Cyber-Physical Systems

Laura Nenzi

Università degli Studi di Trieste
II Semestre 2019

Lecture 8: Temporal Logic

[Many Slides due to J. Deshmukh, USC, LA,USA]

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 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

▶ Regular Expressions (RE) are sequences of characters that specify acceptable pattern 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 (FSA)

Famous equivalence between FSA and regular expressions:

▶ For every regular expression R_i , there is a corresponding FSA A_i that accepts the set of strings generated by R_i .

▶ For every FSA A_i there is a corresponding regular expression that generates the set of strings accepted by A_i .


A_1

q_0 →(a-z)→ q_1 →(a-z,0-9)→ q_f

State

Accepting state

[a-z][a-z 0-9]
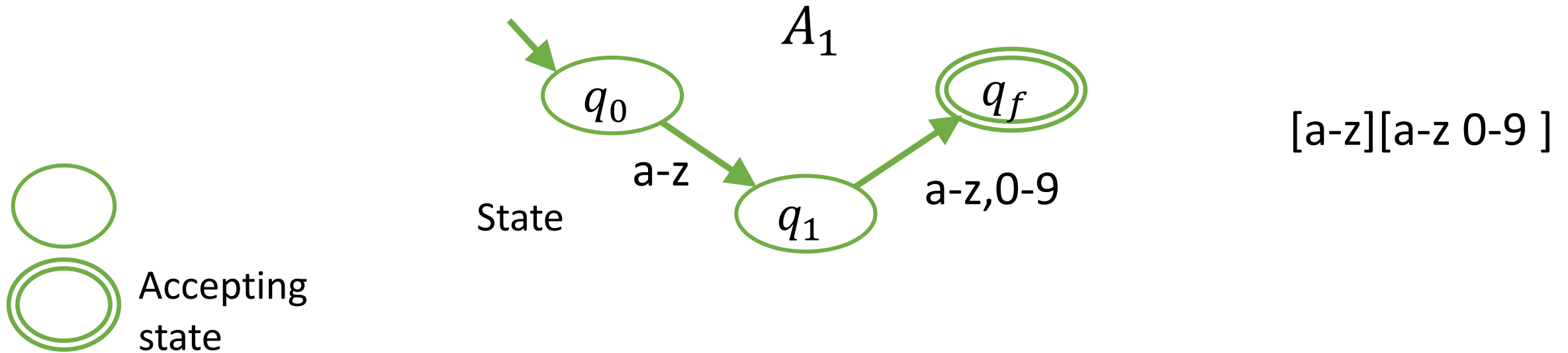
How does a Finite State Automaton work?

A_2

[a-z][0-9]*[a-z]
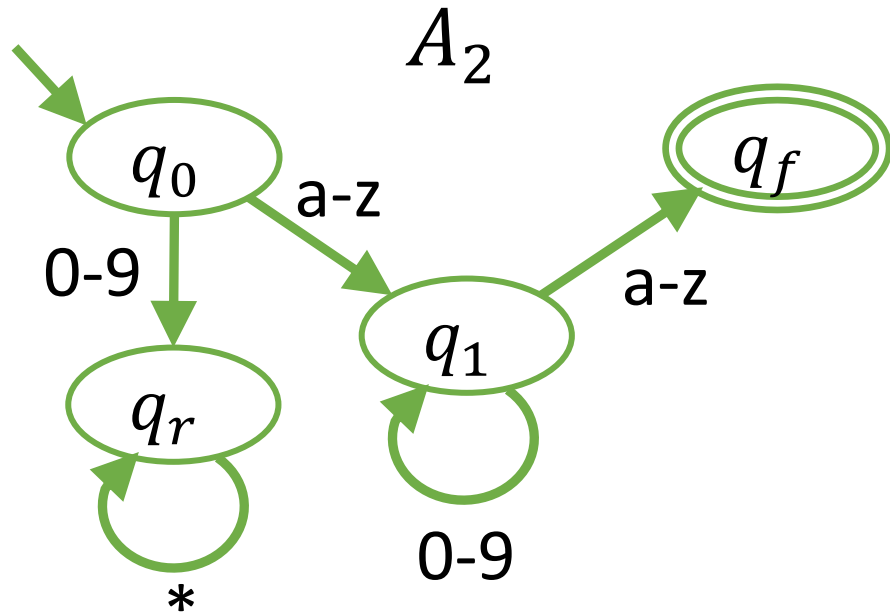
- ▶ Starts at the initial state q_0

- ▶ 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 automaton from q_0 to q_f is **accepted** by the automaton

Language of a finite state automaton
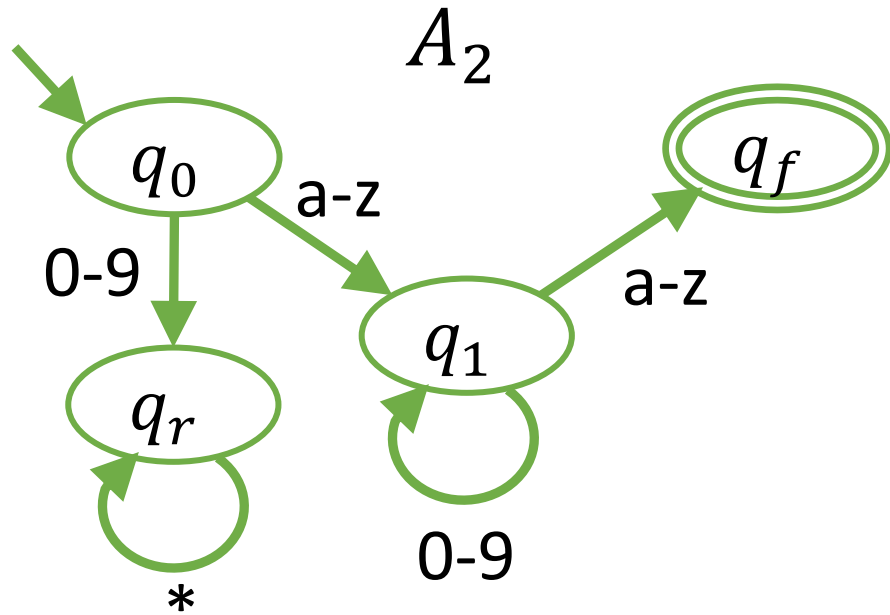
- What strings are accepted by A_2?
  - ab, zy, s2r, q123s, u3123123v, etc.
- What strings are not accepted by A_2?
  - 2b, 334a, etc.
- The set of all strings accepted by A_2 is called its **language**
- The language of a finite state automaton consists of strings, each of which can be arbitrarily long, **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 operators 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:
  $$AP = \{p, q, r, \dots\}$$

  ▶ Boolean connectives:
  $$\wedge, \vee, \neg, \Rightarrow, \equiv$$

▶ Syntax recursively gives how new formulae are constructed from smaller formulae

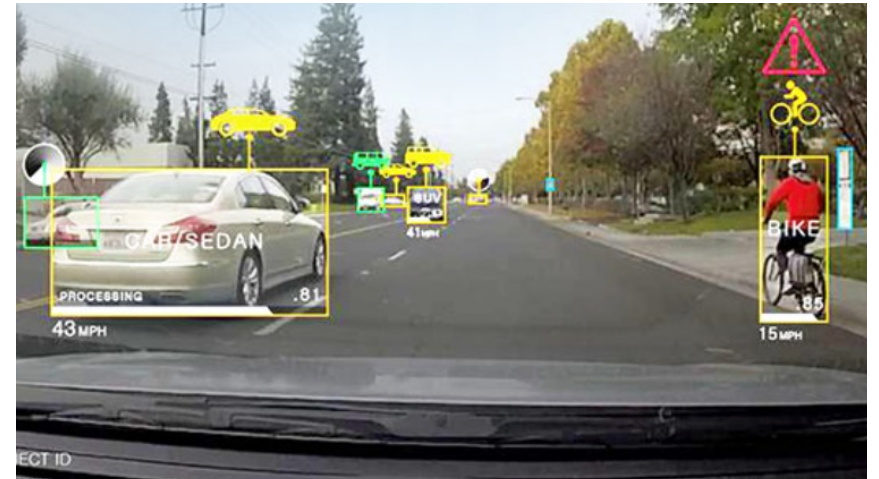| Syntax of Propositional Logic		
$\varphi ::=$	$true$ $\vert$	the true formula
	p $\vert$	p is a prop in AP
	$\neg \varphi$ $\vert$	Negation
	$\varphi \wedge \varphi$ $\vert$	Conjunction
	$\varphi \vee \varphi$ $\vert$	Disjunction
	$\varphi \Rightarrow \varphi$ $\vert$	Implication
	$\varphi \equiv \varphi$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

Semantics

▶ Semantics (i.e. meaning) of a formula can be defined recursively

▶ Semantics of an atomic proposition defined by a **valuation** function ν

▶ 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
$\nu(p)$	1 if $\nu(p) = 1$
$\nu(\neg\varphi)$	1 if $\nu(\varphi) = 0$ 0 if $\nu(\varphi) = 1$
$\nu(\varphi_1 \wedge \varphi_2)$	1 if $\nu(\varphi_1) = 1$ and $\nu(\varphi_2) = 1$, 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

▶ r: the bicycle has a rider

▶ $p \Rightarrow r$: If there is an upright bicycle in the middle of the road, the bicycle has a rider

▶ q : There is car in the field of vision

▶ 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 $v\big((p_1 \wedge p_2) \Rightarrow p_3\big)$?
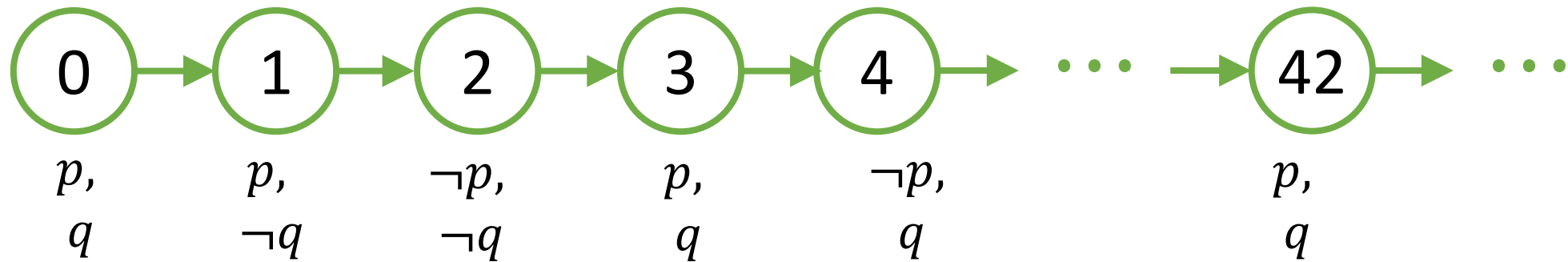
▶ $v\big((p_1 \wedge p_2) \Rightarrow p_3\big)$

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

▶ $v\big((p_1 \Rightarrow p_3) \wedge (p_2 \Rightarrow p_3)\big)$

▶ Is this true? $v\Big(\underline{(p_1 \wedge p_2) \Rightarrow p_3} \equiv \underline{(p_1 \Rightarrow p_3) \wedge (p_2 \Rightarrow p_3)}\Big) = 1$?
(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

▶ ρ:

$$
\underset{\substack{p,\\q}}{0} \rightarrow \underset{\substack{p,\\\neg q}}{1} \rightarrow \underset{\substack{\neg p,\\\neg q}}{2} \rightarrow \underset{\substack{p,\\q}}{3} \rightarrow \underset{\substack{\neg p,\\q}}{4} \rightarrow \cdots \rightarrow \underset{\substack{p,\\q}}{42} \rightarrow \cdots
$$

▶ 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
	$\mathbf{X}\varphi$	|	Ne**X**t Step
	$\mathbf{F}\varphi$	|	Some **F**uture Step
	$\mathbf{G}\varphi$	|	**G**lobally in all steps
	$\varphi \mathbf{U} \varphi$	|	In all steps **U**ntil 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 $\models$ (read models) to show that a trace-point satisfies a formula

▶ $\rho, n \models \varphi$: Read as trace ρ at time n satisfies formula φ

▶ If we omit n, then the meaning is time 0. I.e. $\rho \models \varphi$ is the same as $\rho, 0 \models \varphi$

▶ 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 φ 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 φ_1 and φ_2 ***both hold*** starting time n

Recursive semantics of LTL: II

- $\rho, n \vDash \mathbf{X}\varphi$ if $\rho, n+1 \vDash \varphi$
  - i.e. if φ holds starting at the next time point
- $\rho, n \vDash \mathbf{F}\,\varphi$ if $\exists m \geq n$ such that $\rho, m \vDash \varphi$
  - i.e. φ is true starting now, or there is some future time-point m from where φ is true
- $\rho, n \vDash \mathbf{G}\,\varphi$ if $\forall m \geq n : \rho, m \vDash \varphi$
  - i.e. φ is true starting now, and for all future time-points m, φ is true starting at m
- $\rho, n \vDash \varphi_1 \mathbf{U} \varphi_2$ if $\exists m \geq n$ s.t. $\rho, m \vDash \varphi_2$ and $\forall \ell$ s.t. $m \leq \ell < n$, $\rho, \ell \vDash \varphi_1$
  - i.e. φ_2 eventually holds, and for all positions till φ_2 holds, φ_1 holds

Visualizing the temporal operators

▶ **X**p : Ne**X**t Step
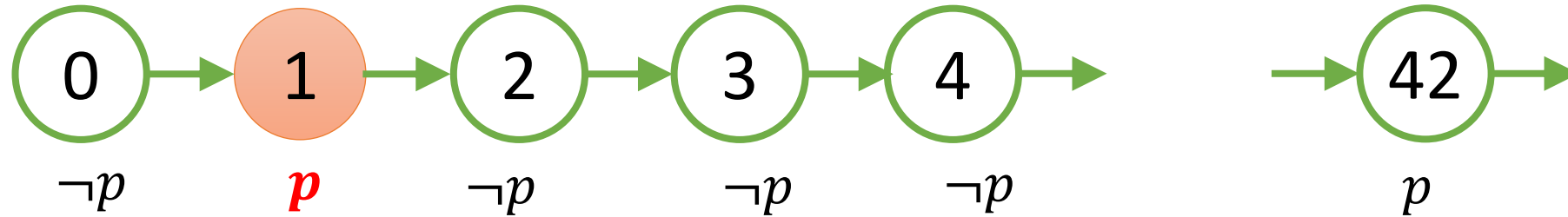
▶ **F**p : Some **F**uture step
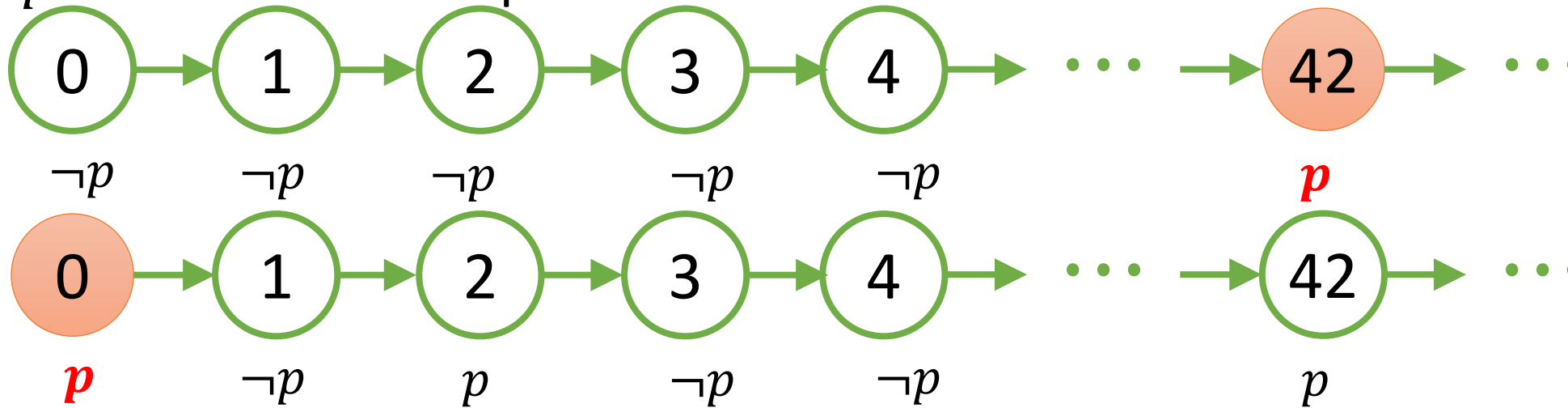
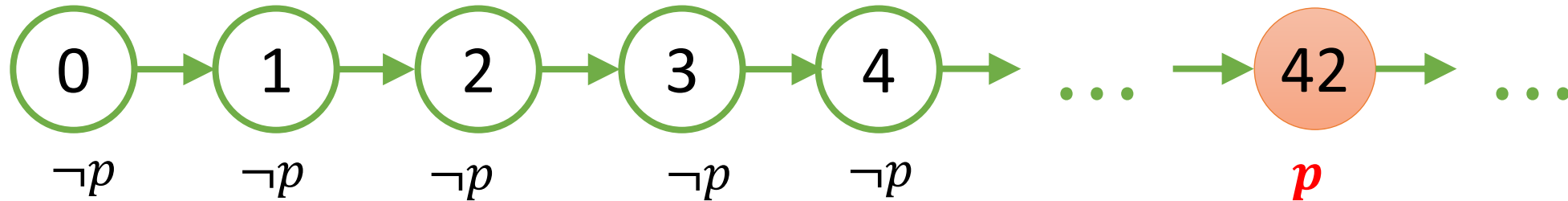# Visualizing the temporal operators

▶ **G**p: **G**lobally p holds
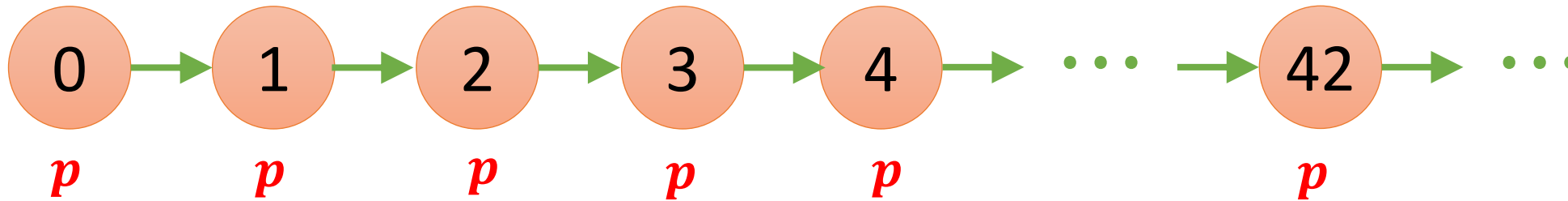
▶ p **U** q: p holds Until q holds
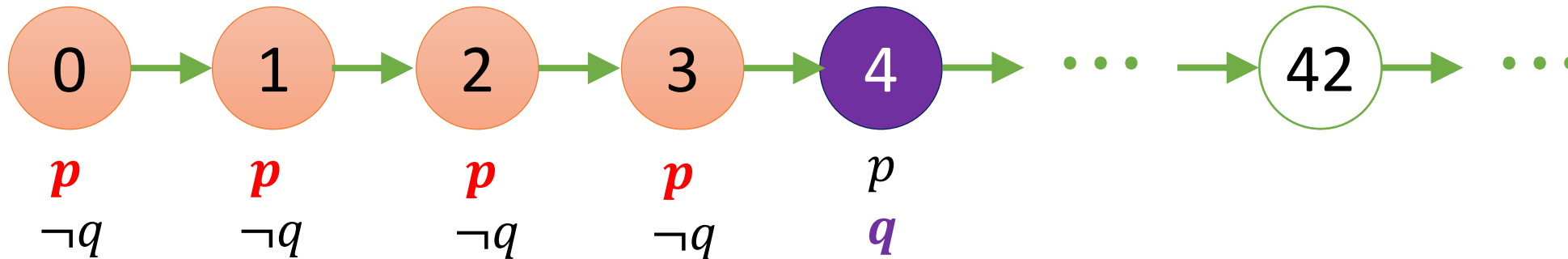
You can nest operators!

▶ 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

$\neg p$ $\neg p$ $\neg p$ $\neg p$ $\neg p$ $\boldsymbol{p}$

▶ 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

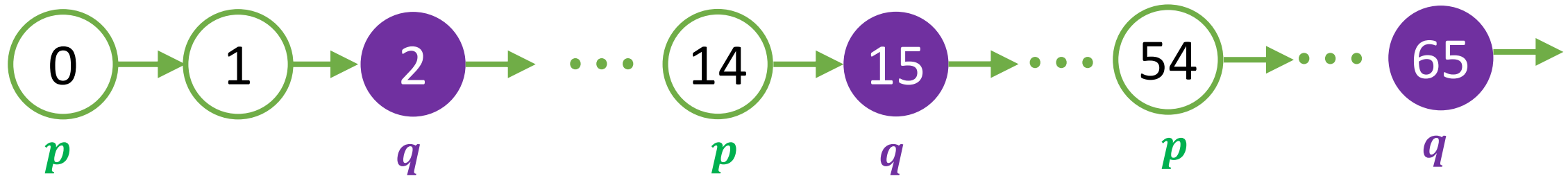$\neg p$ $\neg p$ $\boldsymbol{p}$ $\neg p$ $\boldsymbol{p}$ $\boldsymbol{p}$
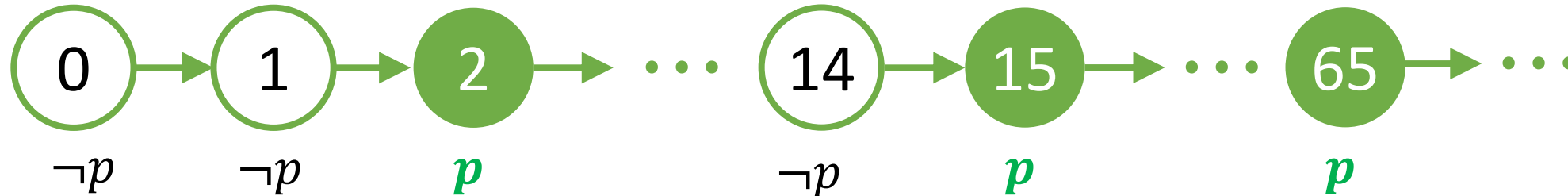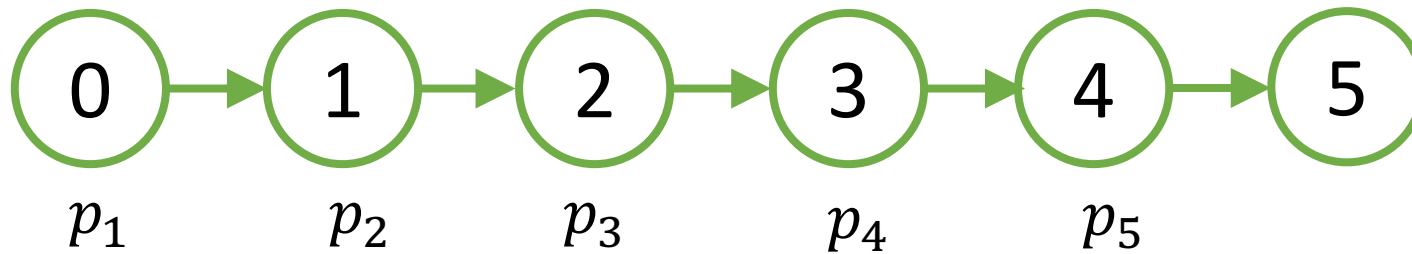
More operator fun

▶ What does **FG**p mean?
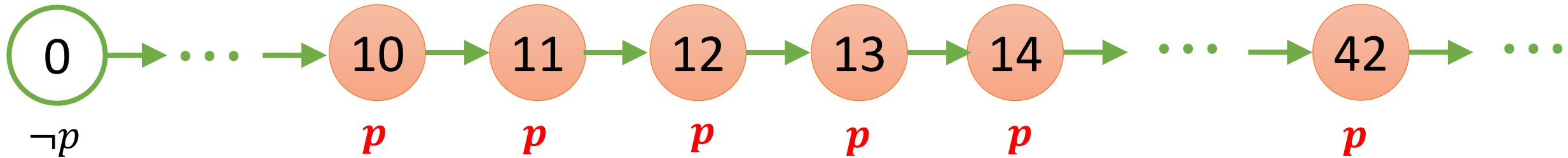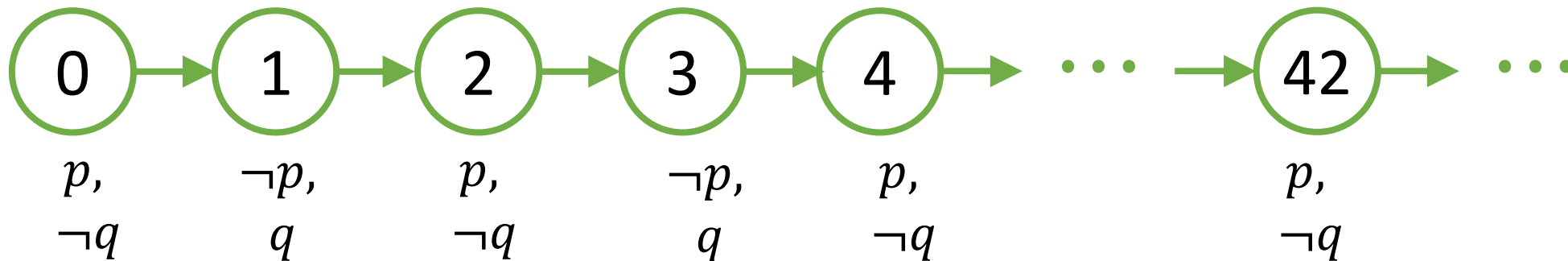

▶ What does **G**$(p \Rightarrow \mathbf{F}q)$ mean?

More, more operator fun

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

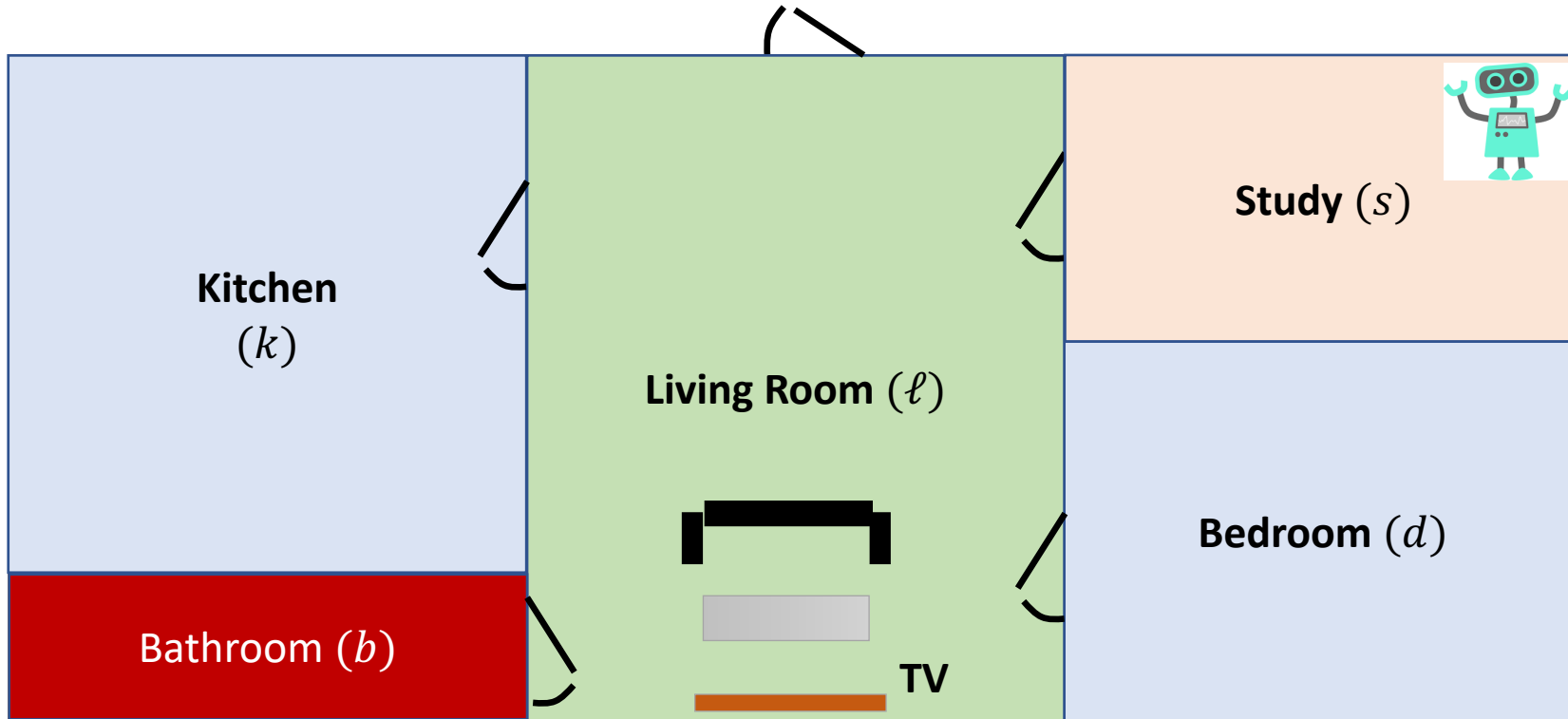▶ Is this true? $\mathbf{F}(p \land q)$ is the same as $\mathbf{F}p \land \mathbf{F}q$?

Operator duality and identities

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

Example specifications 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

Kitchen (k)

Living Room (ℓ)

Study (s)

Bedroom (d)

Bathroom (b)

TV
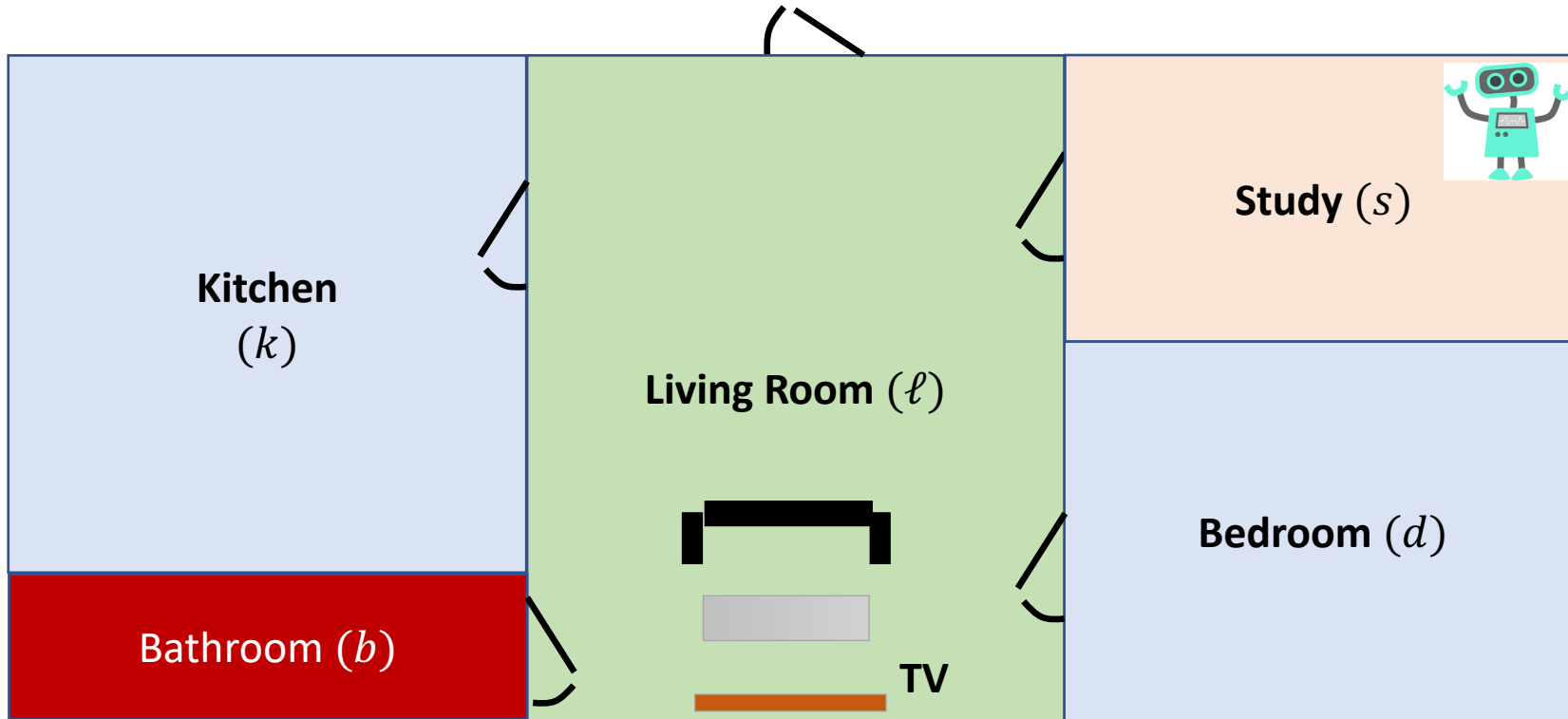
▶ The robot should repeatedly visit the living room

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

▶ 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) $\rightarrow$ $\quad$ 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 φ if **all system behaviors** starting in q_0 satisfy the requirement φ

▶ Denoted as $M, q_0 \vDash \varphi$

▶ E.g. a system is safe w.r.t. a safety requirement φ if all behaviors satisfy φ

▶ Does (**Blinker,** $(x \mapsto 0, y \mapsto 0)) \vDash \mathbf{G}(x \geq 0)$?

Processes & Fairness

nat x := 0; bool y:= 0

A: x := x + 1

B: even(x) →

   y: = 1-y
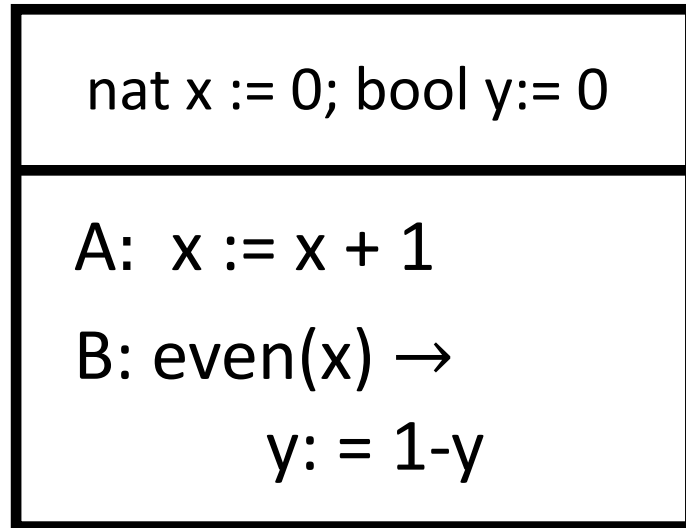
Blinker

▶ 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 true)
  ▶ 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

B: even(x) →

   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,∅} taken := ∅

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:wf(A) $:= (\mathbf{FG}\ guard\_i) \Rightarrow (\mathbf{GF}(\text{taken} = T_i))$

▶ Task A: $guard\_A$ is $true$, so this simplifies to:
$$wf(A) := \mathbf{GF}(\text{taken=A})$$

▶ Task B: wf(B) $:= \mathbf{FG}\ (\text{even}(x)) \Rightarrow \mathbf{GF}\ (\text{taken=B})$

▶ Does (wf(A)∧ wf(B)) $\Rightarrow \mathbf{F}\ (x \geq 10)$?
    ▶ **Yes!**

▶ Does (wf(A)∧ wf(B)) $\Rightarrow \mathbf{F}\ y$?
    ▶ **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:
$$sf(A) := \mathbf{GF}(\text{taken=A})$$

▶ Task B: $sf(B) := \mathbf{GF}\ (\text{even}(x)) \Rightarrow \mathbf{GF}\ (\text{taken=B})$

▶ Does $(sf(A) \wedge sf(B)) \Rightarrow \mathbf{F}\ (x \geq 10)$?
  ▶ **Yes!**

▶ Does $(sf(A) \wedge sf(B)) \Rightarrow \mathbf{F}\ y$?
  ▶ **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!