Cyber-Physical Systems

Laura Nenzi

Università degli Studi di Trieste
II Semestre 2022

Lecture 11: Automata and Temporal Logic

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
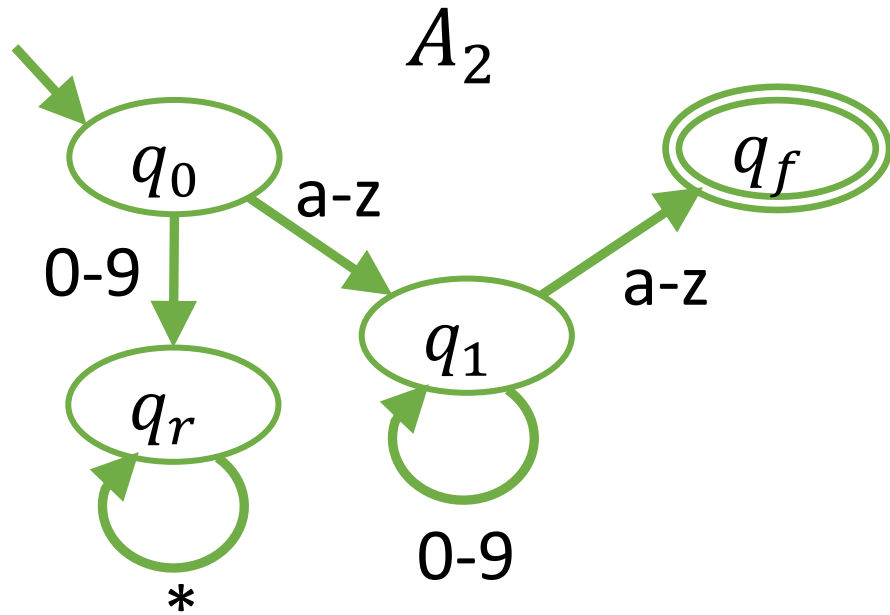
Finite State Automata (FSA)

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

q_f

a-z

State

q_1
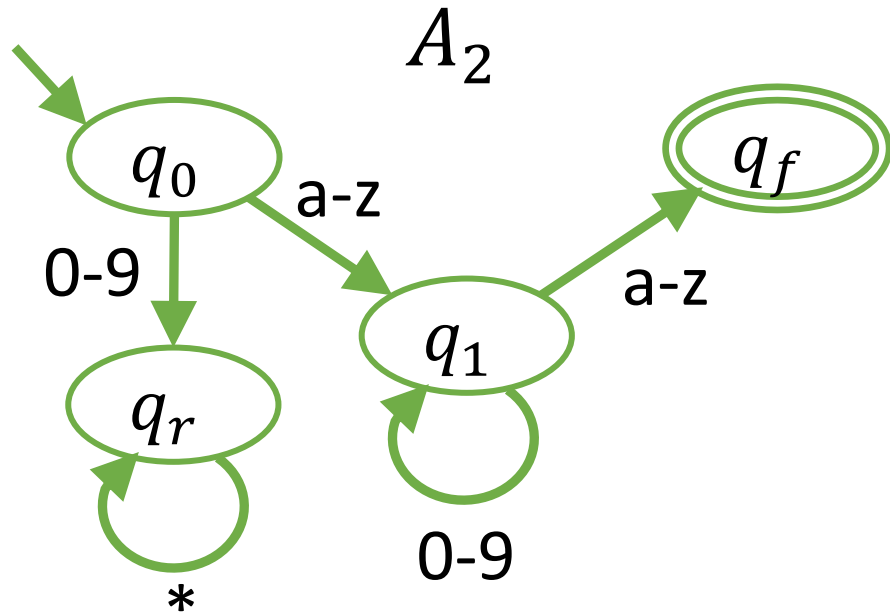
a-z,0-9

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

State

Accepting
state

Language of a finite state automaton

A_2
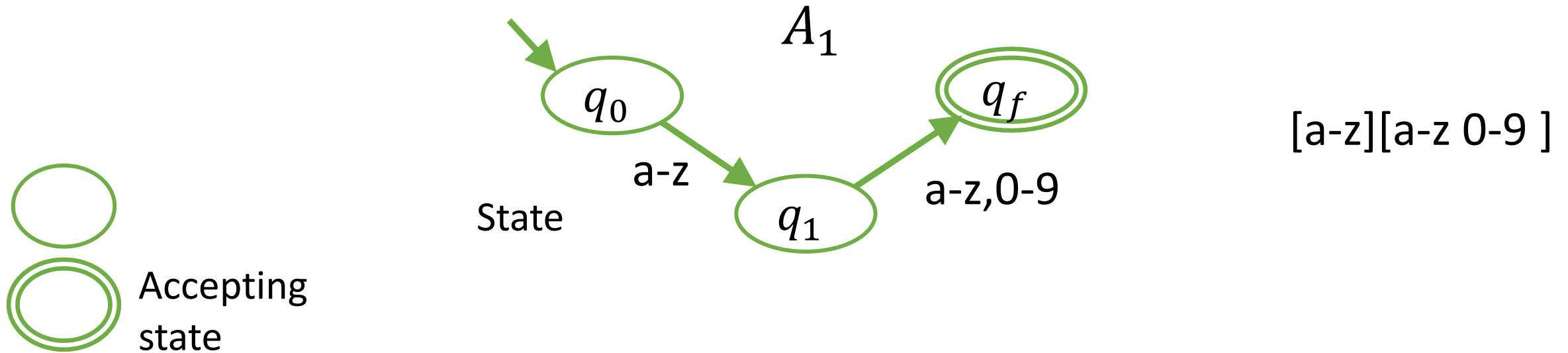
- ▶ What strings are accepted by A_2?
  - ▶ ab, zy, s2r, q123s, u3123123v, etc.
- ▶ What strings are not accepted by A_2?
  - ▶ 2b, 334a, etc.

How does a Finite State Automaton work?

A_2

q_0

0-9

a-z

q_r

q_1

q_f

a-z

0-9

*

[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

▶ 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

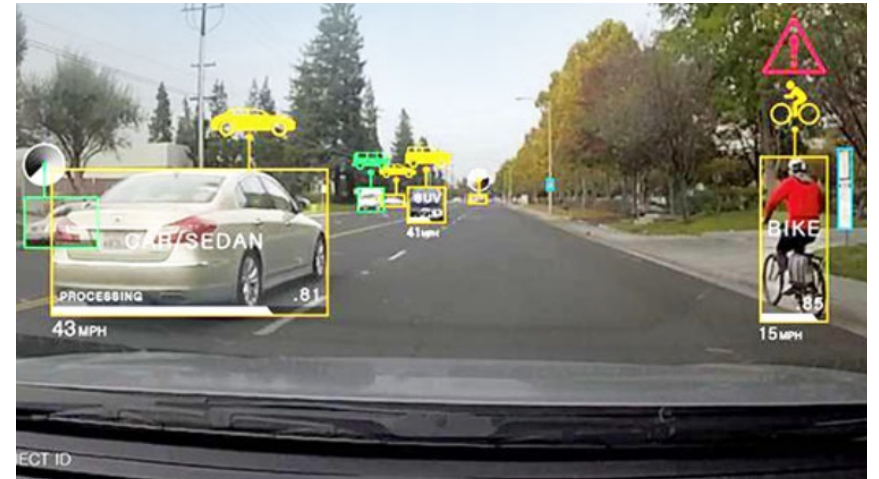| Syntax of Propositional Logic		
$\varphi ::=$	$true \mid$	the true formula
	$p \mid$	p is a prop in AP
	$\neg \varphi \mid$	Negation
	$\varphi \wedge \varphi \mid$	Conjunction
	$\varphi \vee \varphi \mid$	Disjunction
	$\varphi \Rightarrow \varphi \mid$	Implication
	$\varphi \equiv \varphi \mid$	Equivalence

Semantics

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

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

▶ 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	
$v(true)$	1
$v(p)$	1 if $v(p) = 1$
$v(\neg\varphi)$	1 if $v(\varphi) = 0$ 0 if $v(\varphi) = 1$
$v(\varphi_1 \wedge \varphi_2)$	1 if $v(\varphi_1) = 1$ and $v(\varphi_2) = 1$, 0 otherwise
$\varphi_1 \vee \varphi_2$	$v(\neg(\neg\varphi_1 \wedge \neg\varphi_2))$
$\varphi_1 \Rightarrow \varphi_2$	$v(\neg\varphi_1 \vee \varphi_2)$
$\varphi_1 \equiv \varphi_2$	$v\big((\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)\big)$

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

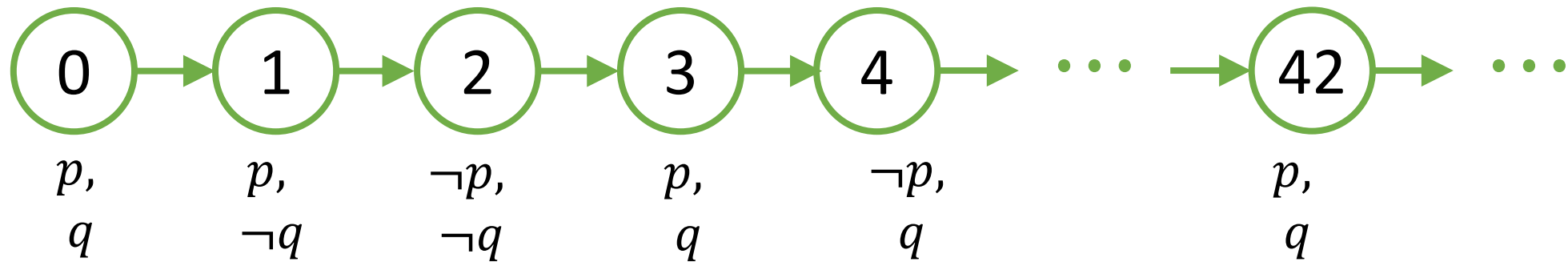▶ $v\big((p_1 \wedge p_2) \Rightarrow p_3\big)$ =1

▶ $v: p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0$. 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? $v\Big(\underline{(p_1 \wedge p_2) \Rightarrow p_3} \equiv \underline{(p_1 \Rightarrow p_3) \wedge (p_2 \Rightarrow p_3)}\Big)$ = 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:

| 0 | 1 | 2 | 3 | 4 | ⋯ | 42 | ⋯ |

$p,$ q $p,$ $\neg q$ $\neg p,$ $\neg q$ $p,$ q $\neg p,$ q $p,$ q

▶ 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 ⊨ (read models) to show that a trace-point satisfies a formula

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

▶ 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 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
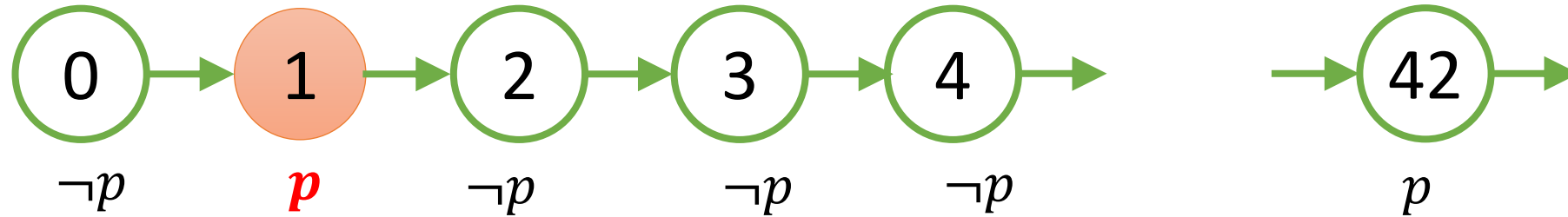
▶ **F**p : Some **F**uture step
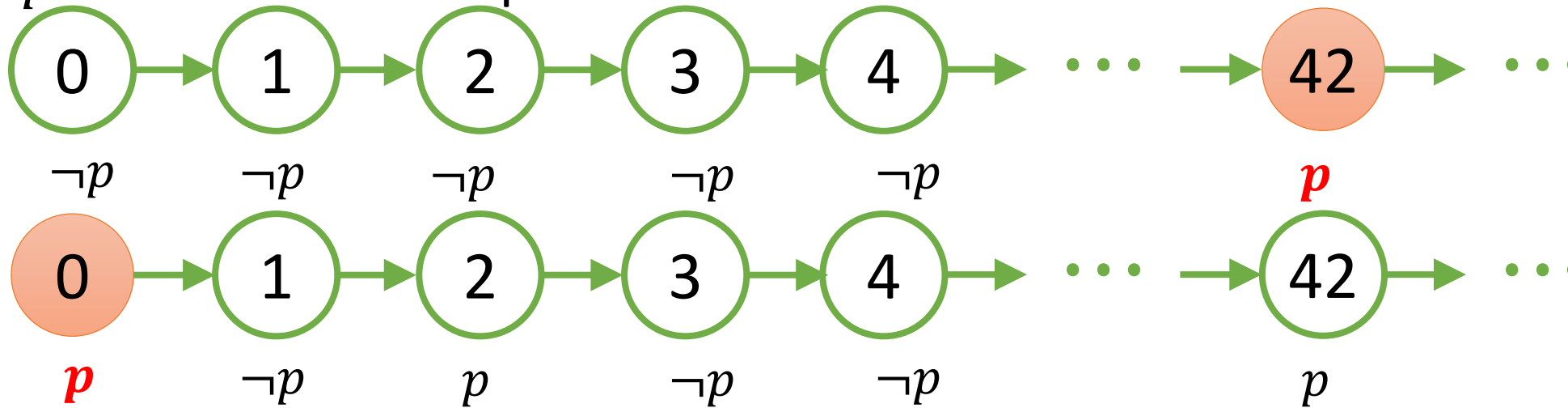
Visualizing the temporal operators
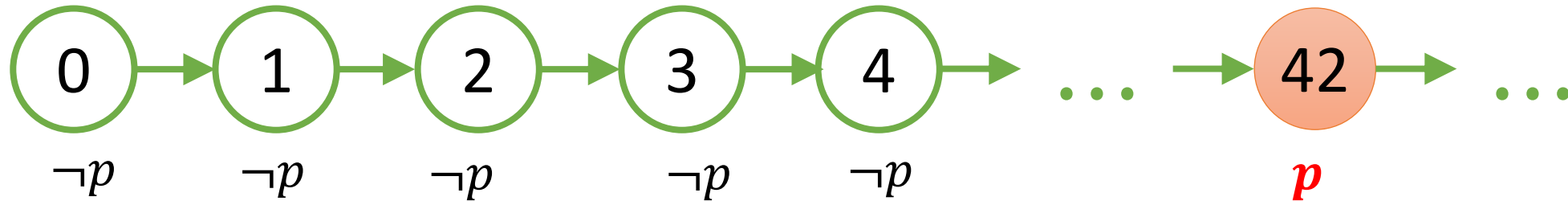
▶ **G**p: **G**lobally p holds
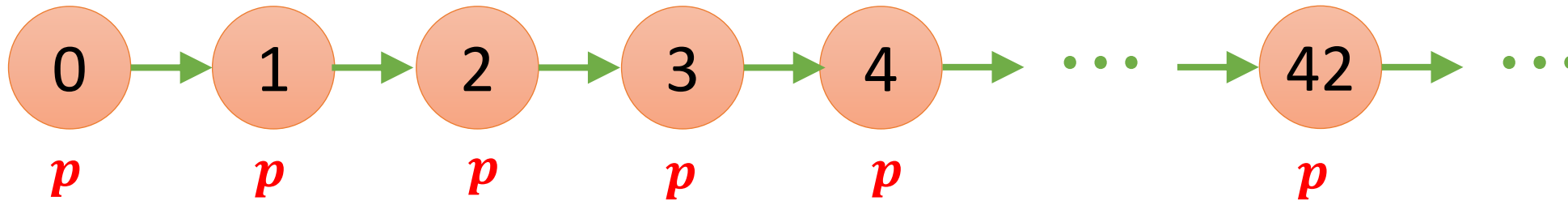
▶ p **U** q: p holds Until q holds
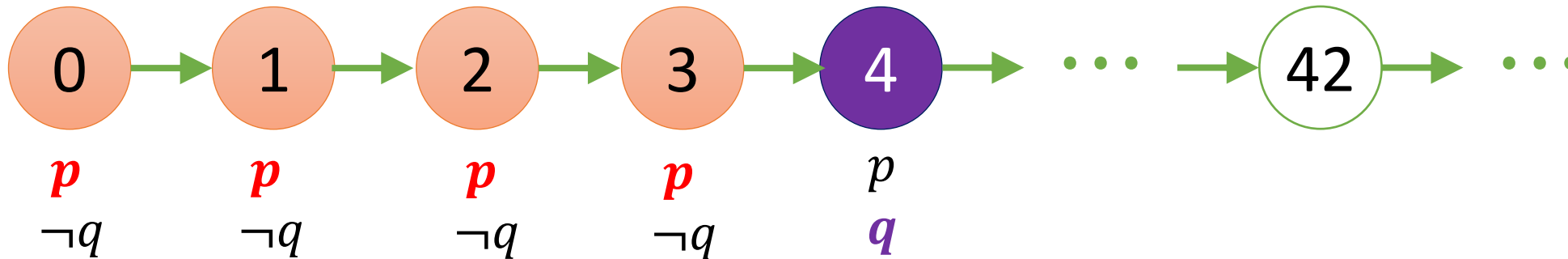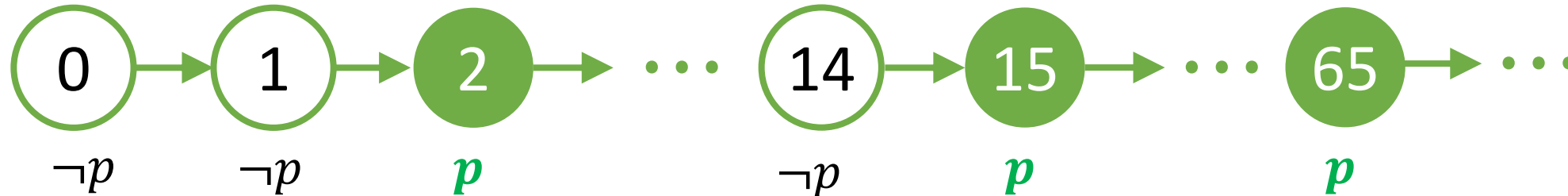
You can nest operators!

▶ What does **XF** p mean?

▸ Trace satisfies **XF**p (at time 0) if at time 1, **F**p holds. I.e. p holds at some point strictly in the future
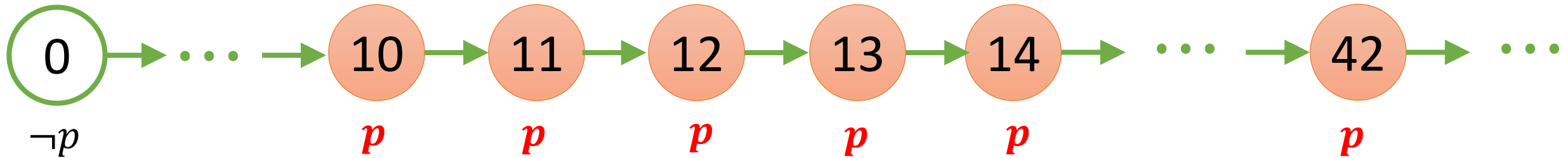

▶ What does **GF** p mean?

▸ Trace satisfies **GF**p (at time 0) if at n, there is always a $\boldsymbol{p}$ in the future
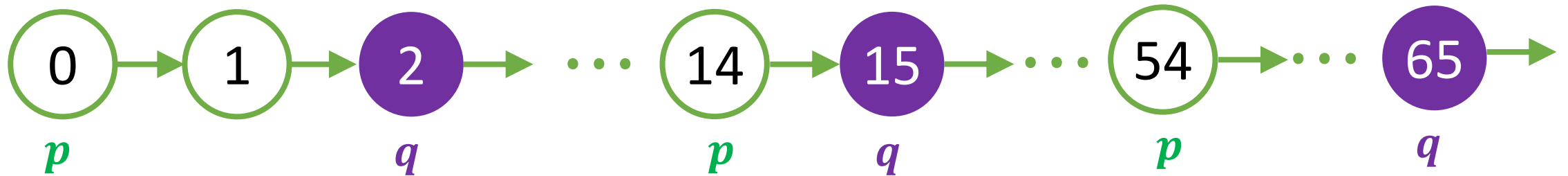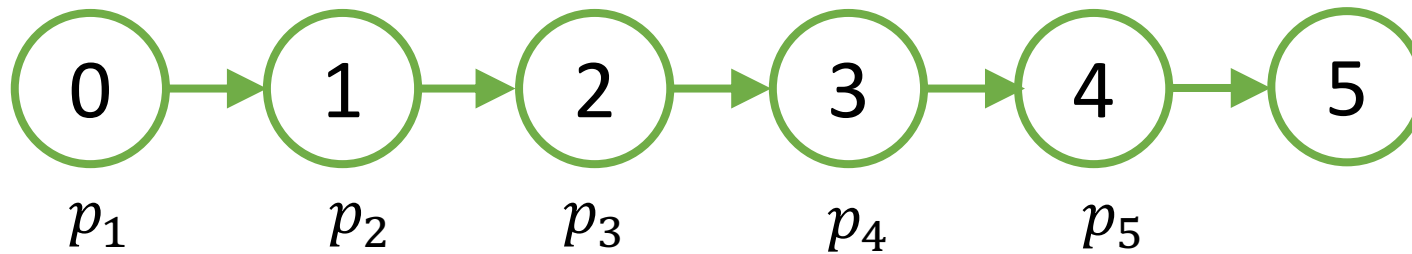
More operator fun

▶ What does **FG**p mean?
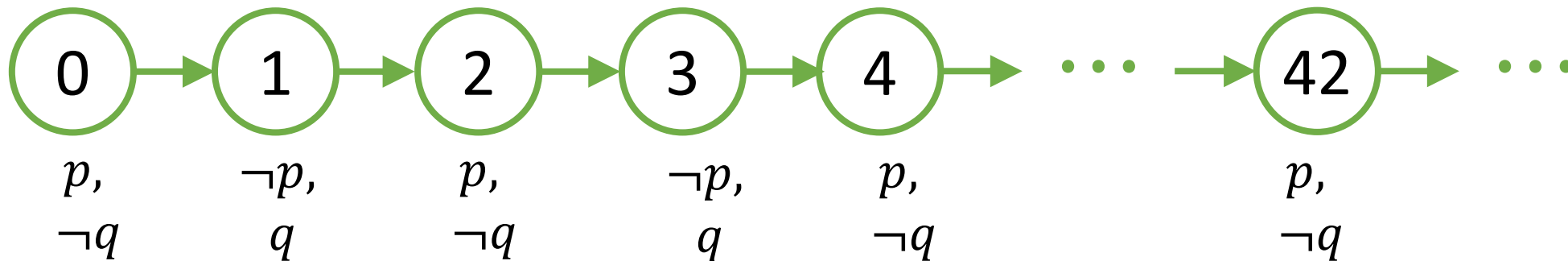

▶ What does **G**$(p \Rightarrow \textbf{F}q)$ mean?

More, more operator fun

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

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

Linear Temporal Logic (LTL) specification

It is a logic interpreted over infinite discrete-time traces

E.g. It is always true that the highest temperature will be below 75 degree and the lowest temperature will be above 60 degree

G(p ∧ q)   p = T<75, q=T>60

Linear Temporal Logic (LTL) specification

It is a logic interpreted over infinite discrete-time traces

E.g. **For the next 3 days** the highest temperature will be below 75 degree and the lowest temperature will be above 60 degree

X (p ∧ q) ∧ X X (p ∧ q) ∧ X X X (p ∧ q)   with p = T<75, q=T>60
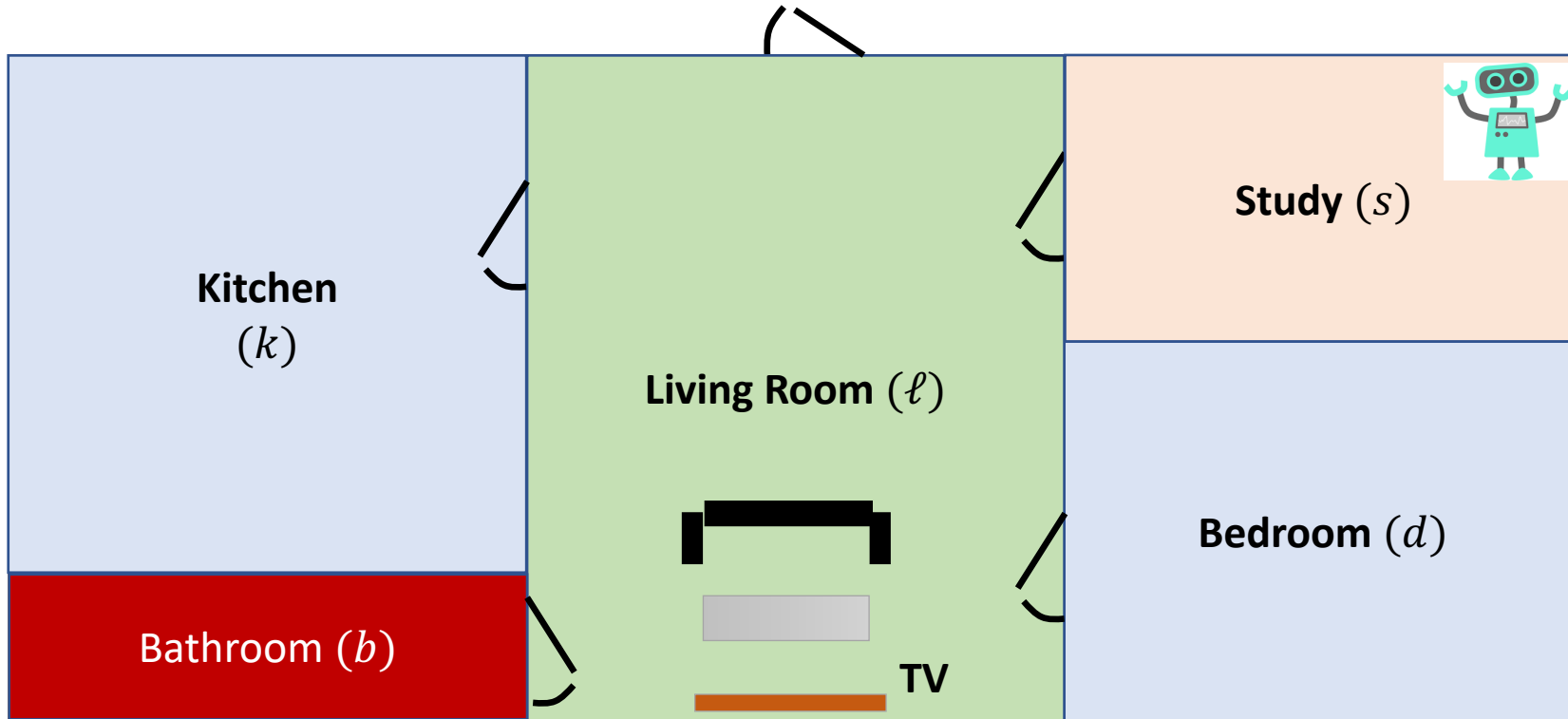
Operator duality and identities

- $\mathbf{F}\varphi \equiv \neg\mathbf{G}\neg\varphi$

- $\mathbf{GF}\varphi \equiv \neg\mathbf{FG}\neg\varphi$

- $\mathbf{F}(\varphi \vee \psi) \equiv \mathbf{F}\varphi \vee \mathbf{F}\psi$

- $\mathbf{G}(\varphi \wedge \psi) \equiv \mathbf{G}\varphi \wedge \mathbf{G}\psi$

- $\mathbf{FF}\varphi \equiv \mathbf{F}\varphi$

- $\mathbf{GG}\varphi \equiv \mathbf{G}\varphi$

- $\mathbf{FGF}\varphi \equiv \mathbf{GF}\varphi$

- $\mathbf{GFG}\varphi \equiv \mathbf{FG}\varphi$

Example specifications 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
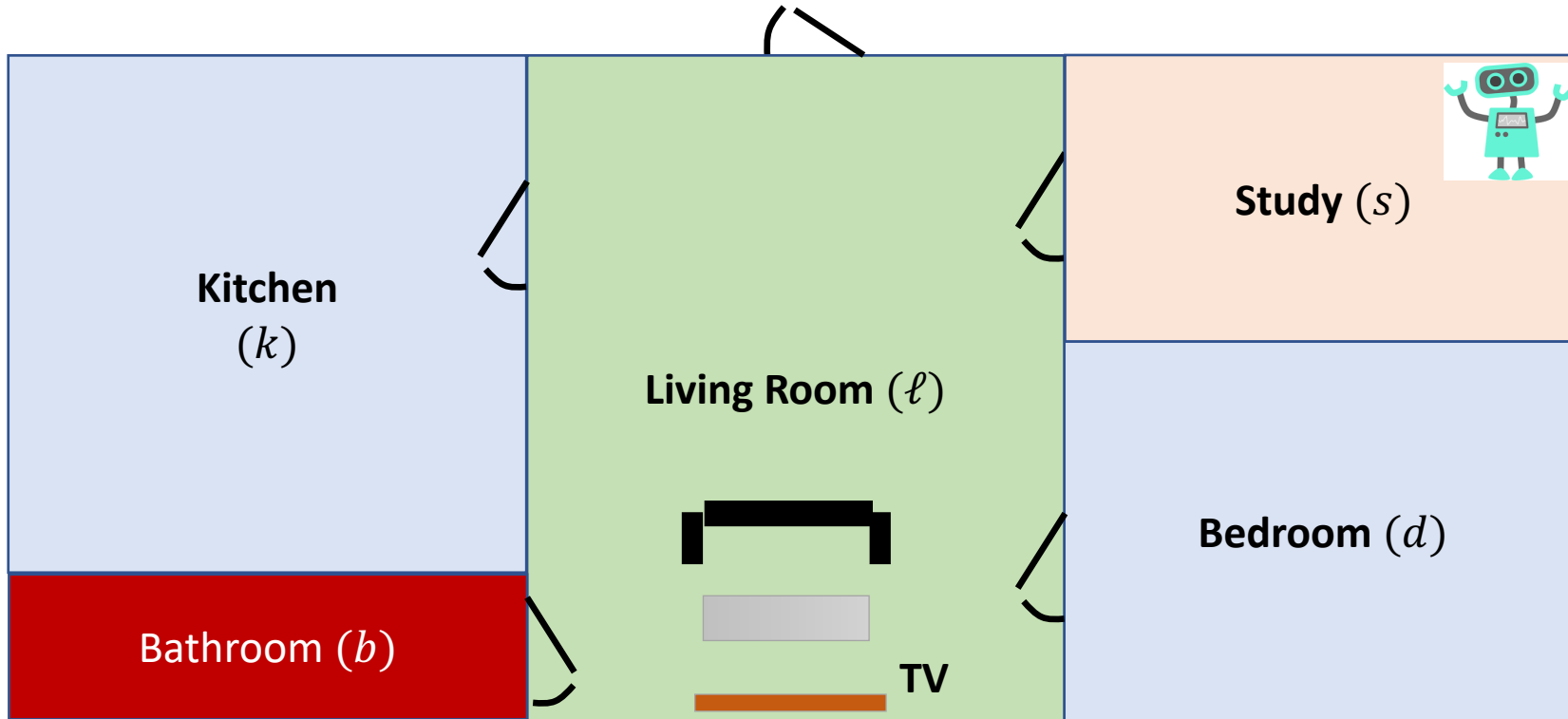
▶ The robot should repeatedly visit the living room

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

▶ 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

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

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

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!

(Hard) Requirements

▶ High assurance/safety-critical, or mission-critical systems should use hard requirements.

▶ Verification check whether the implementation meets the requirements

▶ A system design meets its requirements if all system executions satisfy all the requirements.

▶ There should ideally be clear separation between requirements (what needs to be implemented) and the design (how should it be implemented).

▶ Unfortunately, this simple philosophy is often not followed by designers.

(Hard) Requirements

▶ **Safety** and **liveness** requirements require fundamentally different classes of model checking algorithms

▶ **safety** requirement: "system never does something bad"

"if something bad happens on an infinite run, then it happens already on some finite prefix"

Counterexamples no reachable ERROR state

▶ **liveness** requirement: "system eventually does something good "

"no matter what happens along a finite run, something good could still happen later"

Infinite-length counterexamples, loo

Requirements example

► It cannot happen that both processes are in their critical sections simultaneously

► Whenever process P1 wants to enter the critical section, then process P2 gets to enter at most once before process P1 gets to enter.

► Whenever process P1 wants to enter the critical section, provided process P2 never stays in the critical section forever, P1 gets to enter eventually.

► The elevator will arrive within 30 seconds of being called

► Patient's blood glucose never drops below 80 mg/dL

Requirements example (Safety vs Liveness)

▶ It cannot happen that both processes are in their critical sections simultaneously S

▶ Whenever process P1 wants to enter the critical section, then process P2 gets to enter at most once before process P1 gets to enter. S

▶ Whenever process P1 wants to enter the critical section, provided process P2 never stays in the critical section forever, P1 gets to enter eventually. L

▶ The elevator will arrive within 30 seconds of being called   S (observe the finite prefix of all computation steps until 30 seconds have passed, and decide the property, therefore safety)

▶ Patient's blood glucose never drops below 80 mg/dL. S

.

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 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↦0,y↦0)) $\vDash$ **G**(x≥0)?

Processes & Fairness

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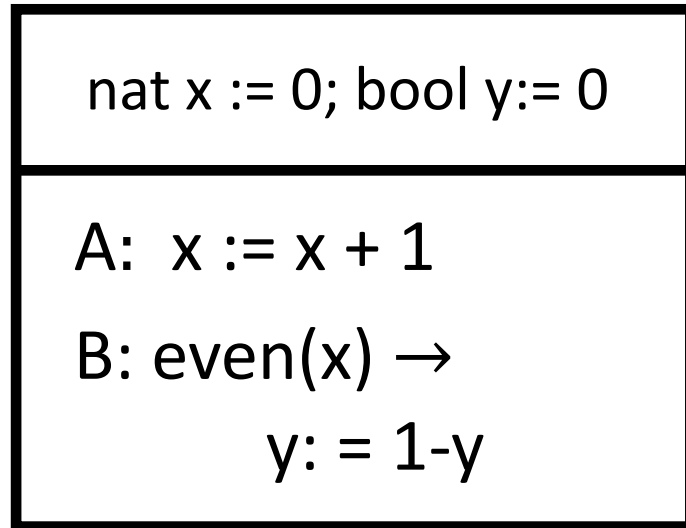
Blinker

▶ Liveness property: $\mathbf{F}$ (x $\geq$ 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:
$$\text{wf(A)} := \mathbf{GF}(\text{taken=A})$$

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

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

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

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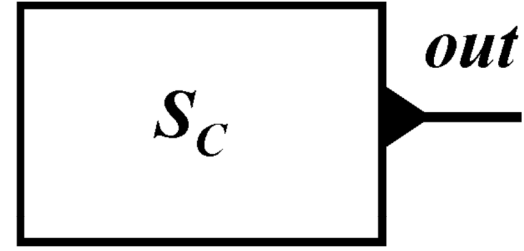 ▶ **Yes!**

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

Open vs. Closed Systems

▶ A closed system is one with no inputs
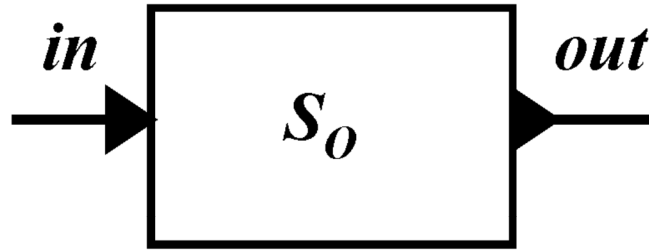
(a) Open system   (b) Closed system

For verification, we obtain a closed system by composing the system and environment models
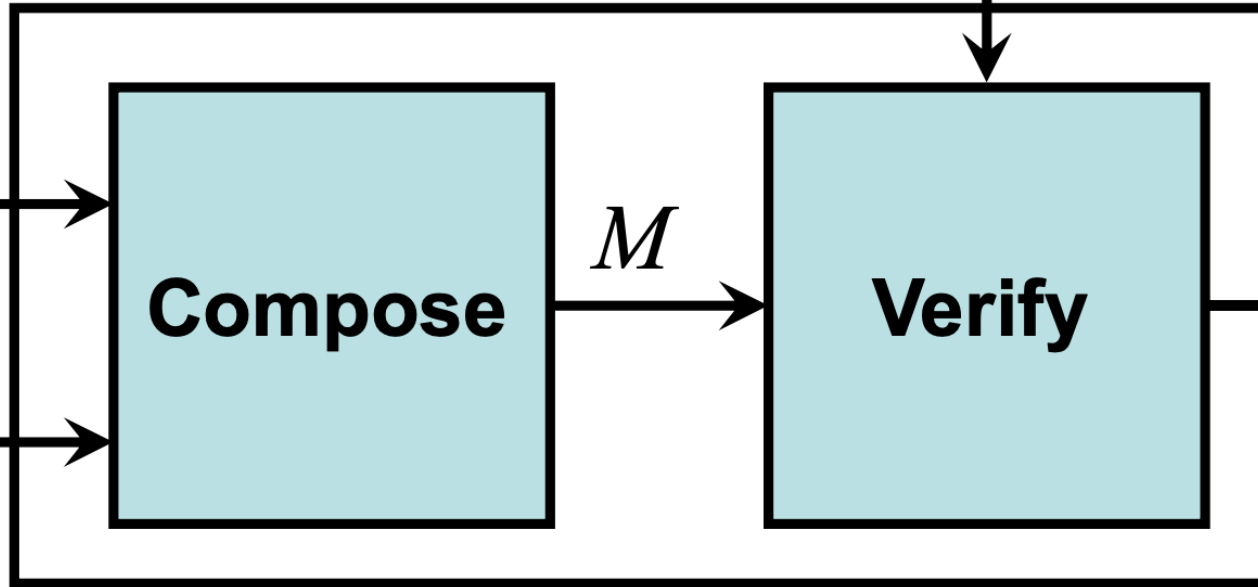
Formal Verification

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 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?

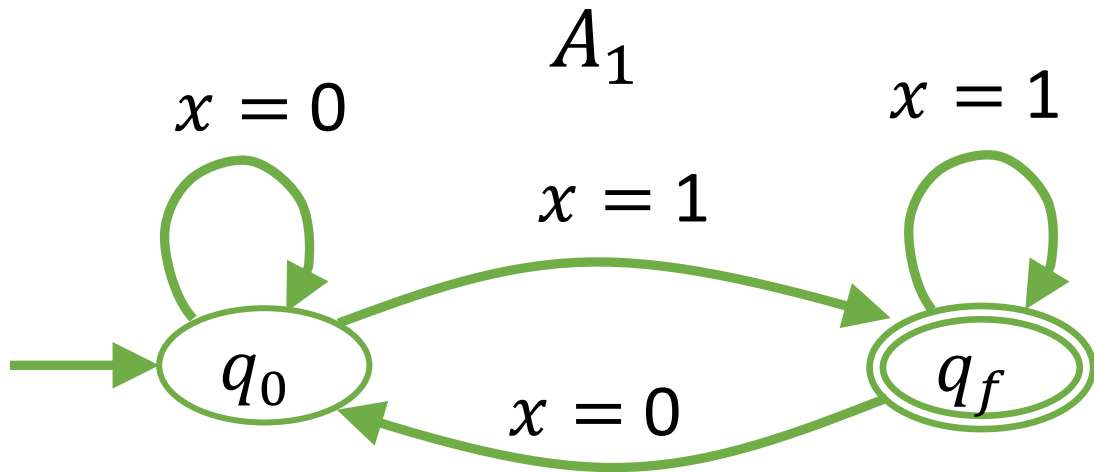▶ Yes, using theoretical model of Büchi automata proposed by J. Richard Büchi in 1960

Büchi automaton Example 1

▶ Extension of finite state automata to accept infinite strings

A_1

$x = 0$

$x = 1$

$x = 1$

$x = 0$

q_0

q_f

▶ States Q: $\{q_0, q_f\}$

▶ Input variable x with domain Σ: $\{0,1\}$

▶ Final state: $\{q_f\}$

▶ Transitions: (as shown)

▶ Given trace ρ (infinite sequence of symbols from Σ), ρ 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

A_2

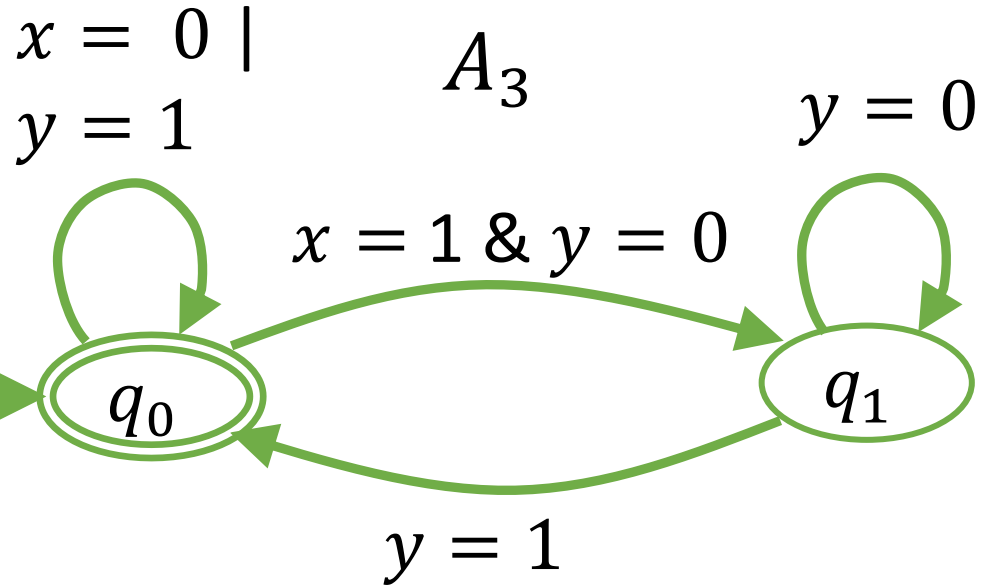$x = 0|1$ $x = 1$

q_0 $x = 0|1$ q_f
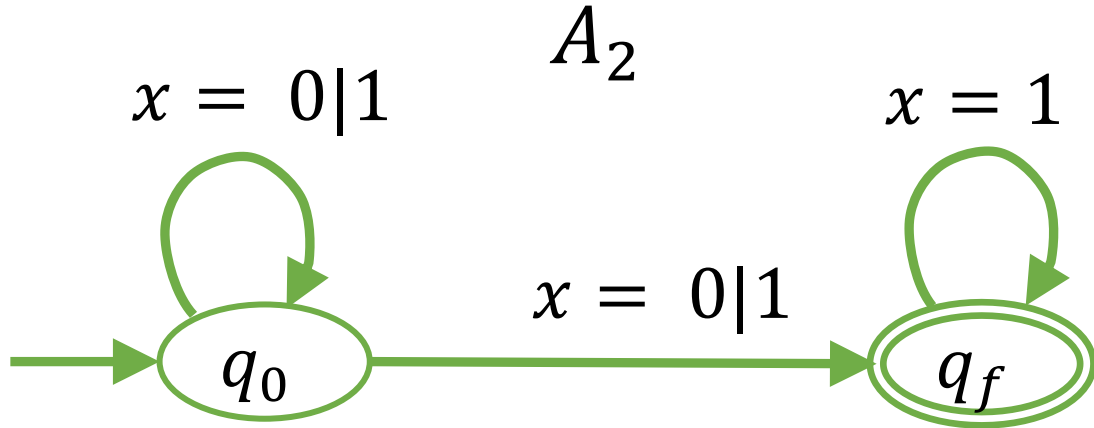
▶ $Q: \{q_0, q_f\}, \Sigma: \{0,1\}, F: \{q_f\}$

▶ 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 ρ 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 φ can be converted to a Büchi monitor/automaton A_φ

▶ Size of A_φ is generally exponential in the size of φ; blow-up unavoidable in general

▶ Construct composition of the original process P and the Büchi monitor A_φ

▶ 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)

Reachability, MC, Monitoring and SMC

▶ **Reachability** analysis is the process of computing the set of reachable states for a system

▶ **Model checking (MC)** is an algorithmic method for determining if a system satisfies a formal specification expressed in temporal logic

$$M \vDash \phi \iff \forall \mathbf{x} \in trace\ (M)\ \ \beta(\ \varphi, \mathbf{x}, 0\ ) = 1$$

Type equation here.

▶ **Monitoring**: computing β for a single trace $\mathbf{x} \in trace\ M$

▶ **Statistical Model Checking (SMC)**: "doing statistics" on $\beta\ (\varphi, \mathbf{x}, 0)$ for a finite-subset of $trace\ (M)$