Lecture 3

Automata-Based Representation of Linear-Time Properties and Linear Temporal Logic (LTL)

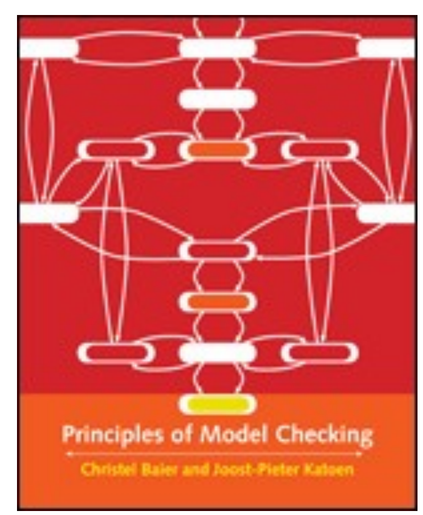
Richard M. Murray

Nok Wongpiromsarn Ufuk Topcu

EECI, 14 May 2012

Outline

- Automata-based representation of linear-time properties
- Syntax and semantics of LTL
- Specifying properties in LTL
- Equivalence of LTL formulas
- Fairness in LTL
- Other temporal logics (if time)



Principles of Model Checking, Christel Baier and Joost-Pieter Katoen. MIT Press, 2008.

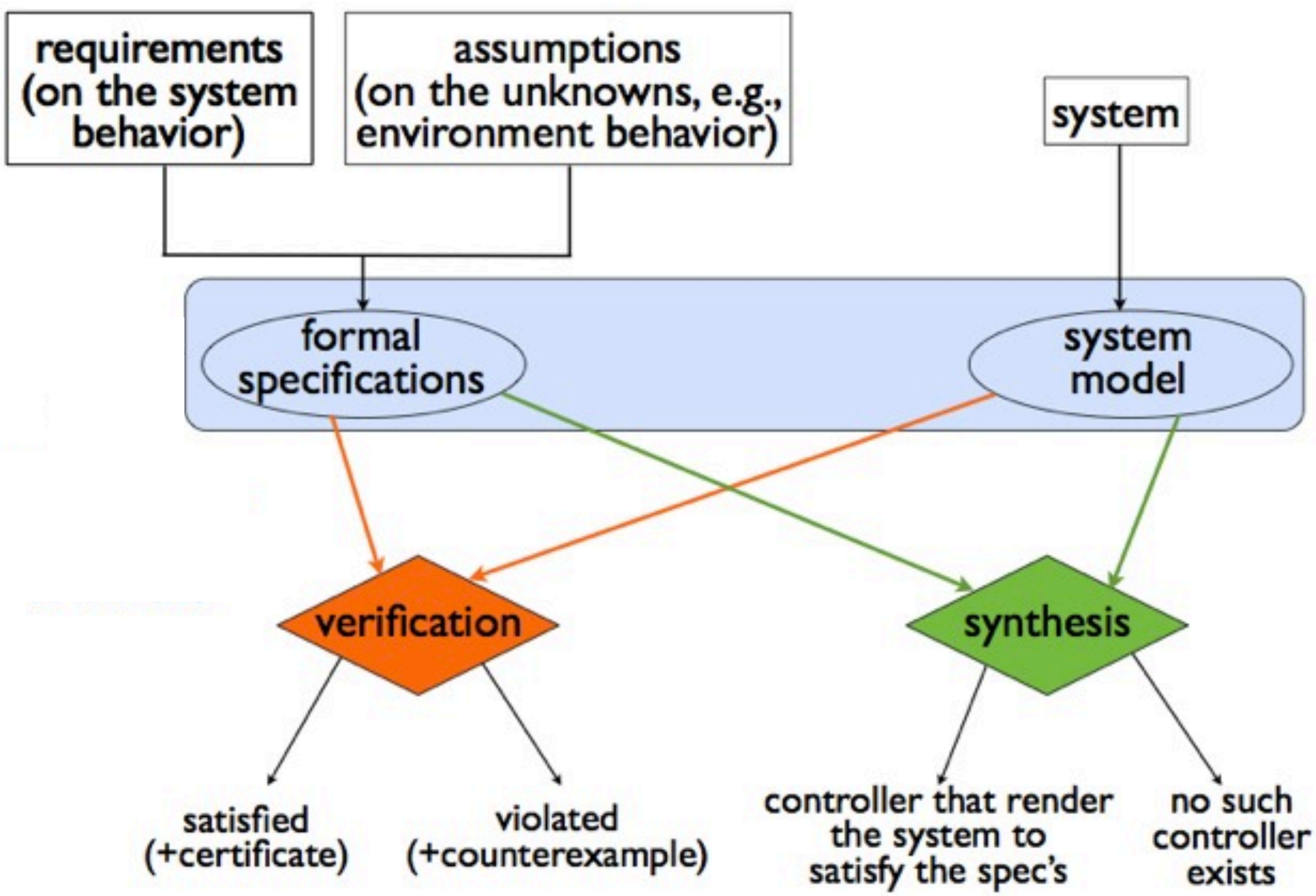
Chapter 5

Representations of linear-time properties

Two more representations of linear-time properties:

- Automata-based: readable by machine
- Linear temporal logic (LTL): readable by humans

- LTL is a formal language for describing linear-time properties
- It provides particularly useful operators for constructing linear-time properties without explicitly specifying sets (of, e.g., infinite sequences of subsets of atomic propositions)



Nondeterministic finite automaton (NFA)

A nondeterministic finite automaton $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ is a tuple with
- A is a set of states,
- Σ is an alphabet,
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function,
- $Q_0 \subseteq Q$ is a set of initial states, and
- $F \subseteq Q$ is a set of accept (or: final) states.

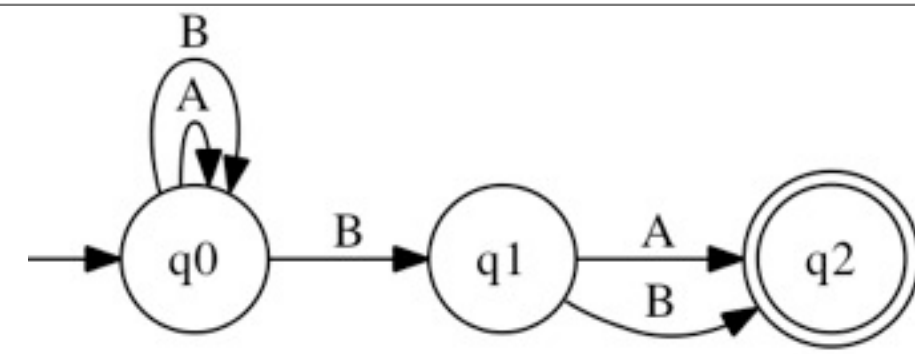
set of finite words

Let w $= A_1 \ldots A_n \in \Sigma^*$ be a finite word.
A *run* for w in $\mathcal{A}$ is a finite sequence of states $q_0 q_1 \ldots q_n$ s.t.
- $q_0 \in Q_0$
- $q_i \xrightarrow{A_{i+1}} q_{i+1}$ for all $0 \leq i < n$.

A run $q_0 q_1 \ldots q_n$ is called accepting if $q_n \in F$.

A finite word in accepted if it leads to an accepting run.

The *accepted language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of finite words in Σ^* accepted by $\mathcal{A}$.



$Q = \{q0, q1, q2\}$, $\Sigma = \{A, B\}$
$Q_0 = \{q0\}$, $F = \{q2\}$

$\delta(q0, A) = \{q0\}$, $\delta(q0, B) = \{q0, q1\}$
$\delta(q1, A) = \{q2\}$, $\delta(q1, B) = \{q2\}$
$\delta(q2, A) = \emptyset$, $\delta(q0, B) = \emptyset$

word	run
empty word	q0
B	q0q1
ABA	q0q0q0q0
BBA	q0q0q0q0
BA	q0q1q2
BBA	q0q0q1q2

accepted

NFA: $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$

Regular safety properties

A set $\mathcal{L} \subseteq \Sigma^*$ of finite strings is called a regular language if there is a nondeterministic finite automaton $\mathcal{A}$ s.t. $\mathcal{L} = \mathcal{L}(\mathcal{A})$.

language (set of finite words) accepted by the NFA

A safety property P_{safe} over AP is called *regular* if its set of bad prefixes constitutes a regular language over 2^{AP}.

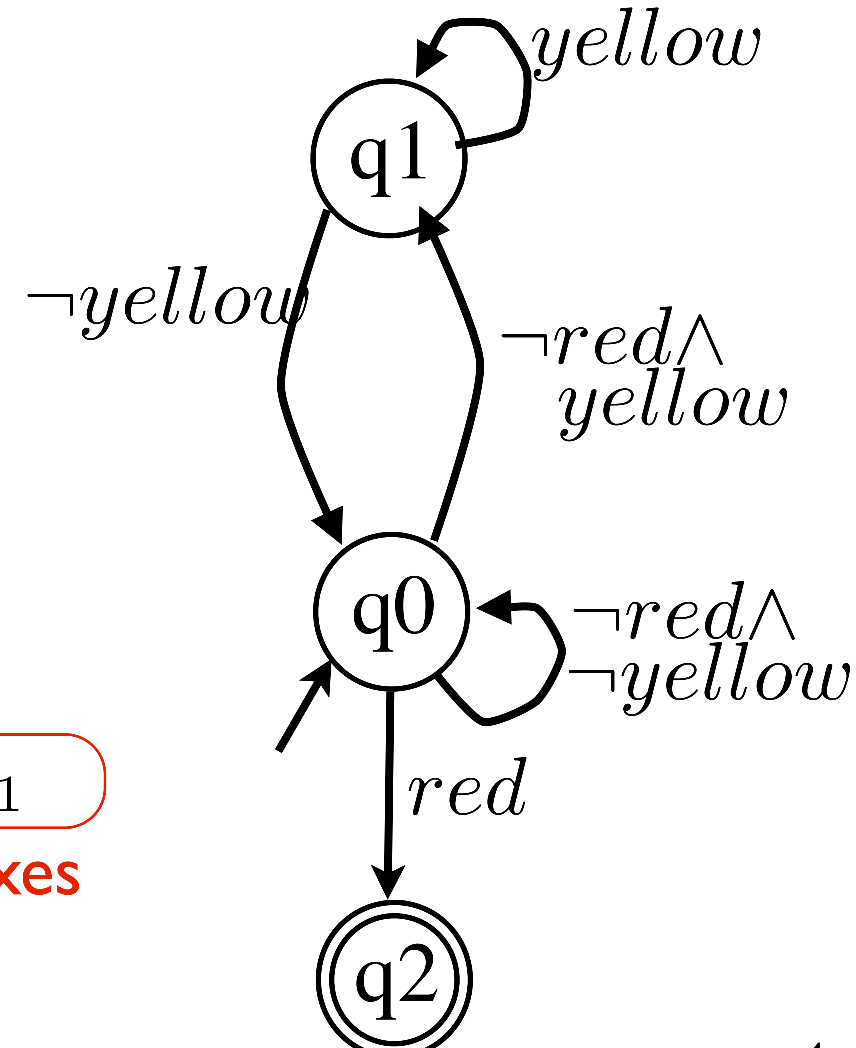
That is: $\exists$ NFA $\mathcal{A}$ s.t. $\mathcal{L}(\mathcal{A}) =$ bad prefixes of P_{safe}

<u>Example</u>: *AP* = {red, green, yellow}
"Each red must be preceded immediately by a yellow"
is a regular safety property.

Sample bad prefixes:

- {}{}{red}
- {}{red}
- {yellow}{yellow}{green}{red}
- $A_0 A_1 \ldots A_n$ s.t. $n > 0, red \in A_n,$ and $yellow \notin A_{n-1}$

general form of minimal bad prefixes

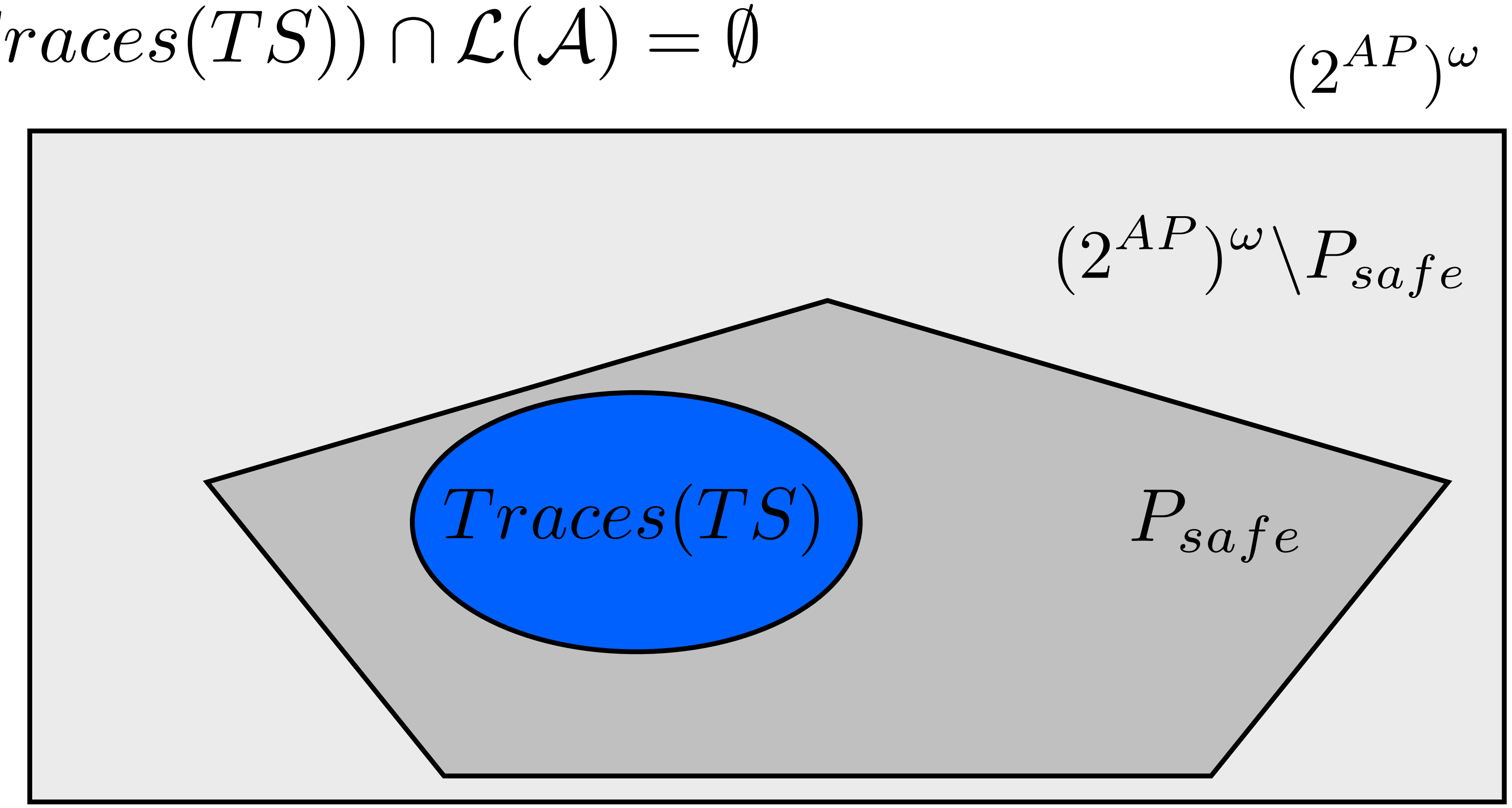


Verifying regular safety properties

Given a transition system TS and a regular safety property P_{safe}, both over the atomic propositions AP.

Let $\mathcal{A}$ be an NFA s.t. $\mathcal{L}(\mathcal{A}) = BadPref(P_{safe})$.

$$
\begin{aligned}
TS \models P_{safe} \quad & \text{iff} \quad Traces(TS) \subseteq P_{safe} \\
& \text{iff} \quad Traces(TS) \cap ((2^{AP})^\omega \backslash P_{safe}) = \emptyset \\
& \text{iff} \quad Traces(TS) \cap BadPref(P_{safe}).(2^{AP})^\omega = \emptyset \\
& \text{iff} \quad pref(Traces(TS)) \cap BadPref(P_{safe}) = \emptyset \\
& \text{iff} \quad \underbrace{pref}_{\text{finite prefixes}}(Traces(TS)) \cap \mathcal{L}(\mathcal{A}) = \emptyset
\end{aligned}
$$



For words w and σ, w.σ denotes their concatenation.

<u>Invariant</u>	<u>Safety</u>	<u>Liveness</u>
state condition	something bad never happens	something good will happen eventually
violated at individual states	any infinite run violating the property has a finite prefix	violated only by infinite runs
verification: find the reachable states and check the invariant condition	verification: based on nondeterministic finite automaton which accepts “finite runs”	verification: ?

Nondeterministic Buchi automaton (NBA)

A nondeterministic Buchi automaton is same as an NFA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ with its runs interpreted differently.

Let $w = A_1 A_2 \ldots \in \Sigma^\omega$ be an infinite string. A *run* for w in $\mathcal{A}$ is an infinite sequence $q_0 q_1 \ldots$ of states s.t.
- $q_0 \in Q_0$ and
- $q_0 \xrightarrow{A_1} q_1 \xrightarrow{A_2} q_2 \xrightarrow{A_3} \ldots$.

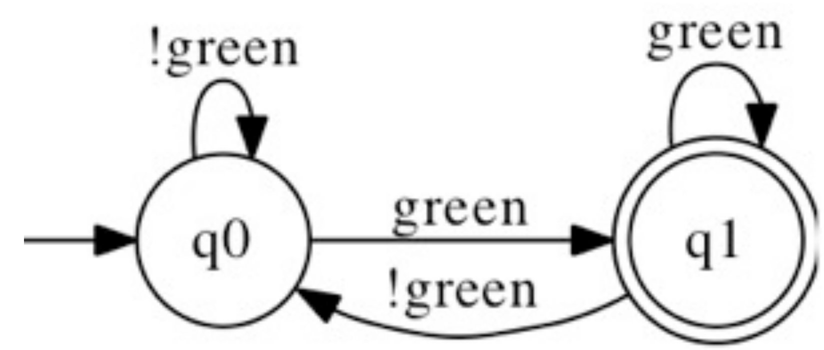
A run is *accepting* if $q_j \in F$ for infinitely many j.

A string w is accepted by $\mathcal{A}$ if there is an accepting run of w in $\mathcal{A}$.

$\mathcal{L}_\omega(\mathcal{A})$: set of infinite strings accepted by $\mathcal{A}$.

A set of infinite string $\mathcal{L}_\omega \subseteq \Sigma^\omega$ is called an ω-regular language if there is an NBA $\mathcal{A}$ s.t. $\mathcal{L}_\omega = \mathcal{L}_\omega(\mathcal{A})$.

AP = {red, green}



input word:
$\{green\}\{\}\{green\}\{\}\{green\}\{\}\ldots$

run:
$q_0 q_1 q_0 q_1 q_0 q_1 \ldots$

input word:
$(\{green, red\}\{\}\{green\}\{red\})^\omega$

run:
$q_0 q_1 q_0 q_1 q_0 q_1 \ldots$

The NBA on the right accepts the infinite words satisfying the LT property: "infinitely often green."

NBA: $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$

ω-regular properties

An LT property P over AP is called ω-regular if P is an ω-regular language over 2^{AP}.

Invariant, regular safety, and various liveness properties are ω-regular.

Let P be an ω-regular property and $\mathcal{A}$ be an NBA that represents the "bad traces" for P.

<u>Basic idea behind model checking ω-regular properties:</u>

$$
\begin{aligned}
TS \not\models P \quad & \text{if and only if} \quad Traces(TS) \not\subseteq P \\
& \text{if and only if} \quad Traces(TS) \cap \left((2^{AP})^{\omega} \setminus P\right) \neq \emptyset \\
& \text{if and only if} \quad Traces(TS) \cap \overline{P} \neq \emptyset \\
& \text{if and only if} \quad Traces(TS) \cap \mathcal{L}_{\omega}(\mathcal{A}) \neq \emptyset
\end{aligned}
$$

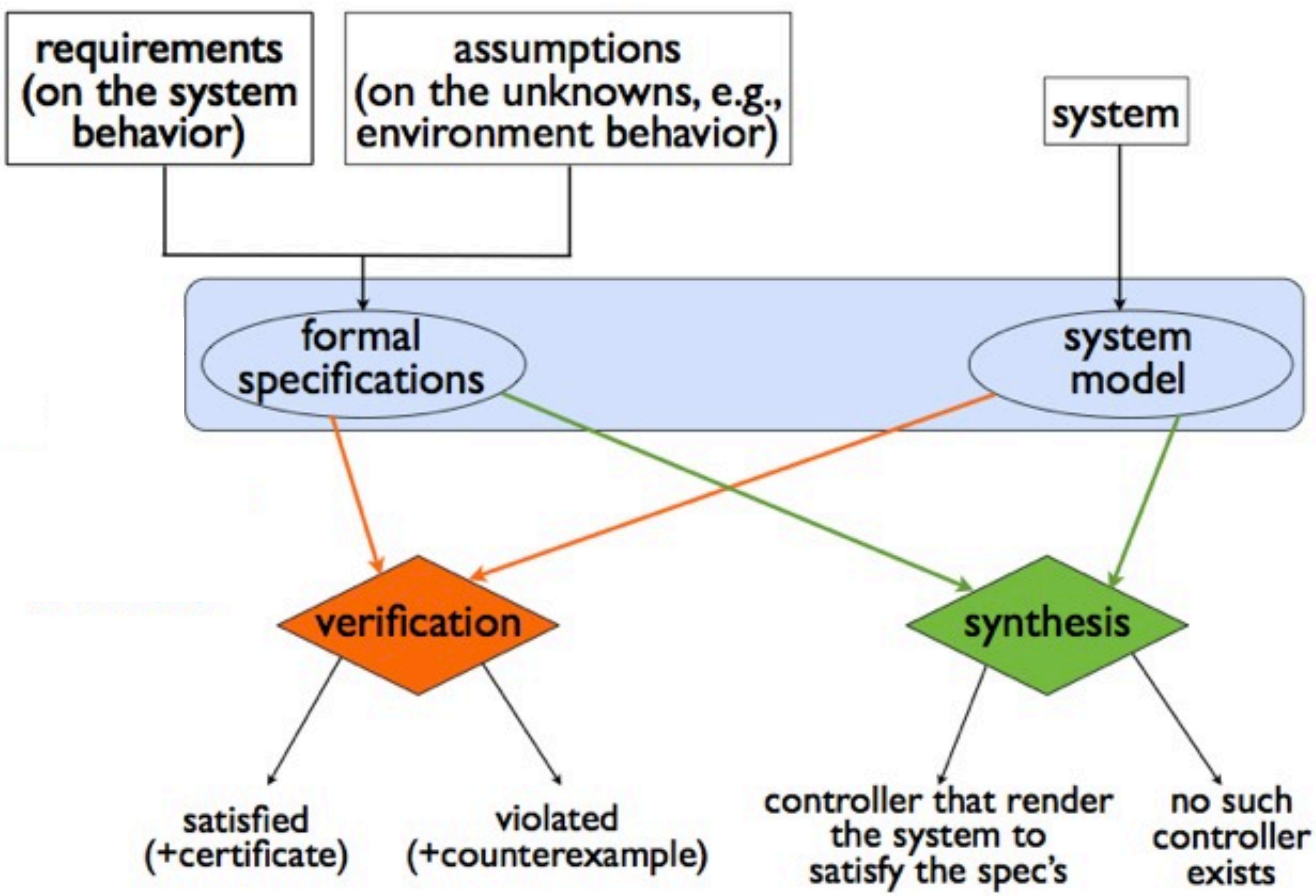
Invariant	Safety	Liveness
state condition	something bad never happens	something good will happen eventually
violated at individual states	any infinite run violating the property has a finite prefix	violated only by infinite runs
verification: find the reachable states and check the invariant condition	verification: based on nondeterministic finite automaton which accepts “finite runs”	verification: based on nondeterministic Buchi automaton which accepts infinite runs

Representations of linear-time properties

Two more representations of linear-time properties:

- Automata-based: readable by machine
- Linear temporal logic (LTL): readable by humans

- LTL is a formal language for describing linear-time properties
- It provides particularly useful operators for constructing linear-time properties without explicitly specifying sets (of, e.g., infinite sequences of subsets of atomic propositions)



Temporal logic

Two key operators in temporal logic

- ◊ “eventually” – a property is satisfied at some point in the future
- □ “always” – a property is satisfied now and forever into the future

“Temporal” refers underlying nature of time

- *Linear* temporal logic ⇒ each moment in time has a well-defined successor moment
- *Branching* temporal logic ⇒ reason about multiple possible time courses
- “Temporal” here refers to “ordered events”; no explicit notion of time

LTL = linear temporal logic

- Specific class of operators for specifying linear time properties
- Introduced by Pneuli in the 1970s (recently passed away)
- Large collection of tools for specification, design, analysis

Other temporal logics

- CTL = computation tree logic (branching time; will see later, if time)
- TCTL = timed CTL - check to make sure certain events occur in a certain time
- TLA = temporal logic of actions (Lamport) [variant of LTL]
- μ calculus = add “least fixed point” operator (more tomorrow)

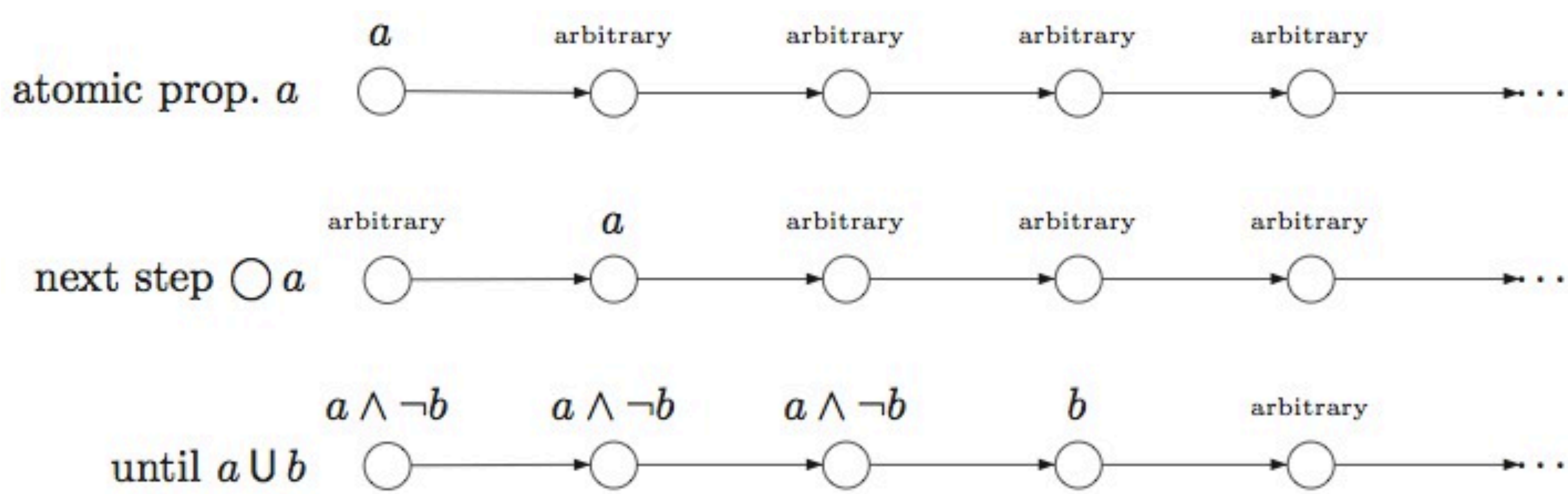
Syntax of LTL

LTL formulas:

$$\varphi ::= \text{true} \;\big|\; a \;\big|\; \varphi_1 \wedge \varphi_2 \;\big|\; \neg\varphi \;\big|\; \bigcirc\, \varphi \;\big|\; \varphi_1 \,\mathsf{U}\, \varphi_2$$

- a = atomic proposition
- $\bigcirc$ = “next”: φ is true at next step
- U = “until”: φ_2 is true at some point, φ_1 is true until that time

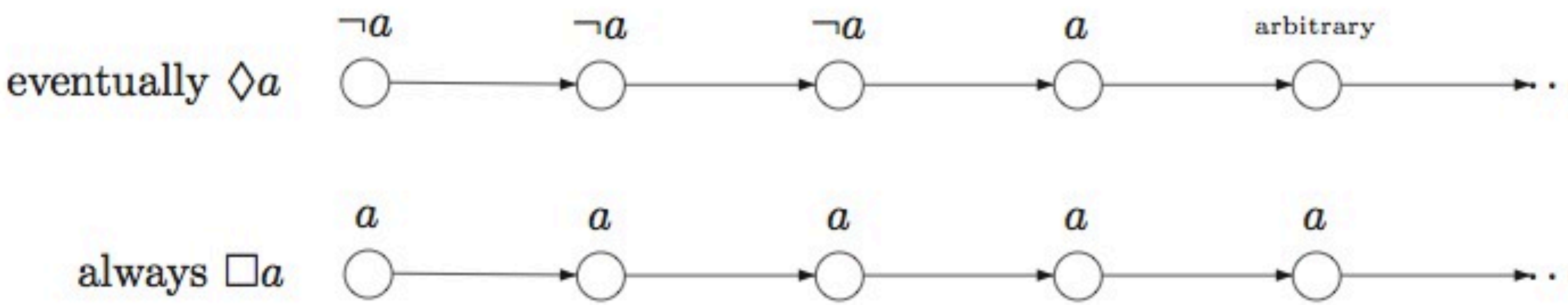
Formula evaluation: evaluate LTL propositions over a sequence of subsets of atomic propositions



Additional operators and formulas

Derived temporal logic operators

- Eventually $\Diamond\phi$:= true U ϕ ϕ will become true at some point in the future
- Always $\Box\phi := \neg\Diamond\neg\phi$ ϕ is always true; "(never (eventually ($\neg\phi$)))"



Some common composite operators

- $p \rightarrow \Diamond q$ p implies eventually q (response)
- $p \rightarrow q \text{ U } r$ p implies q until r (precedence)
- $\Box\Diamond p$ always eventually p (progress)
- $\Diamond\Box p$ eventually always p (stability)
- $\Diamond p \rightarrow \Diamond q$ eventually p implies eventually q (correlation)

Operator precedence

- Unary binds stronger than binary

  $\neg\,\varphi_1 \text{ U } \bigcirc\,\varphi_2 = (\neg\,\varphi_1)\text{U }(\bigcirc\,\varphi_2)$

- Bind from right to left:
  $\Box\Diamond p = (\Box\,(\Diamond p))$
  $p \text{ U } q \text{ U } r = p \text{ U } (q \text{ U } r)$
- U takes precedence over $\wedge$, $\vee$ and $\rightarrow$

Semantics: when does a path satisfy an LTL spec?

Let φ be an LTL formula over *AP*. The linear-time property induced by φ is

$$\mathrm{Words}(\varphi) \;=\; \left\{ \sigma \in (2^{AP})^\omega \mid \sigma \models \varphi \right\}$$

where the satisfaction relation is the smallest relation with the properties

For $\sigma = A_0\, A_1\, A_2 \ldots$, $\sigma[j\ldots] = A_j\, A_{j+1} \ldots$

$$\begin{array}{lcll}
\sigma & \models & \mathrm{true} & \\
\sigma & \models & a & \text{iff} \quad a \in A_0 \quad (\text{i.e., } A_0 \models a) \\
\sigma & \models & \varphi_1 \wedge \varphi_2 & \text{iff} \quad \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma & \models & \neg\varphi & \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models & \bigcirc\varphi & \text{iff} \quad \sigma[1\ldots] = A_1 A_2 A_3 \ldots \models \varphi \\
\sigma & \models & \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} \quad \exists j \geqslant 0.\ \sigma[j\ldots] \models \varphi_2 \ \text{ and } \ \sigma[i\ldots] \models \varphi_1, \text{ for all } 0 \leqslant i < j
\end{array}$$

For derived operators:

$$\begin{array}{lcll}
\sigma & \models & \Diamond\varphi & \text{iff} \quad \exists j \geqslant 0.\ \sigma[j\ldots] \models \varphi \\
\sigma & \models & \Box\varphi & \text{iff} \quad \forall j \geqslant 0.\ \sigma[j\ldots] \models \varphi.
\end{array}$$

$$\begin{array}{lcll}
\sigma & \models & \Box\Diamond\varphi & \text{iff} \quad \overset{\infty}{\exists} j.\ \sigma[j\ldots] \models \varphi \\
\sigma & \models & \Diamond\Box\varphi & \text{iff} \quad \overset{\infty}{\forall} j.\ \sigma[j\ldots] \models \varphi.
\end{array}$$

Sample derivation:

$$\begin{array}{lll}
\sigma \models \Box\varphi = \neg\Diamond\neg\varphi & \text{iff} & \neg\exists j \geqslant 0.\ \sigma[j\ldots] \models \neg\varphi \\
& \text{iff} & \neg\exists j \geqslant 0.\ \sigma[j\ldots] \not\models \varphi \\
& \text{iff} & \forall j \geqslant 0.\ \sigma[j\ldots] \models \varphi.
\end{array}$$

$\overset{\infty}{\exists}\ j$ means $\forall i \geqslant 0.\ \exists j \geqslant i.$

$\overset{\infty}{\forall}\ j$ means $\exists i \geqslant 0.\ \forall j \geqslant i$

Semantics: when does a system satisfy an LTL spec?

Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system without terminal states, and let φ be an LTL-formula over AP.

- For infinite path fragment π of TS, the satisfaction relation is defined by

$$\pi \models \varphi \quad \text{iff} \quad trace(\pi) \models \varphi.$$

- For state $s \in S$, the satisfaction relation $\models$ is defined by

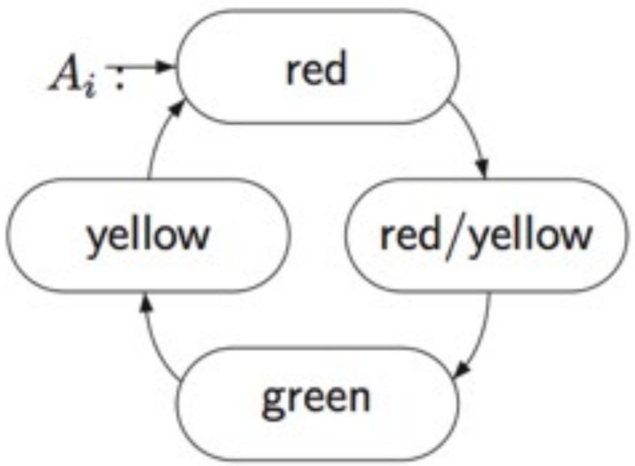
$$s \models \varphi \quad \text{iff} \quad (\forall \pi \in Paths(s).\ \pi \models \varphi).$$

- TS satisfies φ, denoted $TS \models \varphi$, if $Traces(TS) \subseteq Words(\varphi)$.

Putting together:

$TS \models \varphi$	
iff	[Bullet 3 above]
$Traces(TS) \subseteq Words(\varphi)$	
iff	[Definition of satisfaction for LT properties]
$TS \models Words(\varphi)$	
iff	[Definition of *Words*(φ)]
$\pi \models \varphi$ for all $\pi \in Paths(TS)$	
iff	[Bullet 2 above]
$s_0 \models \varphi$ for all $s_0 \in I$.	

Example: traffic light



System description

- Focus on lights in on particular direction
- Light can be any of three colors: green, yellow, read
- Atomic propositions = light color

Ordering specifications

- Liveness: “traffic light is green infinitely often”

$$\Box\Diamond \text{green}$$

- Chronological ordering: “once red, the light cannot become green immediately”

$$\Box\,(\text{red} \rightarrow \neg \bigcirc \text{green})$$

- More detailed: “once red, the light always becomes green eventually after being yellow for some time”

$$\Box(\text{red} \rightarrow (\Diamond\, \text{green} \wedge (\neg\, \text{green}\ \mathsf{U}\ \text{yellow})))$$

$$\Box(\text{red} \rightarrow \bigcirc\, (\text{red}\ \mathsf{U}\ (\text{yellow} \wedge \bigcirc\, (\text{yellow}\ \mathsf{U}\ \text{green}))))$$

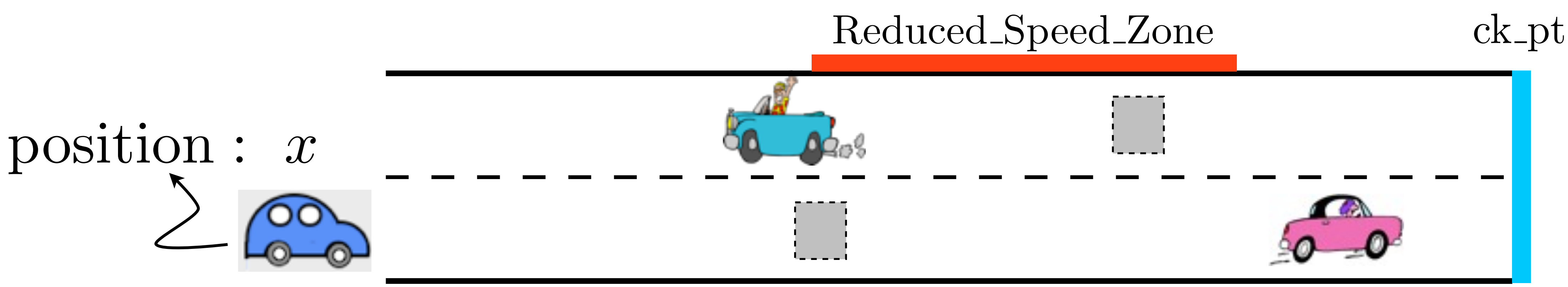
Progress property

- Every request will eventually lead to a response

$$\Box\,(\text{request} \rightarrow \Diamond \text{response})$$

Example: autonomous navigation

Specify safe, allowable, required, or desired behavior of system and/or environment.



Traffic rules:

- No collision $\Box\,(\mathrm{dist}(x, \mathrm{Obs}) \geq X_{\mathrm{safe}} \wedge \mathrm{dist}(x, \mathrm{Loc}(\mathrm{Veh})) \geq X_{\mathrm{safe}})$
- Obey speed limits $\Box\,((x \in \mathrm{Reduced\_Speed\_Zone}) \rightarrow (v \leq v_{\mathrm{reduced}}))$
- Stay in travel lane unless blocked
- Intersection precedence & merging, stop line, passing,...

Goals:

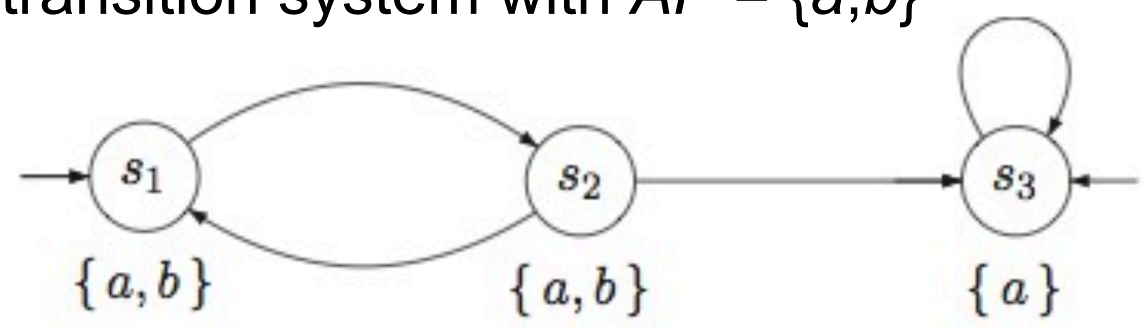
- Eventually visit the check point $\Diamond(x = \mathrm{ck\_pt})$
- Every time check point is reached, eventually come to start
  $\Box((x = \mathrm{ck\_pt}) \rightarrow \Diamond(x = \mathrm{start}))$

Environment assumptions:

- Each intersection is clear infinitely often $\Box\Diamond(\mathrm{Intersection} = \mathrm{empty})$
- Limited sensing range, detect obstacles before too late,...

"Quiz"

Consider the following transition system with *AP* = {*a*,*b*}



Property 1: TS |= [] a?

- Yes, all states are labeled with a

Property 2: TS |= X (a ^ b)?

- No: From s2 or s3, there are transitions for which a ^ b doesn't hold

Property 3: TS |= [] (!b -> [](a ^ !b))?

- True

Property 4: TS |= b U (a ^ !b)?

- False: $(s1s2)^{\omega}$

Equivalence of LTL formulas

LTL formulae φ_1, φ_2 are *equivalent*, denoted $\varphi_1 \equiv \varphi_2$, if $Words(\varphi_1) = Words(\varphi_2)$. ■

duality law $\neg \bigcirc \varphi \equiv \bigcirc \neg \varphi$ $\neg \Diamond \varphi \equiv \Box \neg \varphi$ $\neg \Box \varphi \equiv \Diamond \neg \varphi$	*idempotency law* $\Diamond\Diamond\varphi \equiv \Diamond\varphi$ $\Box\Box\varphi \equiv \Box\varphi$ $\varphi \,\mathsf{U}\, (\varphi \,\mathsf{U}\, \psi) \equiv \varphi \,\mathsf{U}\, \psi$ $(\varphi \,\mathsf{U}\, \psi) \,\mathsf{U}\, \psi \equiv \varphi \,\mathsf{U}\, \psi$
absorption law $\Diamond\Box\Diamond\varphi \equiv \Box\Diamond\varphi$ $\Box\Diamond\Box\varphi \equiv \Diamond\Box\varphi$	*expansion law* $\varphi \,\mathsf{U}\, \psi \equiv \psi \vee (\varphi \wedge \bigcirc(\varphi \,\mathsf{U}\, \psi))$ $\Diamond\psi \equiv \psi \vee \bigcirc\Diamond\psi$ $\Box\psi \equiv \psi \wedge \bigcirc\Box\psi$
distributive law $\bigcirc(\varphi \,\mathsf{U}\, \psi) \equiv (\bigcirc\varphi) \,\mathsf{U}\, (\bigcirc\psi)$ $\Diamond(\varphi \vee \psi) \equiv \Diamond\varphi \vee \Diamond\psi$ $\Box(\varphi \wedge \psi) \equiv \Box\varphi \wedge \Box\psi$	**Non-identities** • $\Diamond(a \wedge b) \not\equiv \Diamond a \wedge \Diamond b$ • $\Box(a \vee b) \not\equiv \Box a \vee \Box b$

Specifying timed properties for synchronous systems

For *synchronous* systems, LTL can be used as a formalism to specify "real-time" properties that refer to a discrete time scale. Recall that in synchronous systems, the involved processes proceed in a lock step fashion, i.e., at each discrete time instance each process performs a (sometimes idle) step. In this kind of system, the next-step operator $\bigcirc$ has a "timed" interpretation: $\bigcirc\varphi$ states that "at the next time instant φ holds". By putting applications of $\bigcirc$ in sequence, we obtain, e.g.:

$$\bigcirc^k \varphi \stackrel{\text{def}}{=} \underbrace{\bigcirc\bigcirc\ldots\bigcirc}_{k\text{-times}}\varphi \qquad \text{"}\varphi\text{ holds after (exactly) } k \text{ time instants".}$$

Assertions like "φ will hold within at most k time instants" are obtained by

$$\Diamond^{\leqslant k}\varphi = \bigvee_{0\leqslant i\leqslant k} \bigcirc^i \varphi.$$

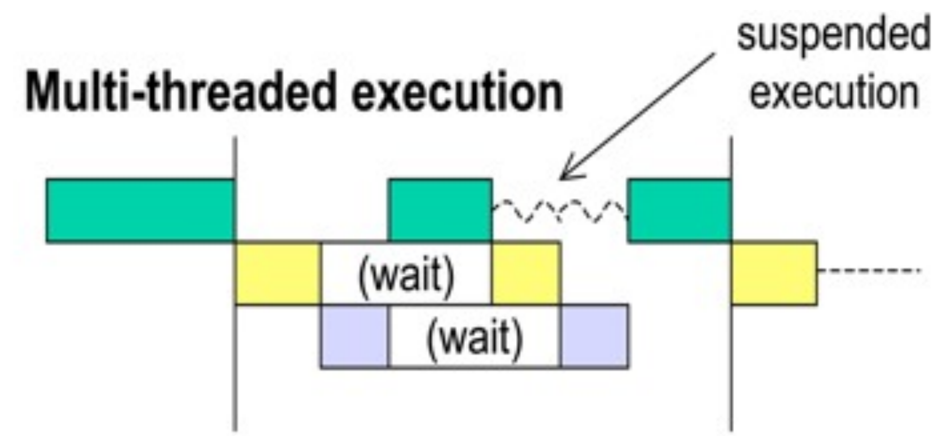
Statements like "φ holds now and will hold during the next k instants" can be represented as follows:

$$\Box^{\leqslant k}\varphi = \neg\Diamond^{\leqslant k}\neg\varphi = \neg\bigvee_{0\leqslant i\leqslant k} \bigcirc^i \neg\varphi.$$

Fairness

Mainly an issue with concurrent processes

- To make sure that the proper interaction occurs, often need to know that each process gets executed reasonably often
- Multi-threaded execution: each thread should receive some fraction of processes time
- To rule out unrealistic behavior



Examples:

- N processors sharing a service: ensure each processor gets access to the service
- In a distributed protocol, ensure that each agent communicates with its “neighbors” regularly (infinitely often)
- Autonomous car at an intersection: ensure the intersection clears or the lights turn green in the future

Two issues:

- Implementation: How do we implement our algorithms to insure that we get “fairness” in execution?
- Specification: How do we model fairness in a formal way to reason about program correctness?

Fairness properties & their LTL representation

Let Φ and Ψ be propositional logical formulas over a set of atomic propositions

Unconditional fairness	"Every process gets its turn infinitely often."	$ufair = \Box\Diamond\Psi.$
Strong fairness	"Every process that is enabled infinitely often gets its turn infinitely often."	$sfair = \Box\Diamond\Phi \longrightarrow \Box\Diamond\Psi.$
Weak fairness	"Every process that is continuously enabled from a certain time on gets its turn infinitely often."	$wfair = \Diamond\Box\Phi \longrightarrow \Box\Diamond\Psi.$

An *LTL fairness assumption*:

$$fair = ufair \wedge sfair \wedge wfair.$$

Rules of thumb

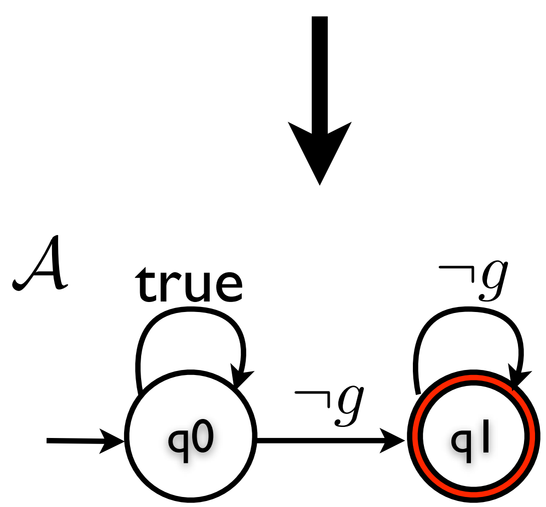
- strong (or unconditional) fairness: useful for solving contentions
- weak fairness: sufficient for resolving the non-determinism due to interleaving (i.e., a possible option is not consistently ignored)

LTL → Nondeterministic Buchi automata

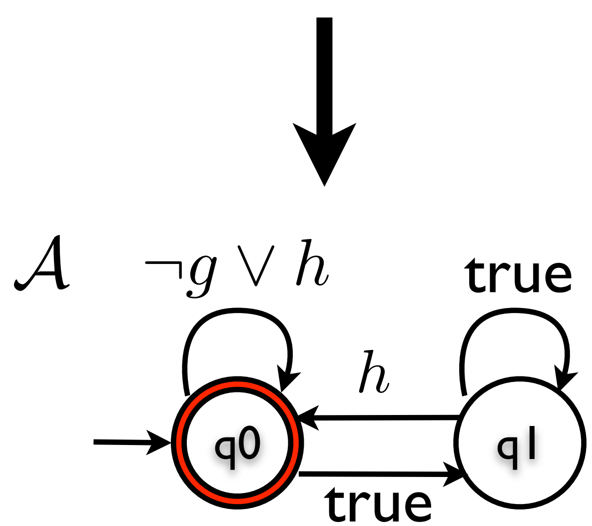
Theorem. *There exists an algorithm that takes an LTL formula Φ and returns a Büchi automaton $\mathcal{A}$ such that*

$$Words(\Phi) = \mathcal{L}_\omega(\mathcal{A})$$

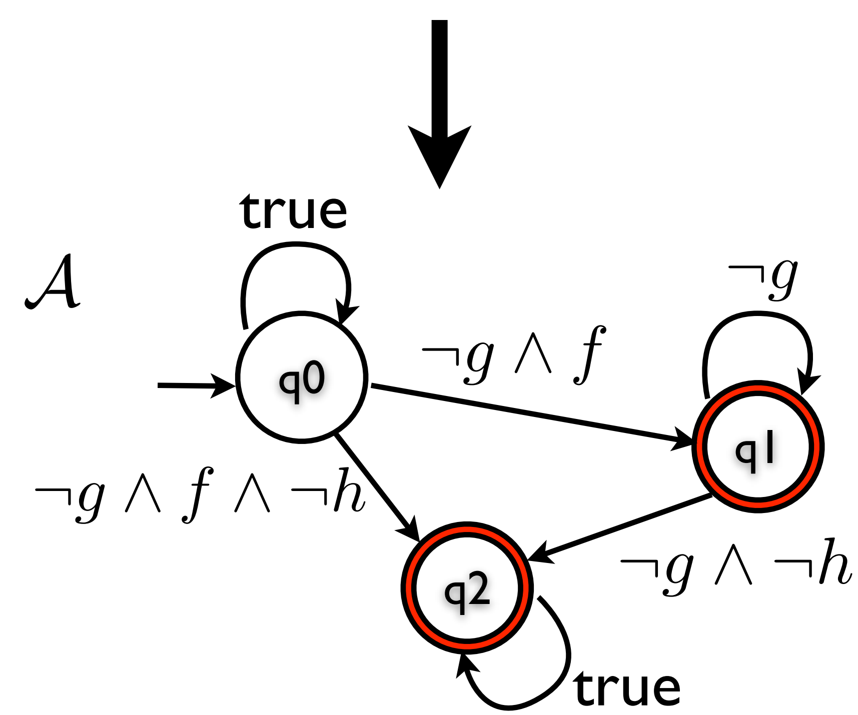
$$\Phi = \neg\Box\Diamond g$$



$$\Phi = \Box(g \to \Diamond h)$$



$$\Phi = \Diamond(f \wedge \neg(g\mathcal{U}h))$$

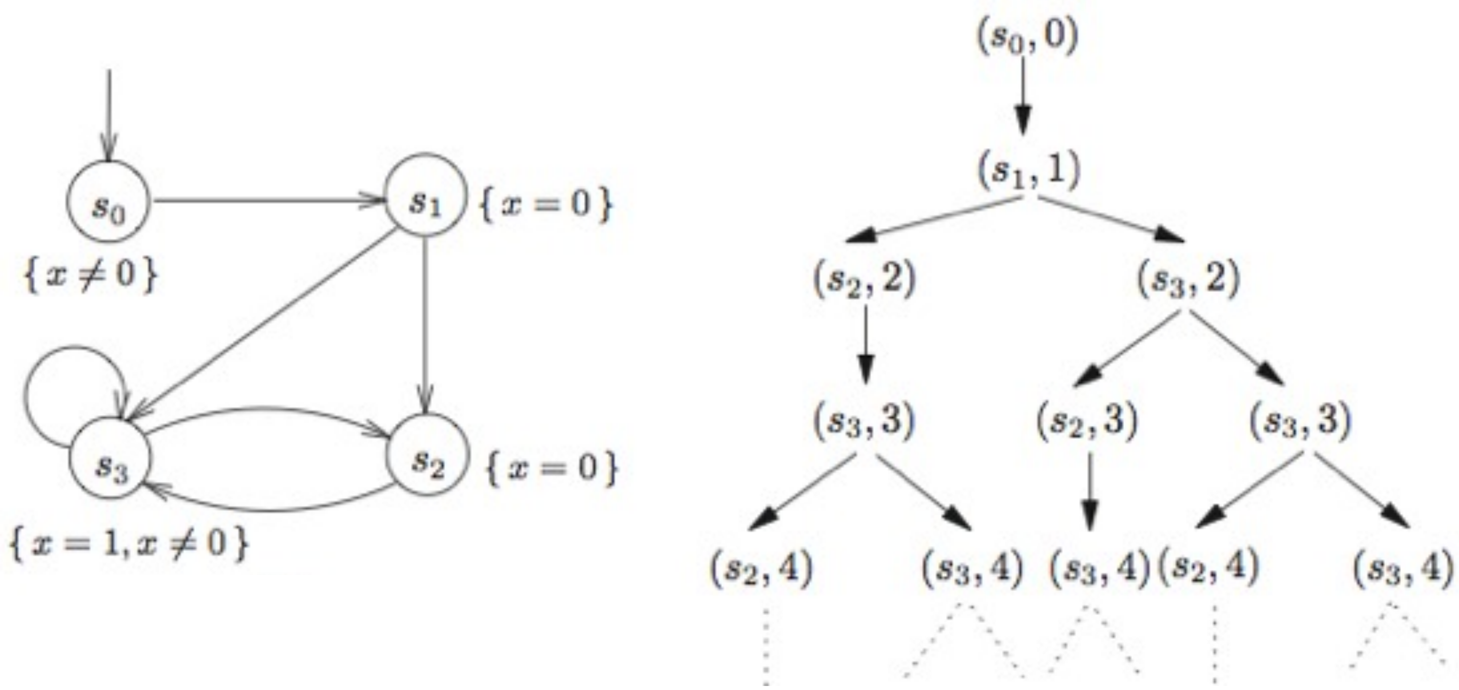


A tool for constructing Buchi automata from LTL formulas: LTL2BA [http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php]

Branching time and computation tree logic (CTL)

LTL formulas are interpreted over paths; hence, there is a clear (and linear) notion of ordering of events over time.

Interpretation an LTL formula at a state: all paths starting from the state satisfy the formula.



LTL does not allow complicated quantification over the paths.

- E.g., “For every execution it is always possible to return to the initial state” cannot be specified in LTL.

Computation tree logic (CTL) allows evaluation over some or all paths.

$$s \models \exists\varphi \quad \text{iff} \quad \pi \models \varphi \text{ for some } \pi \in \mathit{Paths}(s)$$

$$s \models \forall\varphi \quad \text{iff} \quad \pi \models \varphi \text{ for all } \pi \in \mathit{Paths}(s)$$

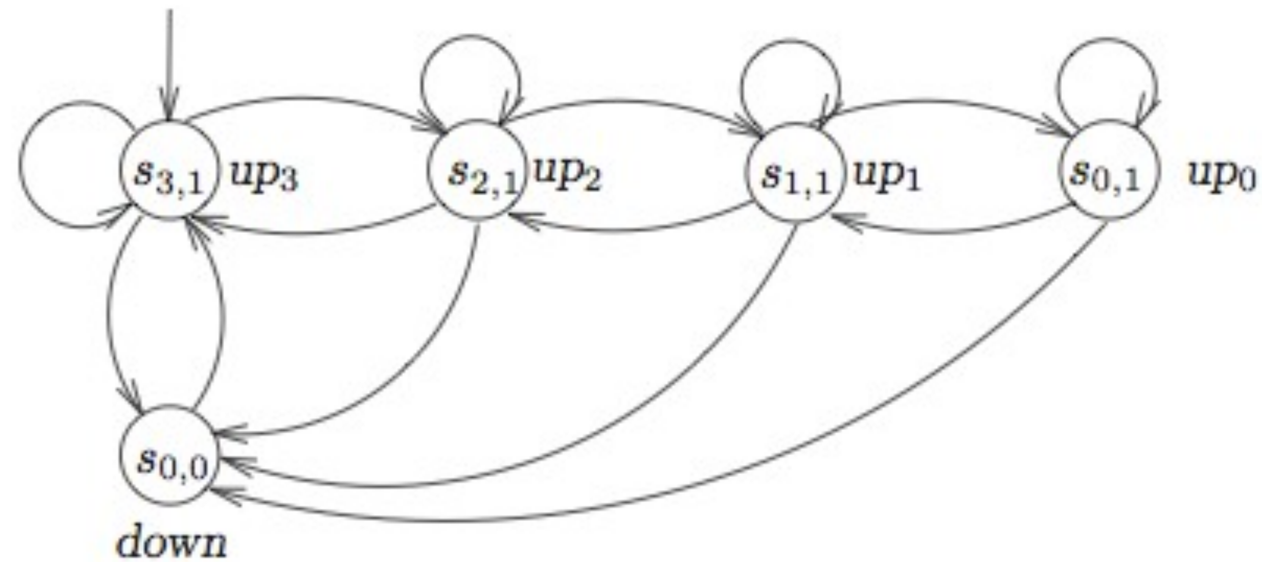
$$\forall\Box\exists\Diamond start$$

for all executions	at any state	it is possible	to eventually reach *start*

Example: triply redundant control systems

Systems consists of three processors and a single voter

- si,j = i processors up, j voters up
- Assume processors fail one at a time; voter can fail at any time
- If voter fails, reset to fully functioning state (all three processors up)
- System in operation if at least 2 processors remain operational



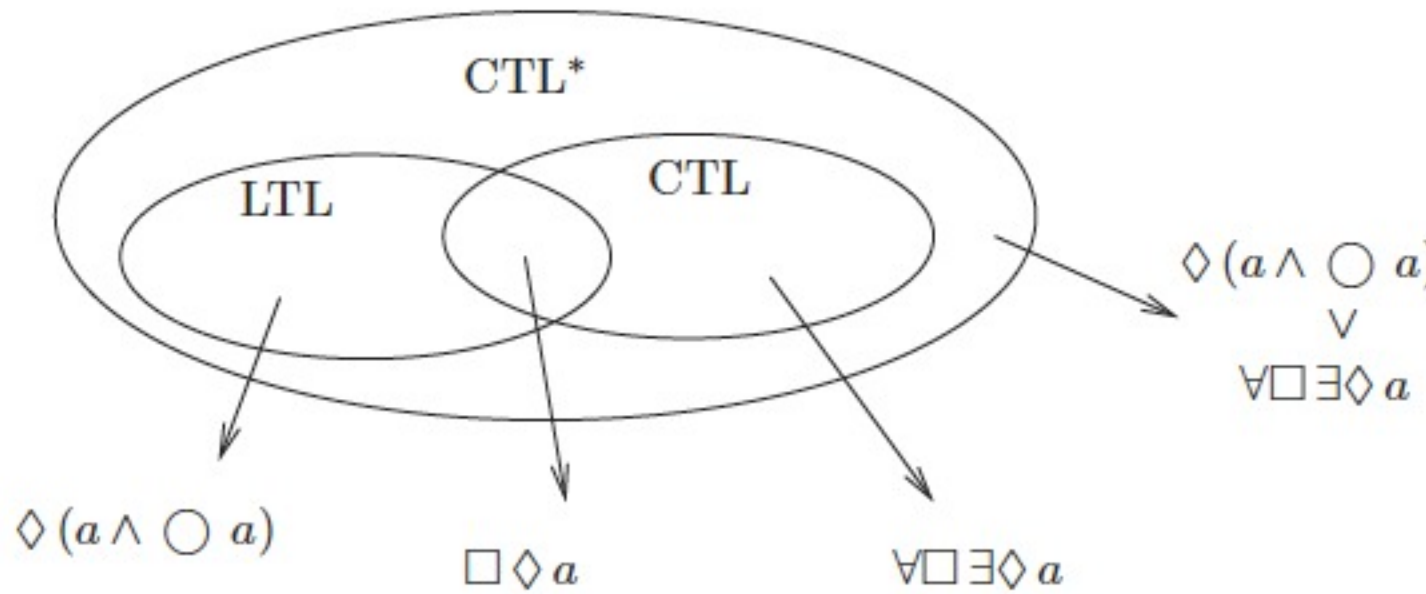
Properties we might like to prove

Property	*Formalization in CTL*	
Possibly the system never goes down	$\exists \square \neg down$	Holds
Invariantly the system never goes down	$\forall \square \neg down$	Doesn't hold
It is always possible to start as new	$\forall \square \exists \lozenge up_3$	Holds
The system always eventually goes down and is operational until going down	$\forall ((up_3 \vee up_2) \, \mathsf{U} \, down)$	Doesn't hold

Other types of temporal logic

CTL ≠ LTL

- Can show that LTL and CTL are not proper sub-sets of each other
- LTL reasons over a complete path; CTL from a give state



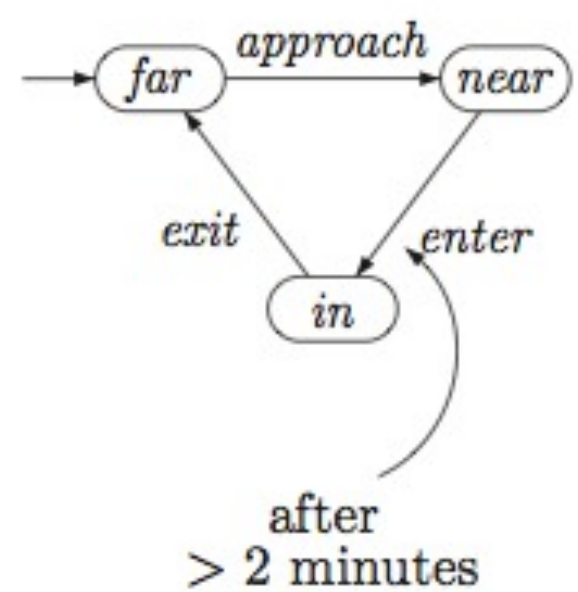
CTL* captures both

$$\Phi ::= \text{true} \;\big|\; a \;\big|\; \Phi_1 \wedge \Phi_2 \;\big|\; \neg\Phi \;\big|\; \exists\varphi \qquad \varphi ::= \Phi \;\big|\; \varphi_1 \wedge \varphi_2 \;\big|\; \neg\varphi \;\big|\; \bigcirc\varphi \;\big|\; \varphi_1 \,\mathsf{U}\, \varphi_2$$

Timed Computational Tree Logic

- Extend notions of transition systems and CTL to include “clocks” (multiple clocks OK)
- Transitions can depend on the value of clocks
- Can require that certain properties happen within a given time window

$$\forall\Box(\mathit{far} \longrightarrow \forall\Diamond^{\leqslant 1}\,\forall\Box^{\leqslant 1}\,\mathit{up})$$



Summary: specifying behavior with (linear) temporal logic

Description

- State of the system is a snapshot of values of all variables
- Reason about *paths* σ: sequence of states of the system
- No strict notion of time, just ordering of events
- *Actions* are relations between states: state *s* is related to state *t* by action *a* if *a* takes *s* to *t* (via prime notation: $x' = x + 1$)
- *Formulas* (specifications) describe the set of allowable behaviors
- Safety specification: what actions are allowed
- Fairness specification: when can a component take an action (eg, infinitely often)

Example

- Action: $a \equiv x' = x + 1$
- Behavior: $\sigma \equiv x := 1, x := 2, x := 3, \ldots$
- Safety: $\Box x > 0$ (true for this behavior)
- Fairness: $\Box(x' = x + 1 \vee x' = x) \wedge \Box\Diamond (x' \neq x)$

- $\Box p \equiv$ **always** *p* (invariance)
- $\Diamond p \equiv$ **eventually** *p* (guarantee)
- $p \rightarrow \Diamond q \equiv$ *p* **implies eventually** *q* (response)
- $p \rightarrow q \,\mathcal{U}\, r \equiv$ *p* **implies** *q* **until** *r* (precedence)
- $\Box\Diamond p \equiv$ **always eventually** *p* (progress)
- $\Diamond\Box p \equiv$ **eventually always** *p* (stability)
- $\Diamond p \rightarrow \Diamond q \equiv$ **eventually** *p* **implies eventually** *q* (correlation)

Properties

- Can reason about time by adding "time variables" ($t' = t + 1$)
- Specifications and proofs can be difficult to interpret by hand, but computer tools existing (eg, TLC, Isabelle, PVS, SPIN, nuSMV, etc)