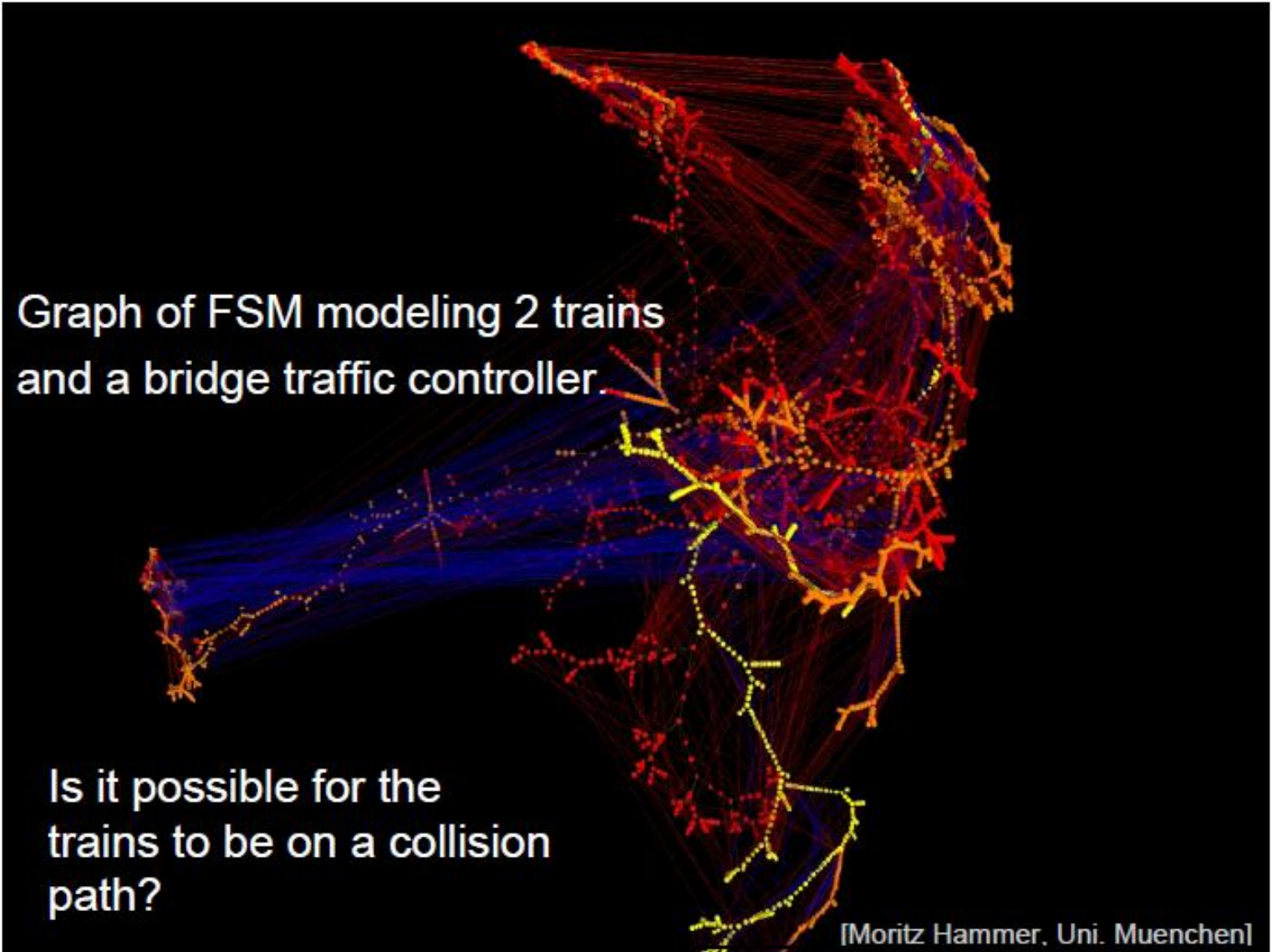


# Embedded Systems Design and Modeling



## Chapter 15 Reachability Analysis

# A Glimpse of Reality



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]

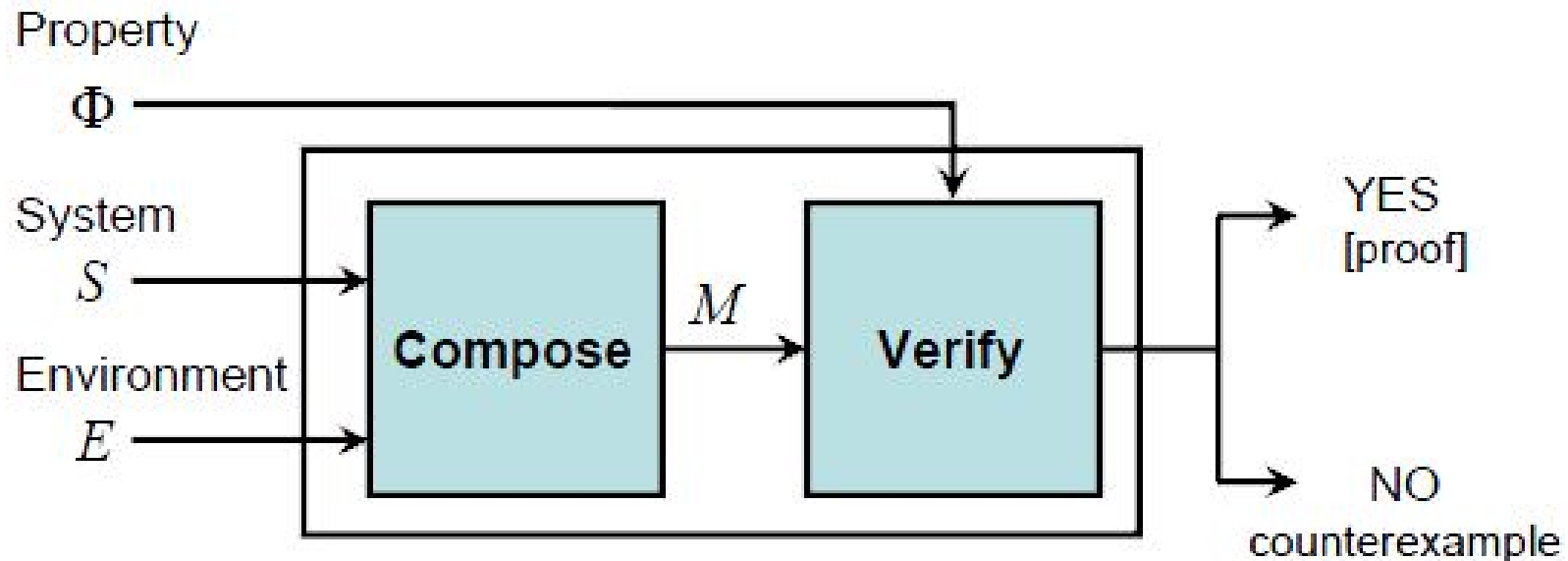
# Reachability Analysis and Model Checking

---

- Reachability analysis: the process of computing the set of reachable states for a system.
- Model checking: an algorithmic method for determining if a system satisfies a formal specification expressed in temporal logic.
- Model checking typically performs reachability analysis.

# Formal Verification

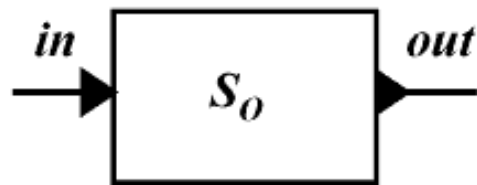
- A system (or systematic method) to formally (mathematically) verify whether a design meets its spec in the environment



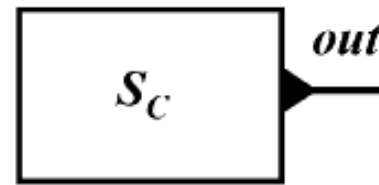
# Open vs. Closed Systems

---

- A closed system has no inputs:



(a) Open system



(b) Closed system

- For verification, we obtain a closed system by composing the system and environment models
  - Somewhat similar to testbench in HW design

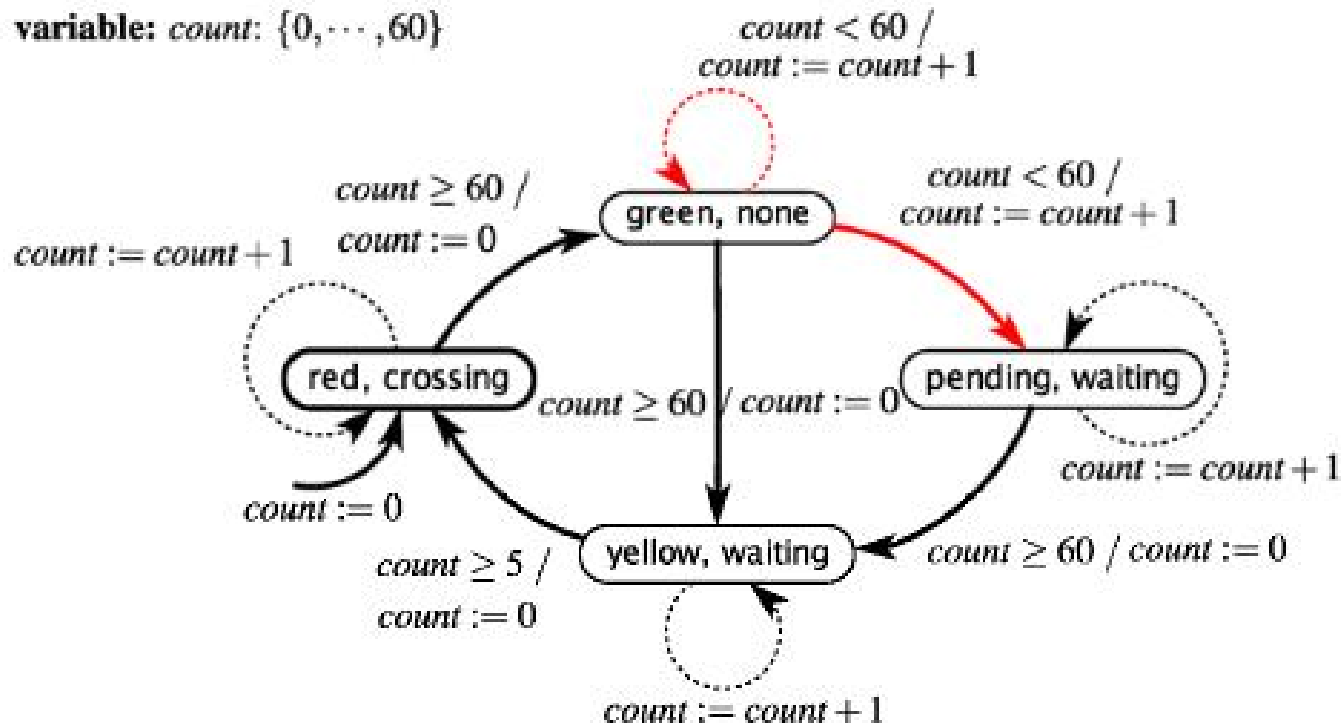
# Model Checking $G p$

---

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

# Large State Space Example

- The composed FSM for traffic light controller
- Property:  $G (\neg (\text{green} \wedge \text{crossing}))$ :
  - The FSM has 188 states (accounting for different values of count)
  - Prohibitively large state space



# Graph Traversal

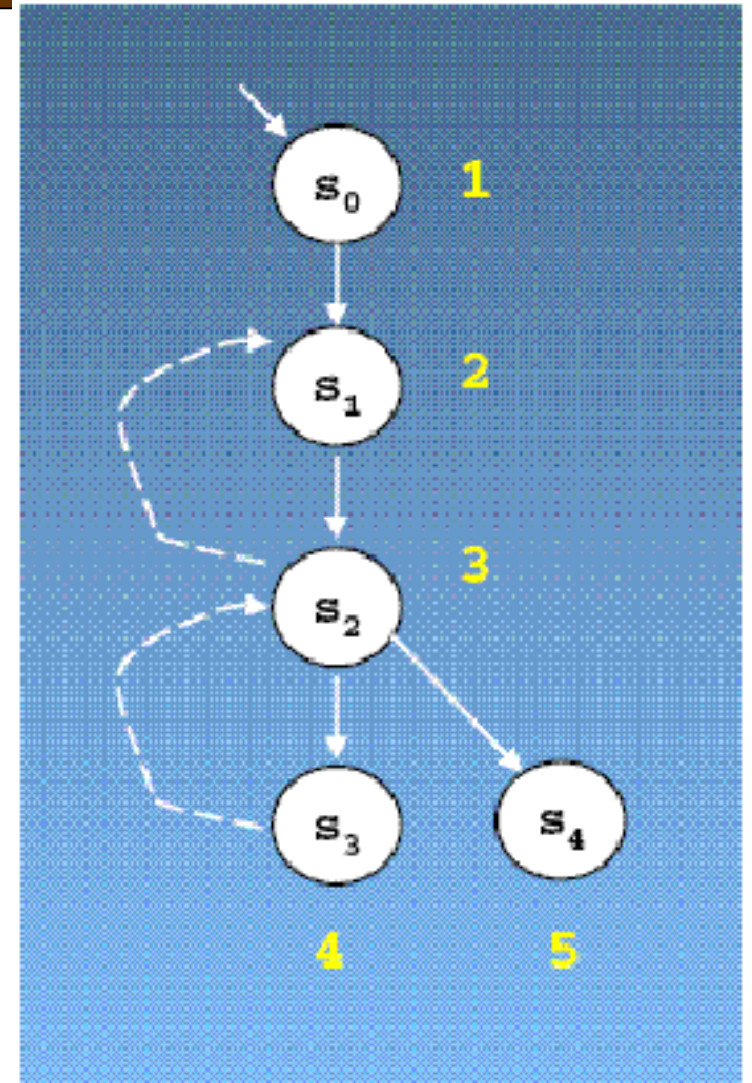
---

- Continue with 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
- Use a common graph traversal algorithm
- Depth-first search algorithm often used as shown in the next slide



# Depth-First Search (DFS)

- Maintain 2 data structures:
  1. Set of visited states  $R$
  2. A stack with the path from the initial state to the existing state
- Look for counterexample(s)
- Potential problems for a huge graph?



# DFS Approach (Continued)

---

- 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
- Traffic light controller example:
  - $R = \{ (\text{red}, \text{crossing}, 0), (\text{red}, \text{crossing}, 1), \dots (\text{red}, \text{crossing}, 60),$   
 $(\text{green}, \text{none}, 0), (\text{green}, \text{none}, 1), \dots, (\text{green}, \text{none}, 60),$   
 $(\text{yellow}, \text{waiting}, 0), \dots (\text{yellow}, \text{waiting}, 5),$   
 $(\text{pending}, \text{waiting}, 1), \dots, (\text{pending}, \text{waiting}, 60) \}$

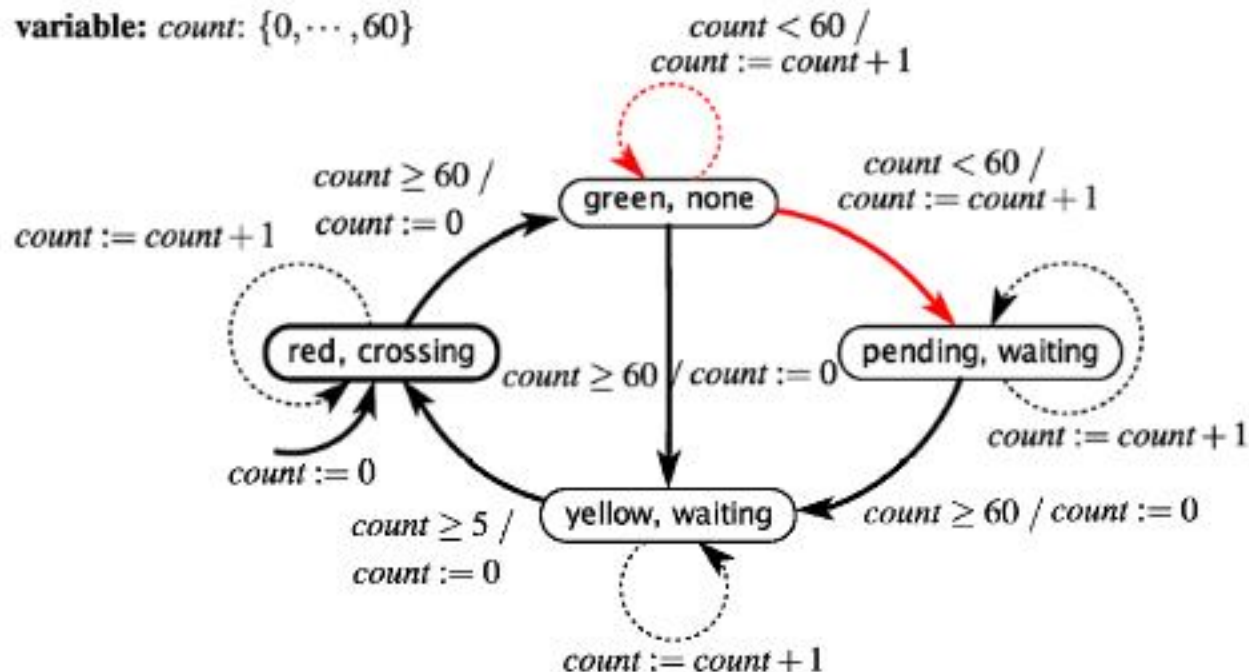
# Alternative: The Symbolic Approach

---

- 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 linear temporal logic or Boolean algebra
- Example in the next slide

# Symbolic Model Checking Example

- Property:  $G (\neg (\text{green} \wedge \text{crossing}))$ :
  - Assume  $v_l$ : traffic light state and  $v_p$ : pedestrian state
  - $R = (v_l = \text{red} \wedge v_p = \text{crossing} \wedge 0 \leq \text{count} \leq 60)$   
or  $(v_l = \text{green} \wedge v_p = \text{none} \wedge 0 \leq \text{count} \leq 60)$   
or  $(v_l = \text{pending} \wedge v_p = \text{waiting} \wedge 1 \leq \text{count} \leq 60)$   
or  $(v_l = \text{yellow} \wedge v_p = \text{waiting} \wedge 0 \leq \text{count} \leq 5)$



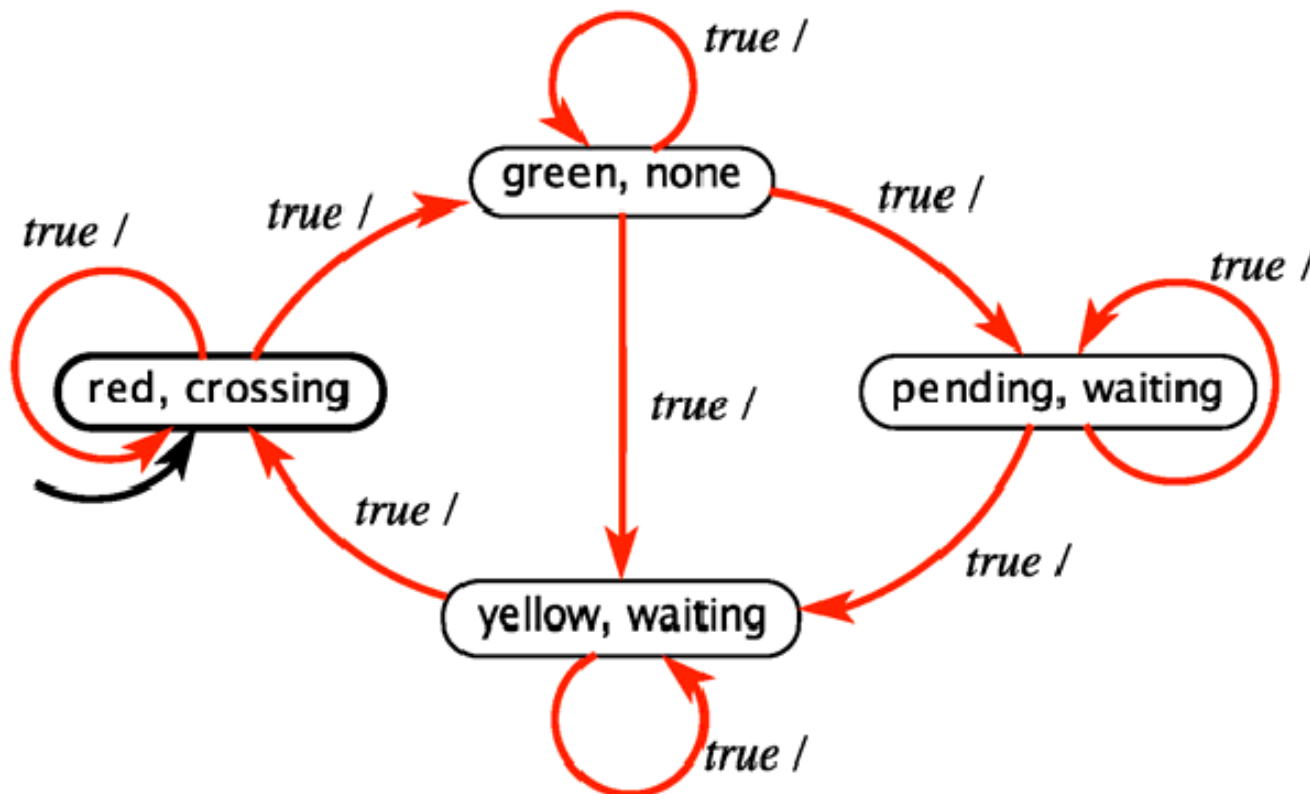
# Abstraction in Model Checking

---

- ❑ Should use simplest model of a system that provides proof of safety.
- ❑ Simpler models have smaller state spaces and are 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 of Traffic Light Controller

- Property:  $G (\neg (\text{green} \wedge \text{crossing}))$
- What variable can be abstracted?
  - The count



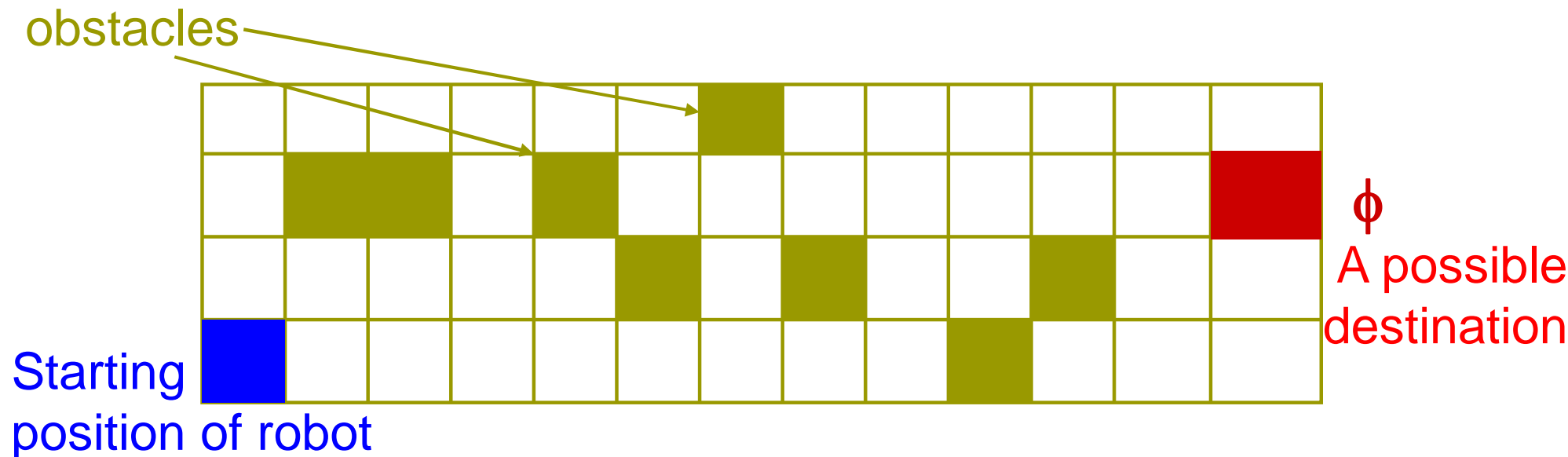
# Example Comparisons

---

- Every behavior of  $M$  can be exhibited by the symbolic version ( $M_{abs}$ ).
  - But opposite is not true!
- Even with the abstraction,  $M_{abs}$  satisfies the  $G (\neg (\text{green} \wedge \text{crossing}))$  property
  - The value of count is irrelevant for this property.
- $M$  has 188 states,  $M_{abs}$  has only 4 states.
- Reachability analysis on  $M_{abs}$  is far easier than for  $M$  as we have far fewer states to explore.

# Another Reachability Example

- A robot delivery service with obstacles:
  - Suppose we have a robot that must pick up multiple things, in any order.





# Reachability Example (Cont.)

---

- How would you state this goal in temporal logic?

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

- The goal to be achieved:  $\mathbf{F}\phi_1 \wedge \mathbf{F}\phi_2 \wedge \dots \wedge \mathbf{F}\phi_n$
- How can we find a strategy to achieve this goal?
  - Do repeated reachability analysis, first from  $\Phi_0$  to reach  $\Phi_1$ , then from  $\Phi_1$  to reach  $\Phi_2$ , then  $\Phi_2$  to reach  $\Phi_3$
  - Problem: What if  $\Phi_2$  is not reachable from  $\Phi_1$ , but reachable from  $\Phi_0$ ?

# More Complicated Example

---

- What if we have a robot that must pick up multiple things, in a specific order?
- The goal to be achieved:

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

# 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.

# Liveness Checking Example

---

- Suppose:
  - An FSM M1 models a system that executes forever and produces a pure output h (for heartbeat)
  - It is required to produce this output at least once every three reactions. That is, if in two successive reactions it fails to produce the output h, then in the third it must.

# Liveness Checking Example (Cont.)

---

- The goal to be achieved:

$$\mathbf{G}(h \vee \mathbf{X}h \vee \mathbf{X}^2h)$$

- Reminder:  $\mathbf{G}\Phi = \neg\mathbf{F}\neg\Phi$
- Its negated form:

$$\mathbf{F}(\neg h \wedge \mathbf{X}\neg h \wedge \mathbf{X}^2\neg h)$$

- It is sufficient to find one counterexample that this doesn't hold!

# Homework Assignments

---

- Chapter 15: your choice
- Due date: any time before final exam