Department of Computer Science
National Tsing Hua University

CS 5244: Introduction to Cyber Physical Systems

Unit 16: Reachability Analysis (Ch. 14)

Instructor: Cheng-Hsin Hsu

Acknowledgement: The instructor thanks Profs. Edward A. Lee & Sanjit A. Seshia at UC Berkeley for sharing their course materials

The Challenge of Dependable Software in Embedded Systems

Today's medical devices run on software… software defects can have life-threatening consequences.

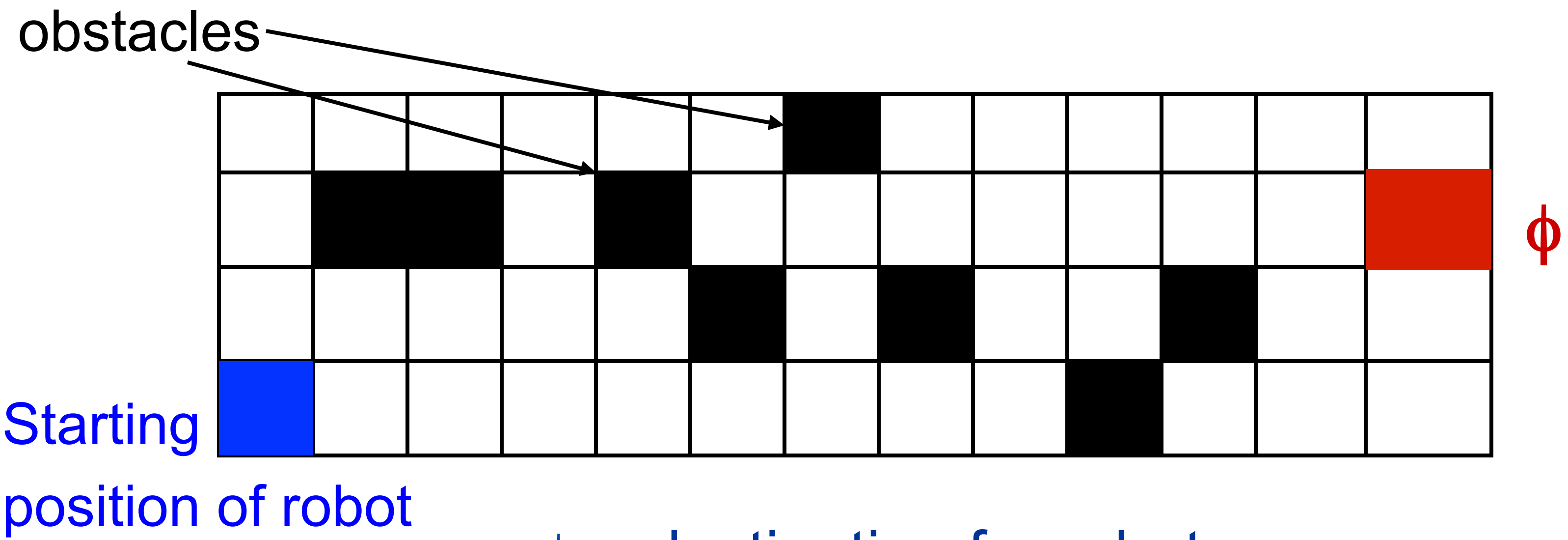
[Journal of Pacing and Clinical Electrophysiology, 2004]

[different device]

"the patient collapsed while walking towards the cashier after refueling his car […] A week later the patient complained to his physician about an increasing feeling of unwell-being since the fall."

"In 1 of every 12,000 settings, the software can cause an error in the programming resulting in the possibility of producing paced rates up to 185 beats/min."

A Robot delivery service, with obstacles



ϕ = destination for robot

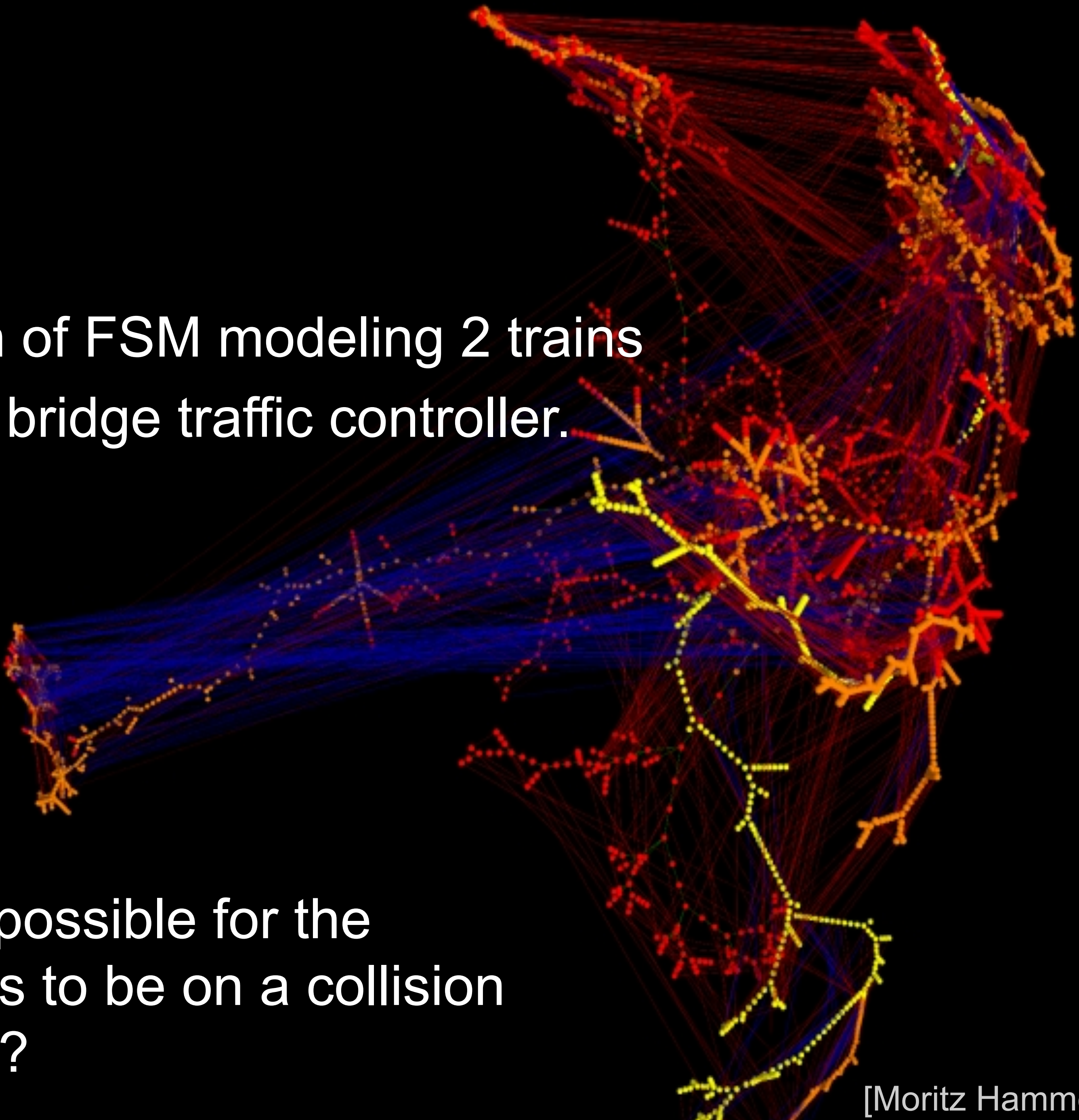
Specification:

The robot eventually reaches ϕ

Suppose there are n destinations $\phi_1, \phi_2, \ldots, \phi_n$

The new specification could be that

The robot visits $\phi_1, \phi_2, \ldots, \phi_n$ in that order



Graph of FSM modeling 2 trains and a bridge traffic controller.

Is it possible for the trains to be on a collision path?

[Moritz Hammer, Uni. Muenchen]

Reachability Analysis and Model Checking

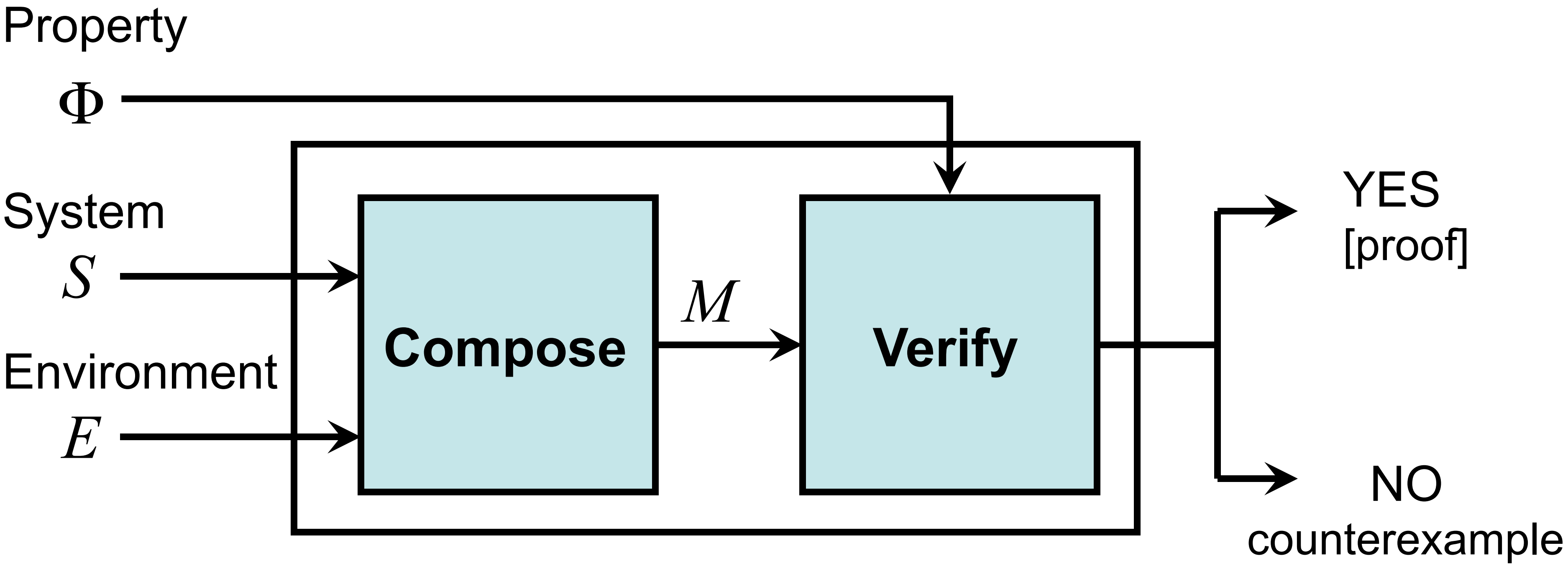
Reachability analysis is the process of computing the set of reachable states for a system.

- all three problems can be solved using reachability analysis

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

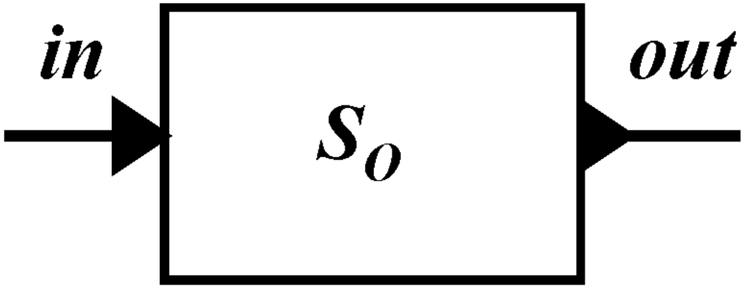
Model checking typically performs reachability analysis.

Formal Verification

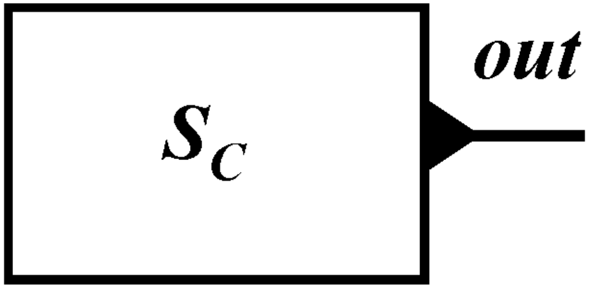


Open vs. Closed Systems

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

Model Checking G *p*

Consider an LTL formula of the form **G**p where p is a proposition (p is a property on a single state)

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

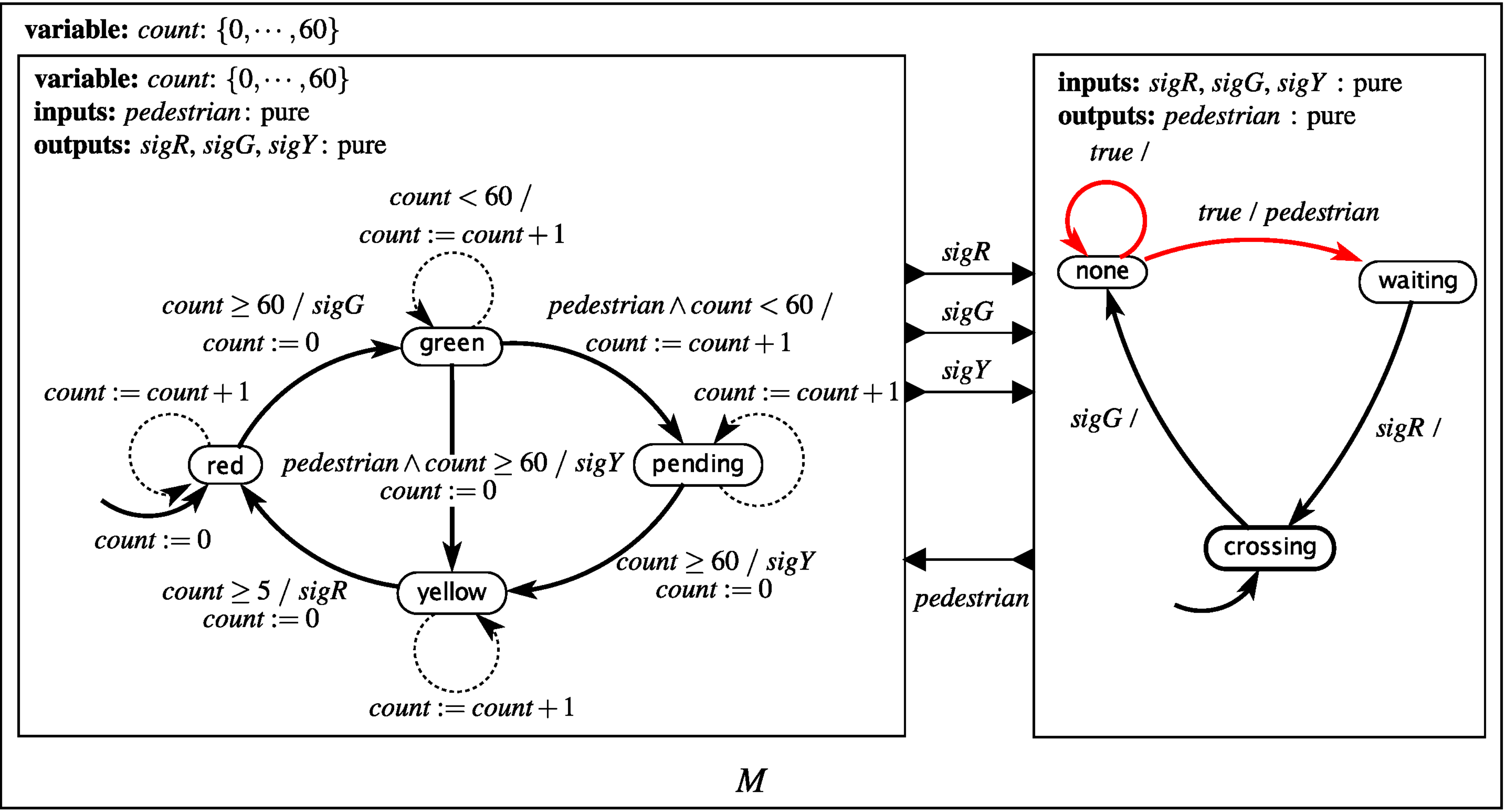
The state space found is typically represented as a directed graph called a state graph.

When M is a finite-state machine, this reachability analysis will terminate (in theory).

In practice, though, the number of states may be prohibitively large consuming too much run-time or memory (the state explosion problem).

Traffic Light Controller Example

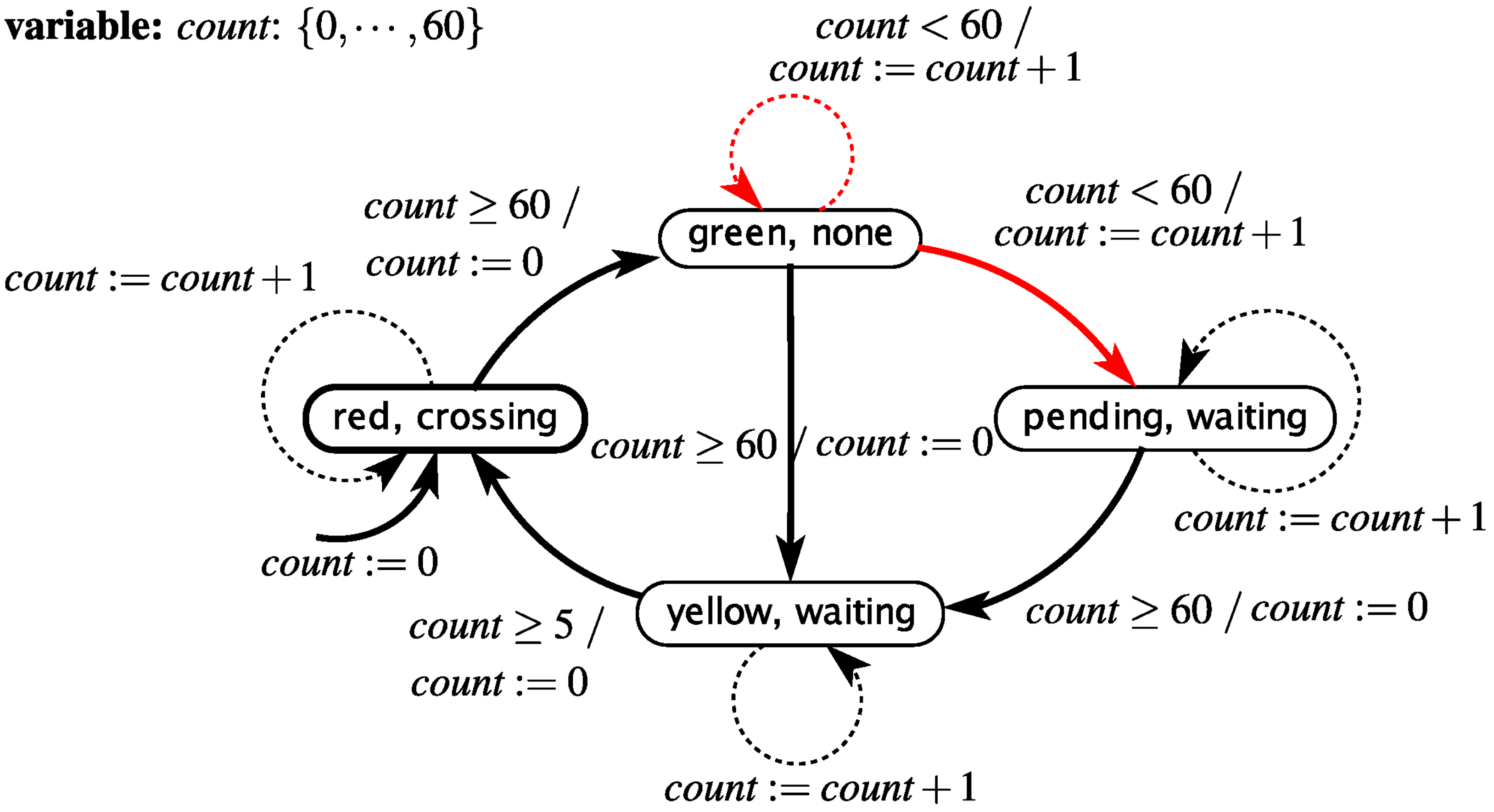
Property: **G** (: (green Æ crossing))



Composed FSM for Traffic Light Controller

Property: **G** (: (green Æ crossing))

This FSM has 188 states (accounting for different values of count)



Reachability Analysis Through Graph Traversal

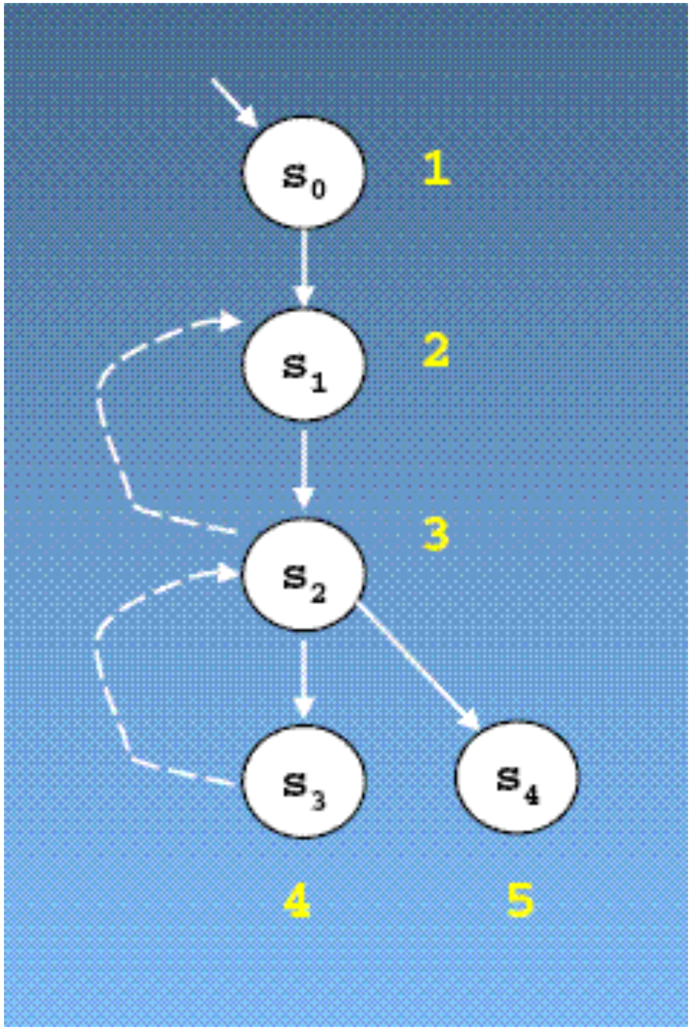
Construct the state graph on the fly

Start with initial state, and explore next states using a suitable graph-traversal strategy.

Depth-First Search (DFS)

Maintain 2 data structures:

1. Set of visited states R
2. Stack with current path from the initial state



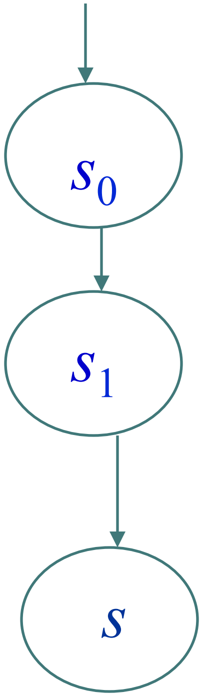
Potential problems for a huge graph?

Generating counterexamples

If the DFS algorithm finds the target (‘error’) state s, how can we generate a trace from the initial state to that state?

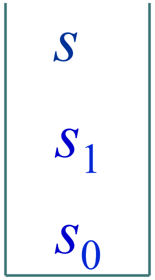
Generating counterexamples

If the DFS algorithm finds the target (‘error’) state s, how can we generate a trace from the initial state to that state?



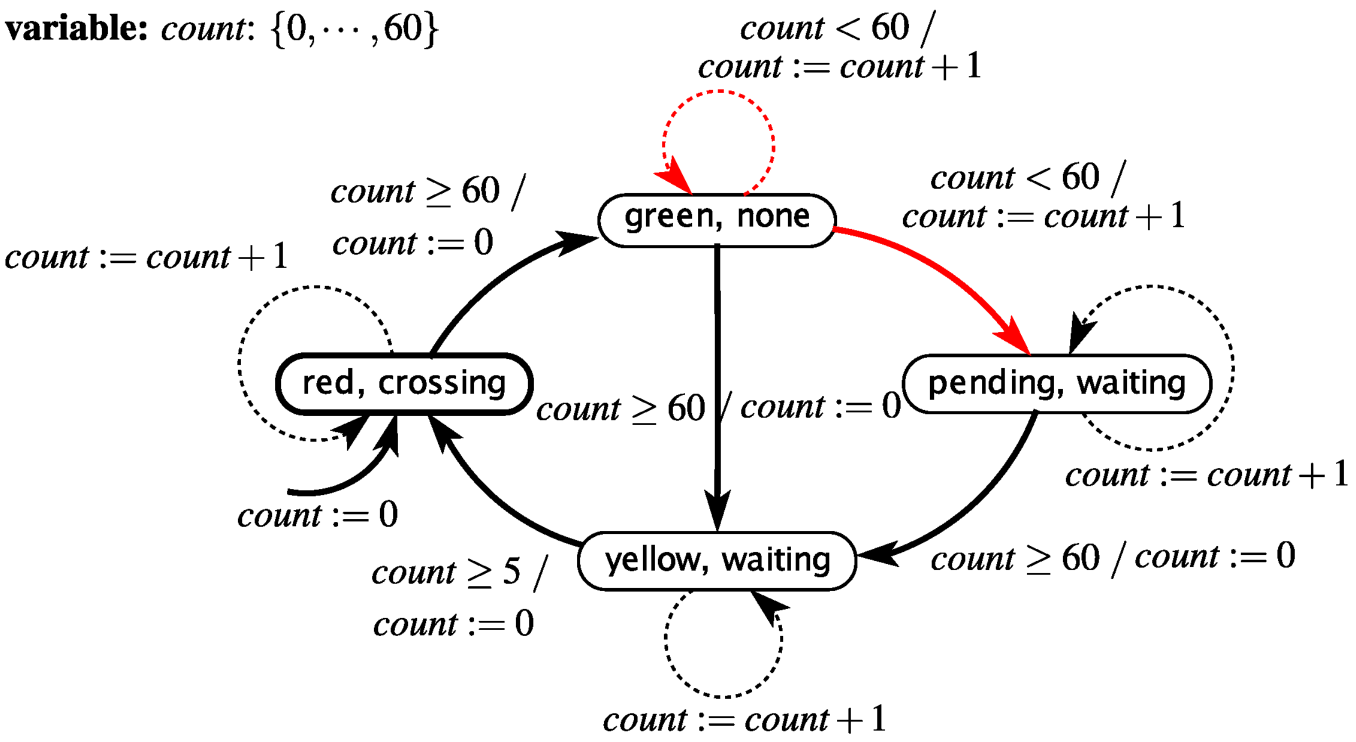
Simply read the trace off the stack

Stack:



Explicit State Model Checking Example

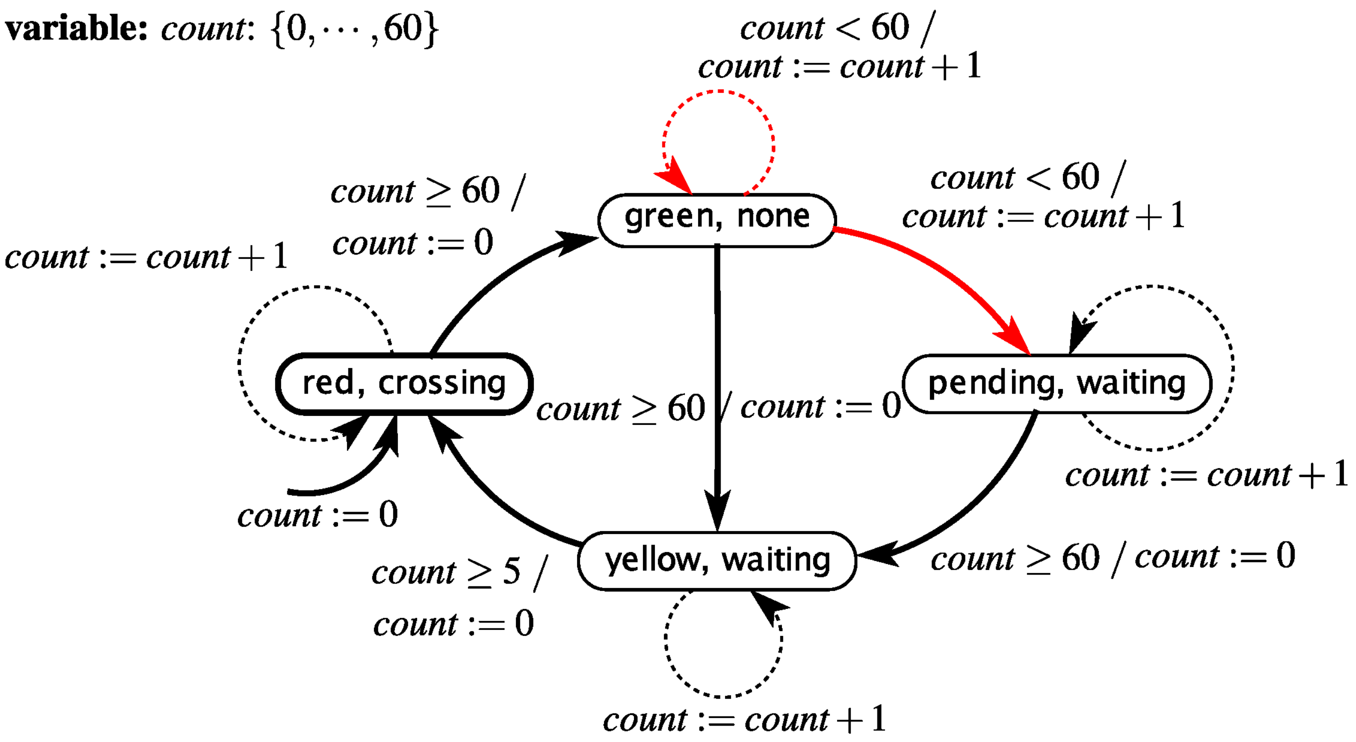
Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0) }

Explicit State Model Checking Example

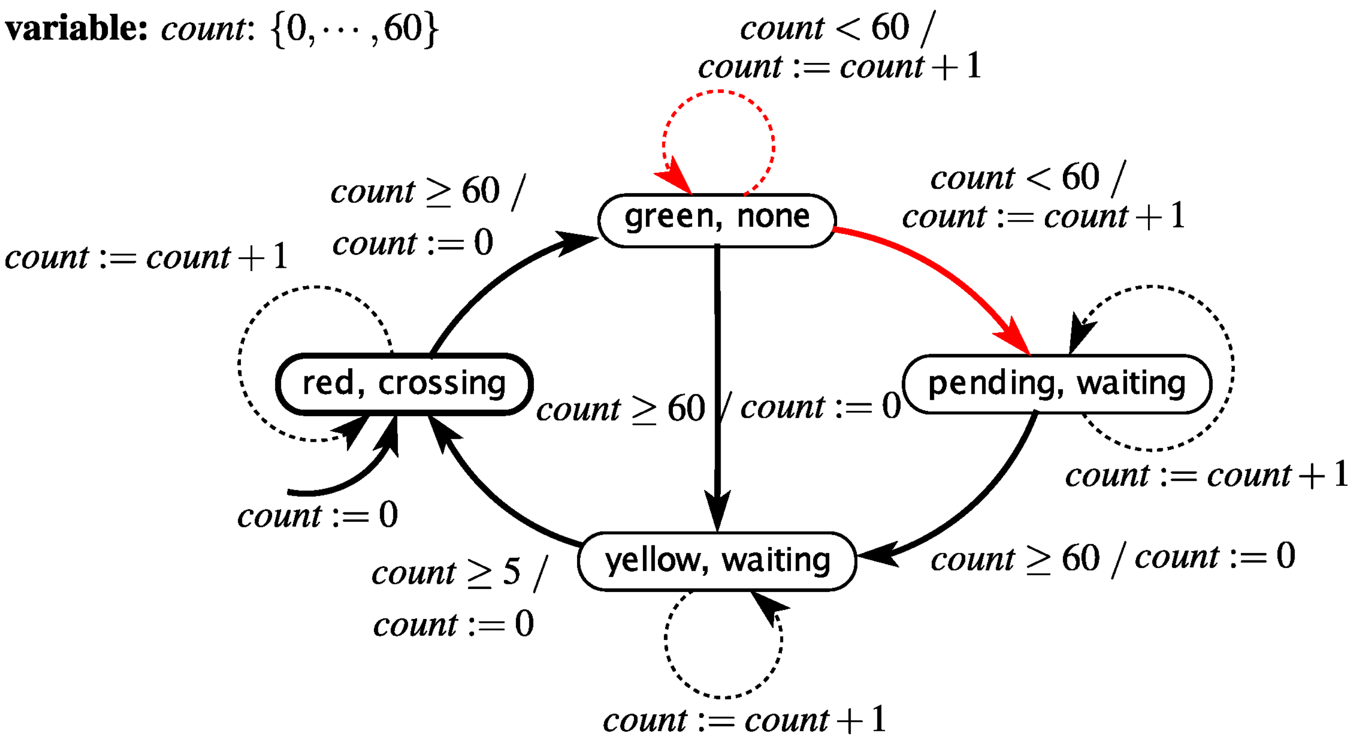
Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0), (red, crossing, 1) }

Explicit State Model Checking Example

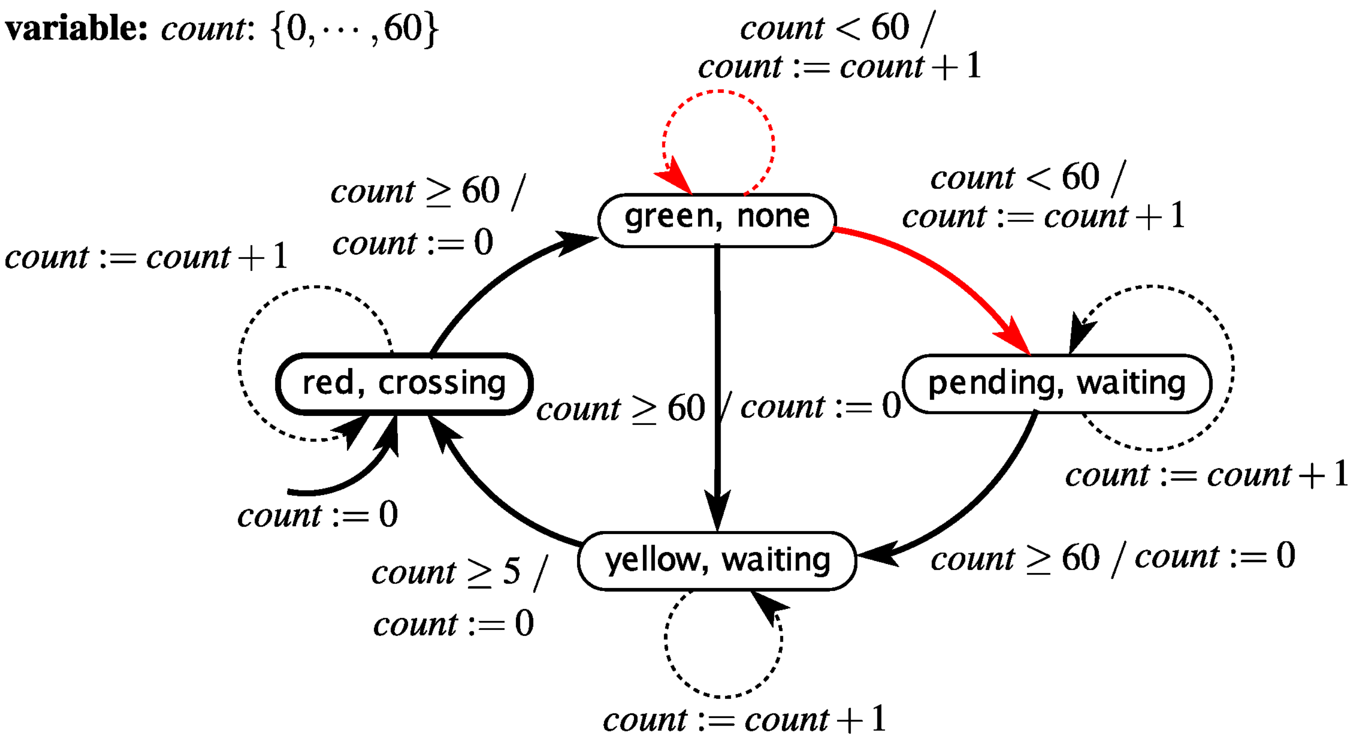
Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0), (red, crossing, 1), … (red, crossing, 60) }

Explicit State Model Checking Example

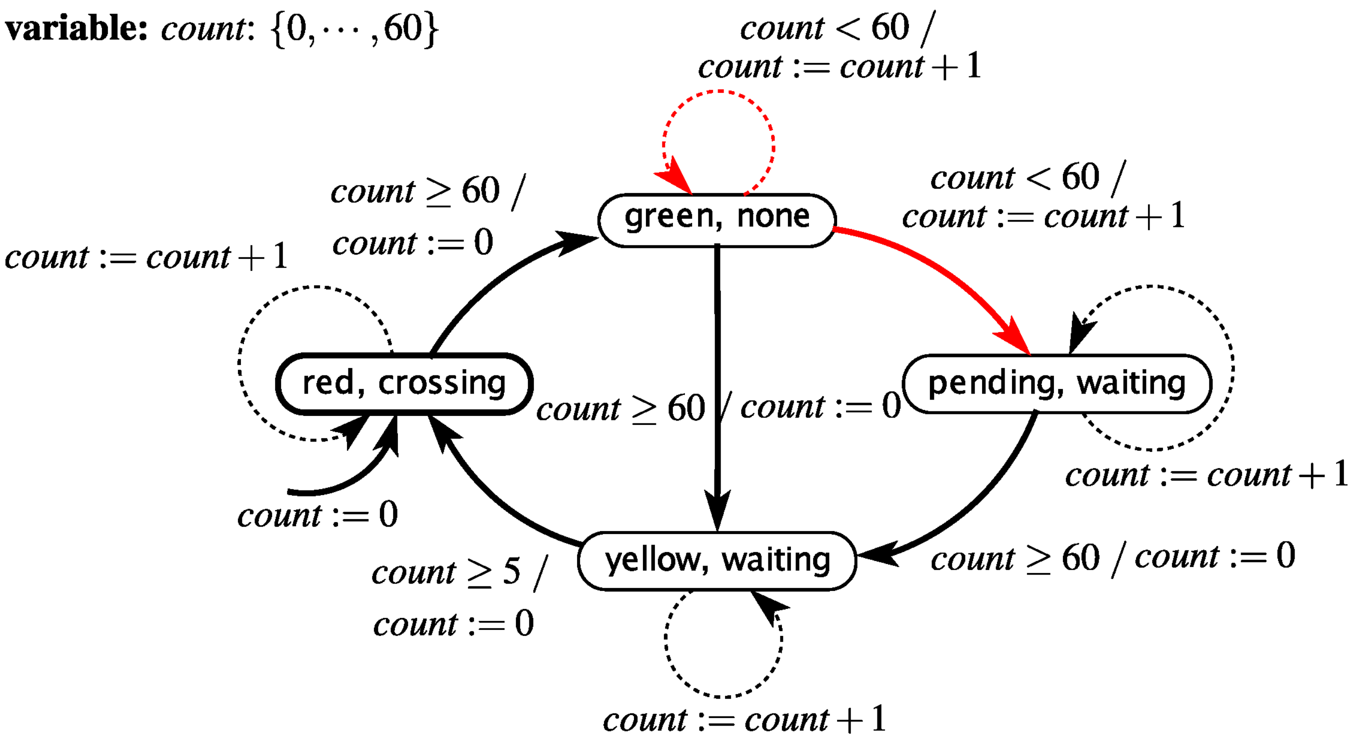
Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0), (red, crossing, 1), … (red, crossing, 60), (green, none, 0) }

Explicit State Model Checking Example

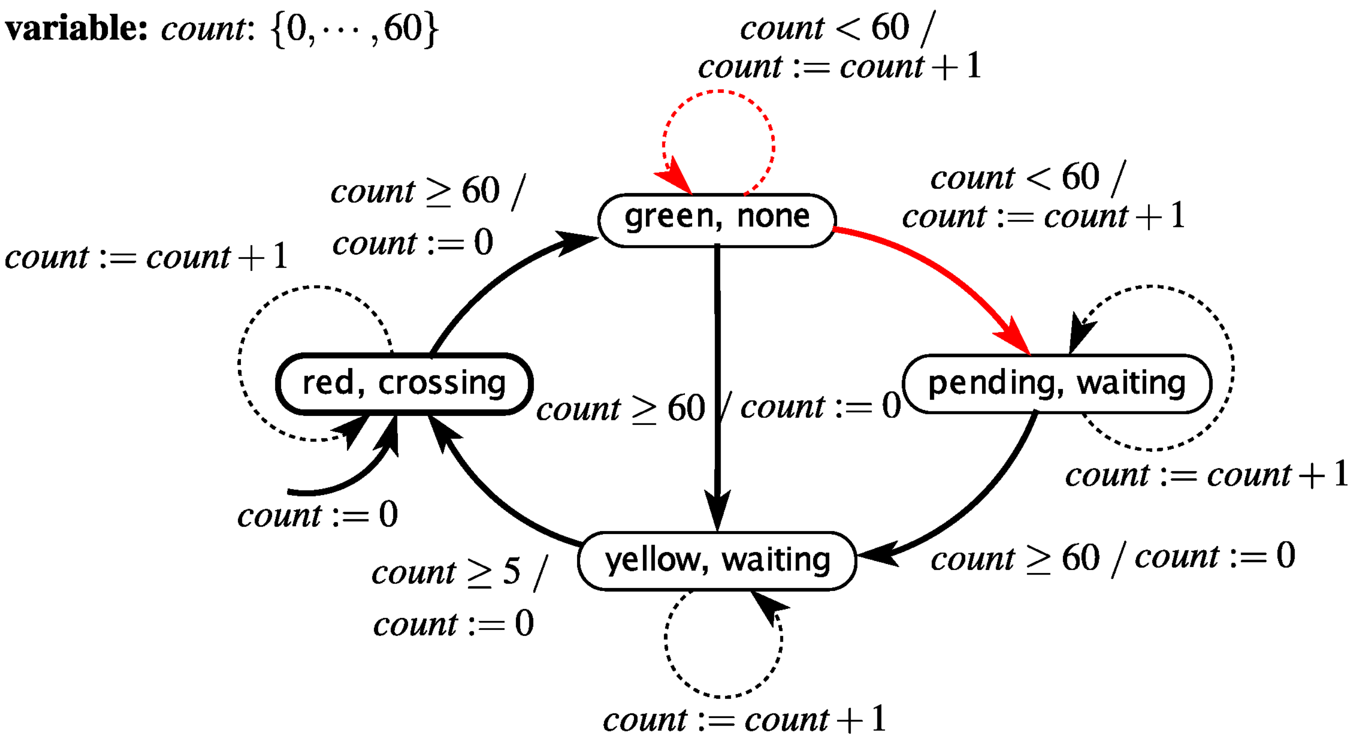
Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0), (red, crossing, 1), … (red, crossing, 60), (green, none, 0), (green, none, 1) }

Explicit State Model Checking Example

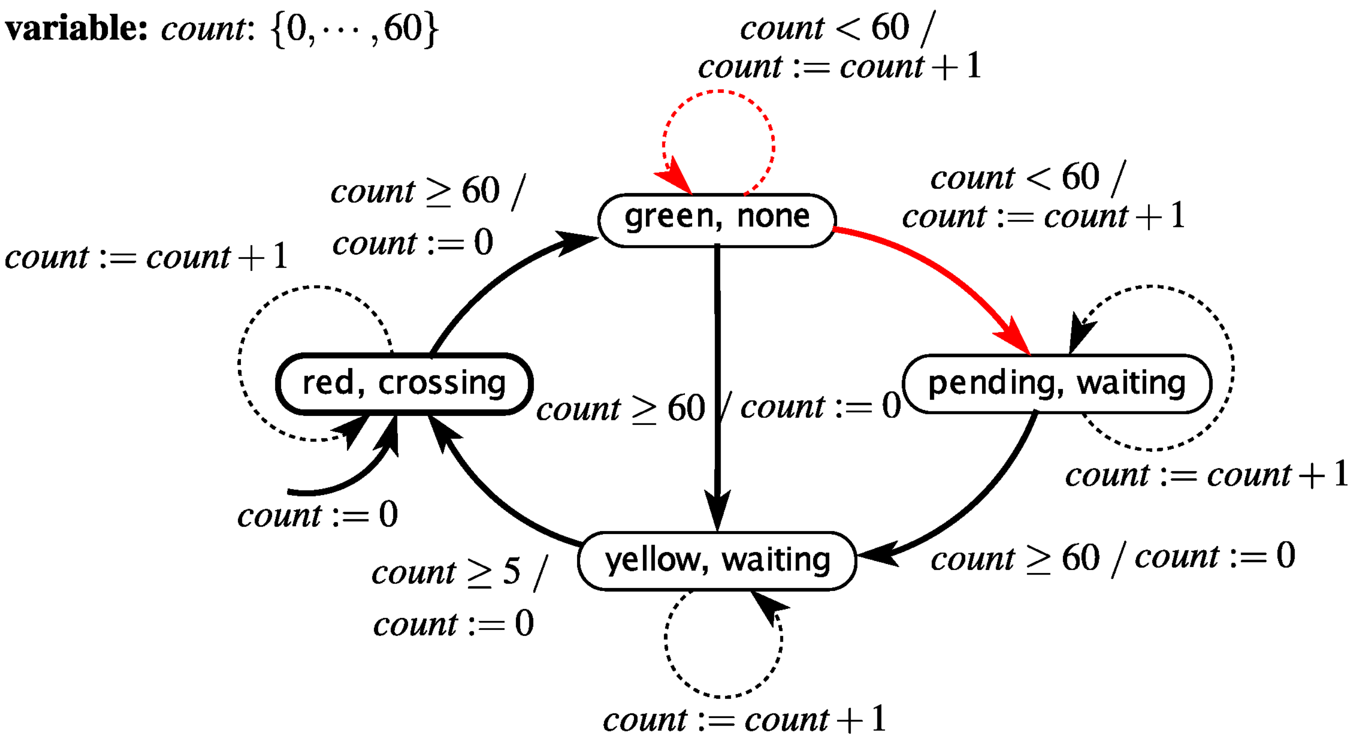
Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0), (red, crossing, 1), … (red, crossing, 60),
(green, none, 0), (green, none, 1), …, (green, none, 60) }

Explicit State Model Checking Example

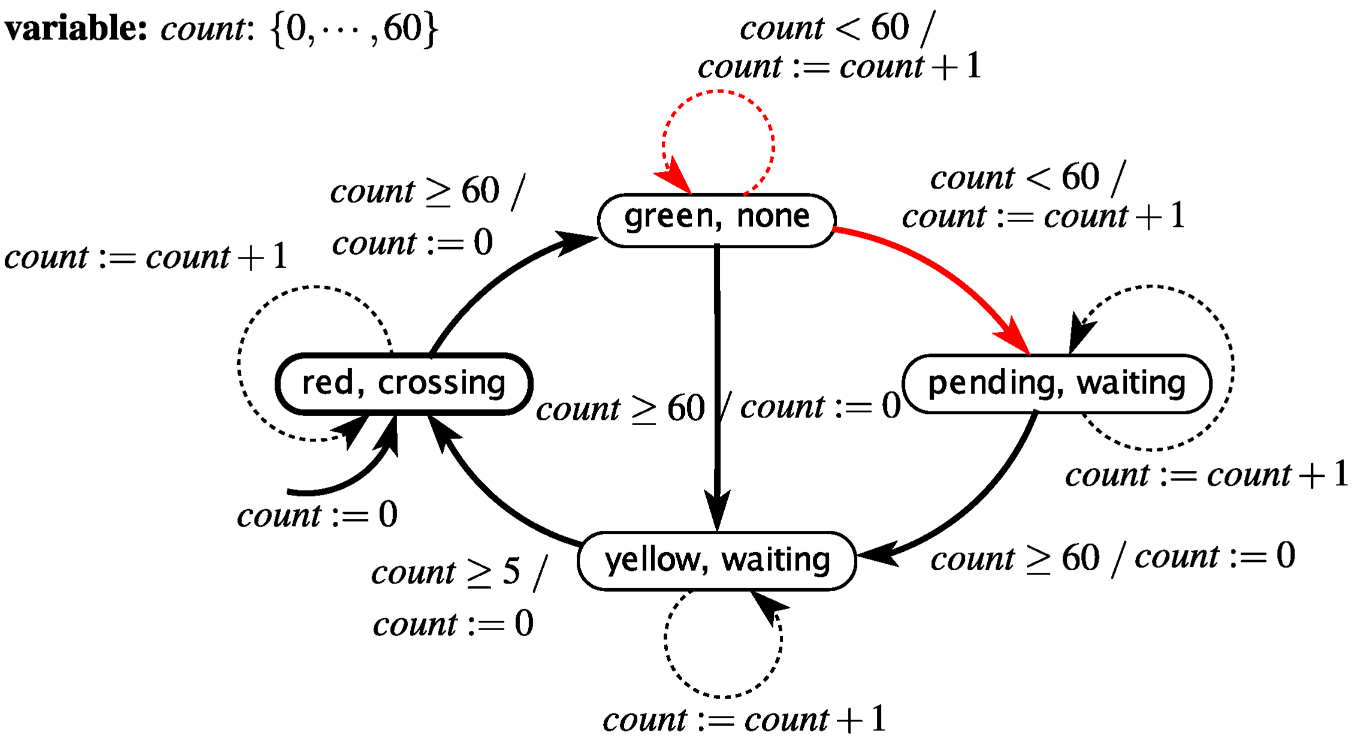
Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0), (red, crossing, 1), … (red, crossing, 60),
(green, none, 0), (green, none, 1), …, (green, none, 60),
(yellow, waiting, 0) }

Explicit State Model Checking Example

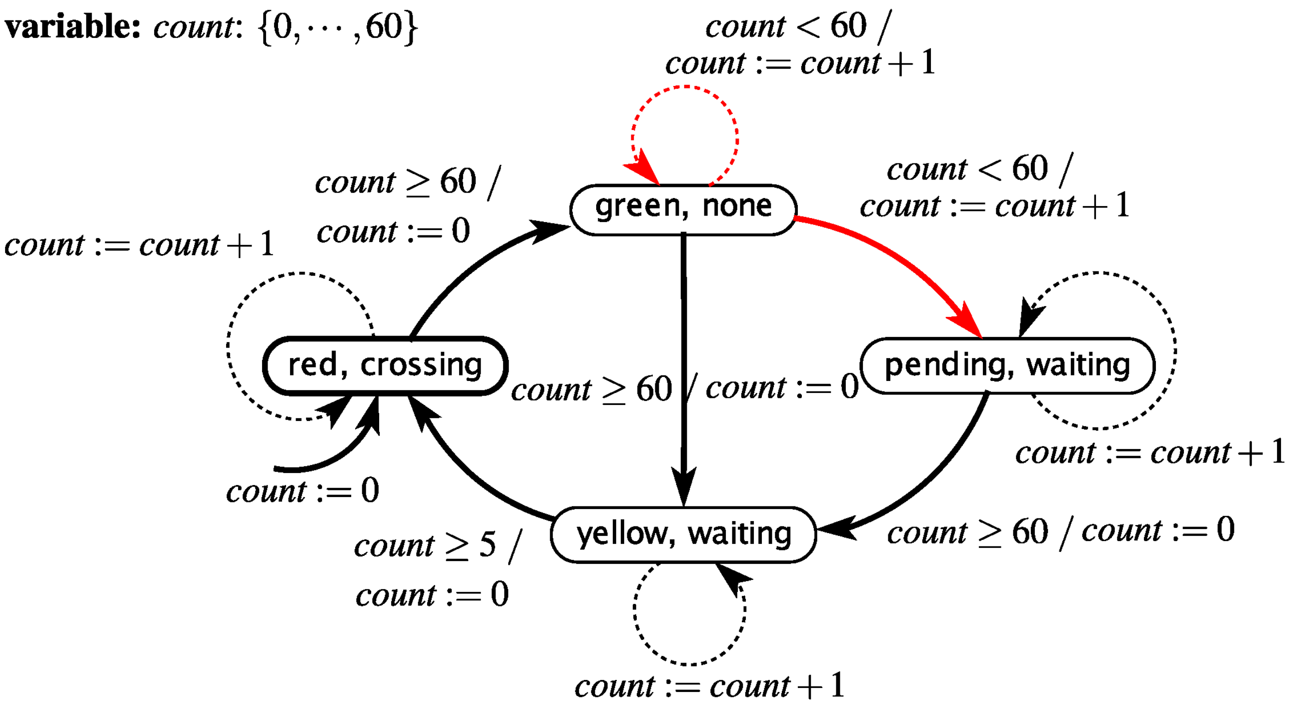
Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0), (red, crossing, 1), … (red, crossing, 60),
(green, none, 0), (green, none, 1), …, (green, none, 60),
(yellow, waiting, 0), … (yellow, waiting, 5) }

Explicit State Model Checking Example

Property: **G** (: (green Æ crossing))



R = { (red, crossing, 0), (red, crossing, 1), … (red, crossing, 60),
(green, none, 0), (green, none, 1), …, (green, none, 60),
(yellow, waiting, 0), … (yellow, waiting, 5),
(pending, waiting, 1), …, (pending, waiting, 60) }

The *Symbolic* Approach

Rather than exploring new reachable states one at a time, we can explore new sets of reachable states

- However, we only represent sets implicitly, as Boolean functions

Set operations can be performed using Boolean algebra

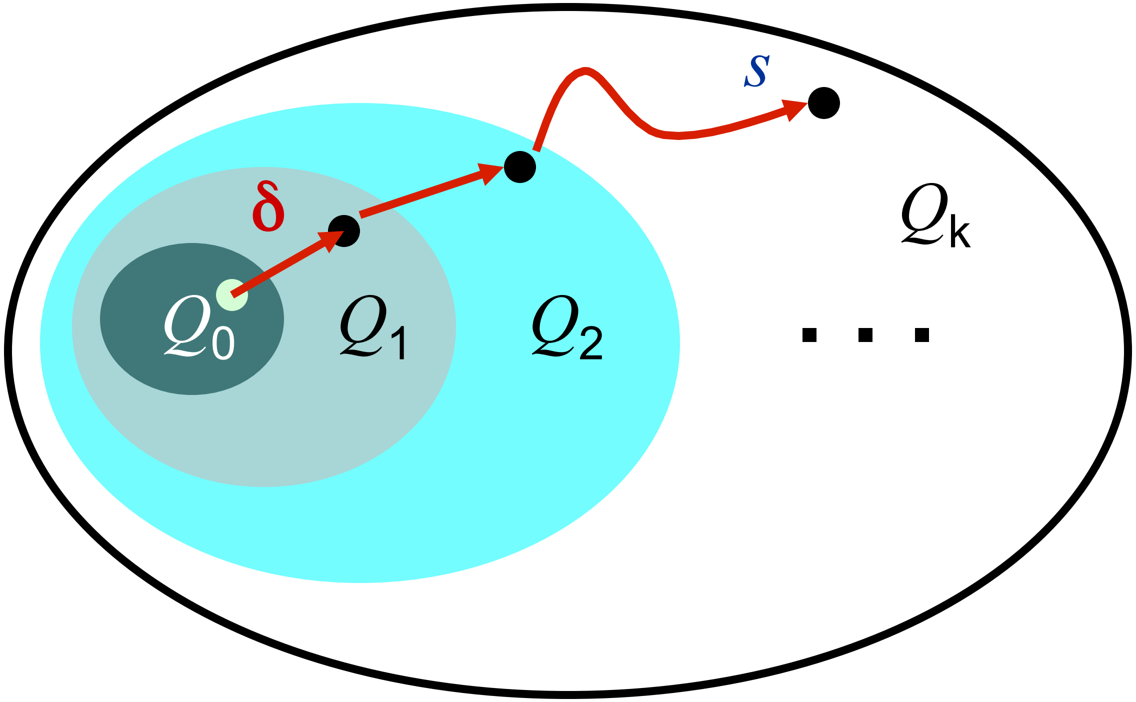
Represent a finite set of states S by its characteristic Boolean function f_S

- $f_S(x) = 1$ iff $x \in S$

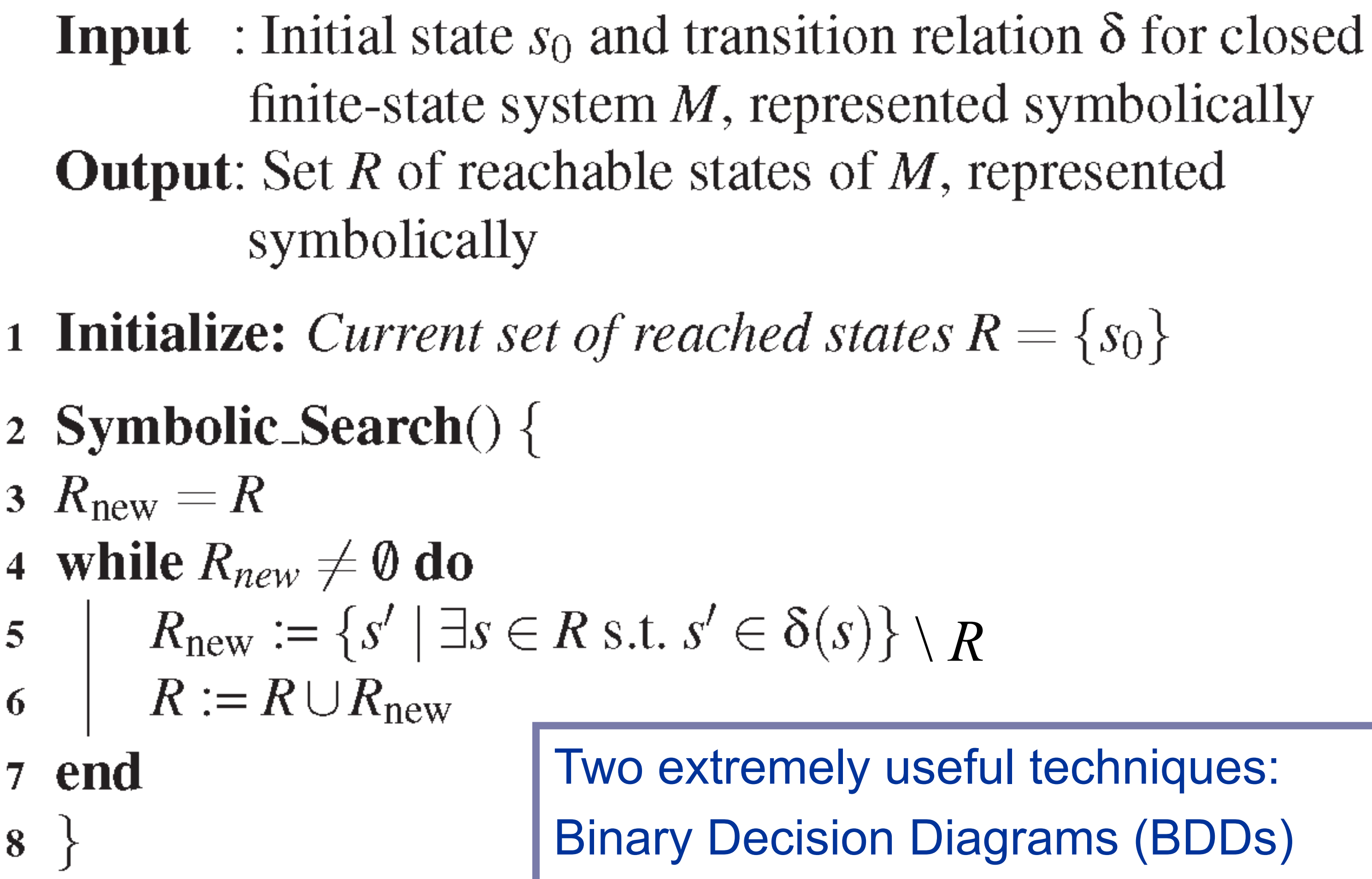
Similarly, δ can be viewed as a finite set of transitions (edges in the FSM), and so can also be represented using a characteristic Boolean function

Symbolic Approach (Breadth First Search)

- Generate the state graph by repeated application of transition function (δ)
- If the goal state reached, stop & report success. Else, continue until all states are seen.



The Symbolic Reachability Algorithm



Input : Initial state s_0 and transition relation δ for closed finite-state system M, represented symbolically

Output: Set R of reachable states of M, represented symbolically

```
Initialize: Current set of reached states $R = \{s_0\}$
Symbolic_Search() {
$R_{\text{new}} = R$
while $R_{new} \neq \emptyset$ do
    $R_{\text{new}} := \{s' \mid \exists s \in R \text{ s.t. } s' \in \delta(s)\} \setminus R$
    $R := R \cup R_{\text{new}}$
end
}
```

Two extremely useful techniques:

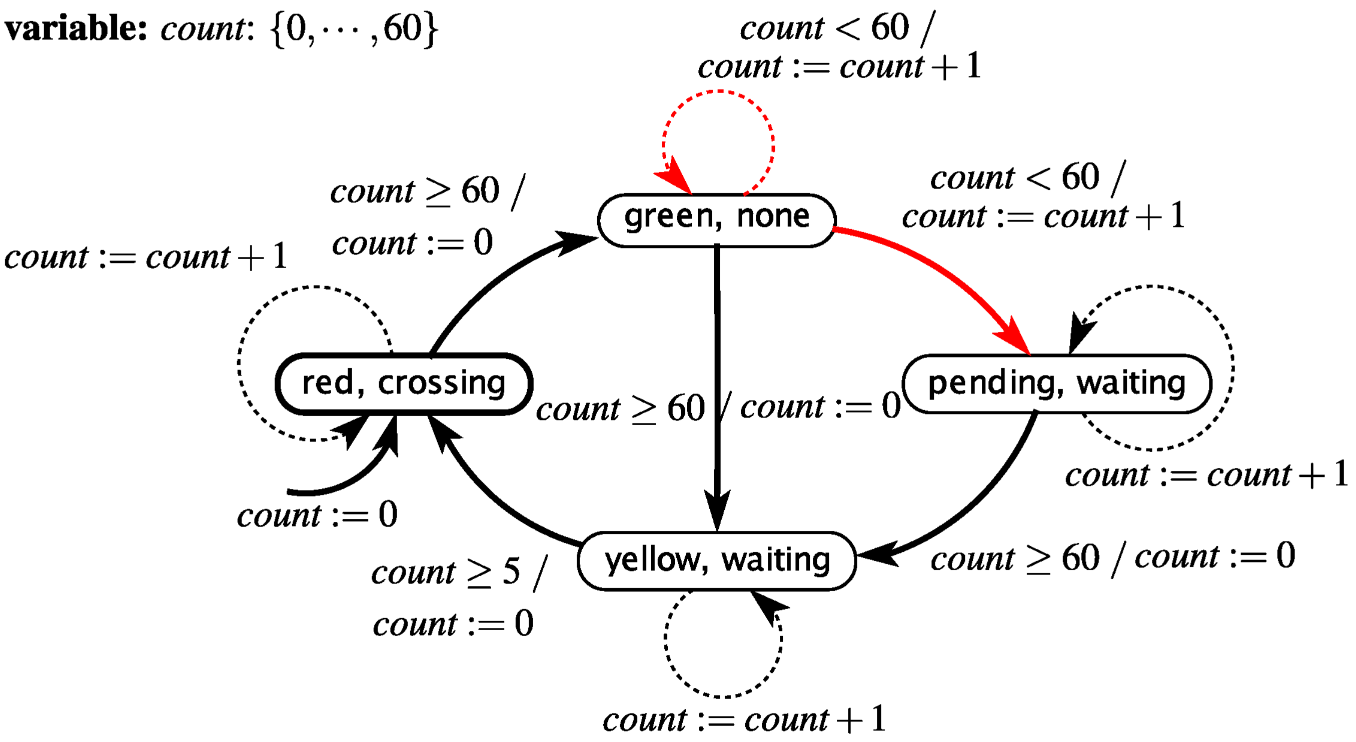
Binary Decision Diagrams (BDDs)

Boolean Satisfiability (SAT)

These are covered in EECS 144

Symbolic Model Checking Example

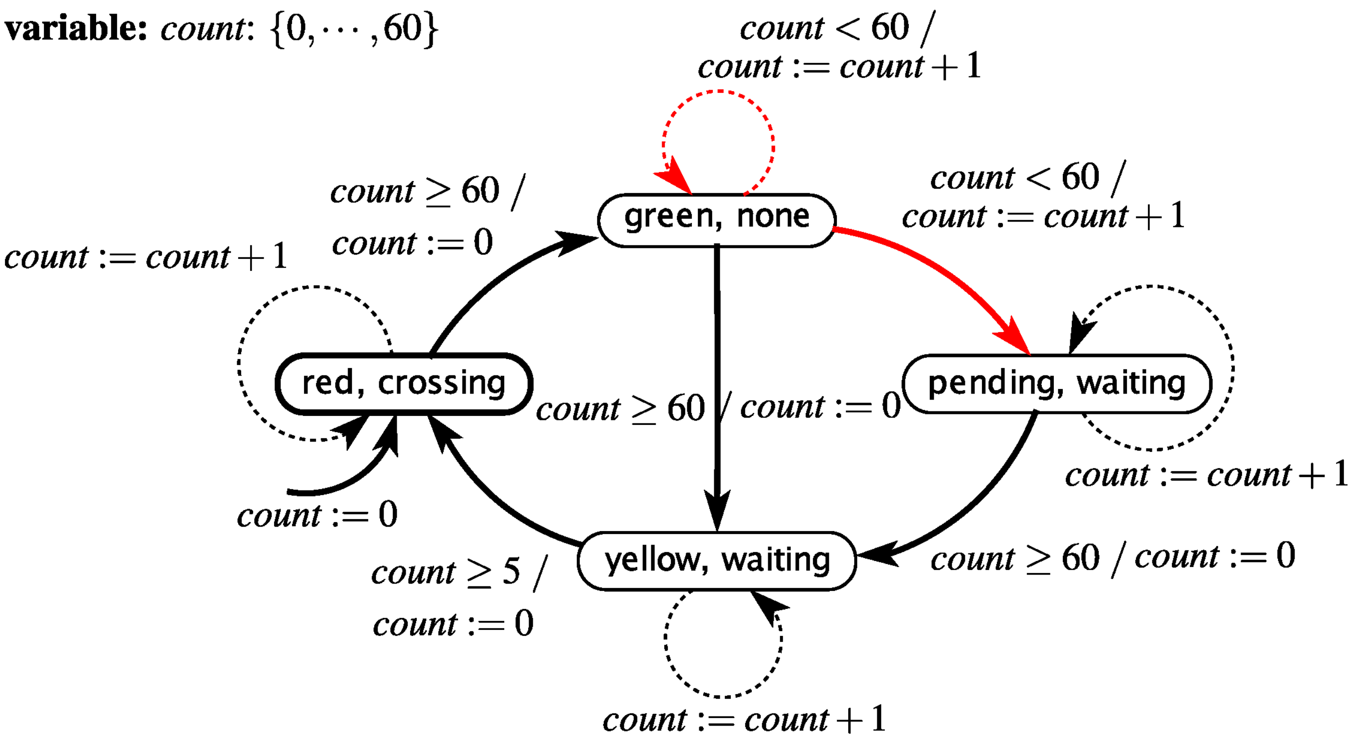
Property: **G** (: (green Æ crossing))



R = (v$_l$ = red Æ v$_p$ = crossing Æ count = 0)

Symbolic Model Checking Example

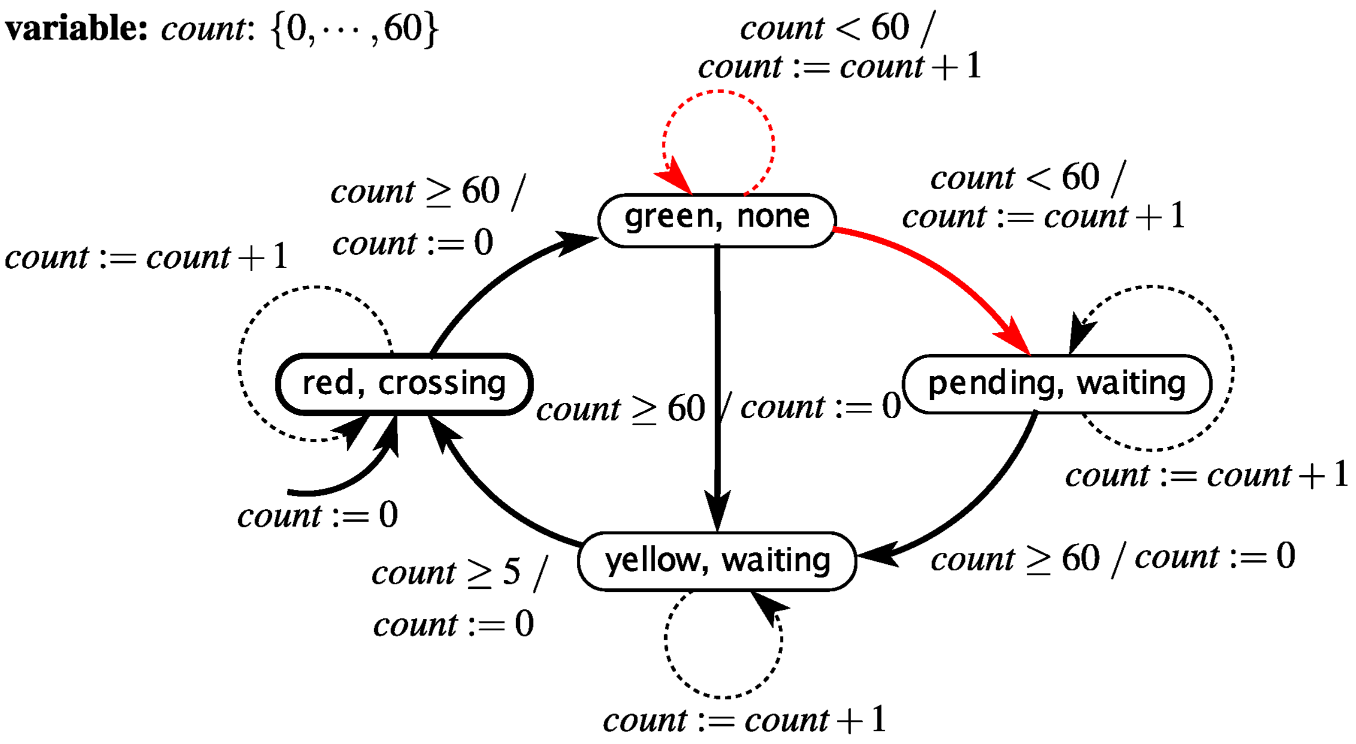
Property: **G** (: (green Æ crossing))



R = (v_l = red Æ v_p = crossing Æ 0 · count · 1)

Symbolic Model Checking Example

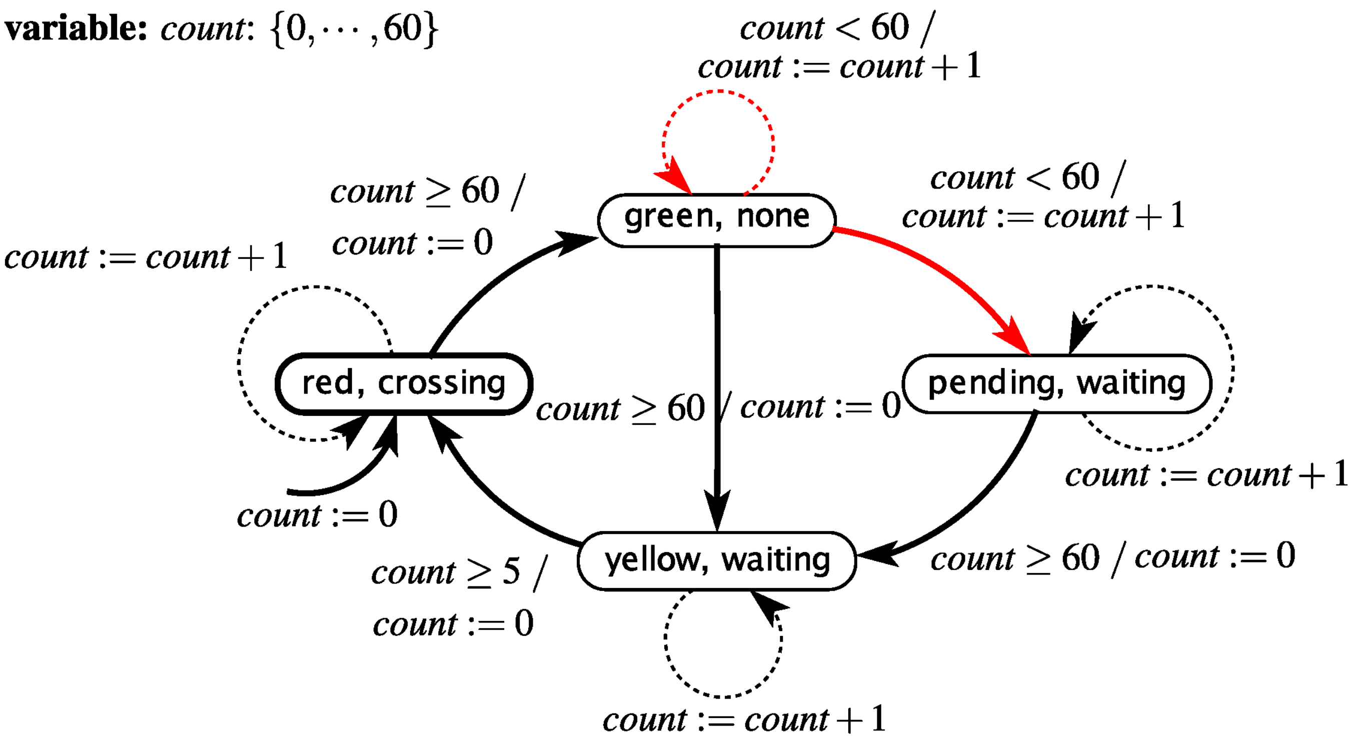
Property: **G** (: (green Æ crossing))



R = (v$_l$ = red Æ v$_p$ = crossing Æ 0 · count · 60)

Symbolic Model Checking Example

Property: **G** (: (green Æ crossing))

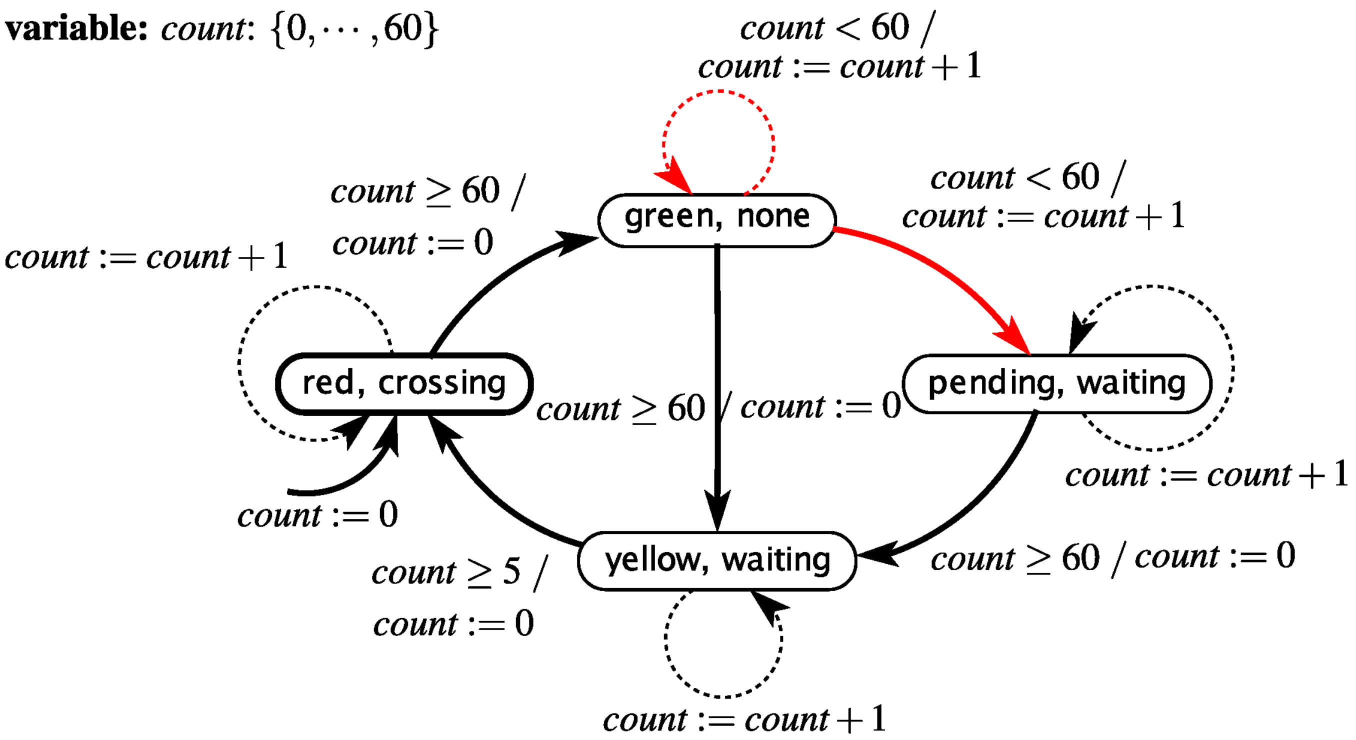


R = (v_l = red Æ v_p = crossing Æ 0 · count · 60)

Ç (v_l = green Æ v_p = none Æ count = 0)

Symbolic Model Checking Example

Property: **G** (: (green Æ crossing))



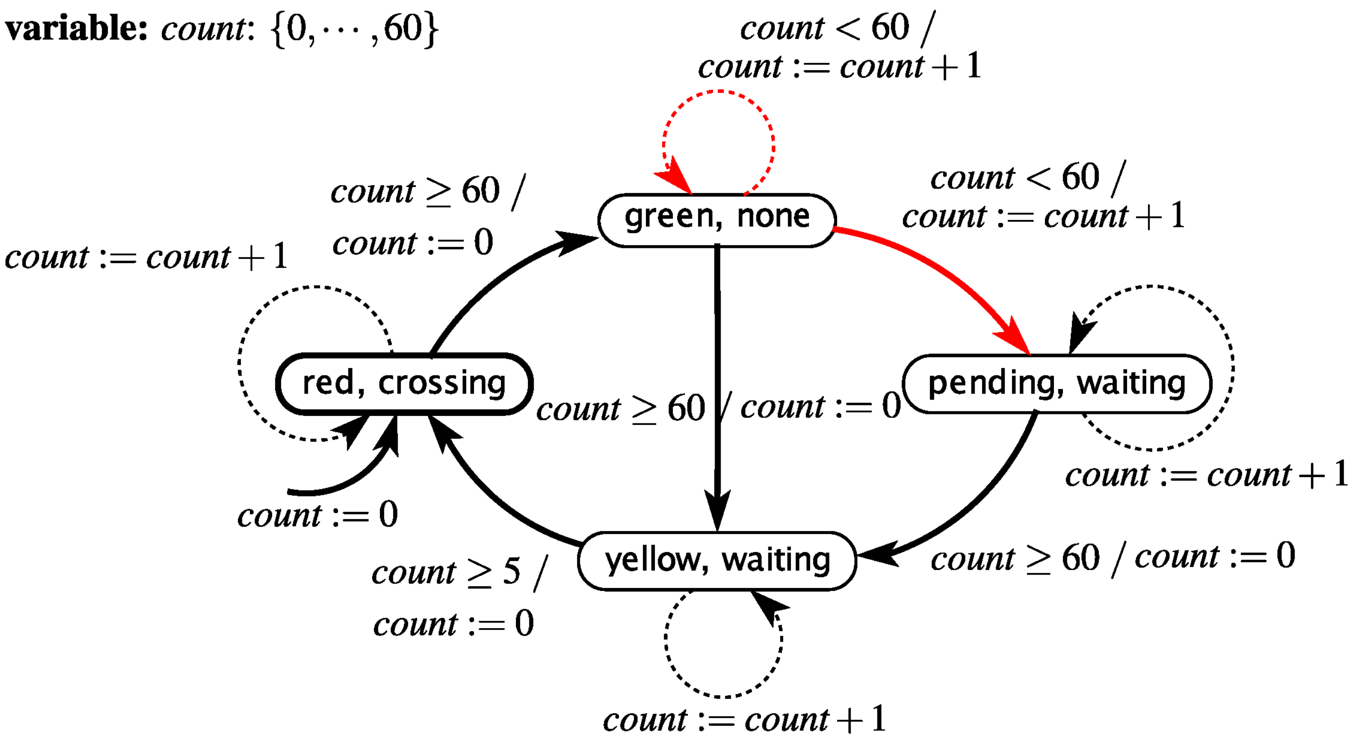
R = (v$_l$ = red Æ v$_p$ = crossing Æ 0 · count · 60)

Ç (v$_l$ = green Æ v$_p$ = none Æ 0 · count · 1)

Ç (v$_l$ = pending Æ v$_p$ = waiting Æ count = 1)

Symbolic Model Checking Example

Property: **G** (: (green Æ crossing))



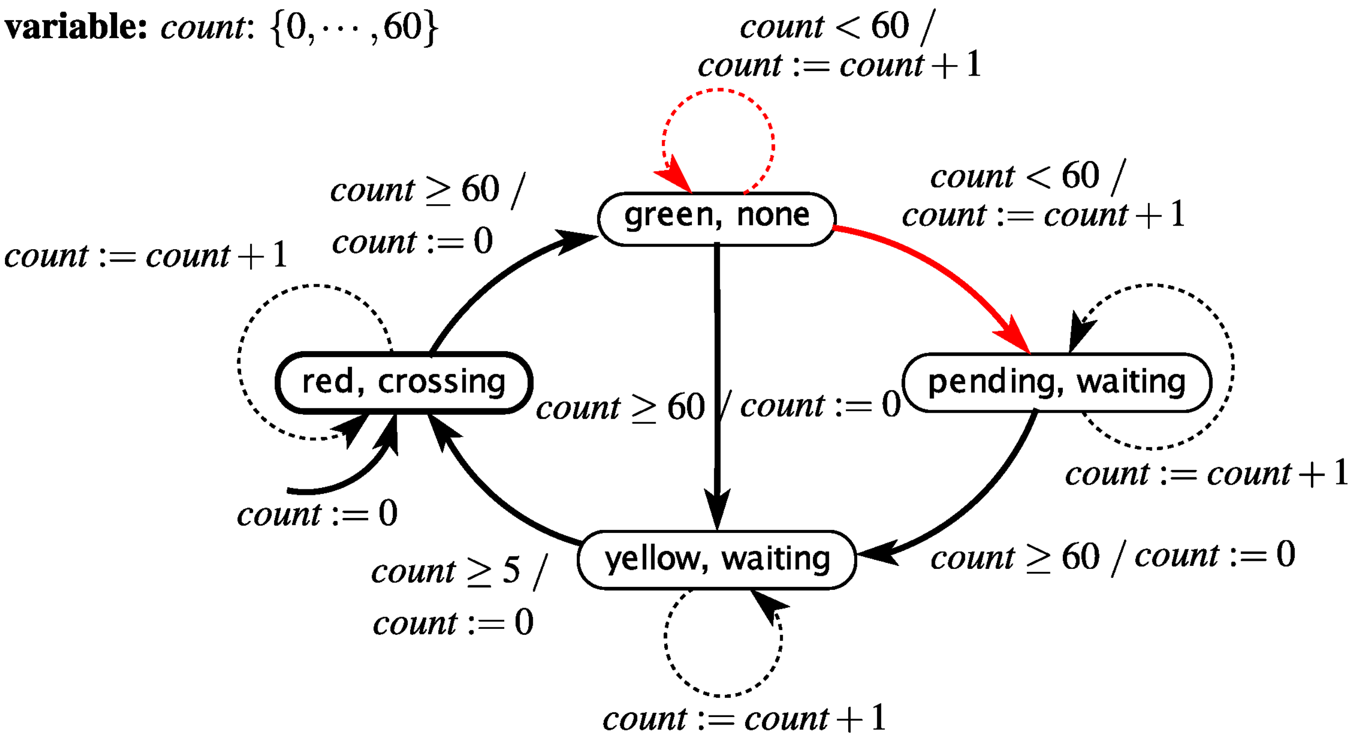
R = (v_l = red Æ v_p = crossing Æ 0 · count · 60)

Ç (v_l = green Æ v_p = none Æ 0 · count · 60)

Ç (v_l = pending Æ v_p = waiting Æ 1 · count · 60)

Symbolic Model Checking Example

Property: **G** (: (green Æ crossing))



R = (v_l = red Æ v_p = crossing Æ 0 · count · 60)

Ç (v_l = green Æ v_p = none Æ 0 · count · 60)

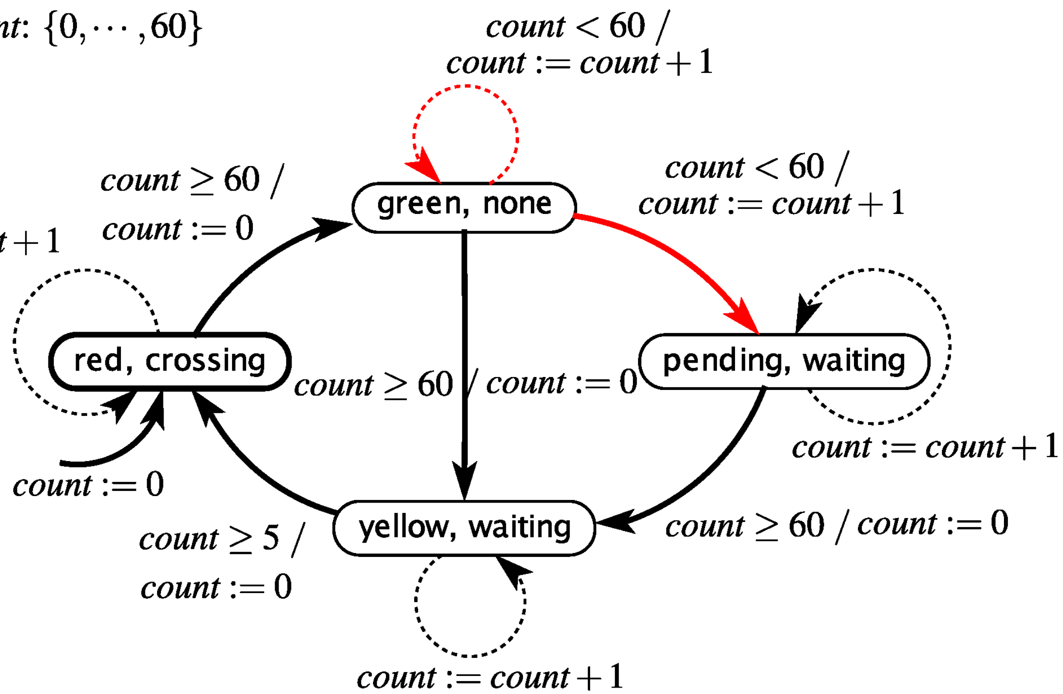
Ç (v_l = pending Æ v_p = waiting Æ 1 · count · 60)

Ç (v_l = yellow Æ v_p = waiting Æ count = 0)

Symbolic Model Checking Example

Property: **G** (: (green Æ crossing))

variable: *count*: $\{0,\cdots,60\}$



R = (v_l = red Æ v_p = crossing Æ 0 · count · 60)

Ç (v_l = green Æ v_p = none Æ 0 · count · 60)

Ç (v_l = pending Æ v_p = waiting Æ 1 · count · 60)

Ç (v_l = yellow Æ v_p = waiting Æ 0 · count · 5)

Abstraction in Model Checking

Should use simplest model of a system that provides proof of safety.

Simpler models have smaller state spaces and easier to check.

The challenge is to know what details can be abstracted away.

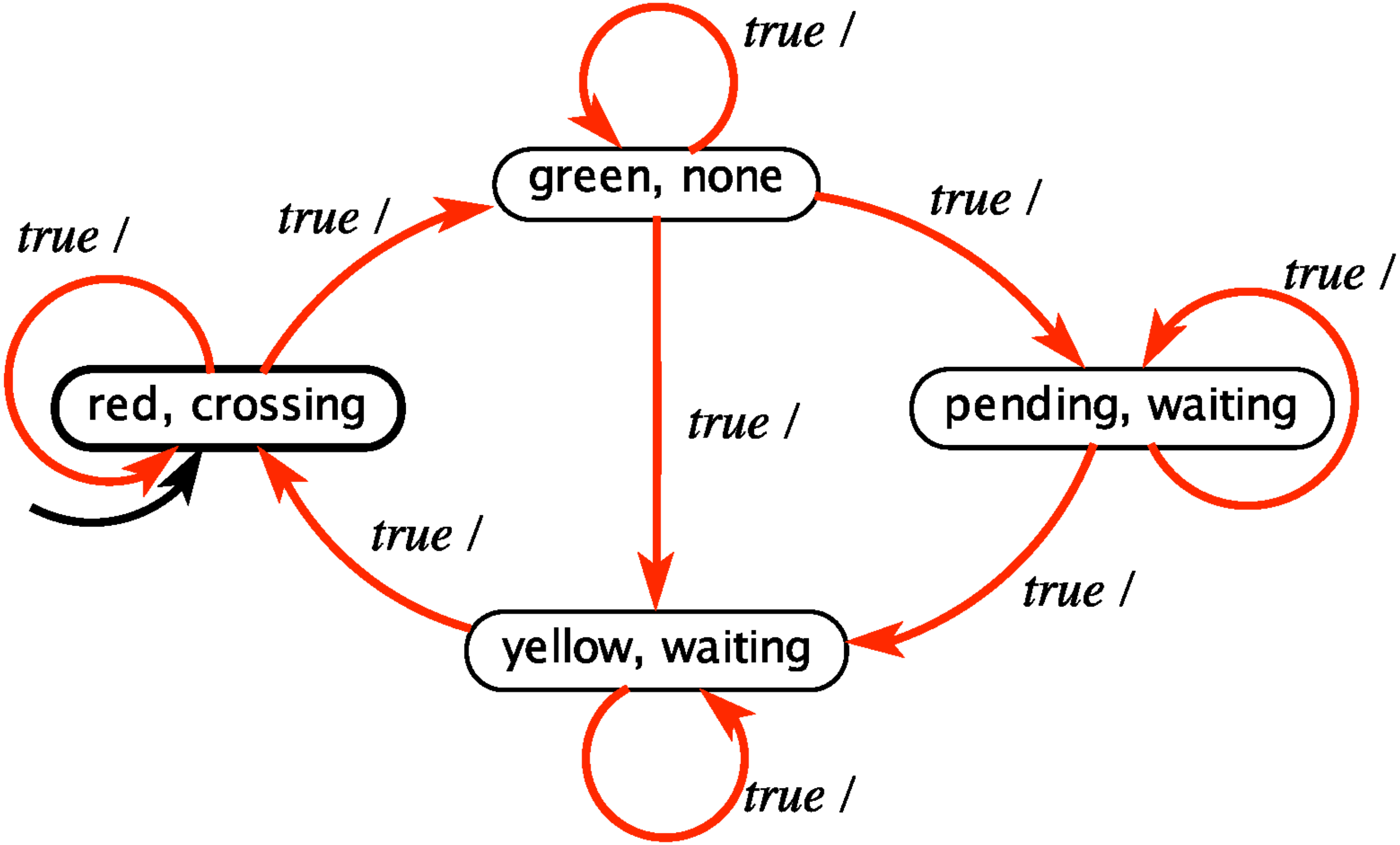
A simple and useful approach is called localization abstraction.

A localization abstraction hides state variables that are irrelevant to the property being verified.

Abstract Model for Traffic Light Example

Property: **G** (: (green Æ crossing))

What' s the hidden variable?



Model Checking Liveness Properties

A **safety** property (informally) states that “nothing bad ever happens” and has finite-length counterexamples.

A **liveness** property, on the other hand, states “something good eventually happens”, and only has infinite-length counterexamples.

Model checking liveness properties is more involved than simply doing a reachability analysis. See Section 14.4 for more information.

Suppose we have a Robot that must pick up multiple things, in any order

$$\phi_i = \text{robot picks up item } i\text{, where } 1 \leq i \leq n$$

How would you state this goal in temporal logic?

Suppose we have a Robot that must pick up multiple things, in any order

$$\phi_i = \text{robot picks up item } i\text{, where } 1 \leq i \leq n$$

Goal to be achieved is:

$$\mathbf{F}\phi_1 \wedge \mathbf{F}\phi_2 \wedge \cdots \wedge \mathbf{F}\phi_n$$

How can we find a strategy to achieve this goal?

Suppose we have a Robot that must pick up multiple things, in any order

$$\phi_i = \text{robot picks up item } i\text{, where } 1 \leq i \leq n$$

Goal to be achieved is:

$$\mathbf{F}\phi_1 \wedge \mathbf{F}\phi_2 \wedge \cdots \wedge \mathbf{F}\phi_n$$

How can we find a strategy to achieve this goal?

→How about this: Do repeated reachability, first from q_0 to reach ϕ_1, then from ϕ_1 to reach ϕ_2, then ϕ_2 to reach ϕ_3,

→Problem: What if ϕ_2 is not reachable from ϕ_1, but reachable from q_0?

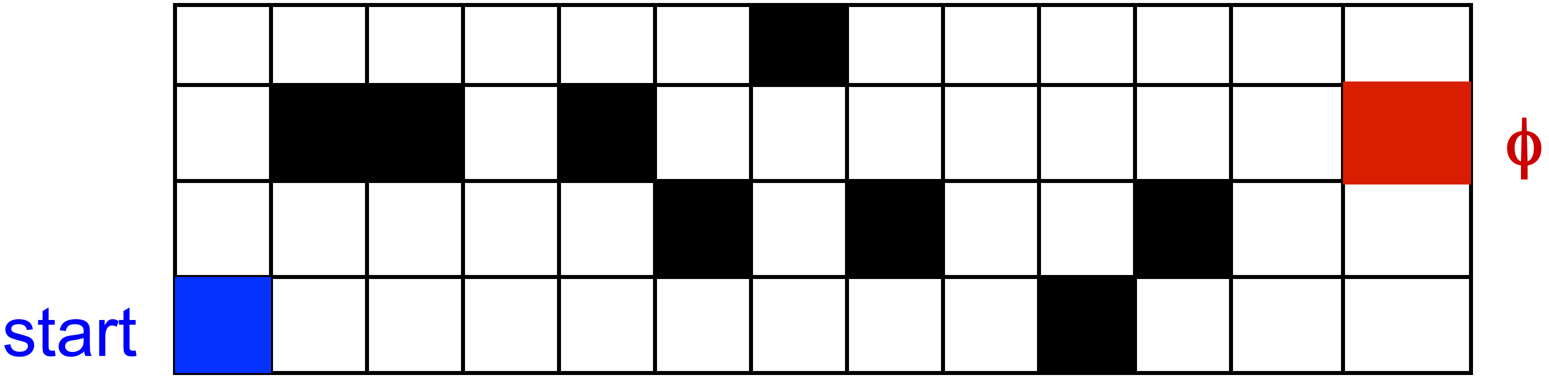
Student question: Suppose we have a Robot that must pick up multiple things, ***in a specified order***

$$\phi_i = \text{robot picks up item } i, \text{ where } 1 \leq i \leq n$$

Goal to be achieved is:

$$\mathbf{F}(\phi_1 \land \mathbf{F}(\phi_2 \land \cdots \land \mathbf{F}\phi_n))$$

A Robot delivery service, with moving obstacles



ϕ = destination for robot

At any time step:

Robot can move Left, Right, Up, Down, Stay Put

Environment can move one obstacle Up or Down or Stay Put

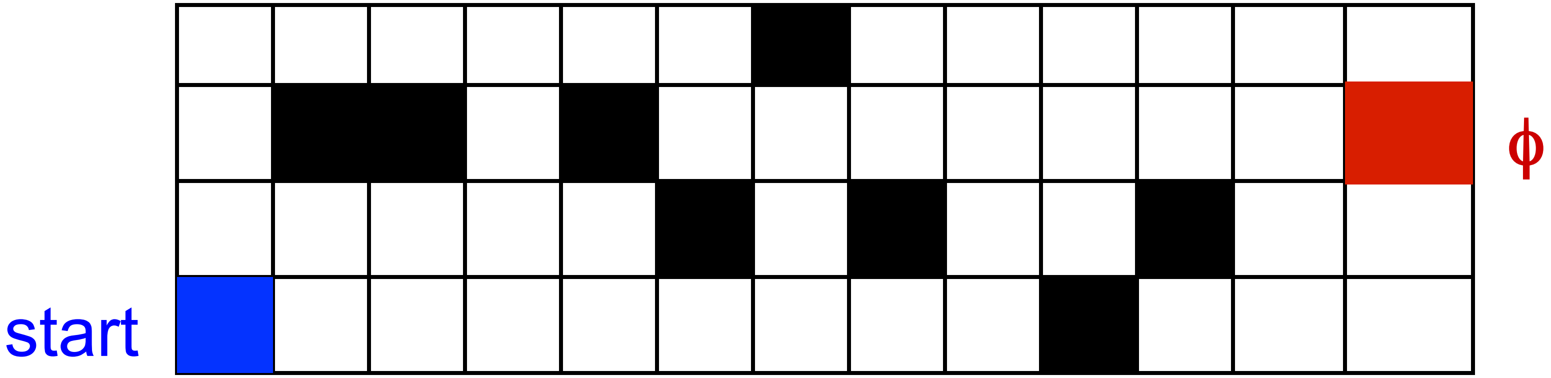
→ But only 3 times total

Can model Robot and Env as FSMs

→ Robot state = its position,

→ Env state = positions of obstacles and counts

A Robot delivery service, with moving obstacles



ϕ = robot delivers item to destination

Goal to be achieved can be stated in temporal logic

$\mathbf{F}\ \phi$

How can we find a path for the robot from starting point to the destination?

→ This is an example of a “reachability problem”