CS181

Applied Logic and Automated Reasoning

Spring 2020

Instructor: Lucas Bang

Quick Introduction: About Me

Quick Introduction: About Me

- Grew up in Las Vegas

Quick Introduction: About Me

- Grew up in Las Vegas
- UNLV: Math and CS

UNLV

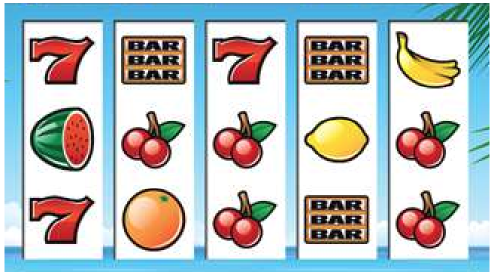
Quick Introduction: About Me

- Grew up in Las Vegas
- UNLV: Math and CS
- In parallel:

  UNLV MS Computer Science

Quick Introduction: About Me

- Grew up in Las Vegas



- UNLV: Math and CS
- In parallel:

UNLV MS Computer Science

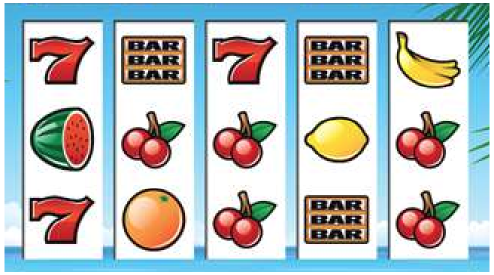
Test and Verification Engineer in Casino Gaming

Quick Introduction: About Me

- Grew up in Las Vegas
- UNLV: Math and CS
- In parallel:

  UNLV MS Computer Science

  Test and Verification Engineer in Casino Gaming

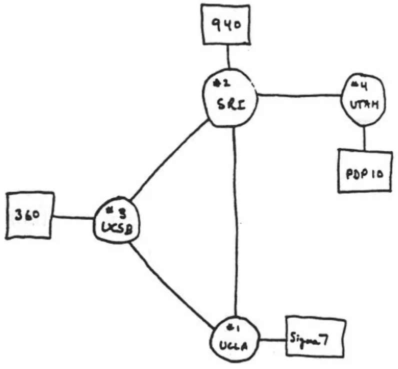
$E($  $) < 0$

Quick Introduction: About Me

- Grew up in Las Vegas
- UNLV: Math and CS
- In parallel:

  UNLV MS Computer Science

  Test and Verification Engineer in Casino Gaming
- PhD, UC Santa Barbara



THE ARPA NETWORK

DEC 1969

4 NODES

Quick Introduction: About Me

- Grew up in Las Vegas
- UNLV: Math and CS
- In parallel:

  UNLV MS Computer Science

  Test and Verification Engineer in Casino Gaming
- PhD, UC Santa Barbara
- HMC Prof. since fall 2018

What is this class about?

By the end of this class, I hope you have an appreciation for both how logic can be used to solve complex problems and how logic fits in the broader context of culture, history, and society.

Syllabus Time!

The rest of today's class:

Example of modeling a system with logic.

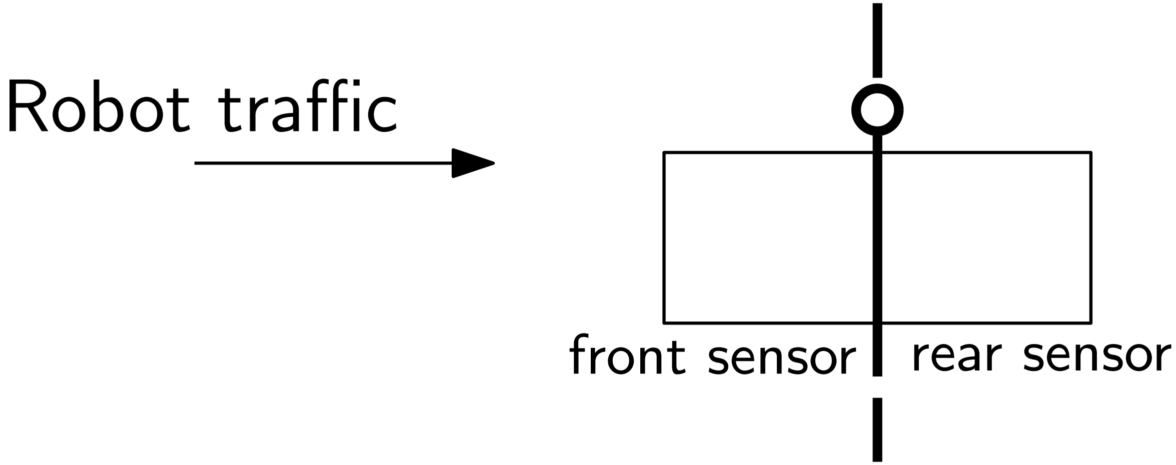
What logical properties might we care about?

Properties in temporal logic.

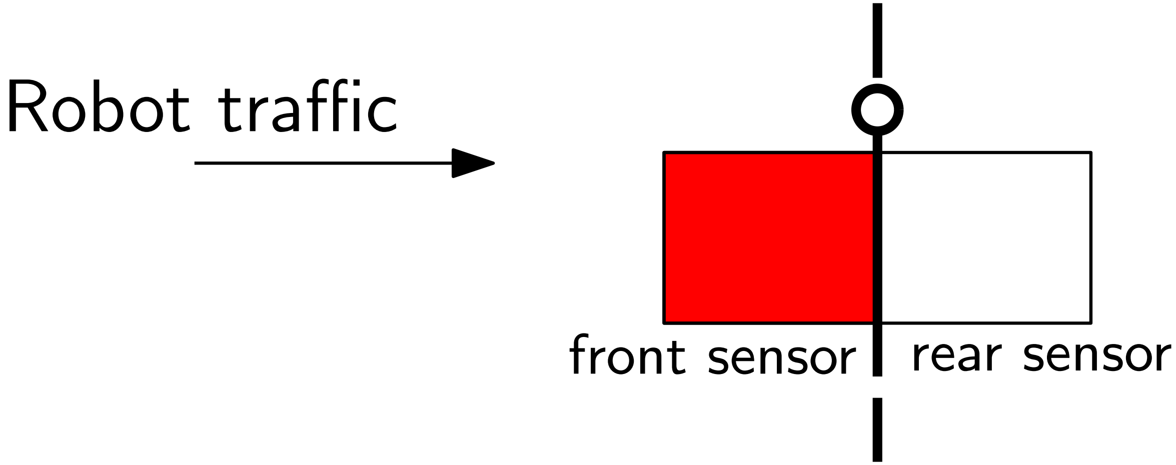
Using NuSMV to check properties.

Course technology and HW preview.

Example: an automatic door controller.

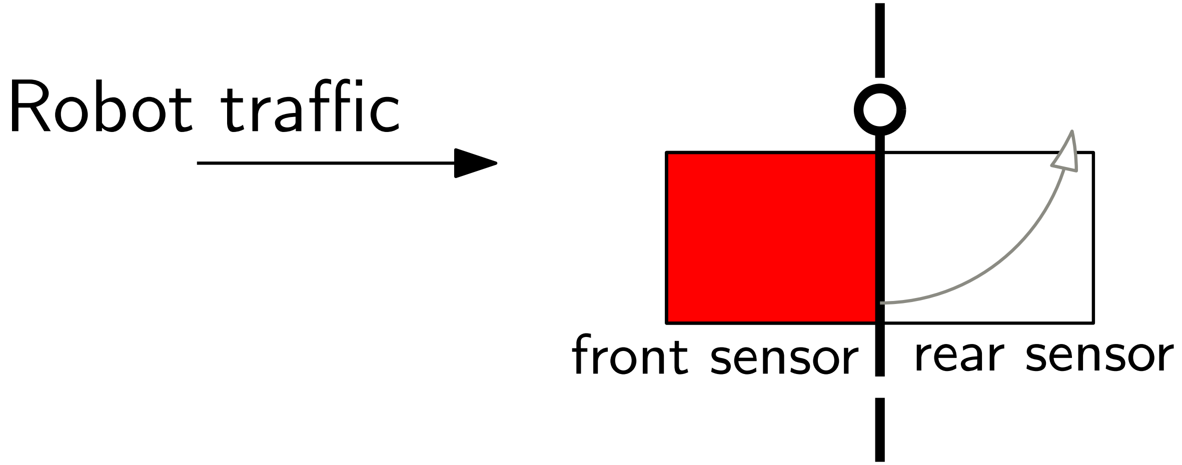


Example: an automatic door controller.



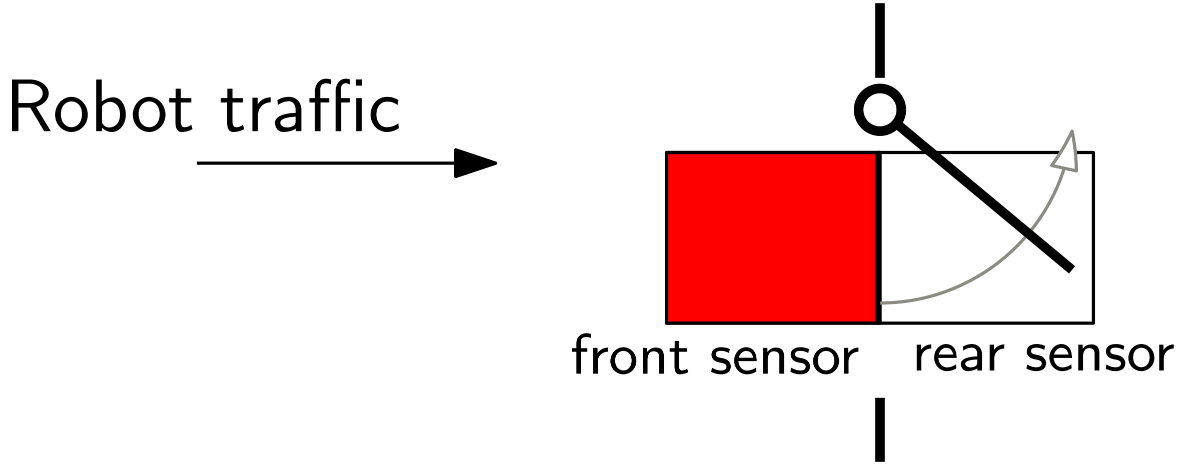
A robot can activate the front sensor.

Example: an automatic door controller.



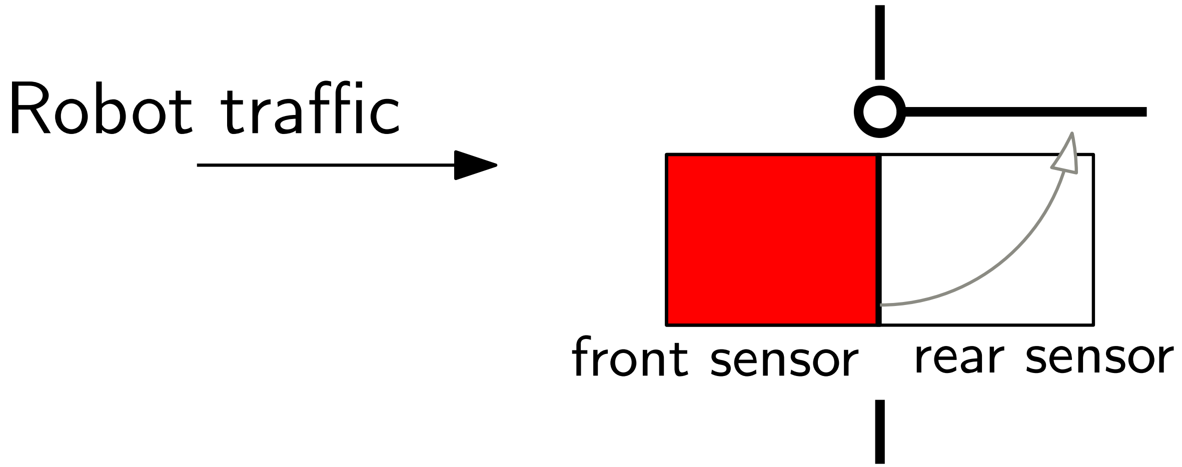
A robot can activate the front sensor.
Activating the front sensor opens the door.

Example: an automatic door controller.



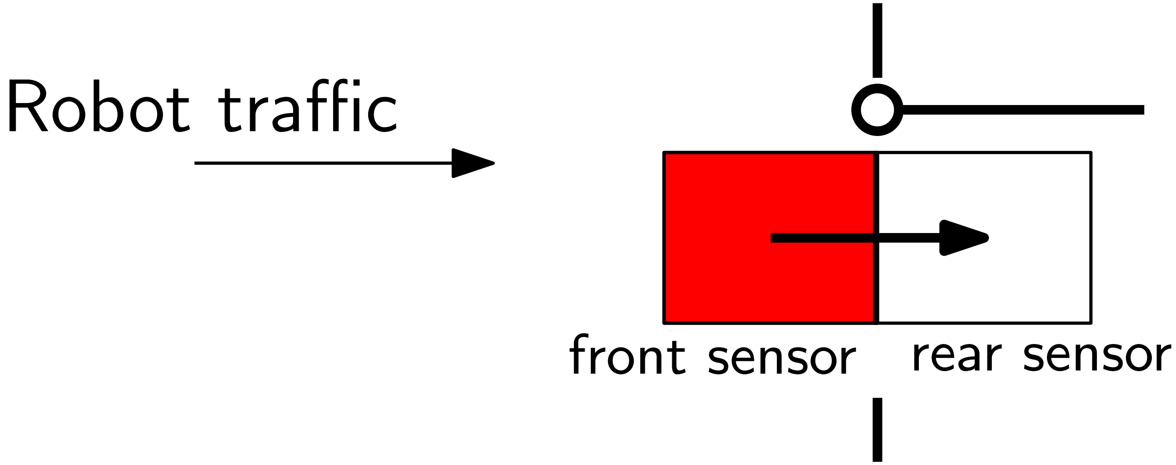
A robot can activate the front sensor.
Activating the front sensor opens the door.

Example: an automatic door controller.



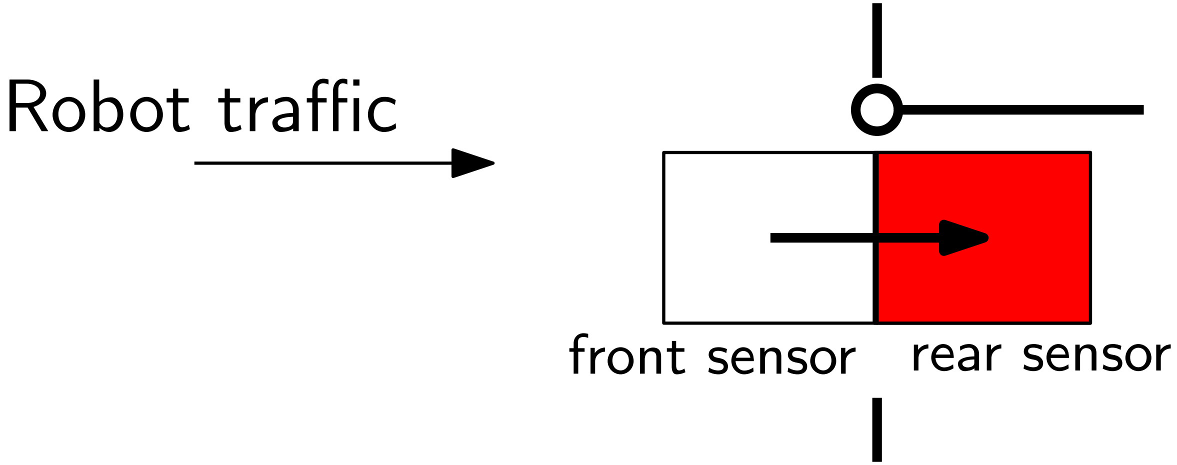
A robot can activate the front sensor.
Activating the front sensor opens the door.

Example: an automatic door controller.



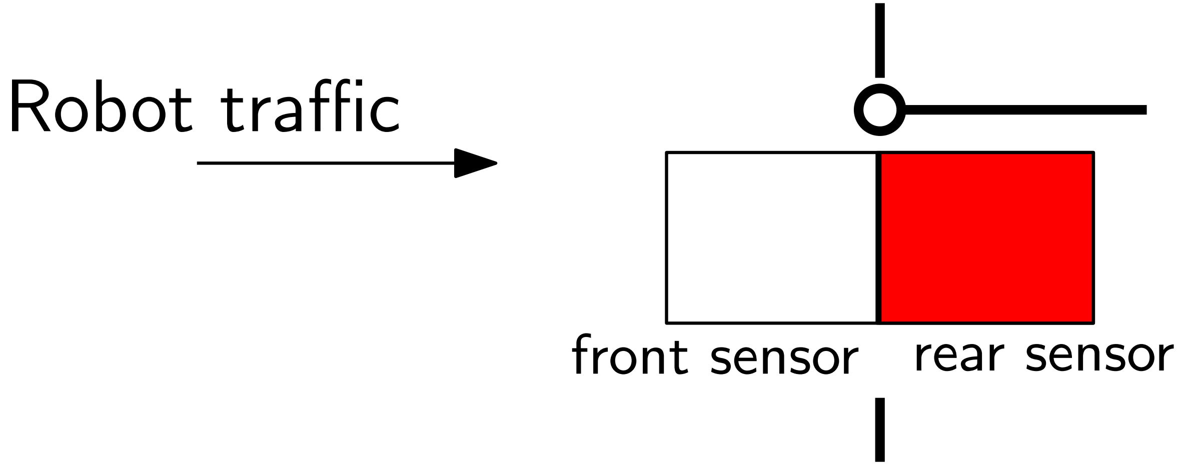
A robot can activate the front sensor.
Activating the front sensor opens the door.

Example: an automatic door controller.



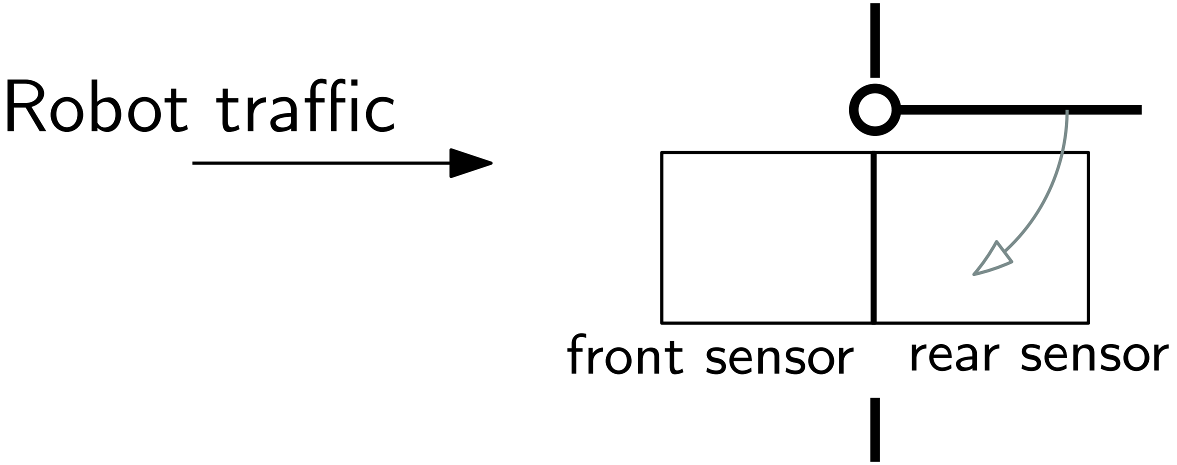
A robot can activate the front sensor.
Activating the front sensor opens the door.
Robot moves through door, activating rear sensor.

Example: an automatic door controller.



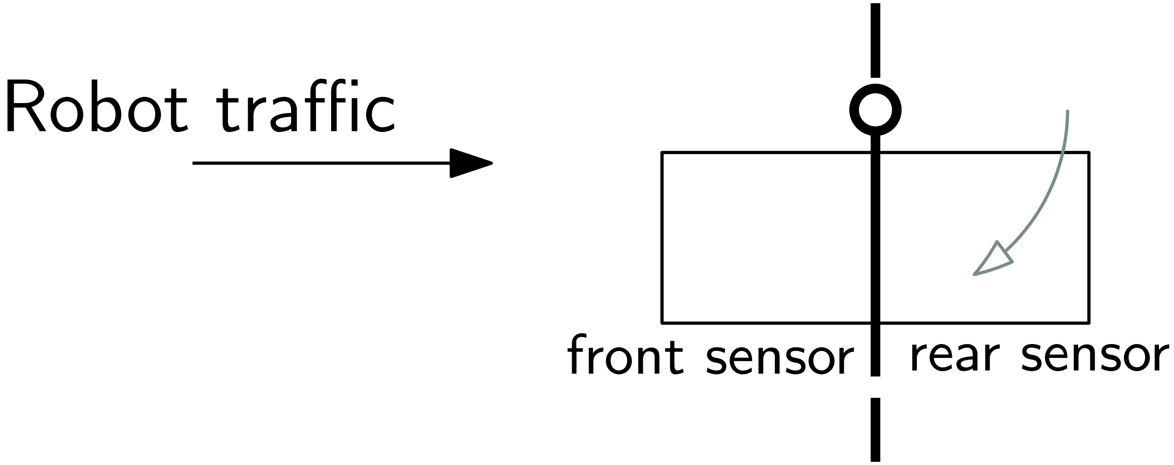
A robot can activate the front sensor.
Activating the front sensor opens the door.
Robot moves through door, activating rear sensor.
The door stays open until the robot moves “out of the way”.

Example: an automatic door controller.



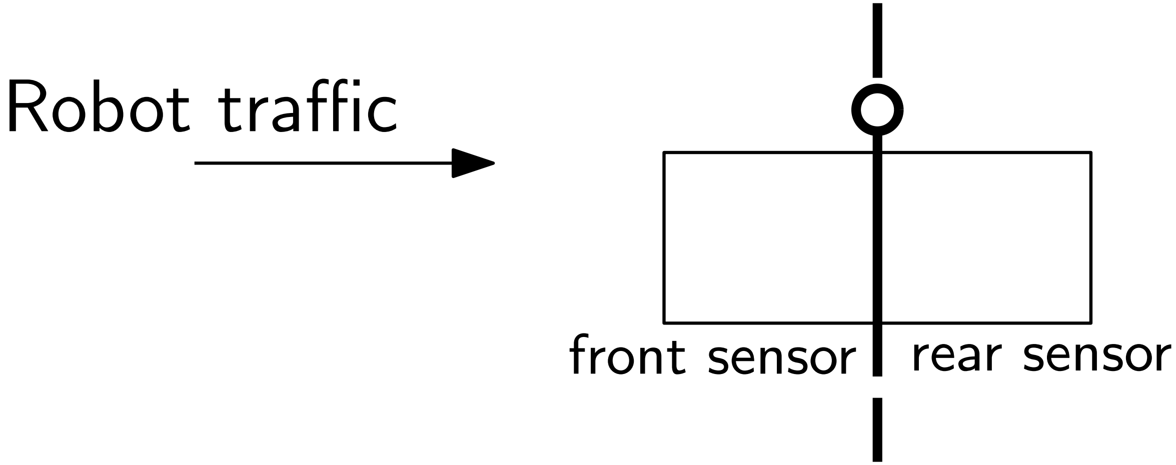
A robot can activate the front sensor.
Activating the front sensor opens the door.
Robot moves through door, activating rear sensor.
The door stays open until the robot moves “out of the way”.
The door closes.

Example: an automatic door controller.



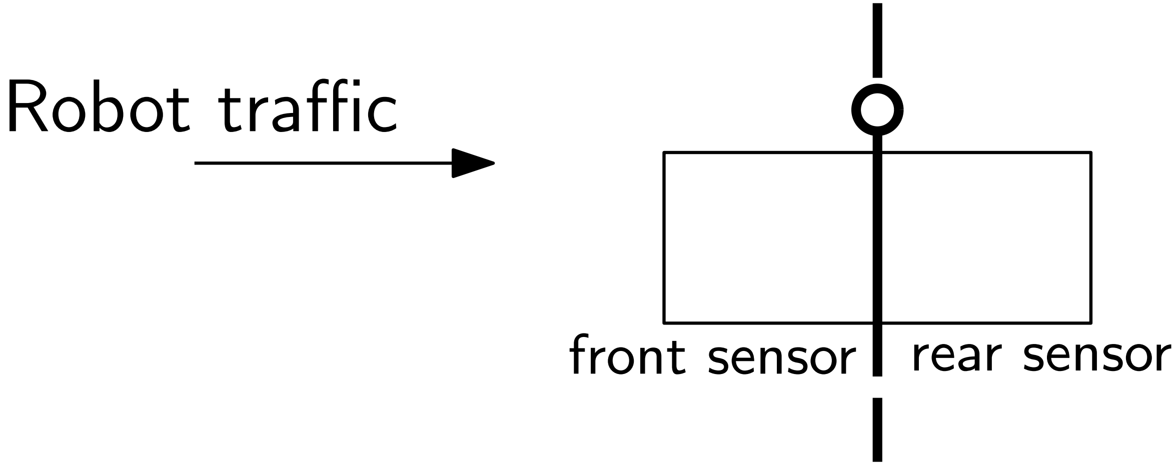
A robot can activate the front sensor.
Activating the front sensor opens the door.
Robot moves through door, activating rear sensor.
The door stays open until the robot moves “out of the way”.
The door closes.

Example: an automatic door controller.



Let's encode our intuition with a transition diagram.

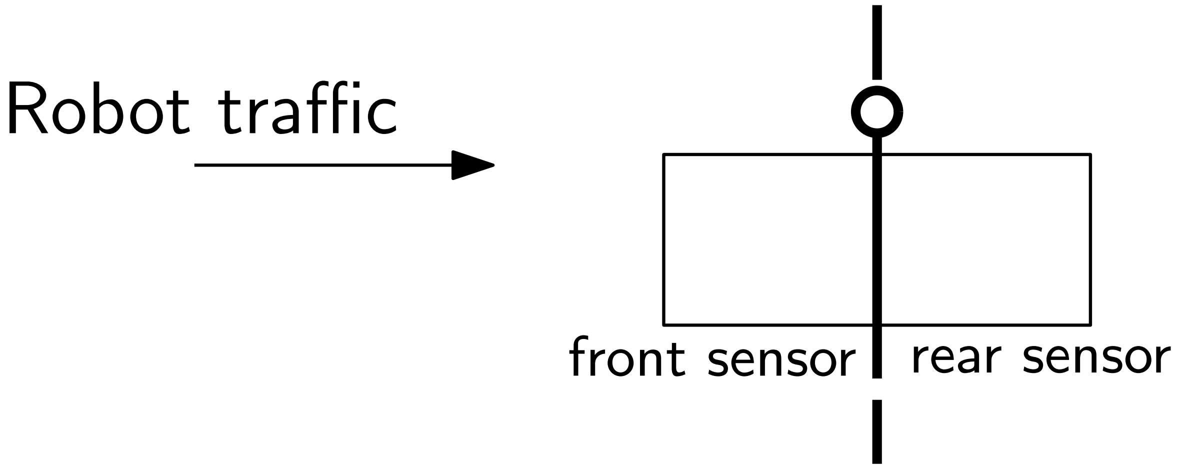
Example: an automatic door controller.



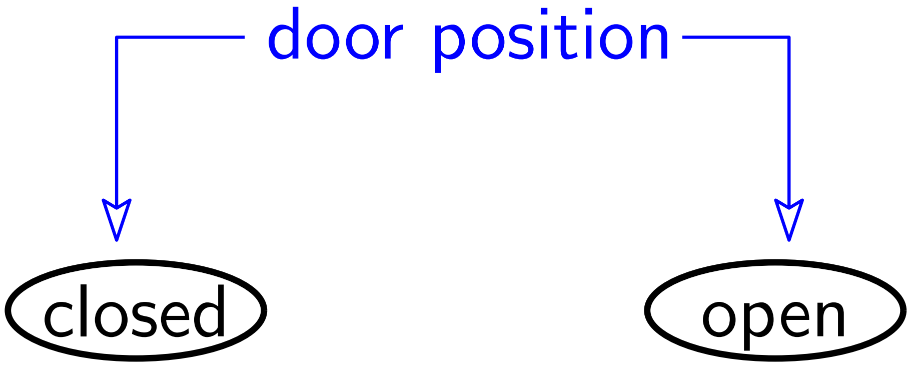
Let's encode our intuition with a transition diagram.

closed
open

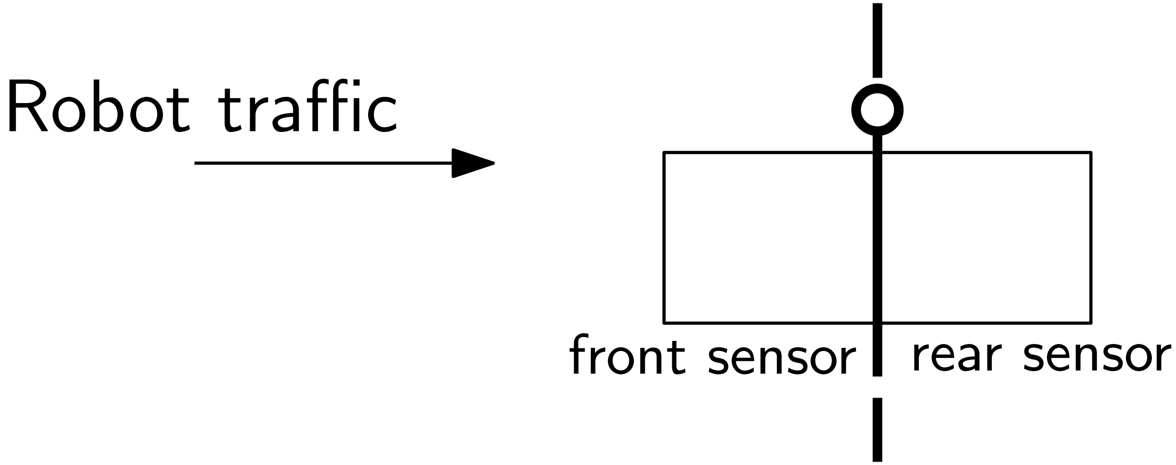
Example: an automatic door controller.



Let's encode our intuition with a transition diagram.



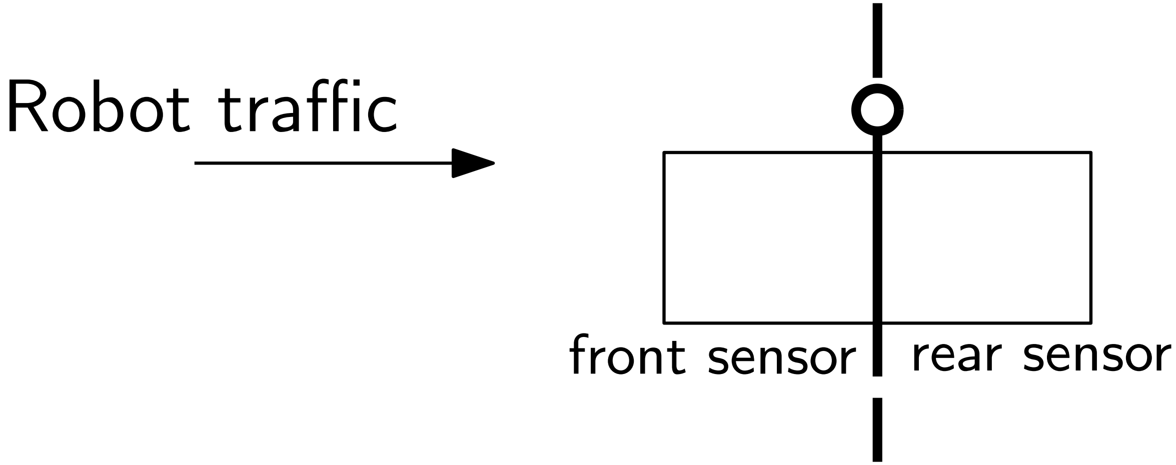
Example: an automatic door controller.



Let's encode our intuition with a transition diagram.

closed
open

Example: an automatic door controller.

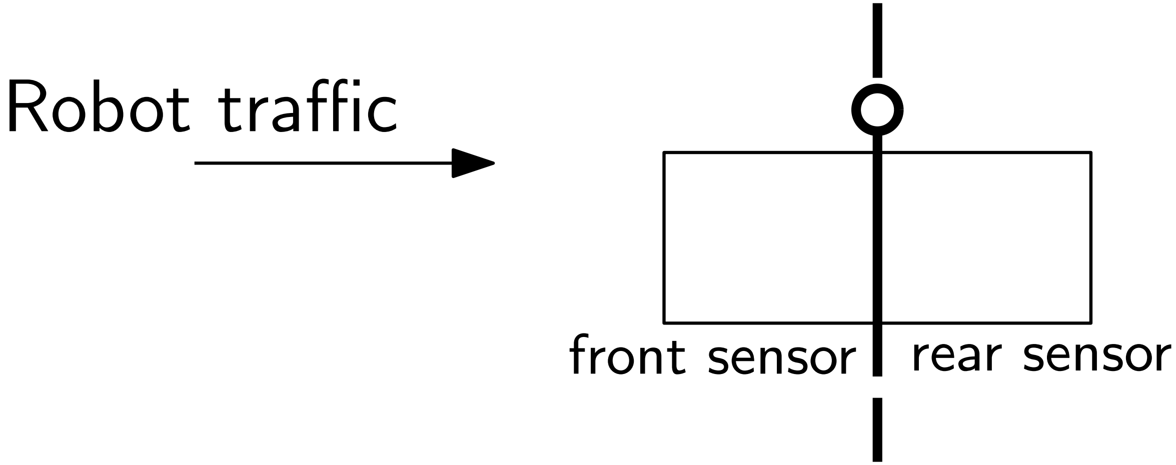


Let's encode our intuition with a transition diagram.

$f \equiv$ front sensor pad active
$r \equiv$ rear sensor pad active

closed
open

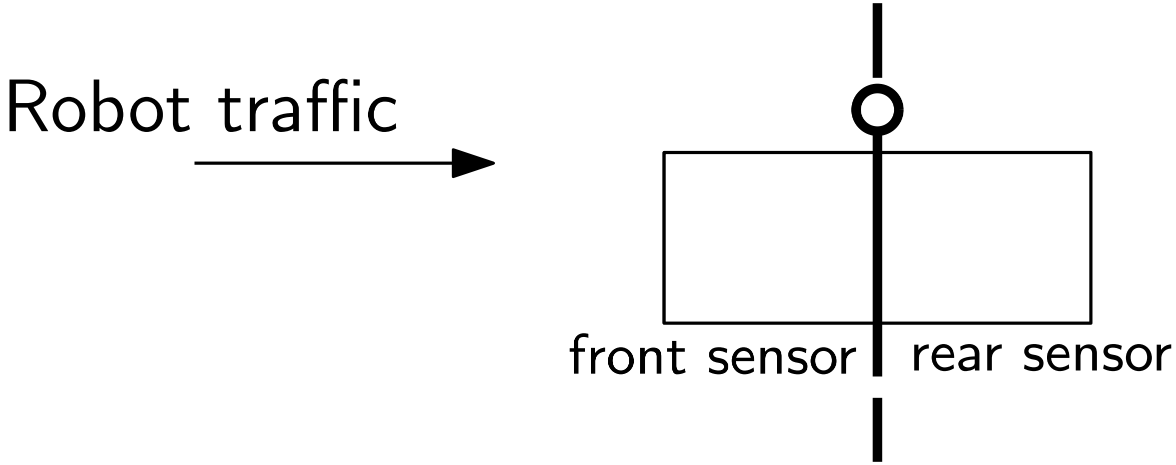
Example: an automatic door controller.



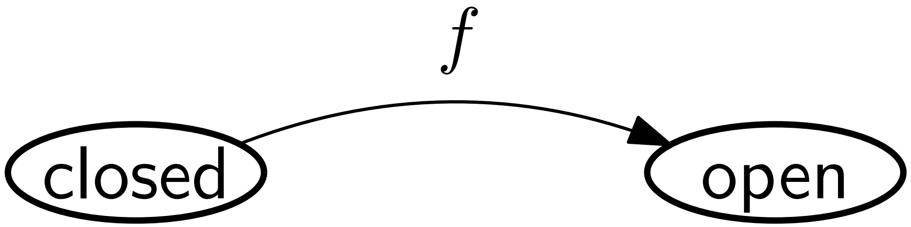
Let's encode our intuition with a transition diagram.

closed
open

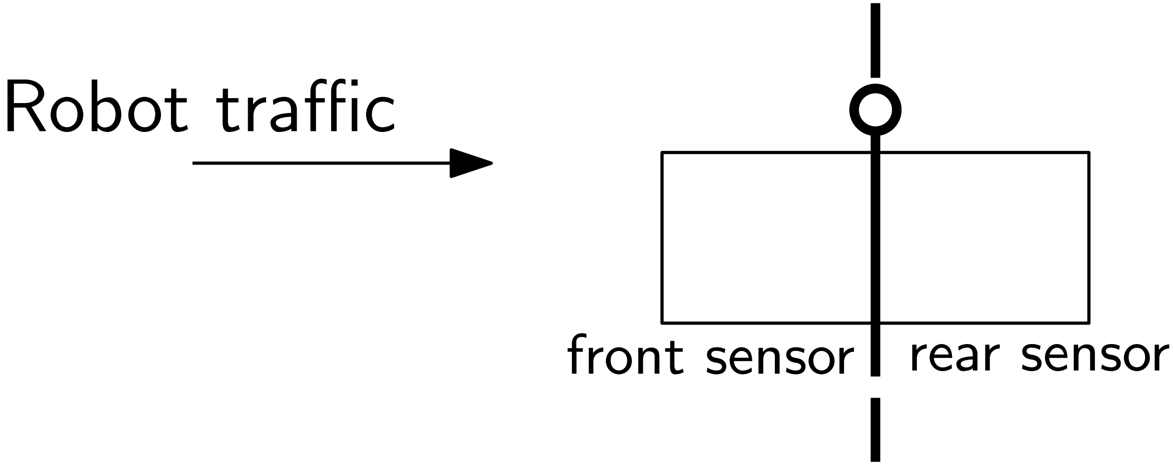
Example: an automatic door controller.



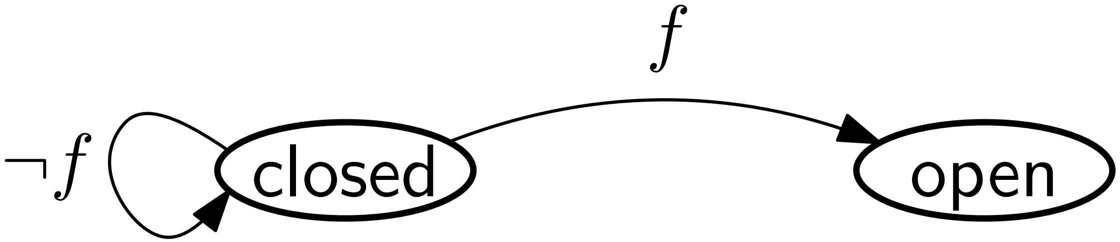
Let's encode our intuition with a transition diagram.



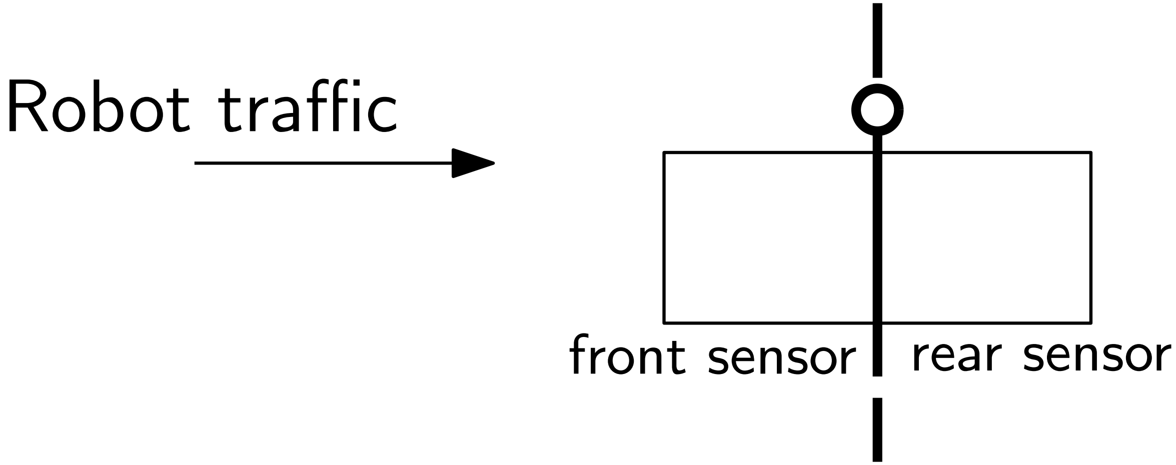
Example: an automatic door controller.



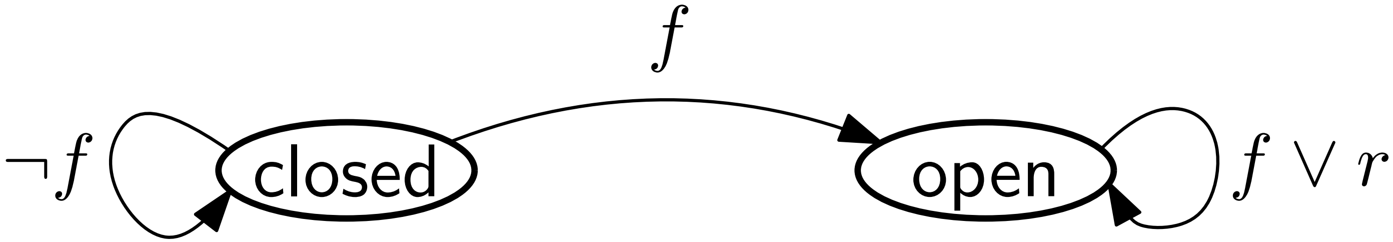
Let's encode our intuition with a transition diagram.



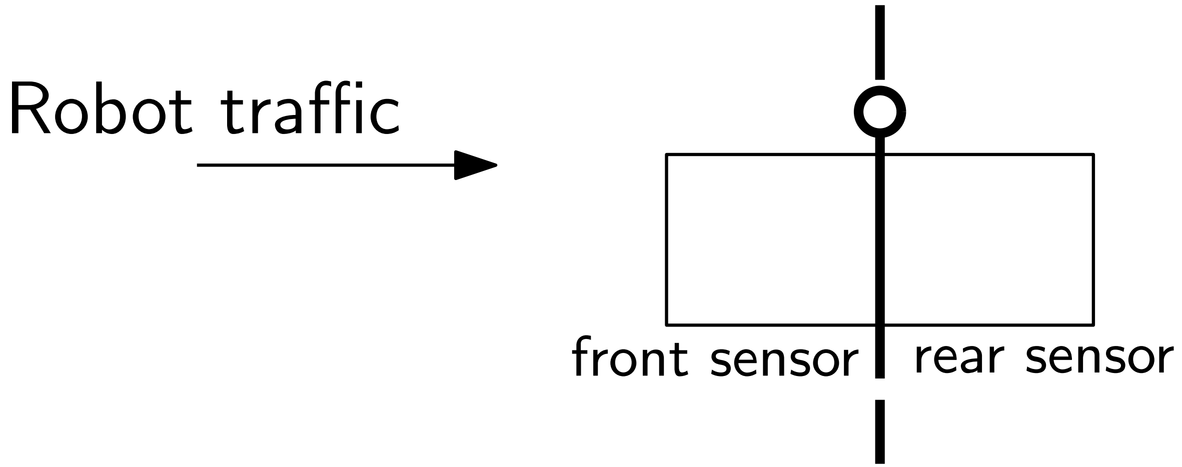
Example: an automatic door controller.



Let's encode our intuition with a transition diagram.



Example: an automatic door controller.

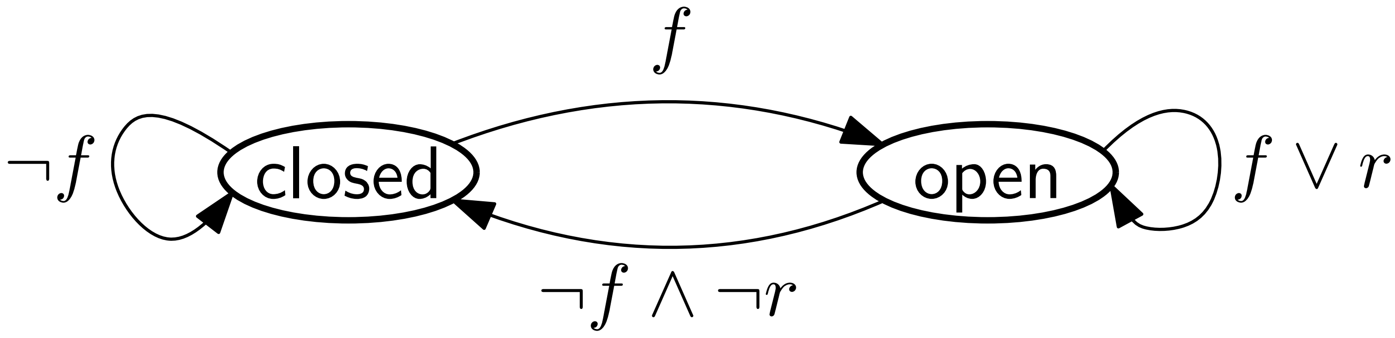


Robot traffic

front sensor

rear sensor

Let's encode our intuition with a transition diagram.



f

$\neg f$

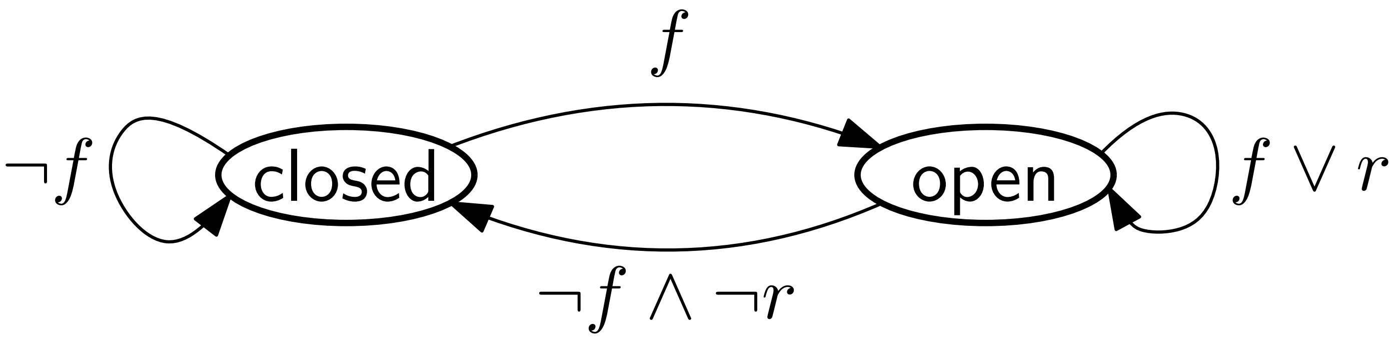
closed

open

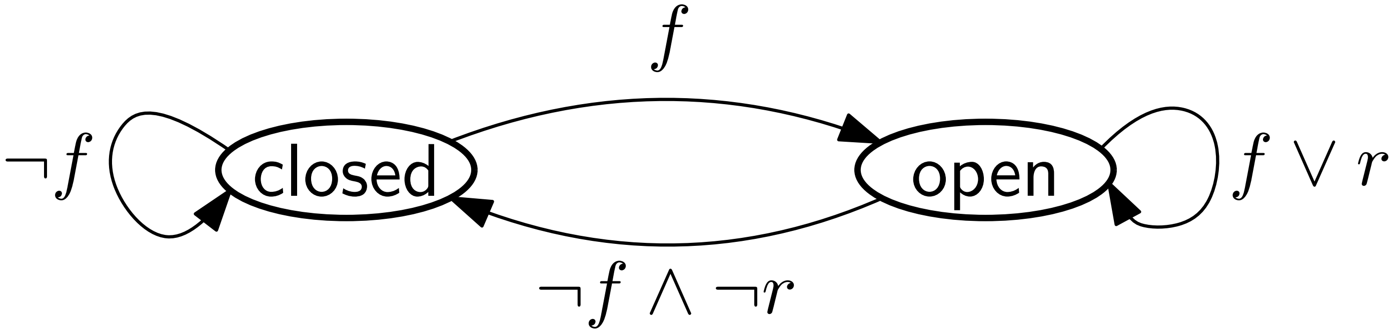
$f \vee r$

$\neg f \wedge \neg r$

Example: an automatic door controller.



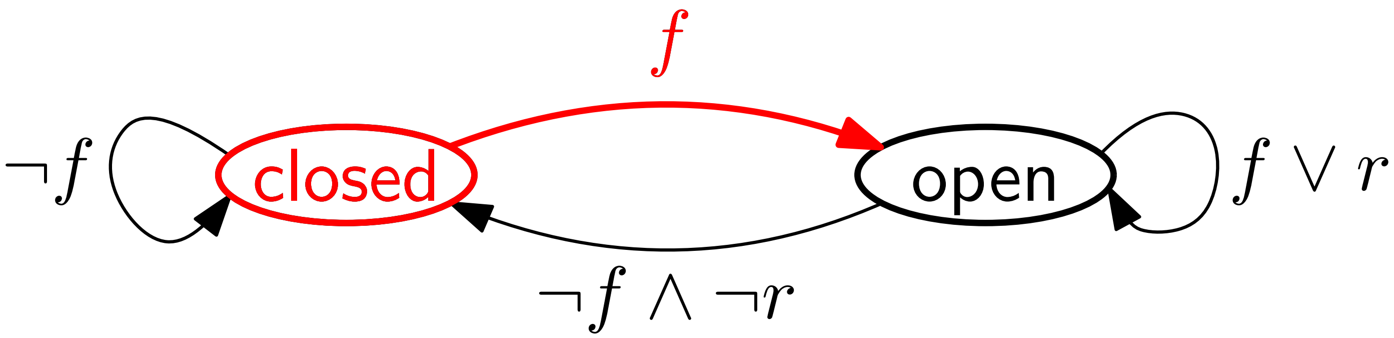
Example: an automatic door controller.



We can encode the same information in a transition table using propositional logic.

condition	next(door)

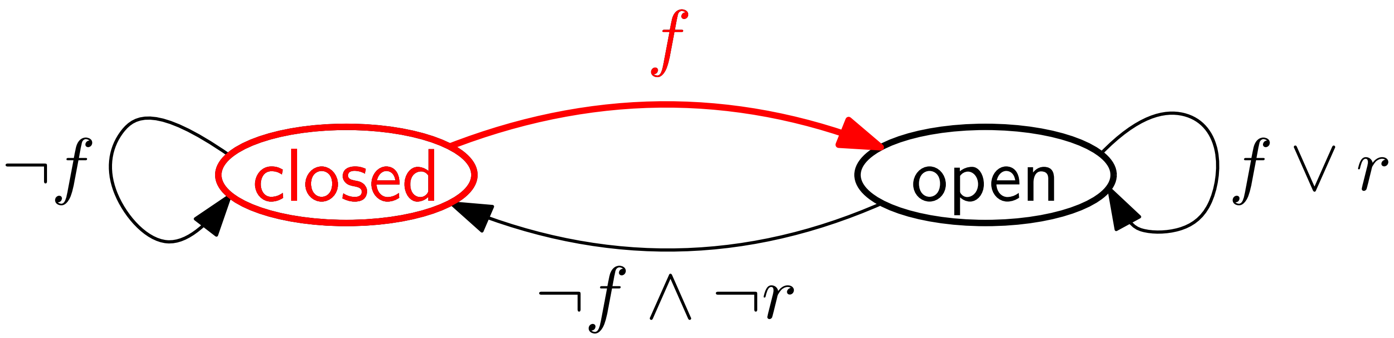
Example: an automatic door controller.



We can encode the same information in a transition table using propositional logic.

condition	next(door)
$(door = closed) \wedge f$	$open$

Example: an automatic door controller.



We can encode the same information in a transition table using propositional logic.

condition	next(door)
$(door = closed) \wedge f$	$open$
$(door = closed) \wedge \neg f$	$closed$
$(door = open) \wedge (f \vee r)$	$open$
$(door = open) \wedge (\neg f \wedge \neg r)$	$closed$

Example: an automatic door controller.

We can further encode this in NuSMV (Symbolic Model Verifier).

Example: an automatic door controller.

We can further encode this in NuSMV (Symbolic Model Verifier).

```
MODULE main

VAR
  door    : {open, closed};
  front   : boolean;
  rear    : boolean;

ASSIGN
  init(door)  := closed;
  init(front) := FALSE;
  init(rear)  := FALSE;

  next(door) :=
    case
      (door = closed) & front            : open;
      (door = closed) & ! front          : closed;
      (door = open)   & (front | rear)   : open;
      (door = open)   & ! front & ! rear : closed;
    esac;
```

Example: an automatic door controller.

We can further encode this in NuSMV (Symbolic Model Verifier).

```
MODULE main

VAR
  door     : {open, closed};
  front    : boolean;
  rear     : boolean;

ASSIGN
  init(door)   := closed;
  init(front)  := FALSE;
  init(rear)   := FALSE;

  next(door) :=
    case
      (door = closed) & front              : open;
      (door = closed) & ! front            : closed;
      (door = open)   & (front | rear)     : open;
      (door = open)   & ! front & ! rear   : closed;
    esac;
```

Variable declarations

Example: an automatic door controller.

We can further encode this in NuSMV (Symbolic Model Verifier).

```
MODULE main

VAR
   door      : {open, closed};
   front     : boolean;
   rear      : boolean;

ASSIGN
   init(door)    := closed;
   init(front)   := FALSE;
   init(rear)    := FALSE;

   next(door) :=
     case
       (door = closed) & front              : open;
       (door = closed) & ! front            : closed;
       (door = open)   & (front | rear)     : open;
       (door = open)   & ! front & ! rear   : closed;
     esac;
```

Initialization

Example: an automatic door controller.

We can further encode this in NuSMV (Symbolic Model Verifier).

```
MODULE main

VAR
  door   : {open, closed};
  front  : boolean;
  rear   : boolean;

ASSIGN
  init(door)  := closed;
  init(front) := FALSE;
  init(rear)  := FALSE;

  next(door) :=
    case
      (door = closed) & front             : open;
      (door = closed) & ! front           : closed;
      (door = open)   & (front | rear)    : open;
      (door = open)   & ! front & ! rear  : closed;
    esac;
```

Transition relation
(compare with table, previous slide)

Example: an automatic door controller.

We can further encode this in NuSMV (Symbolic Model Verifier).

```
MODULE main

VAR
  door    : {open, closed};
  front   : boolean;
  rear    : boolean;

ASSIGN
  init(door)   := closed;
  init(front)  := FALSE;
  init(rear)   := FALSE;

  next(door) :=
    case
      (door = closed) & front             : open;
      (door = closed) & ! front           : closed;
      (door = open)   & (front | rear)    : open;
      (door = open)   & ! front & ! rear  : closed;
    esac;
```

condition	next(door)
$(door = closed) \wedge f$	$open$
$(door = closed) \wedge \neg f$	$closed$
$(door = open) \wedge (f \vee r)$	$open$
$(door = open) \wedge (\neg f \wedge \neg r)$	$closed$

Transition relation
(compare with table, previous slide)

Example: an automatic door controller.

We can further encode this in NuSMV (Symbolic Model Verifier).

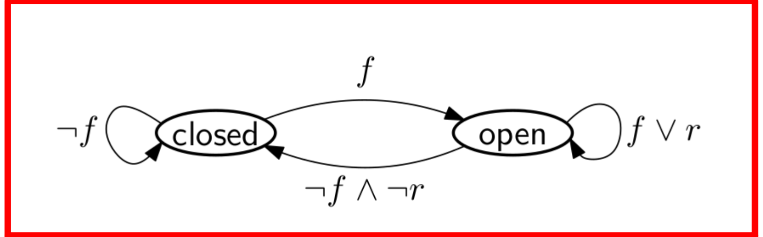
```
MODULE main

VAR
  door     : {open, closed};
  front    : boolean;
  rear     : boolean;

ASSIGN
  init(door)   := closed;
  init(front)  := FALSE;
  init(rear)   := FALSE;

  next(door) :=
    case
      (door = closed) & front             : open;
      (door = closed) & ! front           : closed;
      (door = open)   & (front | rear)    : open;
      (door = open)   & ! front & ! rear  : closed;
    esac;
```

Review: The Modeling Process

Natural language description

If the robot activates front sensor then the door . . .

Unambiguous model expressed in math and logic



condition	next(door)
$(door = closed) \wedge f$	$open$
$(door = closed) \wedge \neg f$	$closed$
$(door = open) \wedge (f \vee r)$	$open$
$(door = open) \wedge (\neg f \wedge \neg r)$	$closed$

A program that we can use to run or simulate our model

```
MODULE main

VAR
  door     : {open, closed};
  front    : boolean;
  rear     : boolean;

ASSIGN
  init(door)  := closed;
  init(front) := FALSE;
  init(rear)  := FALSE;

  next(door) :=
    case
      (door = closed) & front          : open;
      (door = closed) & ! front        : closed;
      (door = open)   & (front | rear) : open;
      (door = open)   & ! front & ! rear  : closed;
    esac;
```

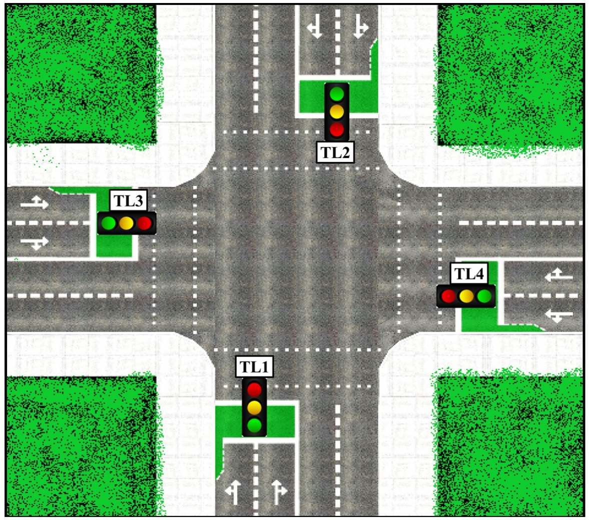
What logical properties might we care about?

One important type of property

Liveness: eventually something “good” happens.

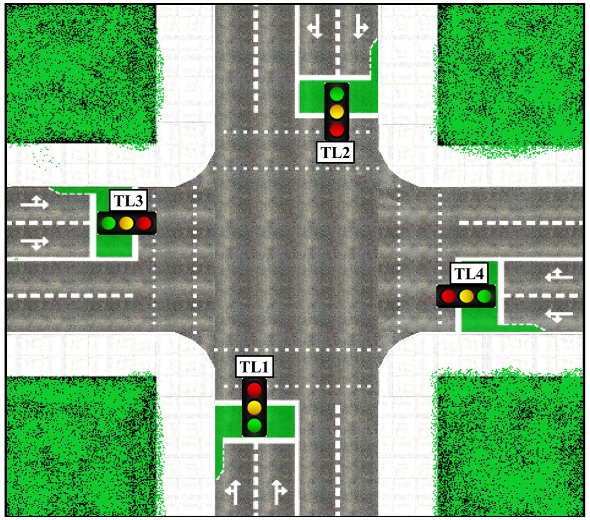
One important type of property

Liveness: eventually something “good” happens.



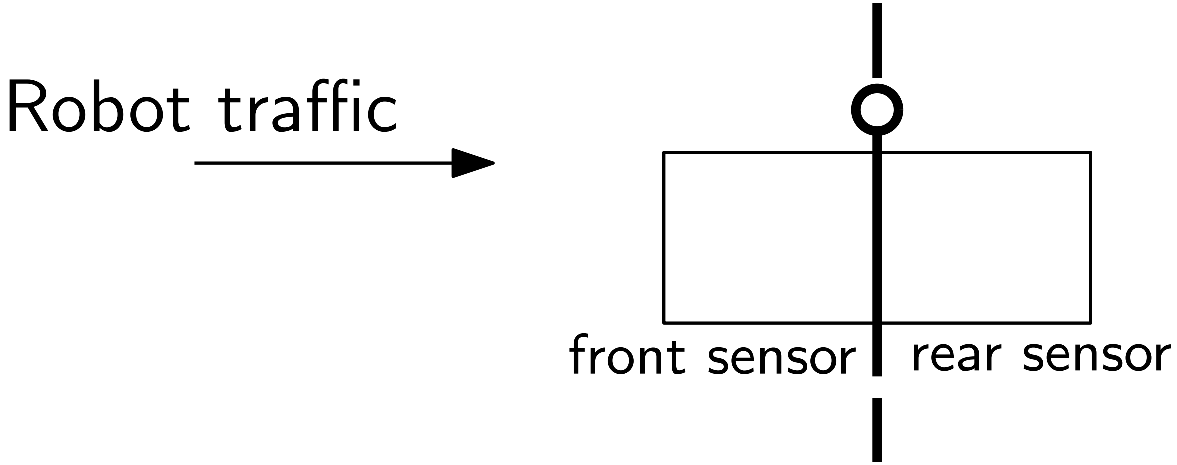
One important type of property

Liveness: eventually something "good" happens.

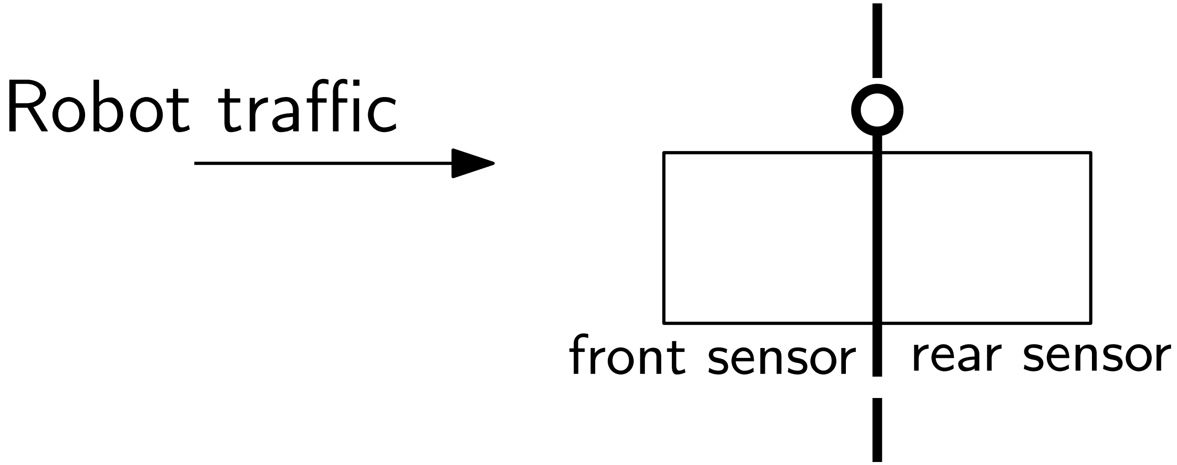


Liveness for traffic lights:
eventually one of the lights is green.

Example: an automatic door controller.



Example: an automatic door controller.



A Liveness Requirement: It is always the case that if the front pad is activated then eventually the door will be open.

Example: an automatic door controller.

A Liveness Requirement:

It is always the case that if the front pad is activated then

eventually the door will be open.

Example: an automatic door controller.

A Liveness Requirement:

It is always the case that if the front pad is activated then

eventually the door will be open.

$$G(front \to F(door = open))$$

Linear Temporal Logic

Example: an automatic door controller.

A Liveness Requirement:

It is always the case that if the front pad is activated then

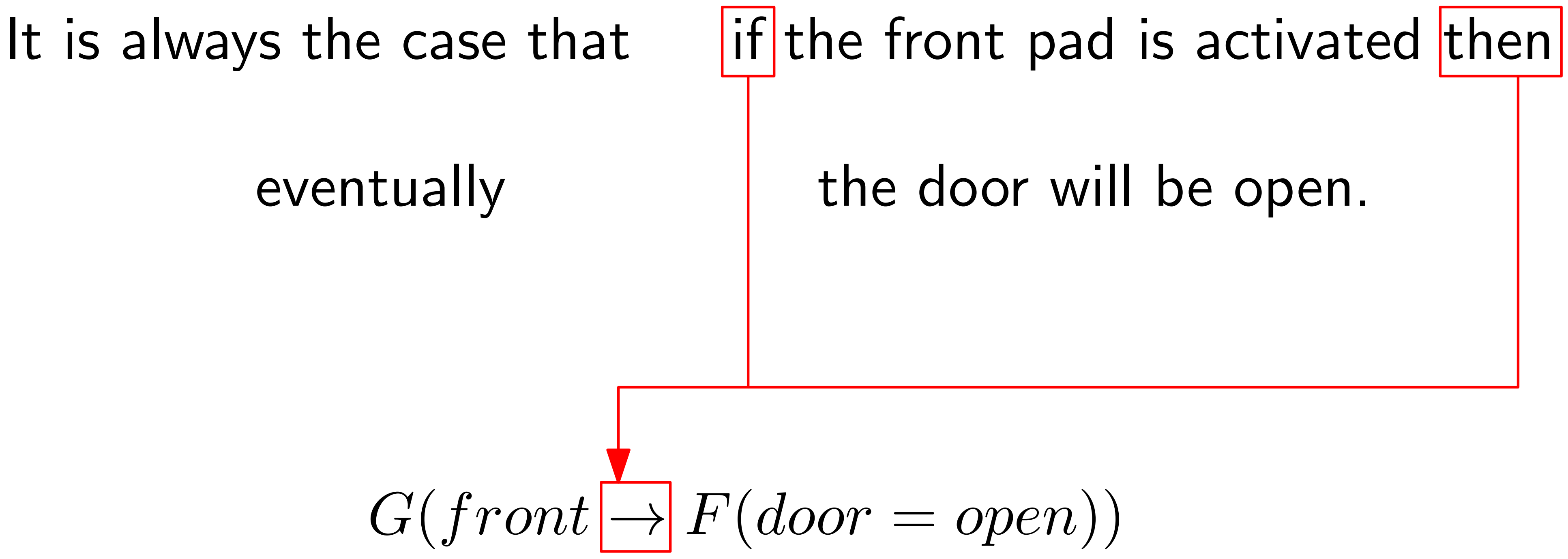
eventually the door will be open.

$$\boxed{G}(front \rightarrow F(door = open))$$

Linear Temporal Logic

Example: an automatic door controller.

A Liveness Requirement:



It is always the case that if the front pad is activated then

eventually the door will be open.

$$G(front \boxed{\to} F(door = open))$$

Linear Temporal Logic

Example: an automatic door controller.

A Liveness Requirement:

It is always the case that if the front pad is activated then

eventually the door will be open.

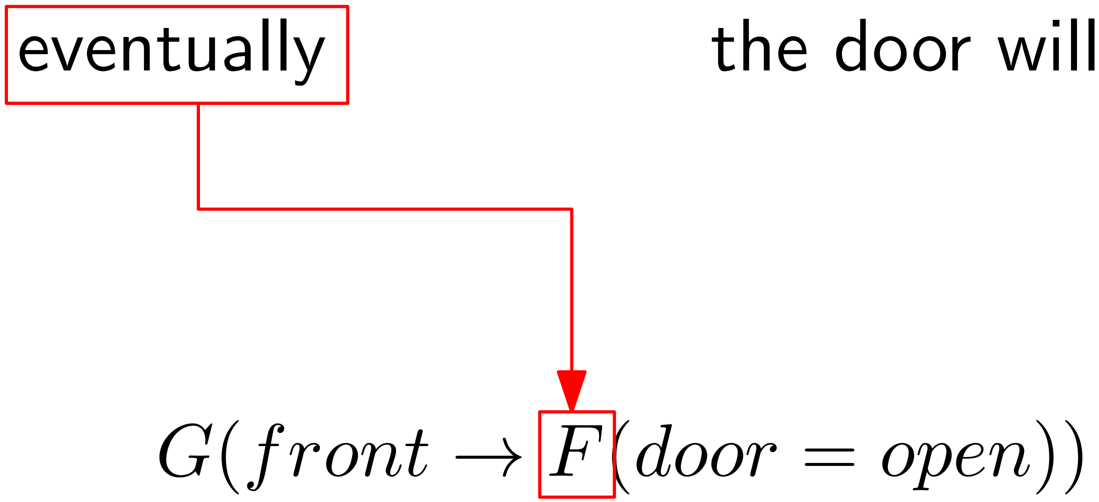
$$G(\boxed{front} \rightarrow F(door = open))$$

Linear Temporal Logic

Example: an automatic door controller.

A Liveness Requirement:

It is always the case that if the front pad is activated then



eventually the door will be open.

$$G(front \rightarrow \boxed{F}(door = open))$$

Linear Temporal Logic

Example: an automatic door controller.

A Liveness Requirement:

It is always the case that if the front pad is activated then

eventually the door will be open.

$$G(front \rightarrow F(\boxed{door = open}))$$

Linear Temporal Logic

Example: an automatic door controller.

A Liveness Requirement:

It is always the case that if the front pad is activated then

eventually the door will be open.

$$G(front \to F(door = open))$$

Linear Temporal Logic

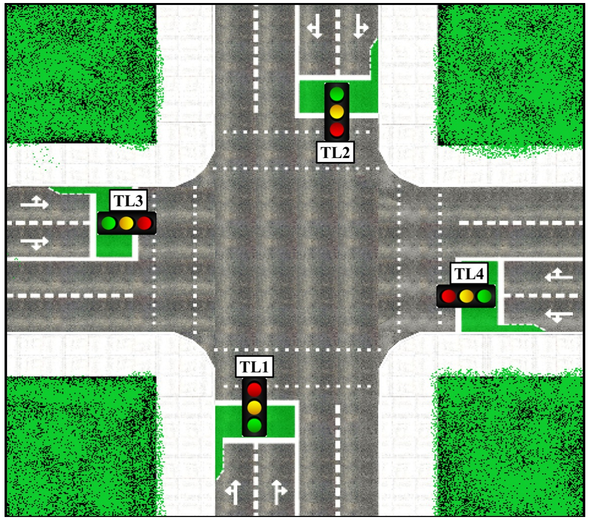
We can check this property with NuSMV!

Another important type of property

Safety: a bad thing never happens.

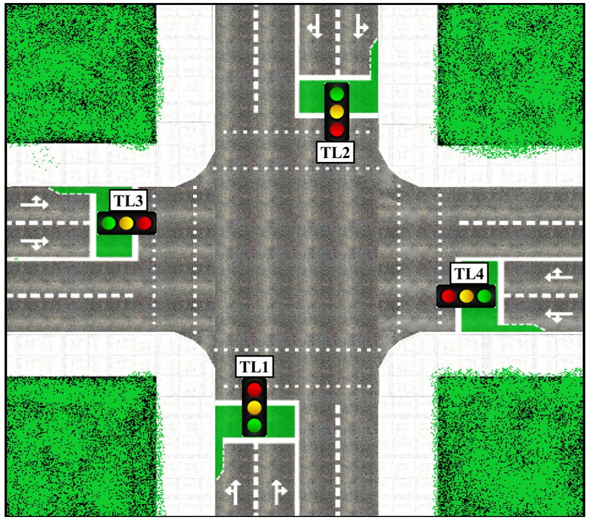
Another important type of property

Safety: a bad thing never happens.



Another important type of property

Safety: a bad thing never happens.



Safety: Any two perpendicular lanes never have corresponding lights that are green at the same time.

Example: an automatic door controller.

A Safety Requirement: It is always the case that if the rear pad is active and the door is closed, then in the next state if the rear pad is still active then the door remains closed.

Example: an automatic door controller.

A Safety Requirement: It is always the case that if the rear pad is active and the door is closed, then in the next state if the rear pad is still active then the door remains closed.

$$G(rear \wedge door = closed \rightarrow X(rear \rightarrow door = closed))$$

Example: an automatic door controller.

A Safety Requirement: It is always the case that if the rear pad is active and the door is closed, then in the next state if the rear pad is still active then the door remains closed.

$$\boxed{G}(rear \wedge door = closed \rightarrow X(rear \rightarrow door = closed))$$

Example: an automatic door controller.

A Safety Requirement: It is always the case that if the rear pad is active and the door is closed, then in the next state if the rear pad is still active then the door remains closed.

$$G(rear \wedge door = closed \boxed{\rightarrow} X(rear \rightarrow door = closed))$$

Example: an automatic door controller.

A Safety Requirement: It is always the case that if the rear pad is active and the door is closed, then in the next state if the rear pad is still active then the door remains closed.

$$G(\boxed{rear \wedge door = closed} \rightarrow X(rear \rightarrow door = closed))$$

Example: an automatic door controller.

A Safety Requirement: It is always the case that if the rear pad is active and the door is closed, then in the next state if the rear pad is still active then the door remains closed.

$$G(rear \wedge door = closed \rightarrow \boxed{X}(rear \rightarrow door = closed))$$

Example: an automatic door controller.

A Safety Requirement: It is always the case that if the rear pad is active and the door is closed, then in the next state if the rear pad is still active then the door remains closed.

$$G(rear \wedge door = closed \rightarrow X(\boxed{rear \rightarrow door = closed}))$$

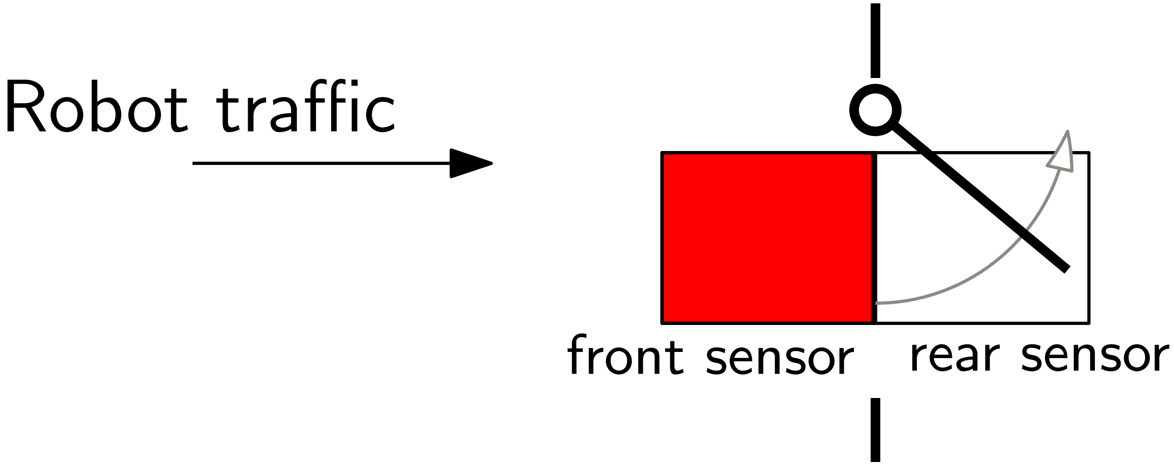
Example: an automatic door controller.

A Safety Requirement: It is always the case that if the rear pad is active and the door is closed, then in the next state if the rear pad is still active then the door remains closed.

$$G(rear \wedge door = closed \rightarrow X(rear \rightarrow door = closed))$$

Let's also check this property with NuSMV . . .

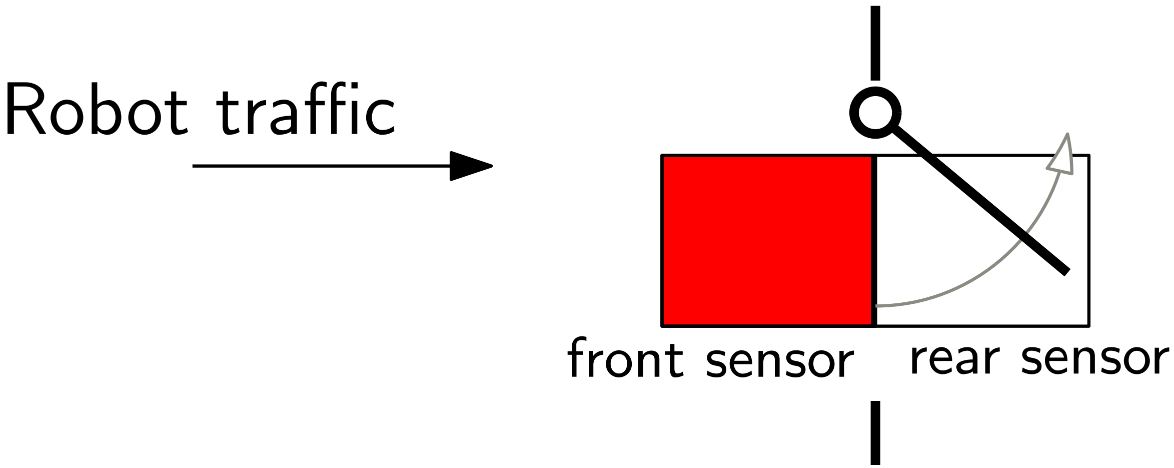
Things to consider...



We saw that our model wasn't quite right yet.

What's missing?

Things to consider...



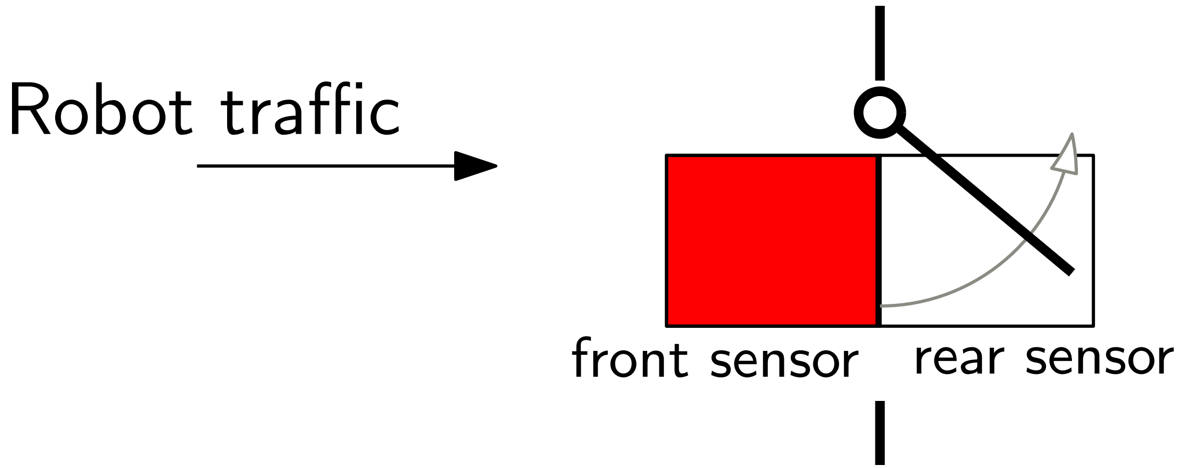
We saw that our model wasn't quite right yet.

What's missing?

Do we need to model intermediate door positions?

$$door \in \{open, opening, closed, closing\} \text{ ??}$$

Things to consider...



Robot traffic

front sensor

rear sensor

We saw that our model wasn't quite right yet.

What's missing?

Do we need to model intermediate door positions?

$$door \in \{open, opening, closed, closing\} \text{ ??}$$

Do we need to model the robot behavior and the sensor state??

$$next(robotPosition) := \ldots,\ next(frontSensor) := \ldots \text{ ??}$$

A common sentiment:

> “I thought I knew how my [program , proof , simulation, model] worked until I ran [NuSMV, Z3, SPIN, JPF, Alloy, etc.] on it!”

Learning automated resoning techniques forces you to think *very carefully* about what you are doing, and often exposes subtle misunderstandings.

First Few Weeks:

Propositional Logic

A python-based domain specific language for propostional logic, satisfiability checking, model counting, and data structures for logic (BDDs).

Middle part of the class:

Transition Systems

We will learn a formal system of specifying transition systems (which we often depict as a transition diagram).

Temporal Logic (LTL)

We will assign symbols for expressing temporal system requirements like *always*, *eventually*, *next*, *until*.

Temporal Logic Software

Symbolic Model Verifier (NuSMV)

Later Weeks:

Automated Theorem Proving

We will use Z3 to help us automatically prove things,
e.g. a python program doesn't have assertion violations

or give us counterexamples
e.g. inputs that cause an assertion violation

Finally:

Presentation about a logic or automated reasoning:

tool or software
theory or foundation
cultural, social, historical, cognitive, linguistic context

Now: Quick preview of HW00.