree is a subtree of ϕ's parse tree.

Semantics of CTL

$M, s \models f$ – means that formula f is true in state s. M is often omitted since we always talk about the same model.

E.g. $s \models p \wedge \neg q$

$\pi = \pi^0 \ \pi^1 \ \pi^2 \ \ldots$ is a path
π^0 is the current state (root)
π^{i+1} is π^i's successor state. Then,

AX $f = \forall\pi \cdot \pi^1 \models f$
EX $f = \exists\pi \cdot \pi^1 \models f$
AG $f = \forall\pi \cdot \forall i \cdot \pi^i \models f$
EG $f = \exists\pi \cdot \forall i \cdot \pi^i \models f$
AF $f = \forall\pi \cdot \exists i \cdot \pi^i \models f$
EF $f = \exists\pi \cdot \exists i \cdot \pi^i \models f$

Semantics (Cont'd)

A[f U g] = $\forall\pi \cdot \exists i \cdot \pi^i \models g \wedge \forall j \cdot 0 \leq j < i \rightarrow \pi^j \models f$
E[f U g] = $\exists\pi \cdot \exists i \cdot \pi^i \models g \wedge \forall i \cdot 0 \leq j < i \rightarrow \pi^j \models f$
A[f R g] = $\forall\pi \cdot \forall j \geq 0 \cdot (\forall i < j \cdot \pi^i \not\models f) \rightarrow \pi^j \models g$
E[f R g] = $\exists\pi \cdot \forall j \geq 0 \cdot (\forall i < j \cdot \pi^i \not\models f) \rightarrow \pi^j \models g$

Note: the i in $\exists i \cdot$ could be 0.

Examples

A process is enabled infinitely often on every computation path.

A process will eventually be deadlocked.

It is always possible to get to a *restart* state.

An elevator does not change direction when it has passengers wishing to go in the same direction.

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

Which situation does this signify: AG($p \longrightarrow$ AF($s \wedge$ AX(AF(t)))

Relationship between CTL operators

$$
\begin{array}{rcll}
\neg \text{AX } f & = & \text{EX } \neg f & \\
\neg \text{AF } f & = & \text{EG } \neg f & \\
\neg \text{EF } f & = & \text{AG } \neg f & \\
\text{AF } f & = & \text{A}[\top \text{ U } f] & \\
\text{EF } f & = & \text{E}[\top \text{ U } f] & \\
\text{A}[\bot \text{ U } f] & = & \text{E}[\bot \text{ U } f] = f & \\
\text{A}[f \text{ U } g] & = & \neg \text{E}[\neg g \text{ U } (\neg f \wedge \neg g)] \wedge \neg \text{EG } \neg g & \\
\text{A}[f \text{ W } g] & = & \neg \text{E}[\neg g \text{ U } (\neg f \wedge \neg g)] & \text{(weak until)} \\
\text{E}[f \text{ U } g] & = & \neg \text{A}[\neg g \text{ W } (\neg f \wedge \neg g)] & \\
\text{AG } f & = & f \wedge \text{AX AG } f & \\
\text{EG } f & = & f \wedge \text{EX EG } f & \\
\text{AF } f & = & f \vee \text{AX AF } f & \\
\text{EF } f & = & f \vee \text{EX EF } f & \\
\text{A}[f \text{ U } g] & = & g \vee (f \wedge \text{AX A}[f \text{ U } g]) & \\
\text{E}[f \text{ U } g] & = & g \vee (f \wedge \text{EX E}[f \text{ U } g]) & \\
\neg \text{E}[f \text{ U } g] & = & \text{A}[\neg f \text{ R } \neg g] & \\
\neg \text{A}[f \text{ U } g] & = & \text{E}[\neg f \text{ R } \neg g] &
\end{array}
$$

Adequate Sets

Definition: A set of connectives is *adequate* if all connectives can be expressed using it.

Example: $\{\neg, \wedge\}$ is adequate for propositional logic: $a \vee b = \neg(\neg a \wedge \neg b)$.

Theorem: The set of operators $\bot$, $\neg$ and $\wedge$ together with EX, EG, and EU is adequate for CTL, e.g., AF $(a \vee$ AX $b)$ = $\neg$ EG $\neg(a \vee$ AX $b)$ = $\neg$ EG($\neg a \wedge$ EX $\neg b$).

EU - reachability. EG - non-termination (presence of infinite behaviours)

Other adequate sets: $\{$AU, EU, EX$\}$, $\{$AF, EU, EX$\}$

Theorem: Every adequate set has to include EU.

Sublanguages of CTL

ACTL - CTL with only universal path quantifiers (AX, AF, AG, AU, AR)

ECTL - CTL with only existential path quantifiers (EX, EF, EG, EU, ER)

Positive normal form (pnf) – negations applied only to atomic propositions. Then, need $\wedge$, $\vee$, and both U and R operators. Also called *negation normal form* (nnf).

Exercise: convert $\neg$ (AG A[p U $\neg q$]) to pnf:

Property Types: Safety

- Safety: nothing bad ever happens
  - Invariants: ”x is always less than 10”
  - Deadlock freedom: ”the system never reaches a state where no moves are possible”
  - Mutual exclusion: ”the system never reaches a state where two processes are in the critical section”

  As soon as you see the ”bad thing”, you know the property is false (so they are falsified by a finite prefix of an execution trace)

Liveness Properties

- Liveness: something good will eventually happen
  - Termination: ”the system eventually terminates”
  - Response: ”if action X occurs then eventually action Y will occur”
- Need to keep looking for the ”good thing” forever
- Liveness can only be falsified by an infinite-suffix of an execution trace
  - Practically, a counter-example here is a set of states starting from initial one and going through a loop where you get stuck and never reach the ”good thing”.

Using CTL

Mutual Exclusion Problem. Aimed to ensure that two processes do not have access to some shared resource (database, file on disk, etc.) at the same time. Identify *critical* sections and ensure that at most one process is in that section.

Interested in the following properties:
(Type?): The protocol allows only one process to be in its critical section at any time.

Formalization:

Why is this not enough?

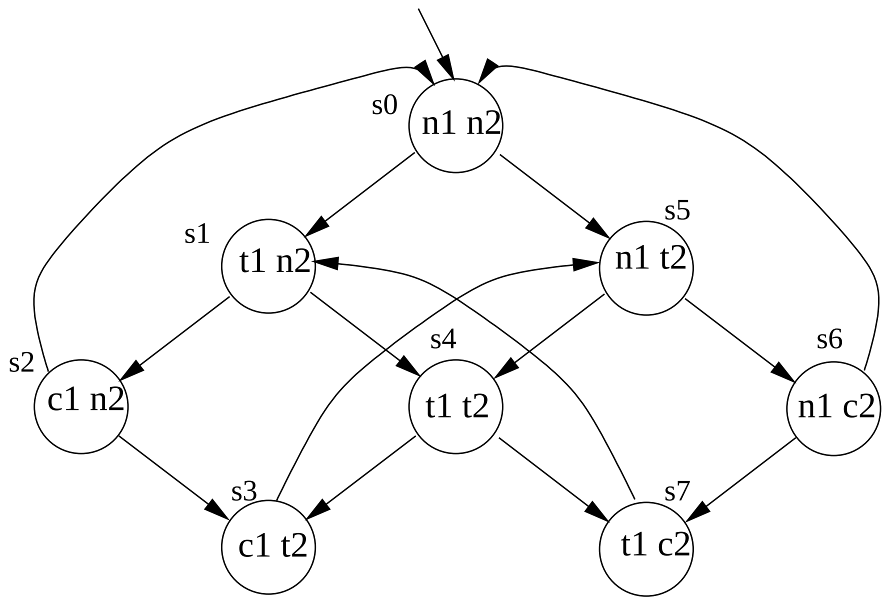
(Type?): Whenever any process wants to enter its critical section, it will eventually be permitted to do so.

Formalization:

(Type?): A process can always request to enter its critical section.

Formalization:

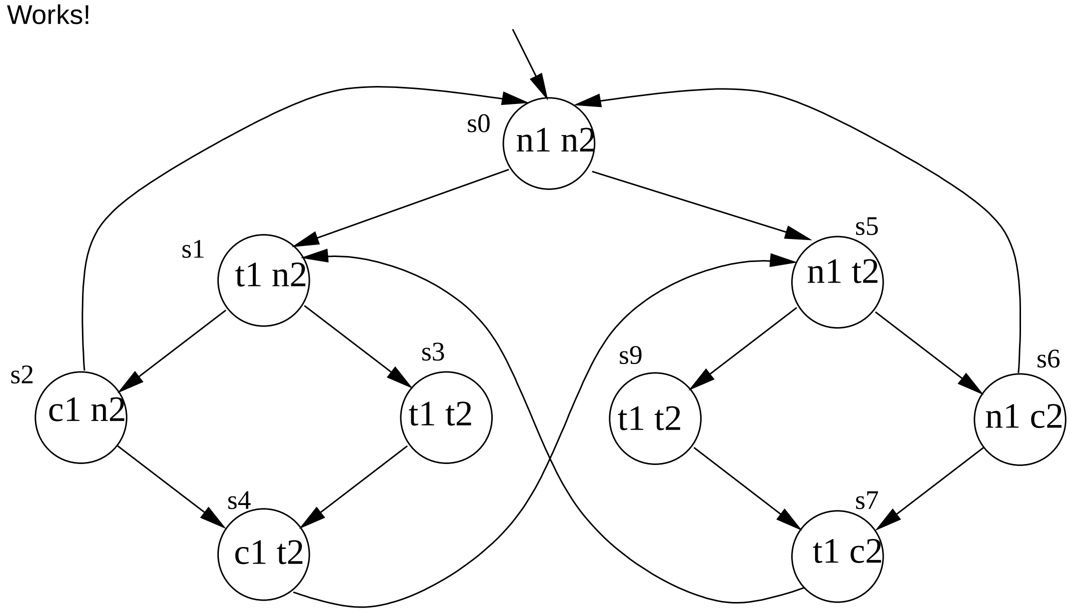
The first modeling attempt



Does it work?

Second modeling attempt

Works!



The problem is a bit simplified (cannot stay in *critical* forever!)

What happens if we do want to model this?

Notion of Fairness

Fairness: a path π is *fair* w.r.t. property ψ if ψ is true on π infinitely often.

We may want to evaluate A and E constraints only over those paths.

Examples: each process will run infinitely often; a process can stay in a critical section arbitrarily long, as long as it eventually leaves.

Two types of fairness: simple

Property ϕ is true infinitely often.

and compound

If ϕ is true infinitely often, then ψ is also true infinitely often.

In this course we will deal only with simple fairness.

Formal Definition of Fairness

Let $C = \{\psi_1, \psi_2, \ldots, \psi_n\}$ be a set of n fairness constraints. A computation path $s_0, s_1, \ldots$ is *fair* w.r.t. C if for each i there are infinitely many j s.t. $s_j \models \psi_i$, that is, each ψ_i is true infinitely often along the path.

We use A_C and E_C for the operators A and E restricted to fair paths.

$\mathrm{E}_C\mathrm{U}$, $\mathrm{E}_C\mathrm{G}$ and $\mathrm{E}_C\mathrm{X}$ form an adequate set.
$\mathrm{E}_C\mathrm{G}\top$ holds in a state if it is the beginning of a fair path.

Also, a path is fair iff any suffix of it is fair. Finally,

$$\mathrm{E}_C[\phi \mathrm{U} \psi] = \mathrm{E}[\phi \mathrm{U}(\psi \wedge \mathrm{E}_C\mathrm{G}\top)]$$

$$\mathrm{E}_C\mathrm{X}\phi = \mathrm{EX}(\phi \wedge \mathrm{E}_C\mathrm{G}\top)$$

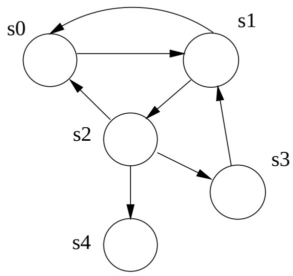
Can fairness be expressed in CTL?

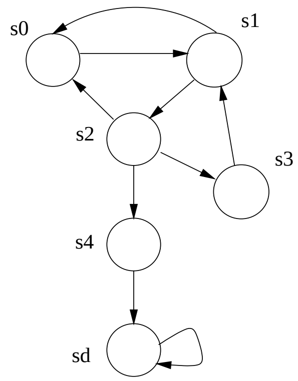
Kripke Structures (Our Model)

Formula is defined with respect to a model $M = \langle AP, S, s_0, R, I \rangle$, where

- AP is a set of atomic propositions
- S is a set of states
- $s_0 \in S$ is the start state
- R is a transition relation (every state has a successor)
- I is a set of interpretations specifying which propositions are true in each state.

Example:





How to deal with deadlock states?

Where Do Models Come from?

Example:

```
x, y:  {0, 1}
x := (x + y) mod 2
initial state:  x = 1, y = 1
```

Description:

$S_0(x,y) \equiv x = 1 \wedge y = 1$

$R(x,y,x',y') \equiv x' = (x+y)\, mod\, 2 \ \wedge \ y' = y$

Pictorially:

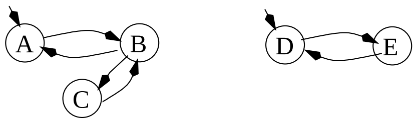
Which states are reachable?

Models of Concurrency

Maximum parallelism (synchronous) – ”simultaneous execution of atomic actions in all system modules capable of performing an operation.”

Interleaving (asynchronous) – ”concurrent execution of modules is represented by interleaving of their atomic actions”.

Example:



MP:		INT:		COMB:	
	AD		A		AD
	BE		D		AE
	CD		B		BE
	BE		C		...
	AD		E		
	...		B		
			A		
			B		
			D		
			...		

Modeling systems (Cont’d)

- Communication: shared vars, message passing, handshaking
- Sequential statements: atomic (assignment, skip, wait, lock, unlock), composition (P_1; P_2), condition (if b then P_1 else P_2), loop (while b do P)
- Concurrency: cobegin $P_1 \mid\mid P_2 \mid\mid ... \mid\mid P_n$ coend

Model state of each process using variable pc.

Example

```
P0::l0: while True do
           NC0:  wait (turn=0);
           CR0:  turn := 1;
        end while
     l0'

P1:: l1: while True do
           NC1:  wait (turn=1);
           CR1:  turn := 0;
        end while
     l1'
```

Variables:

$pc_i : \{l_i, l_i', NC_i, CR_i, \perp\}$

turn: shared, initial value = ???

Initial:

$S_0(V, PC) \equiv pc_0 = \perp \wedge pc_1 = \perp$

Transition Relation

$$
\begin{array}{rcl}
R(V, V', PC, PC') & = & pc_i = l_i \wedge pc_i' = NC_i \wedge \textsf{True} \wedge same(\text{turn}) \\
& \vee & pc_i = NC_i \wedge pc_i' = CR_i \wedge \text{turn} = i \\
& & \wedge\, same(\text{turn}) \\
& \vee & pc_i = CR_i \wedge pc_i' = l_i \wedge \text{turn}' = (i+1)\, mod\, 2 \\
& \vee & pc_i = NC_i \wedge pc_i' = NC_i \wedge \text{turn} \neq i \\
& & \wedge\, same(\text{turn}) \\
& \vee & pc_i = l_i \wedge pc_i' = l_i' \wedge \textsf{False} \wedge same(\text{turn})
\end{array}
$$

Picture

LTL

- If p is an atomic propositional formula, it is a formula in LTL.
- If p and q are LTL formulas, so are $p \wedge q$, $p \vee q$, $\neg p$, p U q, p W q, p R q, $\circ p$ (next), $\diamond p$(eventually), $\Box p$ (always)

Interpretation: over *computations* $\pi : \omega \Rightarrow 2^{AP}$ which assigns truth values to the elements of AP at each time instant:

- $\pi \models \circ f$ iff $\pi^1 \models f$
- $\pi \models f \, \mathrm{U} \, g$ iff $\exists i \cdot \pi^i \models g \wedge \forall j \cdot 0 \leq j < i \rightarrow \pi^j \models f$
- $\pi \models \Box f$ iff $\forall i \cdot \pi^i \models f$
- $\pi \models \diamond f$ iff $\exists i \cdot \pi^i \models f$

Here, π^0 – initial state of the system

Two other operators:

- p W q = $\Box p \vee (p \, \mathrm{U} \, q)$ (p unless q, p waiting for q, p weak-until q)
- p R q = $\neg(\neg p \, \mathrm{U} \, \neg q)$ (release)

Some Temporal Properties

$$
\begin{aligned}
\neg \circ p &= \circ \neg p \\
\diamond p &= \textit{True}\ \mathrm{U}\ p \\
\Box p &= \neg \diamond \neg p \\
p\ \mathrm{W}\ q &= \Box p \vee (p\ \mathrm{U}\ q) \\
p\ \mathrm{R}\ q &= \neg(\neg p\ \mathrm{U}\ \neg q) \\
\Box p &= p \wedge \circ \Box p \\
\diamond p &= p \vee \circ \diamond p \\
p\ \mathrm{U}\ q &= q \vee (p \wedge \circ (p\ \mathrm{U}\ q))
\end{aligned}
$$

A property φ holds in a model if it holds on every path emanating from the initial state.

Expressing Properties in LTL

Good for safety ($\Box\neg$) and liveness ($\diamond$) properties.

- $p \longrightarrow \diamond q$ – If p holds in initial state s_0, then q holds at s_j for some $j \geq 0$.
- $\Box \diamond q$ – Each path contains infinitely many q's.
- $\diamond \Box q$ – At most a finite number of states in each path satisfy $\neg q$. Property q eventually stabilizes.
- $\Box(p\ \mathrm{U}\ q)$ – always p remains true at least until q becomes true.
- $\neg(\diamond(p\ \mathrm{U}\ q))$ – never is there a point in the execution such that p remains true at least until q becomes true.

Express: it is not true that p is true at least until the point s.t. for all paths q is true at least until r is true

Fairness properties

Fairness (strong): "if something is attempted or requested infinitely often, then it will be successful/allocated infinitely often"

Example:
$\forall p \in \text{processes} \cdot \Box \Diamond \text{ready}(p) \rightarrow \Box \Diamond \text{run}(p)$

Different forms of fairness:

- $(\Box \Diamond \text{attempt}) \rightarrow (\Box \Diamond \text{succeed})$
- $(\Box \Diamond \text{attempt}) \rightarrow (\Diamond \text{succeed})$
- $(\Box \text{attempt}) \rightarrow (\Box \Diamond \text{succeed})$
- $(\Box \text{attempt}) \rightarrow (\Diamond \text{succeed})$

Comparison of LTL and CTL

Syntactically: LTL simpler than CTL

Semantically: incomparable!

- CTL formula EF p is not expressible in LTL
- LTL formula $\Diamond\Box p$ not expressible in CTL.

Question: What about AF AG p?
Model: self-loop on p, transition on $\neg p$ to a state with a self-loop on p.

AFAG p is false, FG p is true.

Most useful formulas expressible in both:

- Invariance: $\Box p$ and AG p
- Liveness (response): $\Box(p \rightarrow \Diamond q)$ and AG($p \rightarrow$ AFq).

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

CTL*: Unifying CTL and LTL

- Path quantifiers: A and E
- Temporal quantifiers: X ($\circ$), F ($\diamond$), G ($\square$), U, R
- State formulas:
  1. $p \in AP$
  2. If f and g are state formulas, then $\neg f$, $f \vee g$, $f \wedge g$ are state formulas.
  3. If f is a path formula then E f and A f are state formulas
- Path formulas:
  1. every state formula
  2. If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, X f, G f, F f, f U g, f R g are path formulas.

CTL* – the set of all state formulas.

- Examples: A(FG p), AG p, A(FG p) $\vee$ AG(EF p)
- Formal semantics: see the book, p. 29.
- How to think of LTL as a subset of CTL*?

Property Patterns: Motivations

1. Temporal properties are not always easy to write or read.
   - Ex: $\square\big((Q \wedge \neg R \wedge \diamond R) \rightarrow \big(P \rightarrow (\neg R\, U\, (S \wedge \neg R))\, U\, R\big)$
   - Meaning: P triggers S between Q (e.g., end of system initialization) and R (start of system shutdown)
2. Most useful properties: specifiable both in CTL and LTL.
   - Ex: Action Q must respond to action P:
     - CTL: AG($P \rightarrow$ AF Q)
     - LTL: $\square(P \rightarrow \diamond Q)$
   - Ex: Action S precedes P after Q
     - CTL: A[$\neg Q$ U ($Q \wedge$ A[$\neg P$ U S]))
     - LTL: $\square\neg Q \vee \diamond(Q \wedge (\neg P \,\mathrm{U}\, S))$

Manna & Pnueli Classification

Canonical forms:

- Safety: $\Box p$
- Guarantee: $\Diamond p$
- Obligation: $\Box q \vee \Diamond p$
- Response: $\Box \Diamond p$
- Persistence: $\Diamond \Box p$
- Reactivity: $\Box \Diamond p \vee \Diamond \Box q$

Source: Zohar Manna and Amir Pnueli. The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, 1992

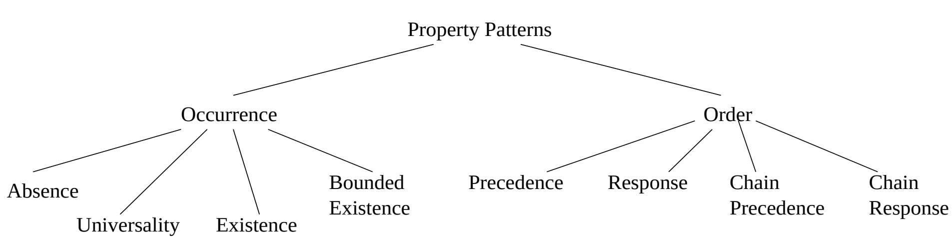
A preferred classification: based on the *semantics* rather than *syntax* of properties so that non-experts can use it!

Pattern Hierarchy

http://www.cis.ksu.edu/santos/spec-patterns

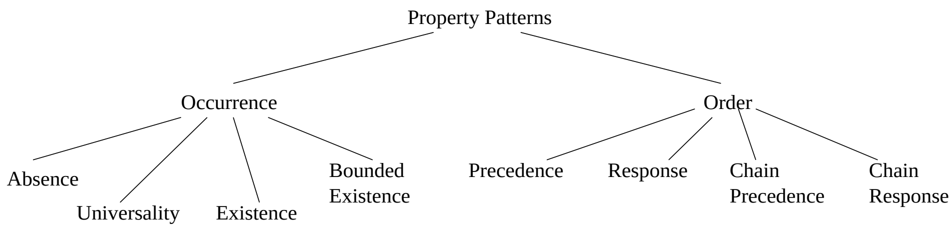
Developed by Dwyer, Avrunin, Corbett

Goal: specifying and reusing property specifications for model-checking



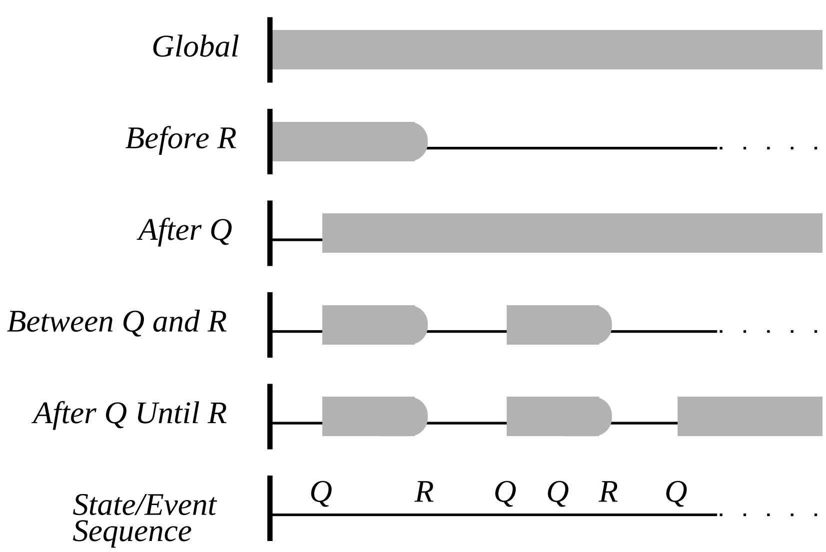
- Occurrence Patterns - require states/events to occur or not
  - Absence: A given state/event *does not occur* within a given scope
  - Existence: A given state/event *must occur* within a given scope
  - Bounded existence: A given state/event *must occur* k *times* (at least k times, at most k times) within a given scope
  - Universality: A given state/event *must occur throughout* a given scope

Pattern Hierarchy (Cont'd)



- Order Patterns - constrain the order of states/events
  - Precedence: A state/event P *must always be preceded* by a state/event Q within a scope
  - Response: A state/event P *must always be followed* by a state/event Q with a scope
  - Chain precedence: A sequence of states/events P_1, ..., P_n *must always be preceded* by a sequence of states/events $Q_1, \ldots, Q_m$ with a scope
  - Chain Response: A sequence of states/events P_1, ..., P_n *must always be followed* by a sequence of states/events Q_1, ..., Q_m within a scope

Pattern Scopes



Using the System

Example: Between an `enqueue()` and `empty()` there must be a `dequeue()`

Propositions: `enqueue()`, `empty()`, `dequeue()`

Pattern and Scope: ”existence” pattern with ”between” scope

Property: `dequeue()` exists between `enqueue()` and `empty()`

LTL: $\Box((\texttt{enqueue()} \wedge \neg\, \texttt{empty()}) \rightarrow (\neg\, \texttt{empty()}\ \mathcal{W}\ (\texttt{dequeue()} \wedge \neg\, \texttt{empty()})))$

CTL: $AG(\texttt{enqueue()} \wedge \neg\, \texttt{empty()} \rightarrow A[\neg\, \texttt{empty()}\ \mathcal{W}\ (\texttt{dequeue()} \wedge \neg \texttt{empty()})])$

Homework: more usage of patterns, some subtle points (open/closed intervals, between vs. after-until)

food for the slide eater