Cyber-Physical Systems

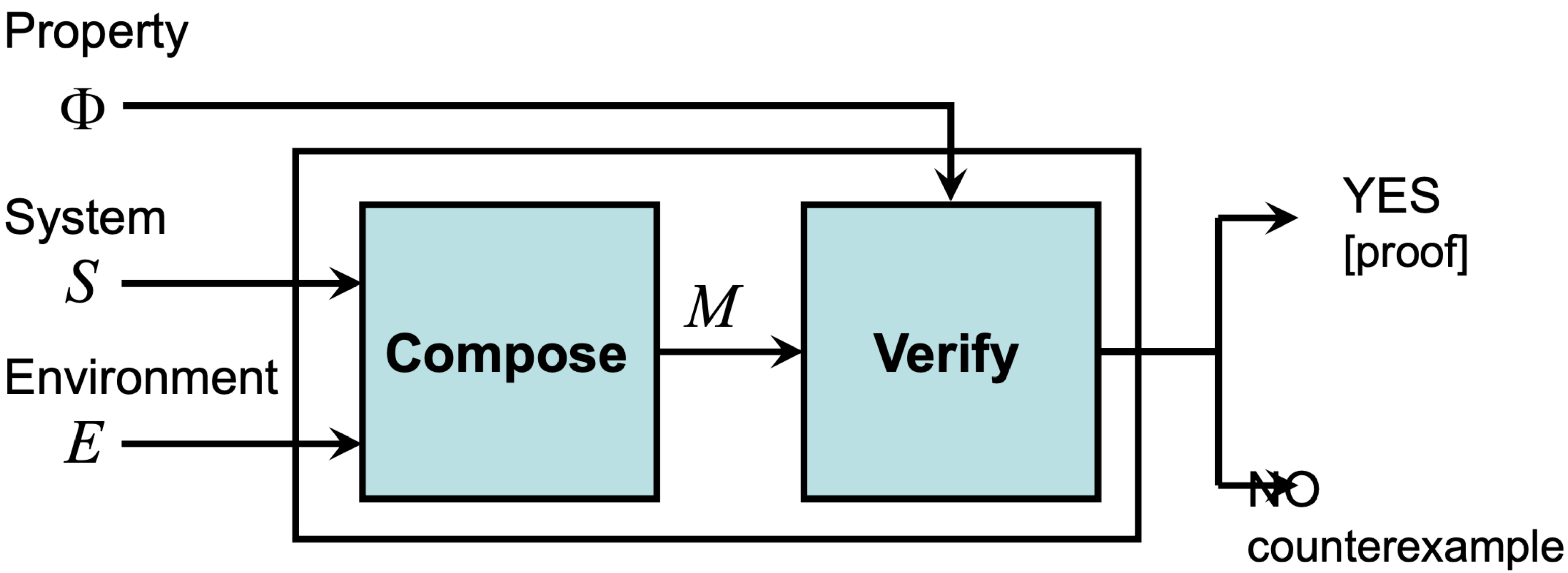
Laura Nenzi

Università degli Studi di Trieste
II Semestre 2018

Lecture 12: Verification

[Many Slides due to J. Deshmukh]

Formal Verification



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

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.

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

Safety Requirements

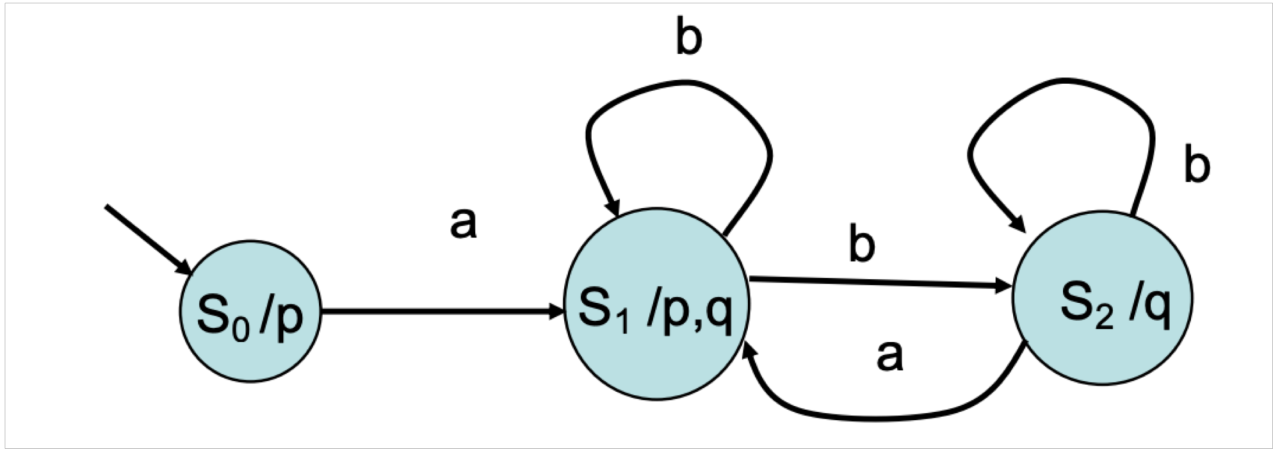
- To verify a safe requirements p on a system M, one simply needs to enumerate all the reachable states and check that they all satisfy p.

- 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.

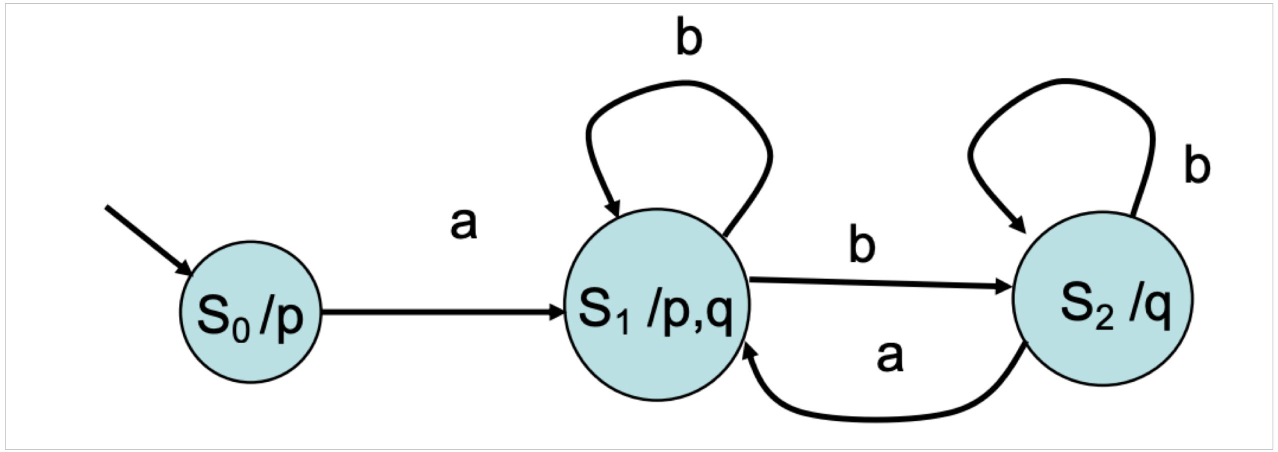
- Safety requirements can be formalized using transition systems

(Label) Transition System

- Transition System is a tuple $< \mathrm{S, I, A}, [\![T]\!], \mathrm{AP, L} >$
  - 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 = $S_0\ S_1S_2S_2S_2$ ...

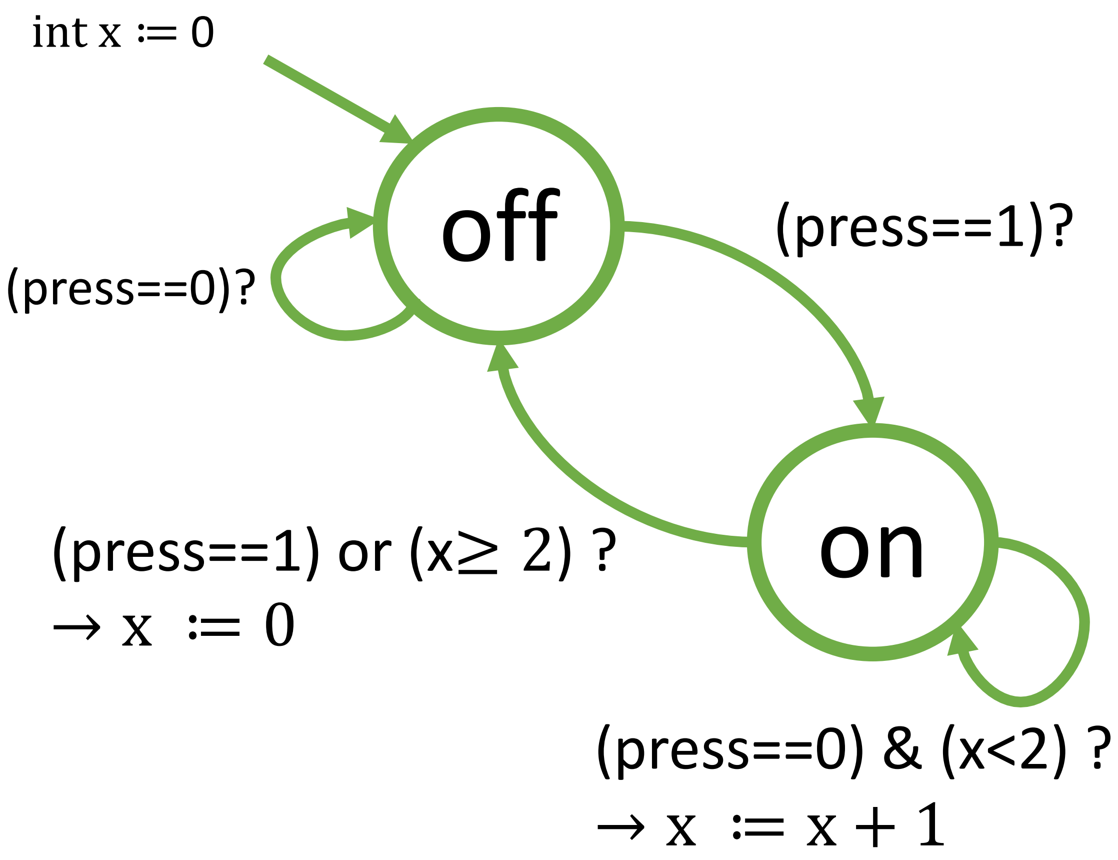
- A **trace** is the corresponding sequence of labels
  e.g. $p\{p,q\}qqq$

- 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

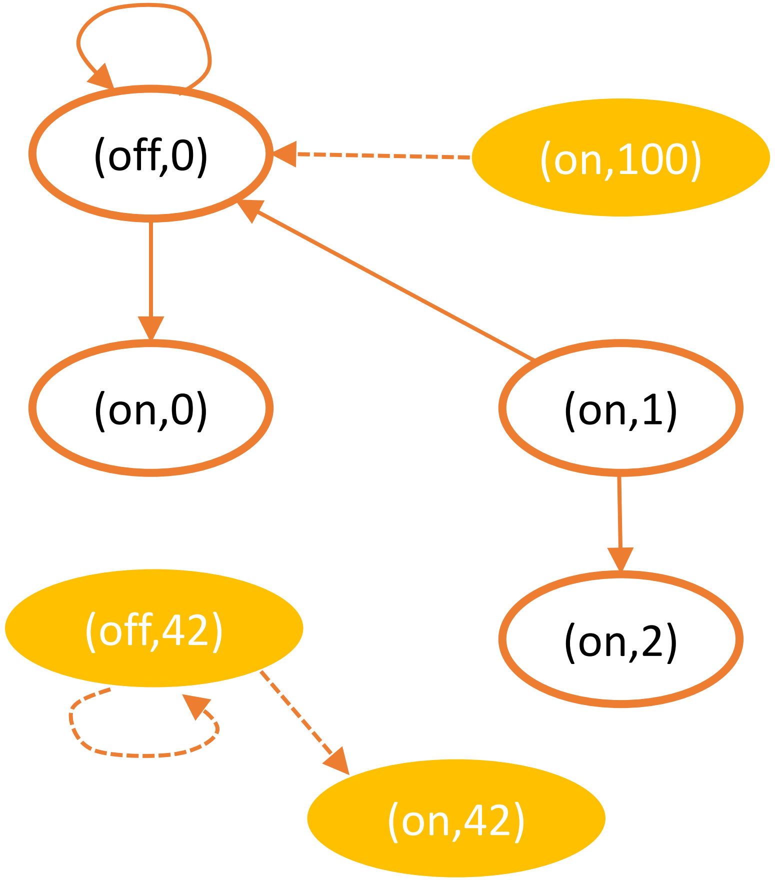


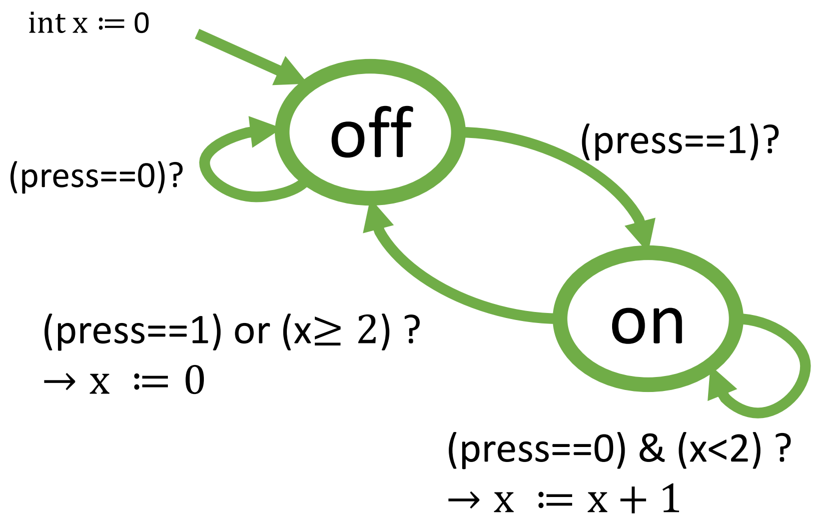
- S = $\{\text{on}, \text{off}\} \times \text{int}$
- I = { off, $x = 0$ }
- $[\![T]\!]$ has an infinite number of transitions:

$$s \to s'$$

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

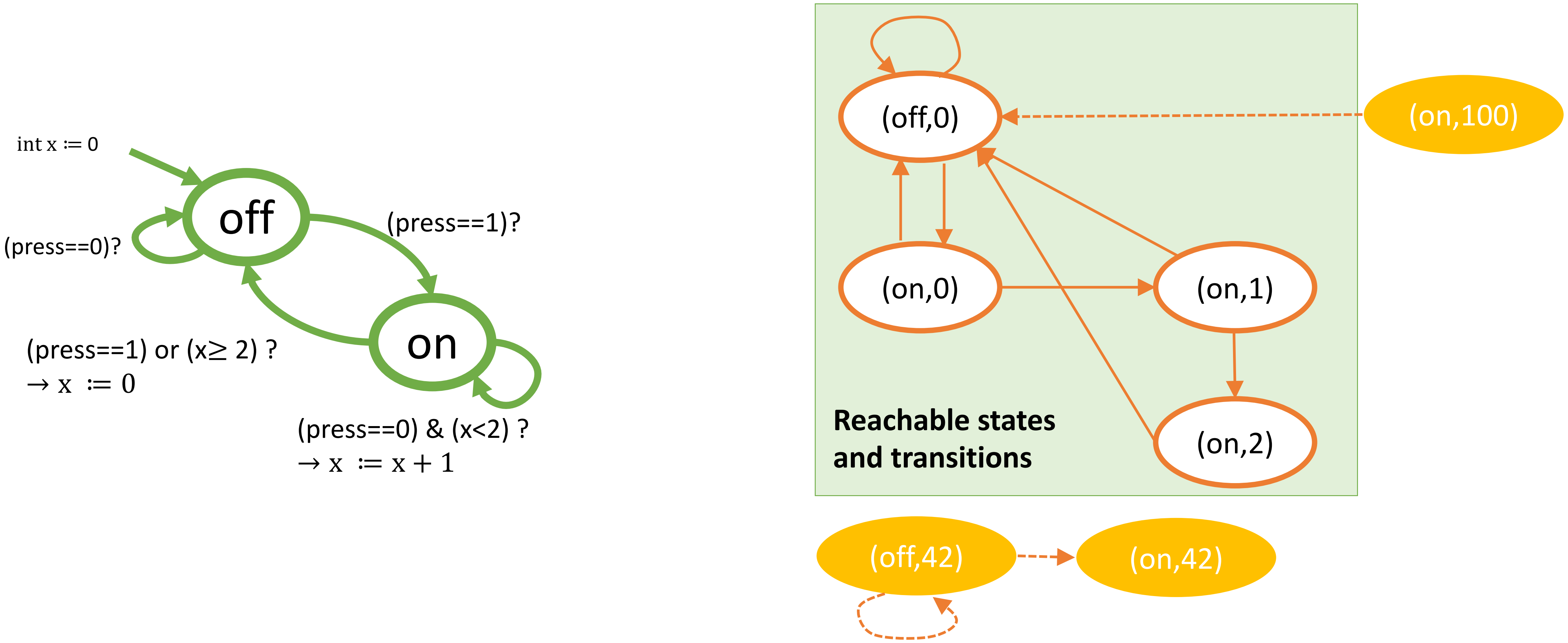
TS describes all possible transitions





- 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 $\{on, off\}\times\mathrm{int}$!

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.

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 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

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.

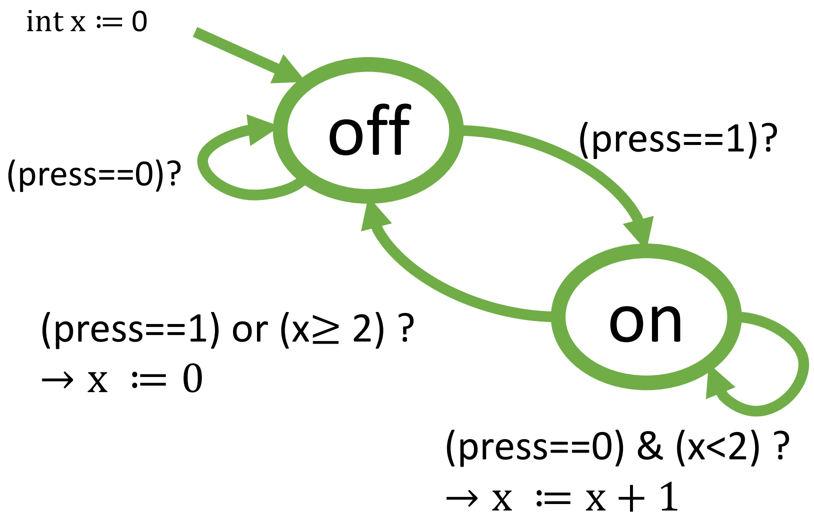
- In other words, to get a proof of safety, do reachability computation, and if **ComputeReach**(TS) ∩ $Unsafe$ = ∅, then the TS is safe

Safety invariants

- An ***invariant*** is a Boolean expression over the state variables of a TS
- A property φ is called an invariant of TS if every reachable state of TS satisfies φ

Examples:

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

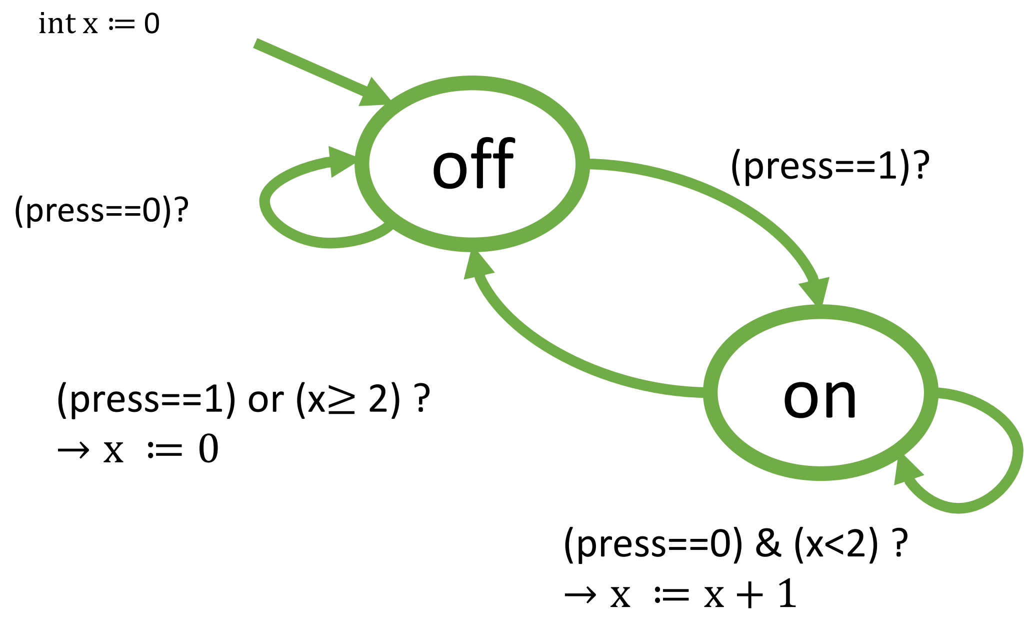


Safety invariants

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

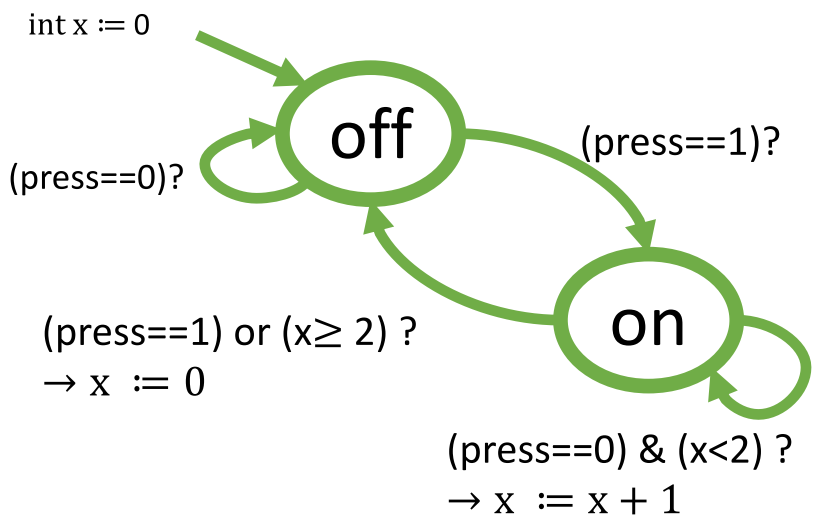
- Suppose, $Safe = \{x | 0 \leq x \leq 3\}$, and $Unsafe = \overline{Safe}$

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



Inductive 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



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

Proving inductive invariants: I

- 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 φ: $(0 \leq \mathrm{x} \leq m)$ an inductive invariant?
- Base case: x is zero, so φ is trivially satisfied

Proving inductive invariants: I

$Init\colon \mathrm{x} \mapsto 0$

T: if $(\mathrm{x} < \mathrm{m})$ then $\mathrm{x} \coloneqq \mathrm{x} + 1$

- 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 $(\mathrm{k} < \mathrm{m}) \Rightarrow (\mathrm{k} + 1) \leq m$
    - If $\mathrm{k} = m$, then $\mathrm{x} = \mathrm{k}$ (because guard is not true), which is $\leq m$.
  - In either case, after the update $0 \leq \mathrm{x} \leq m$
  - So φ is an inductive invariant, and the proof is complete

Proving that something is an invariant

- 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 k transitions, and (s, s') is a transition in $[\![T]\!]$, then s′ is reachable using $k + 1$ transitions
  - Reachable = Reachable using n transitions for some n

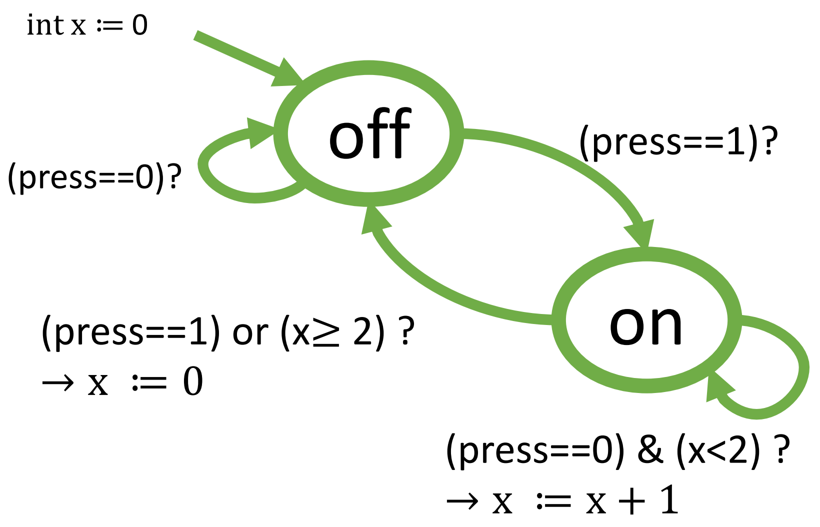
How do we prove safety invariants?

To establish that φ is an invariant of TS:

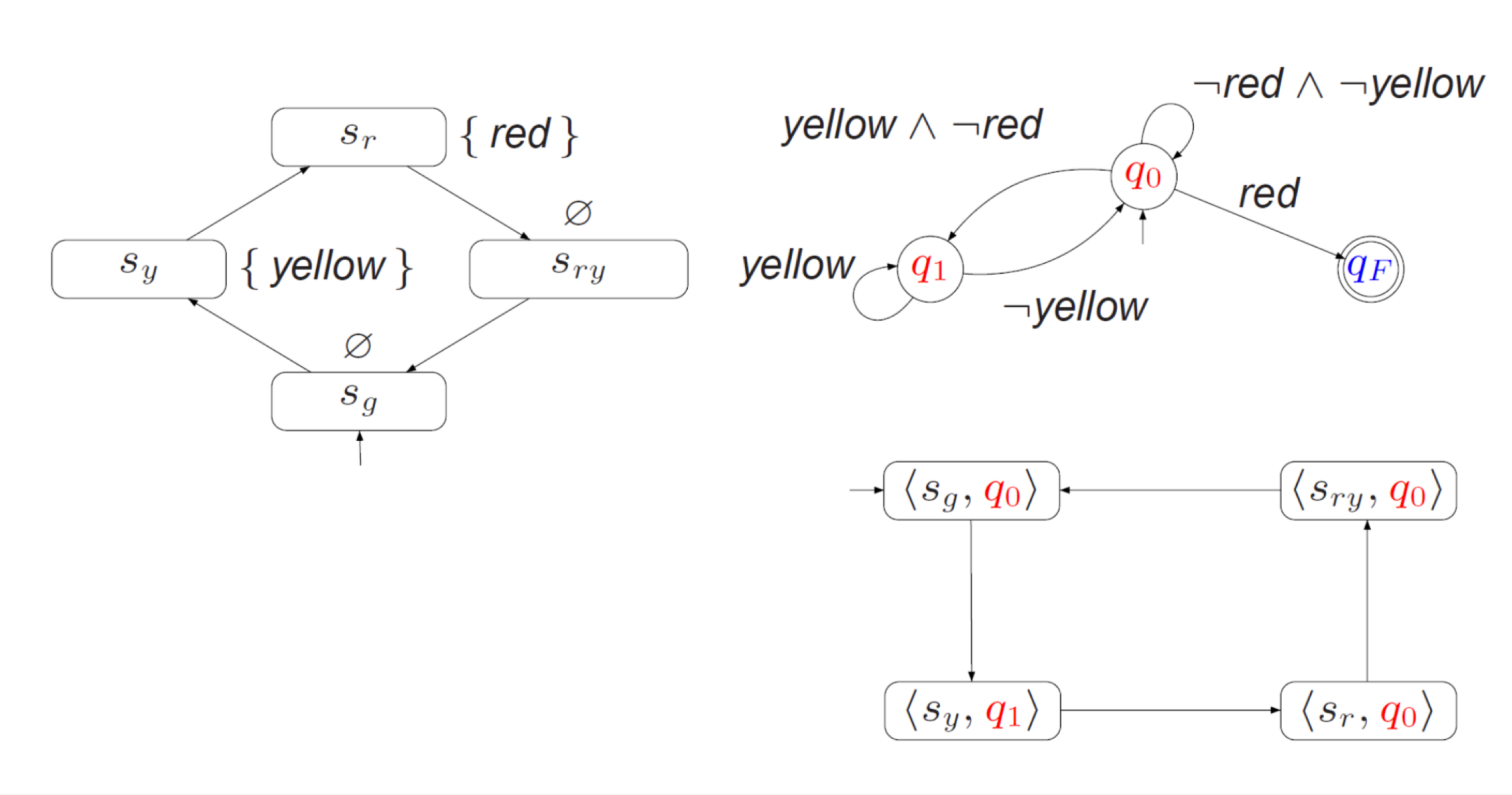
- Find another property ψ such that
  - $\psi \Rightarrow \varphi$ (i.e. every state satisfying ψ must satisfy φ)
  - ψ is an inductive invariant
    - Show initial states satisfy ψ
    - Assume an arbitrary state s satisfies ψ, consider any state q' such that $(s, s') \in [\![T]\!]$, then prove that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

Safety Proof for Switch

- φ: $\{\mathrm{x} | 0 \leq \mathrm{x} \leq 3\}$
- Let's try the inductive invariant: ψ: $\big((mode = off) \Rightarrow (\mathrm{x} = 0)\big) \land \big((mode = on) \Rightarrow (0 \leq \mathrm{x} \leq 2)\big)$
- $Init$: $\mathrm{x} \mapsto 0, mode \mapsto off$
- 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!



Synchronous Product

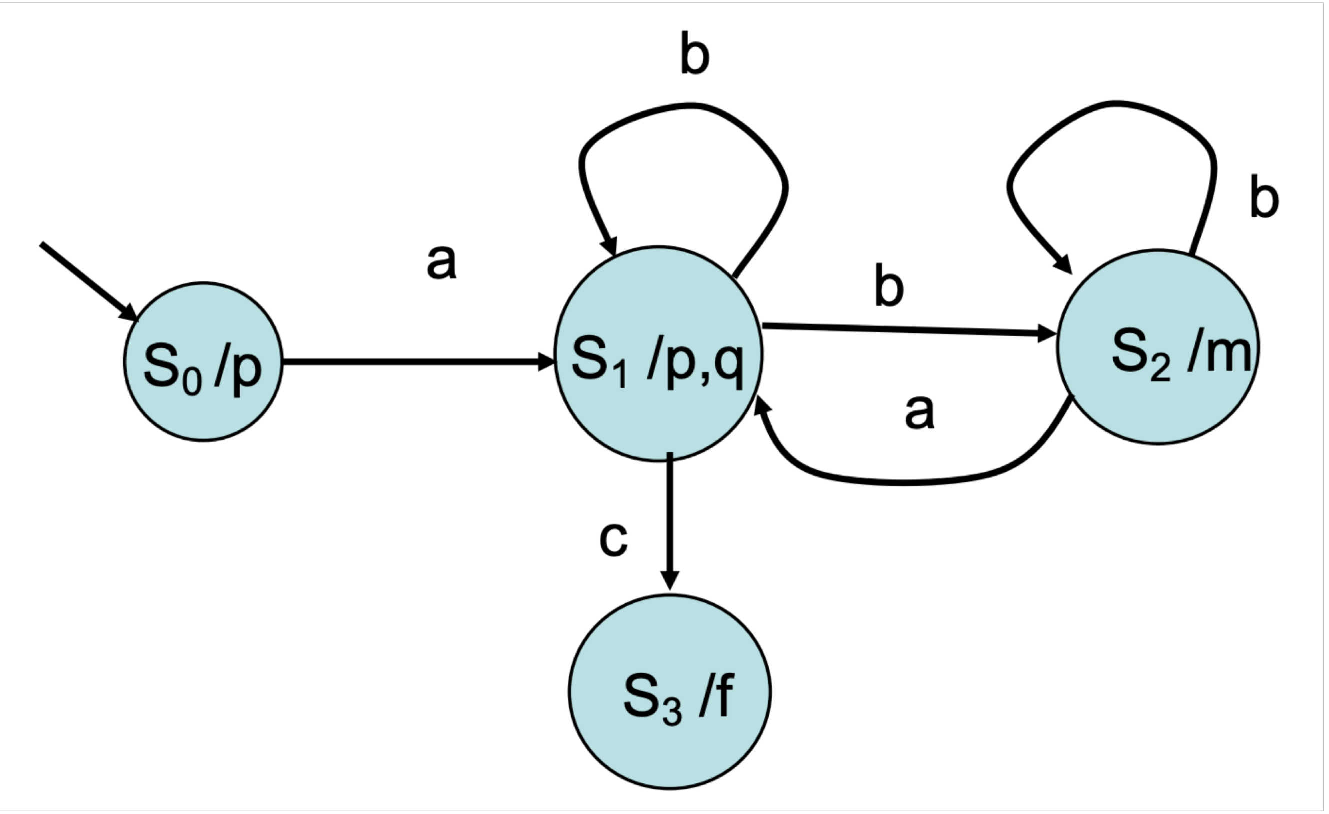


checking of safety properties can hence be reduced to checking an invariant in the product

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

Specification in LTL



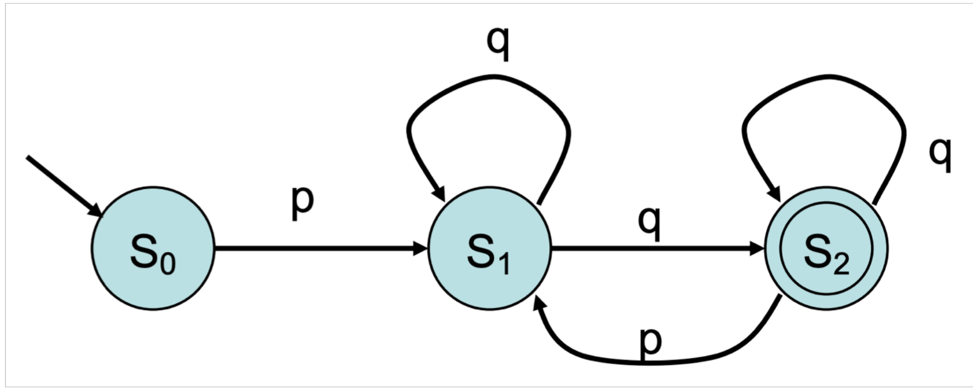
Fm

$G(m \rightarrow Xq)$

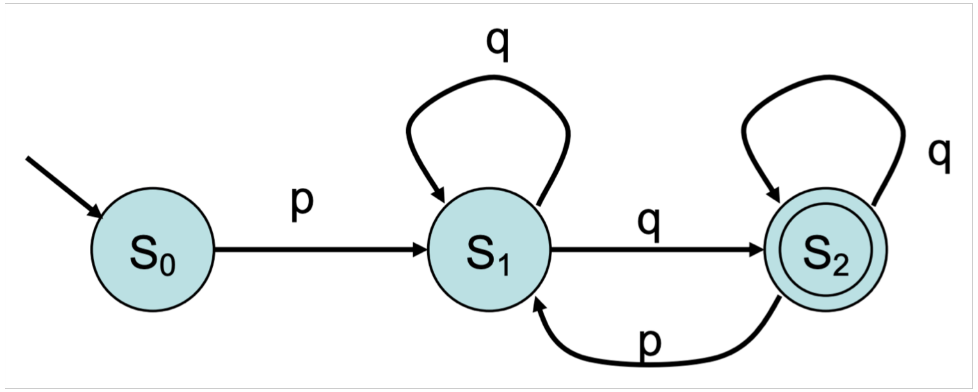
Büchi automaton

- Theoretical result: Every LTL formula φ can be converted to a Büchi monitor/automaton A_{φ}

- It is an automaton which accepts infinite paths
- A Büchi automaton is tuple $\mathrm{B} = < S, I, A, \delta, F >$
  - S finite set of states (like a TS) –
  - I Í S is a set of initial states (like a TS) –
  - A 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) is accepted iff it contains accepting states (from F) infinitely often



Example: accepted words



What words are accepted by this automaton B?

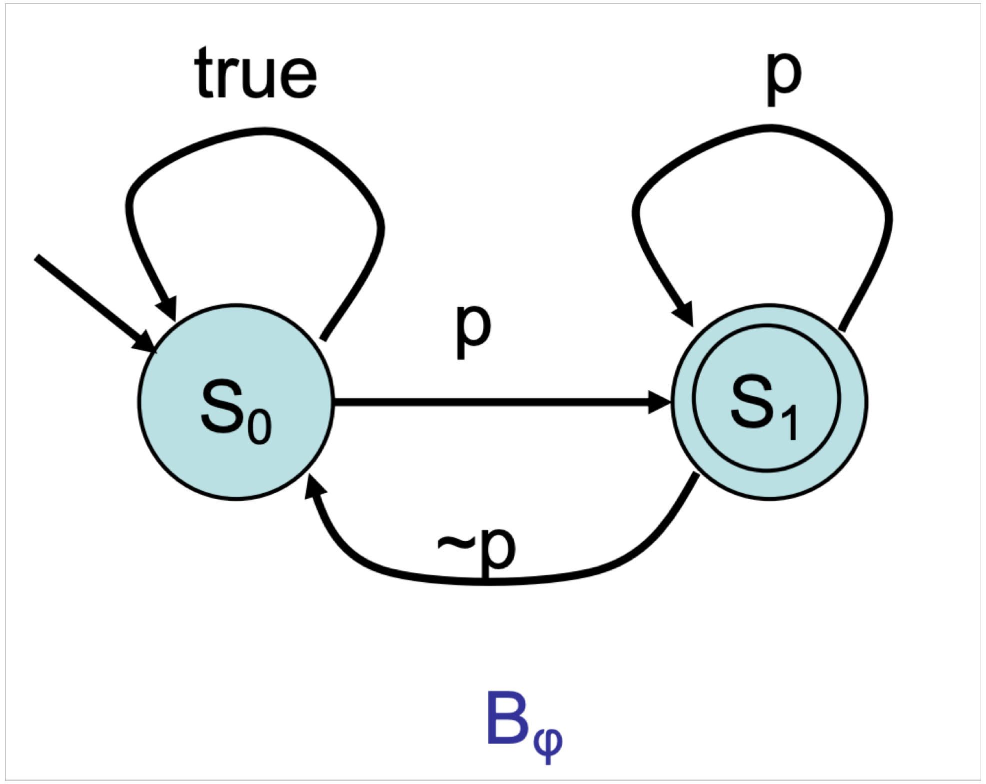
L(B) = pq+(pq+)* L(B) is called the language of B.

It is the set of words for which there exists an accepting run of the automaton.

LTL to Buchi

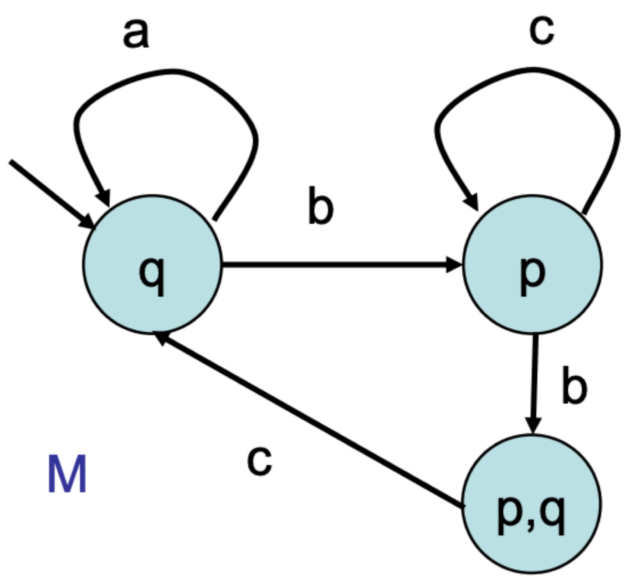
Every LTL formula has a corresponding Buchi automaton that accepts all and only the infinite state traces that satisfy the formula

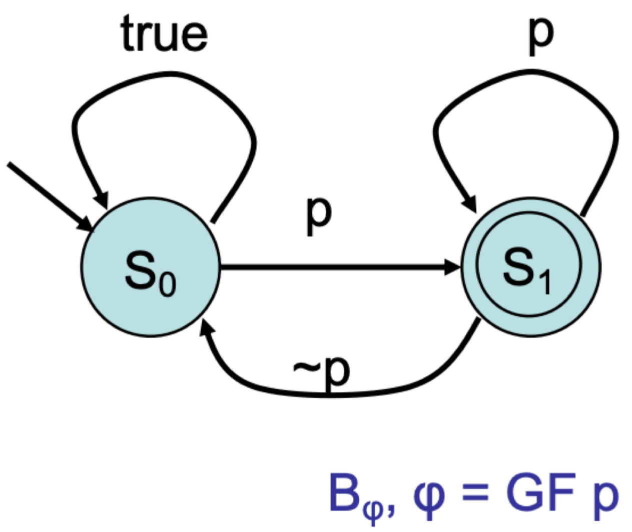
$\phi = G\ F\ p$



LTL Model Checking

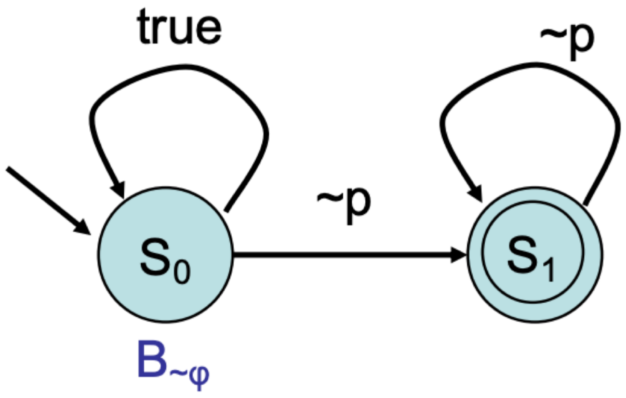
- TS M: input set A = {a,b,c} and AP={p,q}
- Formula φ = G F p
- Traces of M = infinite label sequences (e.g. σ_1=({q},{p},{p,q})* and σ_2={q}*)





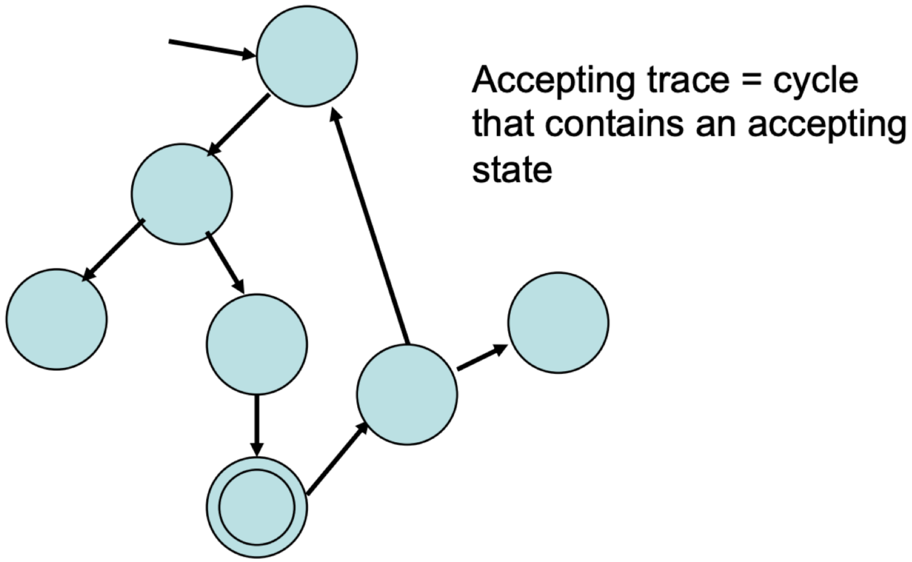
LTL Model Checking

- 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)



Design challenges → Course Concepts

- Modeling protocols, decision layer components
  - Synchronous and Asynchronous processes
  - Understanding system-level safety using synchronous and asynchronous composition
  - Verification using Model Checking, Inductive Invariants
  - Liveness properties with LTL, CTL
  - Model-based and Scenario-based Testing approaches

Design challenges → Course Concepts

- Modeling Controllers, Path planning
  - Timed and Hybrid Processes, Dynamical Systems
  - Markov Decision Processes & Markov Chains
  - Verification using Model Checking, Inductive invariants, Liveness checking with LTL, CTL, STL
  - Testing using Falsification-based approaches
  - Software synthesis using Temporal Logic-based approaches, reinforcement learning

Design Challenges → Course Concepts

- Reasoning about environments, physical processes to be controlled
  - Dynamical systems models, hybrid processes
  - Signal Temporal Logic as a way to express Cyber-Physical systems specifications
  - Testing and Falsification approaches
  - Reasoning about safety

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