

# **EECS498-008**

# **Formal Verification**

# **of Systems Software**

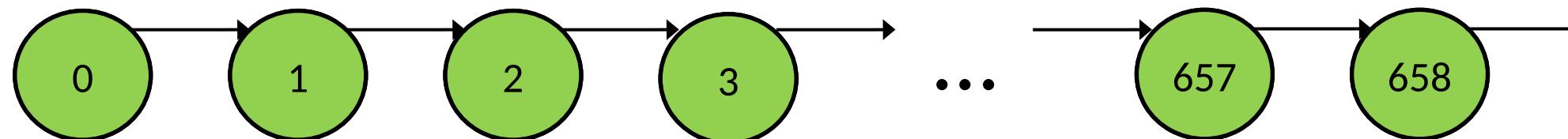
Material and slides created by  
Jon Howell and Manos Kapritsos

# Chapter 4: Proving properties

Expressing a system as a state machine allows us to **prove** that it has certain properties

- We will focus on safety properties
  - i.e. properties that hold throughout the execution

Basic tool: induction

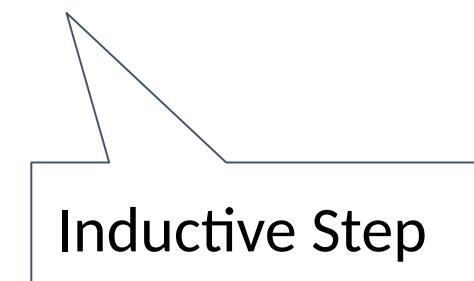


- Show that the property holds on state 0
- Show that if the property holds on state  $k$ , it must hold on state  $k+1$

# Let's prove a safety invariant!

```
predicate Safety(v:Variables) {  
    true // TBD  
}
```

```
lemma SafetyProof()  
    ensures forall v :: Init(v) ==> Safety(v)  
    ensures forall v, v' :: Safety(v) && Next(v, v') ==> Safety(v')  
{  
}
```



# Let's prove a safety invariant!

*Interactive proof development in editor*

*Bisection debugging,  
case analysis,  
existential instantiation*



# Jay Normal Form

As you begin writing more interesting specs, proofs will be nontrivial.

Pull all the nondeterminism into one place, and get a receipt.



image: flickr/afagen CC-by-nc-sa

# Jay Normal Form

```
datatype Step =
| Action1Step( <parameters> )
| Action2Step( <parameters> )
...

predicate NextStep(v: Variables, v': Variables, step:Step)
{
  match step
    case Action1Step(<parameters>) => Action1(v, v', <parameters>)
    case Action2Step(<parameters>) => Action2(v, v', <parameters>)
  ...
}

predicate Next(v: Variables, v': Variables)
{
  exists step :: NextStep(v, v', step)
}
```

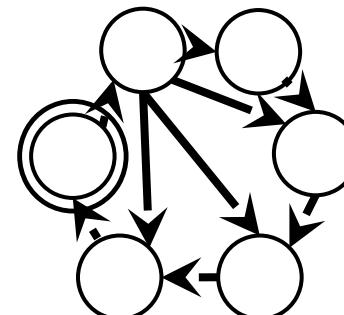
# Administrivia

- Problem Set 2 will be released today
  - Chapters 3 and 4
  - Due Friday, October 7, 11:59pm
- Problem set 3 will be split into PS3 (Chapter 5, pre-project1) and PS4 (Chapter 6, post-project1).
  - Collectively still worth 8% of the total grade
  - Chapters 7 and 8 are now part of PS5
- Midterm coming up on October 12, 6-8pm

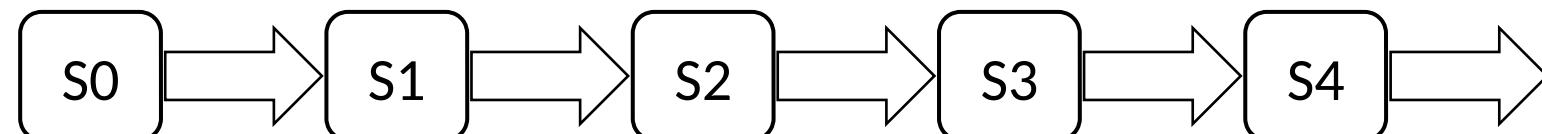
# Safety property

Safety property (a.k.a. invariant): a property that **always** holds

## State machine representation

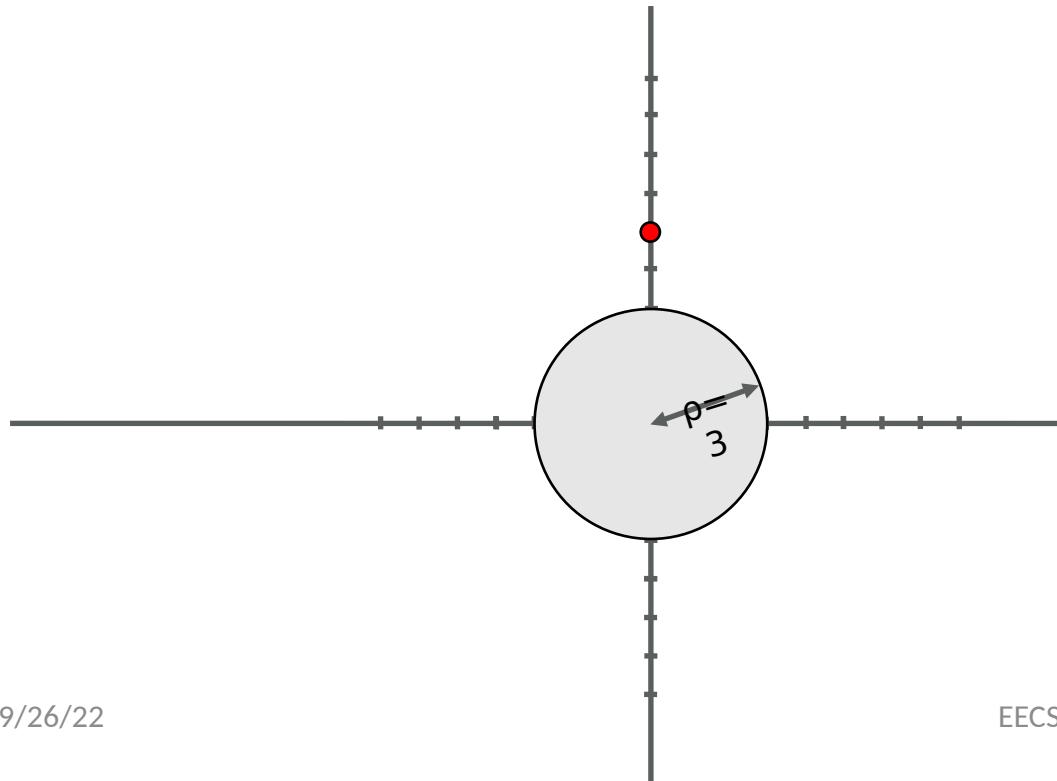


## An execution



# Example: Crawler

- Crawler starts at (0,5)
- It can move 1 step north or 1 step south-east
- Can it ever fall in the hole?

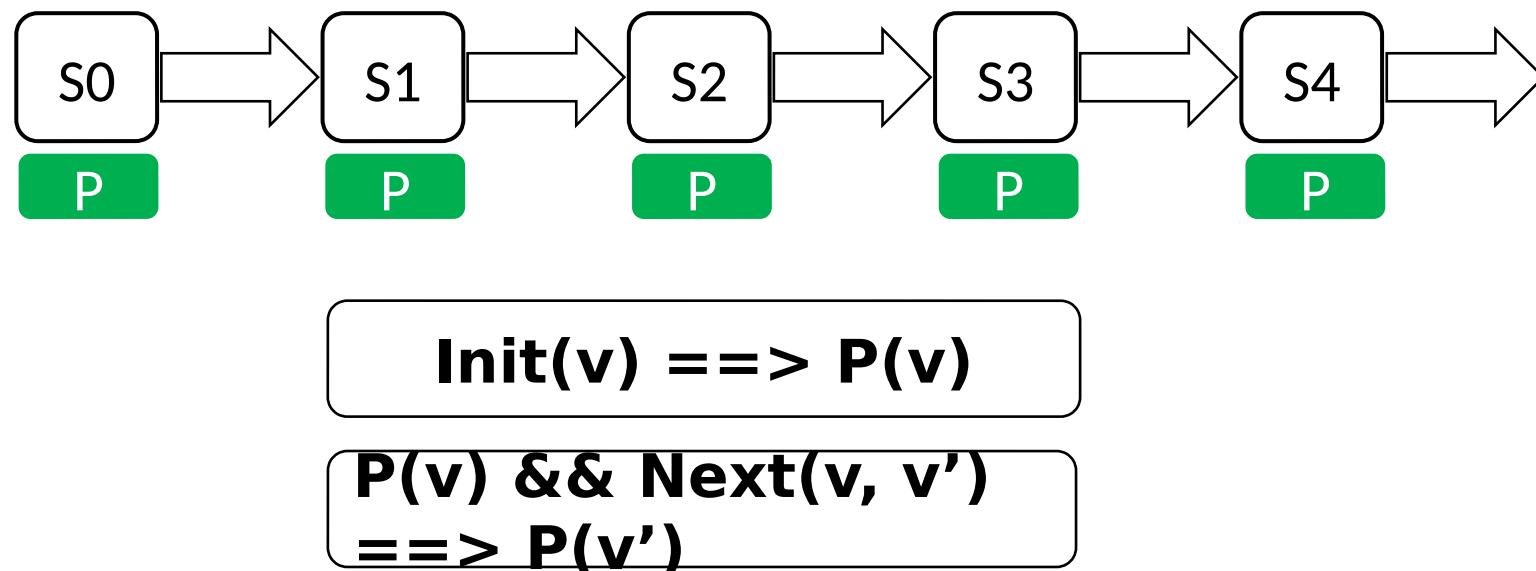


```
predicate Init(v:Variables) {  
  && v.x == 0  
  && v.y == 5  
}  
  
predicate MoveNorth(v:Variables,  
v':Variables) {  
  && v'.x == v.x  
  && v'.y == v.y + 1  
}  
  
predicate MoveSouthEast(v:Variables,  
v':Variables) {  
  && v'.x == v.x + 1  
  && v'.y == v.y - 1  
}
```

# Proving invariants

## Proof by induction

- Prove it holds on the first state
- Prove it holds during a transition



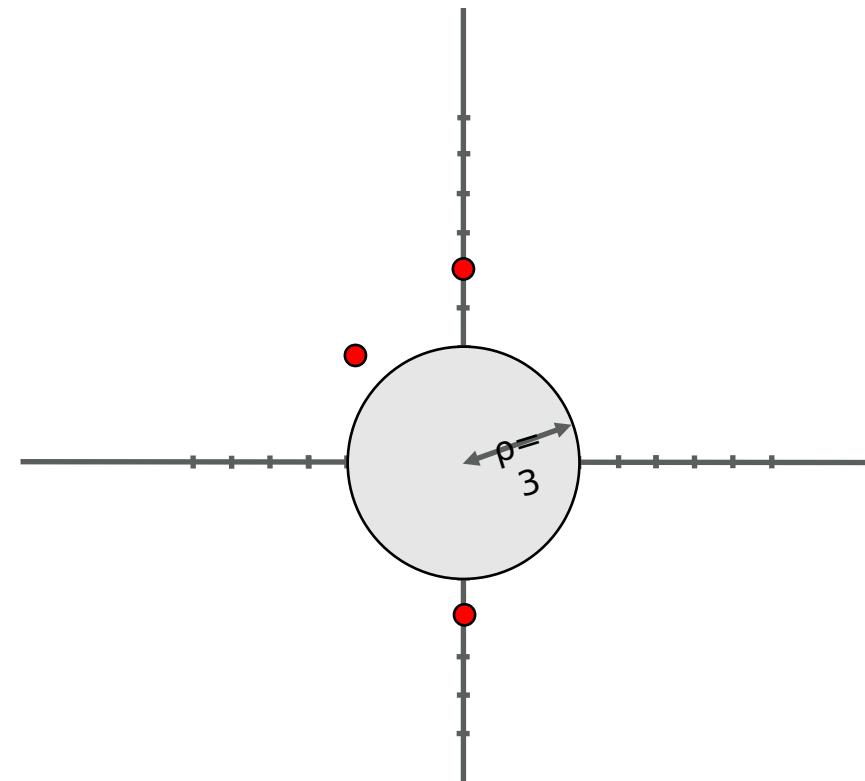
# Proving the Crawler

```
predicate Init(v:Variables) {  
    && v.x == 0  
    && v.y == 5  
}  
  
predicate MoveNorth(v:Variables,  
v':Variables) {  
    && v'.x == v.x  
    && v'.y == v.y + 1  
}  
  
predicate MoveSouthEast(v:Variables,  
v':Variables) {  
    && v'.x == v.x + 1  
    && v'.y == v.y - 1  
}  
  
predicate InHole(v:Variables) {  
    v.x*v.x + v.y*v.y <= 3*3  
}
```

Safety property:  $\neg \text{InHole}(v)$

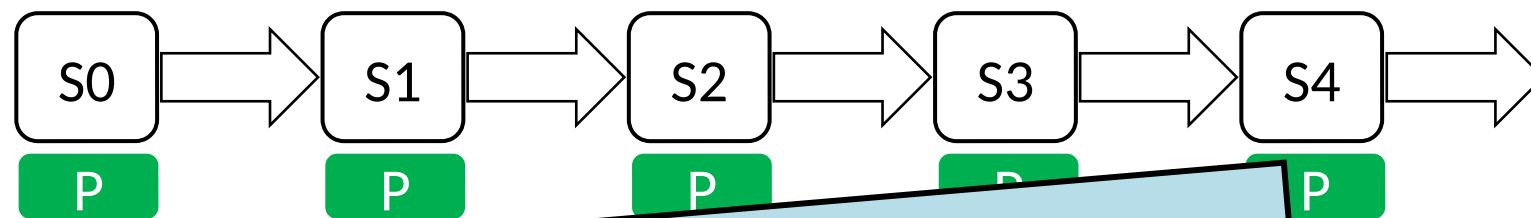
$\text{Init}(v) \implies P(v)$  ✓

$P(v) \& \& \text{Next}(v, v') \implies P(v')$  ✗



# Inductive invariants

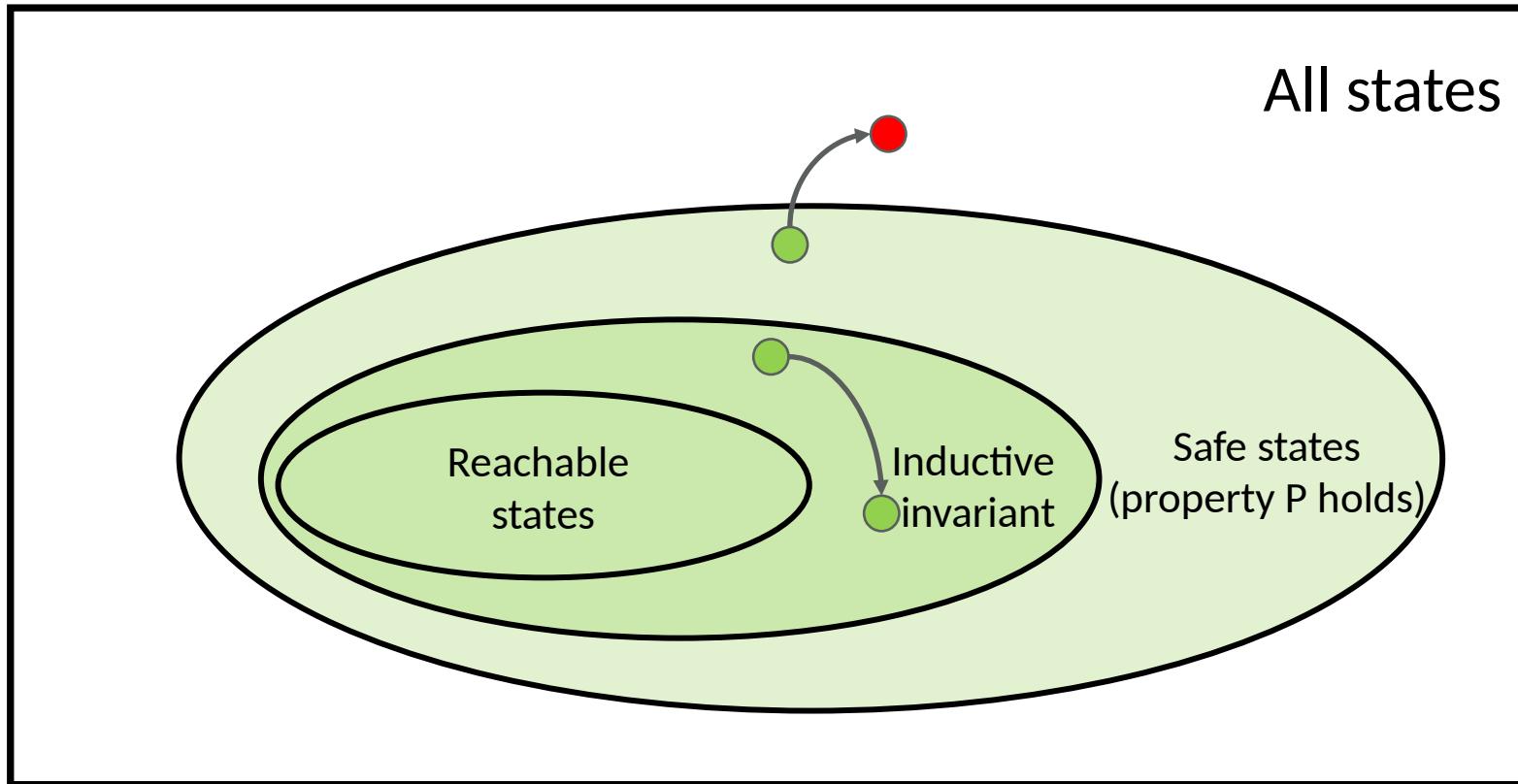
Safety property (a.k.a.  
invariant):  
a property that **always** holds



The problem:  
Property P may **not** be inductive!

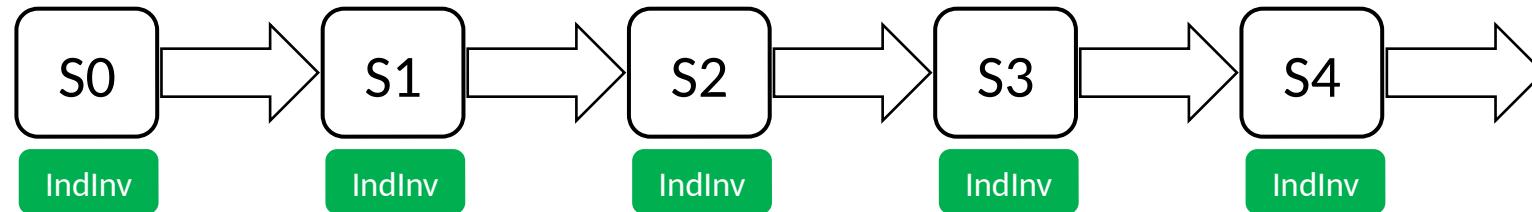
$$\begin{aligned} & \mathbf{P(v) \ \&\& \ Next(v, v')} \\ \implies & \mathbf{P(v')} \end{aligned}$$

# Invariants vs Inductive invariants



# Proving safety with Inductive invariants

**IndInv( $v$ ) ==> P( $v$ )**

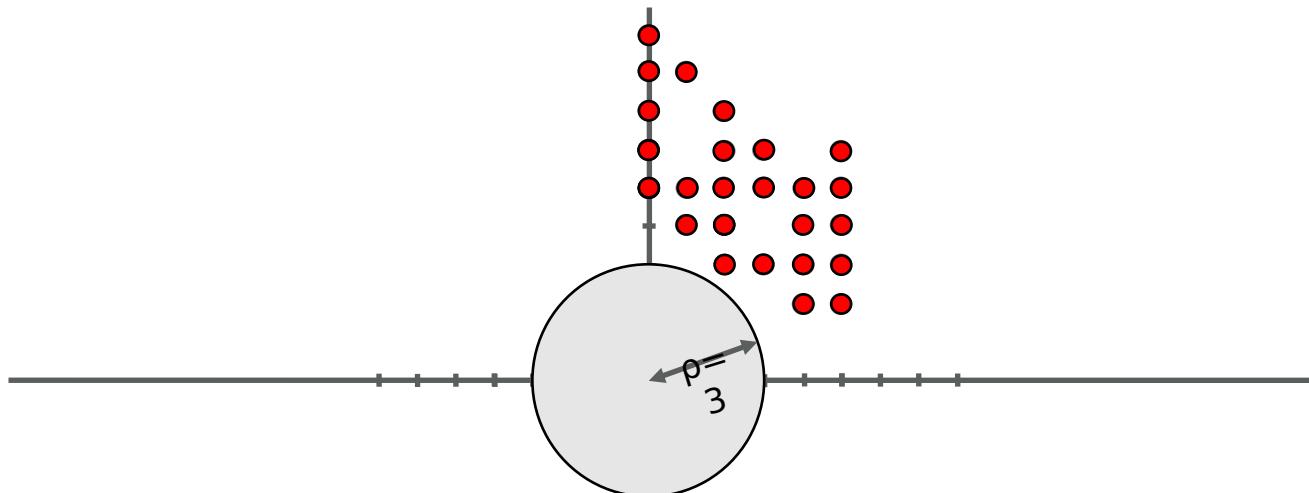


**Init( $v$ ) ==> IndInv( $v$ )**

**IndInv( $v$ ) && Next( $v, v'$ )  
==> IndInv( $v'$ )**

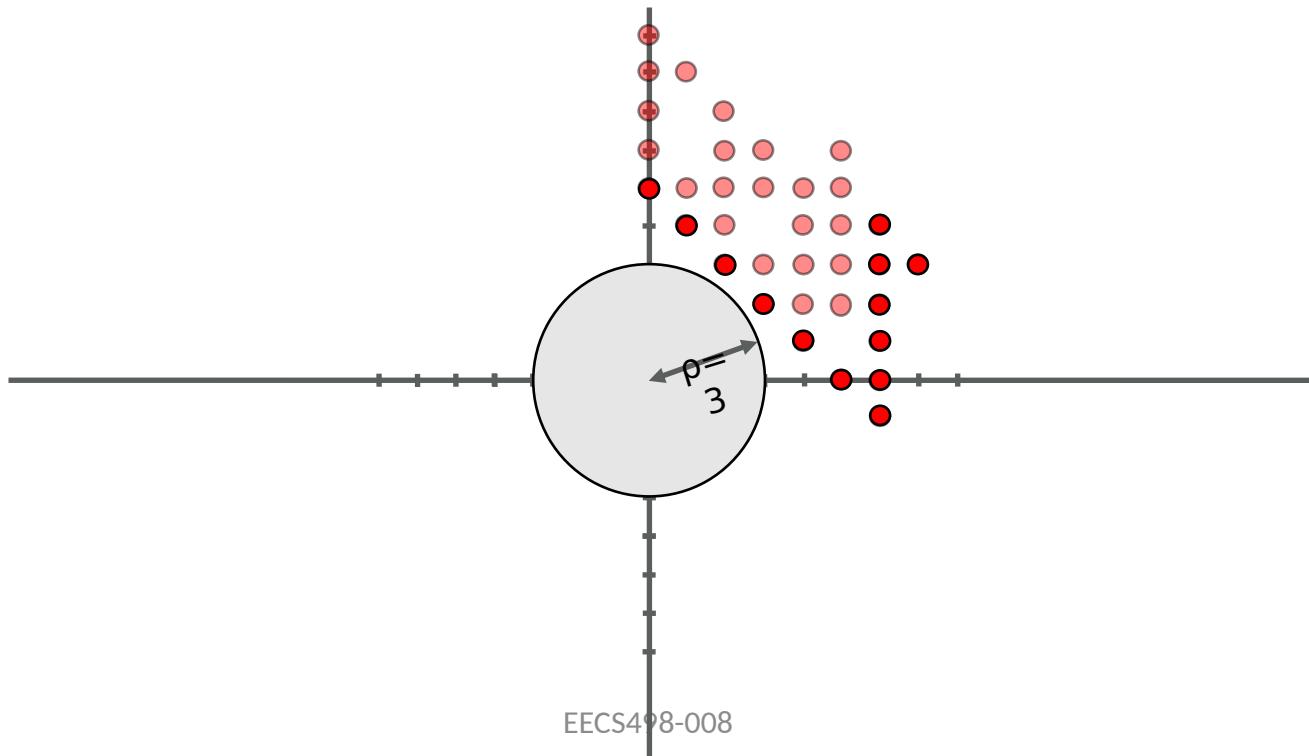
# Proving the Crawler

Can the crawler ever fall in the hole?



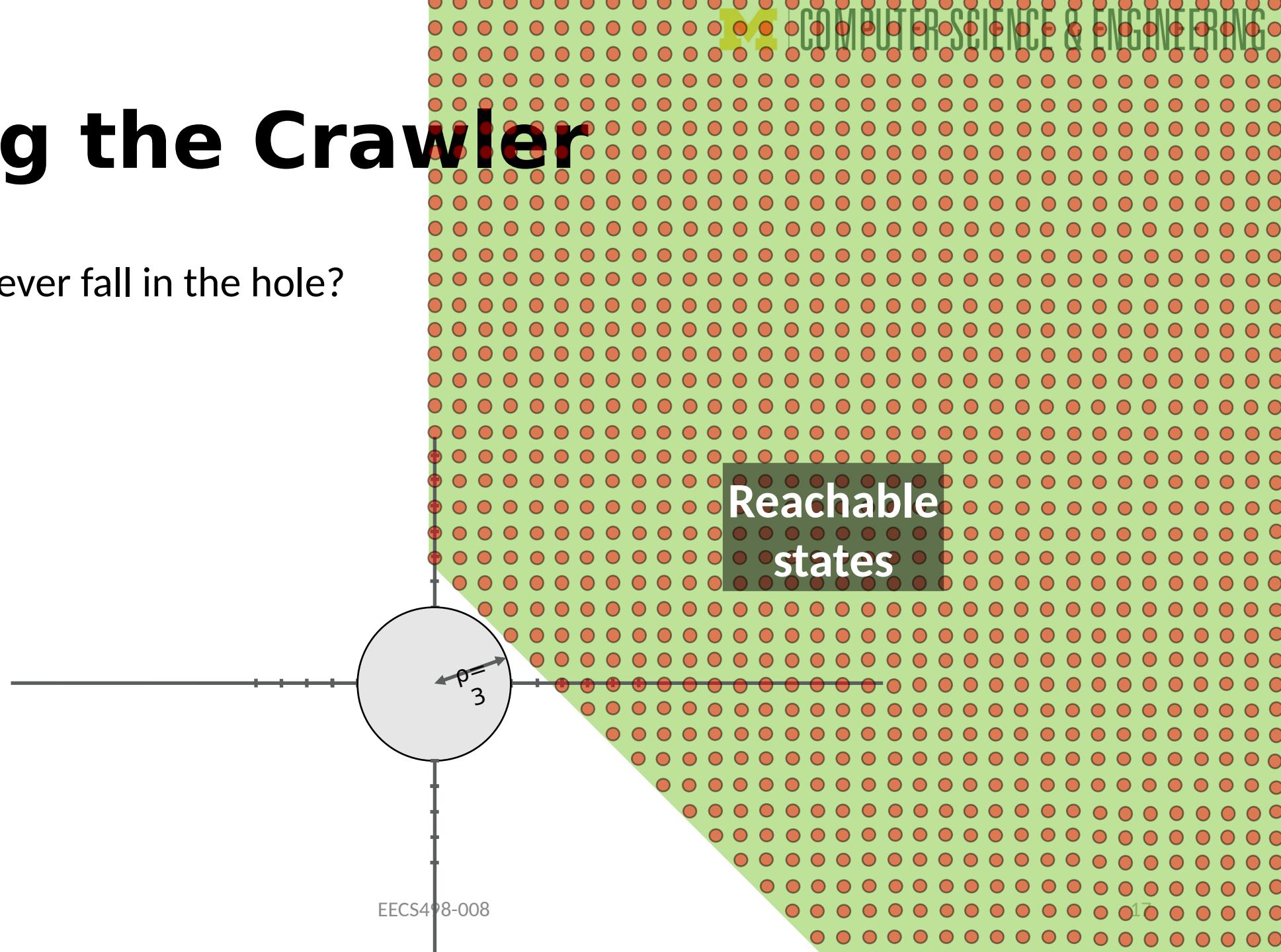
# Proving the Crawler

Can the crawler ever fall in the hole?



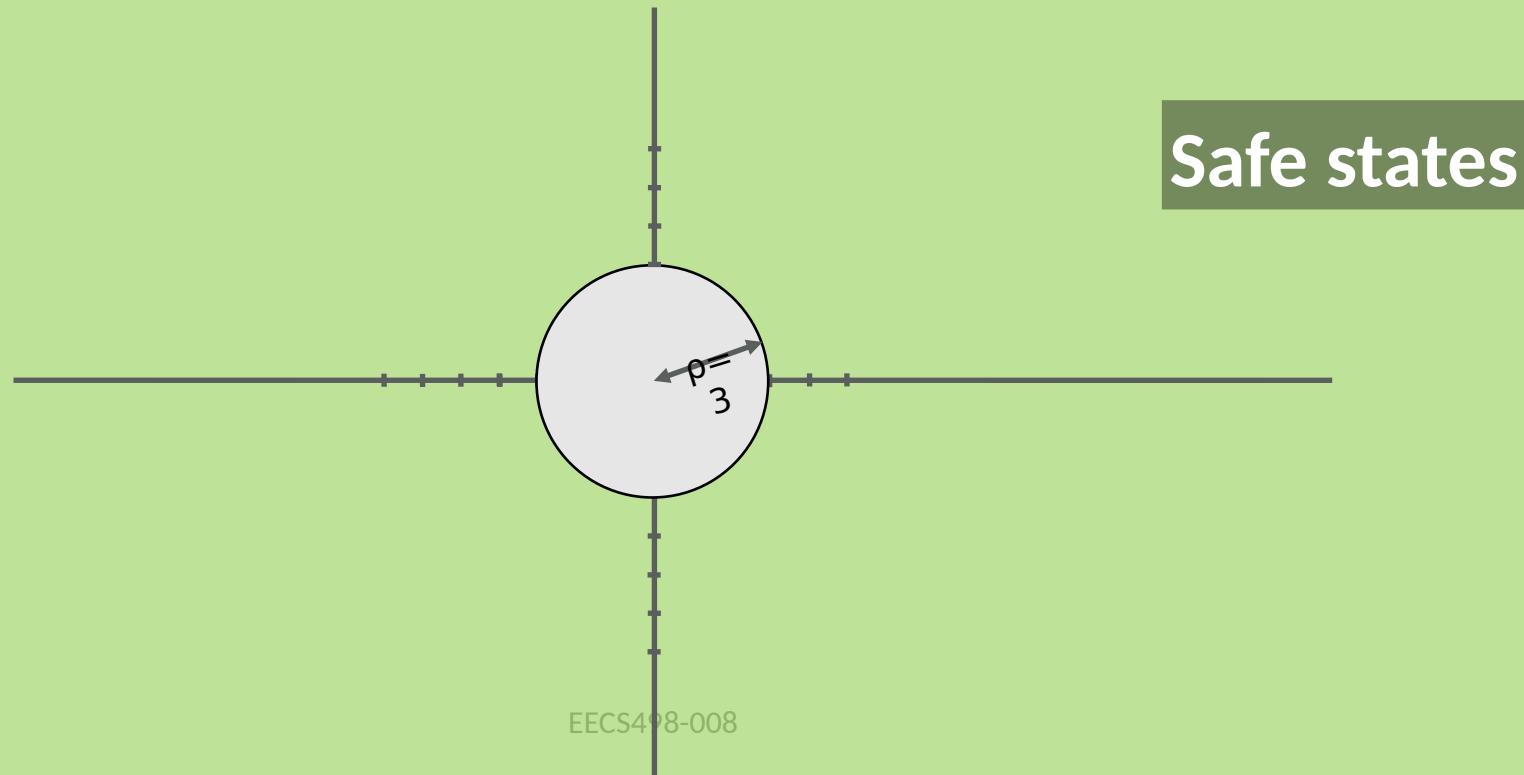
# Proving the Crawler

Can the crawler ever fall in the hole?



## Naïve safety proof

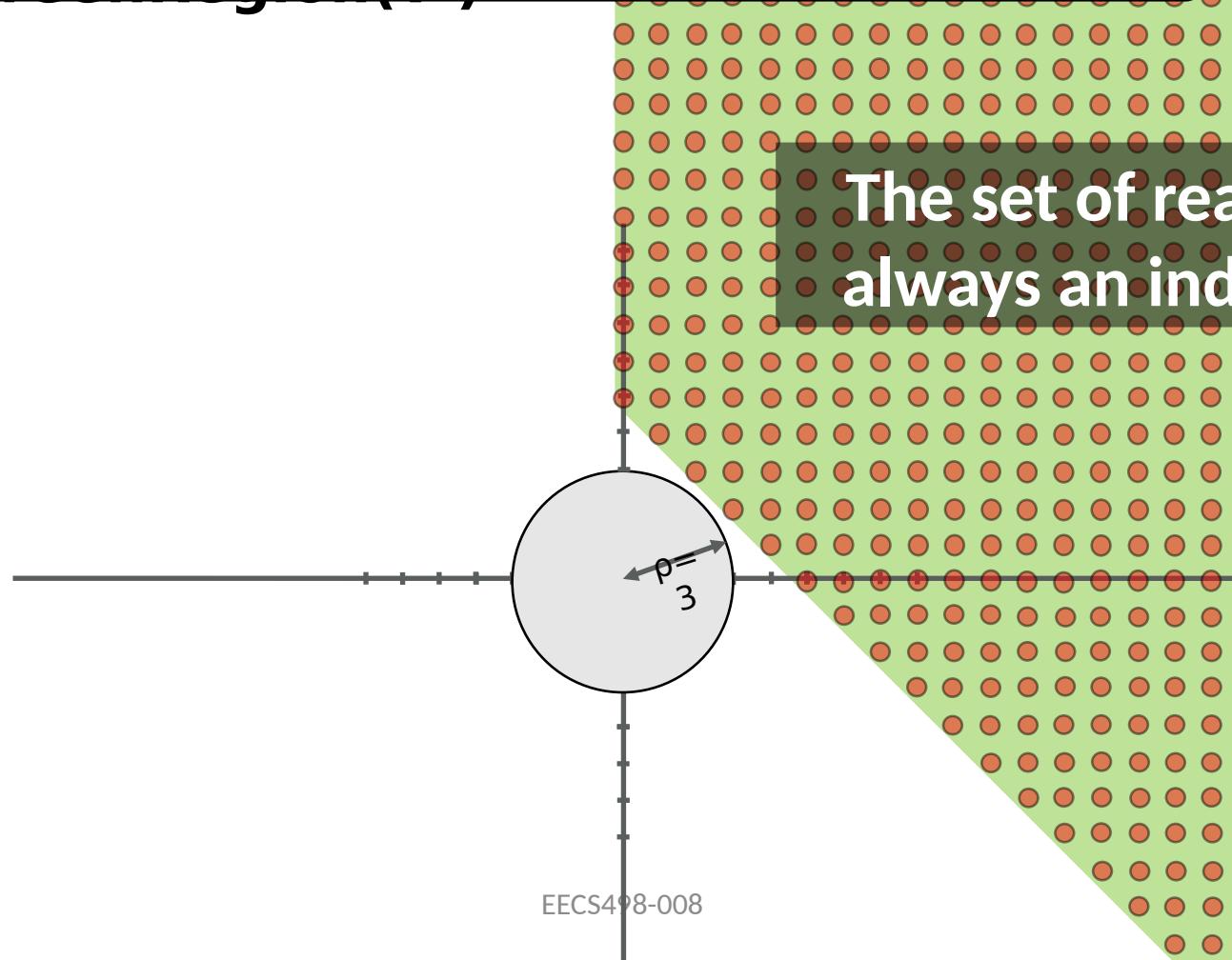
**Safe(v) && Next(v, v')  
==> Safe(v')**

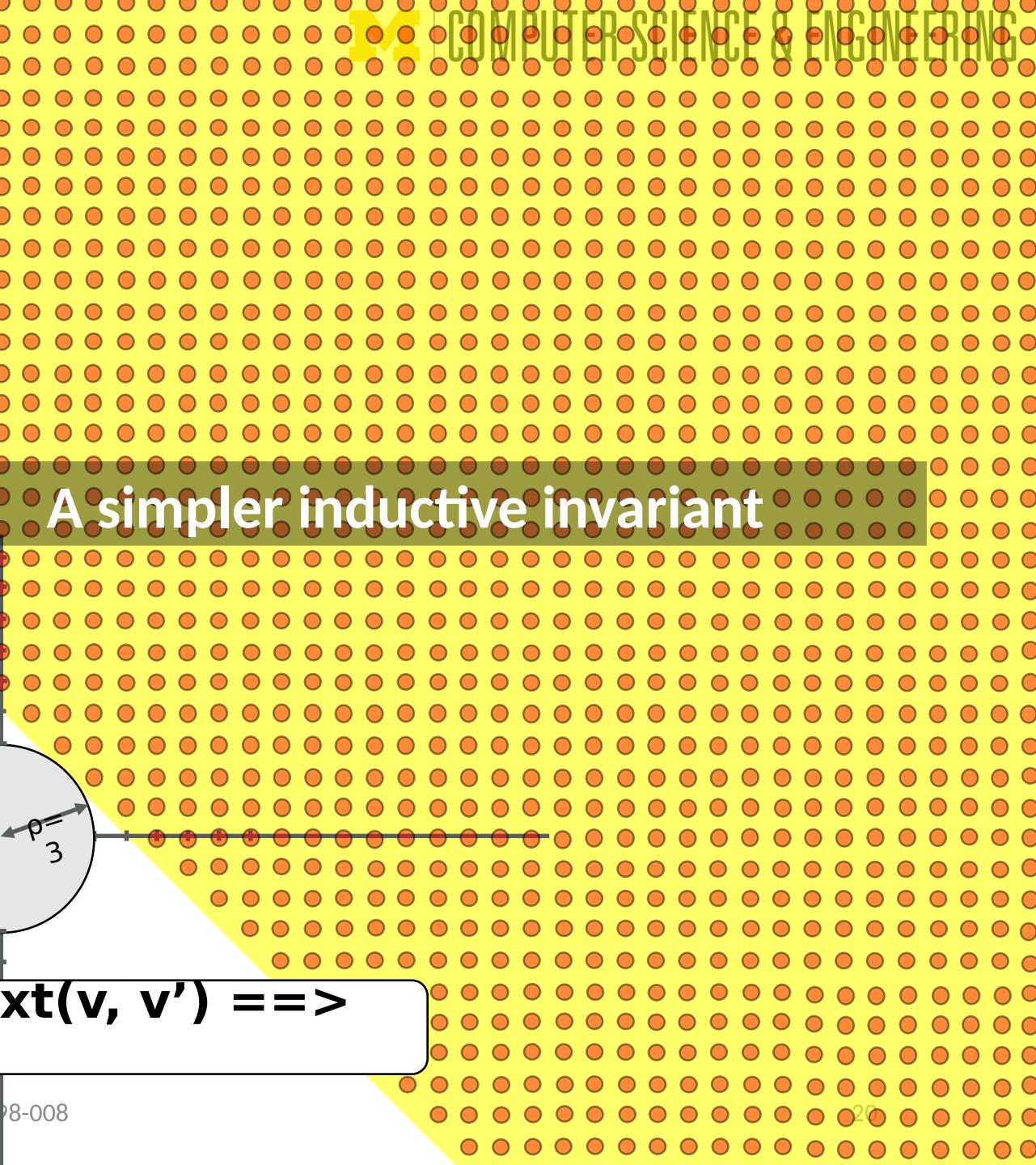


## Safety proof using a stronger invariant

**InGreenRegion(v) && Next(v, v') ==>  
InGreenRegion(v')**

The set of reachable states is  
always an inductive invariant





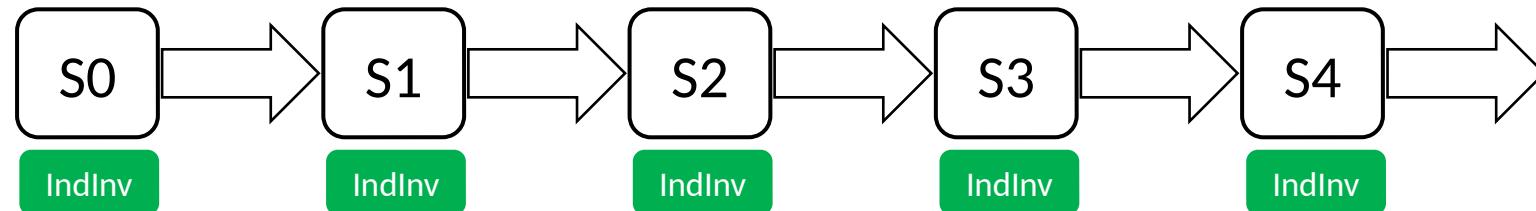
**InYellowRegion(v) && Next(v, v') ==>  
InYellowRegion(v')**

# Proving safety with Inductive invariants

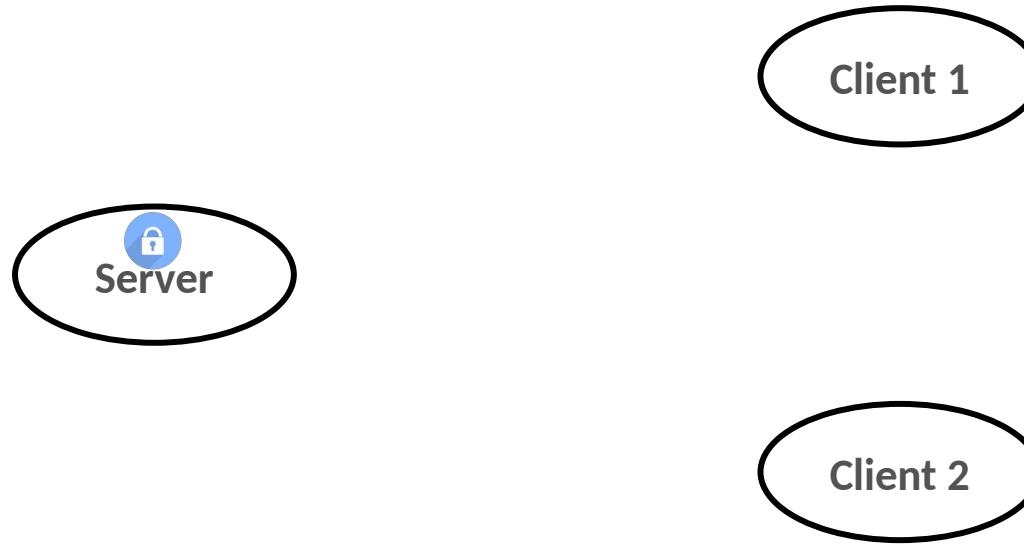
**IndInv( $v$ ) ==> Safety( $v$ )**

**Init( $v$ ) ==> IndInv( $v$ )**

**IndInv( $v$ ) && Next( $v, v'$ )  
==> IndInv( $v'$ )**



# Example: lock server



Safety property:  $\neg(C1 \wedge C2)$

Both clients cannot hold the lock at the same time

# Example: lock server

