Cyber-Physical Systems

Laura Nenzi

Università degli Studi di Trieste

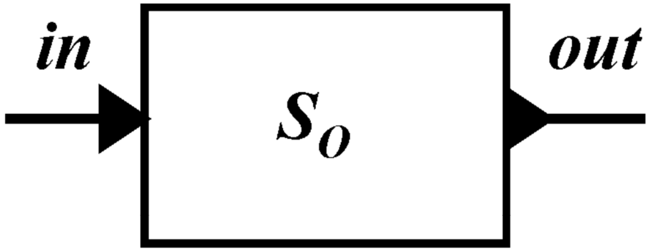
II Semestre 2018

Lecture 11: Verification

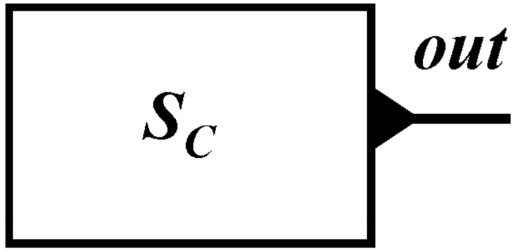
[Many Slides due to J. Deshmukh]

Open vs. Closed Systems

- A closed system is one with no inputs



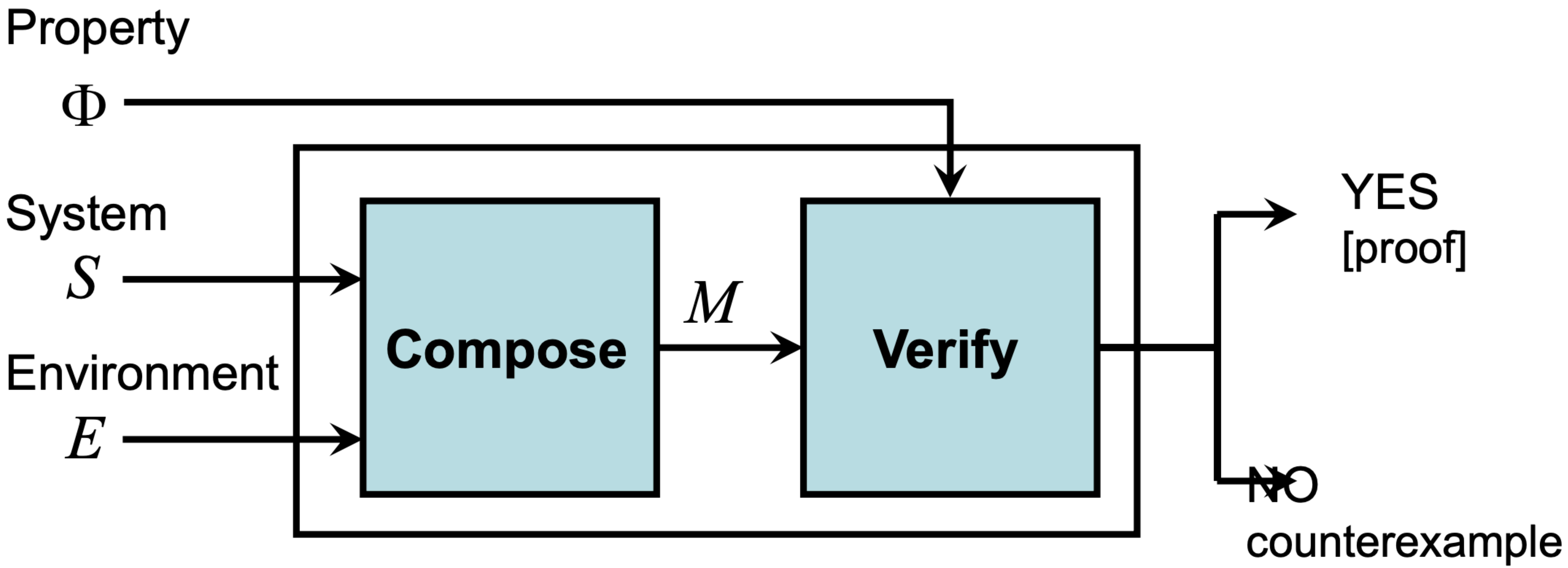
(a) Open system



(b) Closed system

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

Formal Verification



Requirements/Property

- A requirement describes a desirable property of the system behaviors.

- A Model satisfies its requirements if *all* system executions satisfy all the requirements.

- Two broad categories:

  - **safety** requirement: “nothing bad ever happens”,

  - **liveness** requirement: “something good eventually happens”

- **Importance of this classification**: these two classes of properties require fundamentally different classes of model checking algorithms

Requirements/Property

- **safety** requirement:

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

  Counterexamples no reachable ERROR state

- **liveness** requirement:

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

  Infinite-length counterexamples, loop

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

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

Reachability Analysis and Model Checking

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

- **Model checking** is an algorithmic method for determining if a system satisfies a formal specification expressed in temporal logic

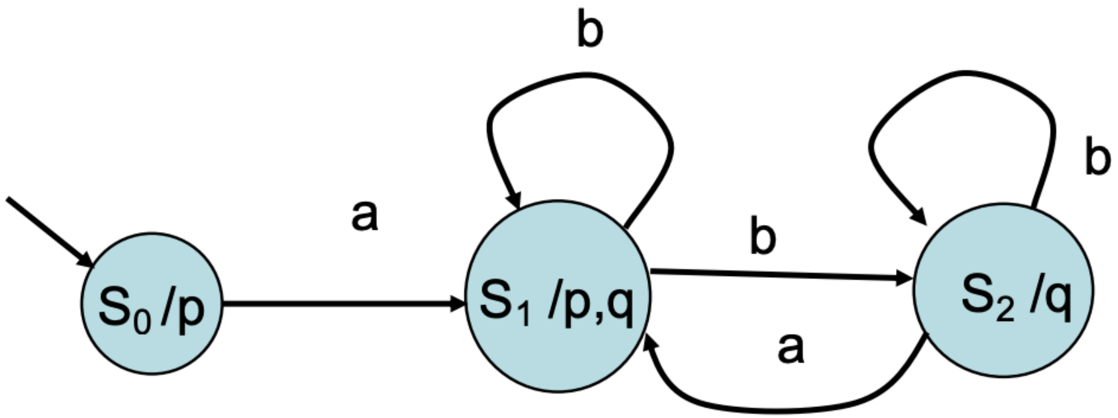
Model checking typically performs reachability analysis.

Safety Requirements

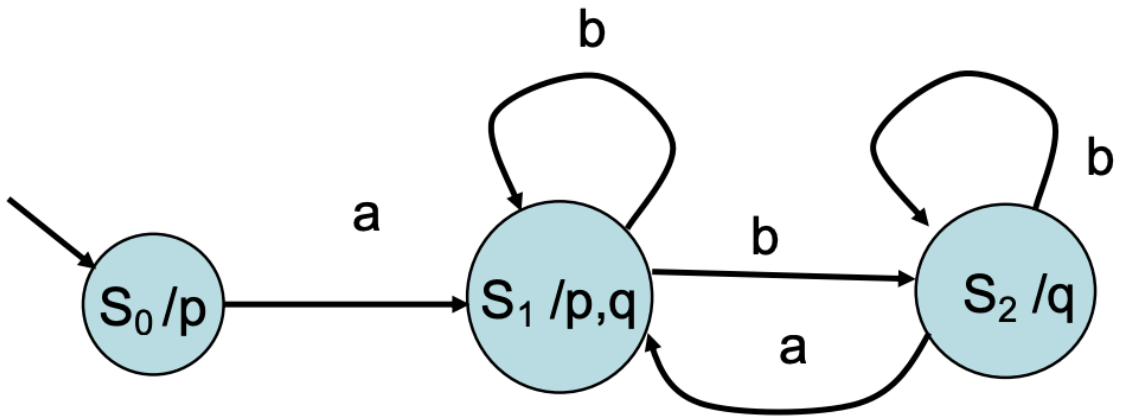
- To verify a safe requirements p on a system M, one simply needs to enumerate all the reachable states and check that they all satisfy p.
- A safety requirement for a system classifies its states into safe and unsafe and asserts that an unsafe state is never encountered during an execution of the system.
- Safety requirements can be formalized using transition systems

(Label) Transition System

- Transition System is a tuple $< S, I, A, [\![T]\!], AP, L >$
  - S: Set of State
  - $I \subseteq S$: set of initial state
  - A: finite set of actions
  - $[\![T]\!]$: is a set of transition relation s $\rightarrow^{a}$ s'
  - AP: set of atomic proposition on S
  - $\mathrm{L}: S \rightarrow 2^{AP}$ is a labeling function



Transition System



- A **path** is an (infinite) sequence of states in the TS
  e.g. σ = $S_0\ S_1 S_2 S_2 S_2$ ...

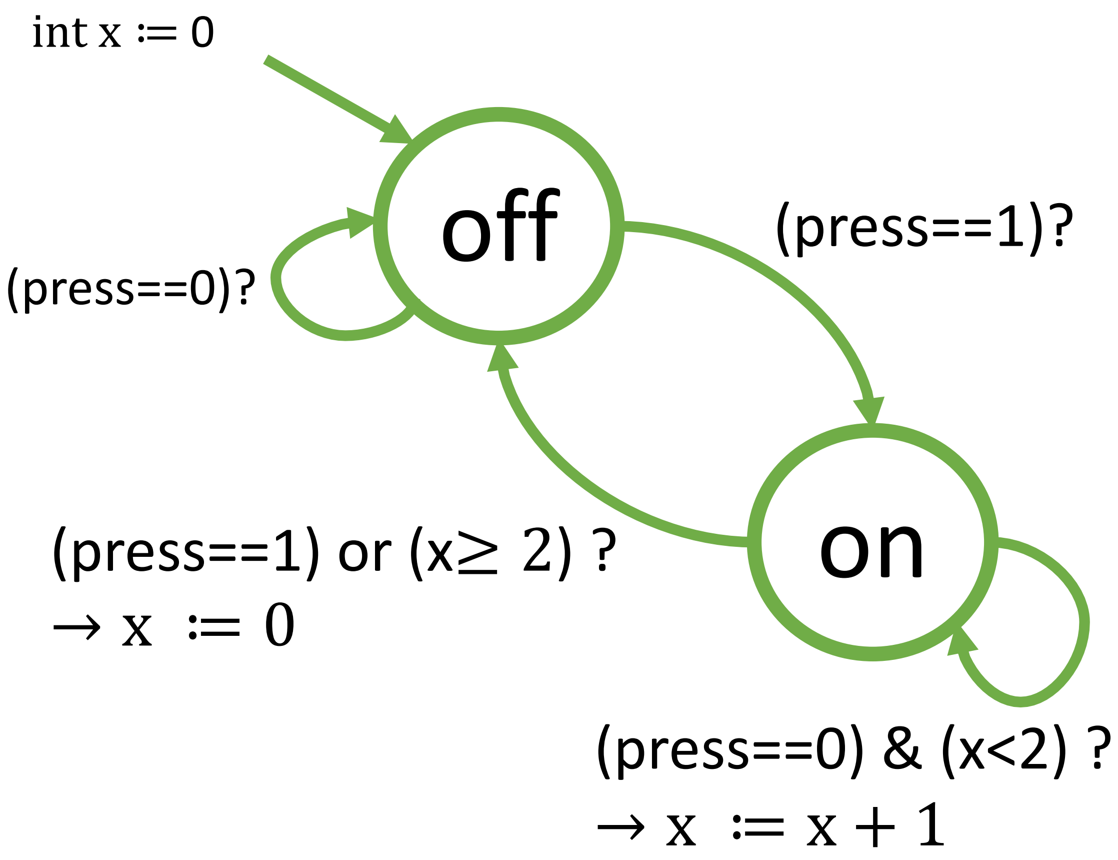
- A **trace** is the corresponding sequence of labels
  e.g. $p\{p, q\}qqq$

- A **word** is a sequence of actions
  e.g. $abbbb$

Transition Systems and state

- All kinds of components (synchronous, asynchronous, timed, hybrid, continuous components have an underlying transition system)
- State in the transition system underlying a component captures any given runtime configuration of the component
- If a component has finite input/output types and a finite number of “states” in its ESM, then it has a finite-state transition system
- Continuous components, Timed Processes, Hybrid Processes in general, have infinite number of states

Example of a TS

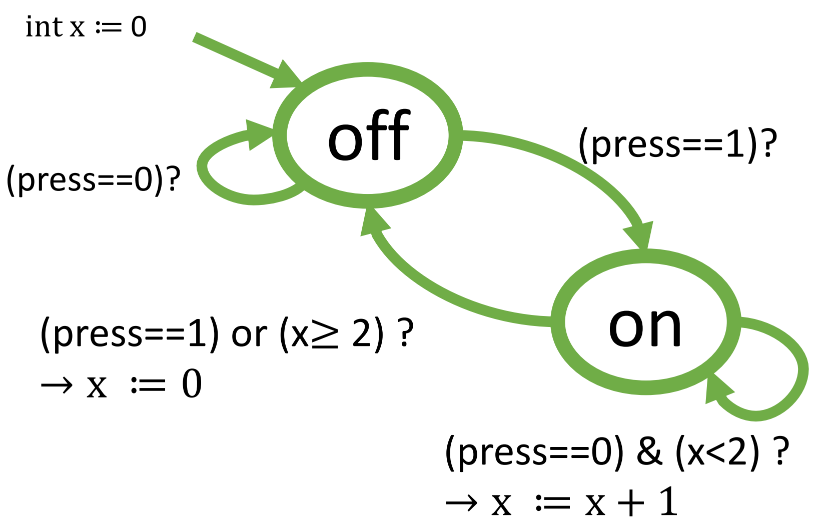


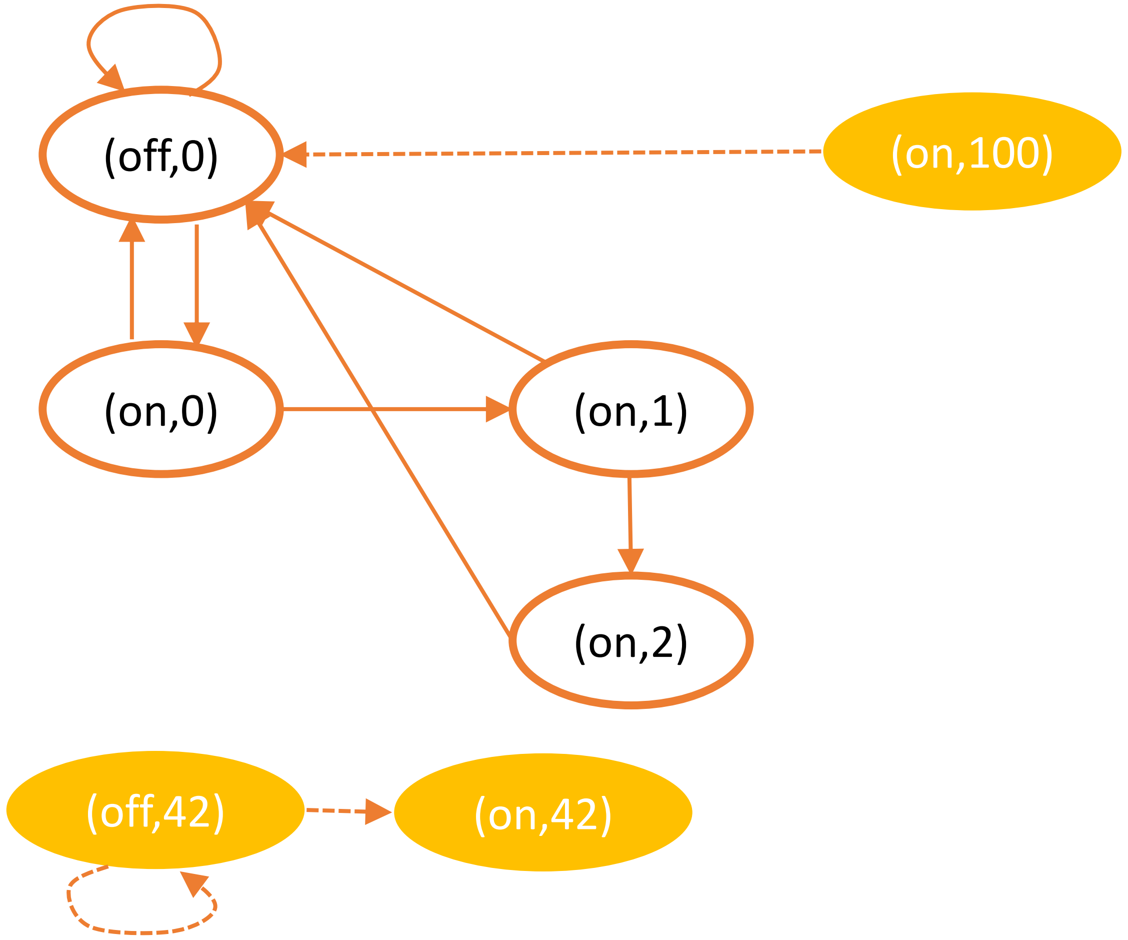
- $S = \{\mathrm{on}, \mathrm{off}\} \times \mathrm{int}$
- I = { off, $x = 0$ }
- $[\![T]\!]$ has an infinite number of transitions:

$$s \to s'$$

- E.g. $(off, 0) \to (on, 0)$ $(on\ 0) \to (on, 1)$

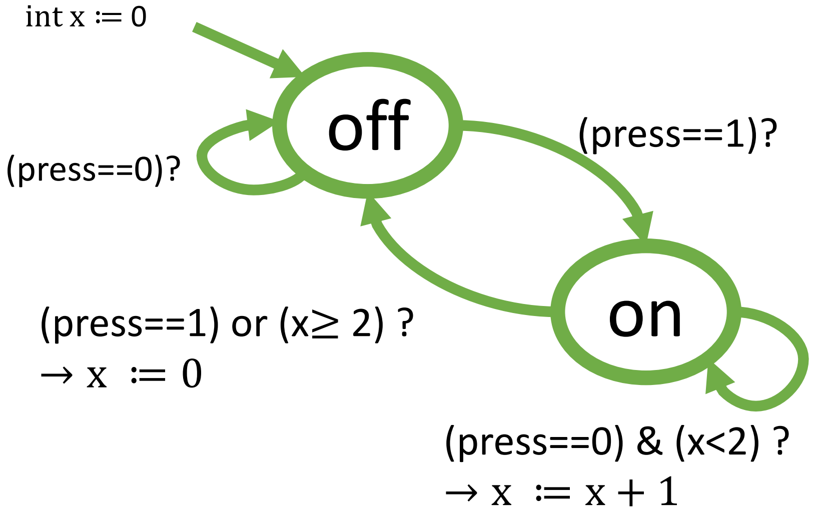
TS describes all possible transitions

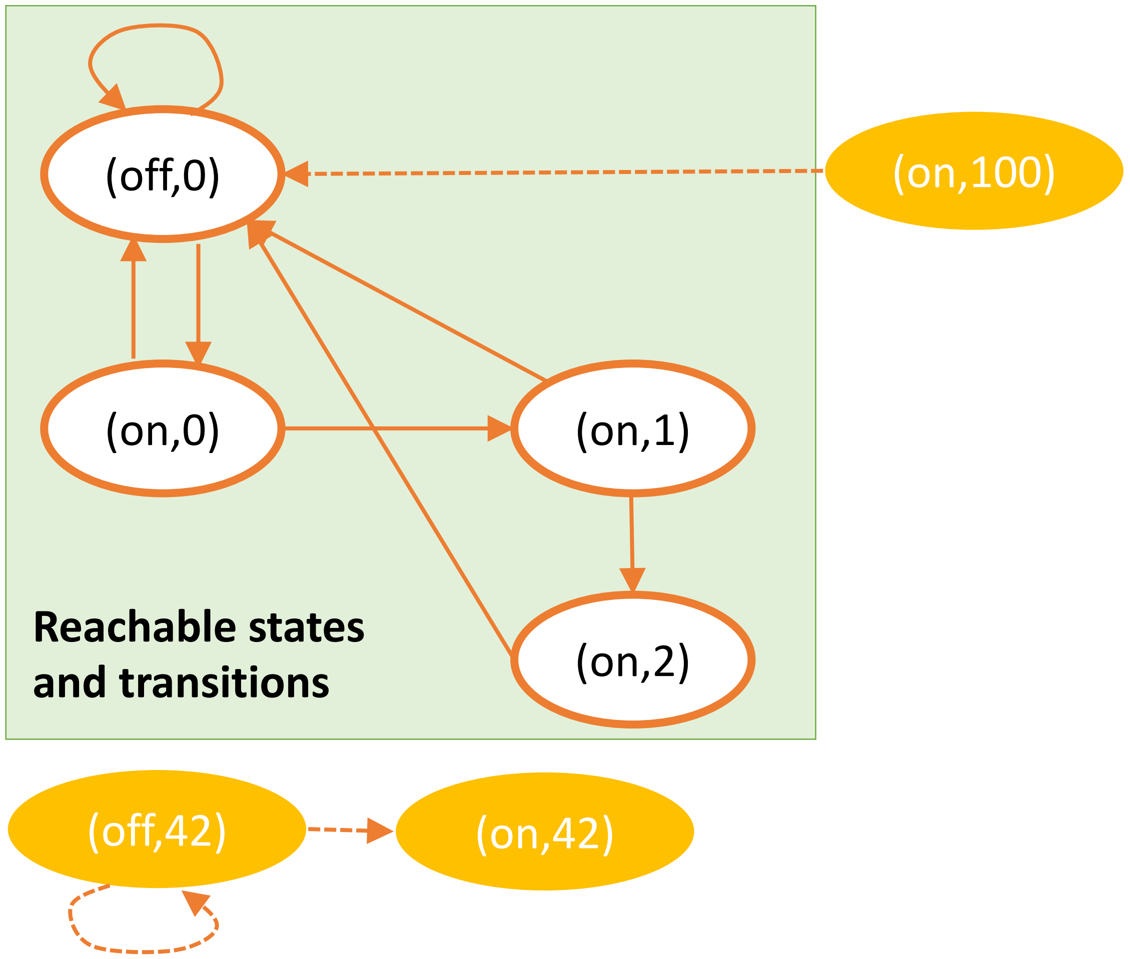




- Transitions indicated as dotted lines can't really happen in the component
- But, the TS will describe then, as the states of the TS are over $\{on, off\}$×int!

Reachable states of a modified switch TS





A state s of a transition system is ***reachable*** if there is an execution starting in some initial state that ends in s.

Desirable behaviors of a TS

- Desirable behavior of a TS: defined in terms of acceptable (finite or infinite) sequences of states

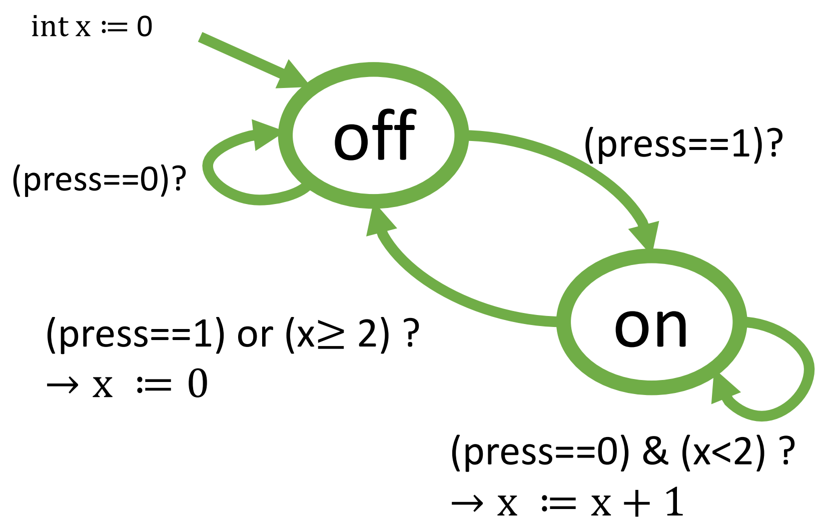
- **Safety property** can be specified by partitioning the states S into a safe/unsafe set
  - $Safe \subseteq S, Unsafe \subseteq S, Safe \cap Unsafe = \emptyset$
  - Any finite sequence that ends in a state $q \in Unsafe$ is a witness to undesirable behavior,
    or if all (infinite) sequences starting from an initial state never include a state from $Unsafe$, then the TS is safe.

Invariants

- A property φ is called an **invariant** of TS if every reachable state of TS satisfies φ

Examples:

- $(mode = off)$
- $(\mathrm{x} < 2)$
- $(mode = off) \Rightarrow (\mathrm{x} = 0)$ | INVARIANT
- $(\mathrm{x} \leq 50)$

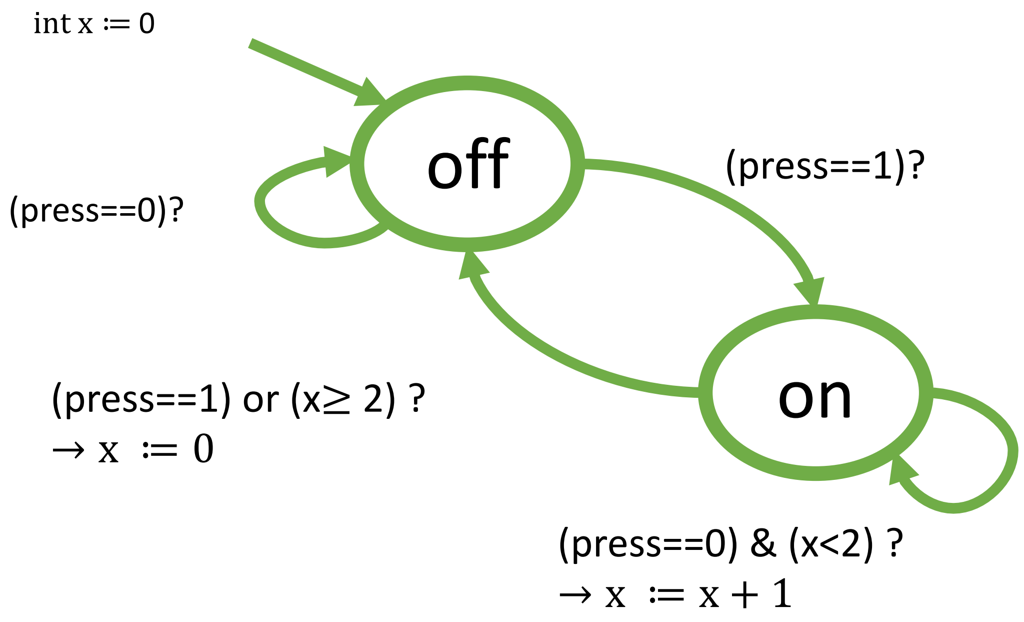


Safety invariants

- An invariant φ is a ***safety invariant*** if
$$\varphi \cap Unsafe = \emptyset$$

- Suppose, $Safe = \{x | 0 \leq x \leq 3\}$, and
$Unsafe = \overline{Safe}$

- Then, we can verify that $0 \leq x \leq 2$ is a ***safety invariant*** for modified switch



Inductive Invariant

- A property φ is an ***inductive invariant*** of a transition system TS if
  - Every initial state satisfies φ
  - If any state s satisfies φ, and $(s, s') \in [\![T]\!]$, then s' satisfies φ

- By definition, if φ is an inductive invariant, then all reachable states of TS satisfy φ, and hence it is also an invariant

Inductive Safety Proof

- Given TS and a property φ, prove that all reachable states of TS satisfy φ
- Base case: Show that all initial states satisfy φ
- Inductive case: assume state s satisfies φ, then show that if $(s, s') \in [\![T]\!]$, then s′ must also satisfy φ

Proving inductive invariants: Example 1

- Consider transition system TS given by
  - $Init: \mathrm{x} \mapsto 0$
  - T: if $(\mathrm{x} < \mathrm{m})$ then $\mathrm{x} \coloneqq \mathrm{x} + 1$ (else x remains unchanged)
- Is $\varphi: (0 \leq \mathrm{x} \leq m)$ an inductive invariant?

Example 1: Is φ:(0≤x≤m) an inductive invariant?

$Init$: $\mathrm{x} \mapsto 0$
T: if $(\mathrm{x} < \mathrm{m})$ then $\mathrm{x} := \mathrm{x} + 1$

- Base case: x is zero, so φ is trivially satisfied
- Inductive case:
  - Pick an arbitrary state (i.e. arbitrary value for state variable x), say $\mathrm{x} \mapsto \mathrm{k}$
  - Now assume k satisfies φ, i.e. $0 \leq \mathrm{k} \leq m$
  - Consider the transition, there are two cases:
    - If $\mathrm{k} < m$, then $\mathrm{x} = \mathrm{k} + 1$ after the transition, and $x \leq m$
    - If $\mathrm{k} = m$, then $\mathrm{x} = \mathrm{k}$ (because guard is not true), which is $\mathrm{x} = m$.
  - In either case, after the update $0 \leq \mathrm{x} \leq m$
  - So φ is an inductive invariant, and the proof is complete

How do we prove safety invariants?

To establish that φ is an invariant of TS:

- Find another property ψ such that
  - $\psi \Rightarrow \varphi$ (i.e. every state satisfying ψ must satisfy φ)
  - ψ is an inductive invariant
    - Show initial states satisfy ψ
    - Assume an arbitrary state s satisfies ψ, consider any state q' such that $(s, s') \in [\![T]\!]$, then prove that s' satisfies ψ

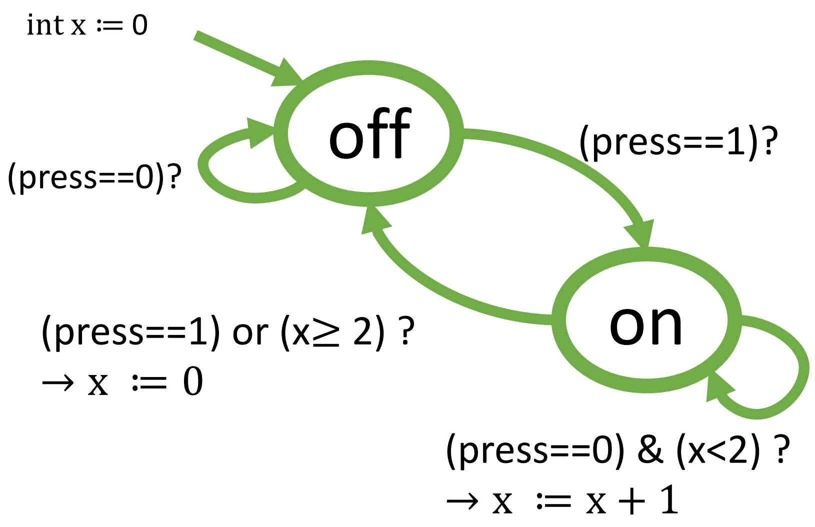
Soundness and Completeness

- Formal system: a set of axioms, a grammar for specifying well-formed formulas, and a set of inference rules for deriving new true formulas from axioms
- Sound: Starting from the axioms and using inference rules of the formal system, we cannot arrive at a formula that is equivalent in meaning to false.
- Complete: Proof system is complete with respect to a property if every formula having that property can be derived using the inference rules
- **Proof rule for proving invariants is sound and complete**:
  - Sound: It is a correct proof technique
  - Complete: If φ is an invariant, there is some stronger inductive invariant ψ satisfying inductive conditions that we can find

Safety Proof for Switch

- φ: $\{\mathrm{x} | 0 \leq \mathrm{x} \leq 2\}$
- Let's try the inductive invariant: ψ: $\big((mode = off) \Rightarrow (\mathrm{x} = 0)\big) \wedge \big((mode = on) \Rightarrow (0 \leq \mathrm{x} \leq 2)\big)$

- $Init$: $\mathrm{x} \mapsto 0, mode \mapsto off$
- Base case: $(off, 0)$ trivially satisfies ψ
- Inductive hypothesis: assume that a state q satisfies ψ
- Inductive step: prove that any q' s.t. $(q, q') \in [\![T]\!]$ satisfies ψ
  - Case I: $q = (off, 0)$
    - q' = $(off, 0)$ [trivial]
    - q' = $(on, 0)$ [satisfies second conjunct in ψ]
  - Case II: $q = (on, n)$
    - $q' = (on, n+1)$ if $n < 2$, this implies that $n + 1 \leq 2$, so q' satisfies ψ
    - $q' = (off, 0)$ otherwise, this again implies that q' satisfies ψ
- So ψ is an inductive invariant
- Further, $\psi \Rightarrow \varphi$ (note that every state satisfying ψ will satisfy φ)
- So φ is an invariant of the TS!



Reachability

- A state q of a transition system is ***reachable*** if there is an execution starting in some initial state that ends in q.
- Algorithm to compute reachable states from a given set of initial states (just Breadth First Search, BFS):

Procedure ComputeReach(TS)
    $Y_0 := [\![Init]\!]$, k:=1;
    While $(Y_k \neq Y_{k-1})$
        Temp := ∅
        ForEach $q \in Y_{k-1}$
                If $((q, q') \in [\![T]\!])$ Temp := Temp $\cup \{q'\}$
        EndForEach
        $Y_k := Y_{k-1} \cup$ Temp, $k := k + 1$
    EndWhile
    Return Y_k
EndProcedure

Proving safety via reachability

If partitioning the states S into a safe/unsafe set.
To get a proof of safety, do reachability computation, and if **ComputeReach**(TS) ∩ $Unsafe$ = ∅, then the TS is safe

Proving that something is an invariant via reachability

- Given TS and a property φ, prove that all reachable states of TS satisfy φ
- **ComputeReach**(TS), it actually gives an inductive definition of reachable states
  - All states specified by I (initial state) are reachable using 0 transitions
  - If a state s is reachable using m transitions, and (s, s') is a transition in $[\![T]\!]$, then s′ is reachable using m+1 transitions
  - Reachable = Reachable using n transitions for some n

Büchi automaton

- 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

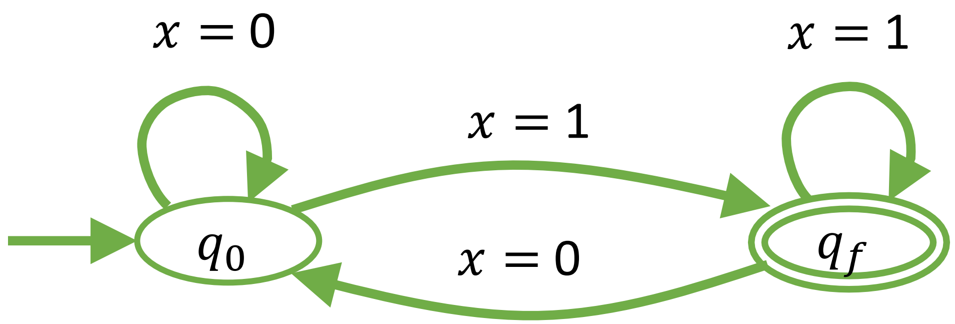
Büchi automaton

- Theoretical result: Every LTL formula φ can be converted to a Büchi monitor/automaton A_φ
- Extension of finite state automata to accept infinite strings
- A Büchi automaton is tuple $\mathrm{B} =< S, I, \Sigma, \delta, F >$
  - S finite set of states (like a TS) –
  - I is a set of initial states (like a TS) –
  - Σ is a finite alphabet (like a TS) –
  - δ is a transition relation (like a TS)
  - F is a set of accepting states
- An infinite sequence of states (a path/trace ρ) is accepted iff it contains accepting states (from F) infinitely often

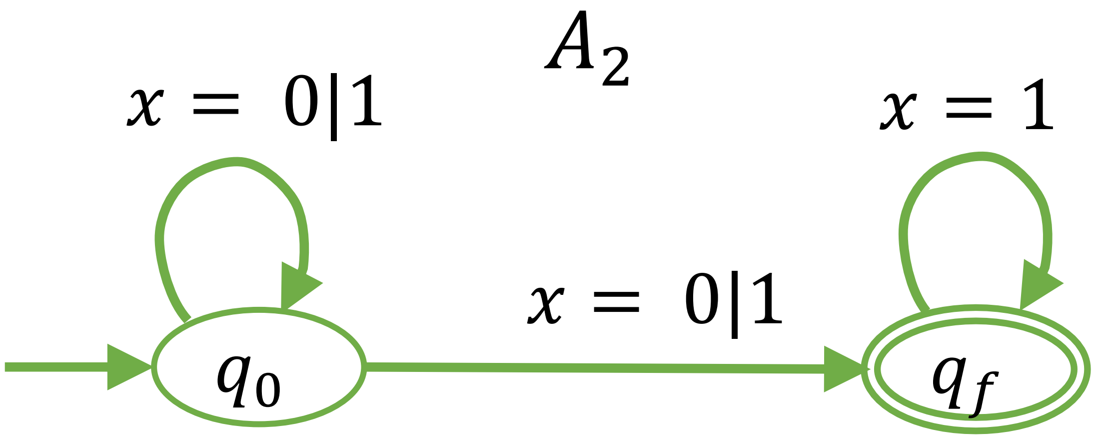
Example: What is the language of A_1?

LTL formula $\mathbf{GF}(x = 1)$

A_1



Büchi automaton Example

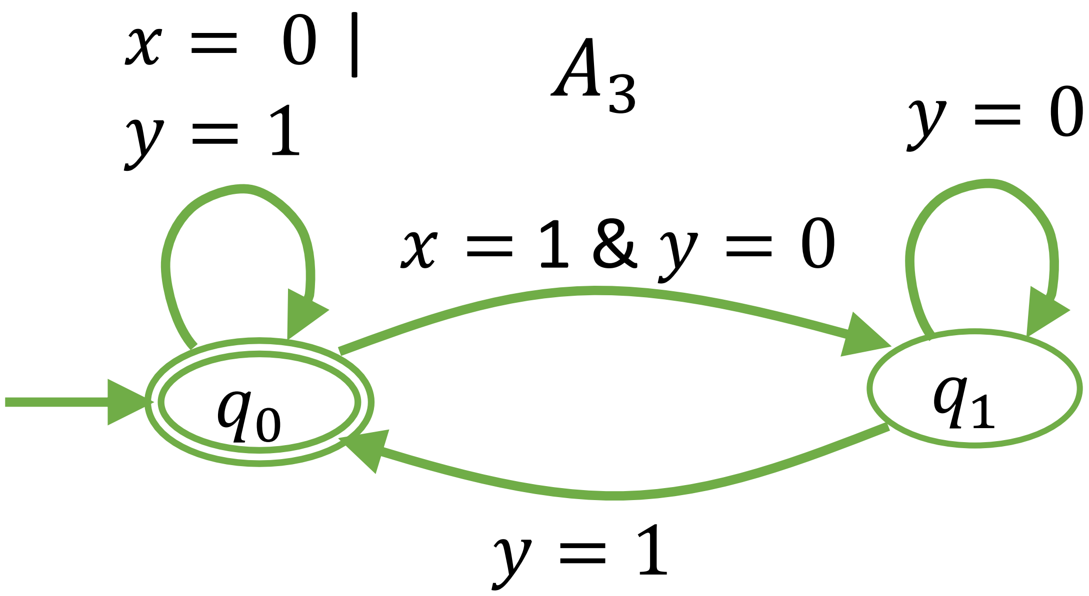


- $S: \{q_0, q_f\}, \Sigma: \{0,1\}, F: \{q_f\}$
- Transitions: (as shown)

- 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 $\mathbf{FG}(x = 1)$

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

Büchi automaton Example 3

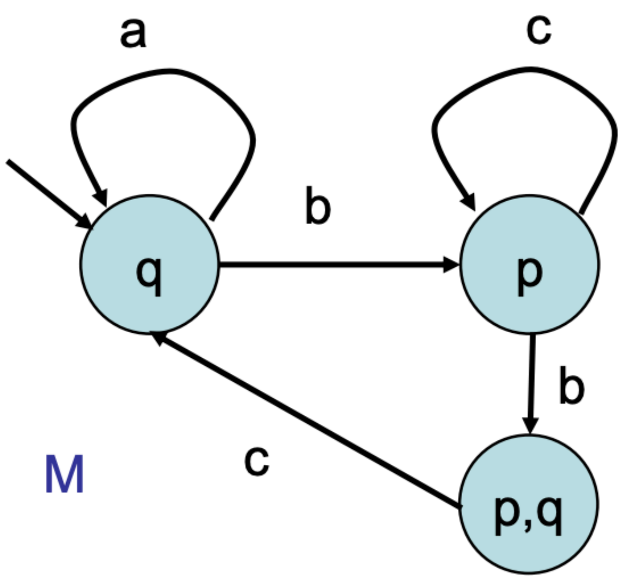


- $S: \{q_0, q_1\}, \Sigma: \{0,1\}, F: \{q_0\}$
- Transitions: (as shown)

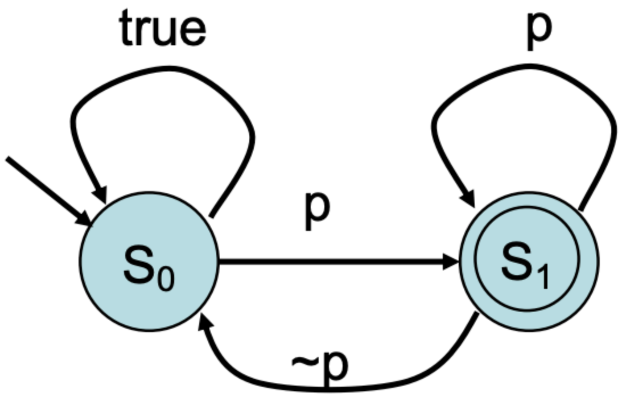
- What is the language of A_3?
  - LTL formula:
    $\mathbf{G}\big((x = 1) \Rightarrow \mathbf{F}(y = 1)\big)$
  - I.e. always when $(x = 1)$, in some future step, $(y = 1)$
  - In other words, $(x = 1)$ must be followed by $(y = 1)$

LTL Model Checking

- TS M: input set A = {a,b,c} and AP={p,q}
- Formula φ = G F p
- Traces of M = infinite label sequences (e.g. σ_1=({q},{p},{p,q})* and σ_2={q}*)



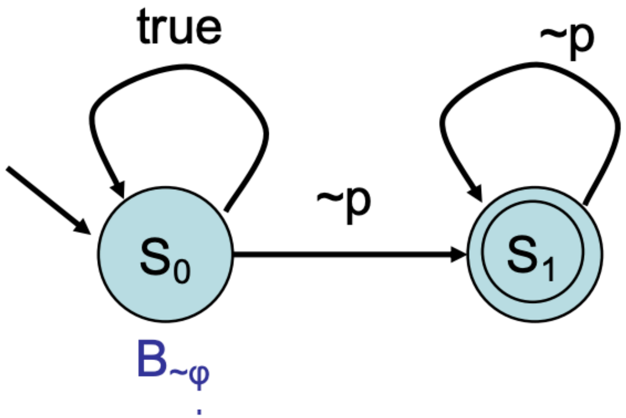
M



B_φ, φ = GF p

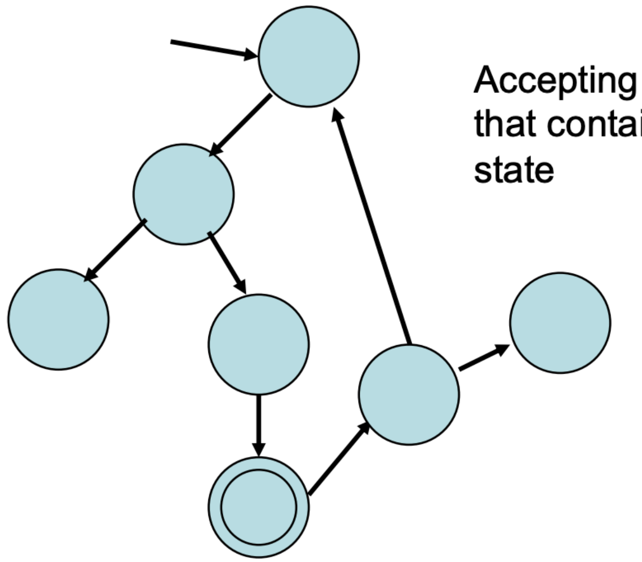
LTL Model Checking

- B_{φ} accepts exactly those traces that satisfy φ
- $B_{\sim\varphi}$ accepts exactly those traces that falsify φ
- $\sim\varphi = \sim(GFp) = F\sim(Fp) = F(G\sim p)$



LTL Model Checking

- If TS generates a trace that is accepted by $B_{\sim\varphi}$, this means, by construction, that the trace violates φ, and so that the TS is incorrect (relative to φ)



Accepting trace = cycle that contains an accepting state

CTL

Computation Tree Logic

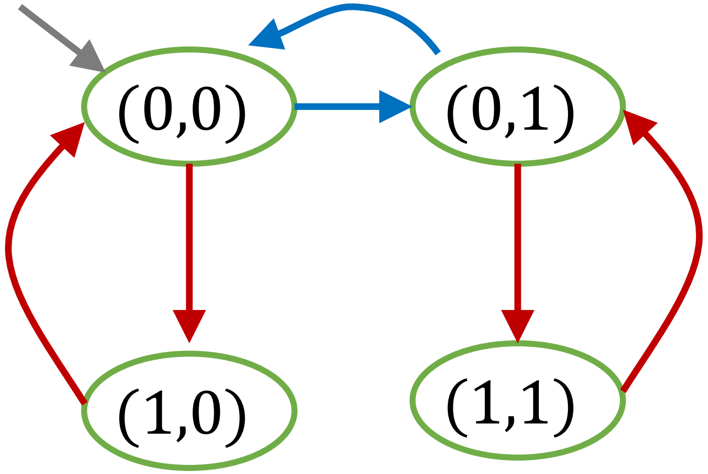
- LTL was a linear-time logic where we reason about traces
- CTL is a logic where we reason over the tree of executions generated by a program, also known as the *computation tree*
- We care about CTL because:
  - There are some properties that cannot be expressed in LTL, but can be expressed in CTL: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

Computation Tree

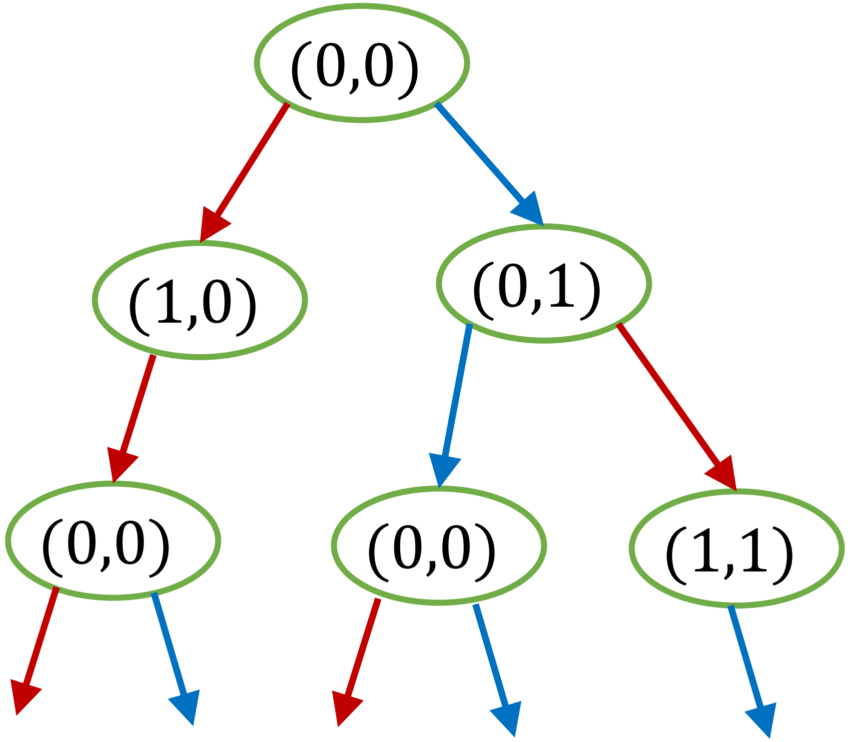
nat x := 0; bool y:= 0

A: x := (x + 1) mod 2
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		
$\varphi ::=$	$p \mid \neg\varphi \mid \varphi \wedge \varphi \quad \mid$	Prop. in AP, negation, conjunction
	$\mathbf{EX}\varphi \quad \mid$	**E**xists Ne**X**t Step
	$\mathbf{EF}\varphi \quad \mid$	**E**xists a **F**uture Step
	$\mathbf{EG}\varphi \quad \mid$	**E**xists an execution where **G**lobally in all steps
	$\mathbf{E}\ \varphi\ \mathbf{U}\ \varphi \quad \mid$	**E**xists an execution where in all steps **U**ntil in some step
	$\mathbf{AX}\varphi \quad \mid$	In **A**ll Ne**X**t Steps
	$\mathbf{AF}\varphi \quad \mid$	In **A**ll possible future paths, there is a future step
	$\mathbf{AG}\varphi \quad \mid$	In **A**ll possible future paths, **G**lobally in all steps
	$\mathbf{A}\ \varphi\ \mathbf{U}\ \varphi \quad \mid$	In **A**ll possible future executions, in all steps **U**ntil in some step

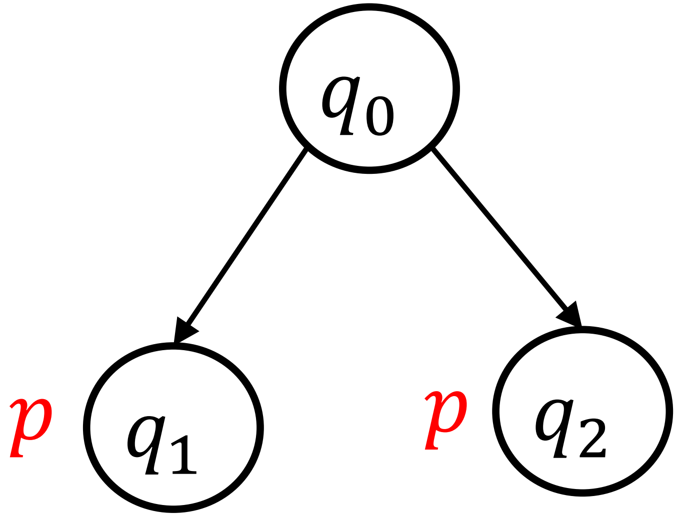
CTL semantics

- *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:

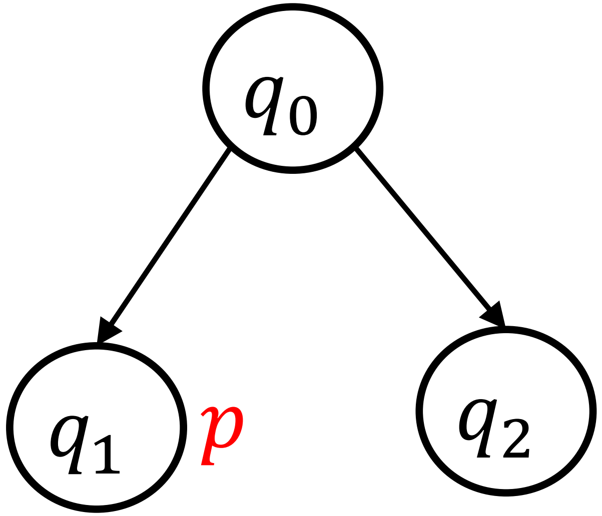
$$\mathbf{A}\,\mathbf{F}\;\; p$$

For **A**ll executions — Eventually/In Some **F**uture step

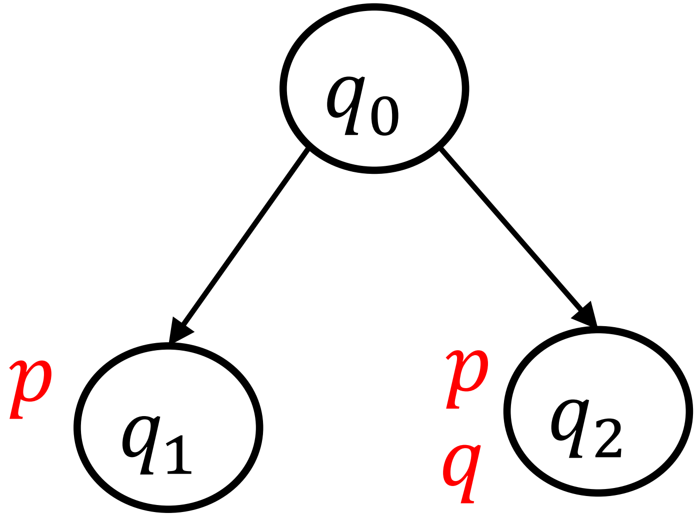
CTL Semantics through examples



$\mathbf{AX}\, p$

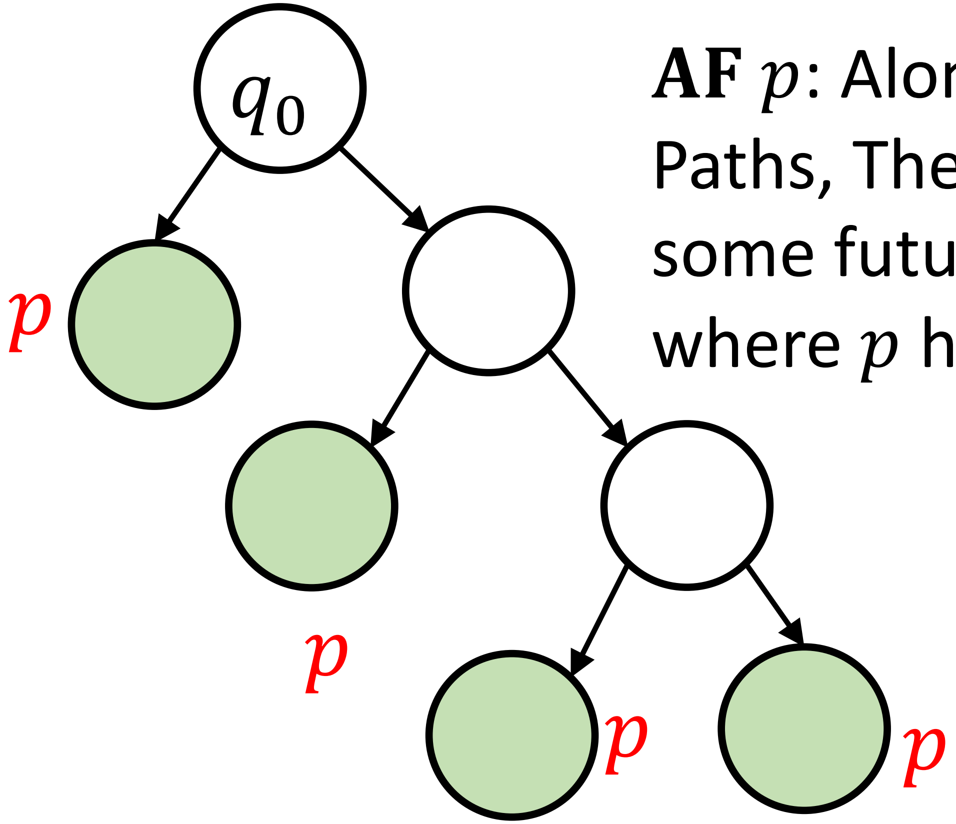


$\mathbf{EX}\, p$

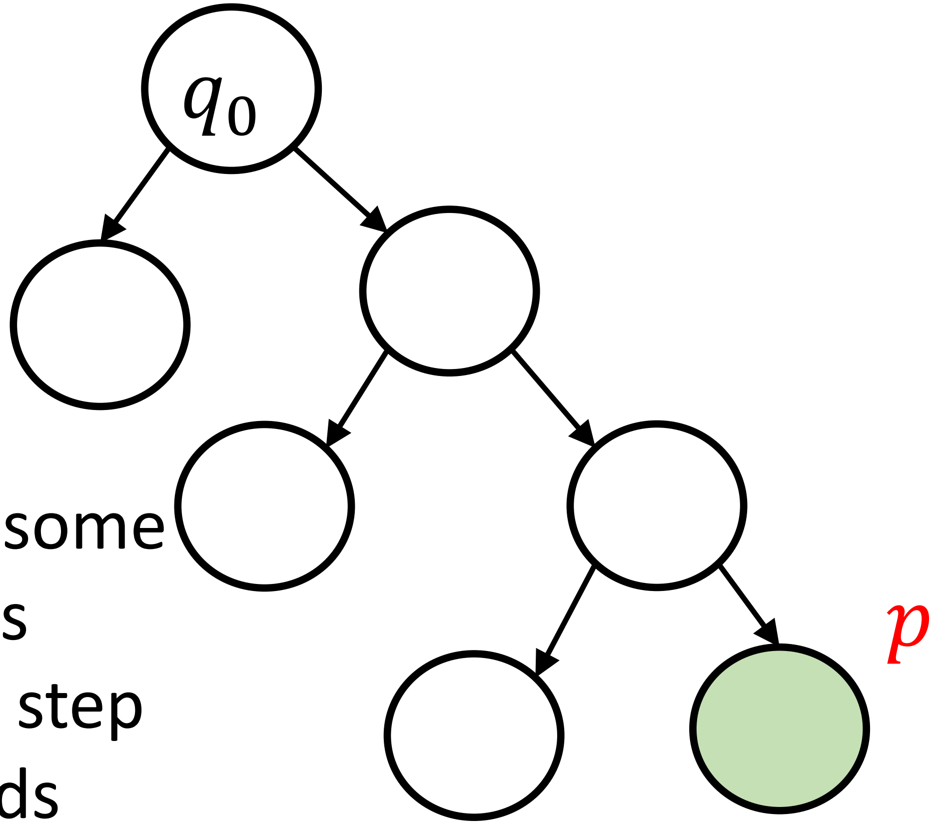


$\mathbf{AX}\, p \wedge \mathbf{EX}\, q$

CTL semantics through examples

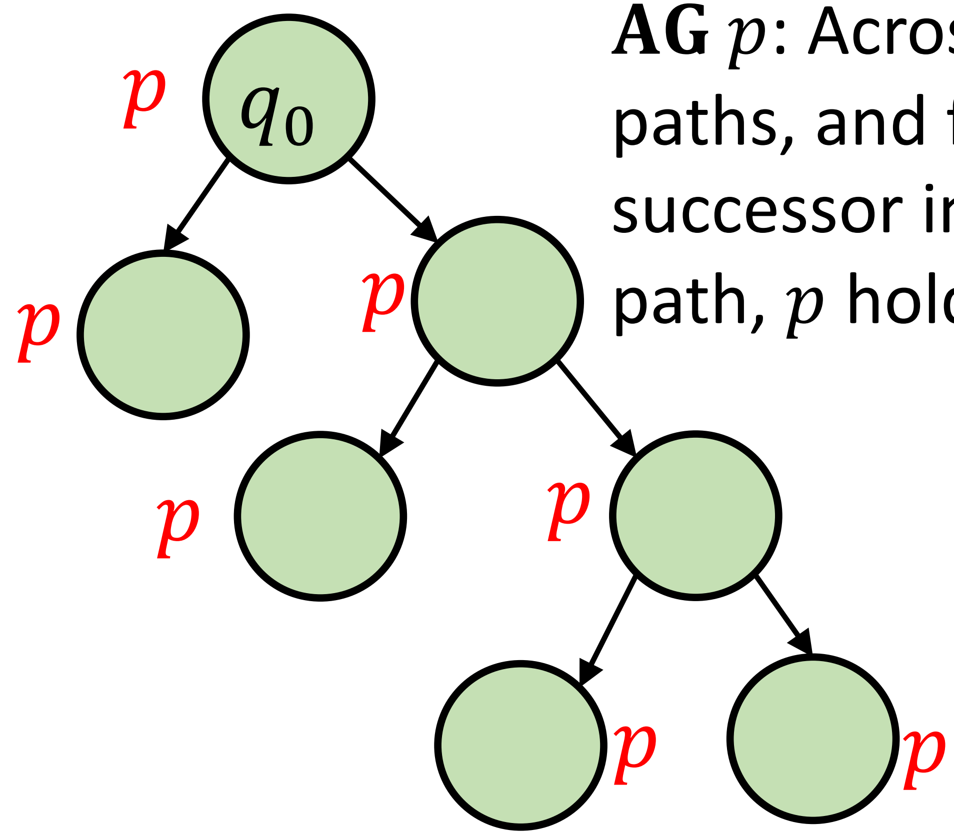


AF p: Along all Paths, There is some future step where p holds

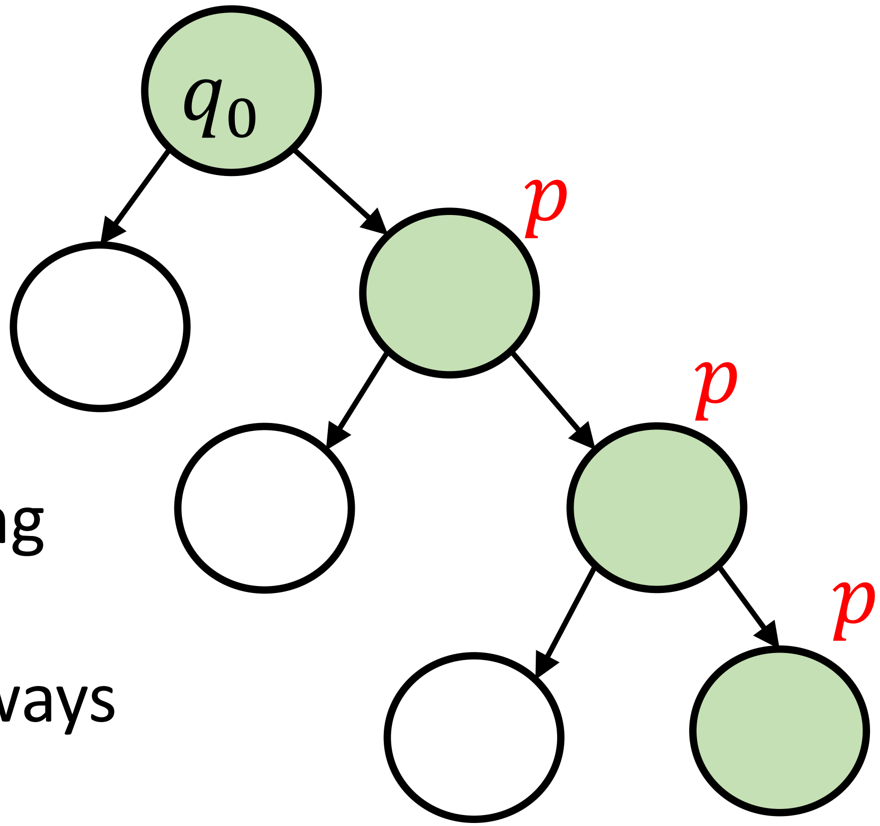


EF p: Along some path, there is some future step where p holds

CTL semantics through examples



$\mathbf{AG}\ p$: Across all paths, and for every successor in the path, p holds



$\mathbf{EG}\ p$: 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
- 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.

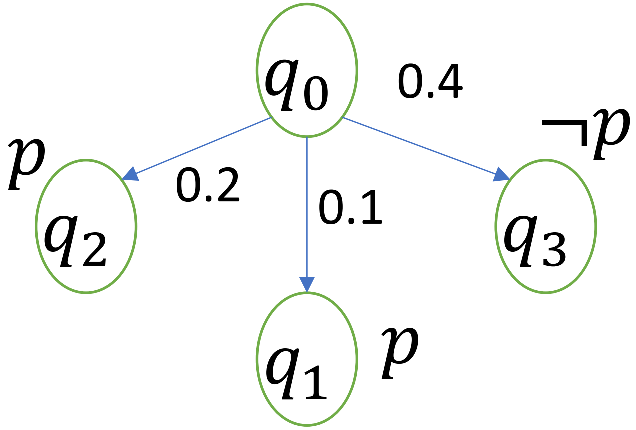
Probabilistic CTL

Syntax of PCTL		
φ ::= (State)	$p \mid \neg\varphi \mid \varphi \land \varphi$ |	Prop. in AP, negation, conjunction
	$P_{\sim\lambda}(\psi)$ |	$\sim \in \{<, \leq, >, \geq\}$, $\lambda \in [0,1]$: Probability of ψ being true
ψ ::= (Path)	$\mathbf{X}\varphi$ |	NeXt Time
	$\varphi\ \mathbf{U}^{\leq k}\varphi$ |	Bounded **U**ntil (up to k steps)
	$\varphi\ \mathbf{U}\ \varphi$	**U**ntil (Recall $\mathbf{F}\varphi = true\ \mathbf{U}\ \varphi$, 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:
  - $Prob(q, \mathbf{X}\varphi)$: $\sum_{q' \vDash \varphi} P(q, q')$
  - 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 $Prob\big(q, \alpha \mathbf{U}^{\leq k} \beta\big)$:
  - 1 if $q \vDash \beta$
  - 0 if $q \nvDash \alpha$ or $q \nvDash \beta$ and $k = 0$
  - $\sum P(q, q').Prob(q', \alpha\, U^{k-1} \beta)$ for $k > 0$, otherwise

PCTL

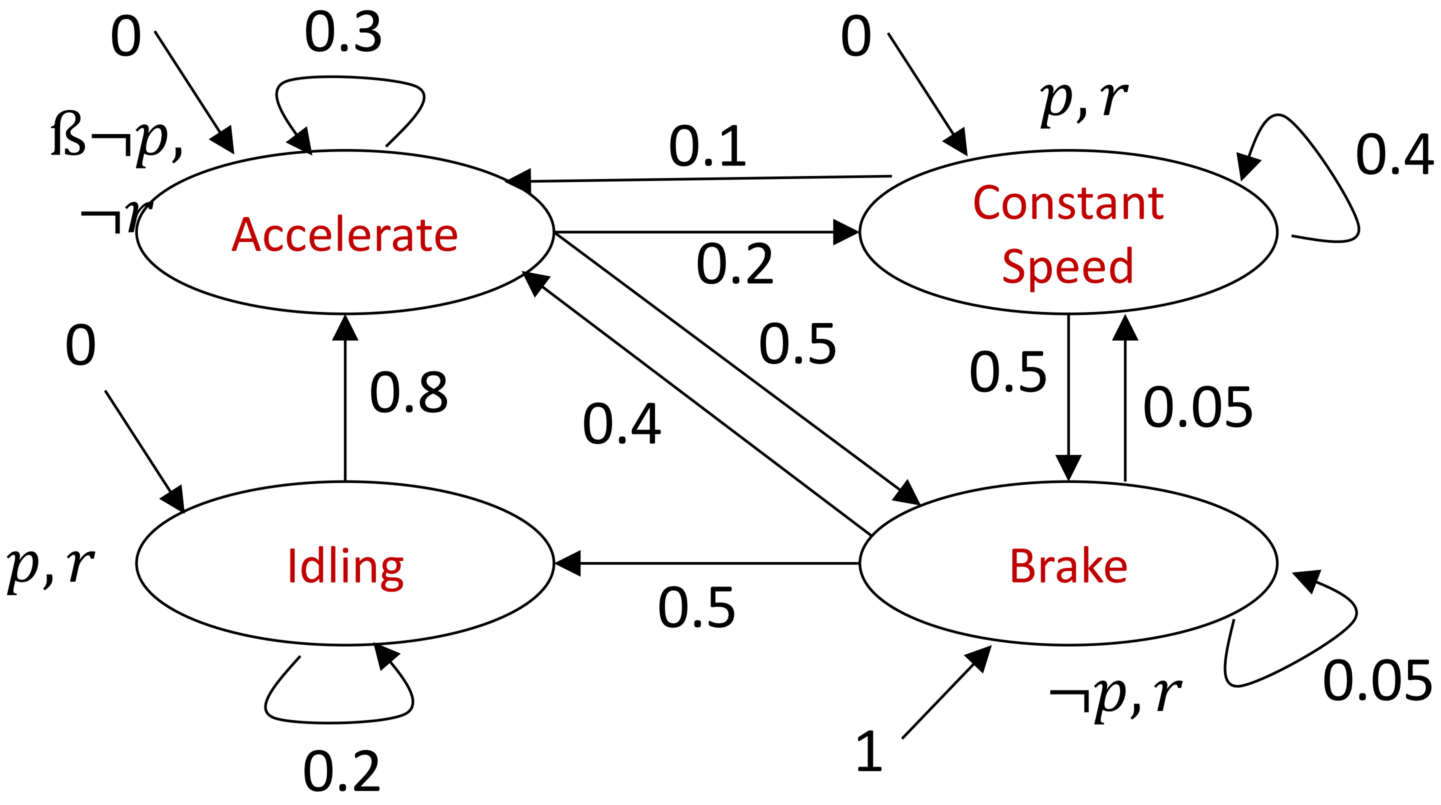
- Does this formula $P_{\geq 0.5}(\mathbf{X}p)$ hold in state Brake?
  - Yes
- Value of ϵ? $P_{\geq \epsilon}(\mathbf{F}^{\leq 2}r)$ in state Accel
  - Compute $Prob(q, \mathbf{F}^{\leq 2}r)$ for all q, pick smallest
  - $P(A,B) + P(A,C) + P(A,A,B) + P(A,A,C)$

    = 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

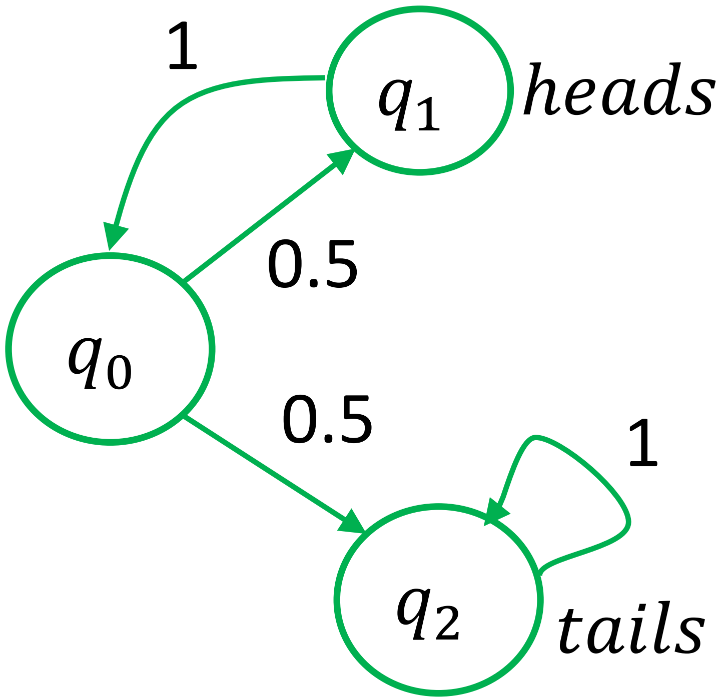
r: Checking cellphone

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? $q_0 q_1 q_0 q_1 \ldots$
- Is the probability of eventually thrown "tails" equal to 1?
  - PCTL: $P_{\geq 1}(\,\mathbf{F}\ tails\,)$
  - Result*: true*
  - Probability of path $q_0 q_1 q_0 q_1 \ldots$ is zero!



How does everything fit together?

- You want to develop a new CPS/IoT system with autonomy
- Analyze its environment: model it as a dynamical system or a stochastic system (e.g. PoMDPs)
- Analyze what models to use for the control algorithms
  - Choices are: Traditional control schemes (PID/MPC), state-machines (synchronous vs. asynchronous based on communication type), AI/planning algorithms, hybrid control algorithms, or combinations of these

Safety is the key!!

- Try to specify the closed-loop system as something you can simulate and see its behaviors
  - Integrative modeling environment such as Simulink (plant models + software models)
  - Specify requirements of how you expect the system to behave (STL, LTL, or your favorite spec. formalism)
    - This step is a DO NOT MISS. It will provide documentation of your intent, and also a machine-checkable artifact
- Test the system a lot, and then test some more
- Apply formal reasoning wherever you can. Proofs are great if you can get them
- Safety doesn’t end at modeling stage; continue reasoning about safety after deployment (through monitoring etc.)

- Models of computation
  - Asynchronous, Synchronous, Timed, Hybrid Processes, Dynamical Systems, MDP

MODELING

- Basics of Control
  - PID, MPC, Nonlinear control, Observer design (Kalman filter)
- Basics of Planning
  - Path planning, Reinforcement learning

AUTONOMY

- Specification Languages (LTL, CTL, STL)
- Falsification and Testing, Parameter Synthesis
- Safety Invariants Reachability, Model Checking

SAFETY