Cyber-Physical Systems

Laura Nenzi &
Stefan Schupp

Università degli Studi di Trieste
I Semestre 2023

Lecture : Model Checking

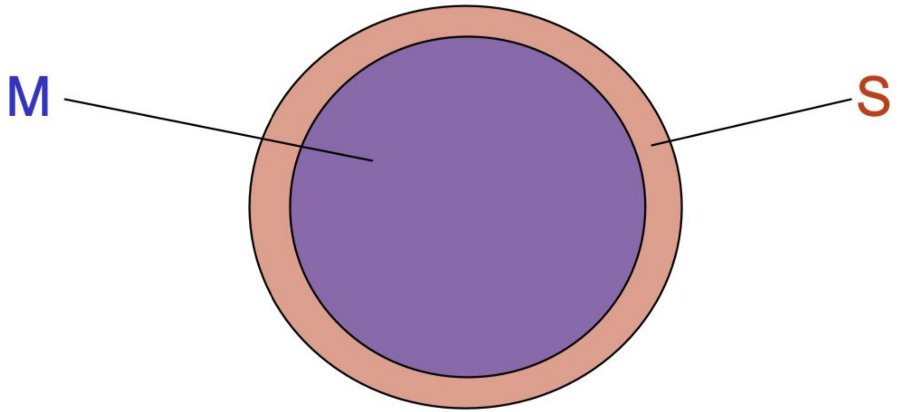
Model Checking

Given a model M and a property specification S, does M satisfy S?

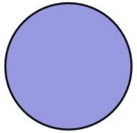
$$M \models S$$

That is the case if the model M does not reveal behaviour violating the specification S

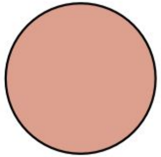
i.e. if every behaviour of M is also behaviour of S



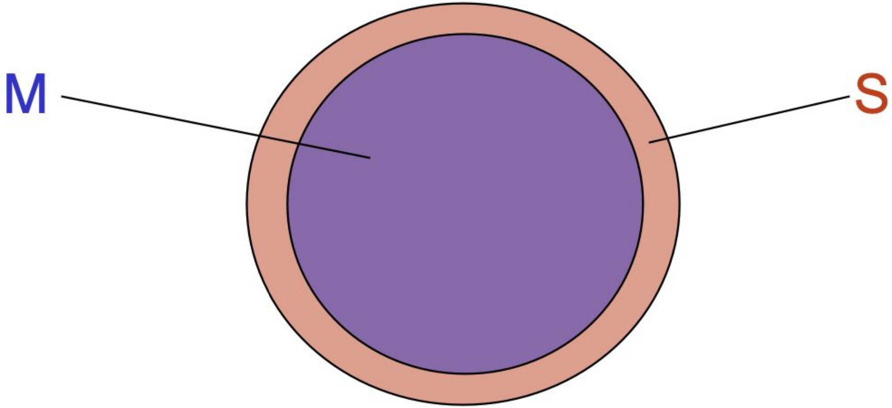
Model Checking



Transition Systems
Mealy and Moore Machines
Communicating FSMs
Extended FSMs



Temporal Logic
ω-automata



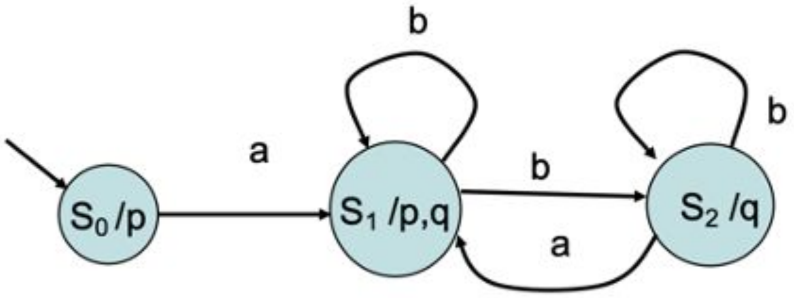
Transition Systems and state

- 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

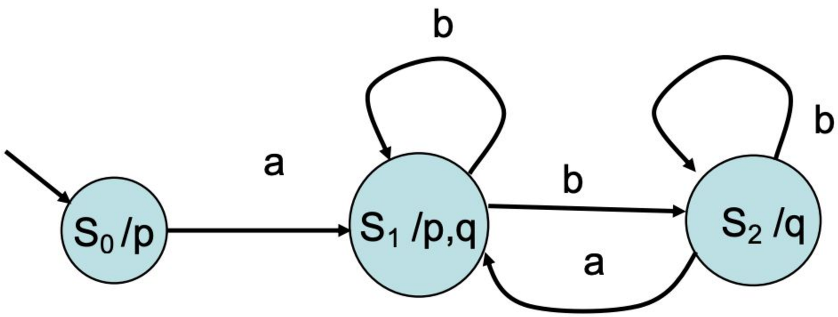
(Label) Transition System

A Transition System TS is a tuple <S, I, Act, [[T]], AP, L>

- S: set of **state**, finite or countable infinite
- I⊆S: set of **initial state**, finite or countable infinite
- Act: Set of **actions**
- [[T]]: is a set of **transition relation** S × Act × S, $s_i \xrightarrow{\alpha_i} s_{i+1}$
- AP: set of **atomic proposition** on S
- L:S $\rightarrow 2^{AP}$ is a **labeling function**, where 2^{AP} is the alphabet

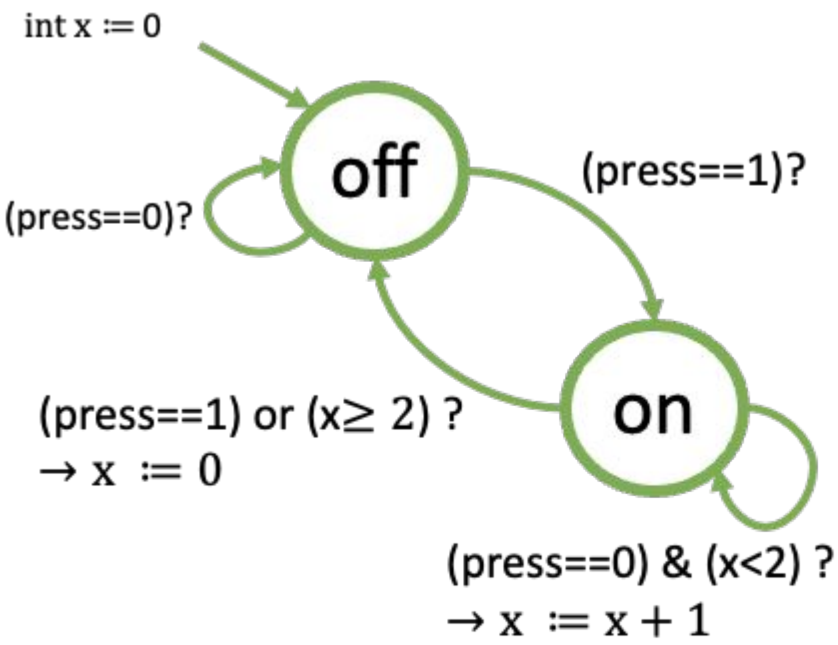


Transition System



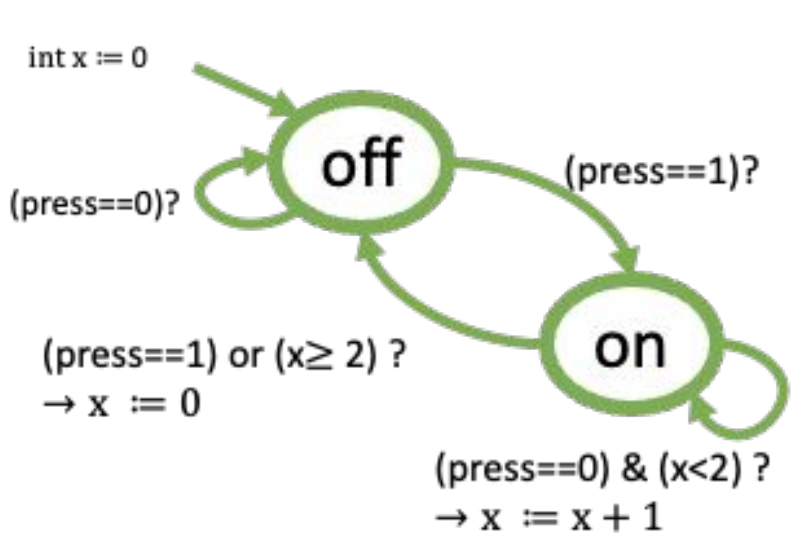
- A **execution** is an (infinite) alternating sequence of states s_i and actions α_i s.t. $S_i \xrightarrow{\alpha_i} s_{i+1}$,
  e.g. $\rho = s_0\ a s_1 b\ s_2 b s_2 b s_2 \ldots$
- A **path** is a sequence of states in the TS, starting from an initial state and either ending in a terminal state, or infinite,
  e.g. $\sigma = s_0\ s_1\ s_2\ s_2\ s_2 \ldots$
- A **trace** is the corresponding sequence of labels over the alphabet
  e.g. $L(s_0)L(s_1)L(s_2)L(s_2)L(s_2)\ldots=p\{p,q\}qqq$

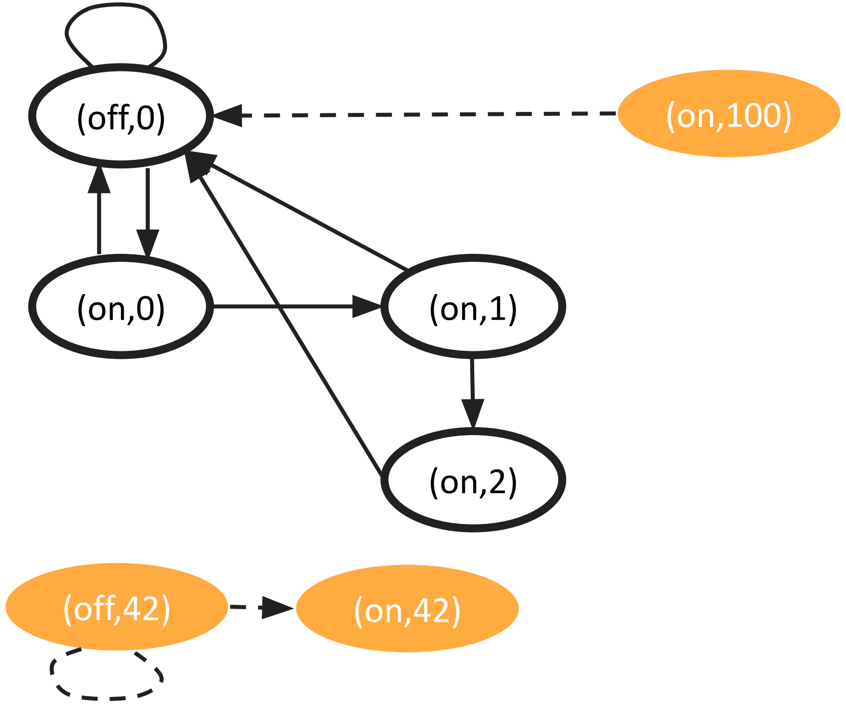
Example of a TS



- $S = \{on, off\} \times int$
- I = { off, x = 0 }
- $[\![T]\!]$ has an infinite number of transitions:
  E.g. $(off, 0) \rightarrow (on, 0)$
  $(on\ 0) \rightarrow (on, 1)$

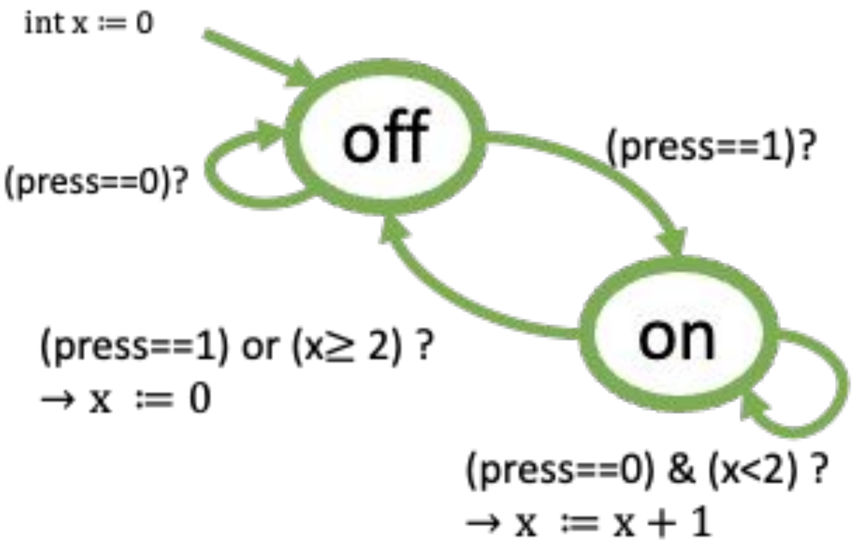
TS describes all possible transitions

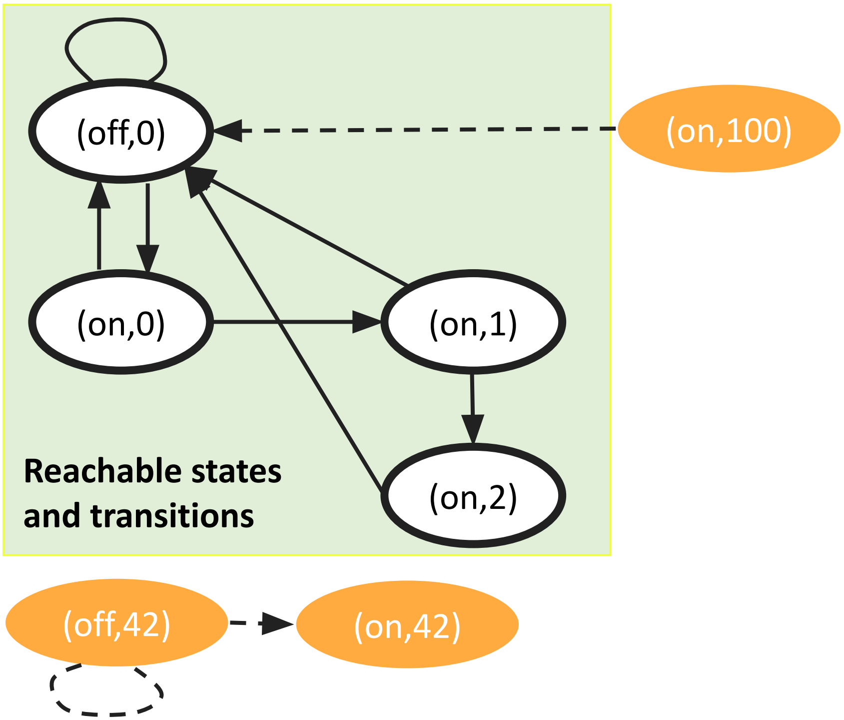




- Transitions indicated as dotted lines can't really happen in the component
- But, the TS will describe then, as the states of the TS are over {on,off}×int!

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
  - $\mathrm{Safe} \subseteq S$, $\mathrm{Unsafe} \subseteq S$, $\mathrm{Safe} \cap \mathrm{Unsafe} = \emptyset$
  - Any finite sequence that ends in a state $q \in \mathrm{Unsafe}$ 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.
- Can we use a monitor to classify infinite behaviors into good or bad?

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

Extension of finite state automata to accept infinite strings

A Büchi automaton is tuple A=<Q,I,δ,Σ,F>:

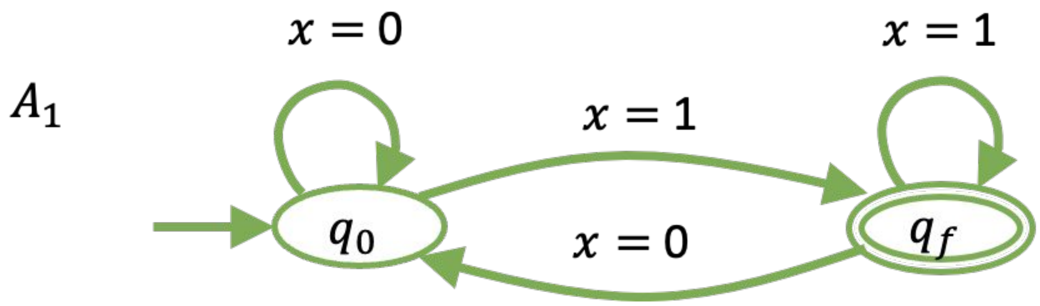
- Q finite set of states (like a TS) –
- Q_0 is a set of initial states (like a TS) –
- Σ is a finite alphabet (like a TS) –
- δ is a transition relation, δ: $S \times \Sigma \rightarrow 2^S$ (like a TS)
- $F \subseteq Q$ is a set of accepting states

An infinite sequence of states (a path/trace ϱ) is accepted iff it contains accepting states (from F) infinitely often

Büchi automaton

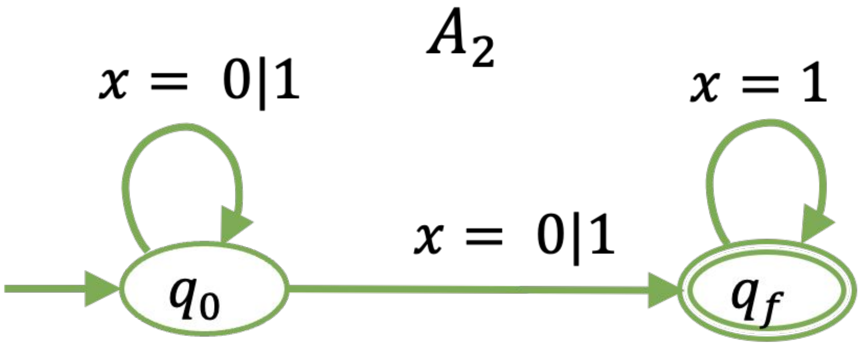
Every LTL formula φ can be converted to a Büchi monitor/automaton A_φ

Example: What is the language of A_1?



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

Büchi automaton Example

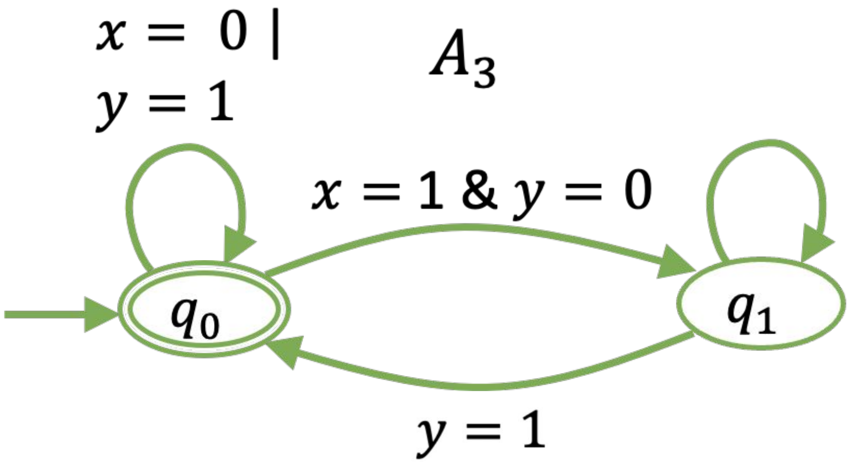


- S: $\{q_0, q_f\}$, Σ: {0,1}, F: $\{q_f\}$
- Transitions: (as shown)

- 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 **FG**(x=1)

Fun fact: there is no deterministic Büchi automaton that accepts this language as it was for Finite Automata

Büchi automaton Example 3



What is the language of A_3?
LTL formula:

G((x=1)⇒**F**(y=1))

- I.e. always when (x=1), in some future step, (y=1)
- In other words, (x=1) must be followed by (y=1)

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

Model Checking Problem

Given a model M, a state *s*, and a property *P*, the model checking problem is to determine if M, *s* |= *P*.

- If *P* is a LTL formula φ, then M, *s* |= φ if and only if σ |= φ for each σ trace of M such that σ[0] = *s*, i.e. if and only if the language of (M, *s*) is contained in the language of φ: L(M, *s*) ⊆ L(φ).
- If *P* is a CTL formula φ, then the satisfaction M, *s* |= φ has the usual meaning.
- Analogously, if φ is given by an automaton A, then M, *s* |= A if and only if L(M, *s*) ⊆ L(A)

MC for LTL

To solve the model checking problem for LTL for a model M_s (fixing the initial state *s*), the idea is:

- negate the LTL formula φ
- covert the LTL formula $\neg\varphi$ into an equivalent Büchi automaton $A_{\neg\varphi}$
- construct the product between the original model and the automaton $A_{\neg\varphi}$, obtaining another Büchi automaton $M_s \otimes A_{\neg\varphi}$
- Apply a graph algorithm (identification of strongly connected components) to the product automaton to test for language emptiness.

MC for LTL

$TS \models \varphi$ if and only if $Traces(TS) \subseteq Words(\varphi)$

if and only if $Traces(TS) \cap \left((2^{AP})^\omega \setminus Words(\varphi)\right) = \varnothing$

if and only if $Traces(TS) \cap \underbrace{Words(\neg\varphi)}_{\mathcal{L}_\omega(\mathcal{A}_{\neg\varphi})} = \varnothing$

if and only if $TS \otimes \mathcal{A}_{\neg\varphi} \models \Diamond\Box \underbrace{\bigwedge_{q \in F} \neg q}_{\neg F}$

LTL model checking is reduced to checking whether an accept state is visited in TS $\otimes$ $A_{\neg\varphi}$ infinitely often

Synchronous Product $\otimes$

For a transition system TS=<S, I, Act, ⟦T⟧, AP, L> and a automata A=<Q,I,δ,2^{AP},F>:

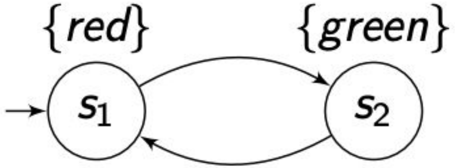
$$TS \otimes A = (S', Act, [\![T]\!]', I', AP', L')$$

- S'=S □ Q
- $I' = \{ \langle s_0 , q \rangle \mid s_0 \in I \wedge \exists\ q_0 \in Q_0 \ .\ q_0 \rightarrow^{L(s0)} q \}$
- Act: Set of **actions**
- AP'=Q
- L'=(<s,q>={q})
- ⟦T⟧':

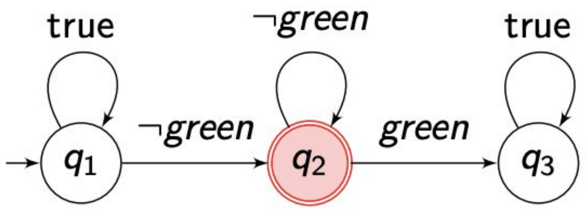
LTL model checking is reduced to checking whether an accept state is visited in $TS \otimes A_{\neg\varphi}$ infinitely often

Synchronous Product

Example: Simple Traffic Light with 2 modes: red and green.
LTL formula to check $\phi = \Box\Diamond green$

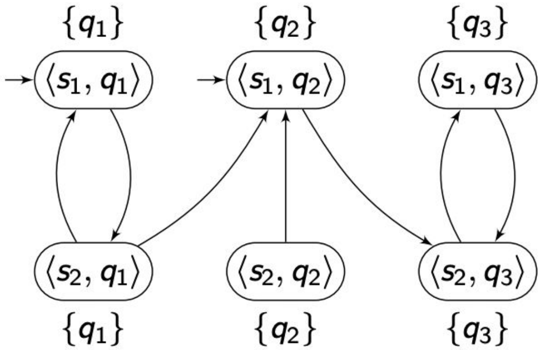


TS T for the traffic light.



NBA A¬φ for $\neg\phi = \Diamond\Box\neg green$.

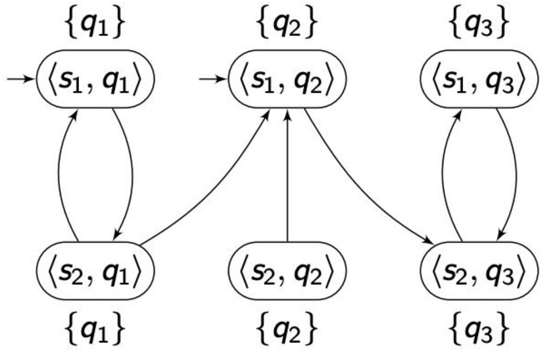
⟹ Blackboard construction of T ⊗ A¬φ.



Synchronous Product

Example: Simple Traffic Light with 2 modes: red and green.

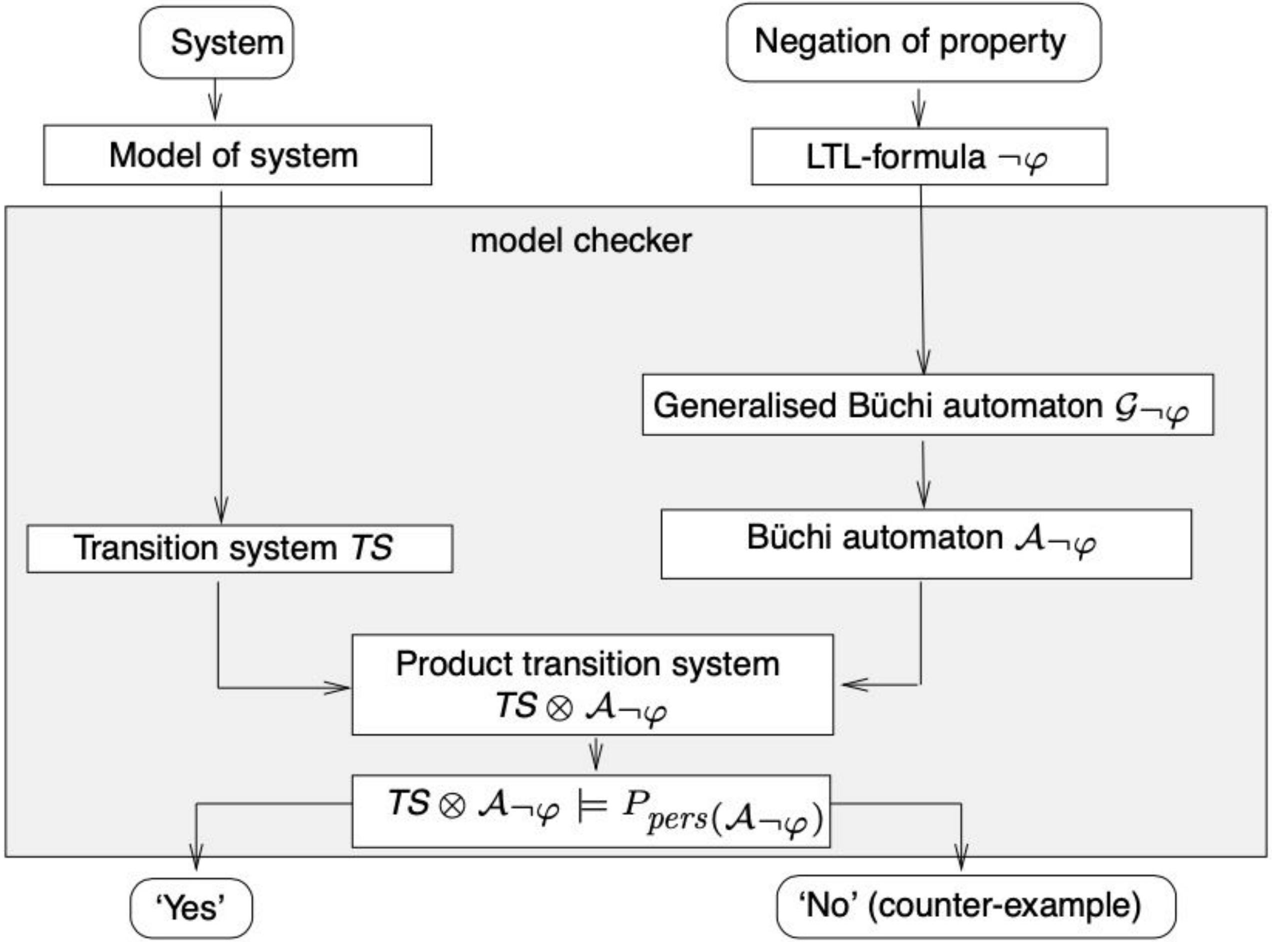
LTL formula to check $\phi = \Box\Diamond \mathit{green}$



$$\mathcal{T} \otimes \mathcal{A}_{\neg\phi} \overset{?}{\models} \Diamond\Box\neg F \text{ with } F = \{q_2\}$$

Yes! State <s1, q2> can be seen at most once, and state <s2,q2> is not reachable.

==⇒ There is no common trace between T and A¬φ


System
Negation of property
Model of system
LTL-formula $\neg\varphi$
model checker
Generalised Büchi automaton $\mathcal{G}_{\neg\varphi}$
Büchi automaton $\mathcal{A}_{\neg\varphi}$
Transition system *TS*
Product transition system
$TS \otimes \mathcal{A}_{\neg\varphi}$
$TS \otimes \mathcal{A}_{\neg\varphi} \models P_{pers(\mathcal{A}_{\neg\varphi})}$
'Yes'
'No' (counter-example)

CTL

Computation Tree Logic

- CTL is a branching time logic, i.e. reasoning over the tree of executions, i.e. one “time instant” may have several possible successor “time instants”

- Its models usually representing computations, in which the branching structure is used to describe uncertainty/ ignorance in a non-deterministic way

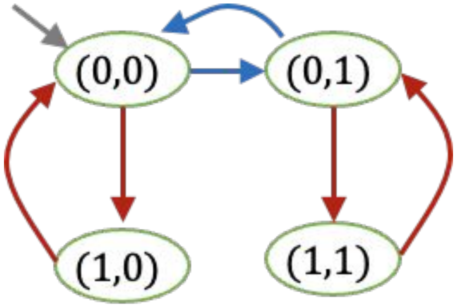
- We care about CTL because:
  - There are some properties that cannot be expressed in LTL, but can be expressed in CTL (and viceversa)
    From every system state, there is a system execution that takes it back to the initial state (also known as the reset property)

  - Can express interesting properties for multi-agent systems

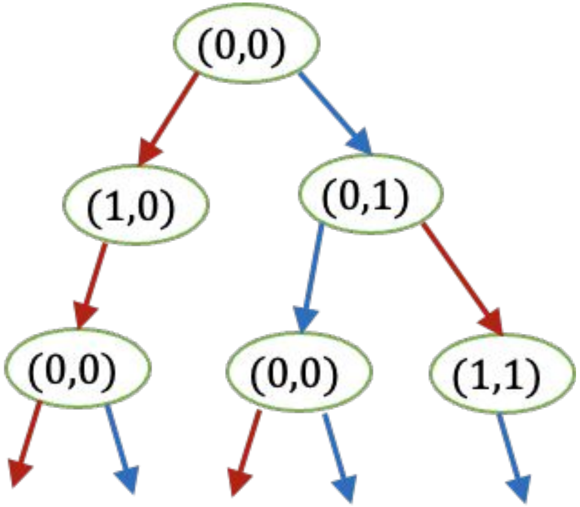
Computation Tree

```
nat x := 0; bool y:= 0
A:  x := (x + 1) mod 2
B: even(x) → y: = 1-y
```

Process



Finite State machine



- Basically a tree that considers “all possibilities” in a reactive program

CTL Syntax

State Formulae

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\psi \mid \mathbf{A}\psi$$

Path Formulae

$$\psi ::= \varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

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

- *Path Quantification:*
  - ***E***ψ, existential quantification: there **exists** a path (out of a given state) for which ψ holds

  - ***A***ψ, universal quantification: for **every** path (out of a given state), ψ holds.

CTL semantics

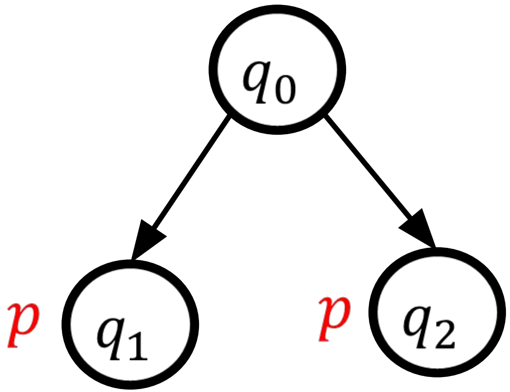
- Example CTL operator:

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

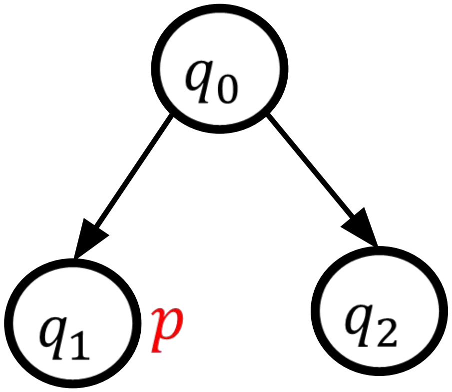
For **A**ll executions

Eventually/In Some **F**uture step

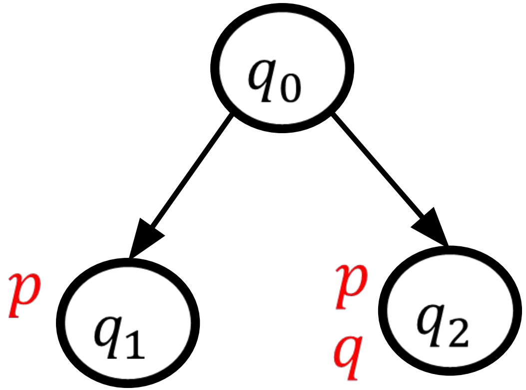
CTL semantics through examples



$\mathbf{AX}\, p$

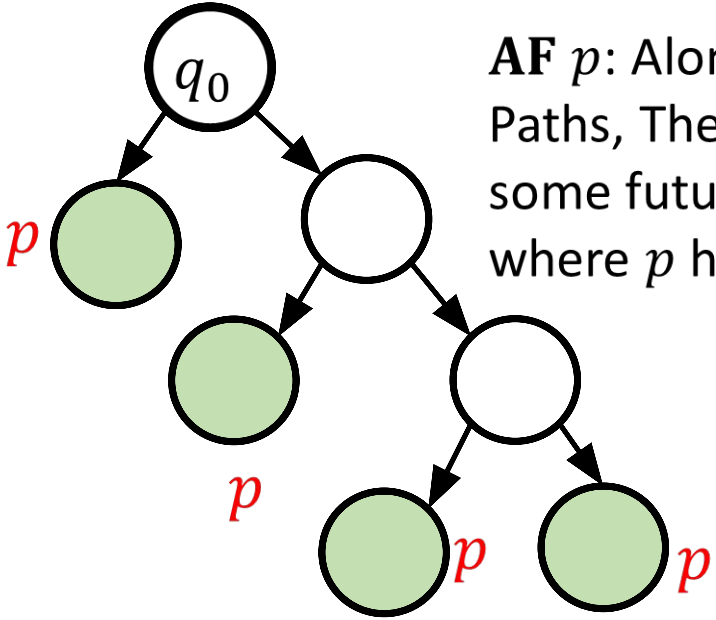


$\mathbf{EX}\, p$

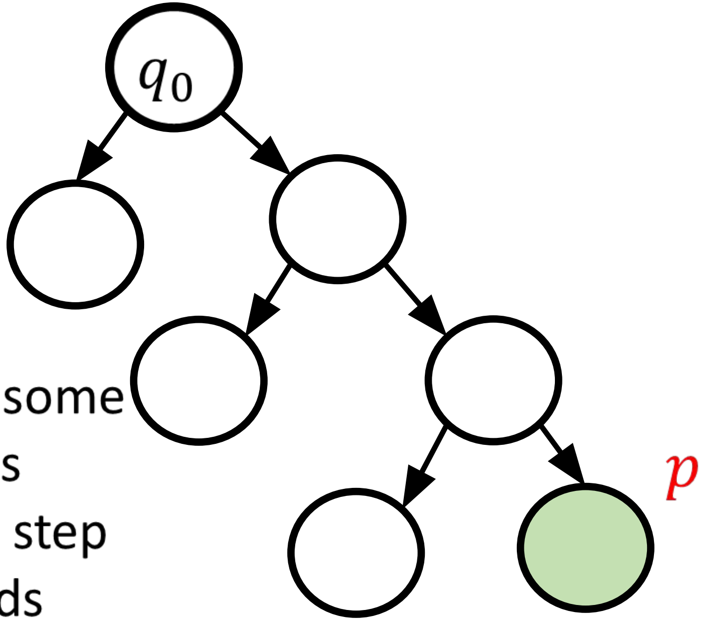


$\mathbf{AX}\, p \wedge \mathbf{EX}\, q$

CTL semantics through examples

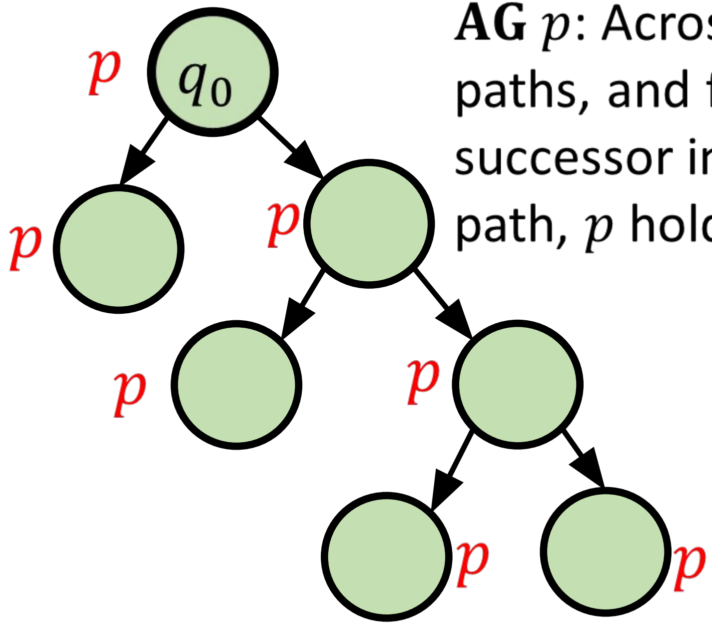


AF p: Along all Paths, There is some future step where p holds

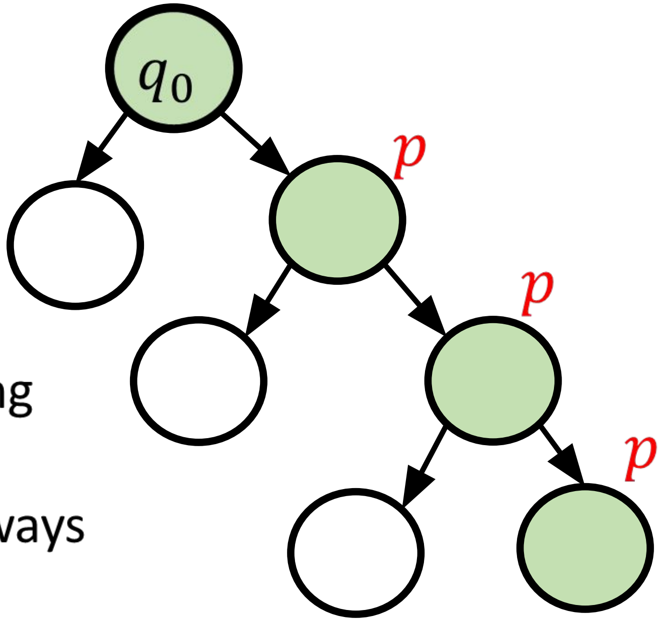


EF p: Along some path, there is some future step where p holds

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
- 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

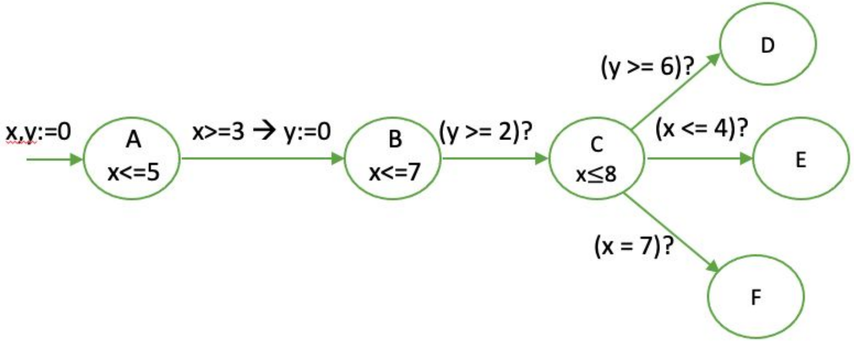
Timed Automata

Finite-state timed automaton: a machine where all state variables other than clock variables have finite types (e.g. Boolean, enums)

State-space of timed automata is infinite (clocks can become arbitrarily large!)

An automata with:

- A set of clock C
- A set of clock constraints on the transition



Timed Computation Tree Logic TCTL

State Formulae

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{E}\psi \mid \mathbf{A}\psi$$

Path Formulae

$$\psi ::= \varphi \mid \varphi \mathbf{U}_I \varphi$$

TCTL Example

- $\mathbf{A}[off\,\mathbf{U}_{[0,15]}\,on]$
- $EF^{(0,2]}\ b$