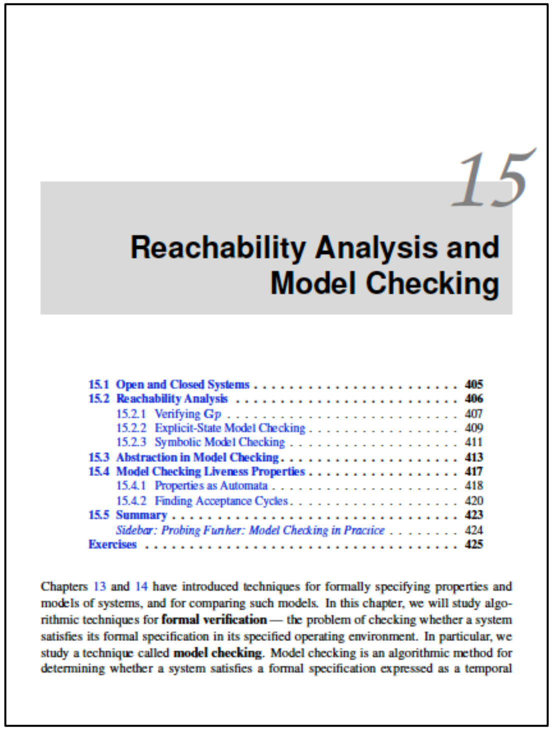
Lectures 20/21: Reachability Analysis and Model Checking

Slides were originally developed by Profs. Edward Lee and Sanjit Seshia, and subsequently updated by Profs. Gavin Buskes and Iman Shames.

Outline

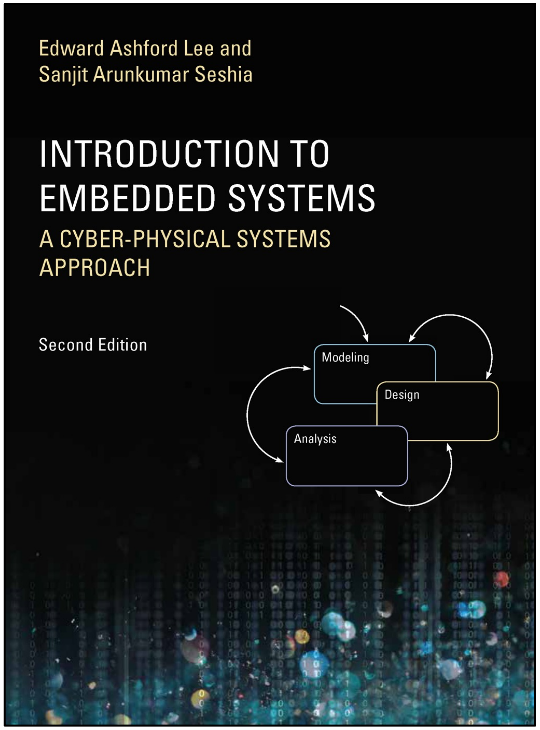
- Reachability
- Model checking



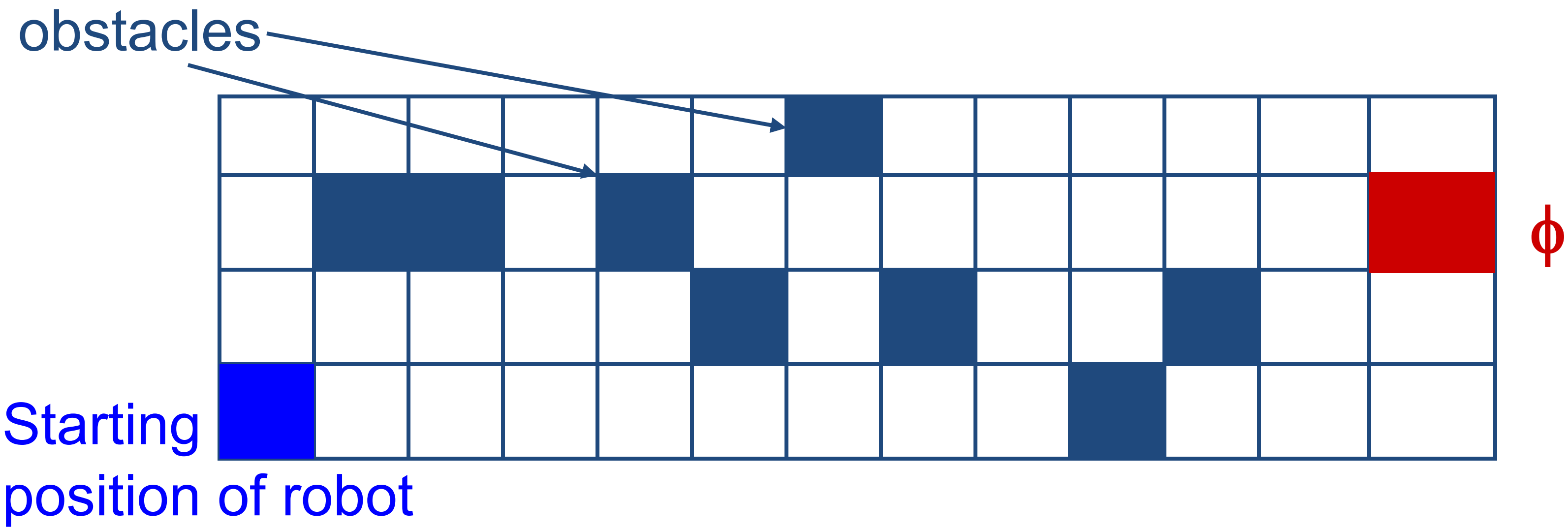
15

Reachability Analysis and Model Checking

Chapters 13 and 14 have introduced techniques for formally specifying properties and models of systems, and for comparing such models. In this chapter, we will study algorithmic techniques for **formal verification** — the problem of checking whether a system satisfies its formal specification in its specified operating environment. In particular, we study a technique called **model checking**. Model checking is an algorithmic method for determining whether a system satisfies a formal specification expressed as a temporal



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

Reachability Analysis and Model Checking

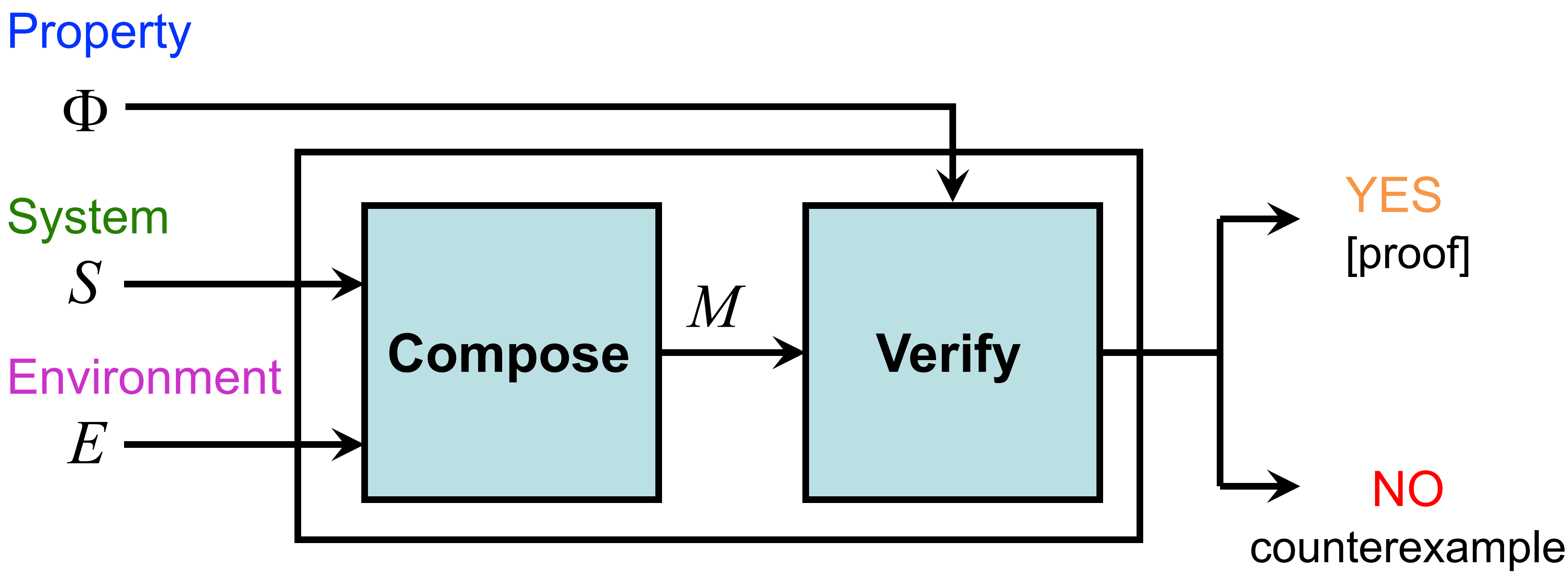
Reachability analysis is the process of computing the set of reachable states for a system.

- preceding problems can be solved using reachability analysis

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

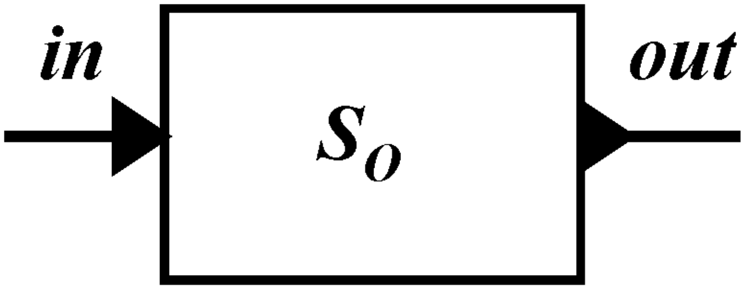
- Model checking typically performs reachability analysis.

Formal Verification

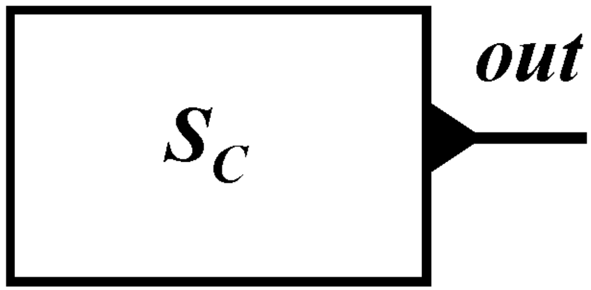


Open vs. Closed Systems

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

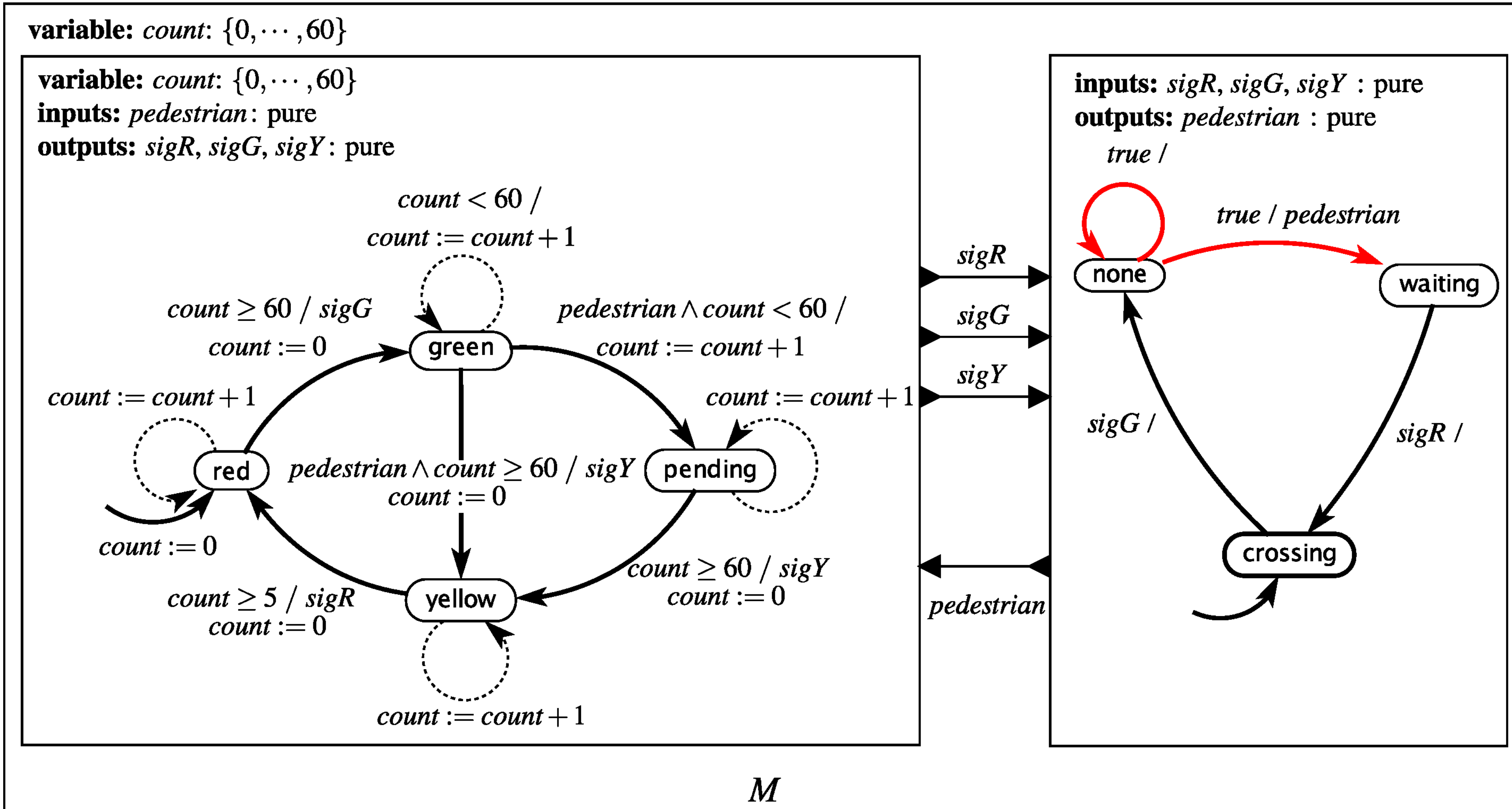
Model Checking G *p*

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.

Traffic Light Controller Example

Property: $G(\neg(green \wedge crossing))$



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.

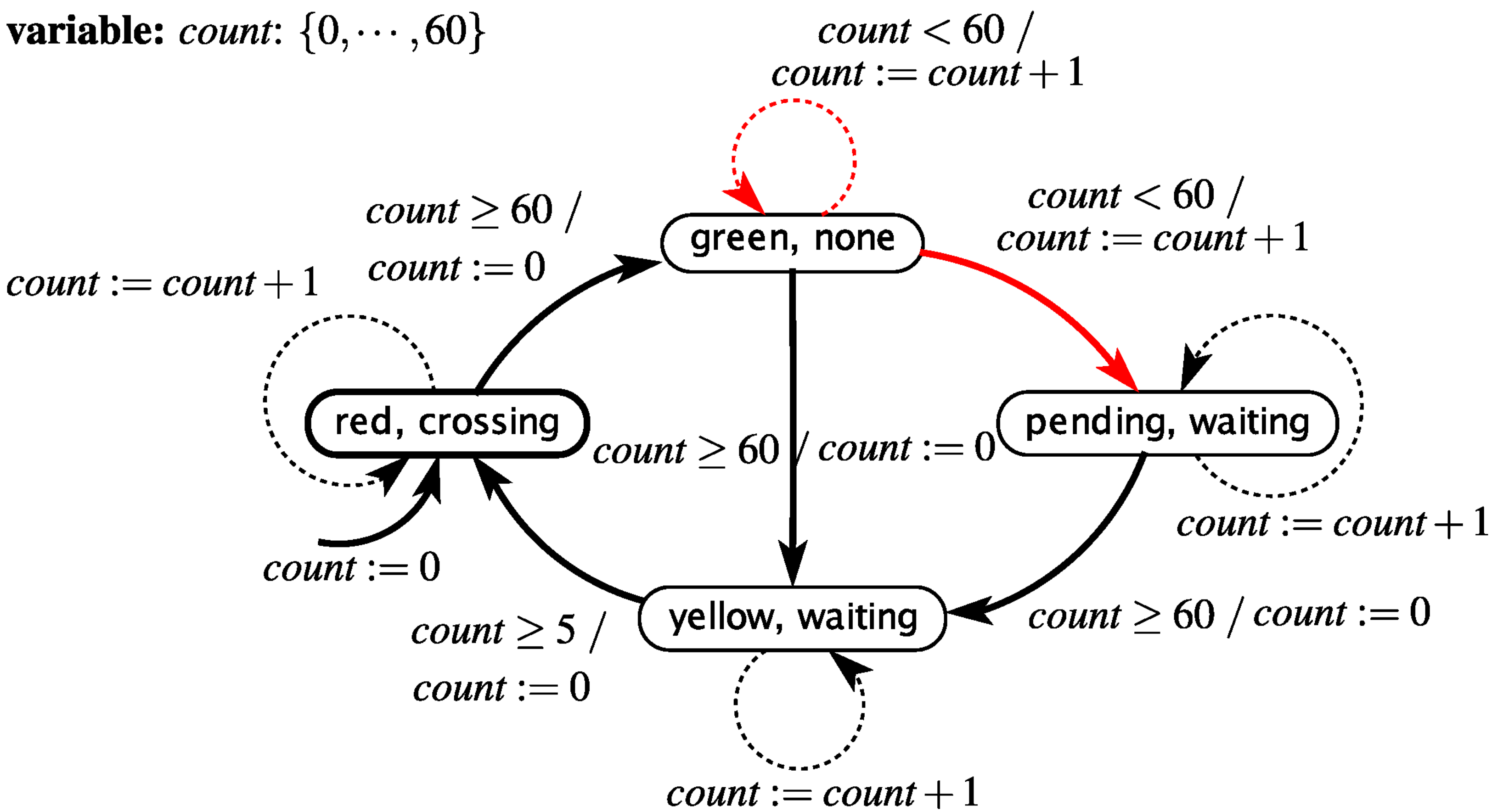
- The state space found is typically represented as a directed graph called a state graph.
- When M is a finite-state machine, this reachability analysis will terminate (in theory).
- In practice, though, the number of states may be prohibitively large consuming too much run-time or memory (the state explosion problem).

Composed FSM for Traffic Light Controller

Property: $G(\neg(green \wedge crossing))$

This FSM has 189 states

(accounting for different values of count)



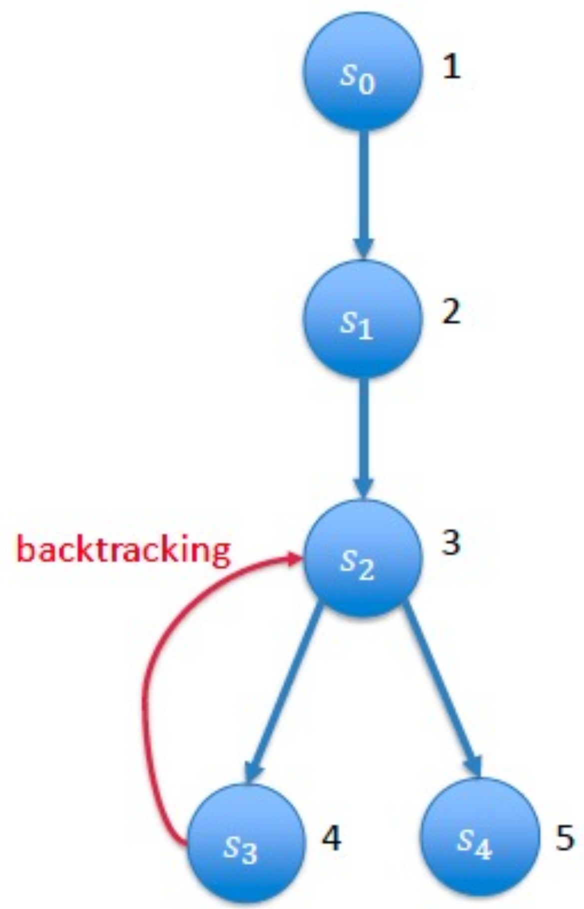
Reachability Analysis Through Graph Traversal

- 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:

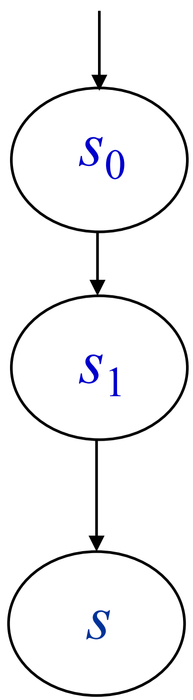
1. Set of visited states R
2. Stack with current path from the initial state



Potential problems for a huge graph?

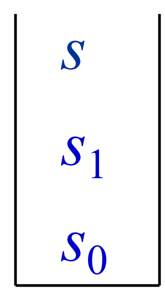
Generating counterexamples

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



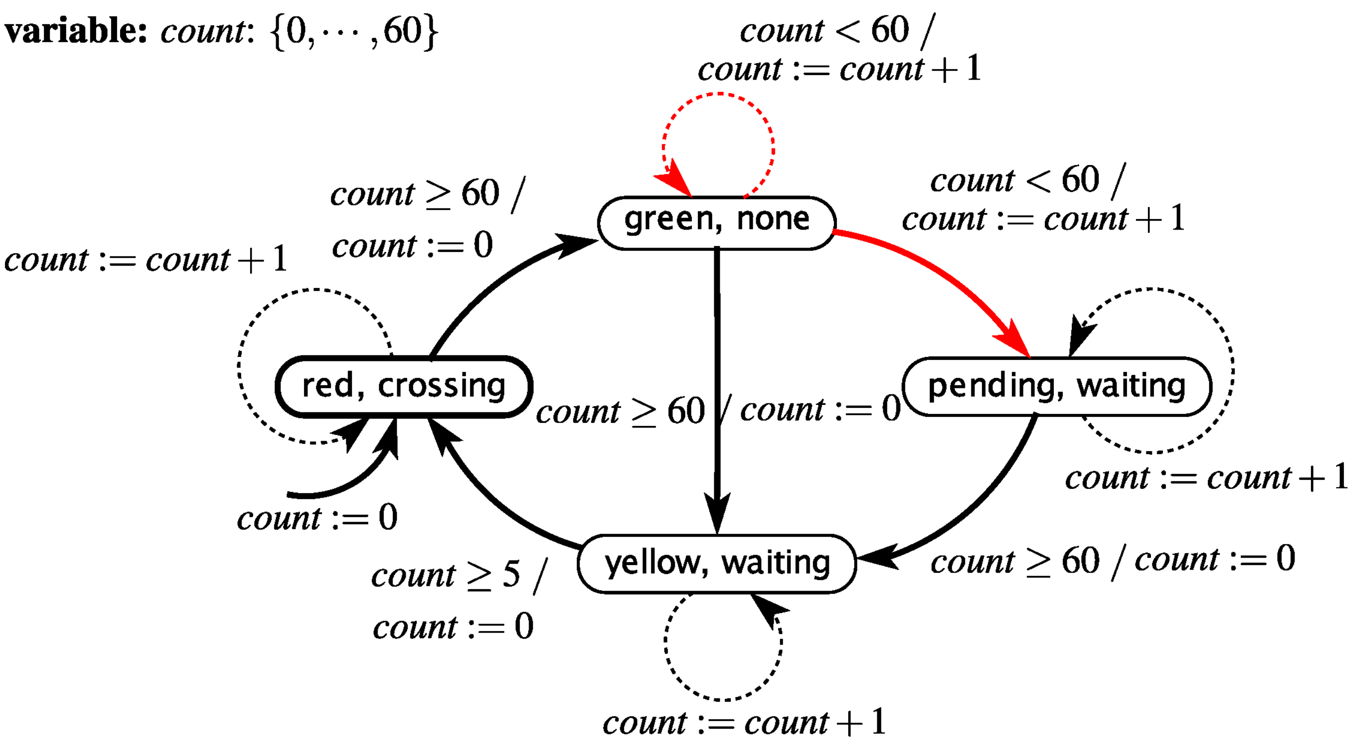
Simply read the trace off the stack

Stack:



Explicit State Model Checking Example

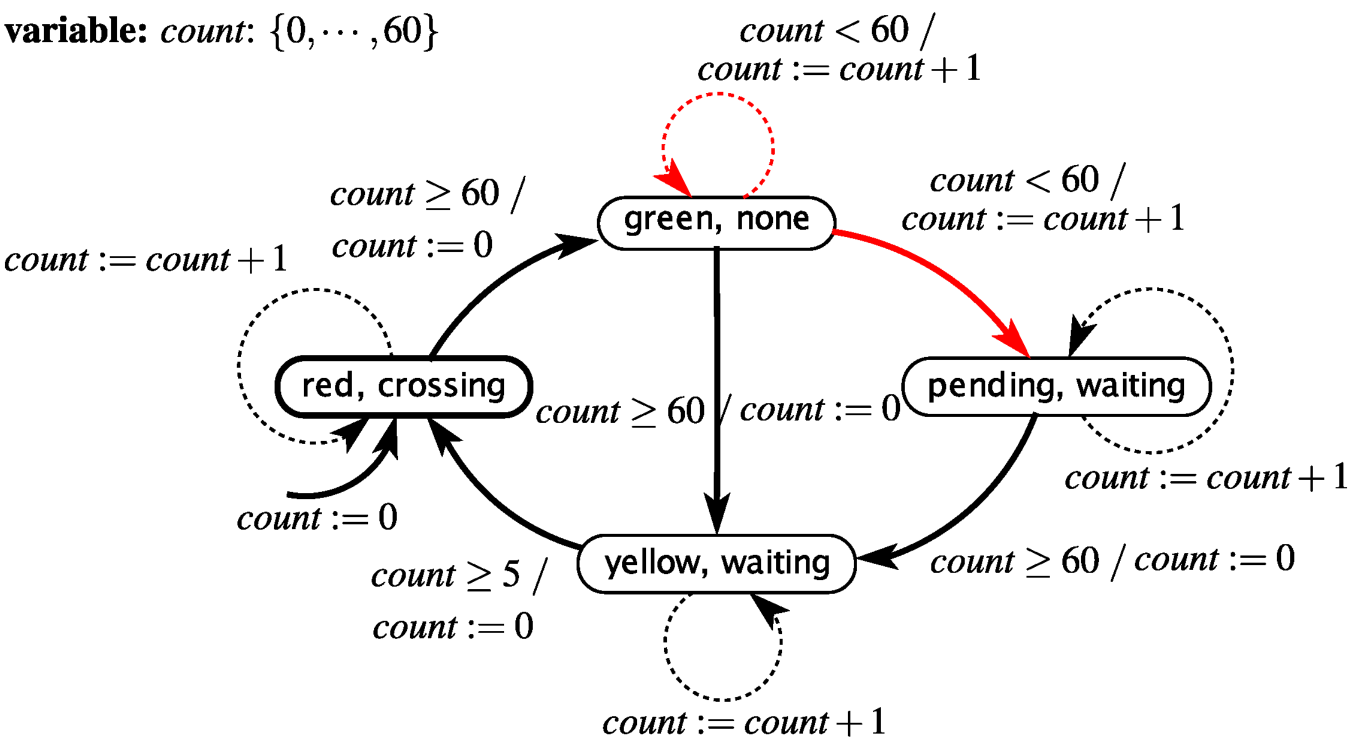
Property: $G(\neg(green \wedge crossing))$



R = { (red, crossing, 0) }

Explicit State Model Checking Example

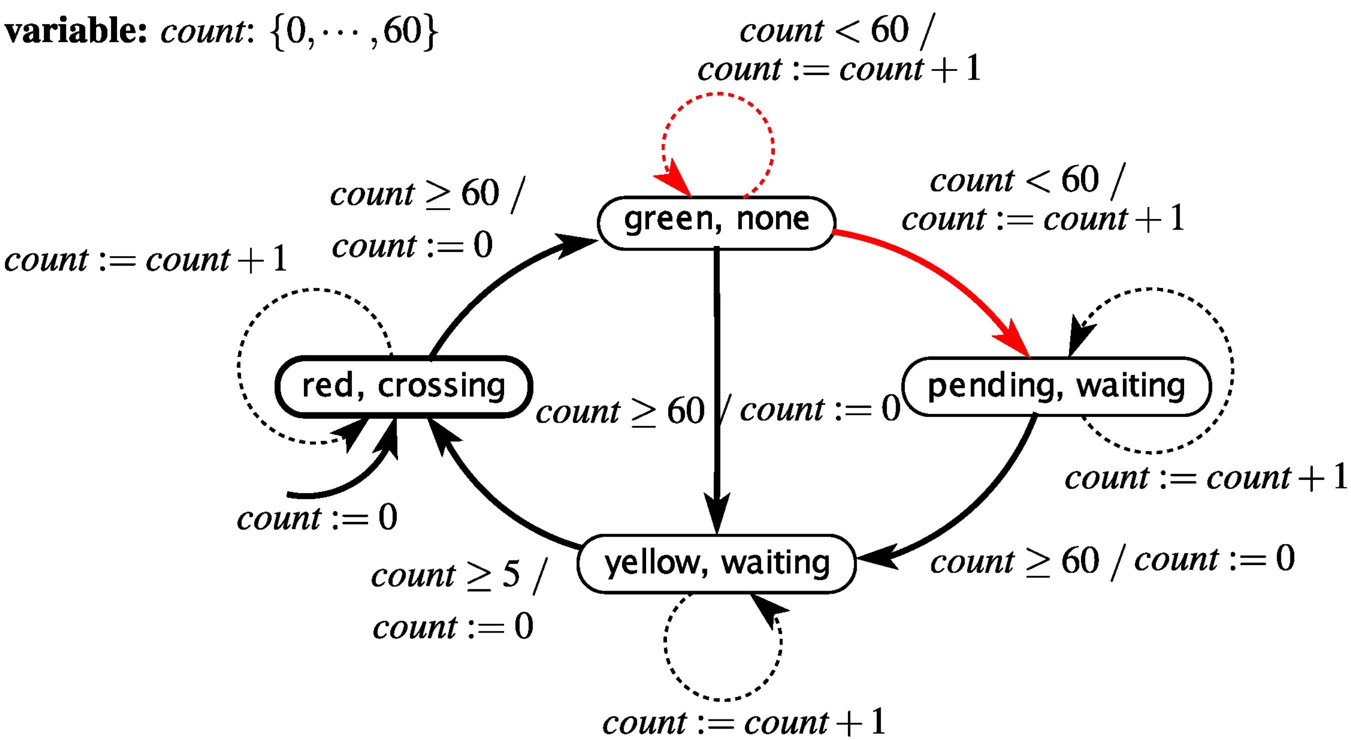
Property: $G(\neg(green \wedge crossing))$



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

Explicit State Model Checking Example

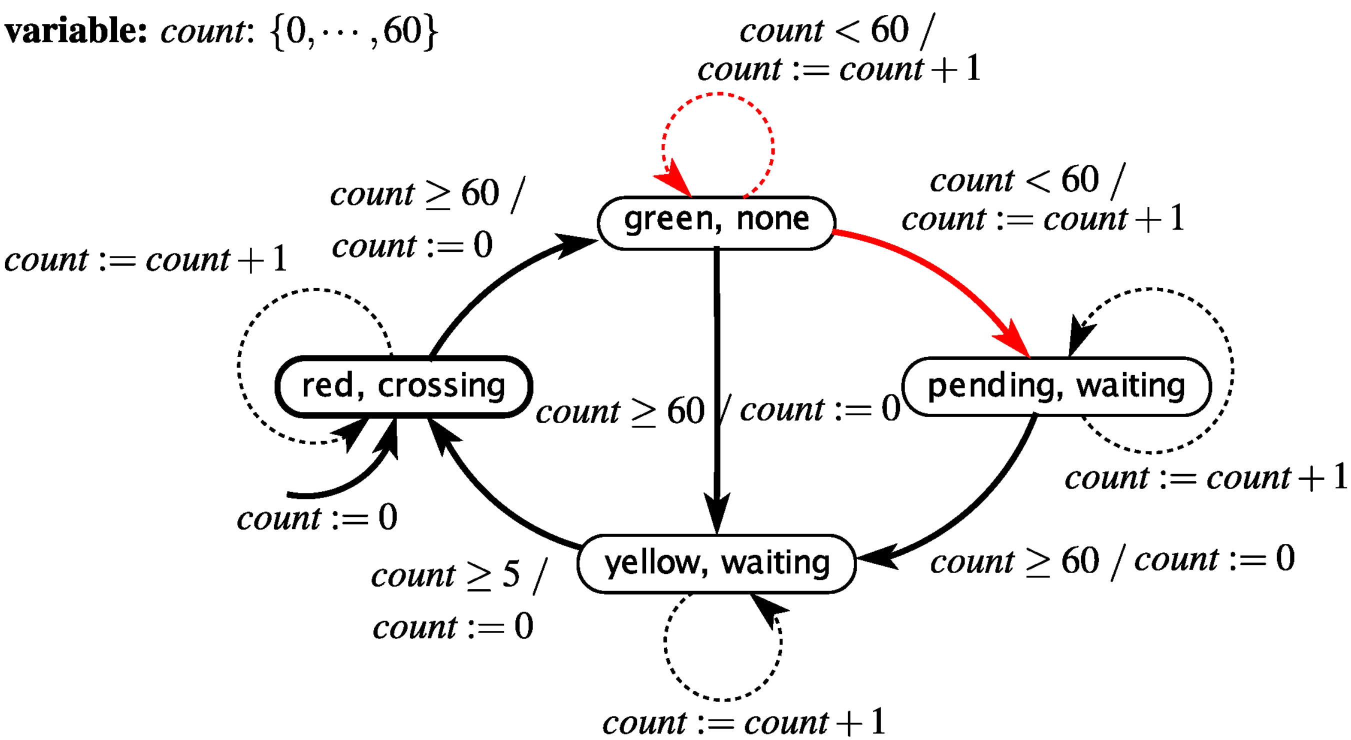
Property: $G(\neg(green \wedge crossing))$



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

Explicit State Model Checking Example

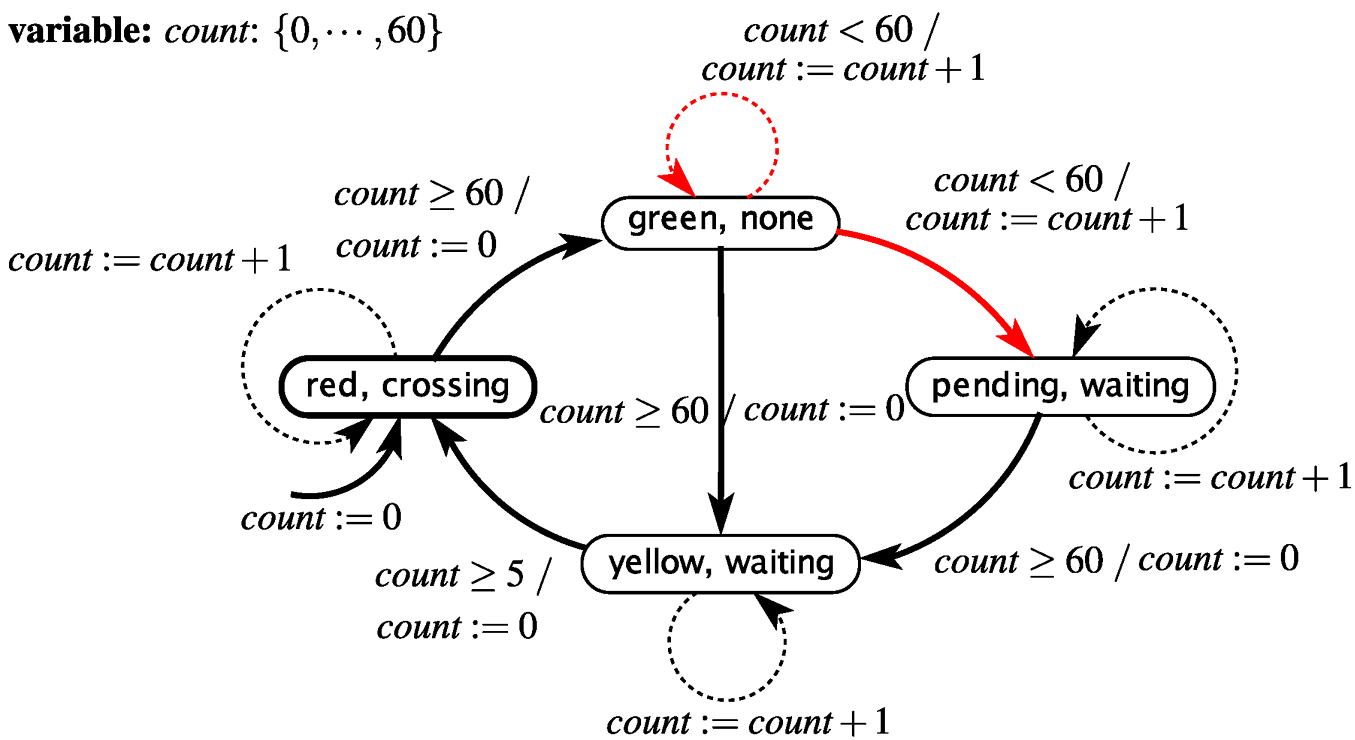
Property: $G(\neg(green \wedge crossing))$



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

Explicit State Model Checking Example

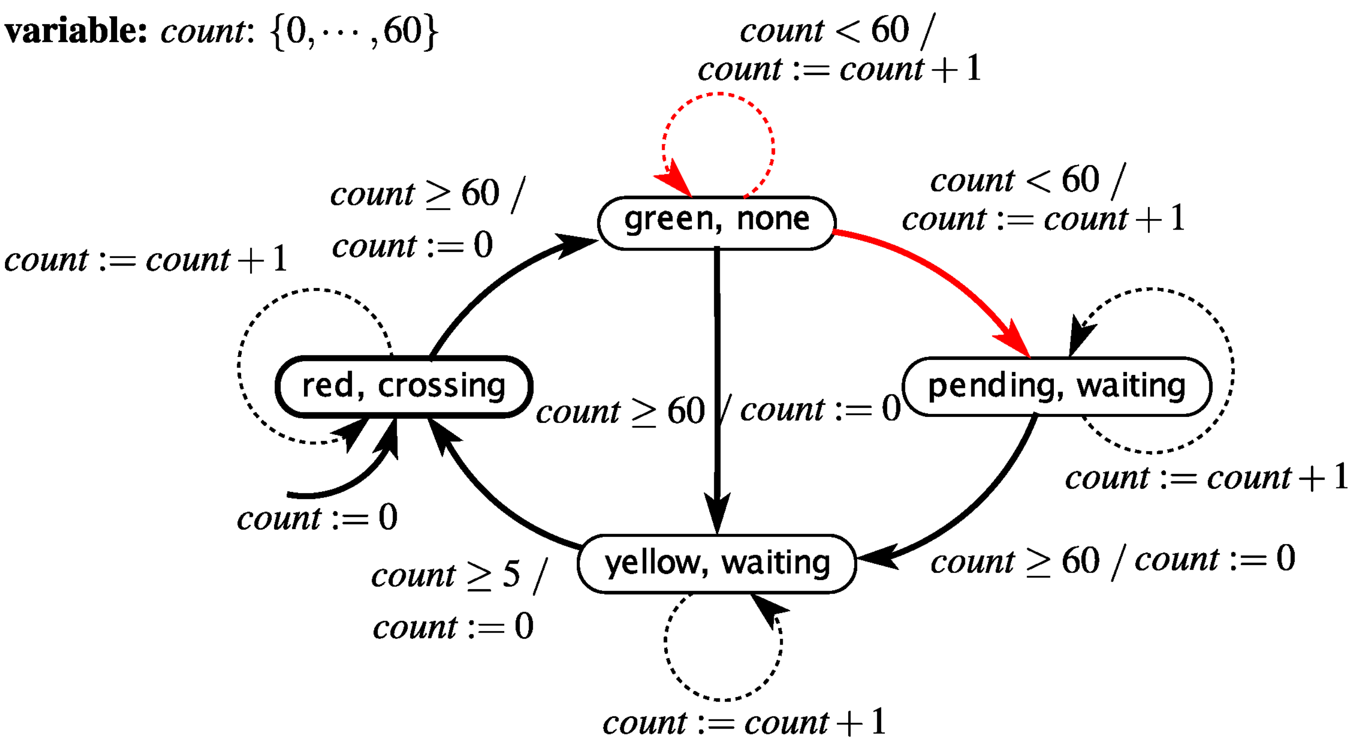
Property: $G(\neg(green \wedge crossing))$



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

Explicit State Model Checking Example

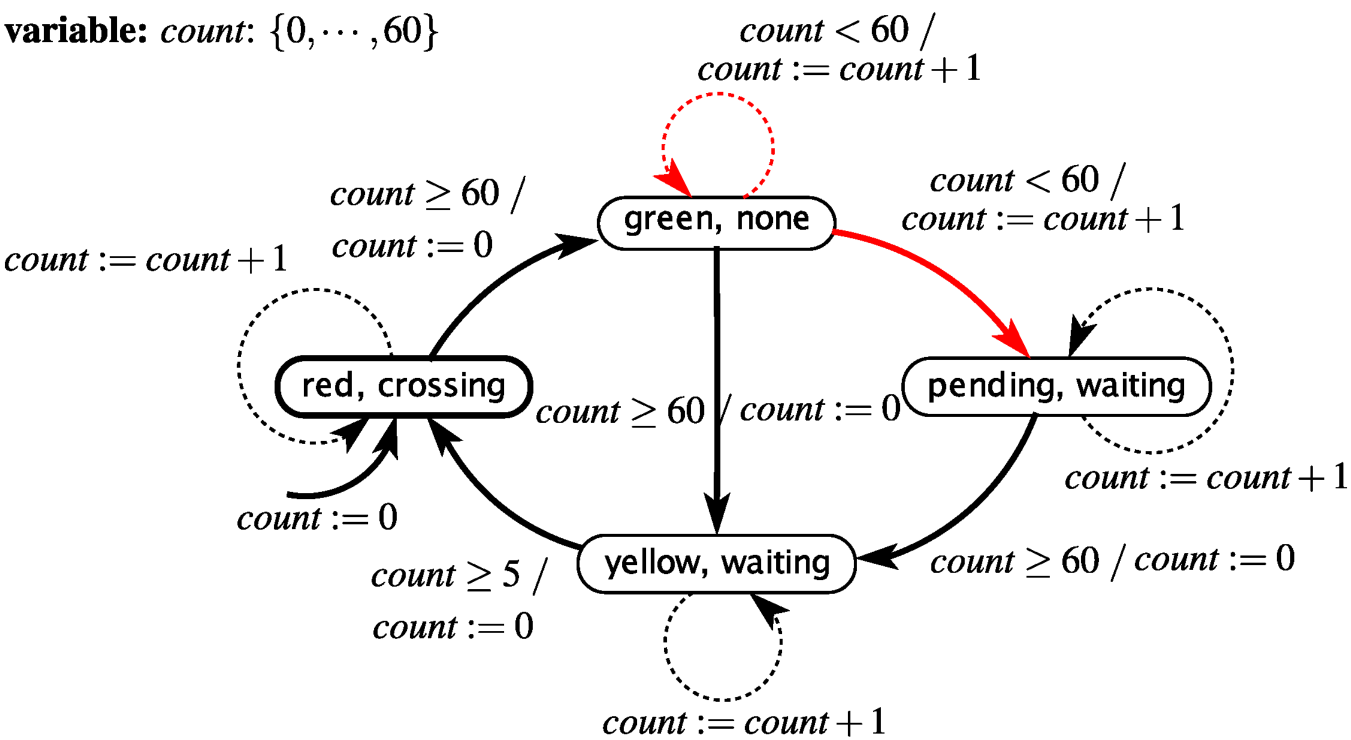
Property: $G(\neg(green \wedge crossing))$



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

Explicit State Model Checking Example

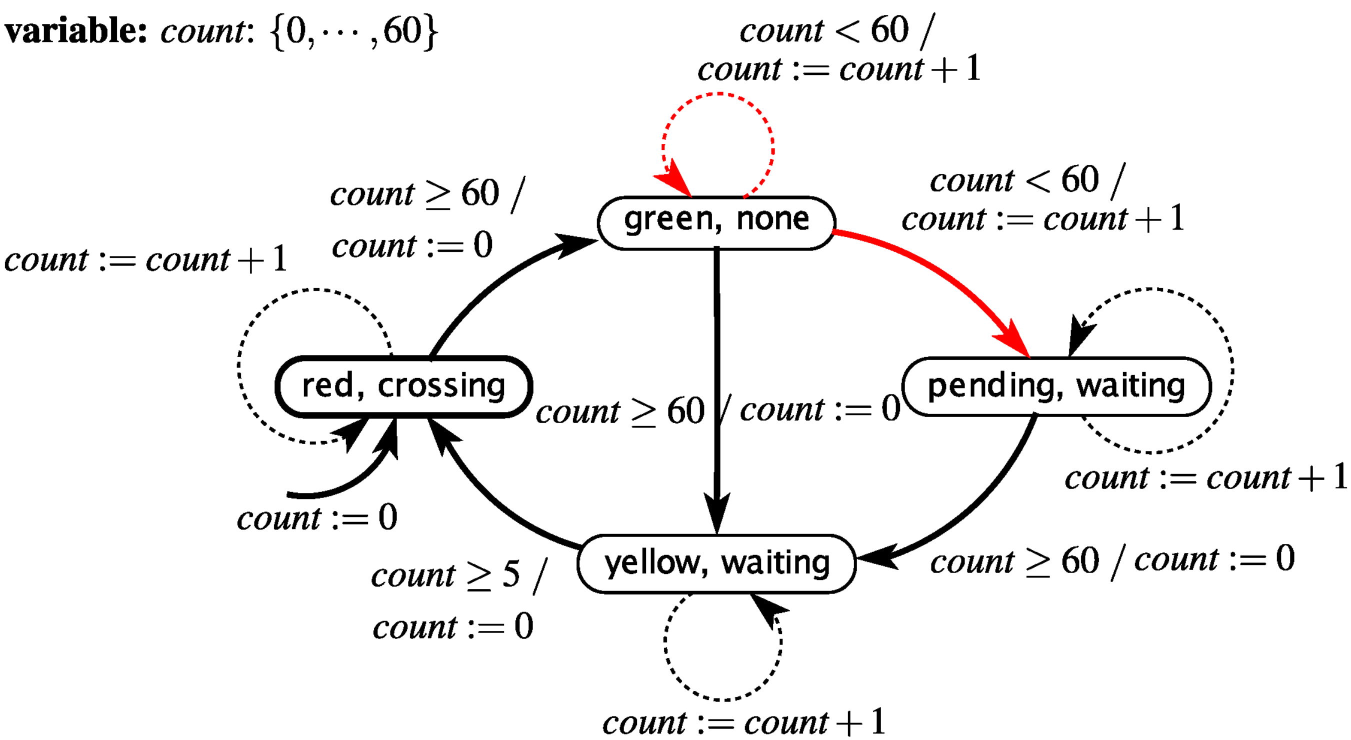
Property: $G(\neg(green \wedge crossing))$



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

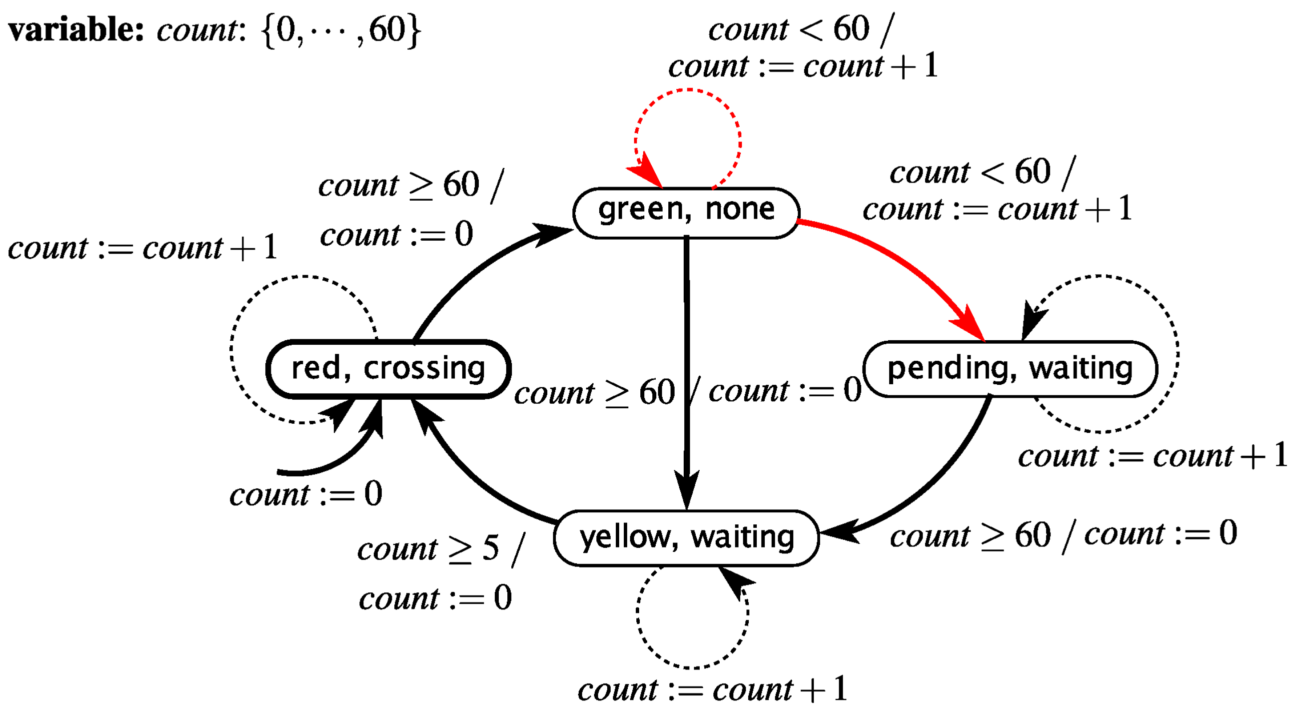
Property: $G(\neg(green \wedge crossing))$



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: $G(\neg(green \land crossing))$



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 state

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

Set operations can be performed using Boolean algebra :

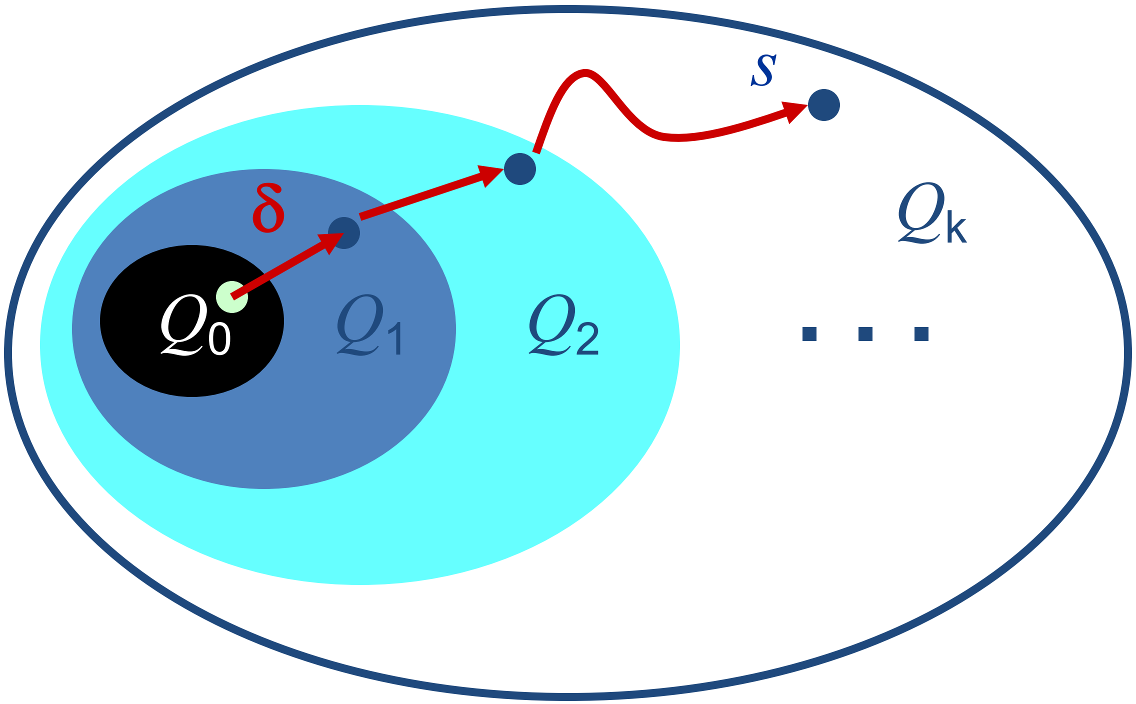
Represent a finite subset of states $C \subseteq S$ by its characteristic Boolean function f_C: S -> $\{0,1\}$ where

$$f_C(x) = 1, \text{ iff } x \in C$$

Similarly, the state transition function δ yields a set $\delta(s)$ of next states from current state s, and so can also be represented using a characteristic Boolean function for each s.

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**

$\quad R_{\text{new}} := \{s' \mid \exists s \in R \text{ s.t. } s' \in \delta(s)\} \setminus R$

$\quad R := R \cup R_{\text{new}}$

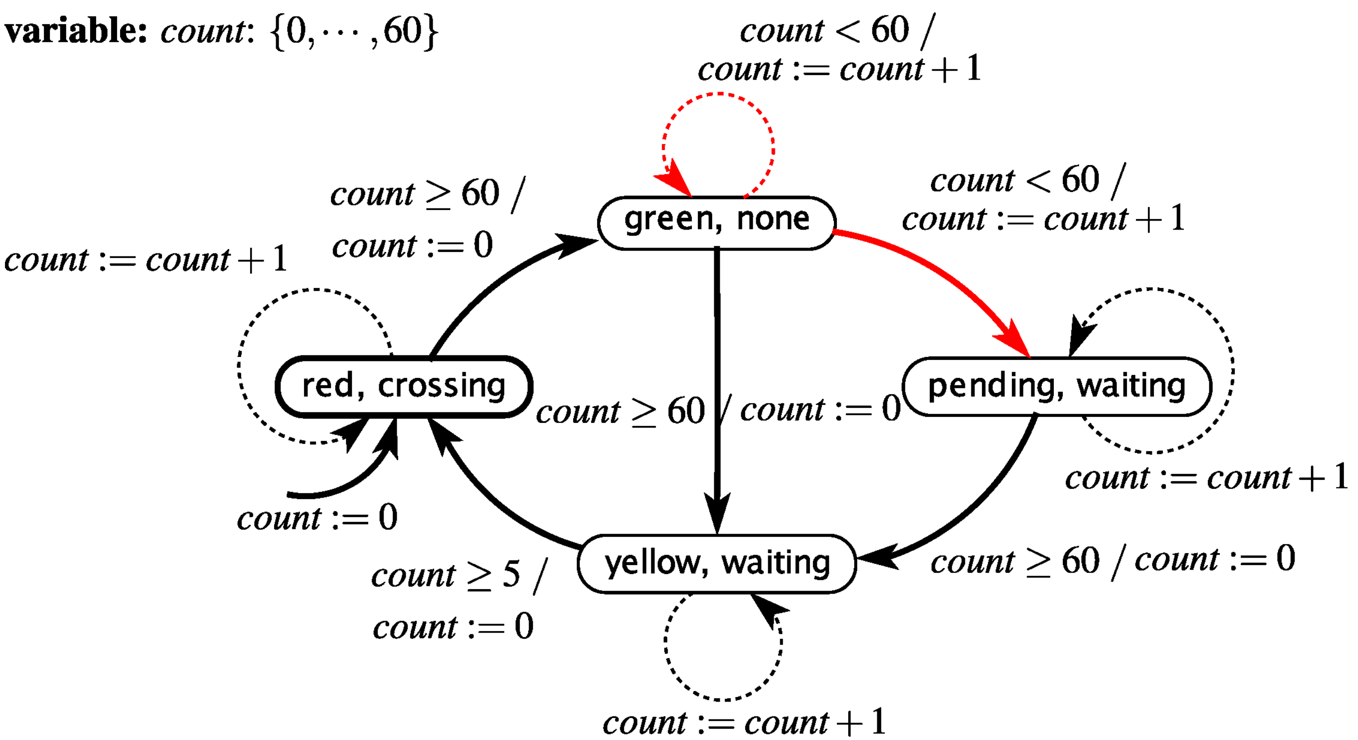
end

}

Two extremely useful techniques:
Binary Decision Diagrams (BDDs)
Boolean Satisfiability (SAT)

Symbolic Model Checking Example

Property: $G(\neg(green \wedge crossing))$

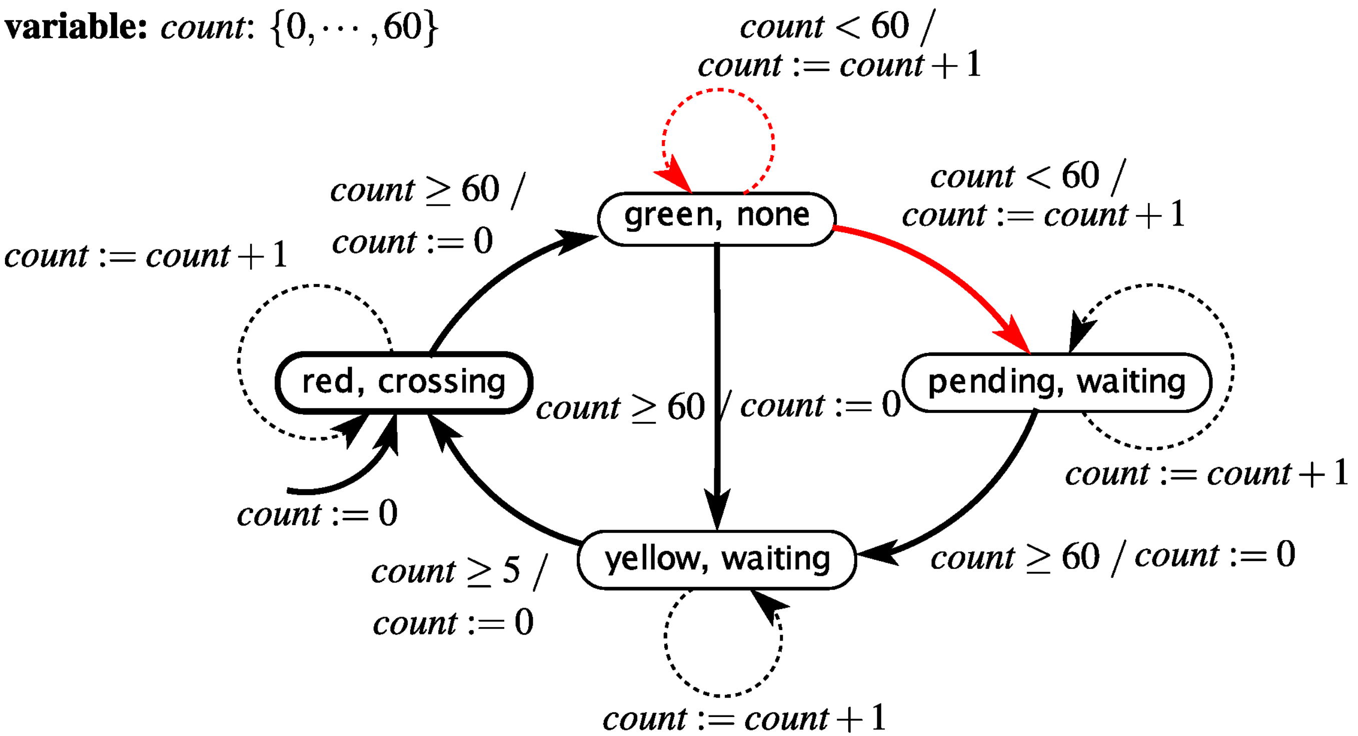


R, set of reachable states, represented by:

$$(v_1 = \text{red} \wedge v_2 = \text{crossing} \wedge \text{count} = 0)$$

Symbolic Model Checking Example

Property: $G(\neg(green \wedge crossing))$

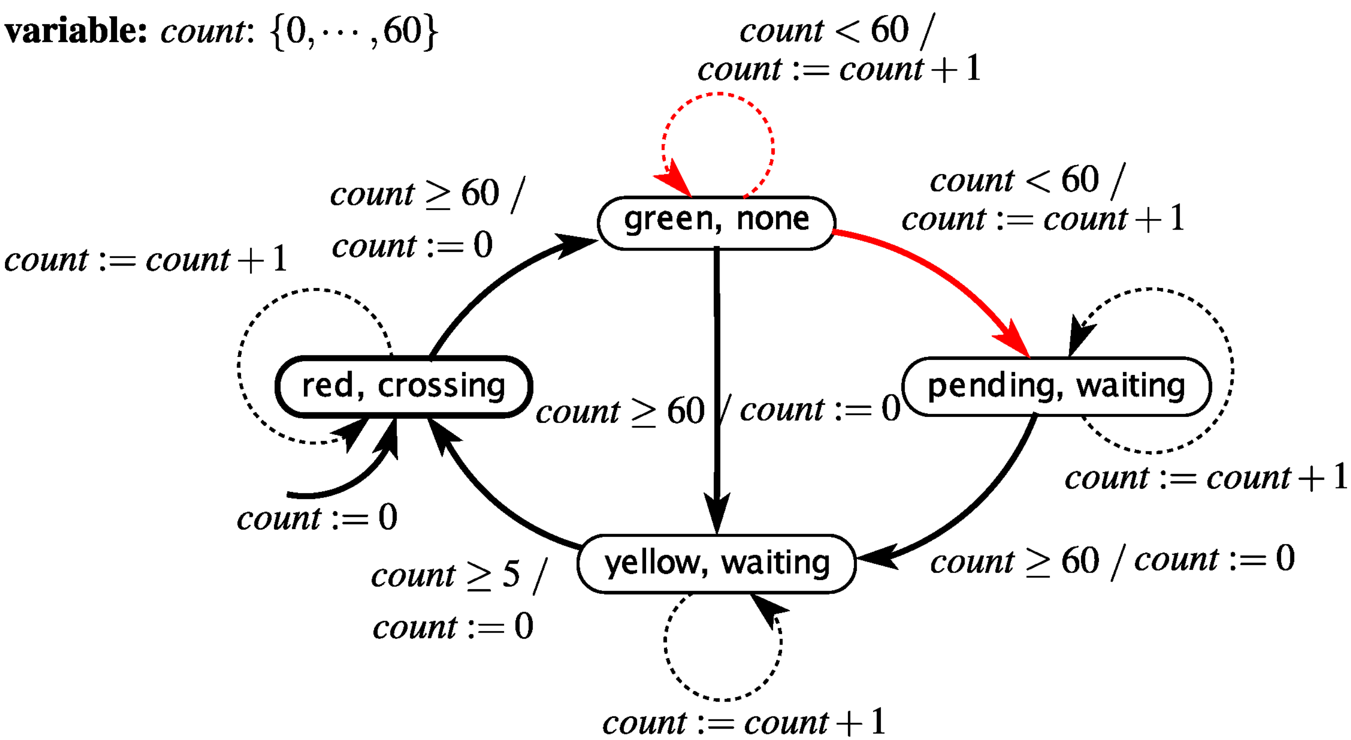


R, set of reachable states, represented by:

$$(v_1 = \text{red} \wedge v_2 = \text{crossing} \wedge 0 \leq \text{count} \leq 1)$$

Symbolic Model Checking Example

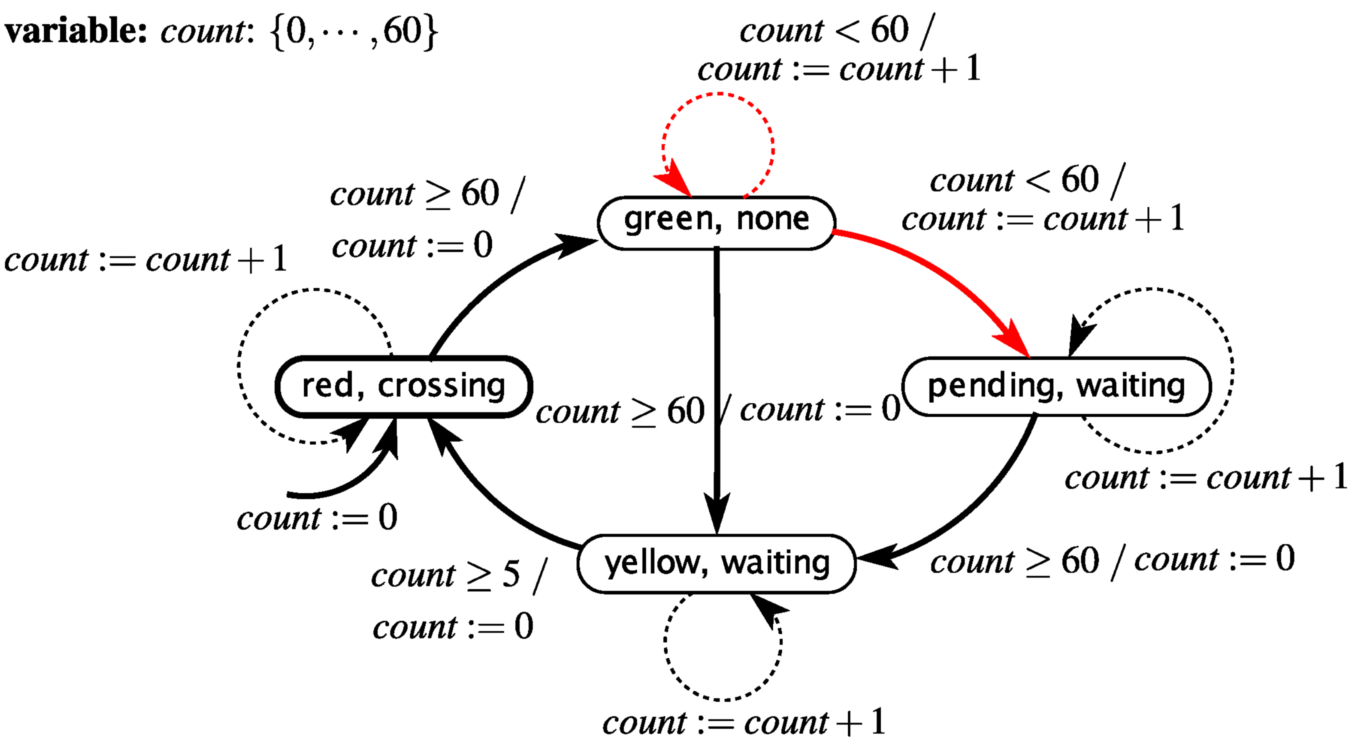
Property: $G(\neg(green \wedge crossing))$



R, set of reachable states, represented by: $(v_1 = \text{red} \wedge v_2 = \text{crossing} \wedge 0 \leq \text{count} \leq 60)$

Symbolic Model Checking Example

Property: $G(\neg(green \wedge crossing))$

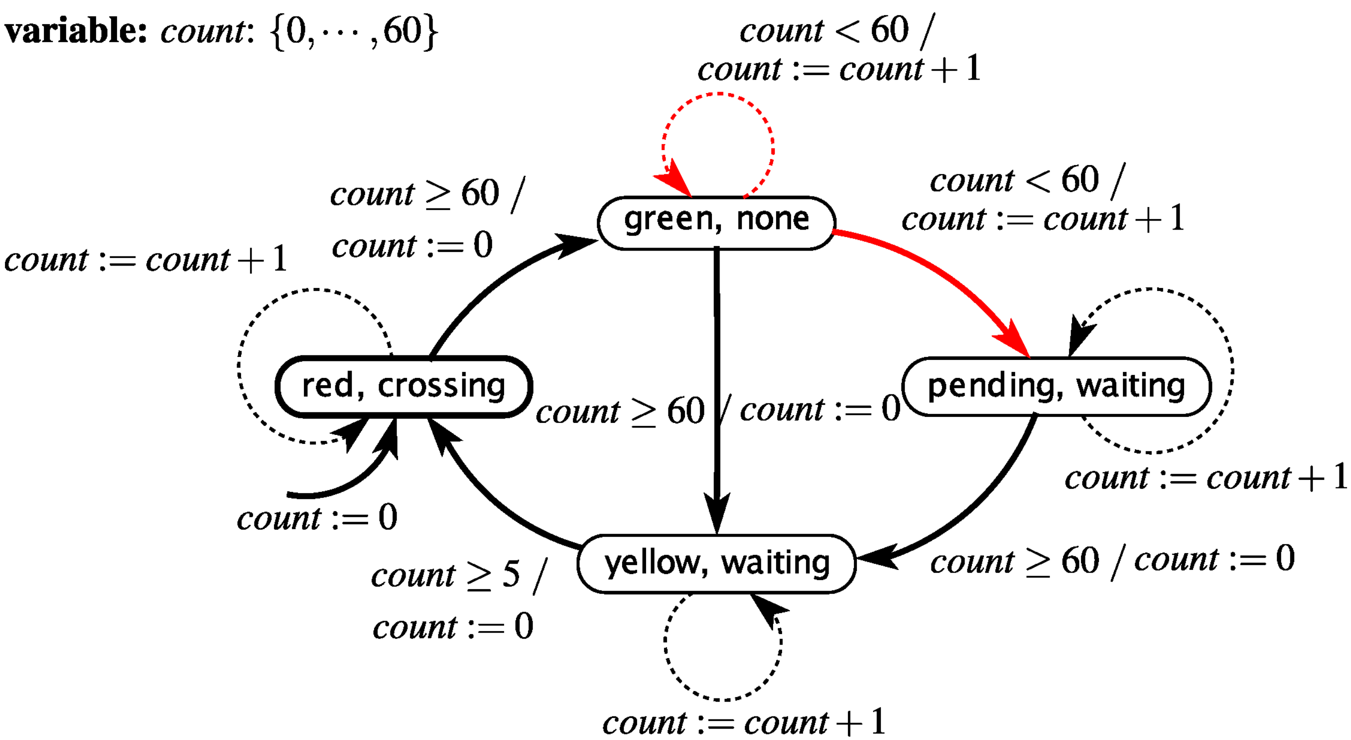


R, set of reachable states, represented by:

$$(v_1 = \text{red} \wedge v_2 = \text{crossing} \wedge 0 \leq \text{count} \leq 60)$$
$$\vee(v_1 = \text{green} \wedge v_2 = \text{none} \wedge \text{count} = 0)$$

Symbolic Model Checking Example

Property: $G(\neg(green \wedge crossing))$

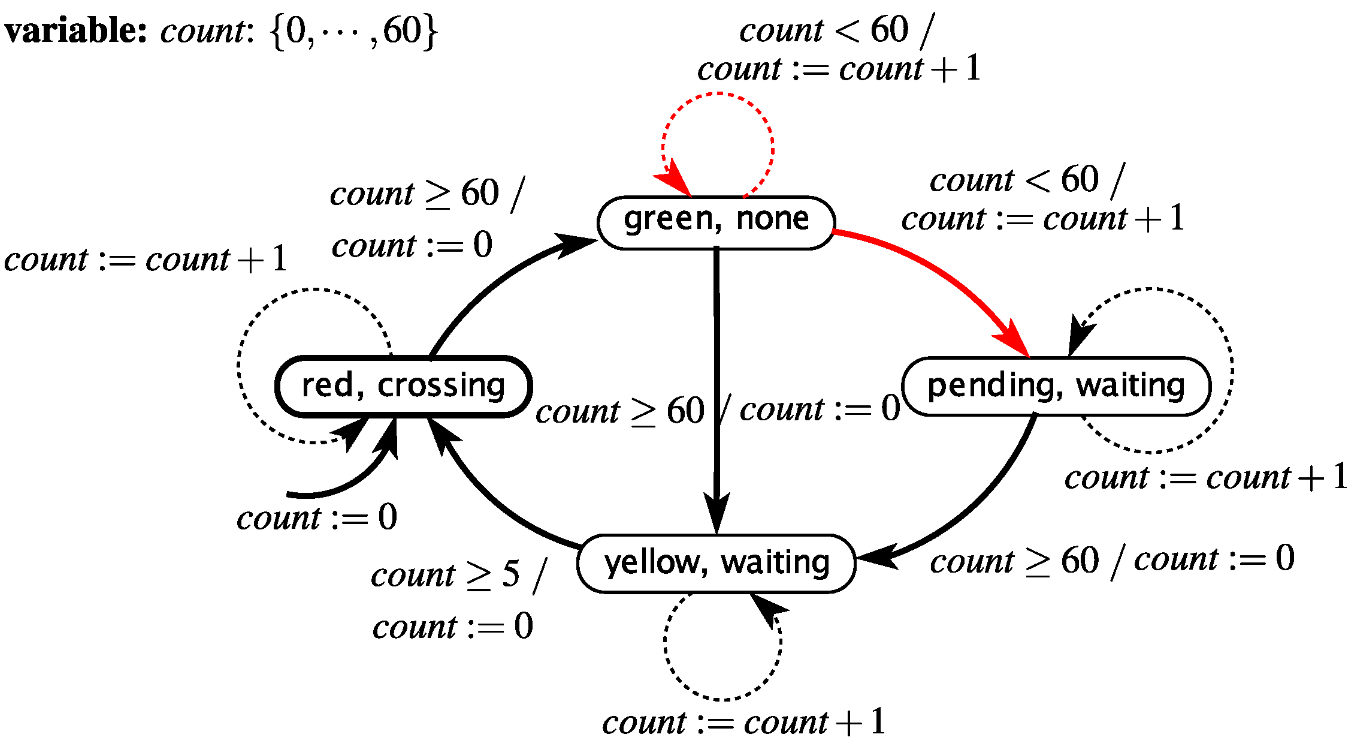


R, set of reachable states, represented by:

$$\begin{aligned}&(v_1 = \text{red} \wedge v_2 = \text{crossing} \wedge 0 \leq \text{count} \leq 60)\\ &\vee(v_1 = \text{green} \wedge v_2 = \text{none} \wedge 0 \leq \text{count} \leq 1)\\ &\vee(v_1 = \text{pending} \wedge v_2 = \text{waiting} \wedge \text{count} = 1)\end{aligned}$$

Symbolic Model Checking Example

Property: $G(\neg(green \wedge crossing))$

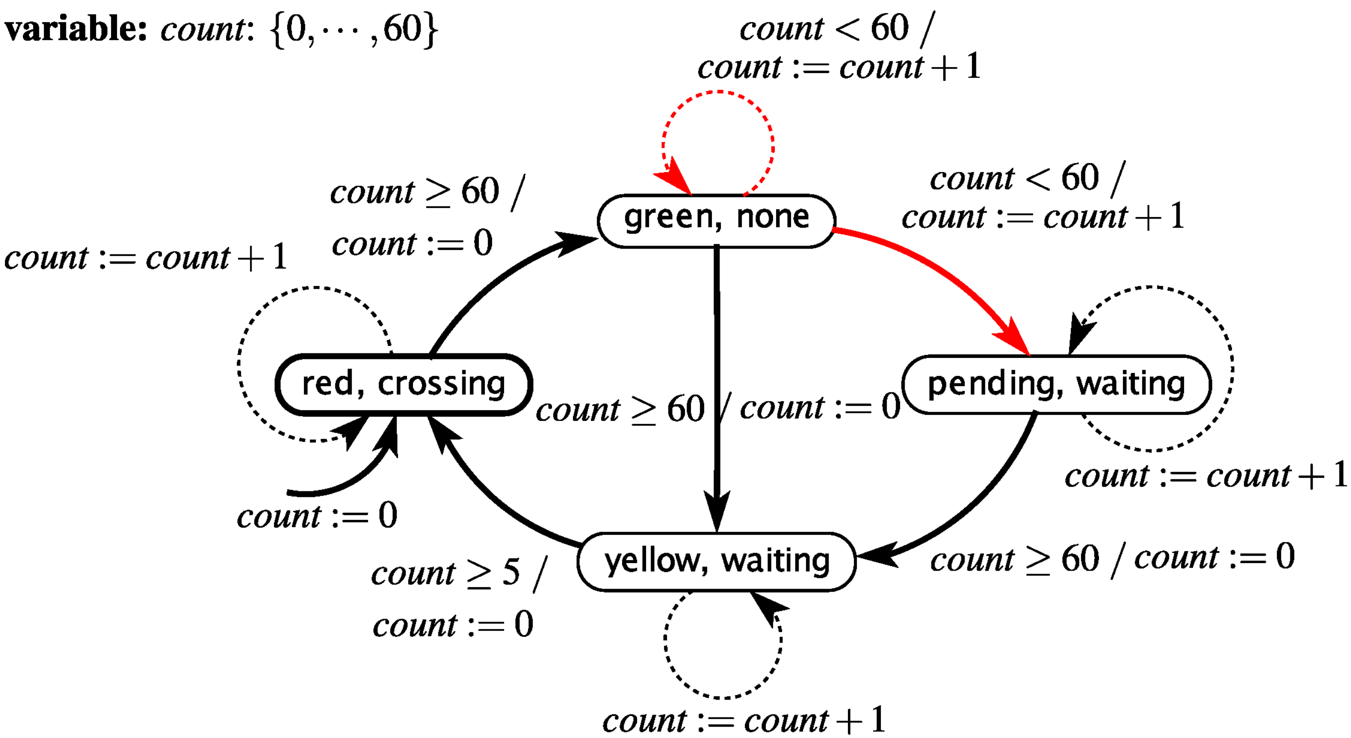


R, set of reachable states, represented by:

$$(v_1 = \text{red} \wedge v_2 = \text{crossing} \wedge 0 \leq \text{count} \leq 60)$$
$$\vee(v_1 = \text{green} \wedge v_2 = \text{none} \wedge 0 \leq \text{count} \leq 60)$$
$$\vee(v_1 = \text{pending} \wedge v_2 = \text{waiting} \wedge 0 \leq \text{count} \leq 60)$$

Symbolic Model Checking Example

Property: $G(\neg(green \wedge crossing))$

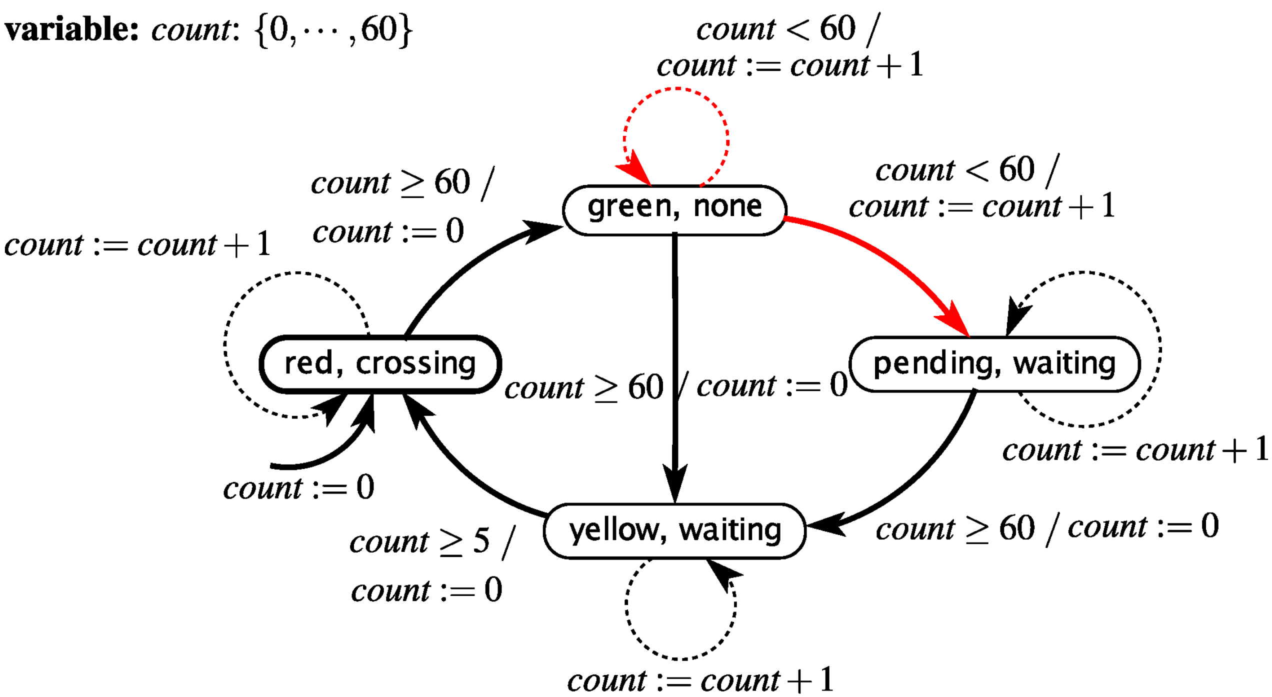


R, set of reachable states, represented by:

$$
\begin{aligned}
&(v_1 = \text{red} \wedge v_2 = \text{crossing} \wedge 0 \leq \text{count} \leq 60) \\
\vee&(v_1 = \text{green} \wedge v_2 = \text{none} \wedge 0 \leq \text{count} \leq 60) \\
\vee&(v_1 = \text{pending} \wedge v_2 = \text{waiting} \wedge 0 \leq \text{count} \leq 60) \\
\vee&(v_1 = \text{yellow} \wedge v_2 = \text{waiting} \wedge \leq \text{count} = 0)
\end{aligned}
$$

Symbolic Model Checking Example

Property: $G(\neg(green \wedge crossing))$



R, set of reachable states, represented by:

$$
\begin{aligned}
&(v_1 = \text{red} \wedge v_2 = \text{crossing} \wedge 0 \le \text{count} \le 60) \\
\vee&(v_1 = \text{green} \wedge v_2 = \text{none} \wedge 0 \le \text{count} \le 60) \\
\vee&(v_1 = \text{pending} \wedge v_2 = \text{waiting} \wedge 0 \le \text{count} \le 60) \\
\vee&(v_1 = \text{yellow} \wedge v_2 = \text{waiting} \wedge 0 \le \text{count} \le 5)
\end{aligned}
$$

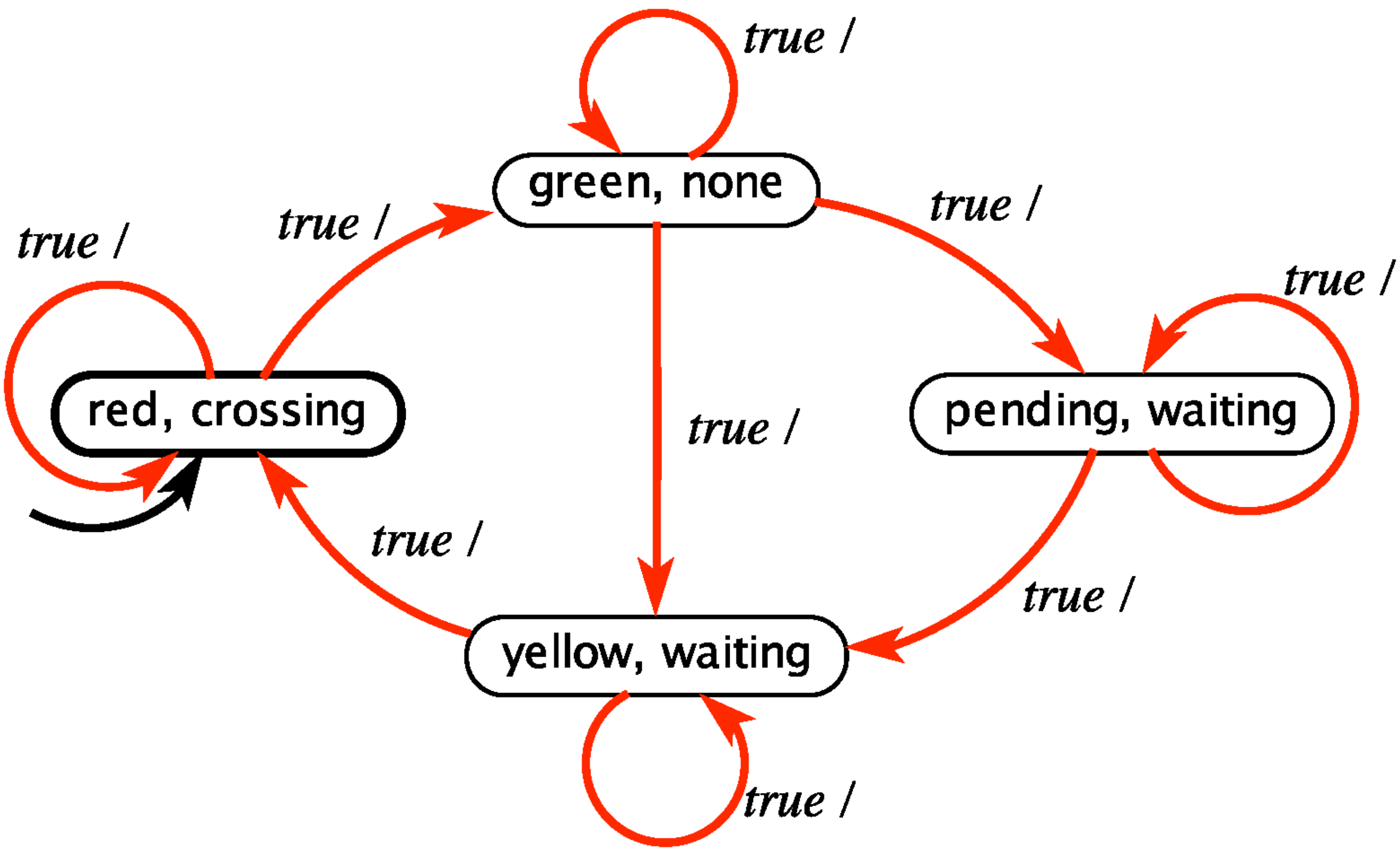
Abstraction in Model Checking

- 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 which details can be abstracted away.
- A simple and useful approach is called *localisation abstraction*.
- A localisation abstraction hides state variables that are irrelevant to the property being verified.

Abstract Model for Traffic Light Example

Property: $G(\neg(green \wedge crossing))$

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 15.4 of the text for more information.

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

$\phi_i =$ robot picks up item i, 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:

Variant: Suppose we have a robot that must pick up multiple things, ***in a specified order***

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

How would you state this goal in temporal logic?

Controller Synthesis

Goal to be achieved is:

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

Consider the first part alone:

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

How can we use model checking to synthesise a control strategy?

$$\mathbf{F}(\phi_1)$$

Controller Synthesis

Recall that:

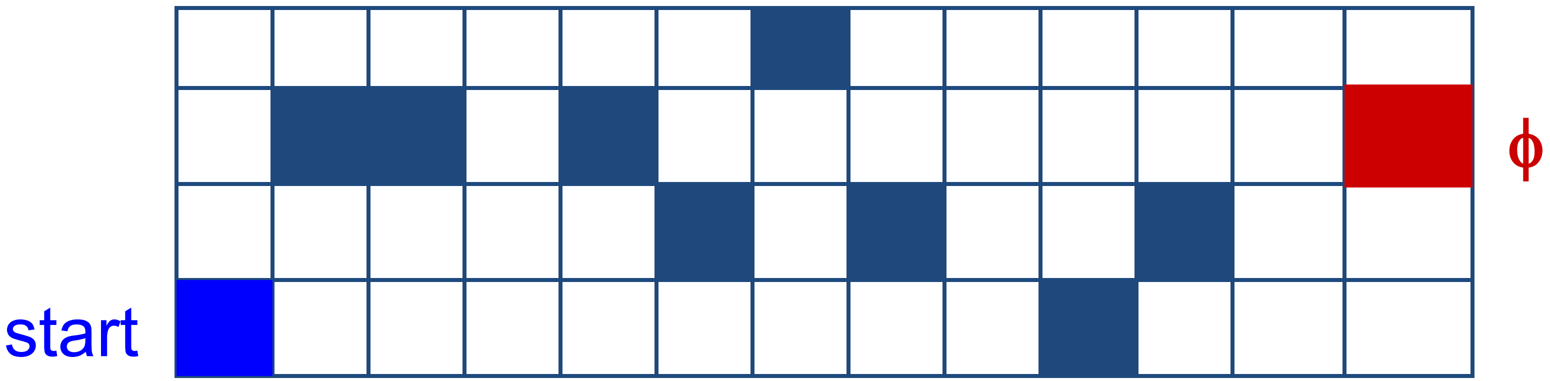
$$\mathbf{F}(\phi_1) = \neg\mathbf{G}(\neg\phi_1)$$

Therefore, we can construct a counterexample to:

$$\mathbf{G}(\neg\phi_1)$$

The counterexample is a trace that gets the robot to the desired point.

A robot delivery service



ϕ = destination for robot

<u>At any time step:</u>

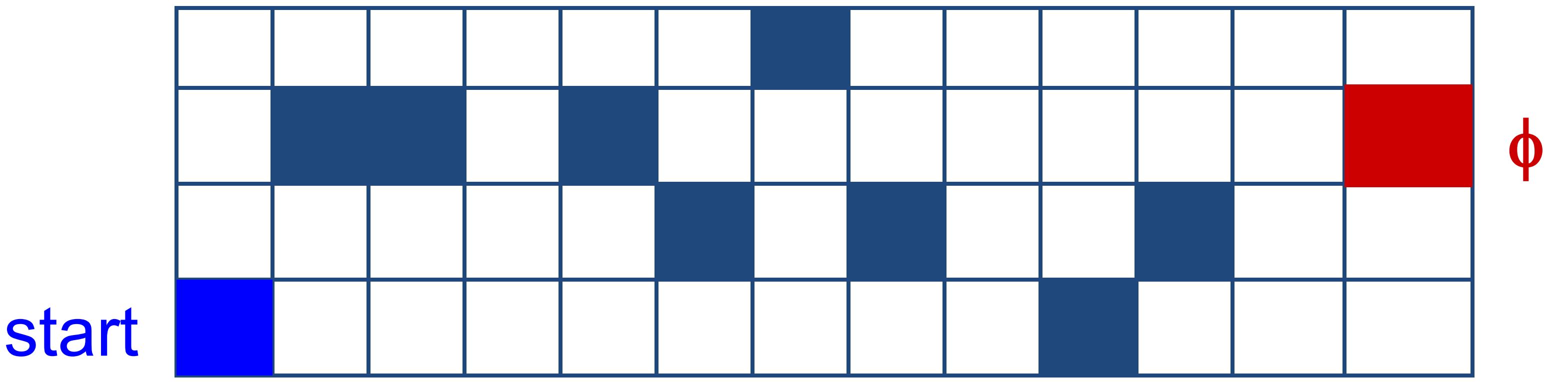
Robot can move Left, Right, Up, Down, Stay Put

Can model Robot as an FSM

→ Robot state = its position,

→ Number of states?

A robot delivery service



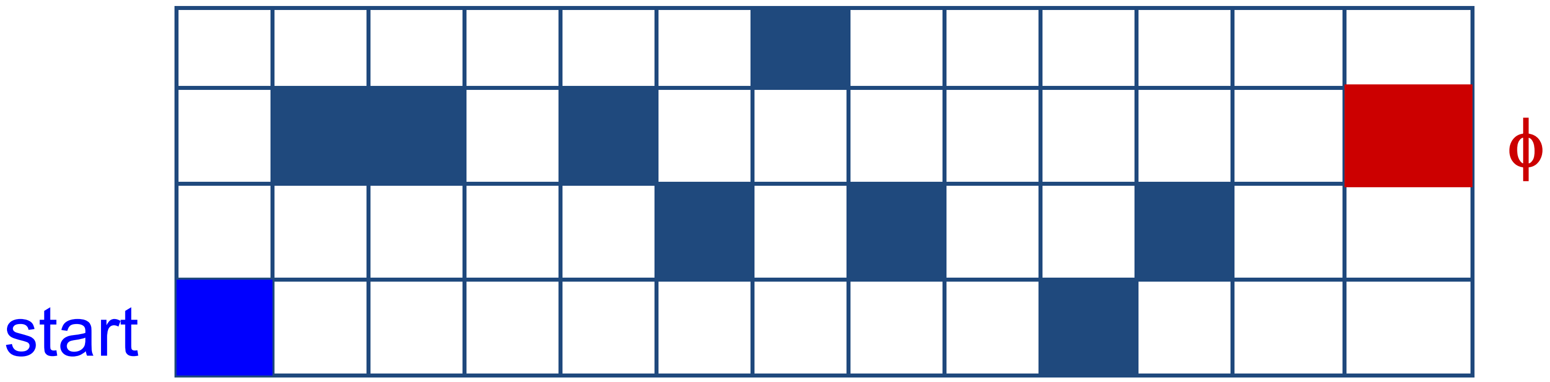
ϕ = robot delivers item to destination

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?

A robot delivery service, with moving obstacles



ϕ = destination for robot

<u>At any time step:</u>

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

Environment can move one obstacle Up or Down or Stay Put

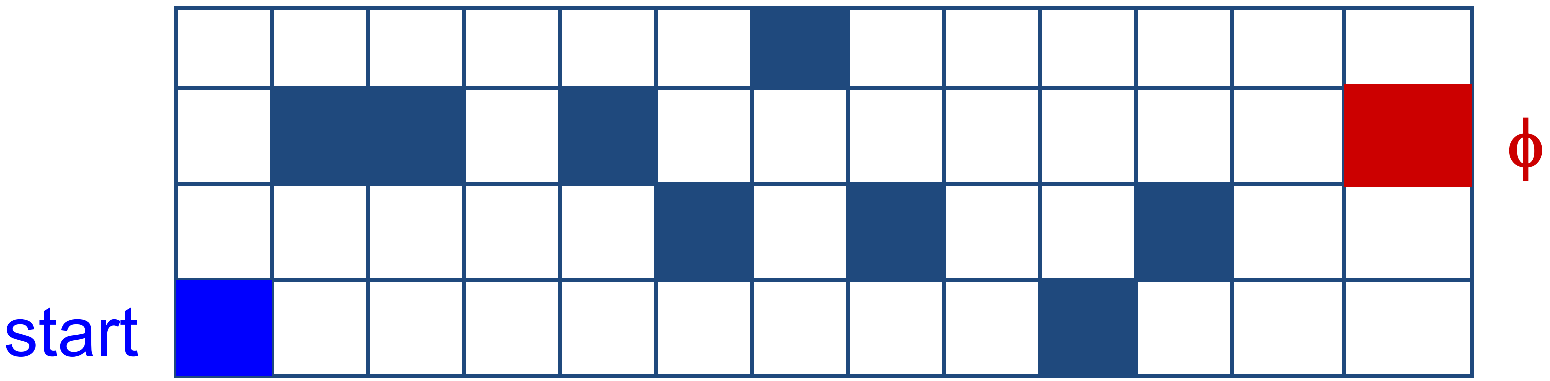
→ But only 3 times total

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 = destination for robot

<u>At any time step:</u>

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

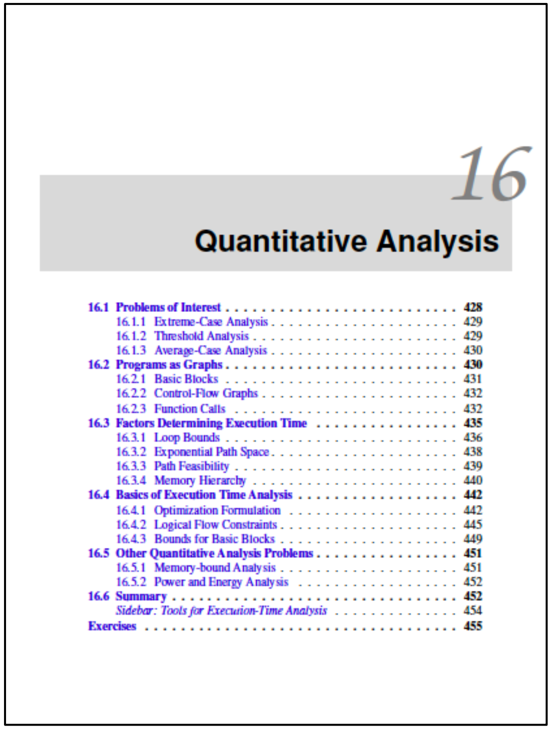
Environment can move one obstacle Up or Down or Stay Put

→ But only 3 times total

How to find an environment policy to prevent ϕ?

Things to do …

- Next lecture: Quantitative Analysis
- Read Chapter 16



16

Quantitative Analysis