Cyber-Physical Systems

Laura Nenzi &
Stefan Schupp

Università degli Studi di Trieste
II Semestre 2022

Lecture : Model Checking

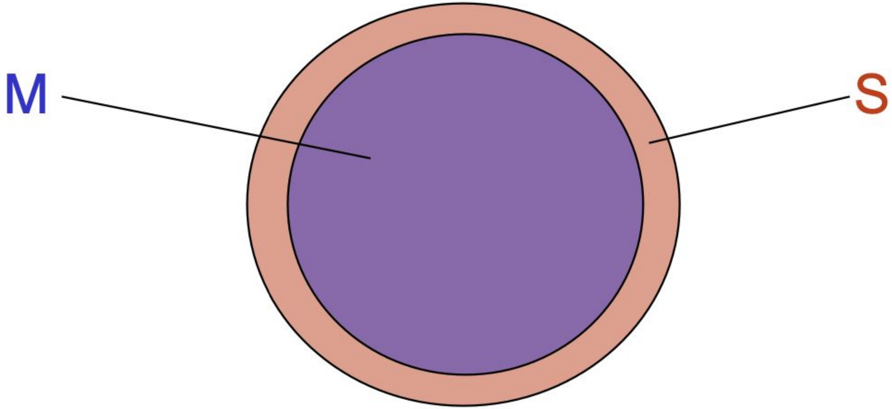
Model Checking

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

$$M \vDash S$$

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

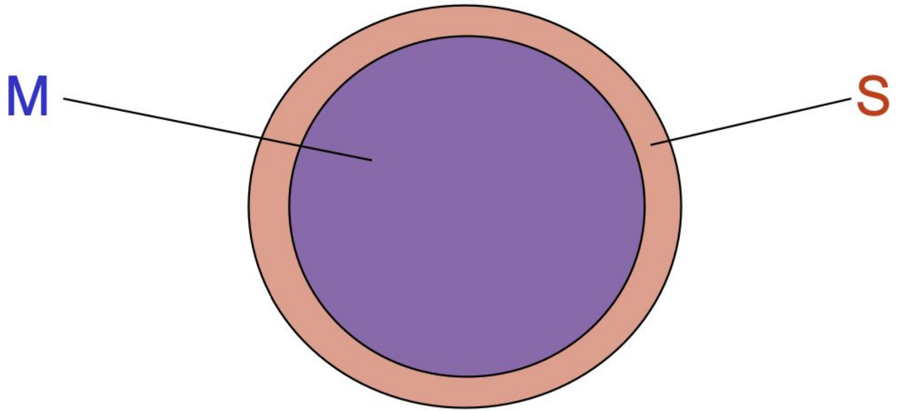
i.e. if every behaviour of M is also behaviour of S



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

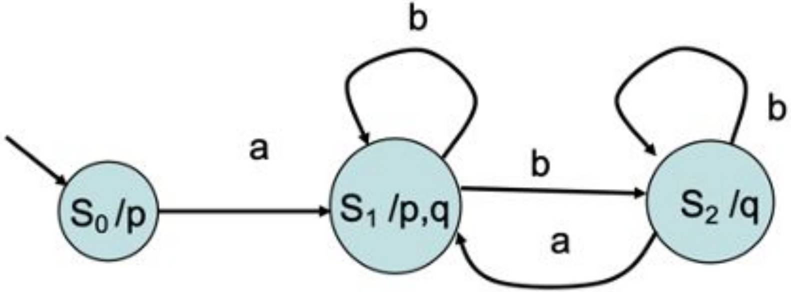
- If a component has finite input/output types and a finite number of “states” in its ESM, then it has a finite-state transition system

- Continuous components, Timed Processes, Hybrid Processes in general, have infinite number of states

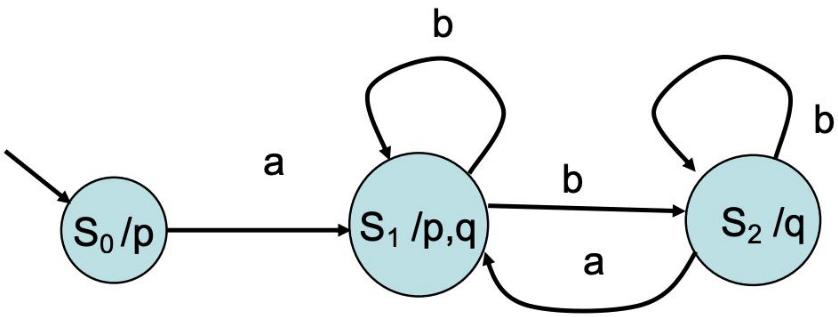
(Label) Transition System

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, $s_i \rightarrow^{\alpha i} s_{i+1}$
- □AP: set of **atomic proposition** on S
- □L:S →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. $\rho = s_0\ as_1b\ s_2bs_2bs_2\ldots$ □
- 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. $L(s_0)L(s_1)L(s_2)L(s_2)L(s_2)\ldots=p\{p,q\}qqq$ □

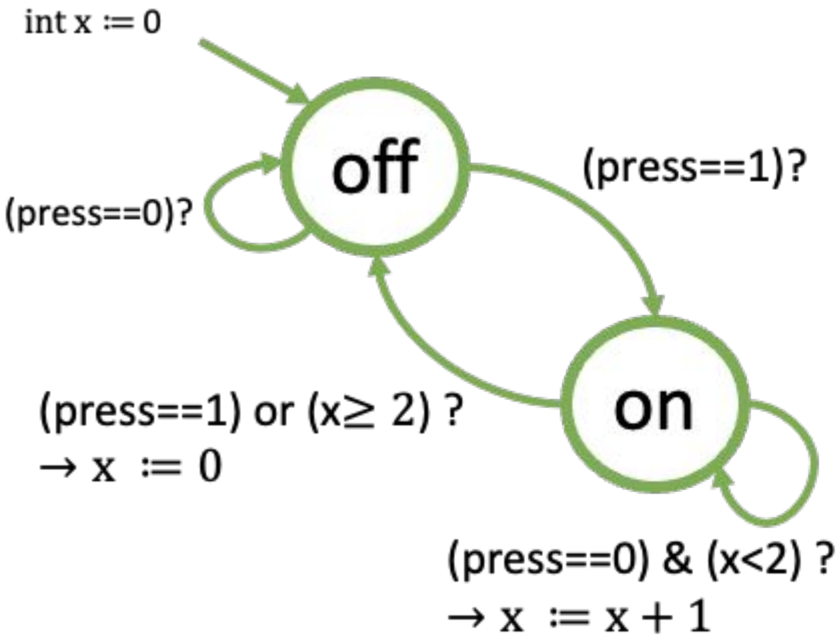
Conditional Transition

$$\mathbf{s} \xrightarrow{g:\alpha} \mathbf{s}'$$

g: a boolean condition on data variables

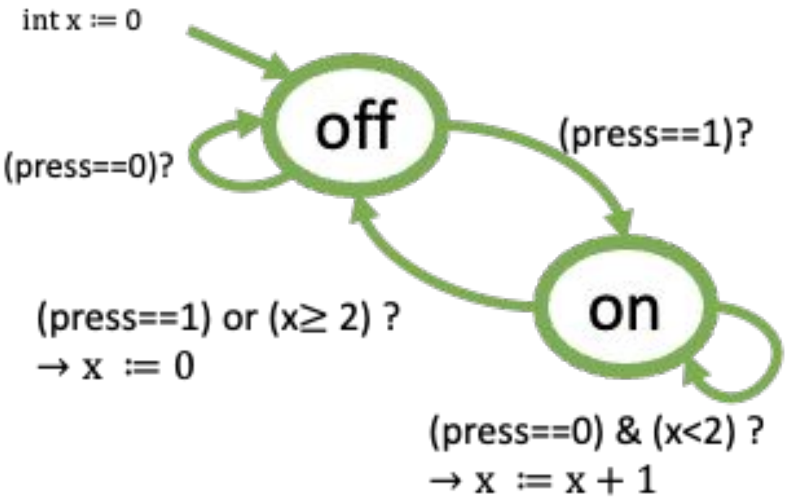
α: an action that is possible if g is satisfied

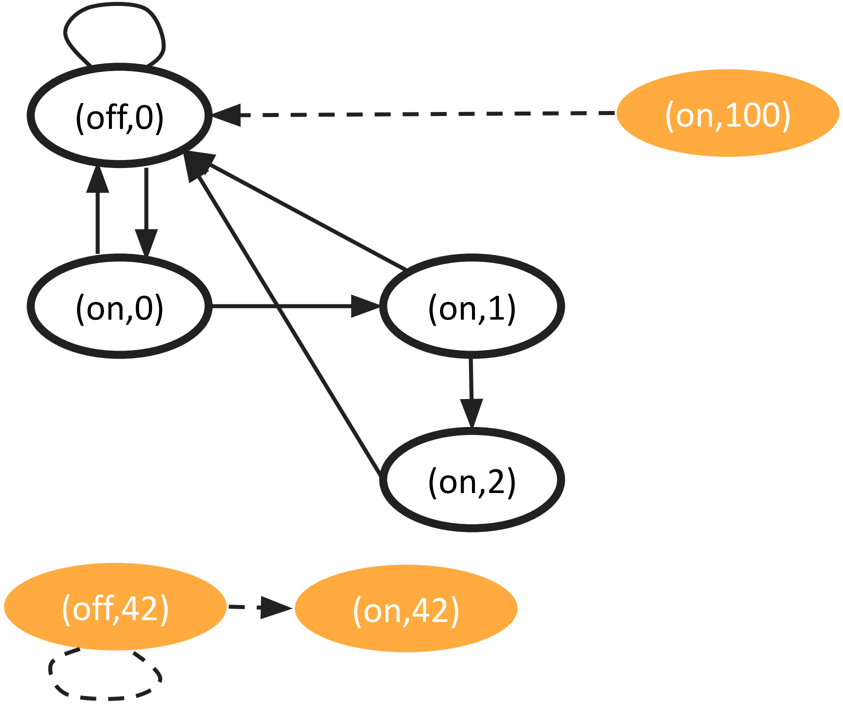
Example of a TS



- S ={on, off}×int
- I = { off, x = 0 }
- ⟦T⟧ has an infinite number of transitions:
  E.g. (off, 0)→(on,0)
  (on 0)→(on,1)

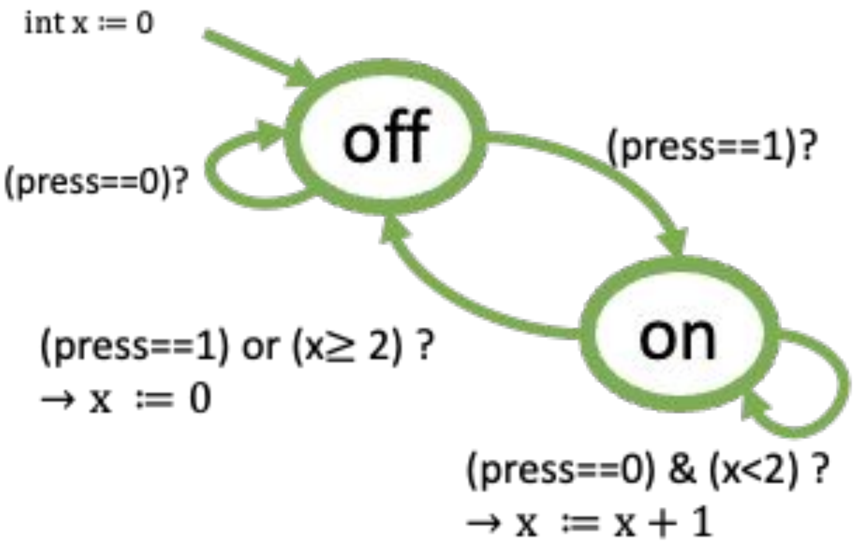
TS describes all possible transitions

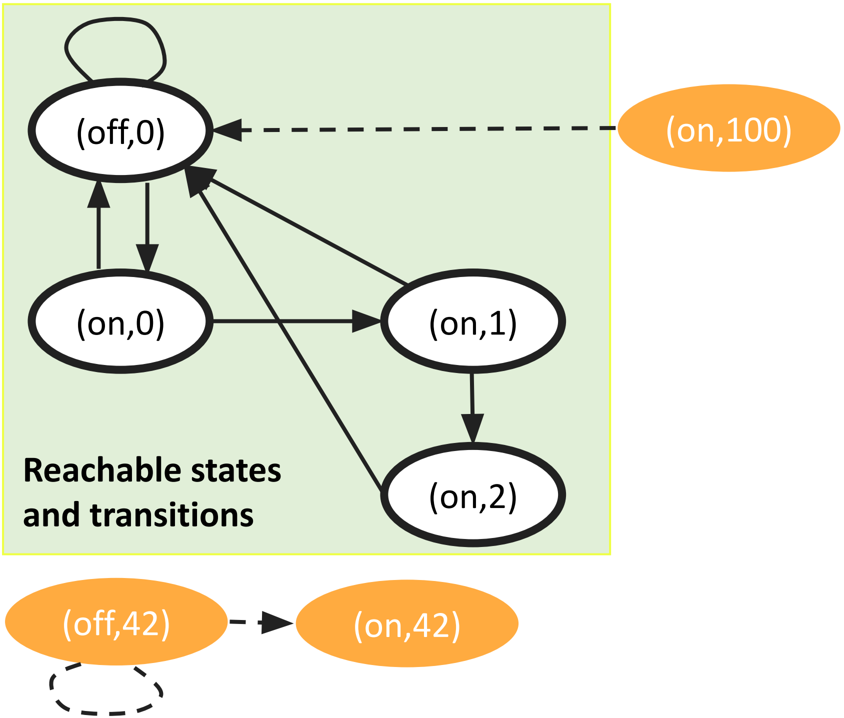




- Transitions indicated as dotted lines can’t really happen in the component
- But, the TS will describe then, as the states of the TS are over {on,off}×int!

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>:

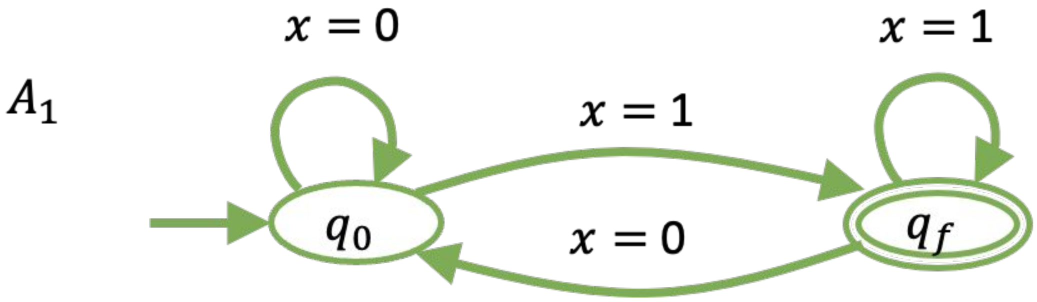
- Q finite set of states (like a TS) –
- Q_0 is a set of initial states (like a TS) –
- Σ is a finite alphabet (like a TS) –
- δ is a transition relation, δ: SxΣ →2^S (like a TS)
- F ⊆ Q is a set of accepting states

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

Büchi automaton

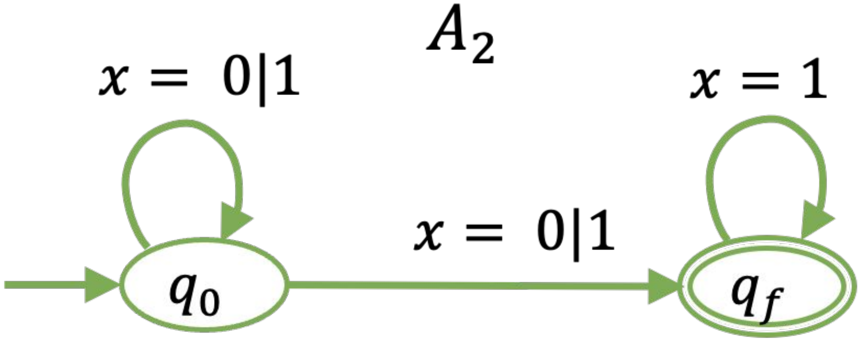
Every LTL formula φ can be converted to a Büchi monitor/automaton A_φ

Example: What is the language of A_1?



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

Büchi automaton Example

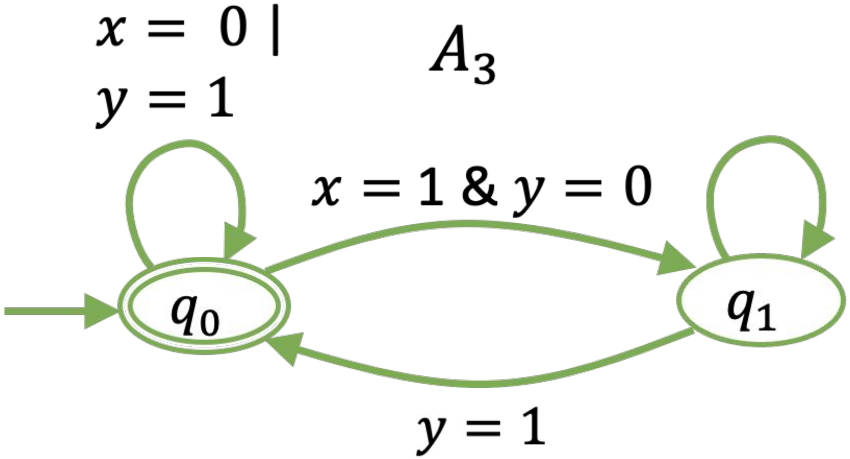


- S: {q_0,q_f }, Σ: {0,1}, F: {q_f}
- Transitions: (as shown)

- 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 ¬φ 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

$$TS \models \varphi \quad \text{if and only if} \quad \mathit{Traces}(TS) \subseteq \mathit{Words}(\varphi)$$

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

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

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

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

Synchronous Product ⊗

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

$$TS \otimes A = (S', Act, ⟦T⟧', I', AP', L')$$

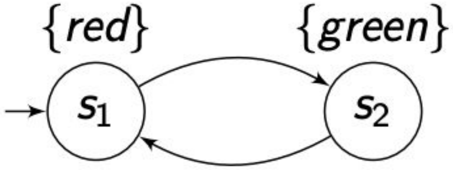
- $S'=S□Q$
- $I' = \{ \langle s_0 , q \rangle \mid s_0 \in I \land \exists\ q_0 \in Q_0 .\ q_0 \rightarrow^{L(s0)} q \}$
- Act: Set of **actions**
- AP'=Q
- L'=(<s,q>={q})
- ⟦T⟧':

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

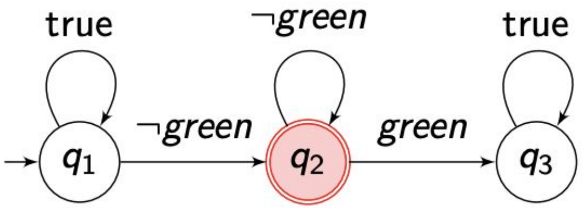
Synchronous Product

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

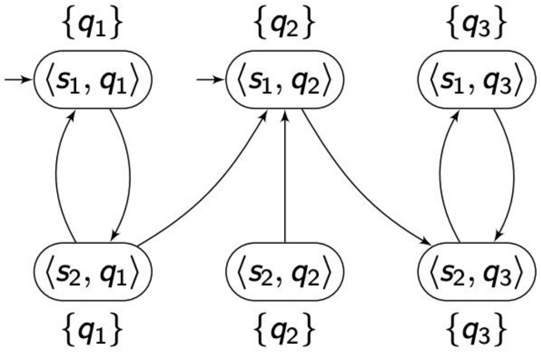
LTL formula to check $\phi = \Box\Diamond \mathit{green}$



TS *T* for the traffic light.



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



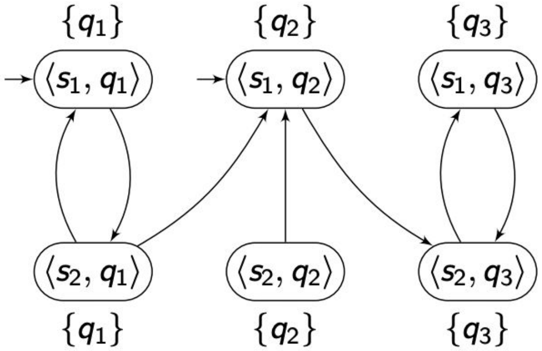
$\{q_1\}$
$\{q_2\}$
$\{q_3\}$
$\langle s_1, q_1\rangle$
$\langle s_1, q_2\rangle$
$\langle s_1, q_3\rangle$
$\langle s_2, q_1\rangle$
$\langle s_2, q_2\rangle$
$\langle s_2, q_3\rangle$
$\{q_1\}$
$\{q_2\}$
$\{q_3\}$

==⇒ Blackboard construction of T ⊗ A¬φ.

Synchronous Product

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

LTL formula to check $\phi = \Box\Diamond green$

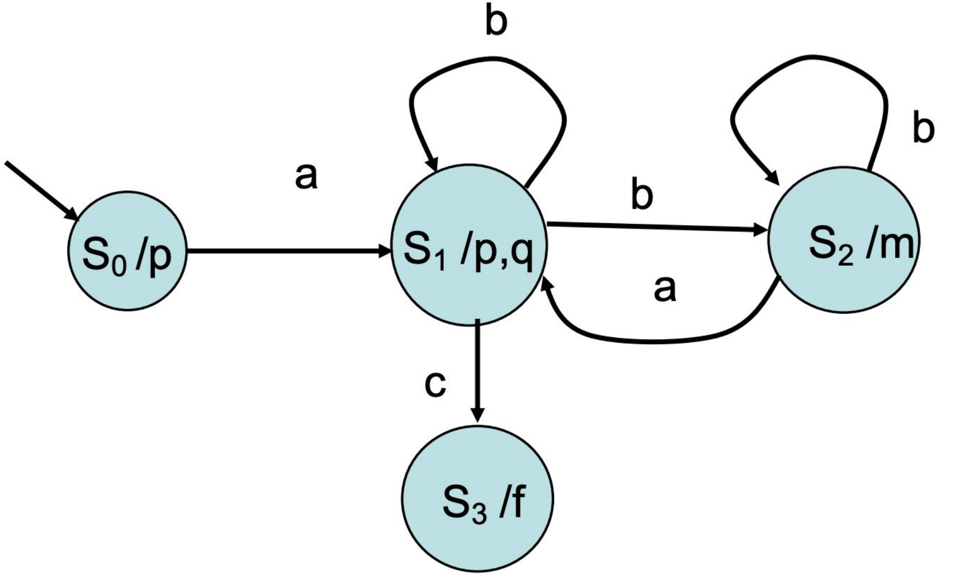


$\{q_1\}$
$\{q_2\}$
$\{q_3\}$
$\langle s_1, q_1\rangle$
$\langle s_1, q_2\rangle$
$\langle s_1, q_3\rangle$
$\langle s_2, q_1\rangle$
$\langle s_2, q_2\rangle$
$\langle s_2, q_3\rangle$
$\{q_1\}$
$\{q_2\}$
$\{q_3\}$

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

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

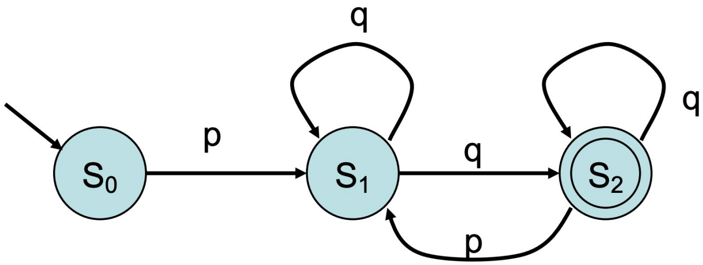
Specification in LTL



$\mathsf{F}m$

$G(m \rightarrow Xq)$

Example: accepted words



What words are accepted by this automaton B?

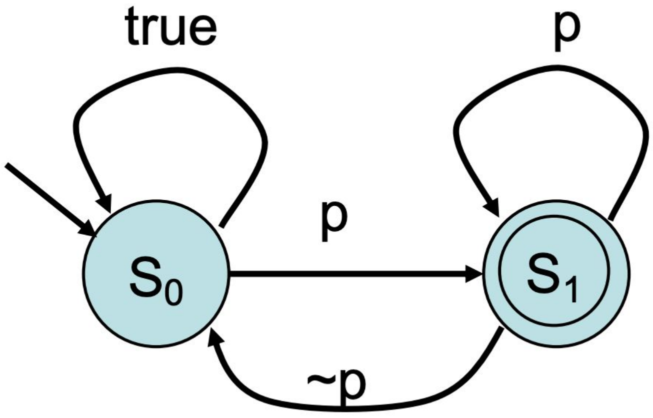
L(B) = pq+(pq+)* L(B) is called the language of B.

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

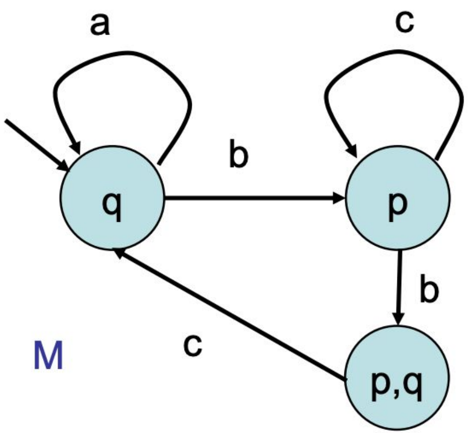
$\phi = G\ F\ p$



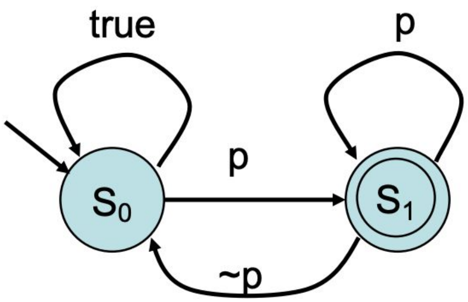
B_φ

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}*)



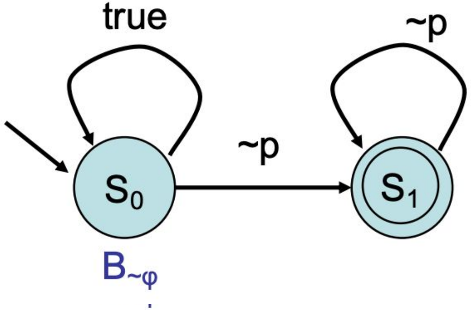
M



B_φ, φ = GF p

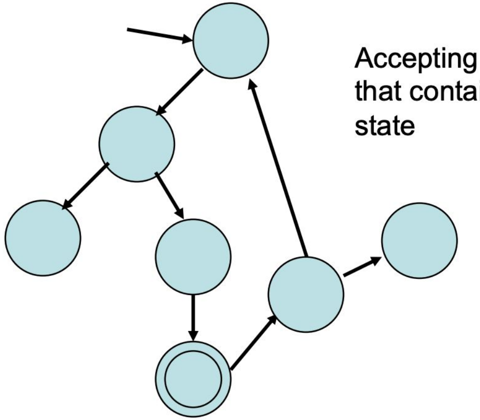
LTL Model Checking

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

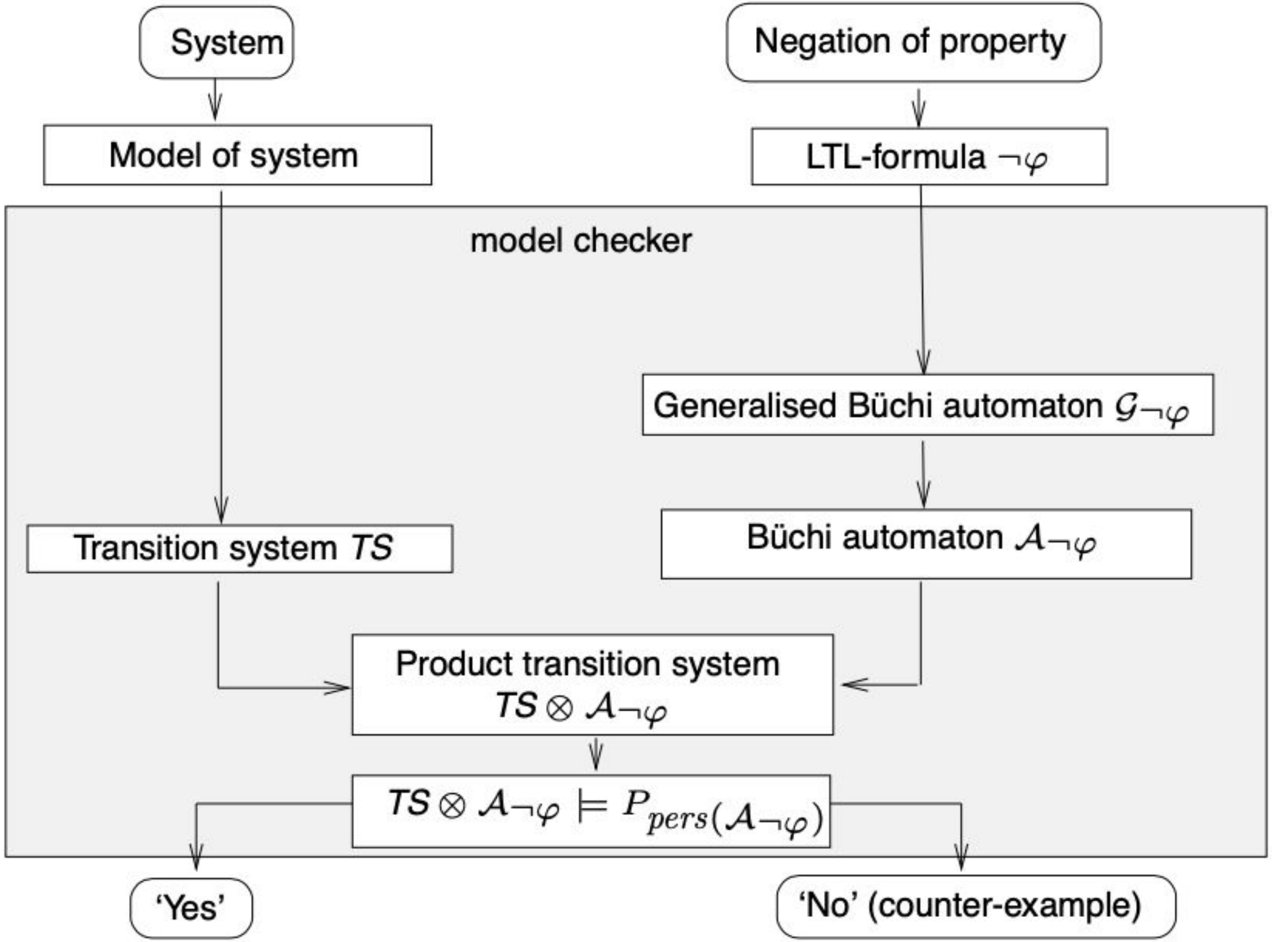


LTL Model Checking

- 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


System
Negation of property
Model of system
LTL-formula $\neg\varphi$
model checker
Generalised Büchi automaton $\mathcal{G}_{\neg\varphi}$
Transition system *TS*
Büchi automaton $\mathcal{A}_{\neg\varphi}$
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

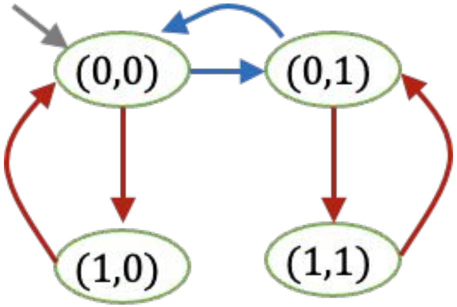
- We care about CTL because:
    - There are some properties that cannot be expressed in LTL, but can be expressed in CTL (and viceversa)
      From every system state, there is a system execution that takes it back to the initial state (also known as the reset property)
    - Can express interesting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

Computation Tree

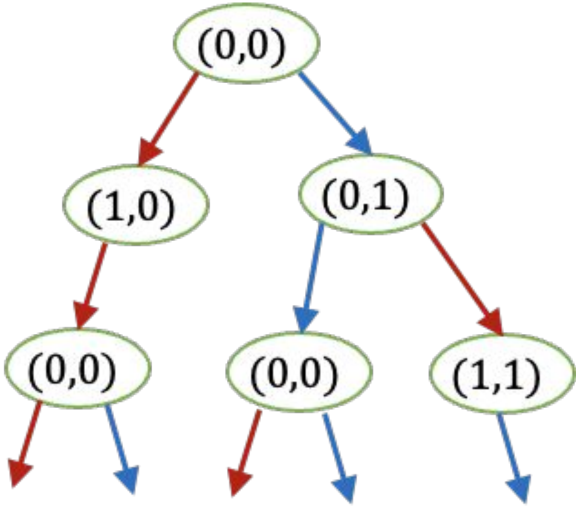
```
nat x := 0; bool y:= 0

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	
$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \quad \mid$	Prop. in *AP*, negation, conjunction
$\mathbf{EX}\varphi \quad \mid$	Exists Ne**X**t Step
$\mathbf{EF}\varphi \quad \mid$	Exists a **F**uture Step
$\mathbf{EG}\varphi \quad \mid$	Exists an execution where **G**lobally in all steps
$\mathbf{E}\varphi\mathbf{U}\varphi \quad \mid$	Exists 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

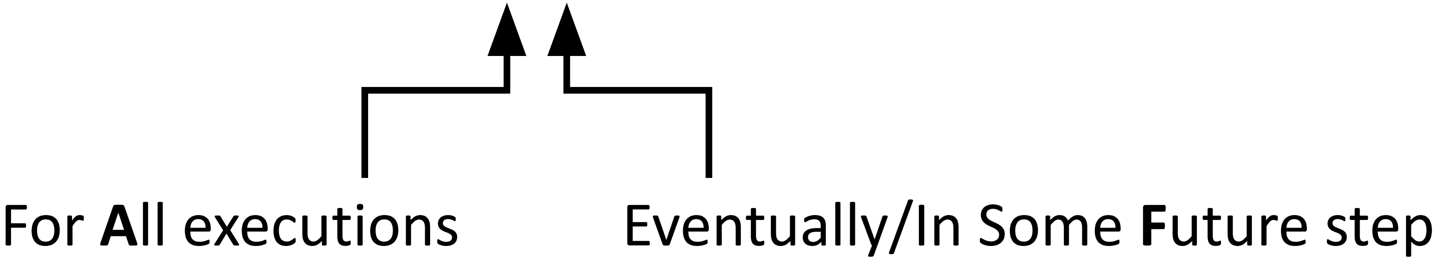
- *Path Quantification:*
  - ***E***ψ, existential quantification: there **exists** a path (out of a given state) for which ψ holds

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

CTL semantics

- Example CTL operator:

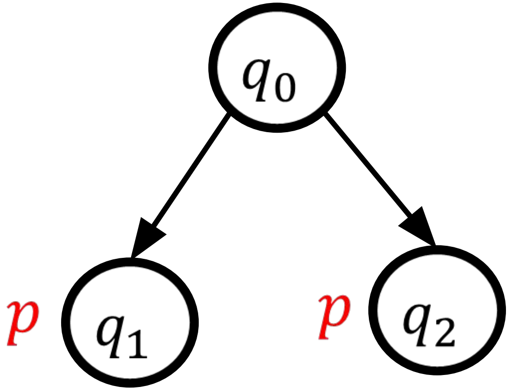
$$\mathbf{A}\,\mathbf{F}\ \ p$$



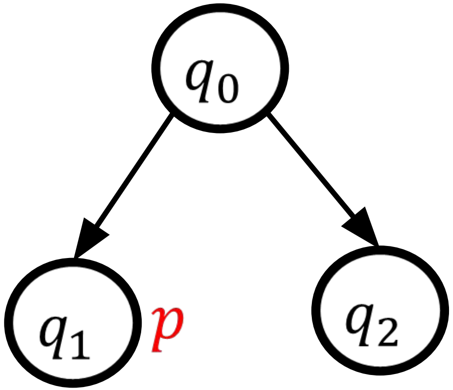
For **A**ll executions

Eventually/In Some **F**uture step

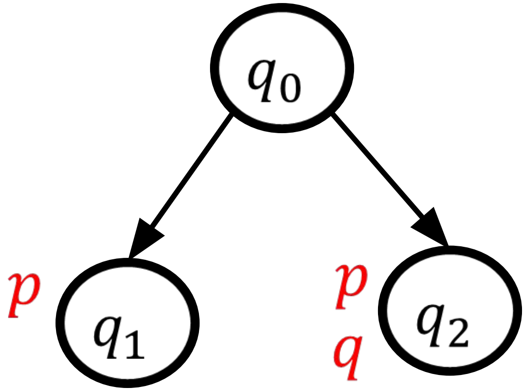
CTL semantics through examples



$\mathbf{AX}\,p$

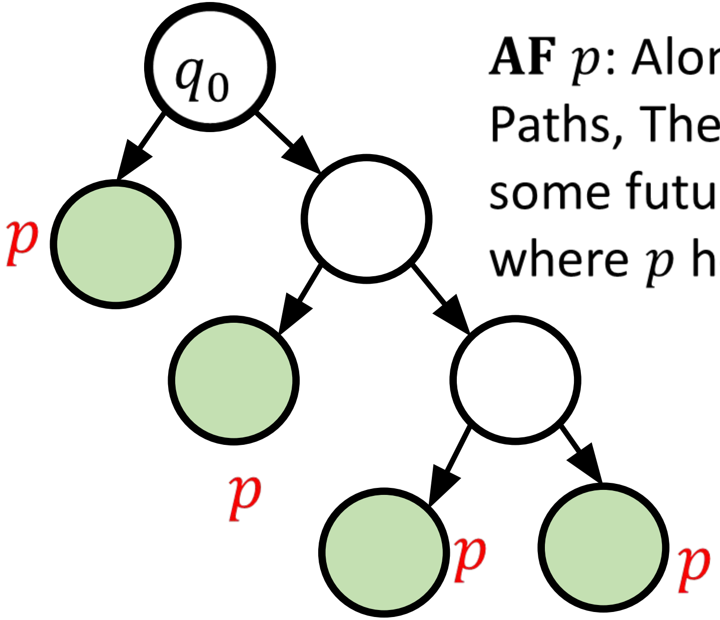


$\mathbf{EX}\,p$

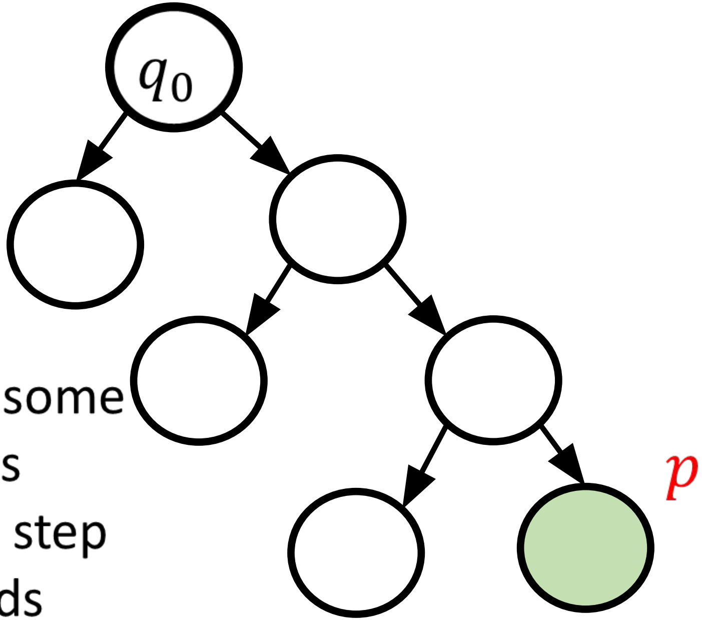


$\mathbf{AX}\,p \land \mathbf{EX}\,q$

CTL semantics through examples

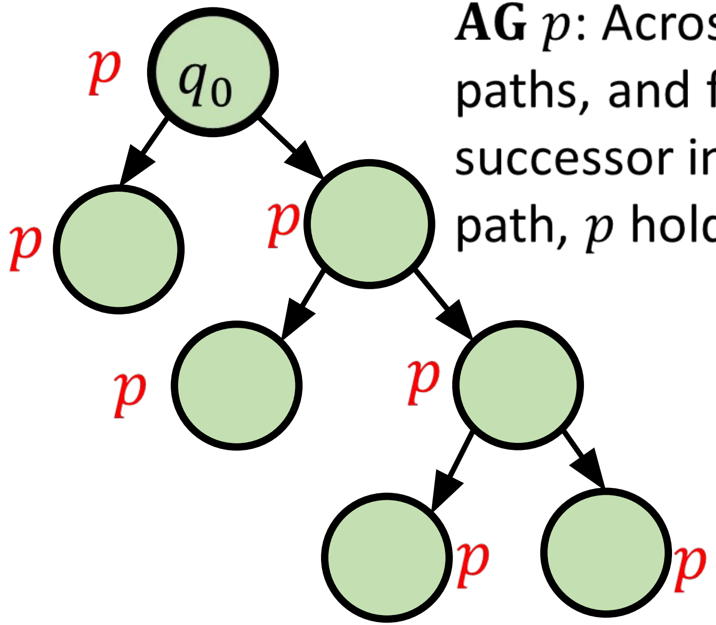


AF p: Along all Paths, There is some future step where p holds

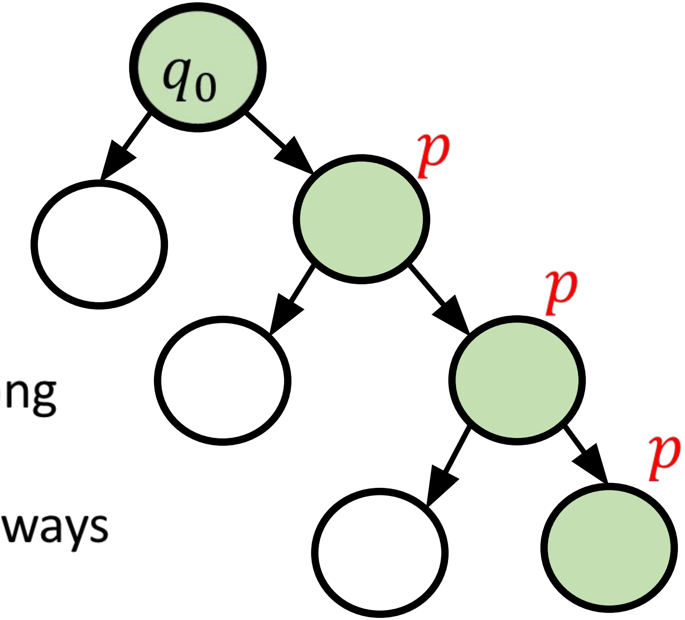


EF p: Along some path, there is some future step where p holds

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

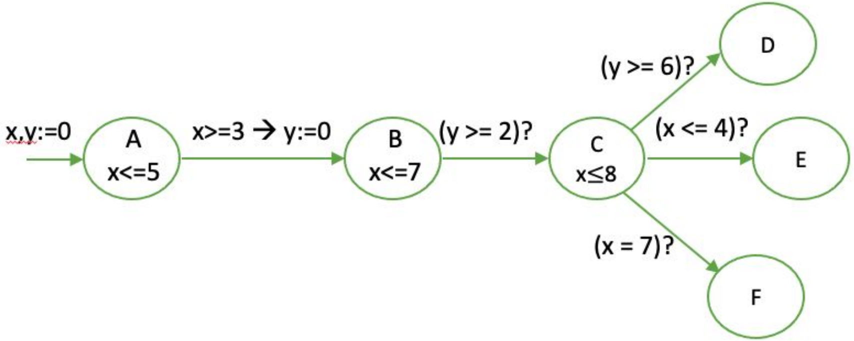
Timed Automata

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

- $\mathbf{A}[\text{off}\mathbf{U}_{[0,15]}\text{on}]$

- $\text{EF}^{(0,2]}\ \text{b}$

Timed Automata Model Checking

Basic Method: Abstraction

- Given: a concrete system (here a timed automaton)
- Goal: reduce the size of the system by abstraction (here reduce infinite state space to a finite one)
- Result: abstract system (here a region transition system)

Behaviorally equivalent abstraction: If treated as a black box, we cannot distinguish the abstraction from the original in experiments.

Example: Input-output behavior for programs.

For model checking: both satisfy the same formulas of the underlying logic.

Model checking for timed automata

Input: timed automaton *T*, TCTL formula ψ

Output: the answer whether *T* $\models \psi$

1. Eliminate timing parameters of ψ, provides CTL formula ψ’ with clock constraints
2. Create finite abstraction of the state space of *T*
3. Create abstract state transition system RTS such that *T* $\models \psi$ iff *RTS* $\models \psi$’
4. Apply CTL model checking to check whether *RTS* $\models \psi$’

1. Eliminating timing parameters

Let T be a timed automaton with clocks C and atomic propositions AP. Let T’ result from T by adding a fresh clock z which never gets reset.

For any state s of T it holds that

1. T,s $\models_{\mathrm{TCTL}}$ $E(\psi\ U^{J}\ \varphi)$ iff
   T’,reset(z) in s $\models_{\mathrm{TCTL}}$ $E(\psi\ U\ (z \in J) \wedge \varphi)$
2. T,s $\models_{\mathrm{TCTL}}$ $A(\psi\ U^{J}\ \varphi)$ iff
   T’,reset(z) in s $\models_{\mathrm{TCTL}}$ $A(\psi\ U\ (z \in J) \wedge \varphi)$
3. T, s $\models_{\mathrm{TCTL}}$ $EF^{\leq 2}\varphi$ iff
   T’,reset(z) in s $\models_{\mathrm{TCTL}}$ $EF\ ((z \leq 2) \wedge \varphi)$
4. T, s $\models_{\mathrm{TCTL}}$ $EG^{\leq 2}\varphi$ iff
   T’,reset(z) in s $\models_{\mathrm{TCTL}}$ $EG\ ((z \leq 2) \rightarrow \varphi)$

2. Finite state space abstraction

We search for an equivalence relation ~ on states such that equivalent states satisfy the same (sub)formulas ψ' occurring in the timed automaton T or in the specification ψ: $s \sim s' \Rightarrow (s \models \psi'$ iff $s' \models \psi')$.

Goal: find a *finite* number of equivalence classes.

Definition (Bisimulation): Assume an LSTS with states Σ and edge relation →. Let AP be a set of atomic propositions and $L: \Sigma \rightarrow 2^{AP}$ a labeling function. A bisimulation for LSTS is an equivalence relation $\approx \subseteq \Sigma \times \Sigma$ such that for all $s_1 \approx s_2$

1. $L(s_1) = L(s_2)$
2. for all $s_1' \in \Sigma$ with $s_1 \rightarrow_a s_1'$ there exists $s_2' \in \Sigma$ such that $s_2 \rightarrow_a s_2'$ and $s_1' \approx s_2'$

Time abstract bisimulation

A *time abstract bisimulation* for a timed automaton T is an equivalence relation $\approx \subseteq \Sigma \times \Sigma$ such that for all $s_1, s_2 \in \Sigma$ satisfying $s_1 \approx s_2$

- $L(s_1) = L(s_2)$
- for all $s_1' \in \Sigma$ with $s_1 \rightarrow_a s_1'$ there exists $s_2' \in \Sigma$ such that $s_2 \rightarrow_a s_2'$ and $s_1' \approx s_2'$
- for all $s_1' \in \Sigma$ with $s_1 \rightarrow_t s_1'$ there exists $s_2' \in \Sigma$ such that $s_2 \rightarrow_t s_2'$ and $s_1' \approx s_2'$

Intuition: given TA T and a timed bisimulation then

$$\begin{tikzcd}
\pi: s \arrow[r] \arrow[d, phantom, sloped, "\approx"] & s_1 \arrow[r] \arrow[d, phantom, sloped, "\approx"] & s_2 \arrow[r] \arrow[d, phantom, sloped, "\approx"] & \ldots \\
\pi': s' \arrow[r] & s_1' \arrow[r] & s_2' \arrow[r] & \ldots
\end{tikzcd}$$

Finite state space abstraction

For timed automata, states s = (l,v) and s' = (l',v') are equivalent, if

- l = l'
- s and s' satisfy the same clock constraints:
  - For $x < c$, $c \in \mathbb{N}$: $v \models x < c \Leftrightarrow v(x) < c \Leftrightarrow \lfloor v(x) \rfloor < c$
  - For $x \leq c$, $c \in \mathbb{N}$: $v \models x \leq c \Leftrightarrow v(x) \leq c \Leftrightarrow \lfloor v(x) \rfloor < c \lor (\lfloor v(x) \rfloor = c \land \mathrm{frac}(v(x)) = 0)$

Notation: v = valuation
v(x) = valuation of variable x

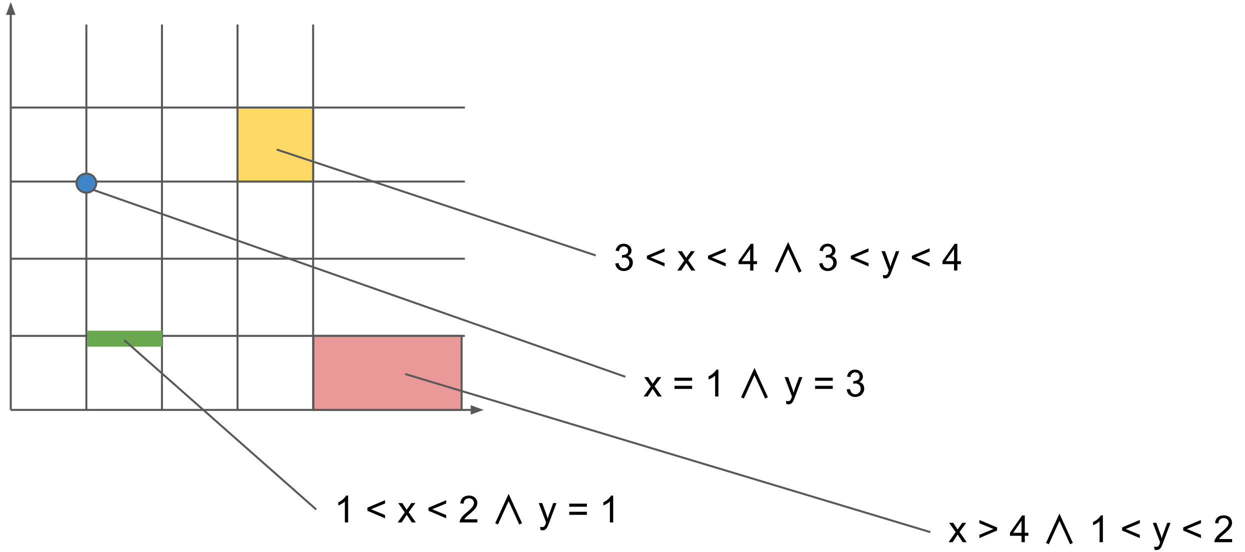
Problem: creates infinitely many classes!

Idea: we cannot distinguish classes for values larger than the largest constant c in T.

Solution: collect all equivalence classes for values larger than c.

Finite state space abstraction

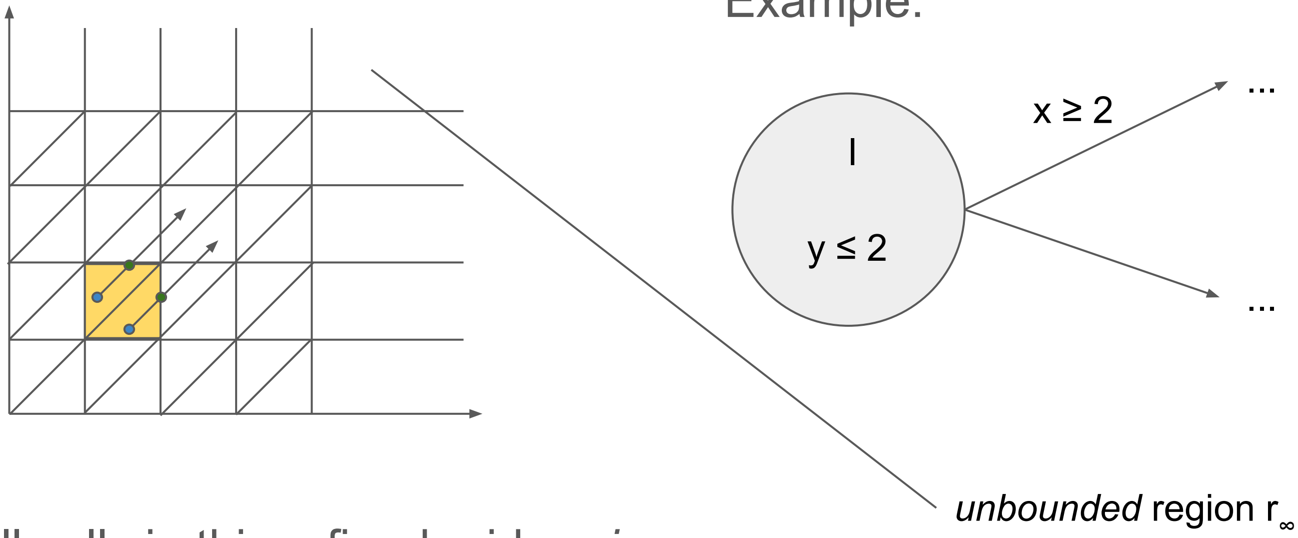
Largest constants $c_x = 4$, $c_y = 4$



Finite state space abstraction

Largest constants $c_x = 4$, $c_y = 4$

Require further refinement!
Example:



We call cells in this refined grid *regions*.

Model checking for timed automata

Input: timed automaton T, TCTL formula ψ

Output: the answer whether $T \models \psi$

1. Eliminate timing parameters of ψ, provides CTL formula ψ' with clock constraints
2. Create finite abstraction of the state space of T
3. Create abstract state transition system RTS such that $T \models \psi$ iff $RTS \models \psi$'
4. Apply CTL model checking to check whether $RTS \models \psi$'

3. Region transition system

We have two kinds of transitions between regions: time-elapse and discrete jumps.

Given regions r, r’, r’ = succ(r) if

- $r = r' = r_\infty$, or
- $r \neq r_\infty$, $r \neq r'$, and for all ν in r:

$$\exists\, d \in \mathbb{R}_{>0}.\ (\nu + d \in r' \land \forall\ 0 \leq d' \leq d.\ \nu + d' \in r \cup r')$$

Intuition: r’ = succ(r) if either both are the unbounded region or if r’ can be reached by time elapse and is the *direct* successor region.

3. Region transition system

The region transition system (RTS) R for a timed automaton T and a TCTL formula ψ over atomic propositions AP is defined as:

- The state set Σ is the set of all regions (l,V) in T where V ∈ Inv(l)
- The initial region is build from the initial states of T
- The transition relation is extended to time-successor regions via succ(r) and jump successor regions (see examples)

The set of atomic propositions AP' of R is given as AP ∪ ACC(T) ∪ ACC(ψ), the labeling function L((l,V))' = L(l) ∪ {g ∈ AP' \ AP | V ⊧ g}.

Idea: Add APs to be able to label regions which satisfy certain atomic clock constraints (ACC).

Model checking for timed automata

Input: timed automaton *T*, TCTL formula ψ

Output: the answer whether *T* ⊧ ψ

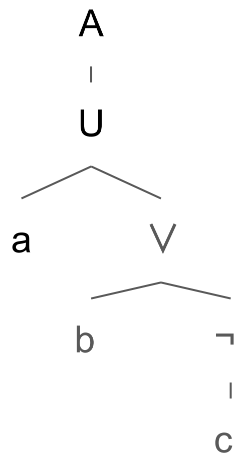
1. Eliminate timing parameters of ψ, provides CTL formula ψ’ with clock constraints
2. Create finite abstraction of the state space of *T*
3. Create abstract state transition system RTS such that *T* ⊧ ψ iff *RTS* ⊧ ψ’
4. Apply CTL model checking to check whether *RTS* ⊧ ψ’

4. CTL Model Checking

- Convert formula to *existential normal form* (ENF)
- Recursively, bottom-up:
  - Use parse tree of the converted formula
  - Compute SAT-sets of leaf nodes
  - Recursively: Compute SAT-set of parent nodes until root is reached

Example parse tree:

ψ: A(a U (b ∨ ¬ c))



Computing Sat-sets

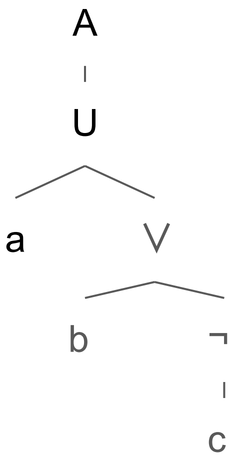
Given LTS with states s ∈ S, atomic propositions AP and CTL formulas ψ,φ it holds:

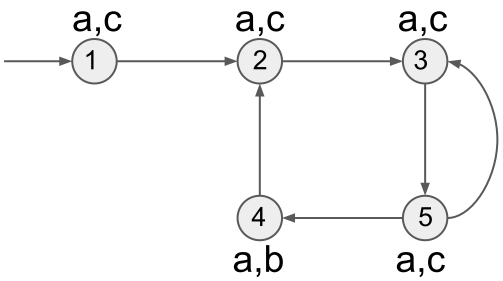
- Sat(true) = S
- Sat(a) = {s ∈ S | a ∈ L(s) } for any a in AP
- Sat($\psi \wedge \varphi$) = Sat(ψ) ∩ Sat(φ)
- Sat(¬φ) = S \ Sat(φ)
- Sat(E(ψUφ)) = smallest subset T of S where
  - Sat(φ) ⊆ T and
  - s ∈ Sat(ψ) and Post(s) ∩ T ≠ ∅ implies s ∈ T
- Sat(EFφ) = {s ∈ S | Post(s) ∩ Sat(φ) ≠ ∅}
- Sat(EGφ) = largest subset T of S where
  - T ⊆ Sat(φ) and
  - s ∈ T implies Post(s) ∩ T ≠ ∅

Intuition (until):
Every state satisfying φ directly satisfies the formula and every state from which such a state can be reached while satisfying ψ is added to the sat-set.

4. CTL Model Checking

ψ: A(a U (b ⋁ ¬ c))

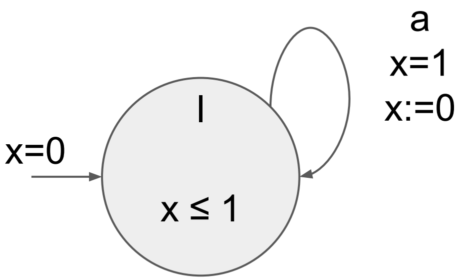




- Sat(a) = {1,2,3,4,5}
- Sat(b) = {4}
- Sat(c) = {1,2,3,5}

- Sat(¬c) = {4}
- Sat(b ⋁¬c) = {4}
- Sat(A(a U (b ⋁ ¬c)) = {4}

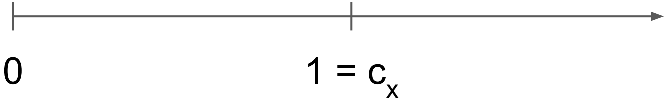
Complete examples



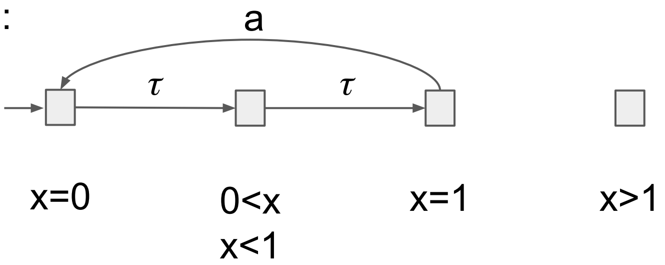
Formula: AGAF x = 0

1: $\underbrace{\neg \underbrace{EF \underbrace{\neg \underbrace{A\ \mathit{true}\ U \underbrace{x = 0}_{1}}_{2}}_{3}}_{4}}_{5}$

2:

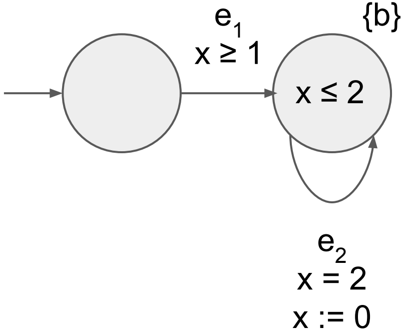


3:



4: 1,2,5   2,5   2,5   3,4

Complete examples



φ: $\mathrm{EF}^{(0,2]}$ b

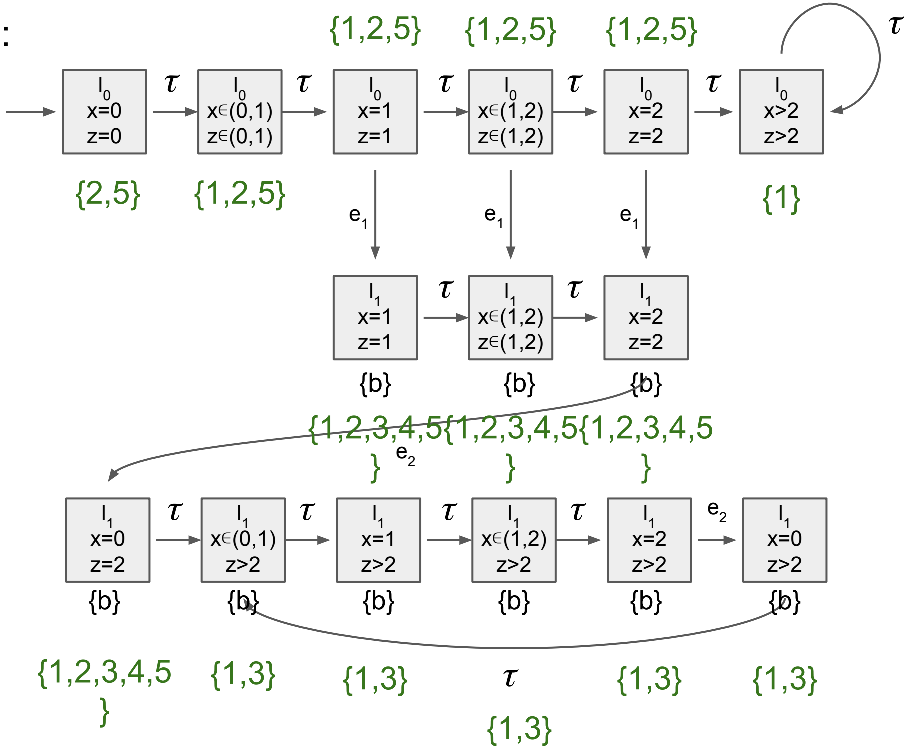
1. E(true U (b ∧ (z > 0 ∧ z ≤ 2)))

3 1 2

4

5

2,3:



{1,2,5}
{1,2,5}
{1,2,5}
τ
l_0 x=0 z=0
τ
l_0 x∈(0,1) z∈(0,1)
τ
l_0 x=1 z=1
τ
l_0 x∈(1,2) z∈(1,2)
τ
l_0 x=2 z=2
τ
l_0 x>2 z>2
{2,5}
{1,2,5}
{1}
e_1
e_1
e_1
l_1 x=1 z=1
τ
l_1 x∈(1,2) z∈(1,2)
τ
l_1 x=2 z=2
{b}
{b}
{b}
{1,2,3,4,5}
{1,2,3,4,5}
{1,2,3,4,5}
e_2
l_1 x=0 z=2
τ
l_1 x∈(0,1) z>2
τ
l_1 x=1 z>2
τ
l_1 x∈(1,2) z>2
τ
l_1 x=2 z>2
e_2
l_1 x=0 z>2
{b}
{b}
{b}
{b}
{b}
{b}
{1,2,3,4,5}
{1,3}
{1,3}
τ
{1,3}
{1,3}
{1,3}