Lecture 14: Introduction to Model Checking

What are models?

- A system: (S, I, T, O)
  - S: States $s_1, s_2 \dots s_n$
  - I: Inputs (could be $\emptyset$)
  - T: Transitions $S \times I \times S$
  - O: Observations $f(S_o), S_o \subseteq S$

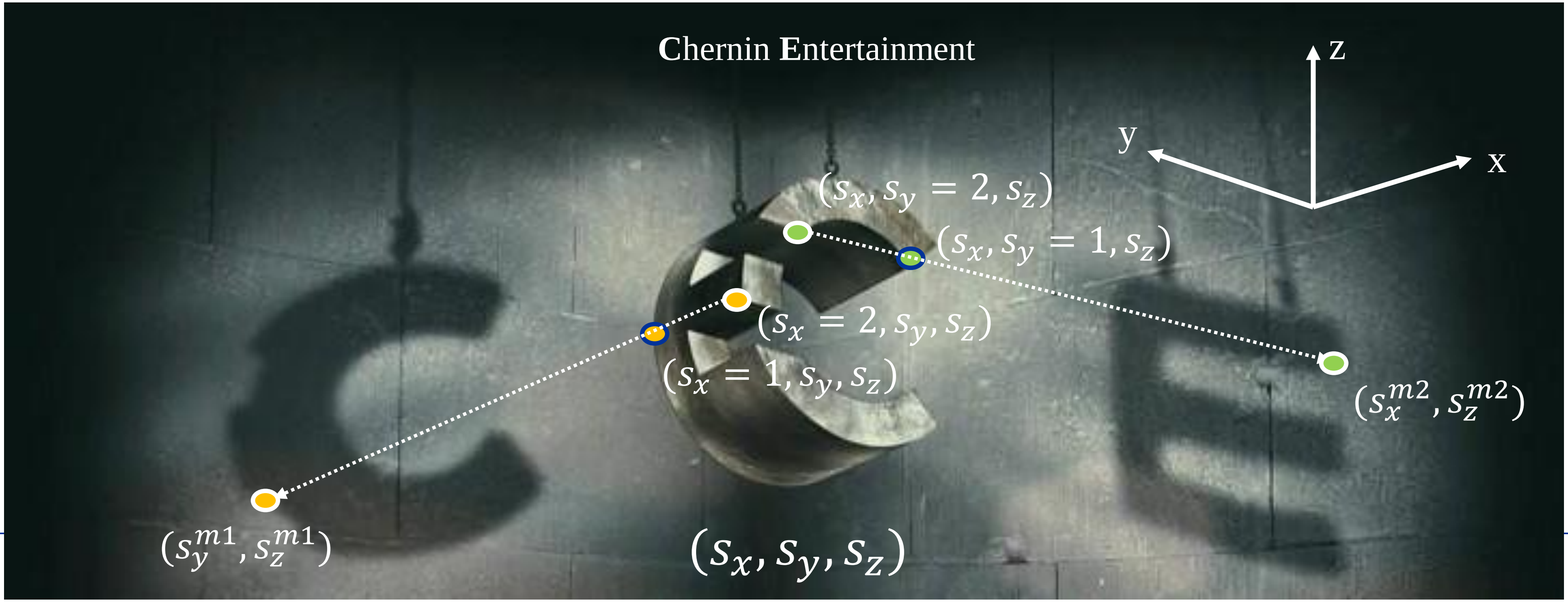
- Model of the system (S^m, I^m, T^m)
  - S^m: Abstraction/approximation of S
    - Much fewer state variables and their values
  - I^{m}: abstraction of I (could be $\emptyset$)
  - T^m: Transitions $S^m \times I^m \times S^m$

Abstraction: Different perspectives

- Models represent what you think are important for a system
- For different modeling objectives you can end up with different models

Abstraction – removal of state variables

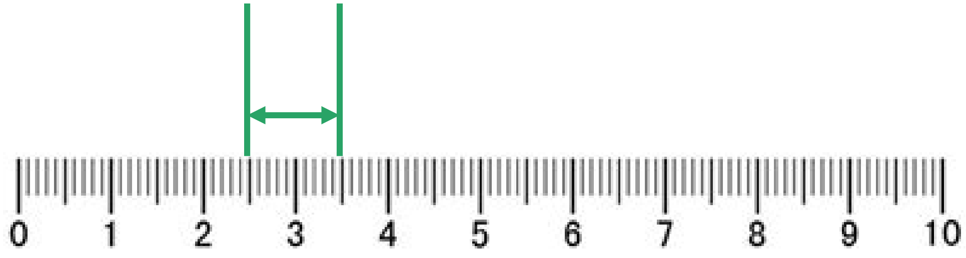
- States (s_x, s_y, s_z) *are abstracted to* (s_y^{m1}, s_z^{m1})
  - $(s_x, s_y, s_z) \rightarrow (s_y^{m1}, s_z^{m1})$
- Loss of information



Abstraction: Granularities of state variable

- Irrational numbers
  - $\pi \approx 3.1415$
  - $\sqrt{2} \approx 1.414$

- Approximation is another way of abstraction



Why do we need models?

- Prediction
  - We know the low-level mechanisms but we want to understand how they affect higher-level behaviors
  - Use simulation instead of testing on the real system
- Explain the data
  - Make assumptions and use our knowledge to explain mechanisms that we don’t understand
- Classification
  - i.e. definitions, machine learning algorithms

What is considered as a “good” model?

- Accuracy
  - All models are wrong!
  - Error accumulates over time
  - Initial condition of the model cannot be determined due to limited observability

- Generality
  - The capability to explain not only training data, but also testing data

- Identifiability
  - Model parameters can be identified from data

- Interpretability
  - S^m are meaningful and interpretable by human

Newton vs. Einstein

- The definition of “goodness” is changing over time
- Newtonian physics is suitable for macro level objects at low speed
- $L = L_0\sqrt{1 - \frac{v^2}{c^2}}$
- A model can only be “good” within the context of its designated application

Modeling methodologies

- Bottom-up modeling
  – "White-box" model
  – Using first principles
  – Pros:
    - Interpretable
    - Convincing
  – Cons:
    - State space explosion
    - Difficult to be general
    - Low identifiability

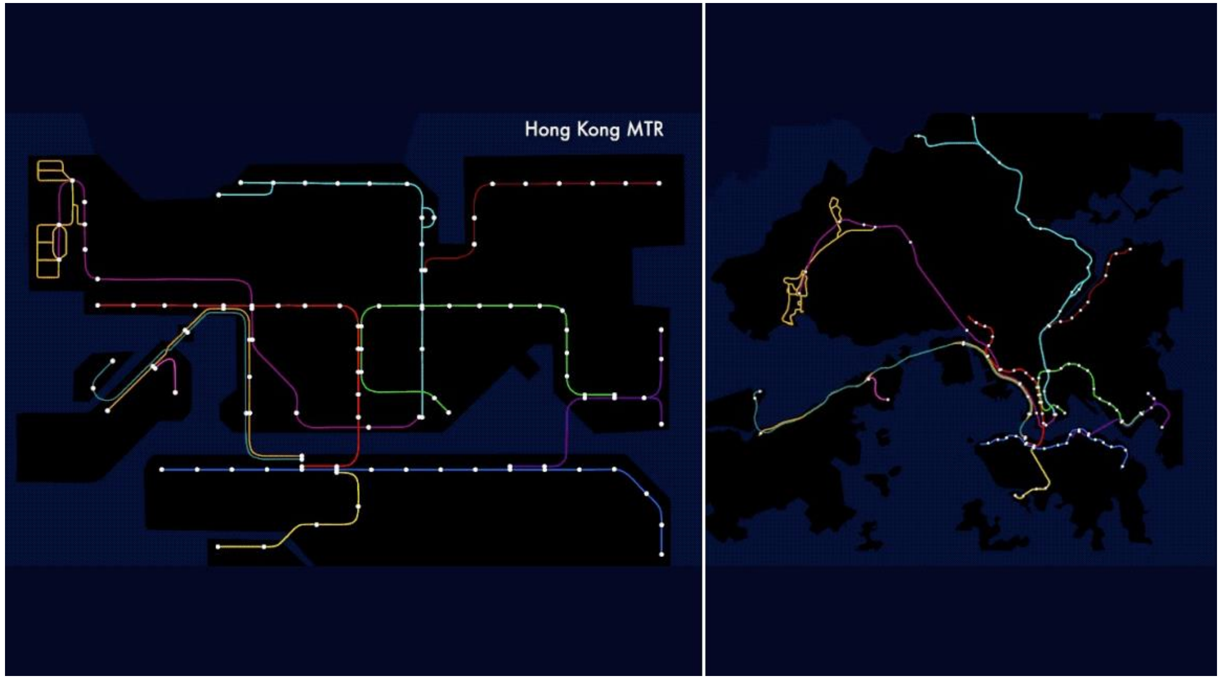
- Data driven models (i.e. Neural networks)
  – "Black-box" model
  – From observable data
  – Pros:
    - No need to know domain knowledge
  – Cons:
    - Large and uninterpretable S^m
    - Depends highly on the quality and quantity of data

The Blind Men and the Elephant

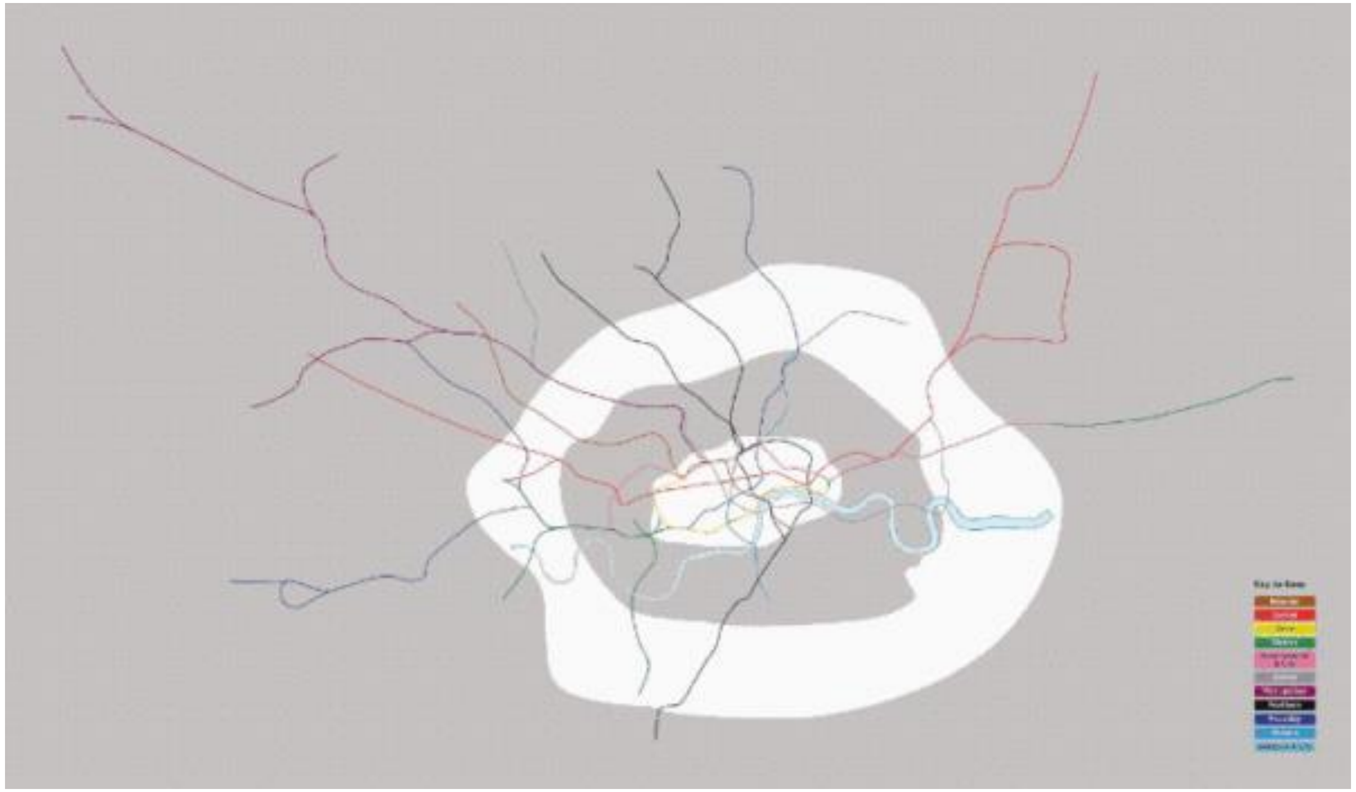
- One can only obtain partial information
- On the other hand, you don’t need full information to understand a system
- How do we extract the least amount of information that are useful?

Abstraction: extracting useful information

- Retain only the topological information
- Exact location/route does not matter



More Abstraction: London MTR



Formal Verification/Validation

Grand challenge:

Automate the process as much as possible !

Analysis Techniques

- Dynamic Analysis (runtime)
  - Execute the system, possibly multiple times with different inputs
  - Check if every execution meets the desired requirement

- Static Analysis (design time)
  - Analyze the source code or the model for possible bugs

- Trade-offs
  - Dynamic analysis is incomplete, but accurate (checks real system, and bugs discovered are real bugs)
  - Static analysis can catch design bugs early !
  - Many static analysis techniques are not scalable (solution: analyze approximate versions, can lead to false warnings)

Verification Methods

- Simulation
  - Simulate the model, possibly multiple times with different inputs
  - Easy to implement, scalable, but no correctness guarantees

- Proof based
  - Construct a proof that system satisfies the invariant
  - Requires manual effort (partial automation possible)

- State-space analysis (Model checking)
  - Algorithm explores “all” reachable states to check invariants
  - Not scalable, but current tools can analyze many real-world designs (relies on many interesting theoretical advances)

Different Requirements

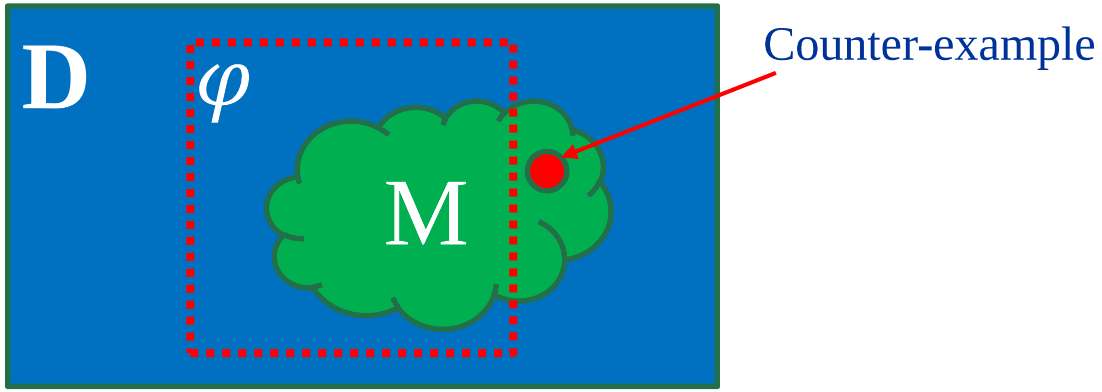
- Safety
  - A system always stays within “good’ states (i.e. a nothing bad ever happens)
  - Leader election: it is never the case that two nodes consider them to be leaders
  - Collision avoidance: Distance between two cars is always greater than some minimum threshold

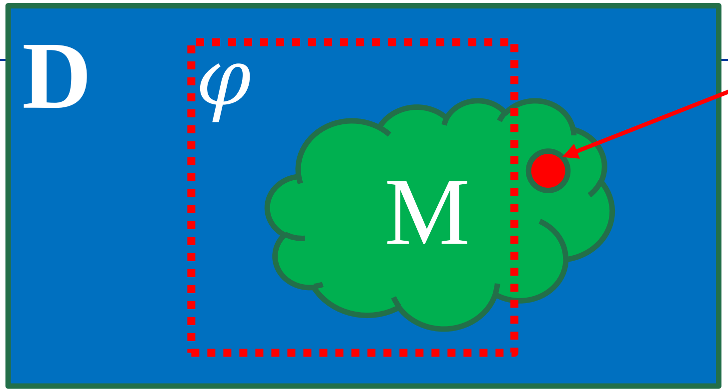
- Liveness
  - System eventually attains its goal
  - Leader election: Each node eventually makes a decision
  - Cruise controller: Actual speed eventually equals desired speed
  - A car will always eventually reach its destination

Model Checking

Model Checking

- A domain D representing the state space of a model
- The reachable state space M for the model
- Define a subset of the state space as property φ
- Explore the whole reachable state space of a model for property violations





Counter-example

Plato vs. Diogenes

- The definition of "*human*"

φ

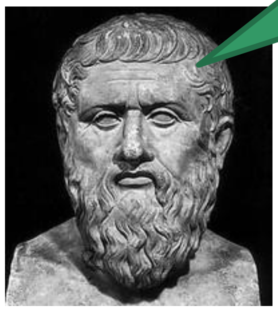
All living creatures D

$Counter - example$

Featherless Biped

M

Here's Plato's human!!!!



Plato

Diogenes

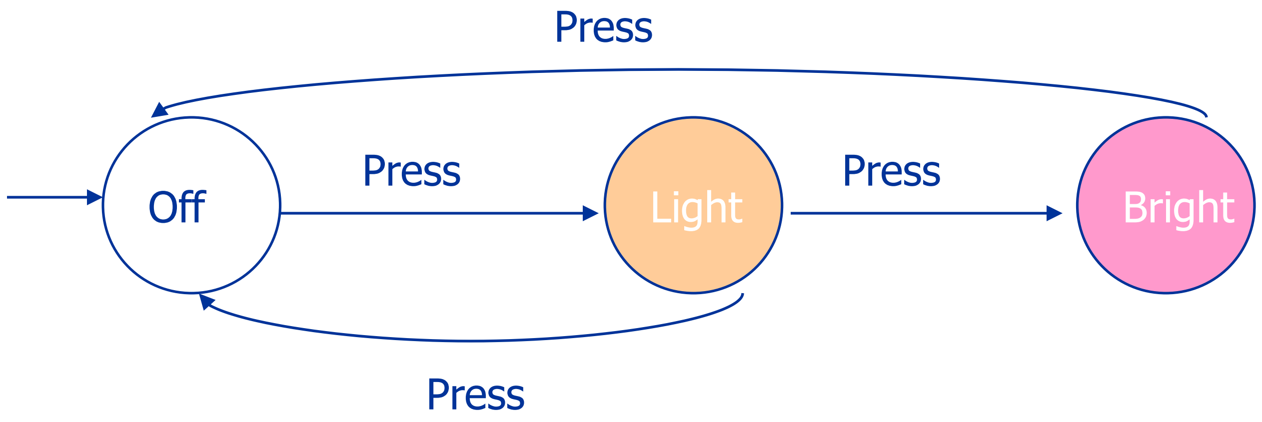
Challenge

- State space explosion
  - Not every model can be model checked!!
  - i.e. Real-number (continuous) state space
  - Complex dynamics between states

- Solution
  - Simple yet expressive formalisms
  - Symbolic states/executions
  - Model abstraction/approximation

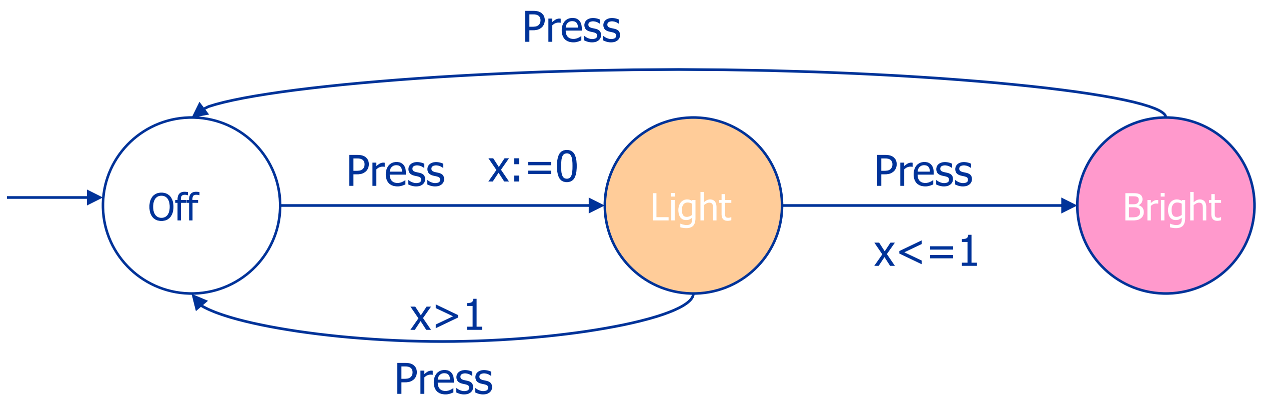
Simple yet expressive formalisms

Basic Finite State Machine (FSM)



WANT: if press is issued twice quickly
then the light will get brighter; otherwise the light is turned off.

FSM with real number time

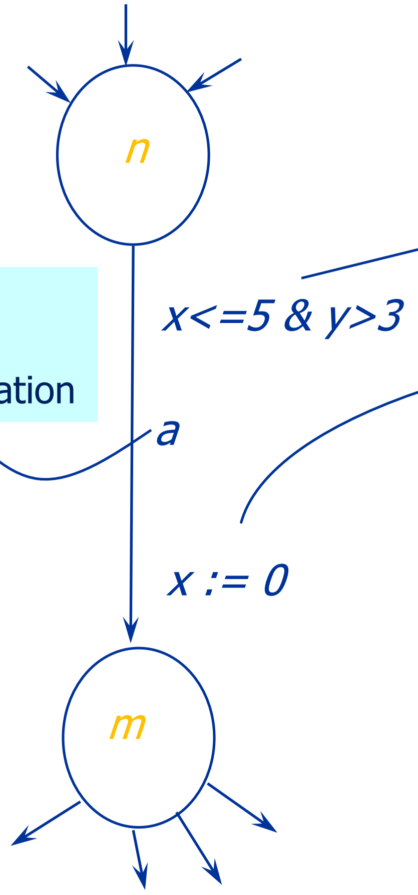


Solution: Add a real-valued clock x

Adding continuous variables to state machines

Timed Automata

Clocks: *x, y*



Action
used
for synchronization

Guard
Boolean combination of comparisons with
Integer/rational bounds

Reset
Action performed on clocks

State

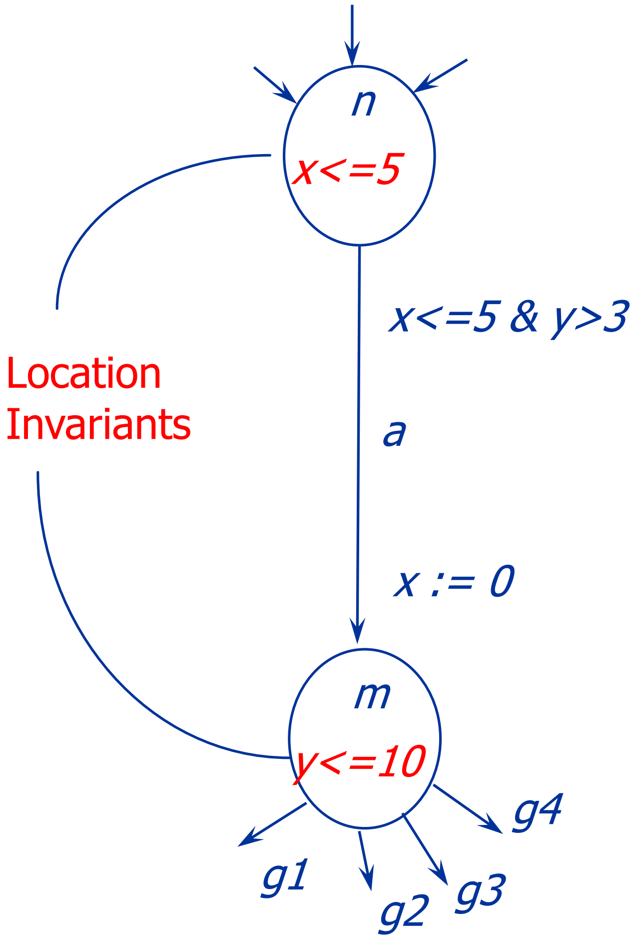
(*location* , *x*=v , *y*=u) where v,u are in $\mathbb{R}$

Transitions

(*n* , *x*=2.4 , *y*=3.1415) $\xrightarrow{a}$ (*m* , *x*=0 , *y*=3.1415)

(*n* , *x*=2.4 , *y*=3.1415) $\xrightarrow{wait(1.1)}$ (*n* , *x*=3.5 , *y*=4.2415)

Adding Invariants



Clocks: x, y

Transitions

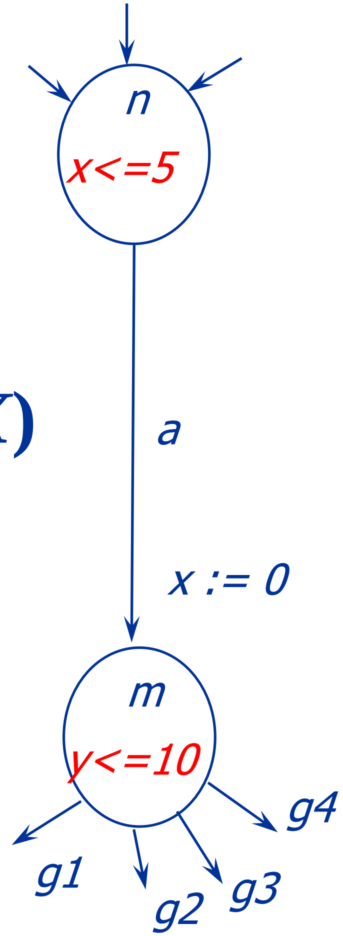
$(\,n\,,\ x{=}2.4\,,\ y{=}3.1415\,) \xrightarrow{\cancel{wait(3.2)}}$

$(\,n\,,\ x{=}2.4\,,\ y{=}3.1415\,) \xrightarrow{wait(1.1)} (\,n\,,\ x{=}3.5\,,\ y{=}4.2415\,)$

Invariants ensure progress!!

Timed Automata: Syntax

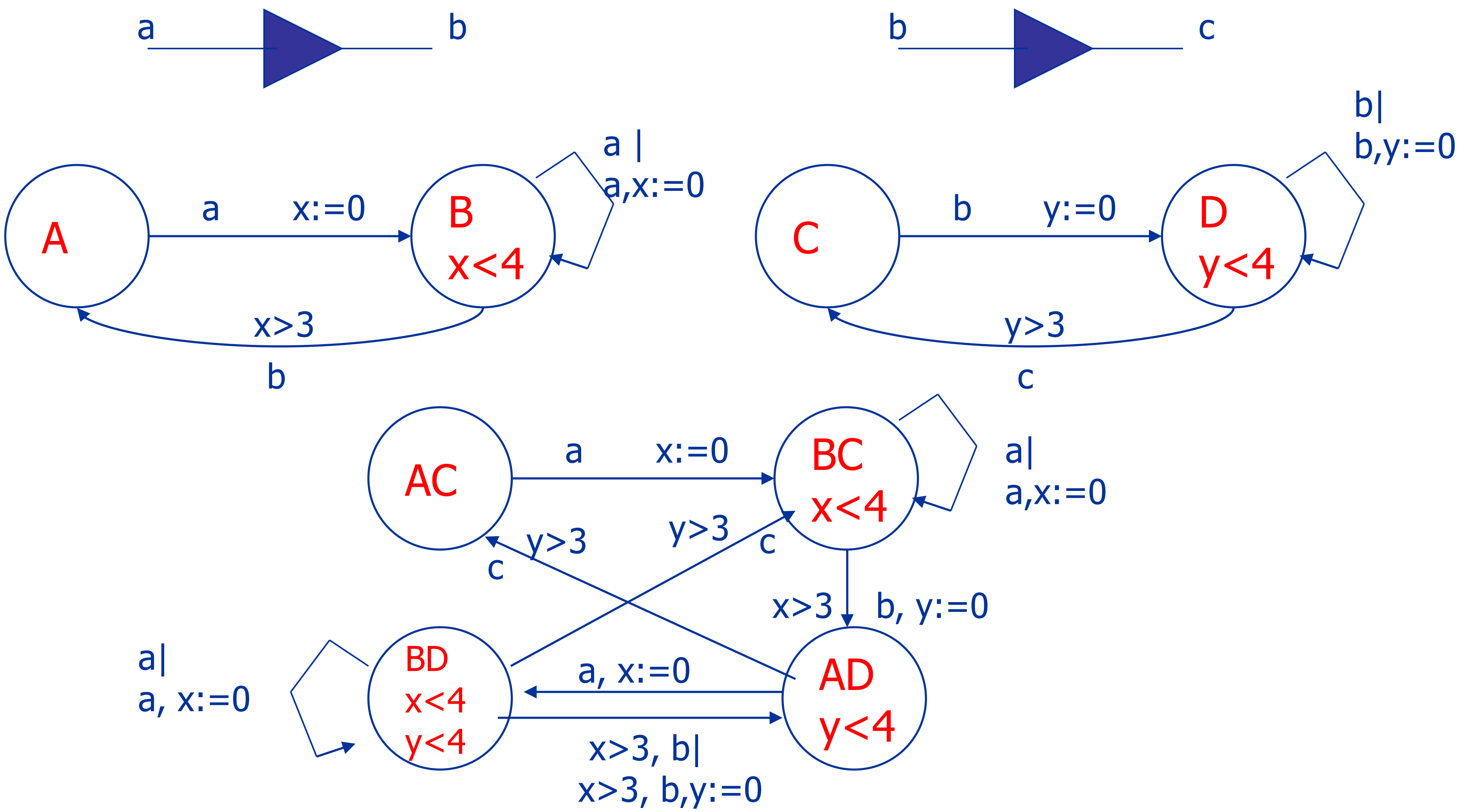
- A finite set V of locations
- A subset V^0 of initial locations
- A finite set Σ of labels (alphabet)
- A finite set X of clocks
- Invariant $Inv(l)$ for each location: (clock constraint over X)
- A finite set E of edges. Each edge has
  - source location l, target location l'
  - label a in Σ (ε labels also allowed)
  - guard g (a clock constraint over X)
  - a subset λ of clocks to be reset



Timed Automata: Semantics

- **For a timed automaton *A*, define an infinite-state transition system *S(A)***
- **States *Q*: a state *q* is a pair *(l,v)*, where *l* is a location, and *v* is a clock vector, mapping clocks in *X* to *R*, satisfying *Inv(l)***
- ***(l,v)* is initial state if *l* is in V^0 and *v(x)=0***
- **Elapse of time transitions: for each nonnegative real number *d*, *(l,v)-d->(l,v+d)* if both *v* and *v+d* satisfy *Inv(l)***
- **Location switch transitions: *(l,v)-a->(l',v')* if there is an edge *(l,a,g,λ,l')* such that *v* satisfies *g* and *v'=v[λ:=0]***

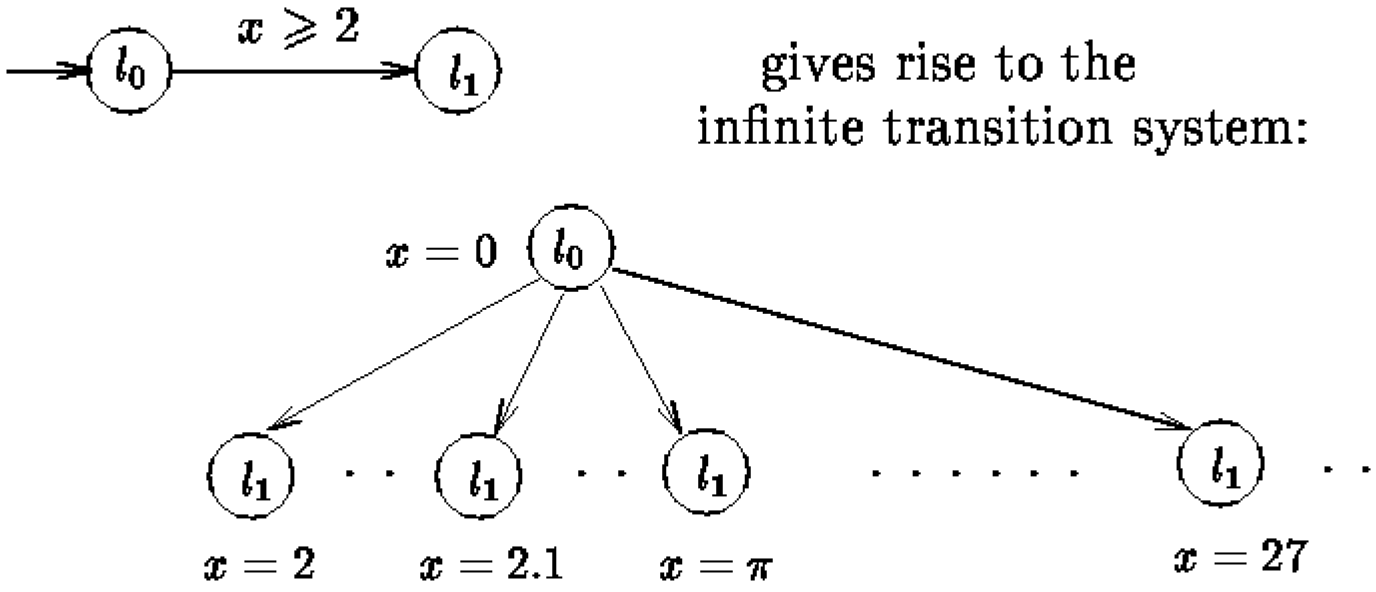
Product Construction



Model Checking: Forward Reachability

- Given a timed automata and a property φ
- R:=I
- Repeat
  - If R intersects $\neg\varphi$, report “yes”
  - Else if R contains Post(R), report “no”
  - Else R := R union Post(R)

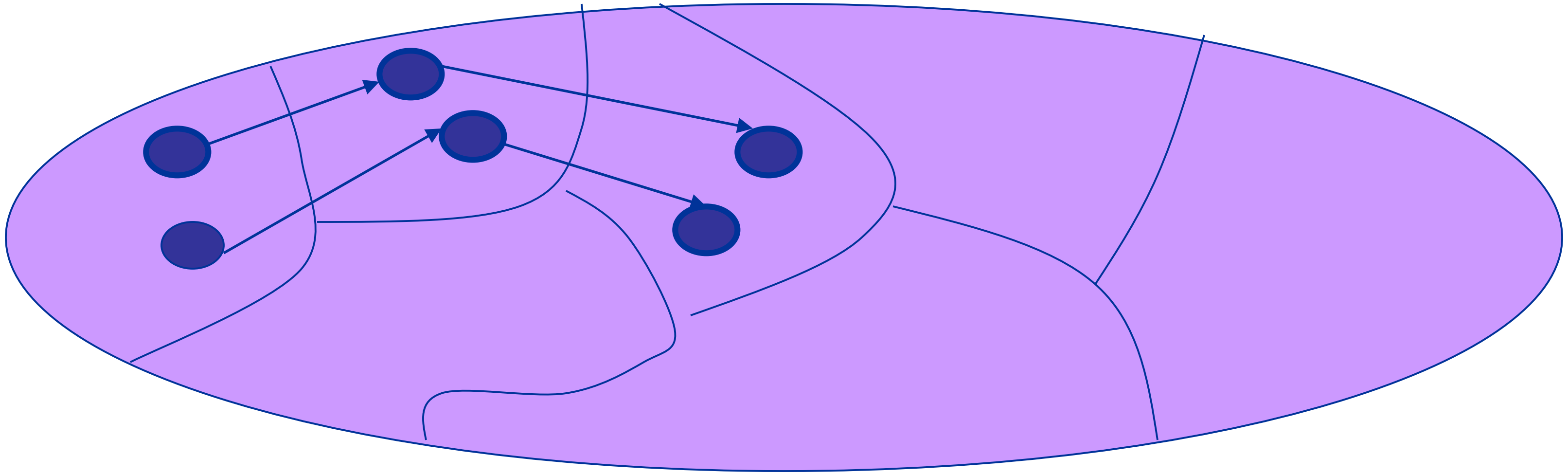
Reachability for Timed Automata



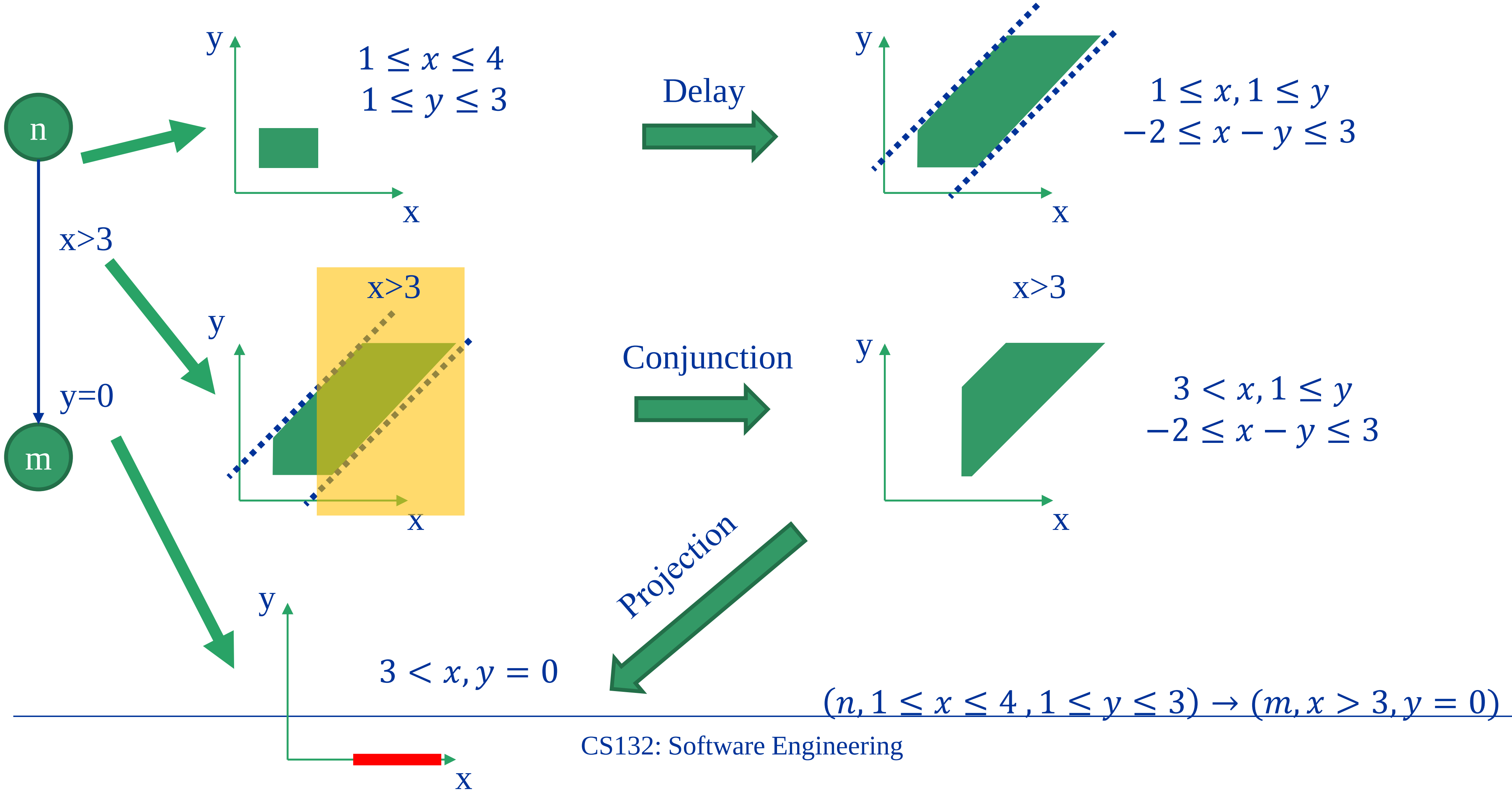
Symbolic states/executions

Finite Partitioning

Goal: To partition state-space into finitely many equivalence classes so that equivalent states exhibit similar behaviors

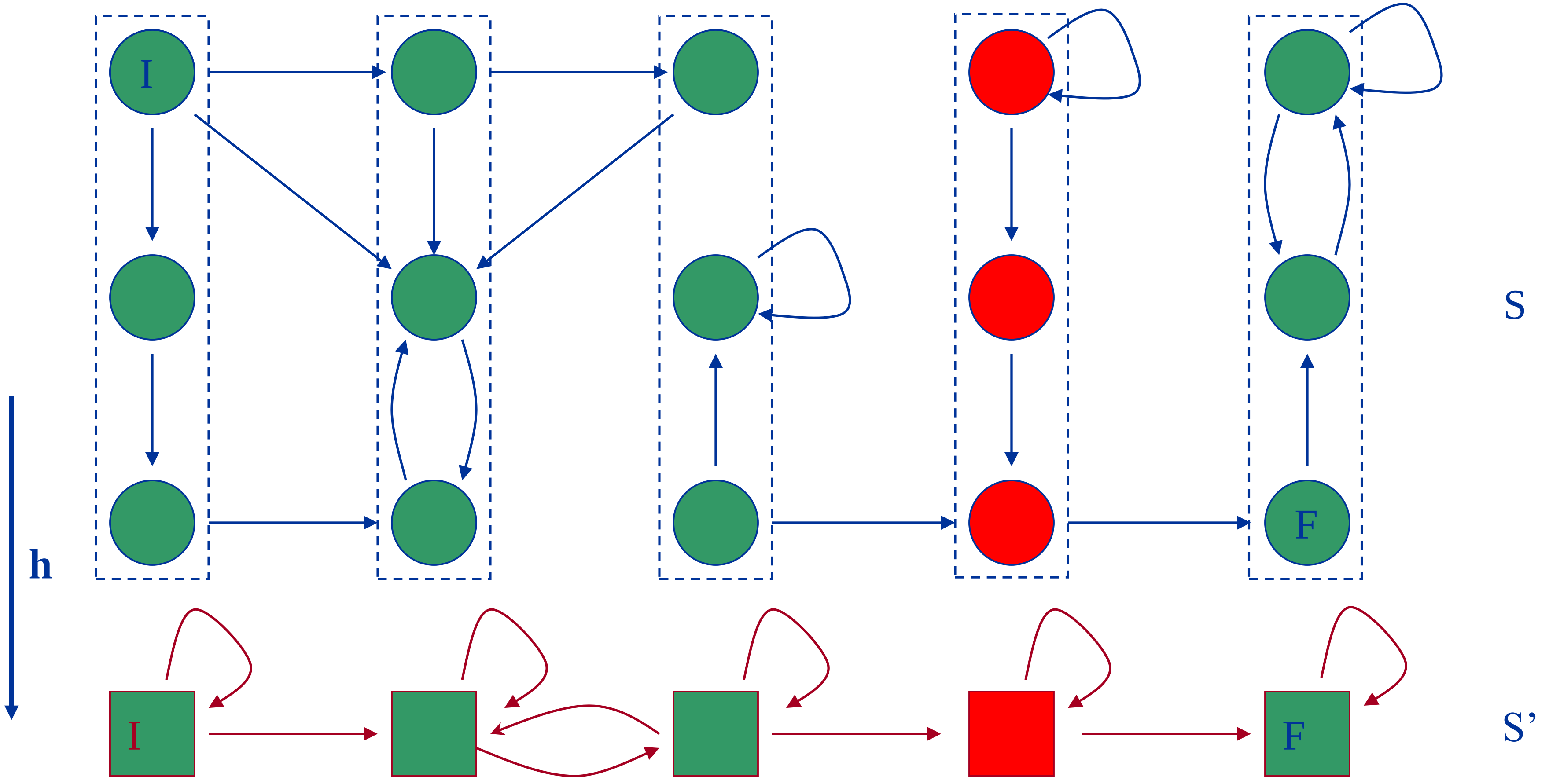


Symbolic States/executions



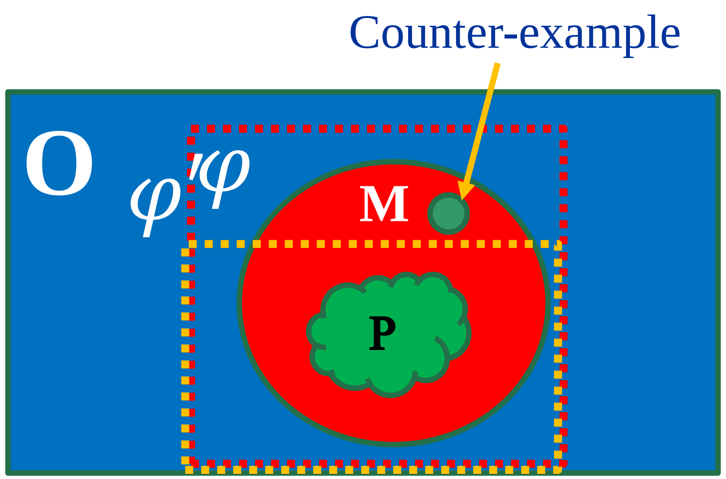
Model abstraction/approximation

Existential Abstraction (Over-approximation)

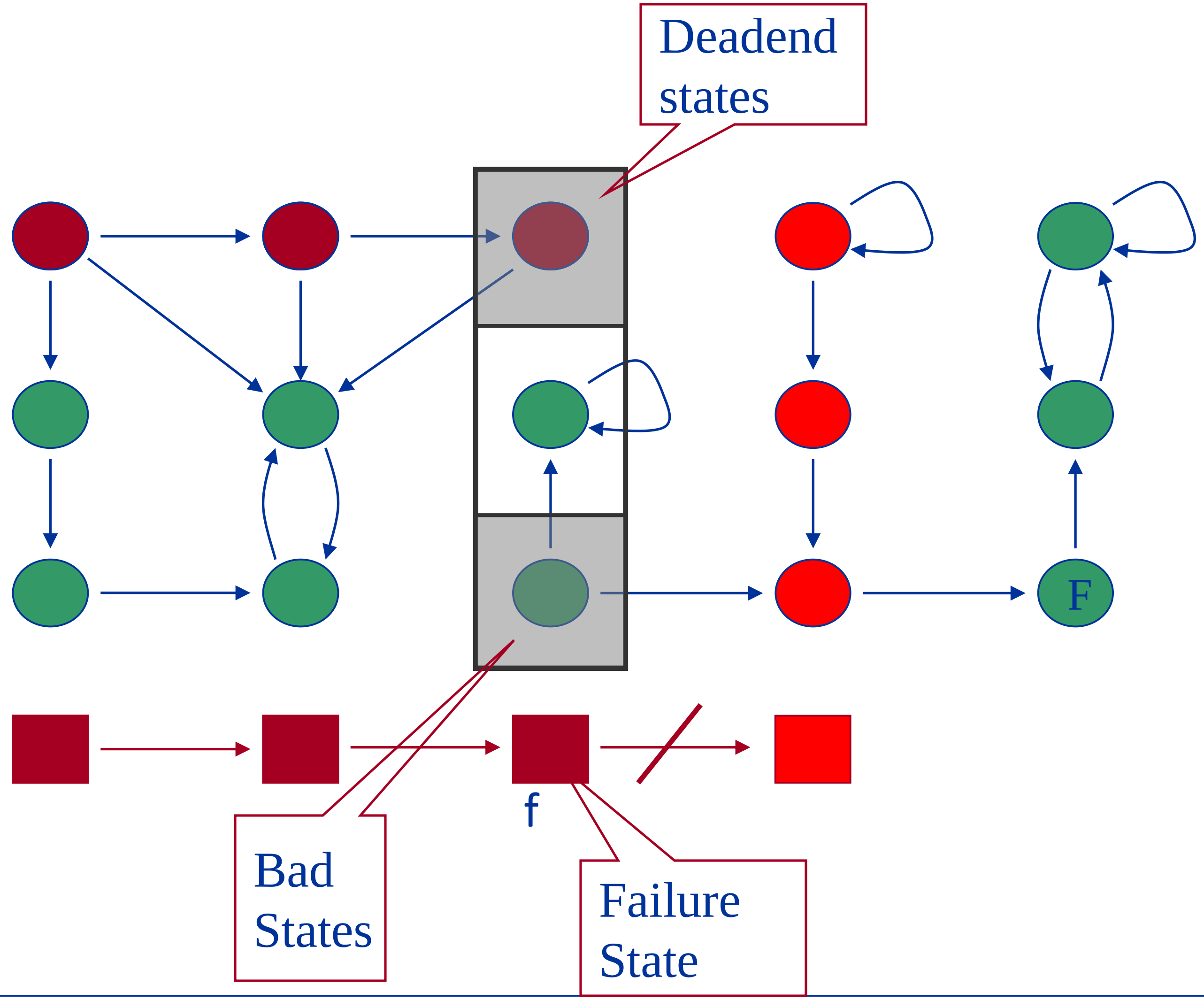


Pros and Cons of Over-approximation

- Properties satisfied by M are also satisfied by P
  - Can model check a less complex model
- M has more behaviors than P
- If a counter-example returns, it may not be a behavior of P



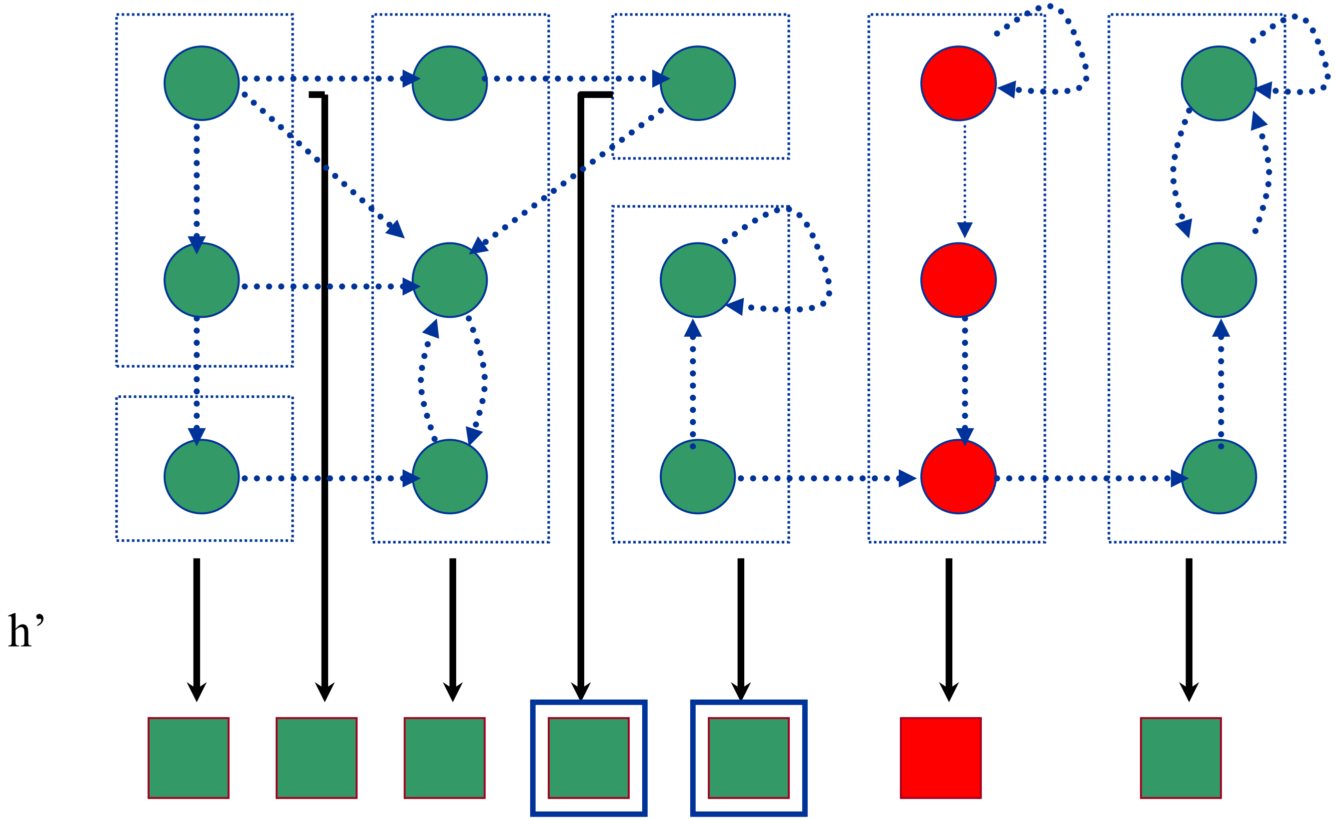
Why spurious counterexample?



Refinement

- Problem: Deadend and Bad States are in the same abstract state.
- Solution: Refine abstraction function.
- The sets of Deadend and Bad states should be separated into different abstract states.

Refinement

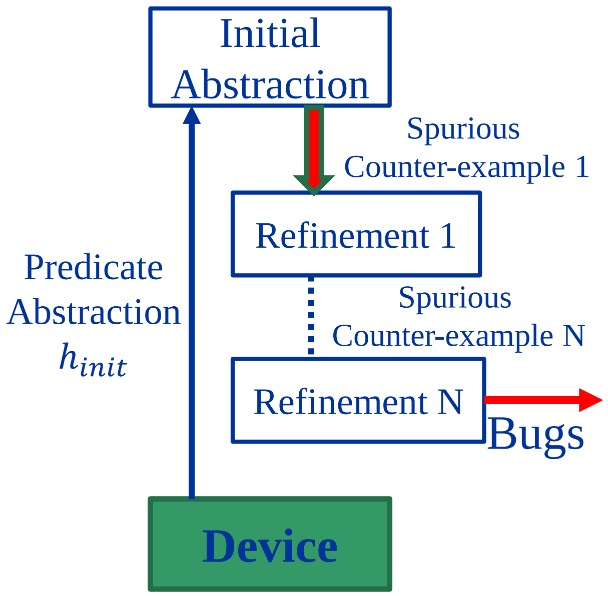


Refinement : h’

Counter-Example-Guided Abstraction and Refinement (CEGAR)

- Obtain initial abstraction

1. Model checking
2. Property satisfied -> no bugs
3. Property unsatisfied -> counter-examples
4. Check whether the CE is spurious
5. If not, bug found
6. If yes, refine the model and start from 1 again



Capture Environmental Variability With Over-approximation

- Properties satisfied by M are also satisfied by P1, P2
- Behaviors not exist in P1, P2 may also be physiologically-valid
- Is this a valid counter-example?
- Need a framework to provide context for counter-examples