Linear Temporal Logics

Describe LT Properties by LTL

Motivations

- It is not practical to write down $P \subseteq \left(2^{AP}\right)^{\omega}$ directly
- Propositional and predicate logics are "static"
- How to express temporal properties in a structured, user-friendly and rigorous manner

Approach

- Use linear temporal logics (LTL)
- Introduce temporal operators in addition to Boolean operators

LTL Syntax

A (propositional) **Linear Temporal Logic (LTL)** formula ϕ over a given set of atomic propositions AP is recursively defined as

$$\phi ::= \mathrm{TRUE} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 U \phi_2$$

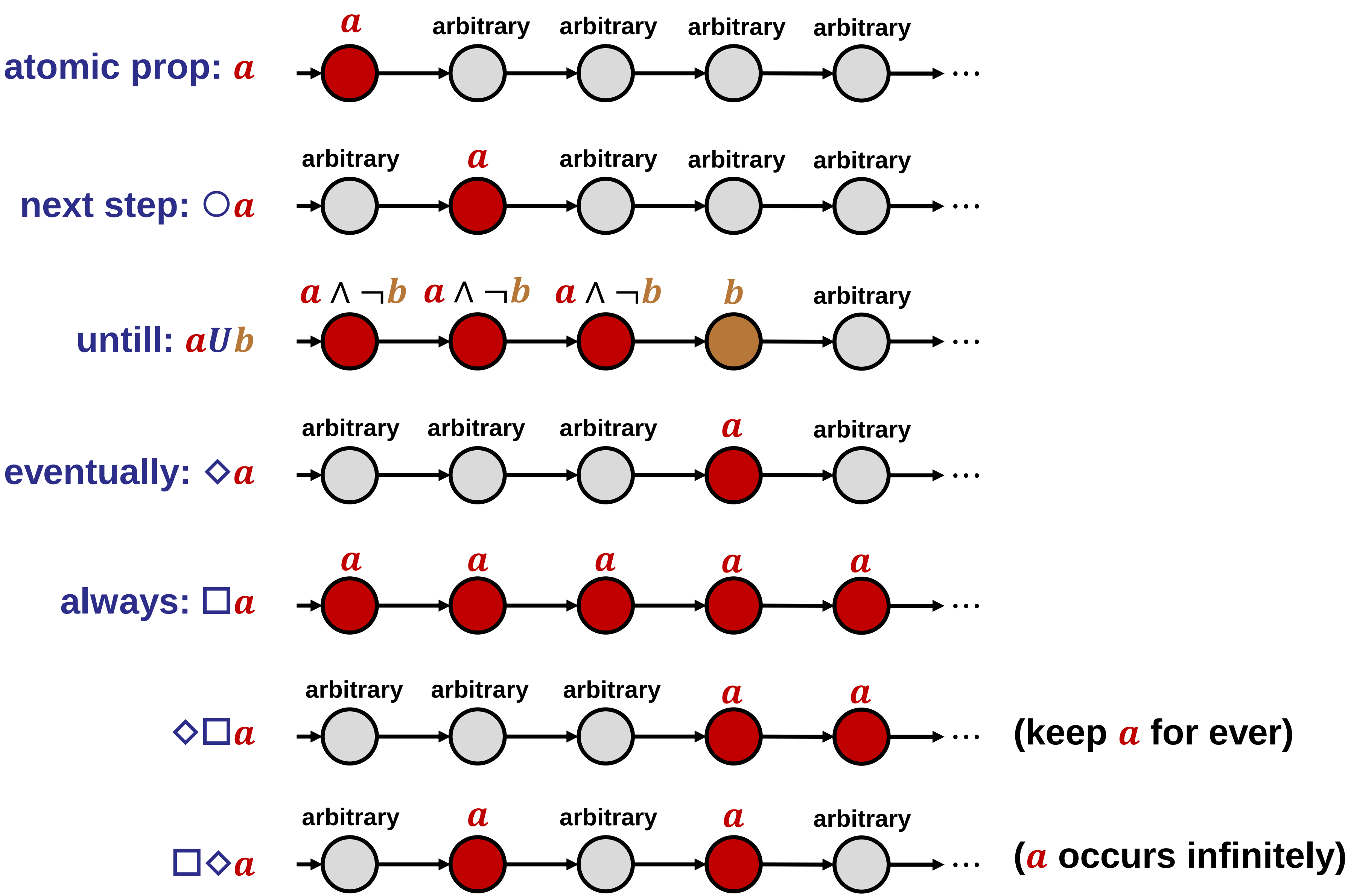
where a is an atomic proposition and ϕ, ϕ_1 and ϕ_2 are LTL formulas.

- Formula $\bigcirc\phi$ holds at the current moment, if ϕ holds in the **next** "step"
- Formula $\phi_1 U \phi_2$ holds at the current moment, if there is some future moment for which ϕ_2 holds and ϕ_1 holds at all moments **until** that future moment.

We can also define the following operators

- Boolean oper.: "**or**" $\phi_1 \vee \phi_2 \coloneqq \neg(\neg\phi_1 \wedge \neg\phi_2)$, "**implies**" $\phi_1 \to \phi_2 \coloneqq \neg\phi_1 \vee \phi_2$
- Temporal oper.: "**eventually**" $\Diamond\phi \coloneqq \mathrm{TRUE}\ U\ \phi$, "**always**" $\Box\phi \coloneqq \neg\Diamond\neg\phi$

LTL Semantics: Informal



LTL Semantics: Formal

- LTL formulas are used to evaluate infinite words over 2^{AP}
- Let $\sigma = A_0 A_1 A_2 \cdots \in \left(2^{AP}\right)^{\omega}$, define $\sigma[j\cdots] = A_j A_{j+1} A_{j+2} \cdots$
- Infinite word σ satisfies formula ϕ, denoted by $\sigma \vDash \phi$, is defined by

$\sigma \vDash \mathrm{True}$		
$\sigma \vDash a$	iff	$a \in A_0$ (i.e., $A_0 \vDash a$)
$\sigma \vDash \phi_1 \wedge \phi_2$	iff	$\sigma \vDash \phi_1$ and $\sigma \vDash \phi_2$
$\sigma \vDash \neg\phi$	iff	$\sigma \nvDash \phi$
$\sigma \vDash \bigcirc\phi$	iff	$\sigma[1\cdots] = A_1 A_2 A_3 \cdots \vDash \phi$
$\sigma \vDash \phi_1 U \phi_2$	iff	$\exists j \geq 0: \sigma[j\cdots] \vDash \phi_2$ and $\forall 0 \leq i < j: \sigma[i\cdots] \vDash \phi_1$

- The set of all words satisfying ϕ is $Word(\phi) = \{\sigma \in \left(2^{AP}\right)^{\omega} : \sigma \vDash \phi\}$, i.e., $\sigma \vDash \phi$ iff $\sigma \in Word(\phi)$

LTL Example: Mutual Exclusion

- **The safety property stating that P_1 and P_2 never simultaneously have access to their critical sections**

$$\Box(\neg crit_1 \vee \neg crit_2)$$

- **The liveness requirement stating that each process P_i is infinitely often in its critical section**

$$(\Box\Diamond crit_1) \wedge (\Box\Diamond crit_2)$$

- **The strong fairness requirement stating that infinitely waiting process will eventually enter its critical section infinitely often:**

$$(\Box\Diamond wait_1 \rightarrow \Box\Diamond crit_1) \wedge (\Box\Diamond wait_2 \rightarrow \Box\Diamond crit_2)$$

LTL Example: Traffic Light

- The traffic light is infinitely often green

$$\Box\Diamond green$$

- Once red, the light cannot become green immediately

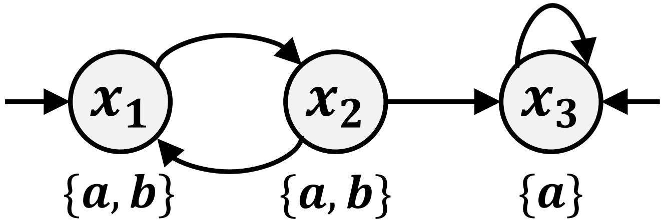
$$\Box(red \rightarrow \neg \bigcirc green)$$

- Once red, the light always becomes green eventually after being yellow for some time

$$\Box(red \rightarrow \bigcirc(red\ U\ (yellow \wedge \bigcirc(yellow\ U\ green))))$$

LTL Semantics on LTSs

- LTL formula ϕ evaluates infinite words over 2^{AP}
- LTS T generates a set of infinite words (traces) from initial states
- A state $x \in X$ in T satisfies ϕ, denoted by $x \vDash \phi$, if all traces generated from x satisfy ϕ
- We say LTL T satisfies ϕ, denoted by $T \vDash \phi$, if all its initial stats satisfy ϕ, i.e., $Trace(T) \subseteq Word(\phi)$

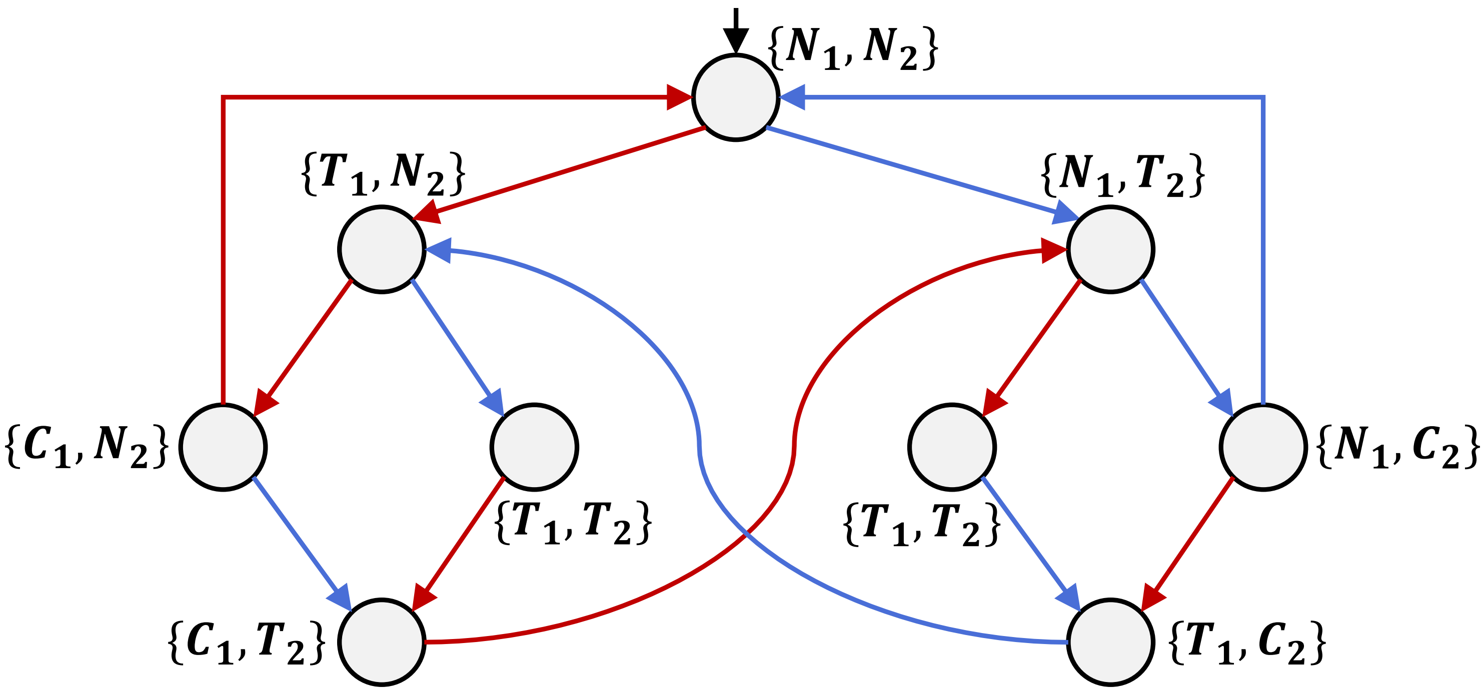


LTS T

- $T \vDash \Box a$
- $T \nvDash \bigcirc(a \wedge b)$ since $x_1 \vDash \bigcirc(a \wedge b)$ but $x_3 \nvDash \bigcirc(a \wedge b)$
- $T \vDash \Box(\neg b \rightarrow \Box(a \wedge \neg b))$
- $T \nvDash b\ U\ (a \wedge \neg b)$ since $\{a, b\}^{\omega} \nvDash b\ U\ (a \wedge \neg b)$

How to check whether $T \vDash \phi$ or not?

LTL Example: Mutually Exclusive Processes



- $T \vDash \Box(T_1 \rightarrow \Diamond C_1)$? **Yes!**
- $T \vDash \Box\Diamond C_1$ **No! Consider trace** $(\{N_1, N_2\}\{N_1, T_2\}\{N_1, C_2\})^\omega$
- $T \vDash \Box\Diamond T_1 \rightarrow \Box\Diamond C_1$? **Yes!**

Co-Safe LTL Syntax

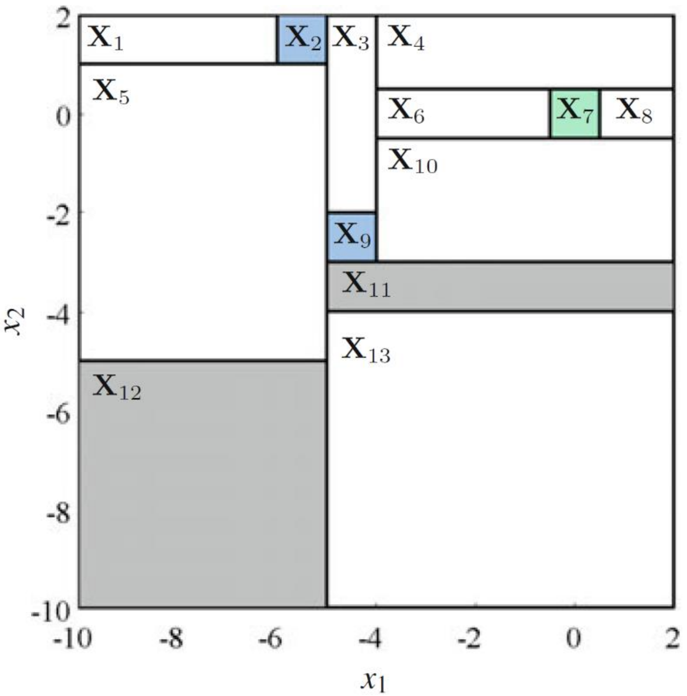
A (propositional) Co-Safe Linear Temporal Logic (scLTL) formula ϕ over a given set of atomic proposition AP is recursively defined as

$$\phi ::= \mathrm{TRUE} \mid a \mid \neg a \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \bigcirc\phi \mid \phi_1 U \phi_2$$

where a is an atomic proposition and ϕ, ϕ_1 and ϕ_2 are LTL formulas.

- Negation can only be used for atomic propositions not a general formula
- "Always" cannot be expressed since $\Box\phi \coloneqq \neg\Diamond\neg\phi$ is not well defined
- We can only use temporal operators $\bigcirc, U$ and $\Diamond$
- Any infinite word satisfying scLTL ϕ has a finite "good" prefix such that any infinite continuation of this good prefix satisfies ϕ
- Denote $\mathcal{L}_{pref,\phi}$ as the set of finite good prefixes of scLTL formula ϕ

Example: Co-Safe LTL Syntax



Consider an agent moving in the planar environment

- Visit regions X_2 or X_9 and then the target region X_7, while avoiding X_{11} and X_{12}, and staying inside of $X = [-10 \;\; 2]^2$ until the target region is reached.

$$\phi = ((\neg X_{11} \wedge \neg X_{12} \wedge \neg Out)\ U\ X_7) \wedge (\neg X_7\ U\ (X_2 \vee X_9))$$

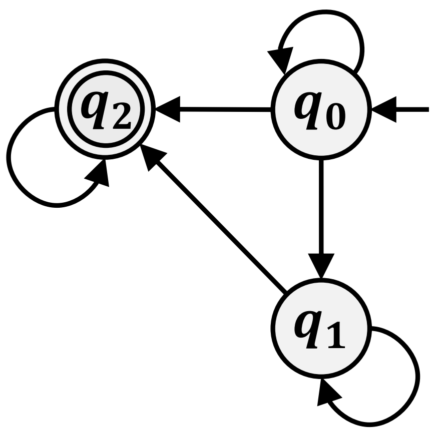
- Good prefix, e.g., $X_2 X_3 X_4 X_7$ or $X_2 X_3 X_9 X_3 X_9 X_{10} X_8 X_7$
- In general, there may have infinite many finite good prefixes

Computation Tree Logic

- **LTL implicitly quantifies universally over paths**

  $\langle T, x \rangle \vDash \phi$ **iff for every path** π **starting at** x**, we have** $\langle T, \pi \rangle \vDash \phi$

- **Properties that assert the existences of a path cannot be expressed, e.g., always has the possibility to reach some states.**



- **I always have the opportunity to reach** q_2
- **Cannot be expressed by LTL!**

Computation Tree Logic

- LTL implicitly quantifies universally over paths

  $\langle T, x \rangle \vDash \phi$ iff for every path π starting at x, we have $\langle T, \pi \rangle \vDash \phi$

- Properties that assert the existences of a path cannot be expressed, e.g., always has the possibility to reach some states.

- The computation tree logic (CTL) solves this problem. The idea is to evaluate over branching-time structures (trees) with path quantifiers:

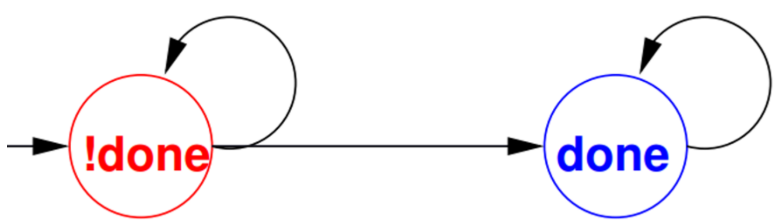
  - For All Paths: A
  - Exists a Path: E
  - Every temporal operator preceded by a path quantifier
  - Notation: $\Box \rightsquigarrow G$ globally in the future

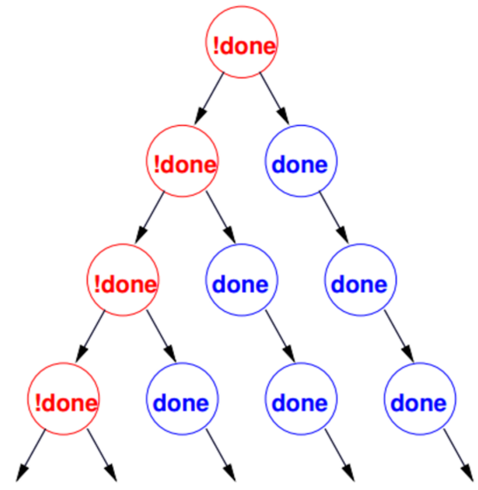
    $\bigcirc \rightsquigarrow X$ next time

    $\Diamond \rightsquigarrow F$ sometime in the future

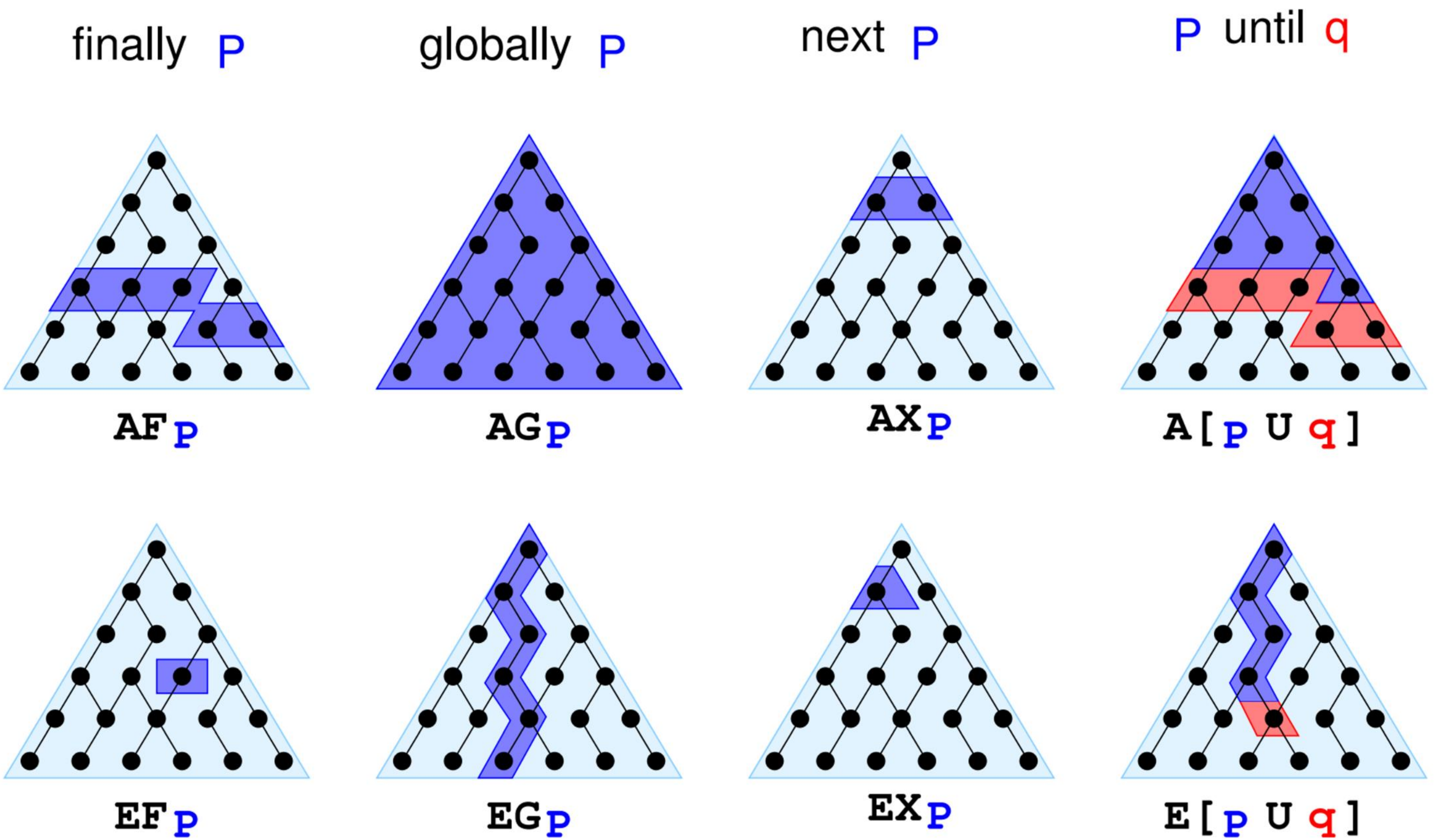
CTL Semantics: Intuitions

- **Globally:** $AG\phi$ is true iff ϕ is always true in the future
- **Necessarily Next:** $AX\phi$ is true iff ϕ is true in every successor state
- **Possibly Next:** $EX\phi$ is true iff ϕ is true in some successor state
- **Necessarily in the Future:** $AF\phi$ is true iff ϕ is inevitably true in some future time
- **Possibly in the Future:** $EF\phi$ is true iff ϕ maybe true in some future time





CTL Semantics: Intuitions



Stage Summary

- LTL provides an user-friendly way for writing down LT properties
- LTL = Temporal operators + Boolean operators
- LTL formulas only evaluate infinite words
- Co-safe LTL can be satisfied in finite horizon
  (recall safety is something that can be violated in finite horizon)
- LTL cannot capture branching-time properties; need CTL
- CTL puts quantifiers for states to capture branching-time properties

Question

- What is $AGEF\phi$?
  - A for all paths
  - E exists a path
  - G globally in the future
  - F sometime in the future

Review of Last Lecture

- **LTL = Temporal operators + Boolean operators**
- **LTL formulas only evaluate infinite words**
- **Co-safe LTL can be satisfied in finite horizon**
- **LTL cannot capture branching-time properties; need CTL**
- **CTL puts quantifiers for states to capture branching-time properties**
- **LTL only tells how to describe a property; it does not tell how to generate the underlying property (language)** $Word(\phi) \subseteq \left(2^{AP}\right)^{\omega}$
- **We use Automata to generate languages describing good behaviors**