# Cyber-Physical Systems:

## Temporal Logic

March 27, 2019

Lecture @ University of Trieste

Jyo Deshmukh

$$\square_{[1,3]}(x > 0) \wedge \Diamond_{[0,0.001]}(y < 0) \Rightarrow (x > 1) \vee (x < -1)$$

$$p_1\, \mathcal{U}_{(a_1,b_1)}\, (p_2\, \mathcal{U}_{(a_2,b_2)}\, (p_3\, \mathcal{U}_{(a_3,b_3)}\, (p_4\, \mathcal{U}_{(a_4,b_4)}\, \mathcal{G}\, p_5)))$$

$$\square_{[1,3]}(x > 0) \wedge \Diamond_{[0,0.001]}(y < 0) \Rightarrow (x > 1) \vee (x < -1)$$

$$\square_{[1,3]}(x > 0) \vee (x < -1)$$

$$\square_{[1,3]}(x > 0) \Rightarrow \Diamond_{[1,3]}((y > 0) \wedge \Diamond_{[0,0.001]}(y < 0) \Rightarrow (x > 1) \vee (x < -1)$$

USC Viterbi
School of Engineering
*Department of Computer Science*

# Specifications/Requirements

- Specifications for most programs: functional
  - Program starts in some state $q$, and terminates in some other state $r$, specification defines a relation between all pairs $(q, r)$ given $q, r \in Q$

- Specifications for reactive and cyber-physical systems:
  - Program never terminates!
  - Starting from some initial state (say $q$), all infinite behaviors of the program should satisfy certain property

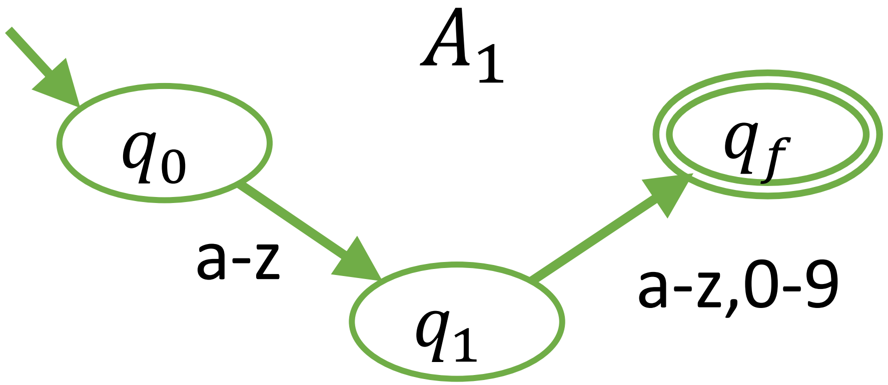
Small detour

# Detour to automata and formal languages

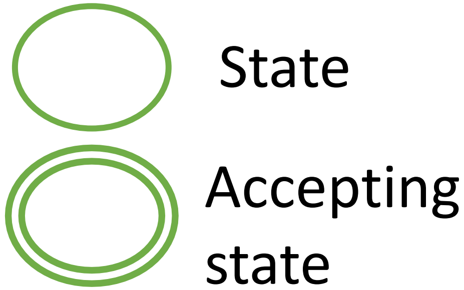
- Most programmers have used regular expressions
- Formally, regular expressions specify acceptable sequences of ***finite*** length
- Example:
  - [a-z][a-z 0-9] : strings starting with a lowercase letter (a-z) followed by ***one*** lowercase letter or number
  - [a-z][0-9]*[a-z] : strings starting with a lowercase letter, followed by ***finitely many*** numbers followed by a lowercase letter

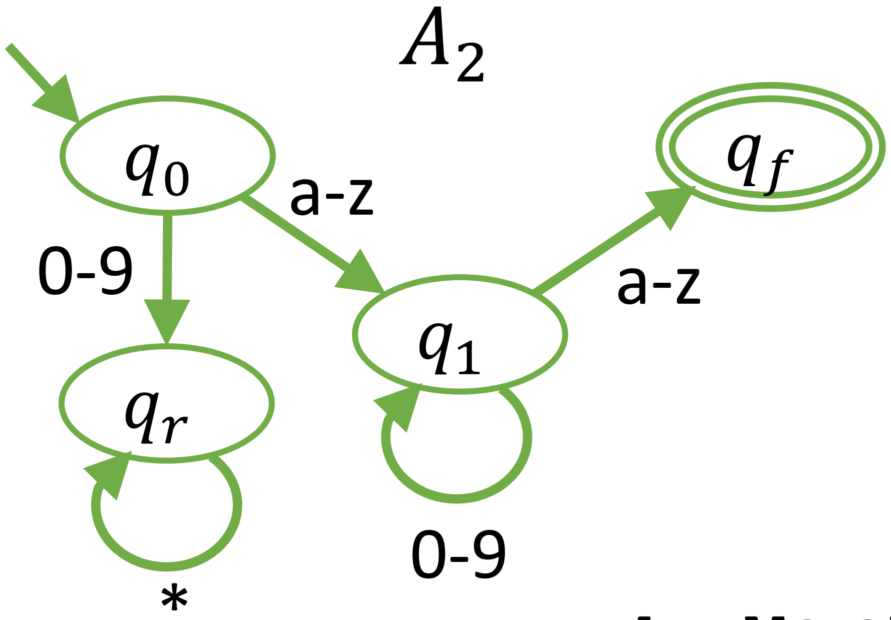
# Finite state automata

- Famous equivalence between finite state automata and regular expressions



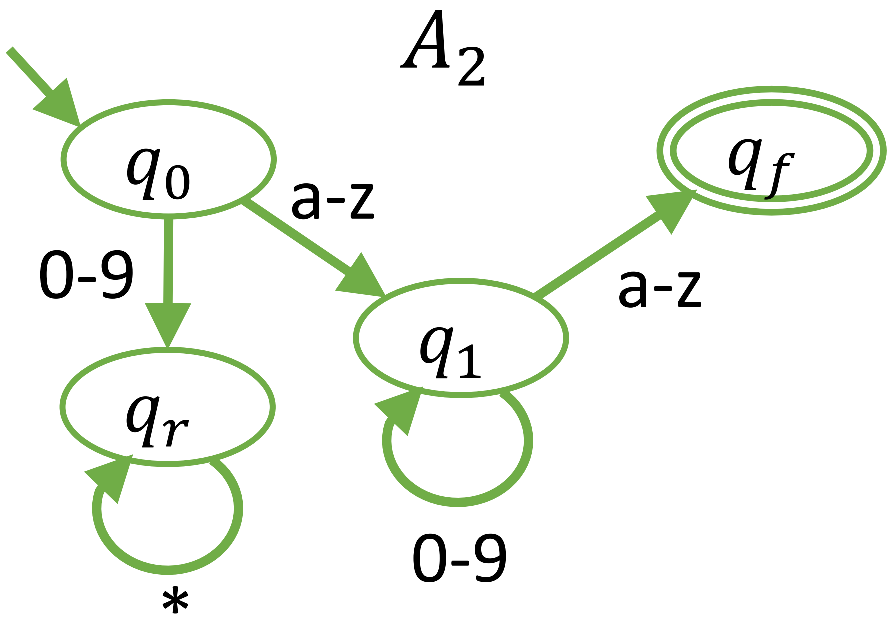

[a-z][a-z 0-9 ]



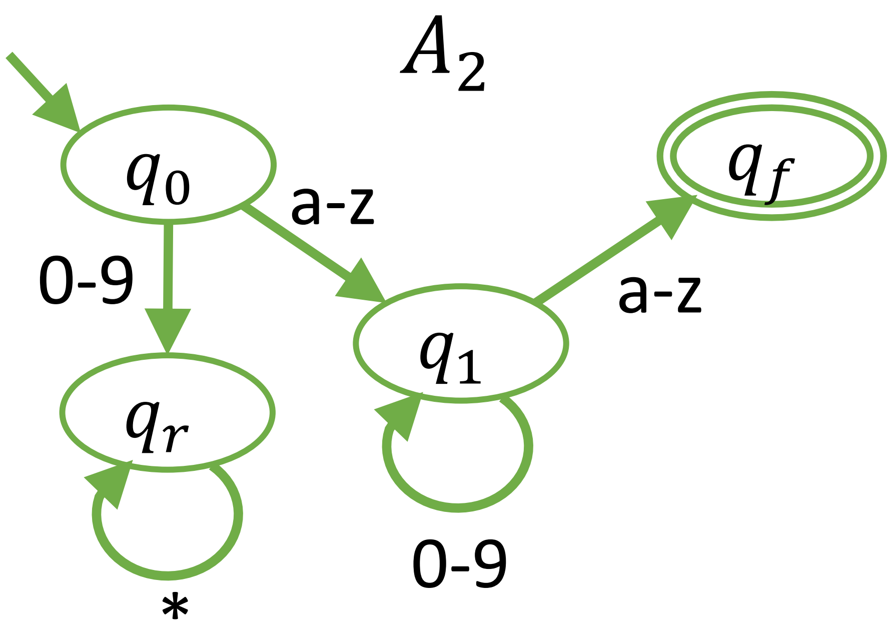

[a-z][0-9]*[a-z]

# How does a finite state automaton work?




- Starts at the initial state $q_0$
- In $q_0$, if it receives a letter in a-z, goes to $q_1$
  else, it goes to $q_r$
- In $q_1$, if it receives a number in 0-9, it stays in $q_1$
  else, it goes to $q_f$ (as it received a-z)
- In $q_r$, no matter what it gets, it stays in $q_r$
- $q_f$ is an accepting state where computation halts
- Any string that takes the machine from $q_0$ to $q_f$ is ***accepted*** by the machine

# Language of a finite state automaton




- What strings are accepted by $A_2$?
  - ab, zy, s2r, q123s, u3123123v, etc.
- What strings are not accepted by $A_2$?
  - 2b, 334a, etc.
- The set of all strings accepted by $A_2$ is called its ***language***
- The language of a finite state automaton consists of strings, each of which can be arbitrarily long, ***but finite***

LTL

# Temporal Logic

- Temporal Logic (literally logic of time) allows us to specify infinite sequences of states using logical formulae
- Amir Pnueli in 1977 used a form of temporal logic called Linear Temporal Logic (LTL) for requirements of reactive systems: later selected for the 1996 Turing Award
- Clarke, Emerson, Sifakis in 2007 received the Turing Award for the model checking algorithm, originally designed for checking Computation Tree Logic (CTL) properties of distributed programs

# What is a logic in context of today's lecture?

- Syntax: A set of syntactic rules that allow us to construct formulas from specific ground terms
- Semantics: A set of rules that assign meanings to well-formed formulas obtained by using above syntactic rules
- Simplest form is Propositional Logic

# Propositional Logic

- Simplest form of logic with a set of atomic propositions and Boolean connectives
- $AP = \{p, q, r, \ldots\}$, Connectives = $\wedge, \vee, \neg, \Rightarrow, \equiv$
- Syntax recursively gives how new formulae are constructed from smaller formulae

**Syntax of Propositional Logic**

| | | | | |
|---|---|---|---|---|
| $\varphi$ | $::=$ | $true$ | \| | the true formula |
| | | $p$ | \| | $p$ is a prop in AP |
| | | $\neg\varphi$ | \| | Negation |
| | | $\varphi \wedge \varphi$ | \| | Conjunction |
| | | $\varphi \vee \varphi$ | \| | Disjunction |
| | | $\varphi \Rightarrow \varphi$ | \| | Implication |
| | | $\varphi \equiv \varphi$ | \| | Equivalence |

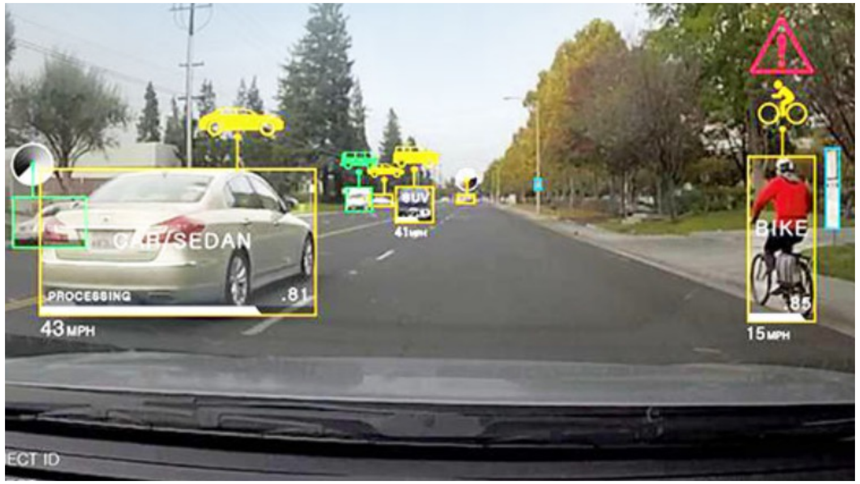
# Semantics

- Semantics (i.e. meaning) of a formula can be defined recursively
- Semantics of an atomic proposition defined by a ***valuation*** function $\nu$
- Valuation function assigns each proposition a value 1 (true) or 0 (false), always assigns the $true$ formula the value 1, and for other formulae is defined recursively

| Semantics of Prop. Logic | |
|---|---|
| $\nu(true)$ | 1 |
| $\nu(\neg\varphi)$ | 1 if $\nu(\varphi) = 0$<br>0 if $\nu(\varphi) = 1$ |
| $\nu(\varphi_1 \wedge \varphi_2)$ | 1 if $\nu(\varphi_1) = 1$ and $\nu(\varphi_2) = 1$,<br>0 otherwise |
| $\varphi_1 \vee \varphi_2$ | $\nu(\neg(\neg\varphi_1 \wedge \neg\varphi_2))$ |
| $\varphi_1 \Rightarrow \varphi_2$ | $\nu(\neg\varphi_1 \vee \varphi_2)$ |
| $\varphi_1 \equiv \varphi_2$ | $\nu\big((\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)\big)$ |

# Examples

- $p$ : There is an upright bicycle in the middle of the road
- $q$ : There is car in the field of vision
- $p \Rightarrow r$: If there is an upright bicycle in the middle of the road, the bicycle has a rider
- $o_i$: Car $i$ is in the intersection
- $(o_1 \wedge \neg o_2) \vee (\neg o_1 \wedge o_2)$



# Interpreting a formula of prop. logic

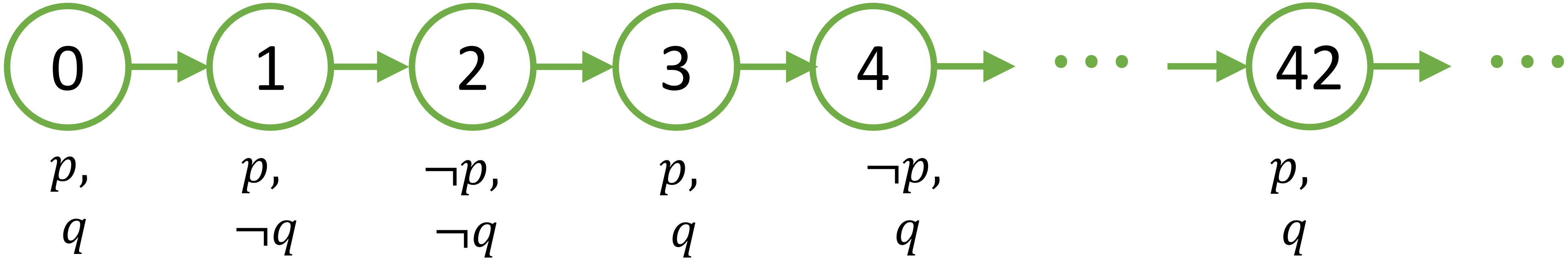
- $v: p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0$. What is $v\big((p_1 \wedge p_2) \Rightarrow p_3\big)$?
- $v\big((p_1 \wedge p_2) \Rightarrow p_3\big) = 1$

- $v: p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0$. What is $v\big((p_1 \Rightarrow p_3) \wedge (p_2 \Rightarrow p_3)\big)$
- $v\big((p_1 \Rightarrow p_3) \wedge (p_2 \Rightarrow p_3)\big) = 0$

- Is this true? $v\left(\underline{(p_1 \wedge p_2) \Rightarrow p_3} \equiv \underline{(p_1 \Rightarrow p_3) \wedge (p_2 \Rightarrow p_3)}\right) = 1$? (For all valuations)?

# Temporal Logic = Prop. Logic + Temporal Operators

- Propositional Logic is interpreted over valuations to atoms
- Temporal Logic is interpreted over traces/sequences/strings
- Trace is an infinite sequence of valuations
- $\rho$:




- Can also write as: (0,1,1), (1,1,0), (2,0,0), (3,1,1),(4,0,1),… ,(42,1,1), …

# Linear Temporal Logic

- LTL is a logic interpreted over infinite traces
- Temporal logic with a view that time evolves in a linear fashion
  - Other logics where time is branching!
- Assumes that a trace is a discrete-time trace, with equal time intervals
- Actual interval between time-points does not matter : similar to rounds in synchronous reactive components
- LTL can be used to express safety and liveness properties!

# LTL Syntax

- LTL formulas are built from propositions and other smaller LTL formulas using:
  - Boolean connectives
  - Temporal Operators
- Only shown $\wedge$ and $\neg$, but can define $\vee, \Rightarrow, \equiv$ for convenience

| Syntax of LTL | | |
|---|---|---|
| $\varphi ::=$ $p$ | \| | $p$ is a prop in AP |
| $\neg\varphi$ | \| | Negation |
| $\varphi \wedge \varphi$ | \| | Conjunction |
| $\mathbf{X}\varphi$ | \| | NeXt Step |
| $\mathbf{F}\varphi$ | \| | Some Future Step |
| $\mathbf{G}\varphi$ | \| | Globally in all steps |
| $\varphi\ \mathbf{U}\ \varphi$ | \| | In all steps Until in some step |

# LTL Semantics

- Semantics of LTL is defined by a valuation function that assigns to each proposition at each time-point in the trace a truth value (0 or 1)
- We use the symbol $\vDash$ (read models) to show that a trace-point satisfies a formula
- $\rho, n \vDash \varphi$ : Read as trace $\rho$ at time $n$ satisfies formula $\varphi$
- If we omit $n$, then the meaning is time 0. I.e. $\rho \vDash \varphi$ is the same as $\rho, 0 \vDash \varphi$
- Semantics is defined recursively over the formula
- Base case: Propositional formulas, Recursion over structure of formula

# Recursive semantics of LTL: I

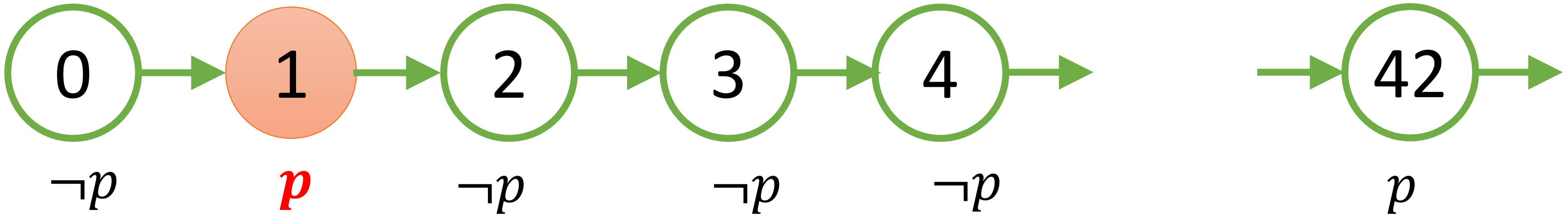
- $\rho, n \vDash p$ if $v_n(p) = 1$,
  - i.e. if $p$ is true at time $n$
- $\rho, n \vDash \neg\varphi$ if $\rho, n \nvDash \varphi$,
  - i.e. if $\varphi$ is ***not*** true for the trace starting time $n$
- $\rho, n \vDash \varphi_1 \wedge \varphi_2$ if $\rho, n \vDash \varphi_1$ and $\rho, n \vDash \varphi_2$
  - i.e. if $\varphi_1$ and $\varphi_2$ ***both hold*** starting time $n$
- $\rho, n \vDash \mathbf{X}\varphi$ if $\rho, n+1 \vDash \varphi$
  - i.e. if $\varphi$ holds starting at the next time point

# Recursive semantics of LTL: II

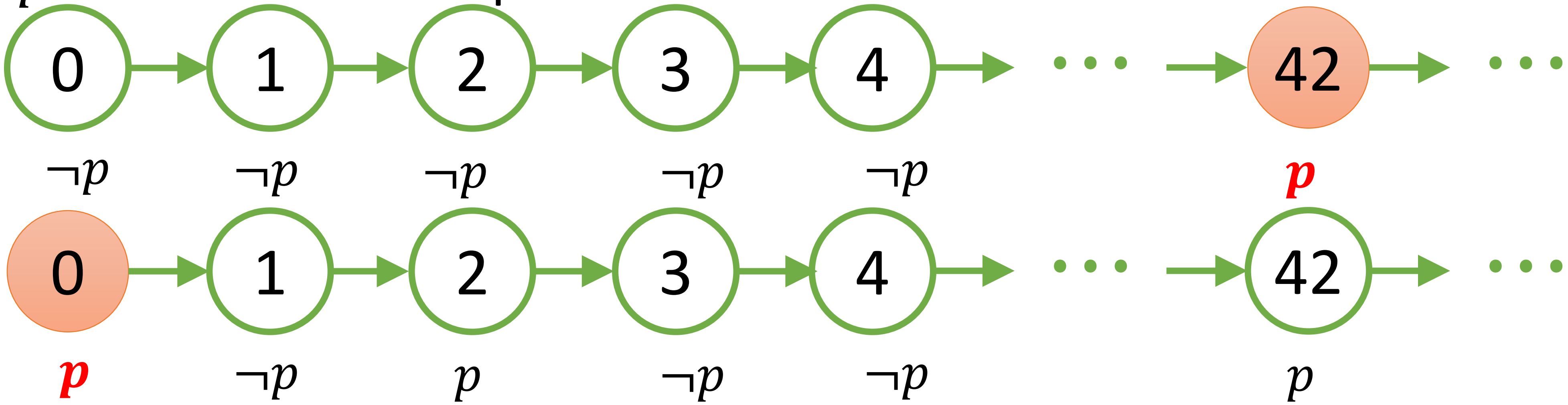
- $\rho, n \models \mathbf{F}\,\varphi$ if $\exists m \geq n$ such that $\rho, m \models \varphi$
  - i.e. $\varphi$ is true starting now, or there is some future time-point $m$ from where $\varphi$ is true
- $\rho, n \models \mathbf{G}\,\varphi$ if $\forall m \geq n : \rho, m \models \varphi$
  - i.e. $\varphi$ is true starting now, and for all future time-points $m$, $\varphi$ is true starting at $m$
- $\rho, n \models \varphi_1 \mathbf{U} \varphi_2$ if $\exists m \geq n$ s.t. $\rho, m \models \varphi_2$ and $\forall \ell$ s.t. $m \leq \ell < n$, $\rho, \ell \models \varphi_1$
  - i.e. $\varphi_2$ eventually holds, and for all positions till $\varphi_2$ holds, $\varphi_1$ holds

# Visualizing the temporal operators

- $\mathbf{X}p$ : Ne**X**t Step

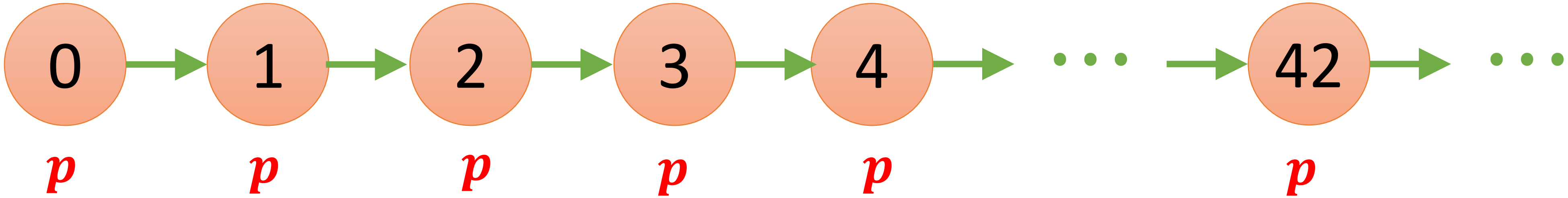



- $\mathbf{F}p$ : Some **F**uture step

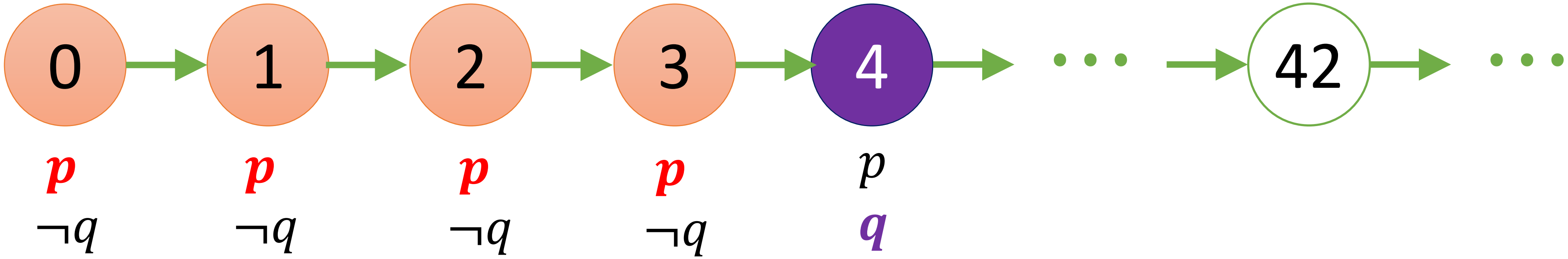


# Visualizing the temporal operators

- $\mathbf{G}p$: **Globally** $p$ holds

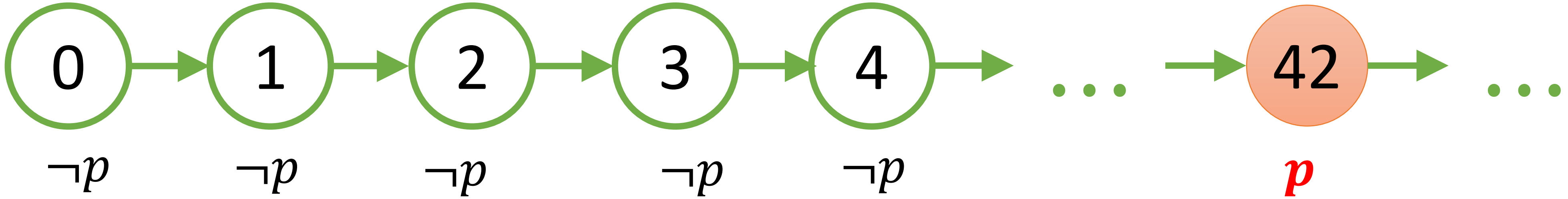



- $p\ \mathbf{U}\ q$: $p$ holds Until $q$ holds

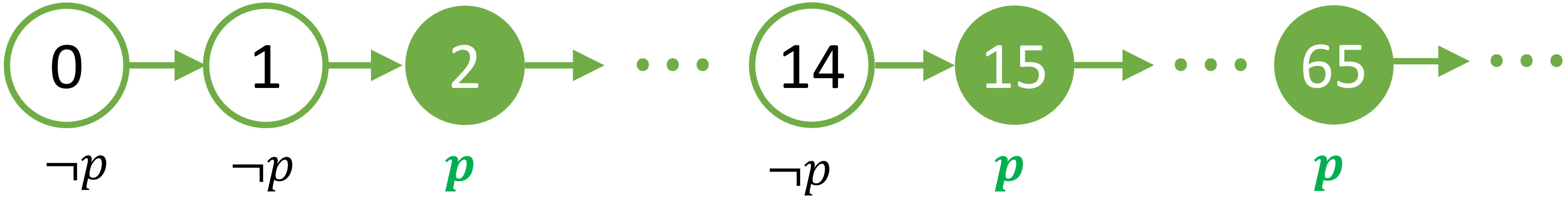


# You can nest operators!

- What does $\mathbf{XF}\ p$ mean?
  - Trace satisfies $\mathbf{XF}p$ (at time 0) if at time 1, $\mathbf{F}p$ holds. I.e. $p$ holds at some point strictly in the future

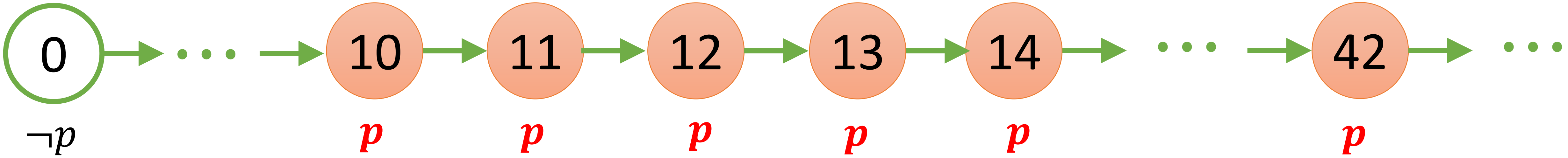



- What does $\mathbf{GF}\ p$ mean?
  - Trace satisfies $\mathbf{GF}p$ (at time 0) if at $n$, there is always a $\boldsymbol{p}$ in the future

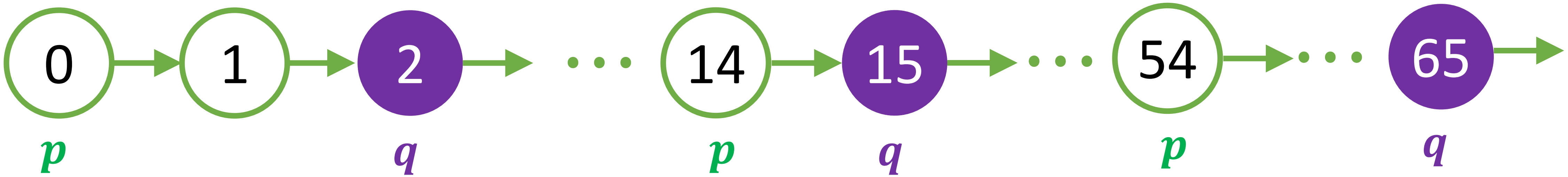


# More operator fun

- What does $\mathbf{FG}p$ mean?

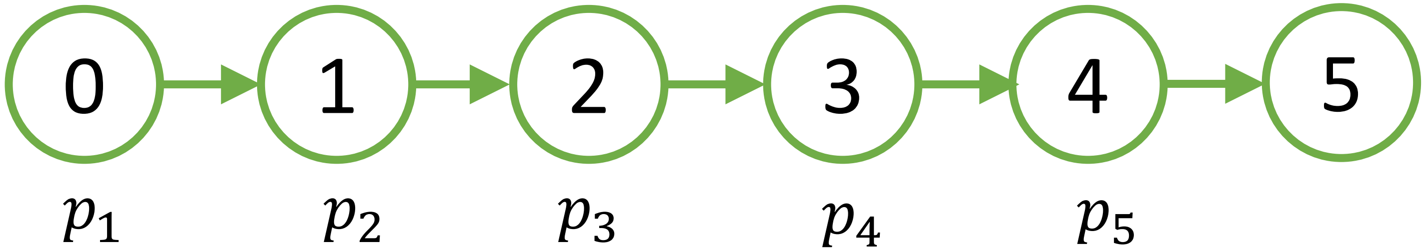



- What does $\mathbf{G}(p \Rightarrow \mathbf{F}q)$ mean?

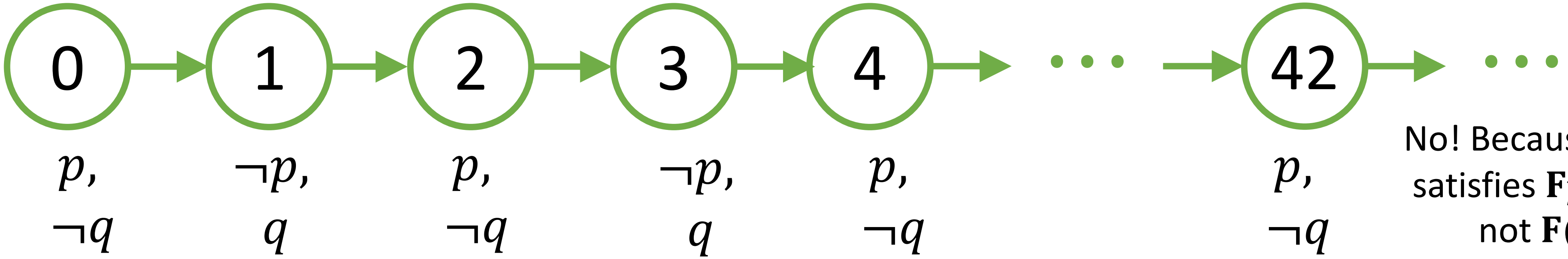


# More, more operator fun

- What does the following formula mean: $p_1 \wedge \mathbf{X}\big(p_2 \wedge \mathbf{X}(p_3 \wedge \mathbf{X}(p_4 \wedge \mathbf{X}p_5))\big)$?




- Is this true? $\mathbf{F}(p \wedge q)$ is the same as $\mathbf{F}p \wedge \mathbf{F}q$?



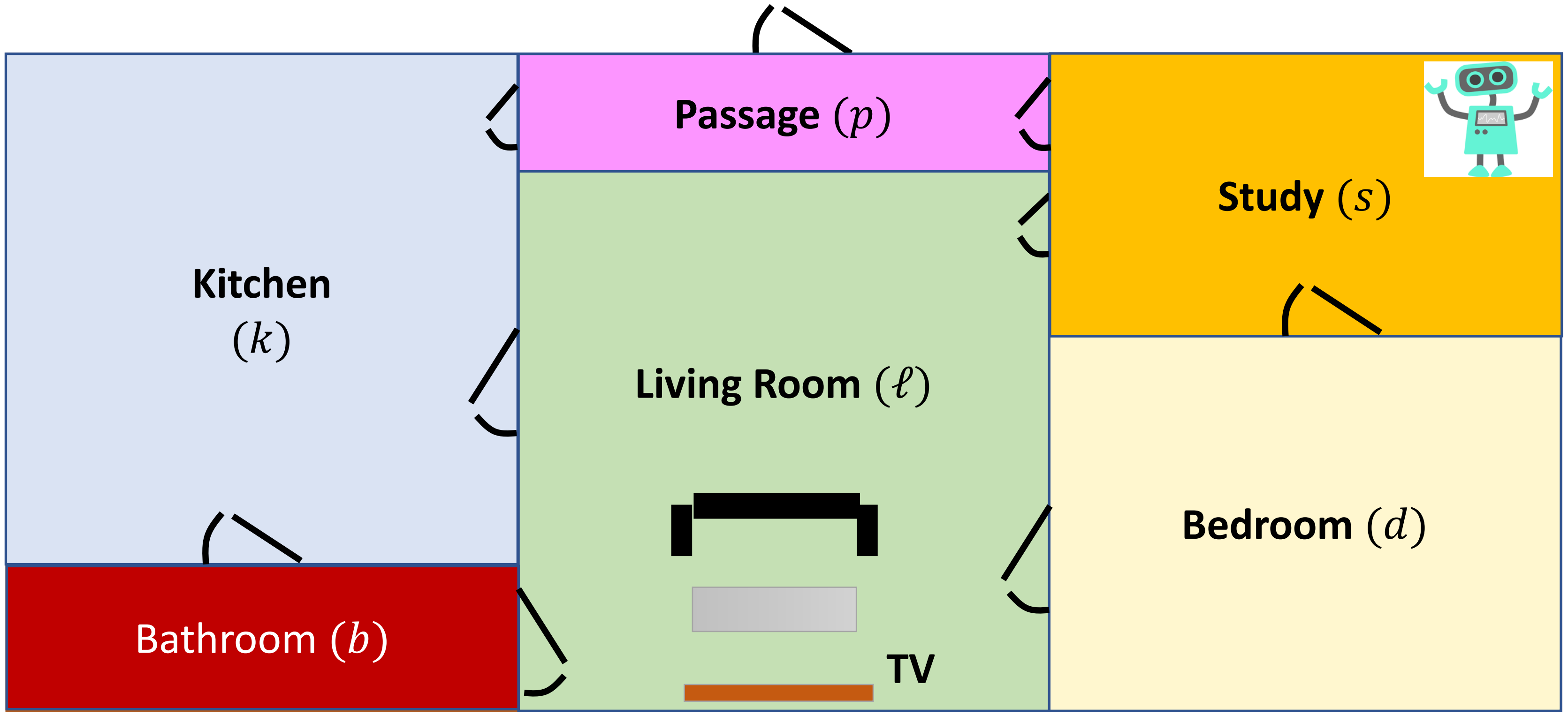

No! Because this trace satisfies $\mathbf{F}p \wedge \mathbf{F}q$, but not $\mathbf{F}(p \wedge q)$

# Operator duality and identities

- $\mathbf{F}\varphi \equiv \neg\mathbf{G}\neg\varphi$
- $\mathbf{GF}\varphi \equiv \neg\mathbf{FG}\neg\varphi$
- $\mathbf{F}(\varphi \vee \psi) \equiv \mathbf{F}\varphi \vee \mathbf{F}\psi$
- $\mathbf{G}(\varphi \wedge \psi) \equiv \mathbf{G}\varphi \wedge \mathbf{G}\psi$
- $\mathbf{FF}\varphi \equiv \mathbf{F}\varphi$
- $\mathbf{GG}\varphi \equiv \mathbf{G}\varphi$
- $\mathbf{FGF}\varphi \equiv \mathbf{GF}\varphi$
- $\mathbf{GFG}\varphi \equiv \mathbf{FG}\varphi$

# Example specifications

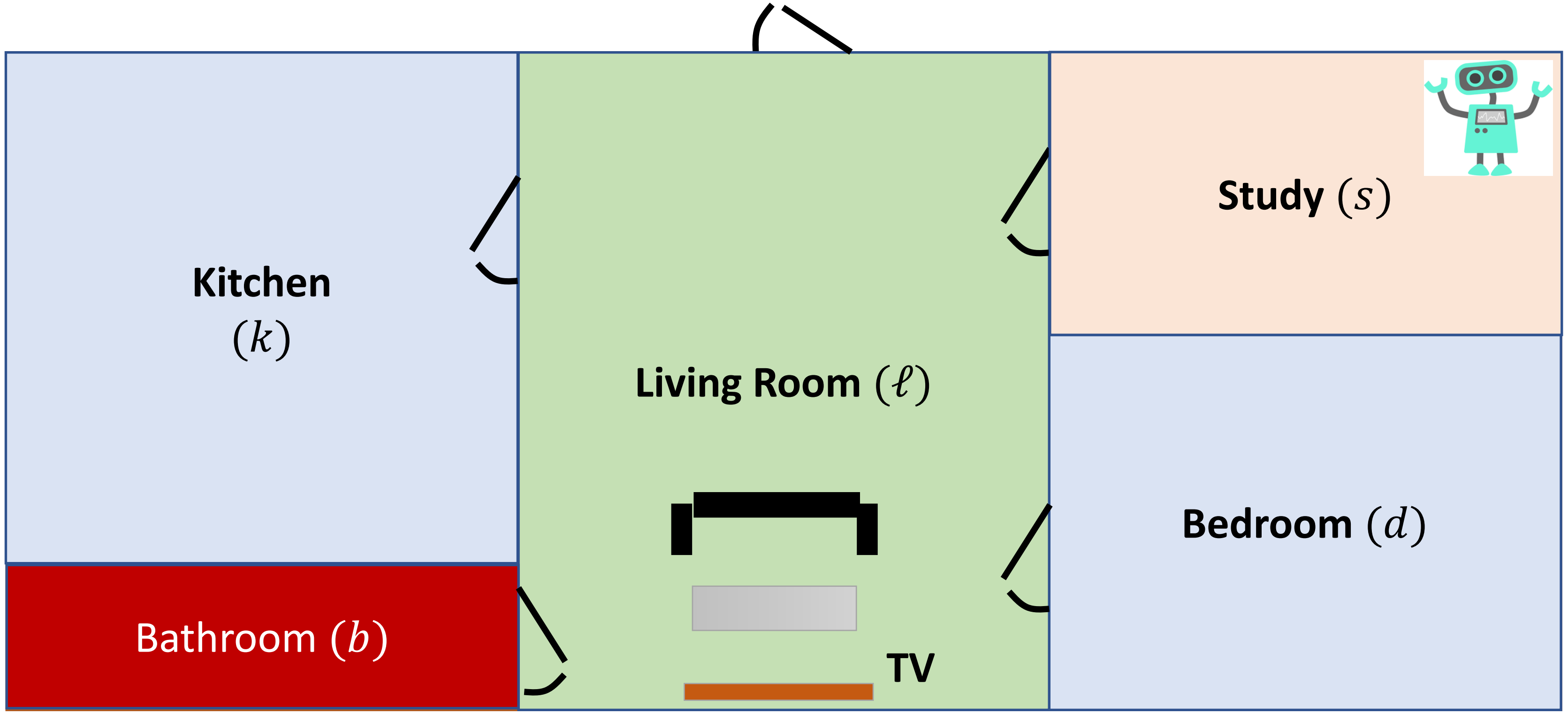
- Suppose you are designing a robot that has to do a number of missions




- Whenever the robot visits the kitchen, it should visit the bedroom after.
- Robot should never go to the bathroom
- The robot should keep working until its battery becomes low
- The robot should repeatedly visit the living room
- Whenever the TV is on and the living room has no person in it, then within three steps, the robot should turn off the TV

# Example specifications in LTL

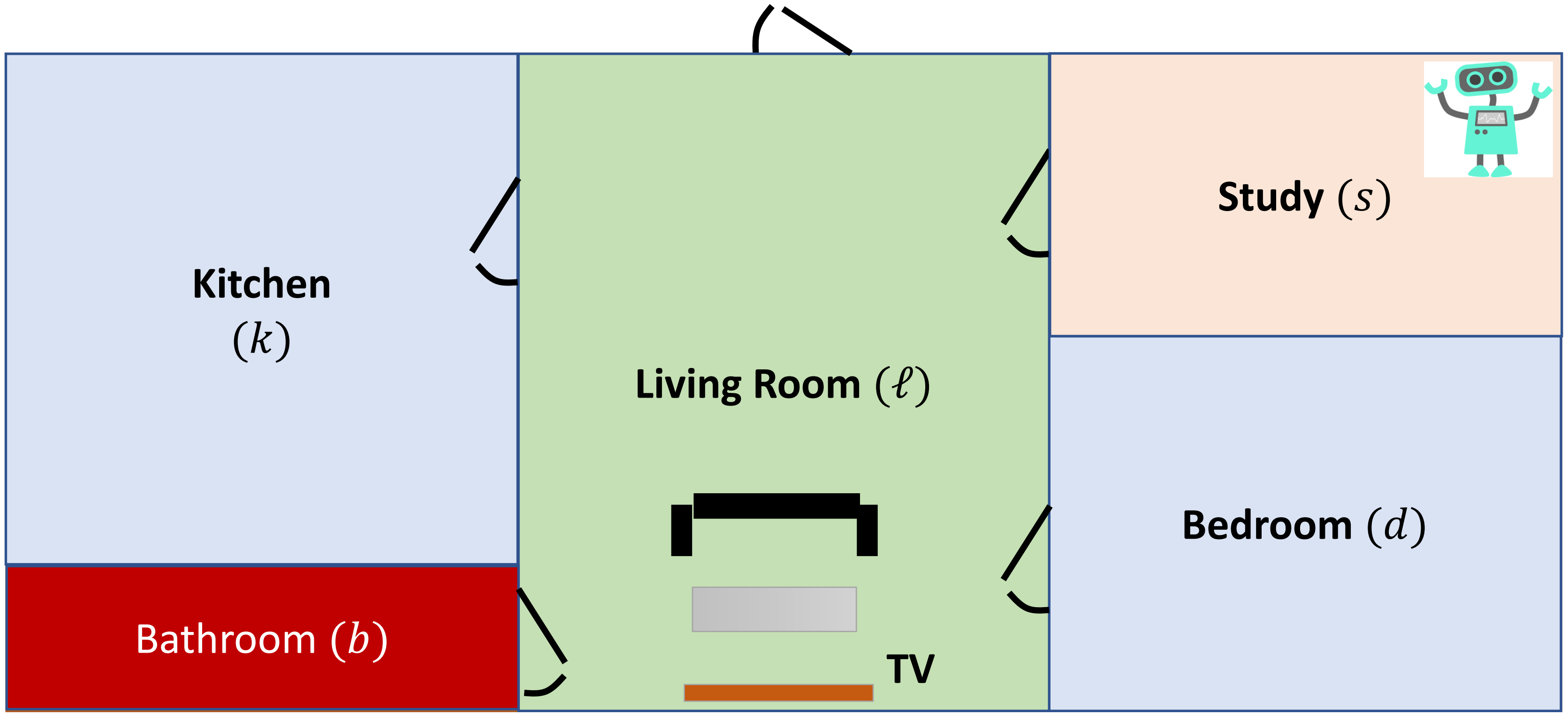
- Suppose you are designing a robot that has to do a number of missions




- Whenever the robot visits the kitchen, it should visit the bedroom after.
  $$\mathbf{G}(k_r \Rightarrow \mathbf{F}\ d_r)$$
- Robot should never go to the bathroom.
  $$\mathbf{G}\neg b_r$$
- The robot should keep working until its battery becomes low
  $$working\ \mathbf{U}\ low\_battery$$

# Example specifications in LTL

- Suppose you are designing a robot that has to do a number of missions




- The robot should repeatedly visit the living room

$$\mathbf{GF}\ \ell$$

- Whenever the TV is on and the living room has no person in it, then within three steps, the robot should turn off the TV

$o(r)$: room occupied by a person

$$\mathbf{G}\left((\neg o(\ell) \wedge TV_{on}) \Rightarrow \mathbf{F}^{\leq 3}(TV_{off})\right)$$

$$\mathbf{F}^{\leq 3}\varphi \equiv \varphi \vee \mathbf{X}\varphi \vee \mathbf{XX}\varphi \vee \mathbf{XXX}\varphi$$

# LTL is a language for expressing system requirements

```
nat x := 0; bool y:= 0
---
A:  x := x + 1
B: even(x) →
        y: = 1-y
```

**Blinker**

- So far we have seen how we can express behaviors of individual system traces using LTL
- A system $M$ starting from some initial state $q_0$ satisfies a LTL requirement $\varphi$ if ***all system behaviors*** starting in $q_0$ satisfy the requirement $\varphi$
- Denoted as $M, q_0 \vDash \varphi$
- E.g. a system is safe w.r.t. a safety requirement $\varphi$ if all behaviors satisfy $\varphi$
- Does (**Blinker,** (x↦0,y↦0)) ⊨ $\mathbf{G}$(x≥0)?

# Processes & Fairness

| nat x := 0; bool y:= 0 |
|---|
| A: x := x + 1<br>B: even(x) → <br>y: = 1-y |

**Blinker**

- Liveness property: $\mathbf{F}$ (x ≥ 10)
  - Is this property guaranteed to hold?
  - No, task A may be executed less than 10 times.
- Liveness Property: $\mathbf{F}$ y (eventually y is 1)
  - Is this property guaranteed to hold?
  - No, task B may never be selected for execution!
- But, this seems like a very unrealistic or broken scheduler!
- For infinite executions involving multiple tasks, it is important for the execution to be *fair* to each task

# Weak vs. Strong fairness

| nat x := 0; bool y:= 0 |
|---|
| A: x := x + 1<br>B: even(x) → <br>y: = 1-y |

**Blinker**

- A *fairness assumption* is a property that encodes the meaning of what it means for an infinite execution to be fair with respect to a task.
- **Weak fairness:** If a task is persistently enabled, then it is repeatedly executed.
  - I.e. if after some point the task guard is always true, then the task is infinitely often executed.
- **Strong fairness**: If a task is repeatedly enabled, then it is repeatedly executed.
  - I.e. if the task guard is infinitely often true, then the task is infinitely often executed.

# Expressing fairness assumptions in LTL: I

```
nat x :=  0; bool y :=  0
{A,B,Ø} taken :=  Ø
```
```
A:  x := x + 1; taken:= A
 B: even(x) →
        y: = 1-y; taken := B
```

**Blinker**

- Fairness assumptions can be expressed in LTL!
- Add a new variable *taken* that takes value 'A', 'B'
- Weak fairness: $(\mathbf{FG}\ guard\_i) \Rightarrow (\mathbf{GF}(\text{taken} = T_i))$
- Task A: $guard\_A$ is $true$, so this simplifies to: wf(A) := $\mathbf{GF}$(taken=A)
- Task B: wf(B) := $\mathbf{FG}$ (even(x)) ⇒ $\mathbf{GF}$ (taken=B)
- Does (wf(A)∧ wf(B)) ⇒ $\mathbf{F}$ (x ≥ 10)?
  - **Yes!**
- Does (wf(A)∧ wf(B)) ⇒ $\mathbf{F}$ y?
  - **No!**

# Expressing fairness assumptions in LTL: II

```
nat x :=  0; bool y :=  0
{A,B,∅} taken :=  ∅

A:  x := x + 1; taken:= A
 B: even(x) →
         y: = 1-y; taken := B
```

**Blinker**

- Strong fairness: $(\mathbf{GF}\ guard\_i) \Rightarrow (\mathbf{GF}(\text{taken} = T_i))$
- Task A: $guard\_A$ is $true$, so this simplifies to: sf(A) ≔ $\mathbf{GF}$(taken=A)
- Task B: sf(B) ≔ $\mathbf{GF}$ (even(x)) ⇒ $\mathbf{GF}$ (taken=B)
- Does (sf(A)∧ sf(B)) ⇒ $\mathbf{F}$ $(x \geq 0)$?
  - **Yes!**
- Does (sf(A)∧ sf(B)) ⇒ $\mathbf{F}$ y?
  - **Yes!**

If a process satisfies a liveness requirement under strong fairness, it satisfies it under weak fairness: strong fairness is a **stronger formula** than weak fairness

# Types of Specifications/Requirements

- Hard Requirements: Violation leads to endangering safety-criticality or mission-criticality
  - Safety Requirements: system never does something bad
  - Liveness Requirements: from any point of time, system eventually does something good
- Soft Requirements: Violations lead to inefficiency, but are not critical
  - (Absolute) Performance Requirements: system performance is not worst than a certain level
  - (Average) Performance Requirements: average system performance is at a certain level

# Other kind of requirements

- Security Requirements: system should protect against modifications in its behavior by an adversarial actor
  - Failure to satisfy security requirements may lead to a hard requirement violation
- Privacy Requirements: the data revealed by the system to the external world should not leak sensitive information
- These requirements will become increasingly important for autonomous CPS, especially as IoT technologies and smart transportation initiatives are deployed!

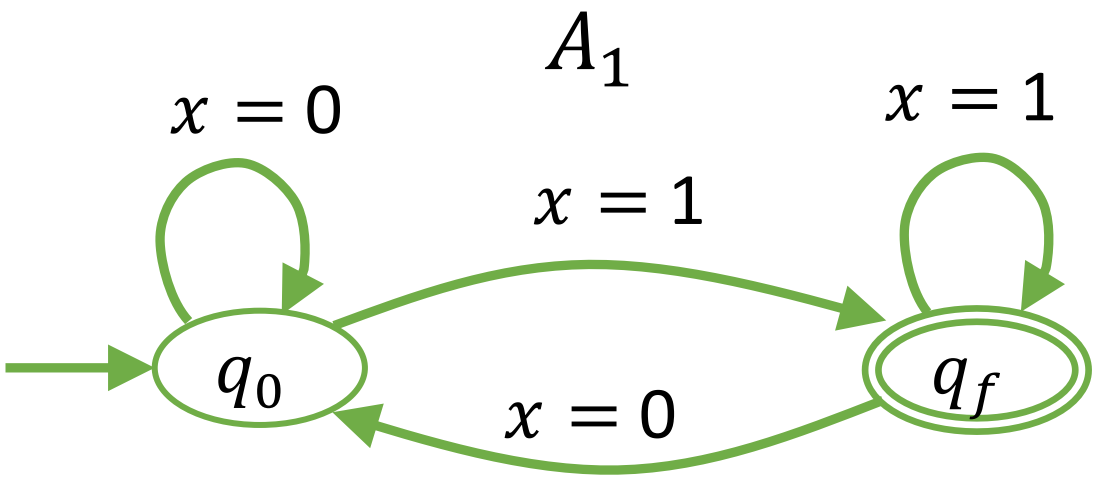
# Büchi Automata

# Monitors

- A safety monitor classifies system behaviors into good and bad
- Safety verification can be done using inductive invariants or analyzing reachable state space of the system
  - A bug is an execution that drives the monitor into an error state
- Can we use a monitor to classify infinite behaviors into good or bad?
- Yes, using theoretical model of Büchi automata proposed by J. Richard Büchi in 1960

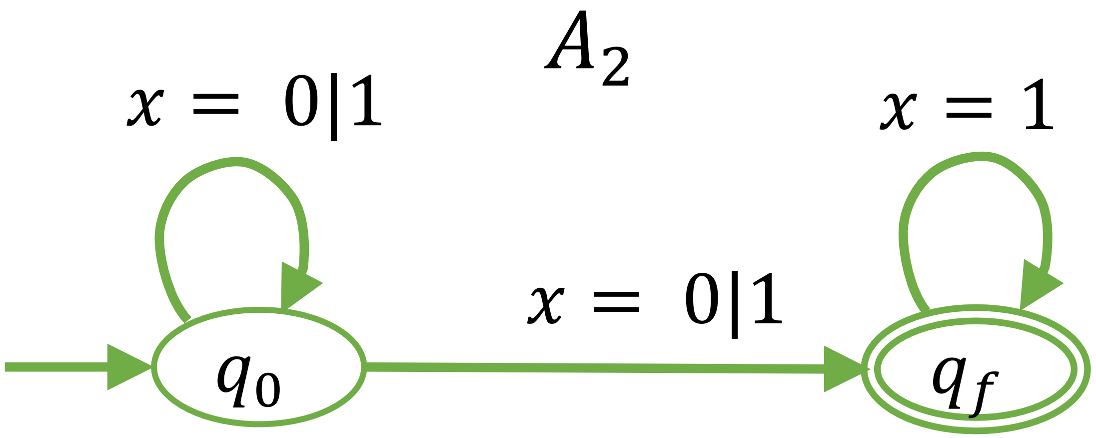
# Büchi automaton Example 1

- Extension of finite state automata to accept infinite strings




- States $Q$: $\{q_0, q_f\}$
- Input variable $x$ with domain $\Sigma$: $\{0,1\}$
- Final state: $\{q_f\}$
- Transitions: (as shown)
- Given trace $\rho$ (infinite sequence of symbols from $\Sigma$), $\rho$ is accepted by $A_1$, if $q_f$ appears inf. often
- What is the language of $A_1$?
  - LTL formula $\mathbf{GF}(x = 1)$

# Büchi automaton Example 2

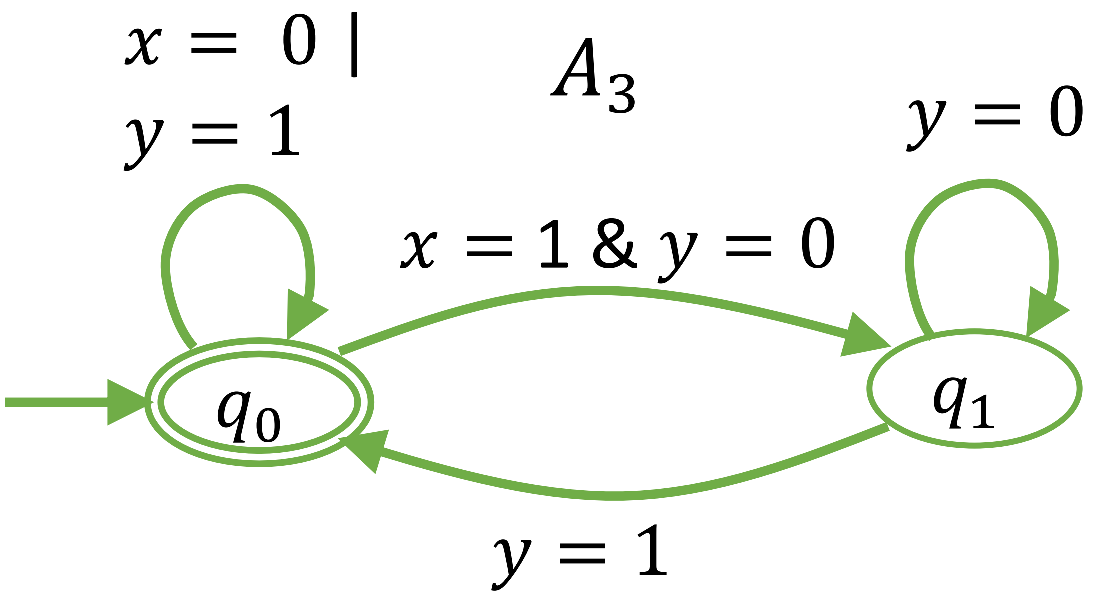



- $Q: \{q_0, q_f\}$, $\Sigma: \{0,1\}$, $F: \{q_f\}$
- Transitions: (as shown)

Fun fact: there is no deterministic Büchi automaton that accepts this language

- Note that this is a nondeterministic Büchi automaton
- $A_2$ accepts $\rho$ if ***there exists a path*** along which a state in $F$ appears infinitely often
- What is the language of $A_2$?
  - LTL formula $\mathbf{FG}(x = 1)$

# Büchi automaton Example 3




- $Q$: $\{q_0, q_1\}$, $\Sigma$: $\{0,1\}$, $F$: $\{q_f\}$
- Transitions: (as shown)

- What is the language of $A_3$?
  - LTL formula:
    $\mathbf{G}\big((x = 1) \Rightarrow \mathbf{F}(y = 1)\big)$
  - I.e. always when $(x = 1)$, in some future step, $(y = 1)$
  - In other words, $(x = 1)$ must be followed by $(y = 1)$

# Using Büchi monitors

- Theoretical result: Every LTL formula $\varphi$ can be converted to a Büchi monitor/automaton $A_\varphi$
- Size of $A_\varphi$ is generally exponential in the size of $\varphi$; blow-up unavoidable in general
- Construct composition of the original process $P$ and the Büchi monitor $A_\varphi$
- If there are cycles in the composite process that do not visit the states specified by the liveness property, then we have found a violation.
- Reachable cycles in process composition correspond to counterexamples to liveness properties
- Implemented in many verification tools (e.g. the SPIN model checker developed at NASA JPL)

CTL

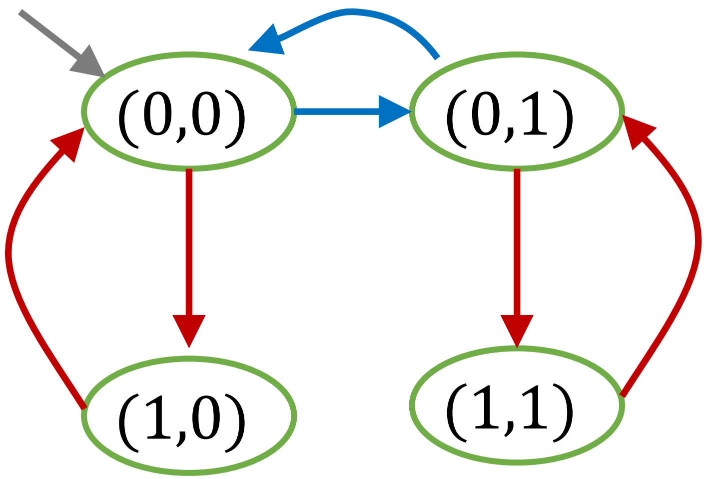
# Computation Tree Logic

- LTL was a linear-time logic where we reason about traces
- CTL is a logic where we reason over the tree of executions generated by a program, also known as the *computation tree*
- We care about CTL because:
  - There are some properties that cannot be expressed in LTL, but can be expressed in CTL: From every system state, there is a system execution that takes it back to the initial state (also known as the reset property)
  - To understand pCTL (Probabilistic CTL), it's good if you understand CTL ☺
  - Can express interesting properties for multi-agent systems

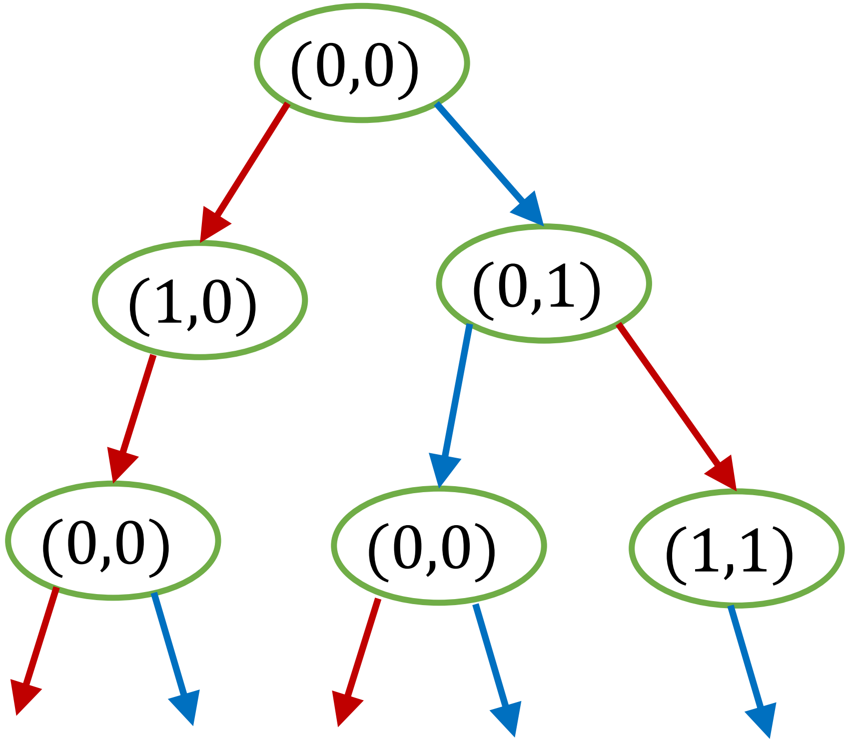
# Computation Tree

| nat x := 0; bool y:= 0 |
|---|
| A: x := (x + 1) mod 2<br>B: even(x) → y: = 1-y |

**Process**




**Finite State machine**




- We saw computation trees when understanding semantics of asynchronous processes
- Basically a tree that considers "all possibilities" in a reactive program

# CTL Syntax

| Syntax of CTL | | |
|---|---|---|
| $\varphi ::=$ | $p \mid \neg\varphi \mid \varphi \wedge \varphi$ \| | Prop. in $AP$, negation, conjunction |
| | $\mathbf{EX}\varphi$ \| | **E**xists Ne**X**t Step |
| | $\mathbf{EF}\varphi$ \| | **E**xists a **F**uture Step |
| | $\mathbf{EG}\varphi$ \| | **E**xists an execution where **G**lobally in all steps |
| | $\mathbf{E}\ \varphi\ \mathbf{U}\ \varphi$ \| | **E**xists an execution where in all steps **U**ntil in some step |
| | $\mathbf{AX}\varphi$ \| | In **A**ll Ne**X**t Steps |
| | $\mathbf{AF}\varphi$ \| | In **A**ll possible future paths, there is a future step |
| | $\mathbf{AG}\varphi$ \| | In **A**ll possible future paths, **G**lobally in all steps |
| | $\mathbf{A}\ \varphi\ \mathbf{U}\ \varphi$ \| | In **A**ll possible future executions, in all steps **U**ntil in some step |

# CTL semantics

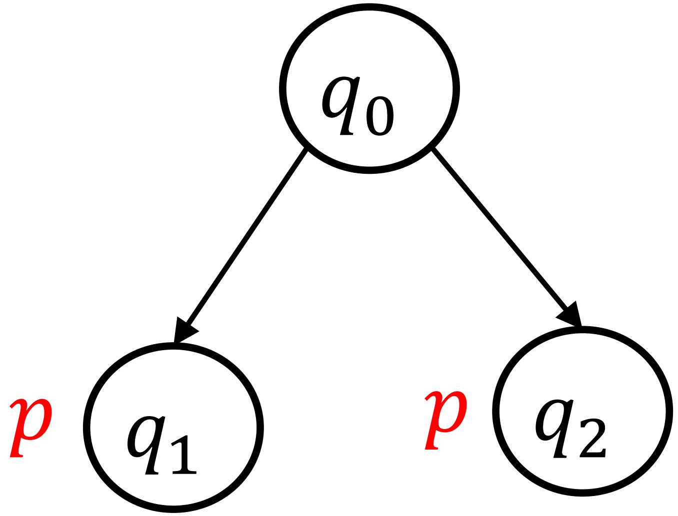
- *Path properties:* properties of any given path or execution in the program
- *Quantification over runs:* Checking if a property holds over **all** paths or over **some** path
- Example CTL operator:

$$\mathbf{A}\ \mathbf{F}\ \ p$$

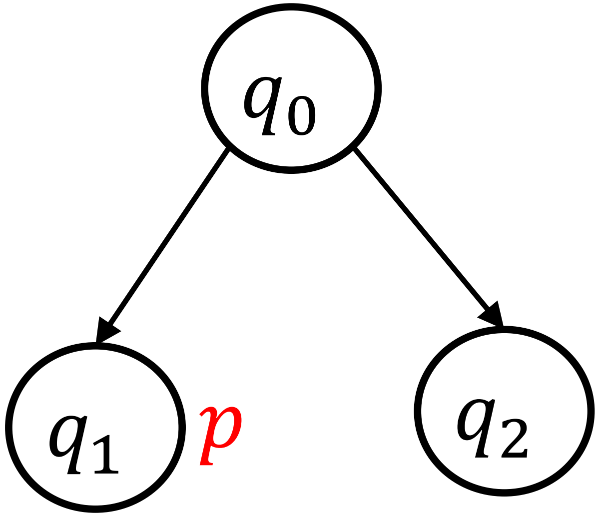
For **A**ll executions

Eventually/In Some **F**uture step

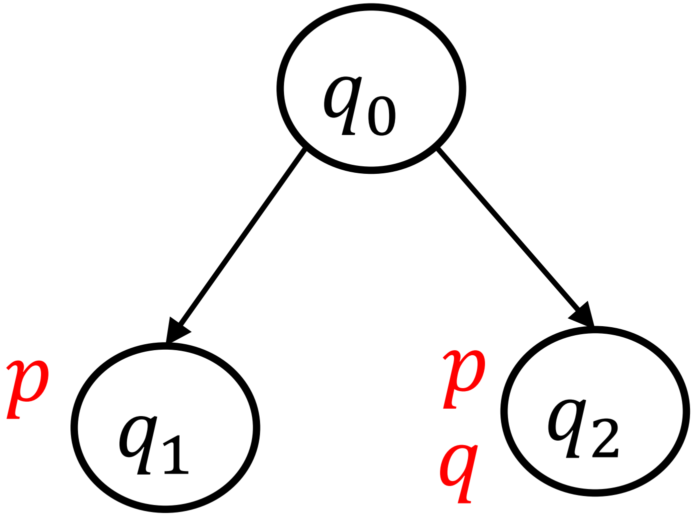
# CTL Semantics through examples




$\mathbf{AX}\, p$

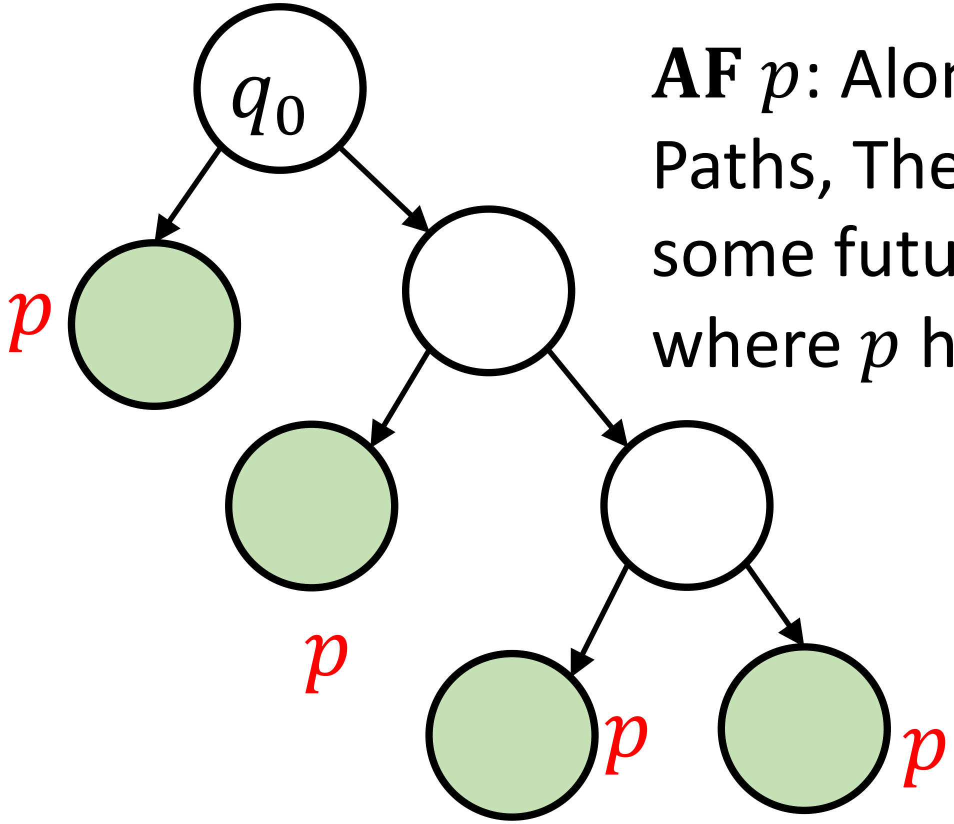



$\mathbf{EX}\, p$

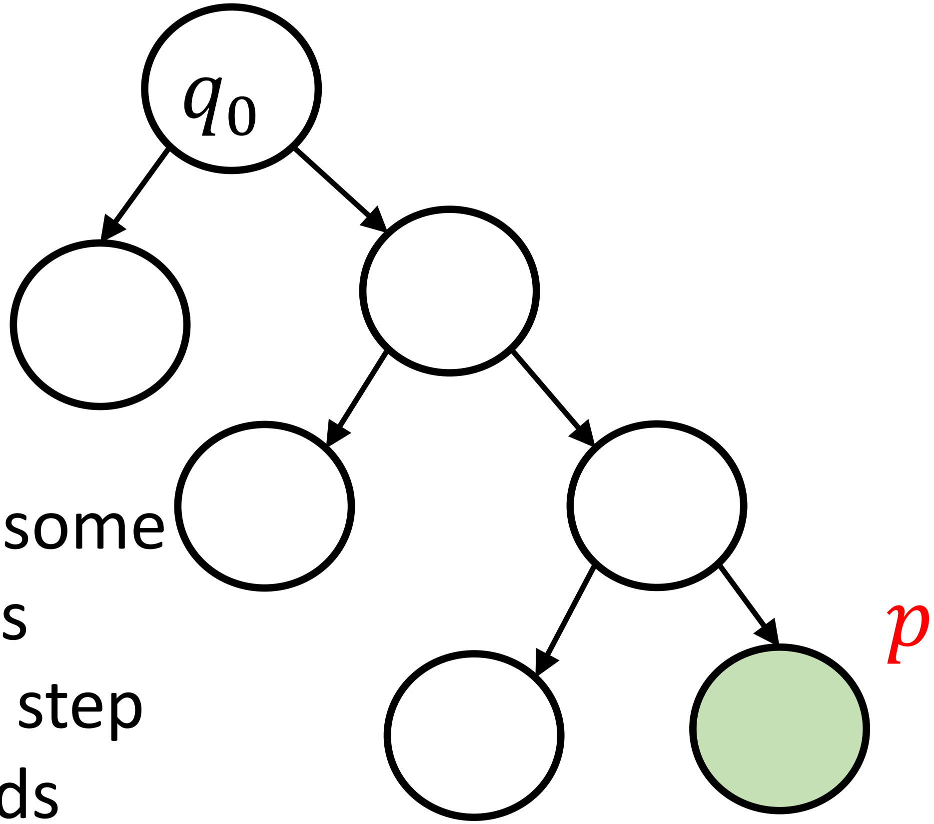



$\mathbf{AX}\, p \wedge \mathbf{EX}\, q$

# CTL semantics through examples

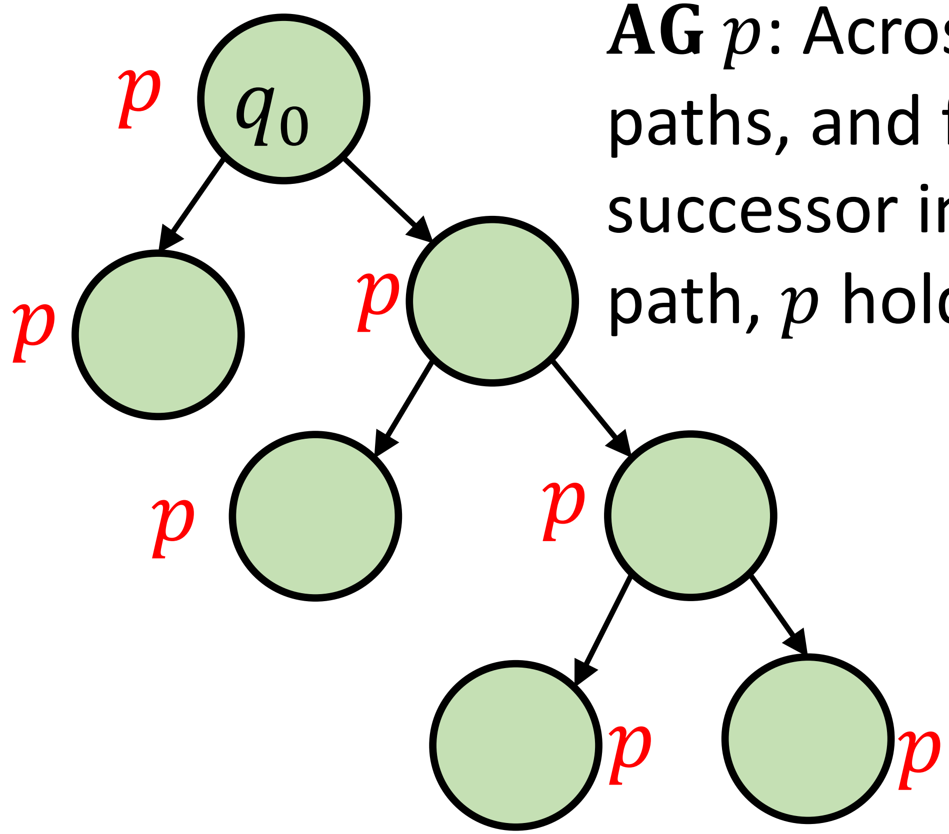



**AF** $p$: Along all Paths, There is some future step where $p$ holds

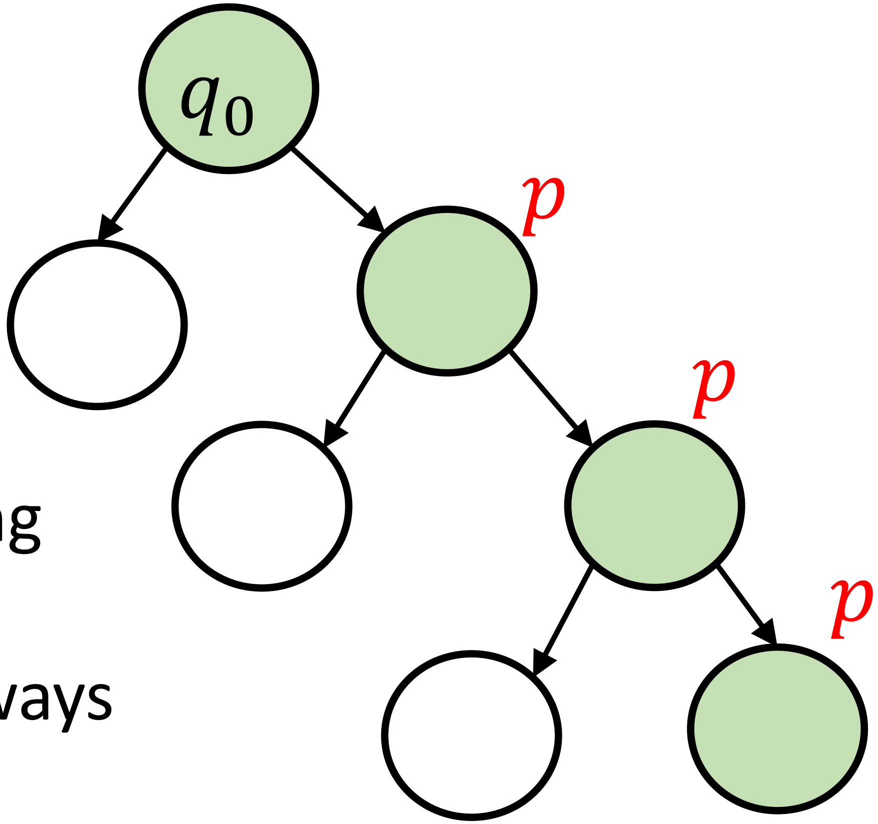



**EF** $p$: Along some path, there is some future step where $p$ holds

# CTL semantics through examples




**AG** $p$: Across all paths, and for every successor in the path, $p$ holds




**EG** $p$: Along some path, $p$ always holds

# CTL Operator fun

- $\mathbf{AGEF}\ p$
- $\mathbf{AGAF}\ p$
- $\mathbf{EGAF}\ p$
- $\mathbf{AG}\ (p \Rightarrow \mathbf{EX}\ q)$

# CTL advantages and limitations

- Checking if a given state machine (program) satisfies a CTL formula can be done quite efficiently (linear in the size of the machine and the property)
- Native CTL cannot express fairness properties
  - Extension Fair CTL can express fairness
- $\text{CTL}^*$ is a logic that combines CTL and LTL: You can have formulas like $\mathbf{AGF}\, p$
- CTL: Less used than LTL, but an important logic in the history of temporal logic

# PCTL

# Probabilistic CTL

- LTL
  - Can be interpreted over individual executions
  - Can be interpreted over a state machine: do all paths satisfy property
- CTL
  - Is interpreted over a computation tree
- PCTL
  - Is interpreted over a discrete-time Markov chain
  - Encodes uncertainties in computation due to environment etc.

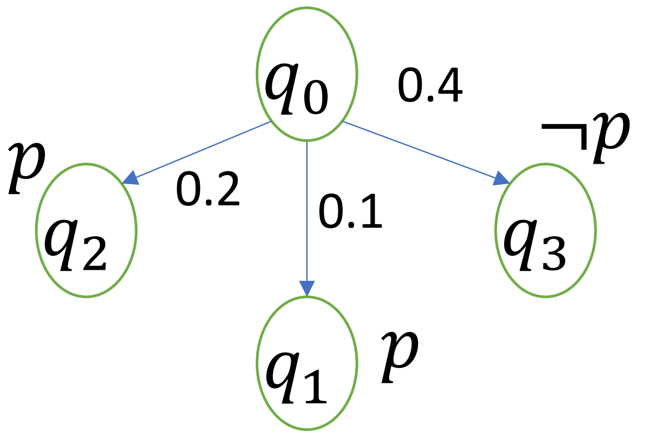
# Probabilistic CTL

| Syntax of PCTL | | | |
|---|---|---|---|
| $\varphi$ ::= | $p \mid \neg\varphi \mid \varphi \wedge \varphi$ | \| | Prop. in $AP$, negation, conjunction |
| (State) | $P_{\sim\lambda}(\psi)$ | \| | $\sim \in \{<, \leq, >, \geq\}$, $\lambda \in [0,1]$ : Probability of $\psi$ being true |
| $\psi$ ::= | $\mathbf{X}\varphi$ | \| | NeXt Time |
| (Path) | $\varphi\ \mathbf{U}^{\leq k}\varphi$ | \| | Bounded **U**ntil (upto $k$ steps) |
| | $\varphi\ \mathbf{U}\ \varphi$ | | **U**ntil (Recall $\mathbf{F}\varphi = \mathit{true}\ \mathbf{U}\ \varphi$, and $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$ |

PCTL formulas are state formulas, path formulas used to define how to build a PCTL formula

# Semantics

- Semantics of path formulas is straightforward (similar to LTL/CTL)
- Semantics of state formula with Probabilistic operator:
  - $Prob(q, \mathbf{X}\varphi)$: $\sum_{q' \models \varphi} P(q, q')$
  - Does $P_{\geq 0.5}(\mathbf{X}\, p)$ hold in state $q_0$?
  - No, because $P(q_0, \mathbf{X}\, p) = 0.1 + 0.2 = 0.3$



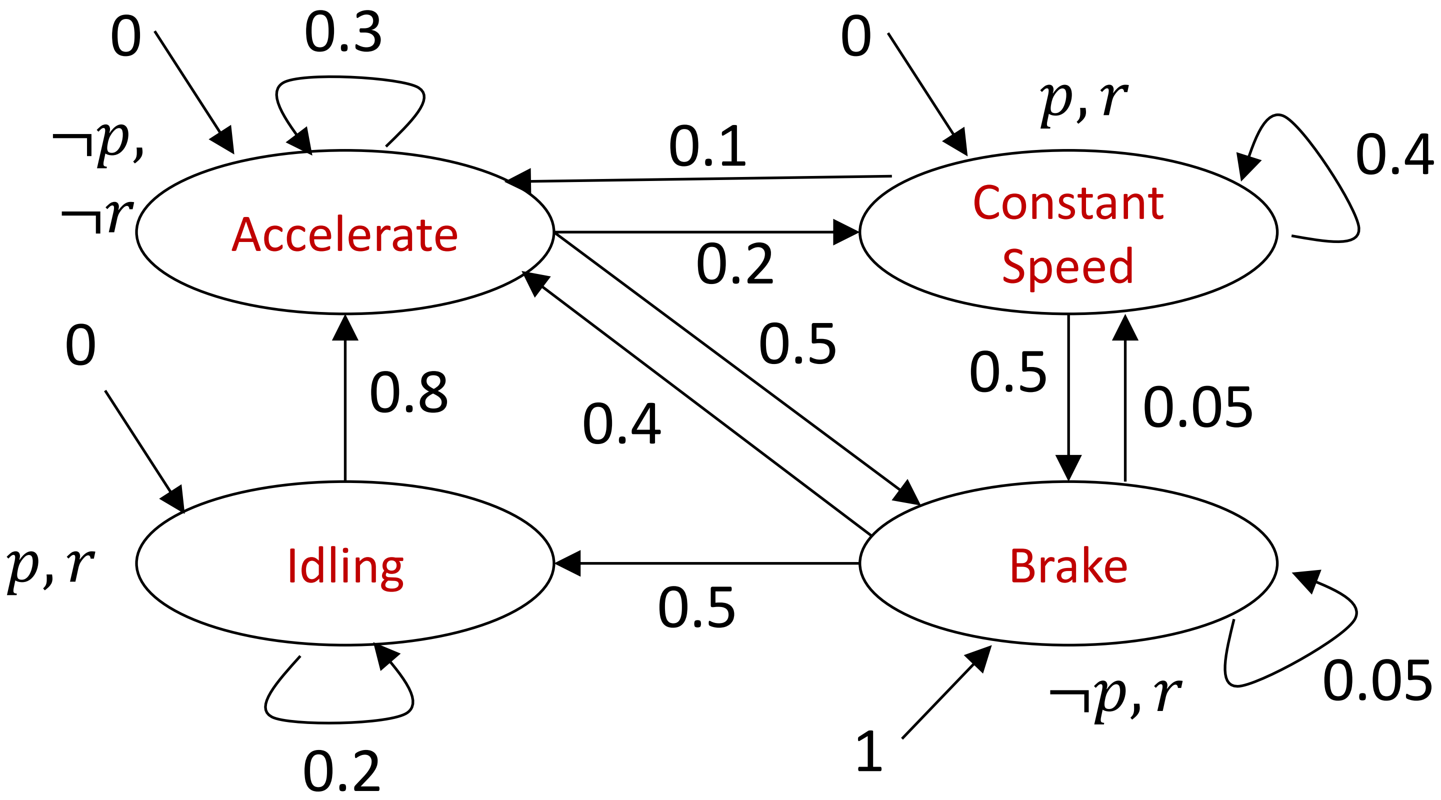

- Semantics of state formula with Until $Prob\left(q, \alpha \mathbf{U}^{\leq \boldsymbol{k}} \beta\right)$:
  - 1 if $q \models \beta$
  - 0 if $q \not\models$ α or $q \not\models \beta$ and $k = 0$
  - $\sum P(q, q'). Prob(q', \alpha\, U^{k-1} \beta)$ for $k > 0$, otherwise

# PCTL

- Does this formula $P_{\geq 0.5}(\mathbf{X}p)$ hold in state Brake?
  - Yes
- Value of $\epsilon$? $P_{\geq \epsilon}(\mathbf{F}^{\leq 2} r)$ in state Accel
  - Compute $Prob(q, \mathbf{F}^{\leq 2} r)$ for all $q$, pick smallest
  - $P(A,B) + P(A,C) + P(A,A,B) + P(A,A,C)$
    = 0.5 + 0.2 + 0.3*0.5 + 0.3*0.2
    = 0.91
  - $\epsilon = 0.91$
- I.e. with probability ≥ 0.91, driver checks cell phone within 2 steps

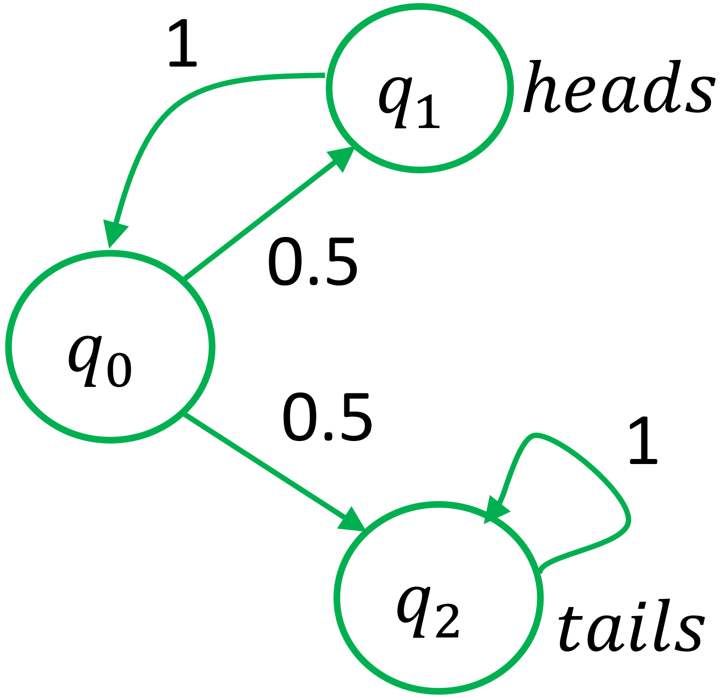
$r$: Checking cellphone

$p$: Feeling sleepy



# Quantitative in PCTL vs. Qualitative in CTL

- Toss a coin repeatedly until "tails" is thrown
- Is "tails" eventually thrown along all paths?
  - CTL: AF tails
  - Result: false
  - Why? $q_0 q_1 q_0 q_1$ …
- Is the probability of eventually thrown "tails" equal to 1?
  - PCTL: $P_{\geq 1}(\, \mathbf{F}\; tails\, )$
  - Result*: true*
  - Probability of path $q_0 q_1 q_0 q_1$ … is zero!



# Summary

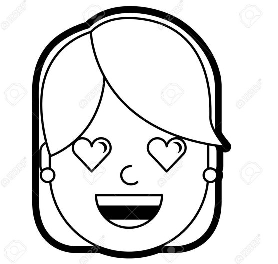
- LTL
- Büchi automata
- CTL
- PCTL

$\Box_{[1,3]}(x > 0) \wedge \Diamond_{[0,0.001]}(y < 0) \Rightarrow (x > 1) \vee (x < -1)$

$p_1\, \mathcal{U}_{(a_1,b_1)}\, (p_2\, \mathcal{U}_{(a_2,b_2)}\, (p_3\, \mathcal{U}_{(a_3,b_3)}\, (p_4\, \mathcal{U}_{(a_4,b_4)}\, \mathcal{G}\, p_5)))$

$\Box_{[1,3]}(x > 0) \wedge \Diamond_{[0,0.001]}(y < 0) \Rightarrow (x > 1) \vee (x < -1)$

$\Box_{[1,3]}(x > 0) \vee (x < -1)$



$\Box_{[1,3]}(x > 0) \Rightarrow \Diamond_{[1,3]}((y > 0) \wedge \Diamond_{[0,0.001]}(y < 0) \Rightarrow (x > 1) \vee (x < -1)$