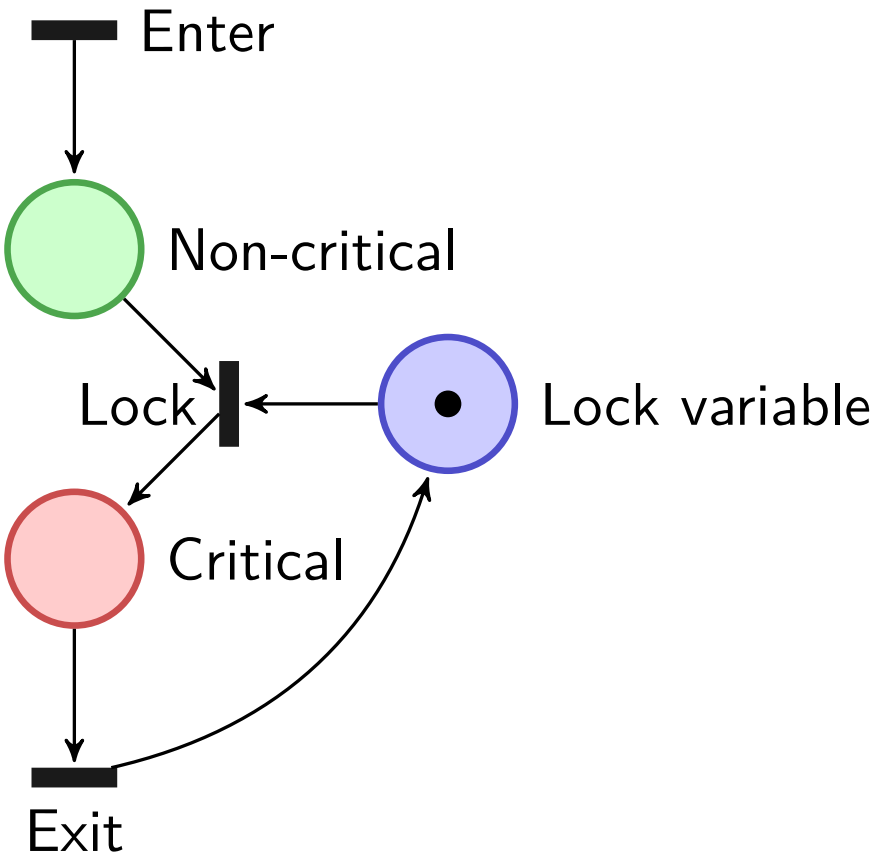
Incremental, Inductive Coverability

Johannes Kloos
Rupak Majumdar Filip Niksic Ruzica Piskac

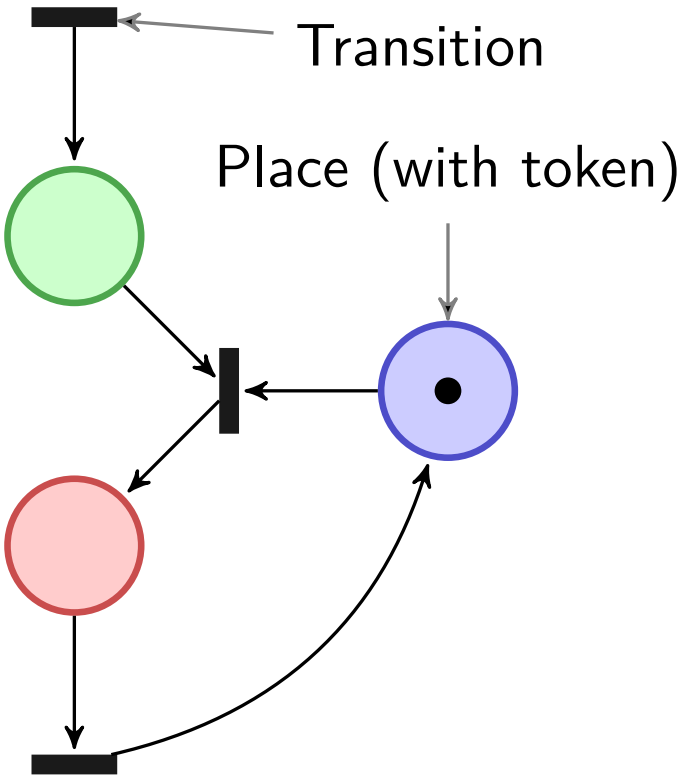
Max Planck Institute for Software Systems

July 24, 2013

Petri nets are a formalism for modelling concurrency

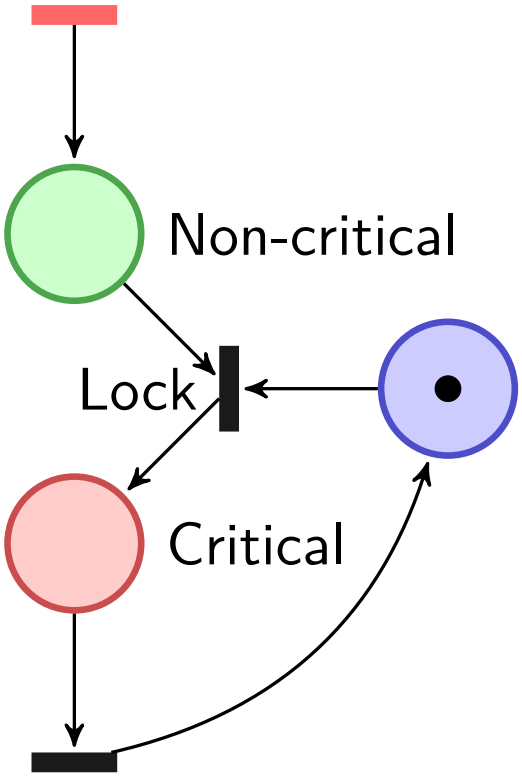


Petri nets are a formalism for modelling concurrency



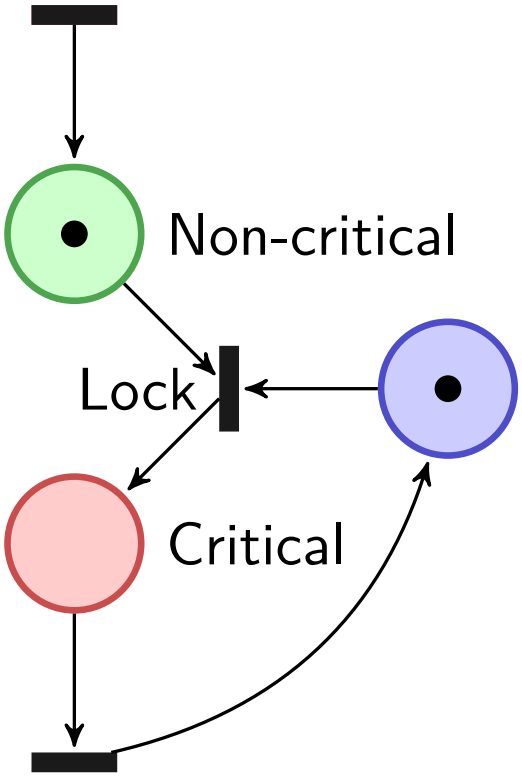
State
$(0, 1, 0)$

Petri nets are a formalism for modelling concurrency



State
$(0, 1, 0)$

Petri nets are a formalism for modelling concurrency



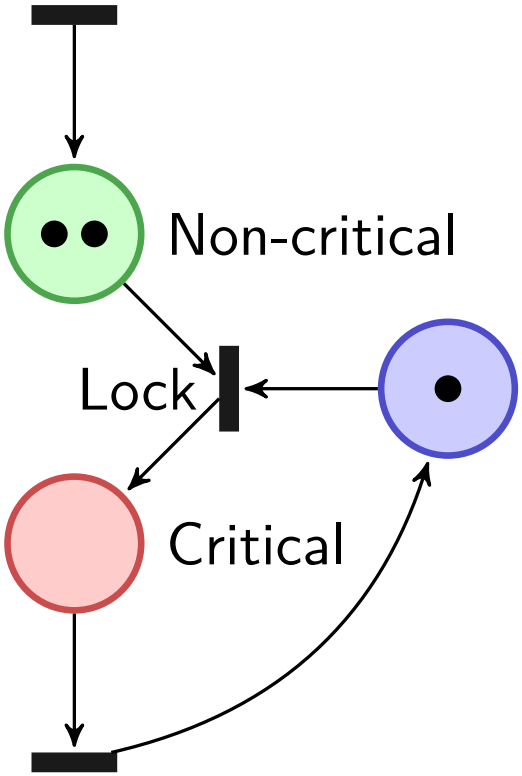
State

$(0, 1, 0)$

$\downarrow$

$(1, 1, 0)$

Petri nets are a formalism for modelling concurrency



State

$(0, 1, 0)$

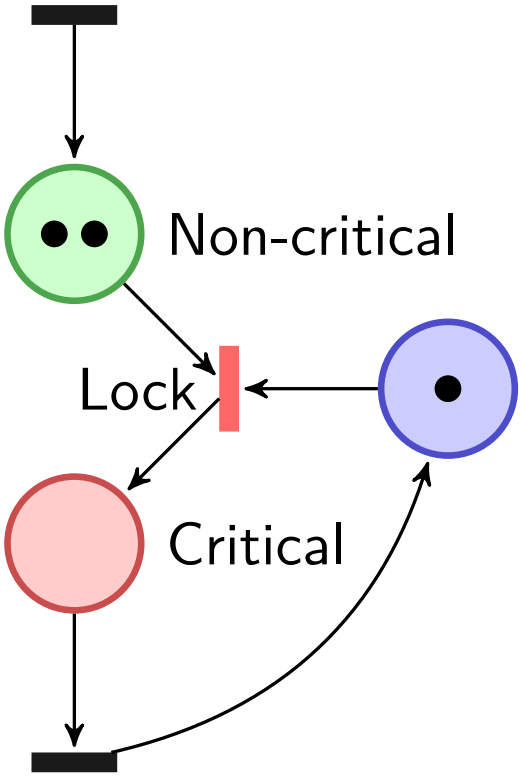
$\downarrow$

$(1, 1, 0)$

$\downarrow$

$(2, 1, 0)$

Petri nets are a formalism for modelling concurrency



State

$(0, 1, 0)$

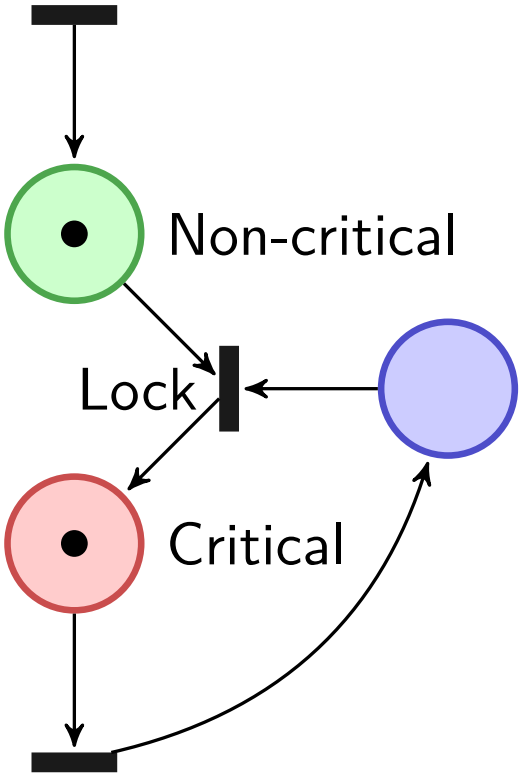
$\downarrow$

$(1, 1, 0)$

$\downarrow$

$(2, 1, 0)$

Petri nets are a formalism for modelling concurrency



State

$(0, 1, 0)$

$\downarrow$

$(1, 1, 0)$

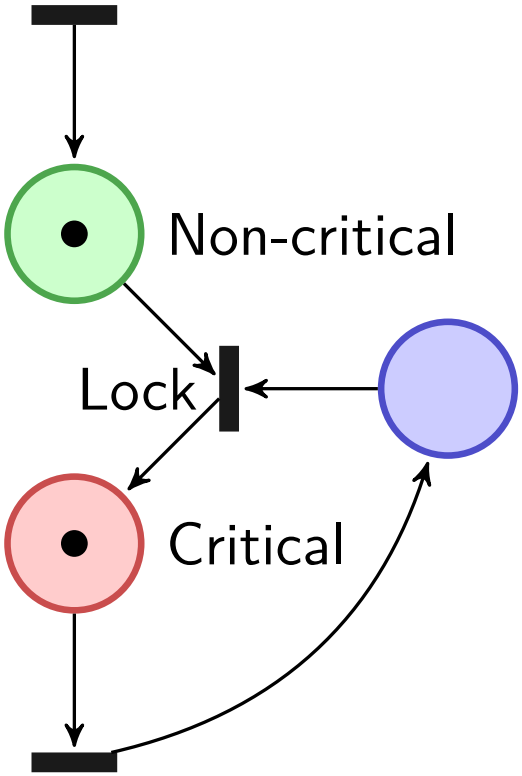
$\downarrow$

$(2, 1, 0)$

$\downarrow$

$(1, 0, 1)$

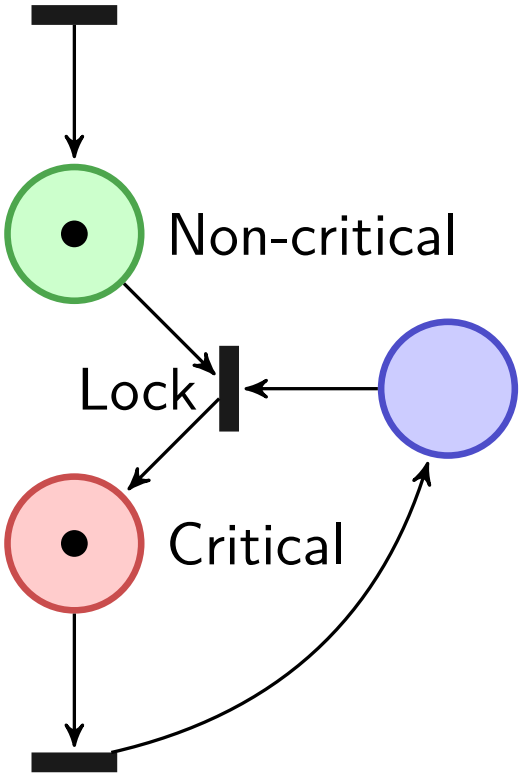
Petri nets are a formalism for modelling concurrency



Petri nets are *infinite-state transition systems*.

Analysis problem: Is (x, y, z) with $z > 1$ reachable?

Petri nets are a formalism for modelling concurrency



Petri nets are *infinite-state transition systems*.

Analysis problem: Is (x, y, z) with $z > 1$ reachable?

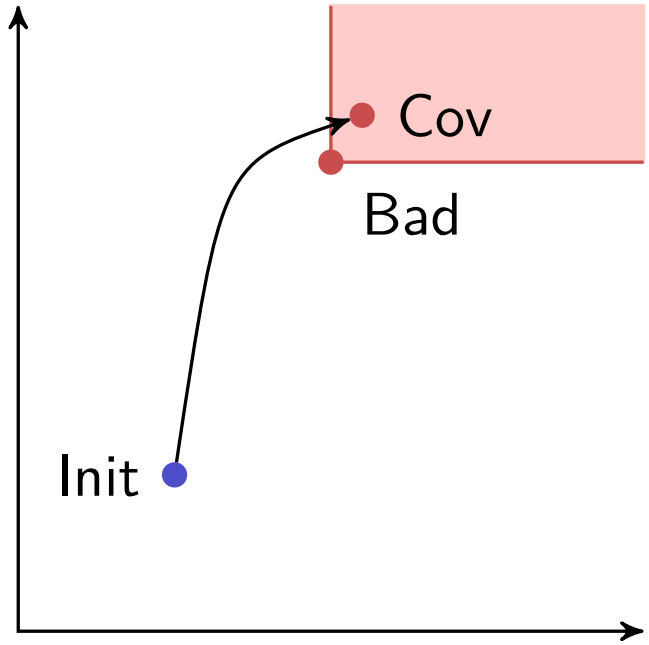
Coverability Problem

Definition

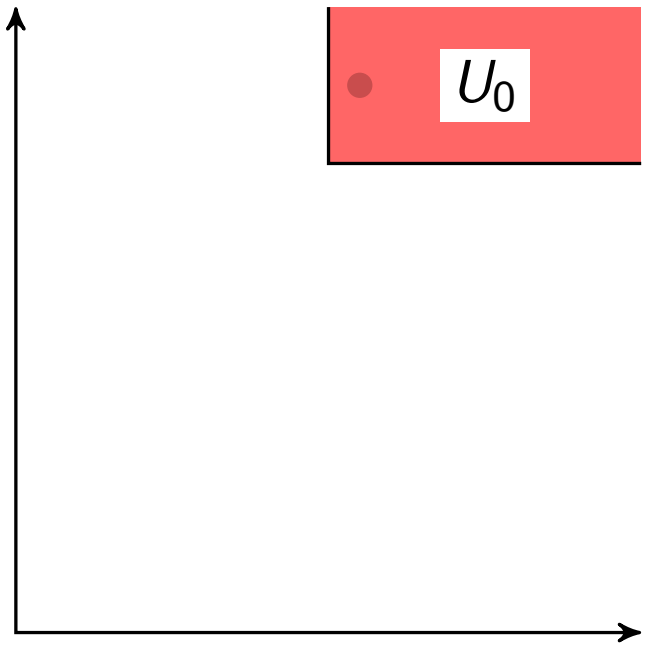
Given

- a Petri net with initial state Init
- a bad state Bad.

does Init *cover* the state Bad, i.e., is there a state Cov such that Init $\rightarrow^*$ Cov and Bad $\leq$ Cov, where $\leq$ is defined pointwise?

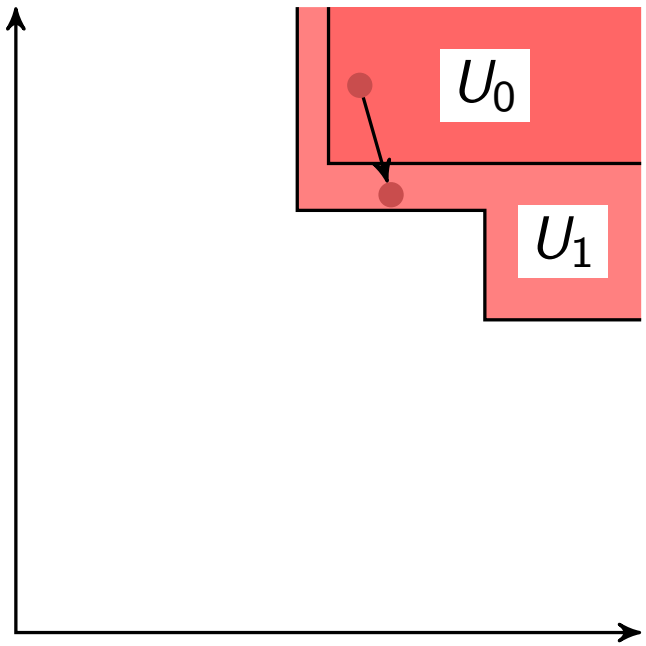


A classical solution for the coverability problem



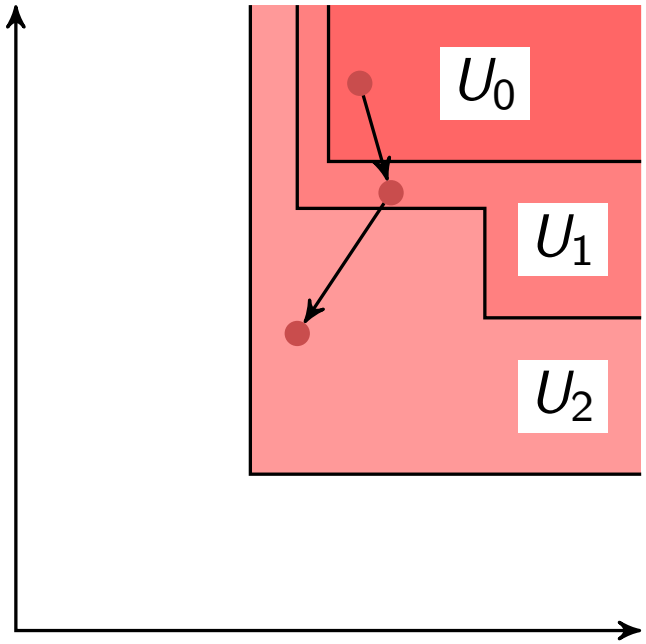
- $U_0 =$ bad states
- $U_{i+1} = U_i \cup \mathrm{pre}(U_i)$
- If $U_i = U_{i+1}$, $U_i = \mathrm{pre}^*(U_0)$.
- Init covers a bad state iff $\mathrm{Init} \in \mathrm{pre}^*(U_0)$.

A classical solution for the coverability problem



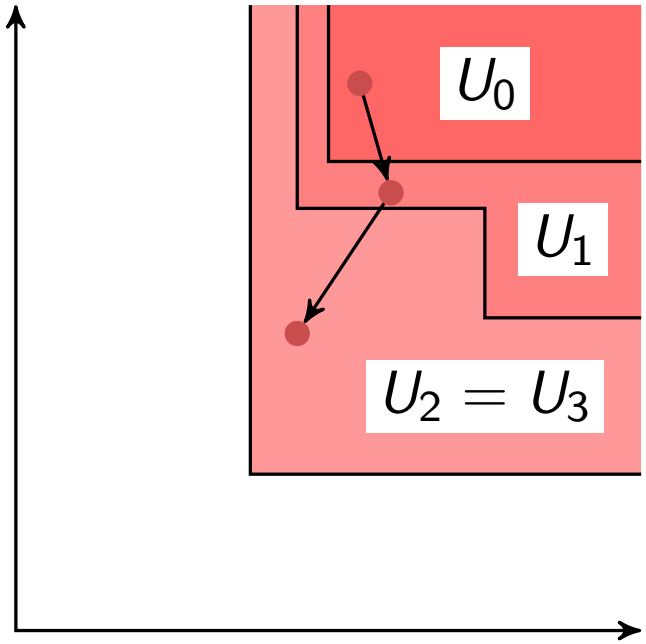
- $U_0 =$ bad states
- $U_{i+1} = U_i \cup \mathsf{pre}(U_i)$
- If $U_i = U_{i+1}$, $U_i = \mathsf{pre}^*(U_0)$.
- Init covers a bad state iff $\mathsf{Init} \in \mathsf{pre}^*(U_0)$.

A classical solution for the coverability problem



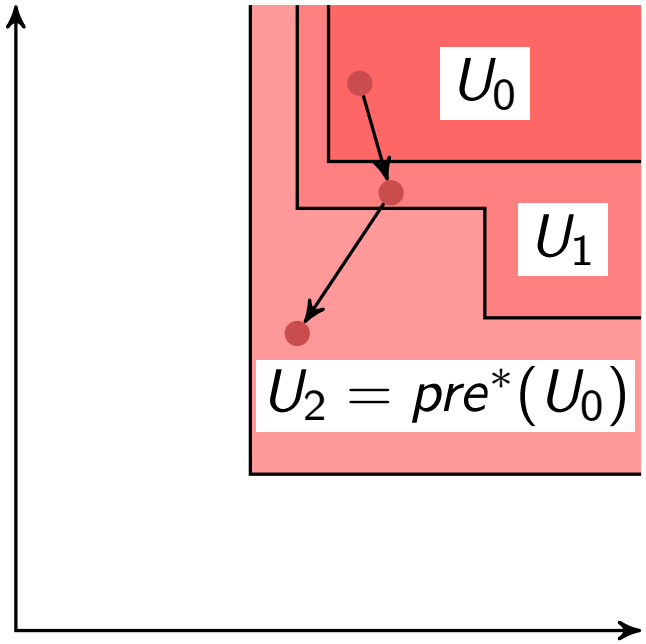
- $U_0 =$ bad states
- $U_{i+1} = U_i \cup \mathsf{pre}(U_i)$
- If $U_i = U_{i+1}$, $U_i = \mathsf{pre}^*(U_0)$.
- Init covers a bad state iff $\mathsf{Init} \in \mathsf{pre}^*(U_0)$.

A classical solution for the coverability problem



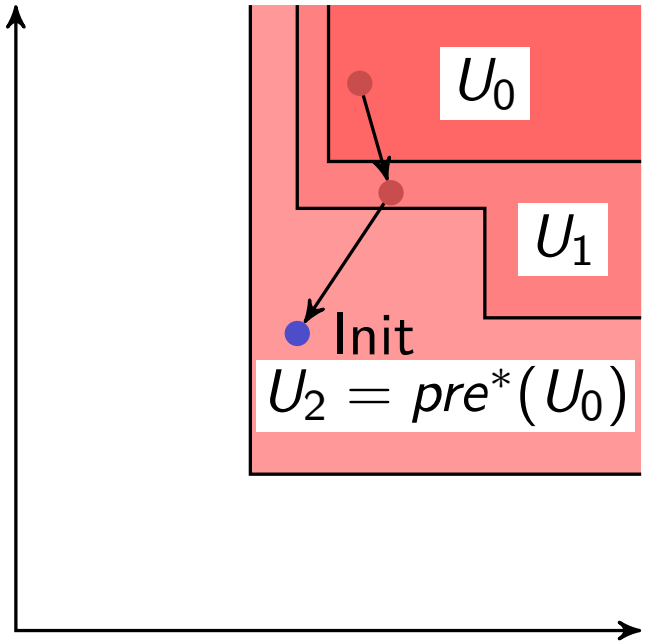
- $U_0 =$ bad states
- $U_{i+1} = U_i \cup \mathrm{pre}(U_i)$
- If $U_i = U_{i+1}$, $U_i = \mathrm{pre}^*(U_0)$.
- Init covers a bad state iff $\mathrm{Init} \in \mathrm{pre}^*(U_0)$.

A classical solution for the coverability problem



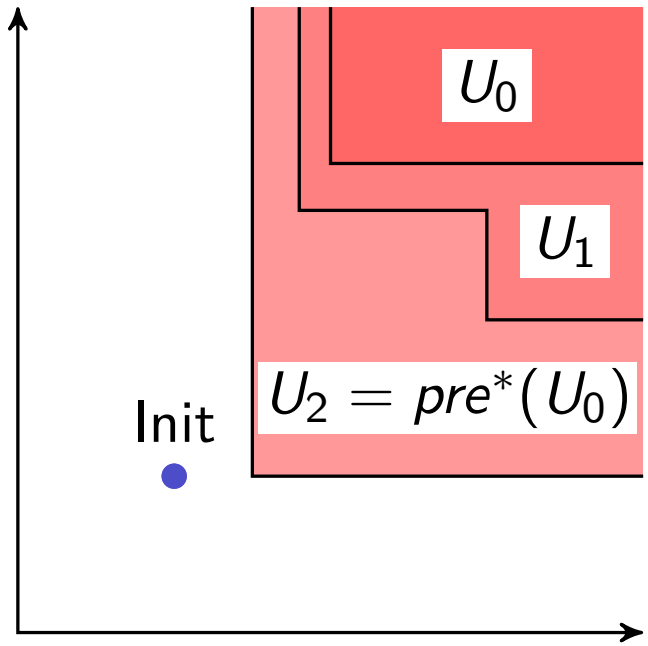
- $U_0 =$ bad states
- $U_{i+1} = U_i \cup \text{pre}(U_i)$
- If $U_i = U_{i+1}$, $U_i = \text{pre}^*(U_0)$.
- Init covers a bad state iff Init $\in \text{pre}^*(U_0)$.

A classical solution for the coverability problem



- $U_0 =$ bad states
- $U_{i+1} = U_i \cup \text{pre}(U_i)$
- If $U_i = U_{i+1}$, $U_i = \text{pre}^*(U_0)$.
- Init covers a bad state iff $\text{Init} \in \text{pre}^*(U_0)$.

A classical solution for the coverability problem



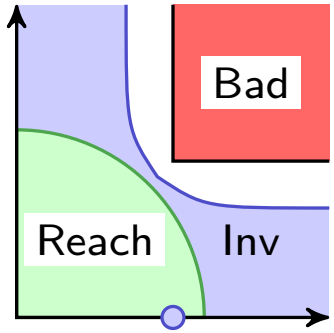
- $U_0 =$ bad states
- $U_{i+1} = U_i \cup \text{pre}(U_i)$
- If $U_i = U_{i+1}$, $U_i = \text{pre}^*(U_0)$.
- Init covers a bad state iff $\text{Init} \in \text{pre}^*(U_0)$.

Solving the coverability problem: Inductive approach

Plain backward search is known to be inefficient.

Inductive methods are often better:
Find a set Inv such that

- $\text{Post}(\text{Inv}) \subseteq \text{Inv}$,
- $\text{Init} \subseteq \text{Inv}$,
- $\text{Inv} \cap \text{Bad} = \varnothing$.



Contribution: Incremental, Inductive Coverability

Starting point: Bradley's IC3 algorithm

- IC3: invariant-based reachability for *finite-state* systems.
- IIC: invariant-based coverability for *infinite-state* systems: Petri nets (in fact, well-structured transition systems) .

Properties of IIC:

- terminates for Petri nets, (in fact, for downward-finite WSTS)
- performs reasonably well in experiments.

Contribution: Incremental, Inductive Coverability

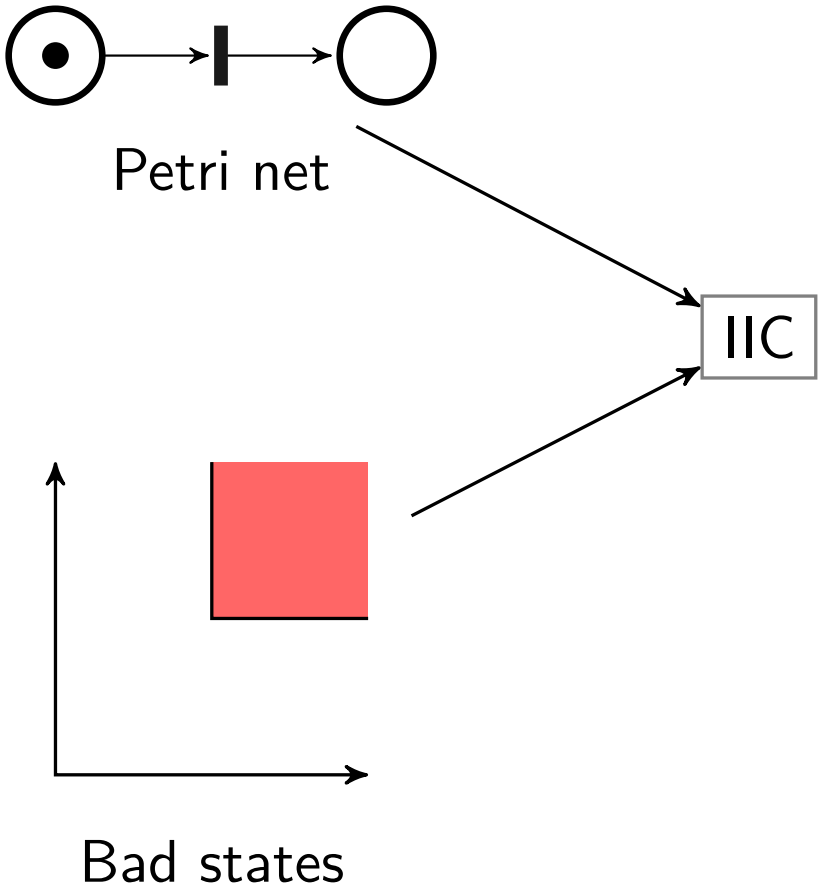
Starting point: Bradley's IC3 algorithm

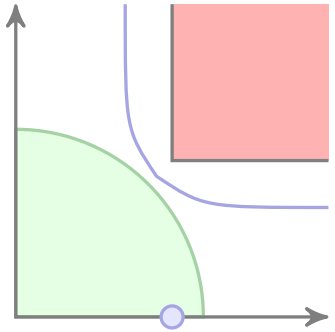
- IC3: invariant-based reachability for *finite-state* systems.
- IIC: invariant-based coverability for *infinite-state* systems: Petri nets (in fact, well-structured transition systems) .

Properties of IIC:

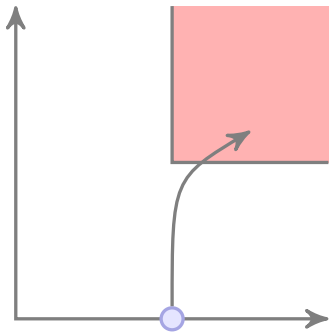
- terminates for Petri nets, (in fact, for downward-finite WSTS)
- performs reasonably well in experiments.

What IIC does



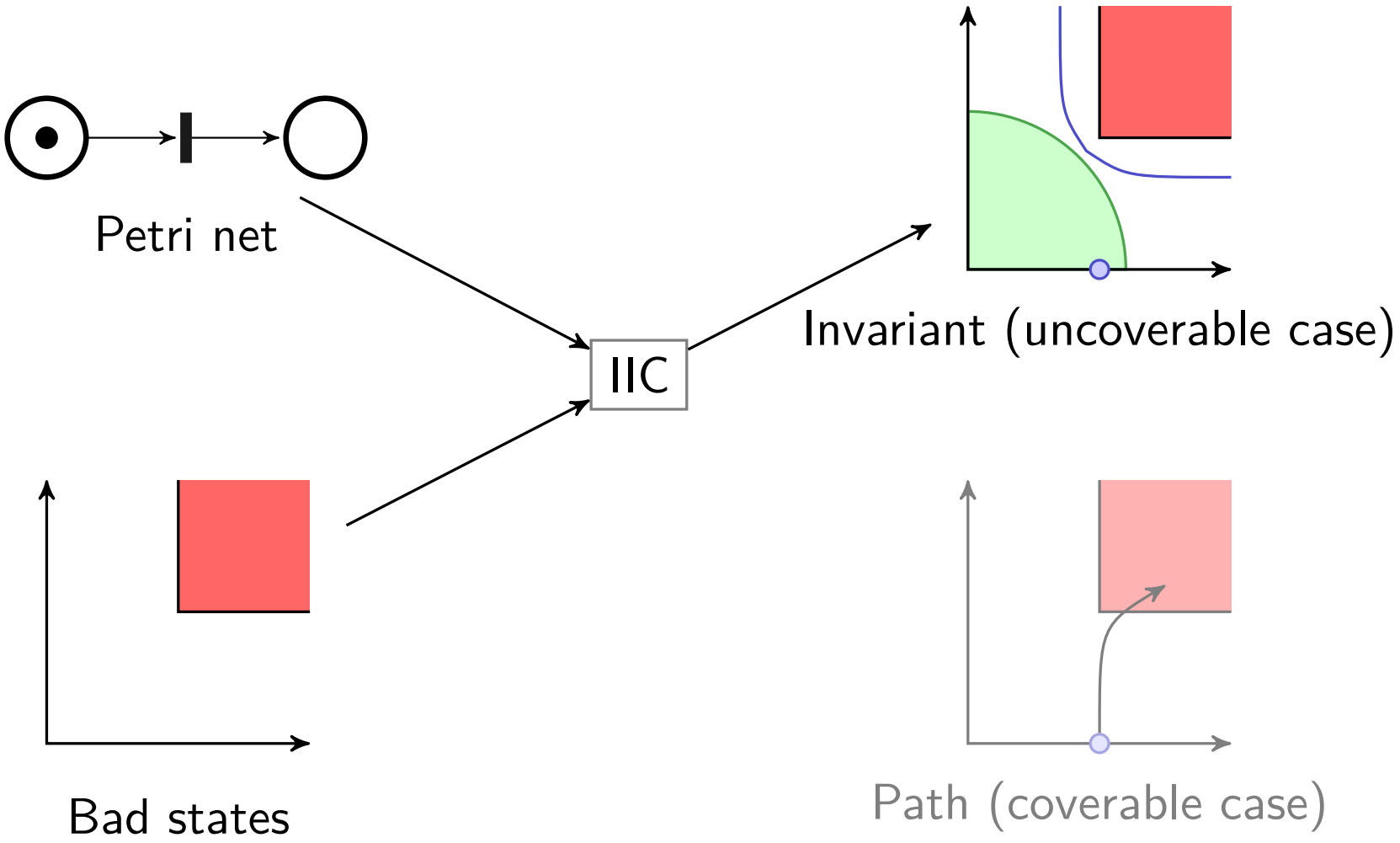


Invariant (uncoverable case)

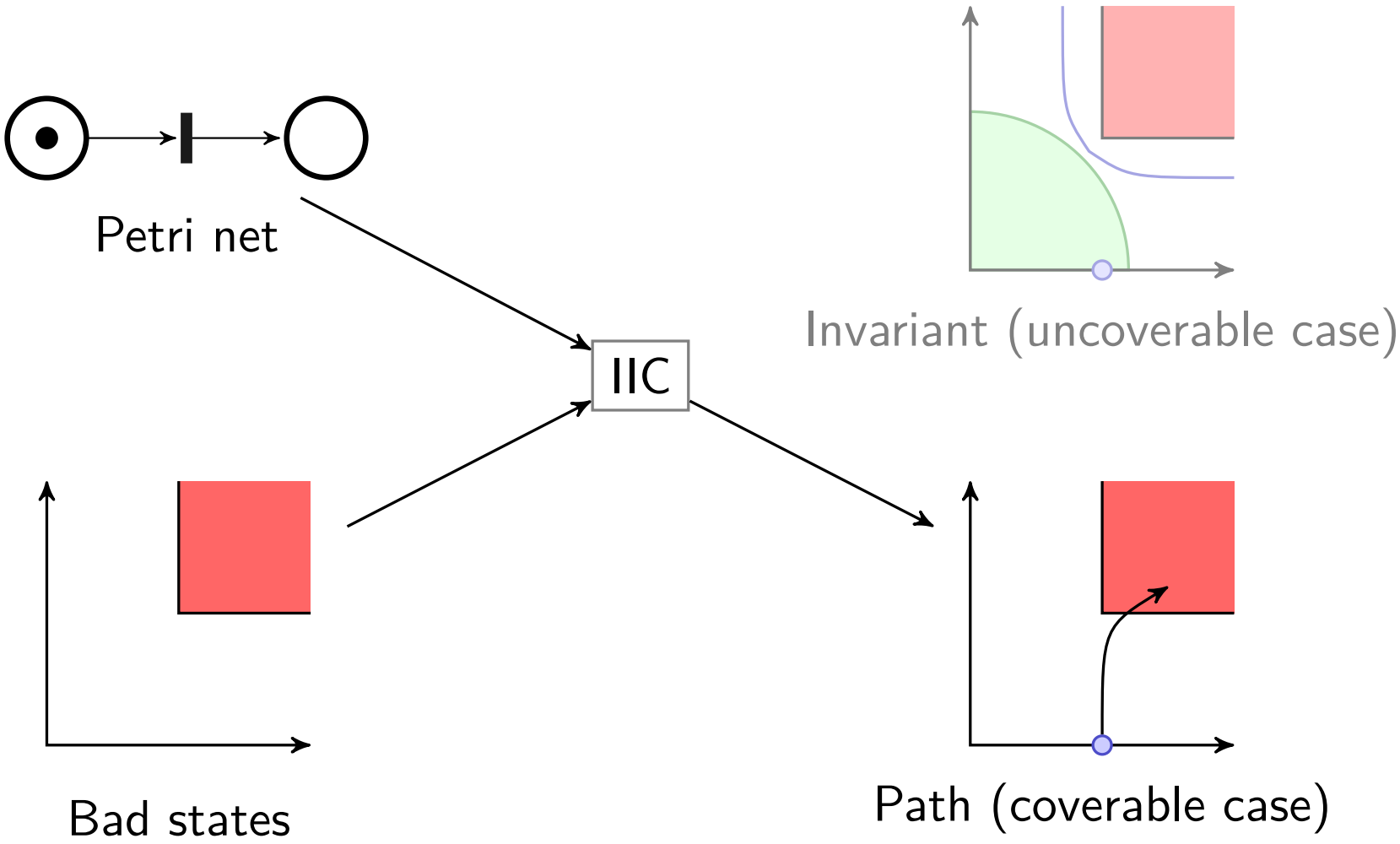


Path (coverable case)

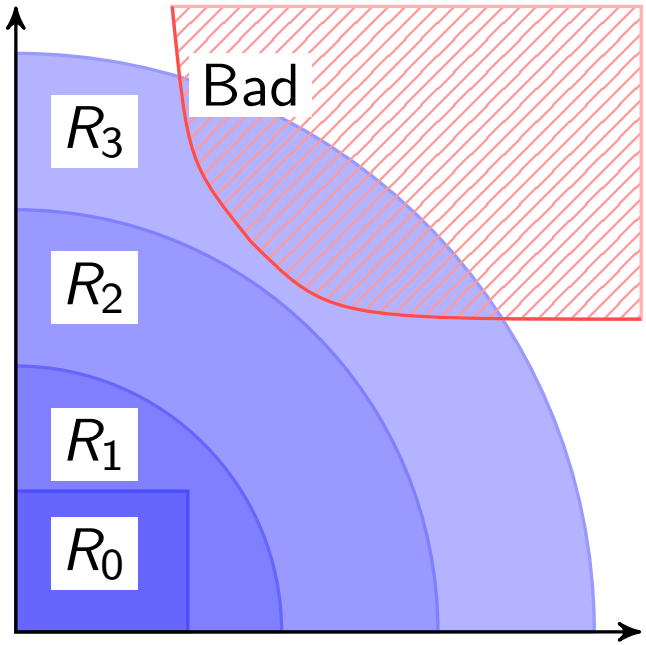
What IIC does



What IIC does



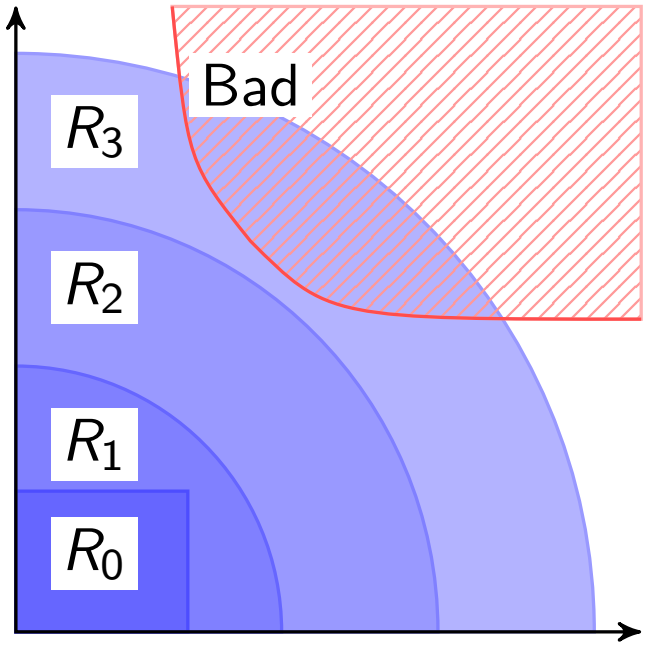
IC3 and IIC use an approximation sequence



$N = 3$

- $R_0 = \text{Init}\downarrow$,
- $R_i \subseteq R_{i+1}$,
- $\text{Post}(R_i) \subseteq R_{i+1}$,
- R_i overapproximates states reachable in i steps.
- $R_0, \ldots, R_N$: Only R_N may contain bad states.

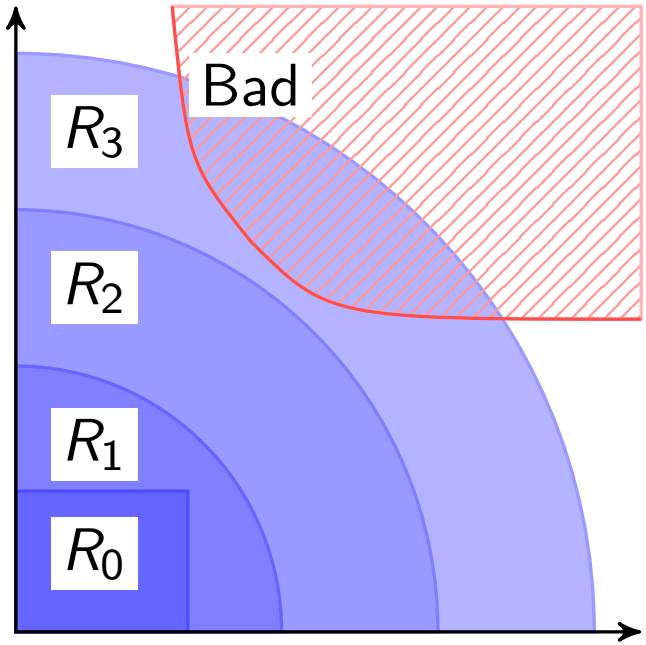
IC3 and IIC use an approximation sequence



$N = 3$

- $R_0 = \mathsf{Init}\downarrow$,
- $R_i \subseteq R_{i+1}$,
- $\mathsf{Post}(R_i) \subseteq R_{i+1}$,
- R_i overapproximates states reachable in i steps.
- $R_0, \ldots, R_N$: Only R_N may contain bad states.

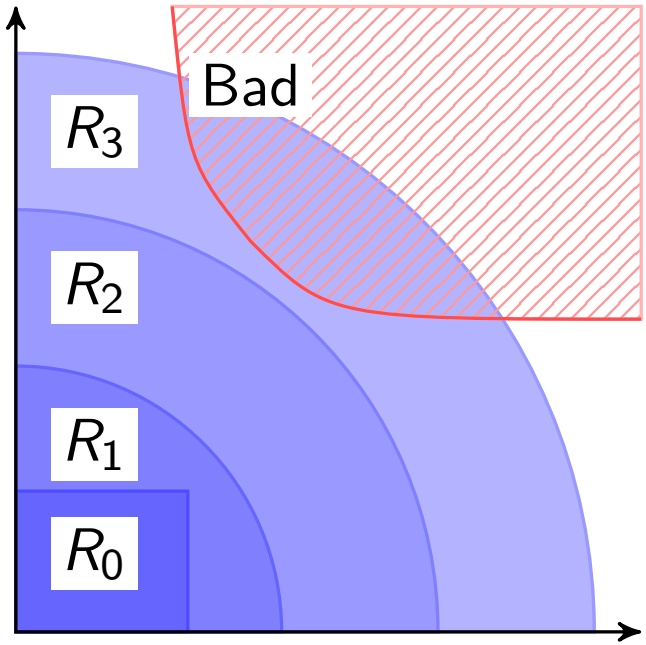
IC3 and IIC use an approximation sequence



$N = 3$

- $R_0 = \mathsf{Init} \downarrow$,
- $R_i \subseteq R_{i+1}$,
- $\mathsf{Post}(R_i) \subseteq R_{i+1}$,
- R_i overapproximates states reachable in i steps.
- $R_0, \ldots, R_N$: Only R_N may contain bad states.

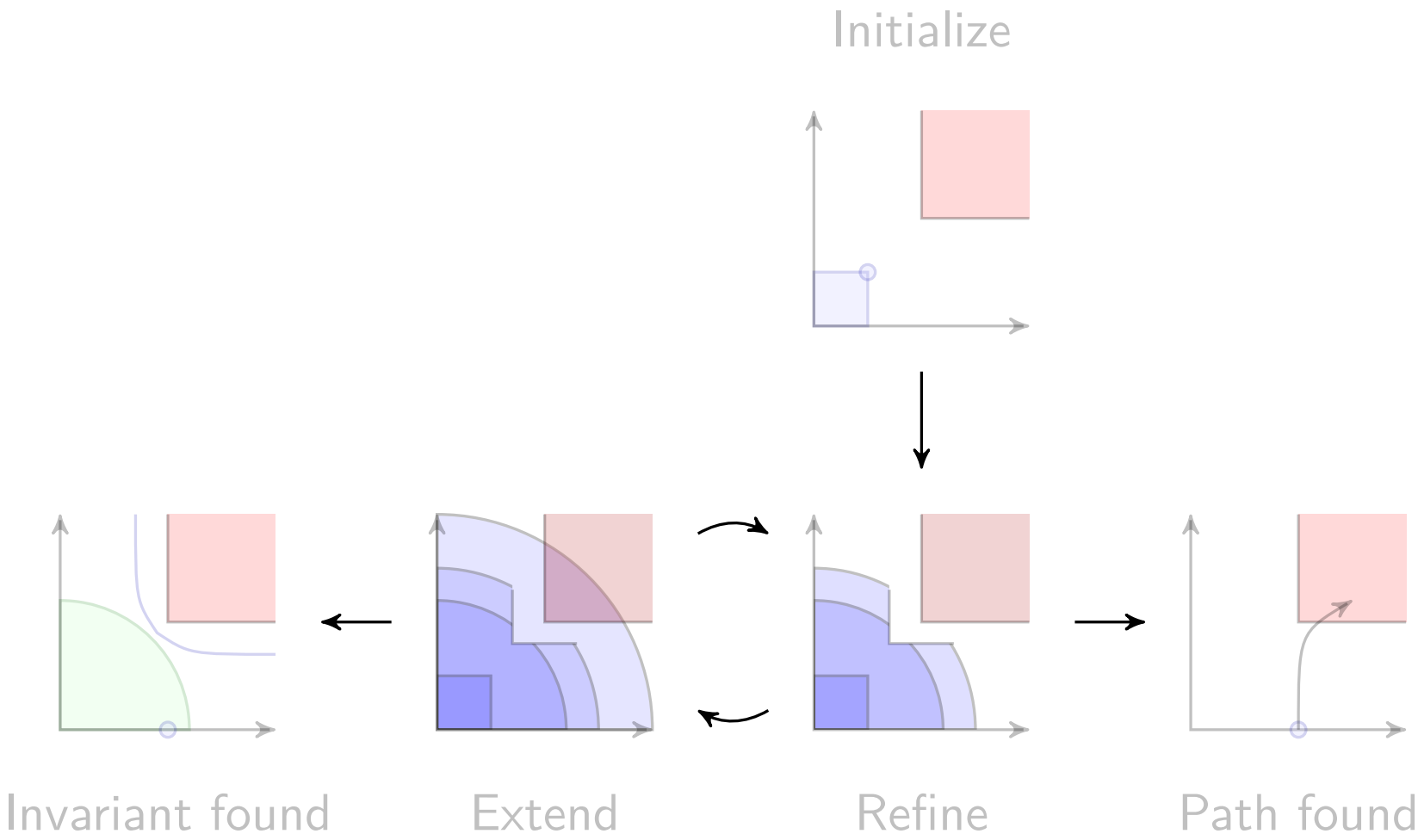
IC3 and IIC use an approximation sequence



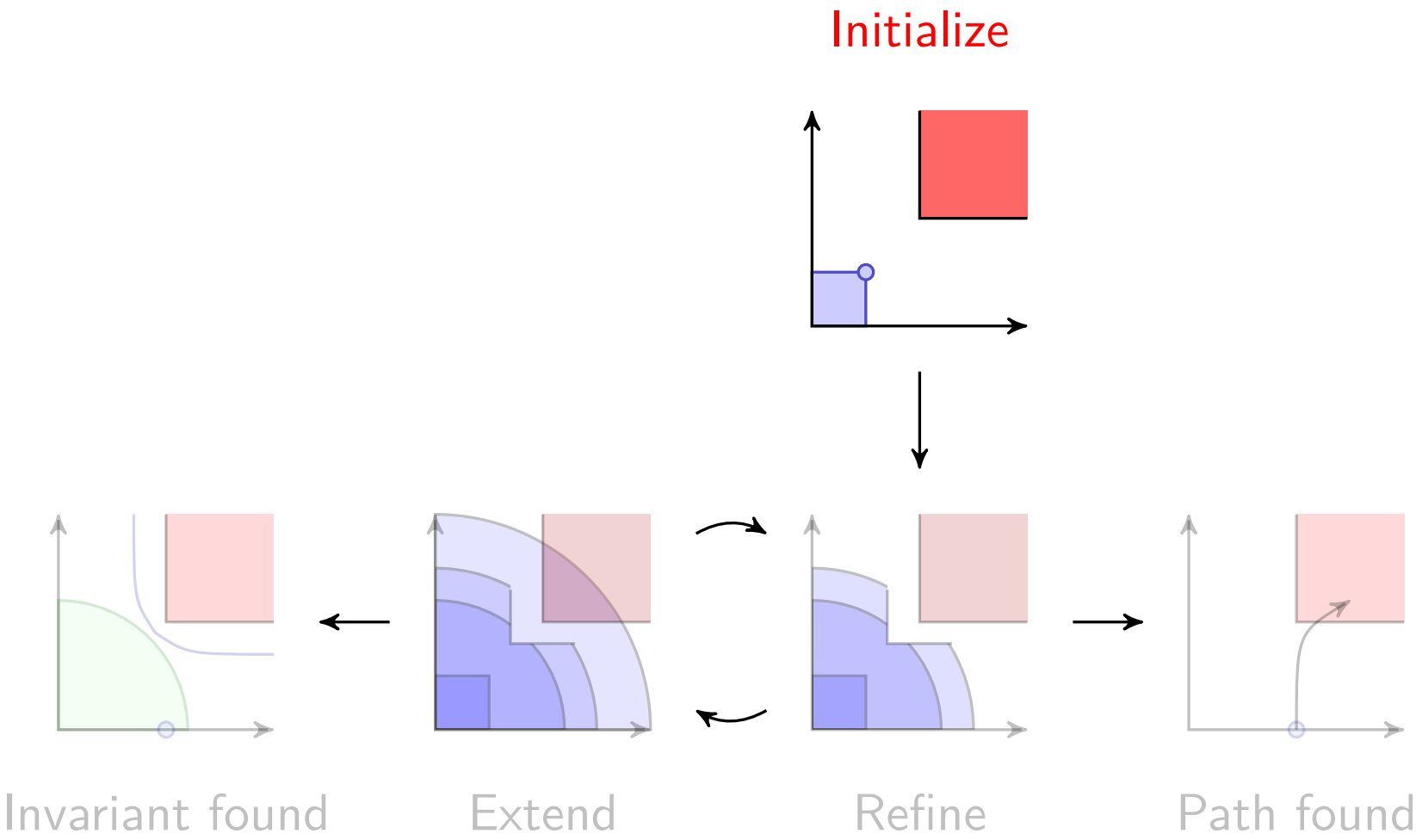
$N = 3$

- $R_0 = \text{Init}\downarrow$,
- $R_i \subseteq R_{i+1}$,
- $\text{Post}(R_i) \subseteq R_{i+1}$,
- R_i overapproximates states reachable in i steps.
- $R_0, \ldots, R_N$: Only R_N may contain bad states.

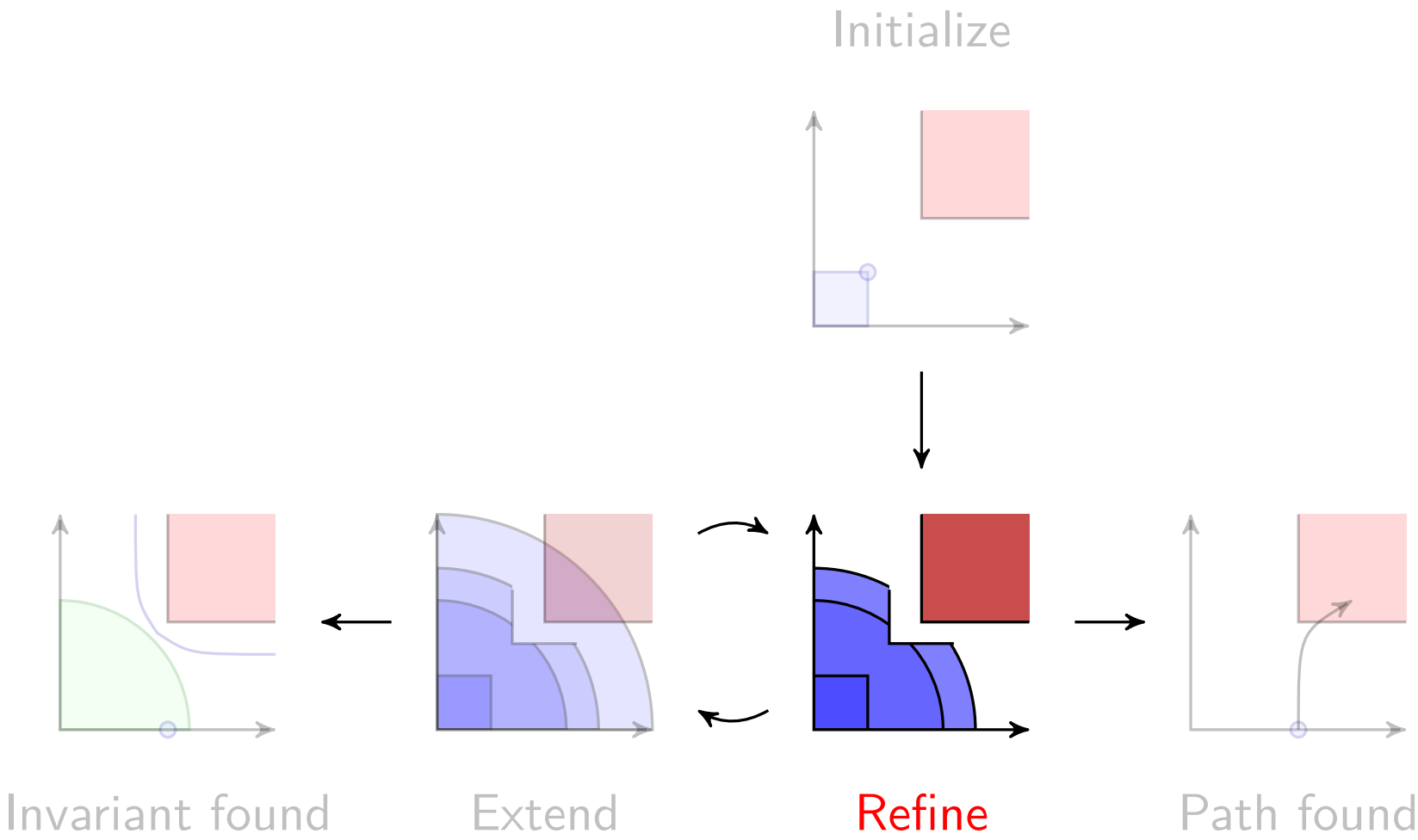
IC3 and IIC in a nutshell



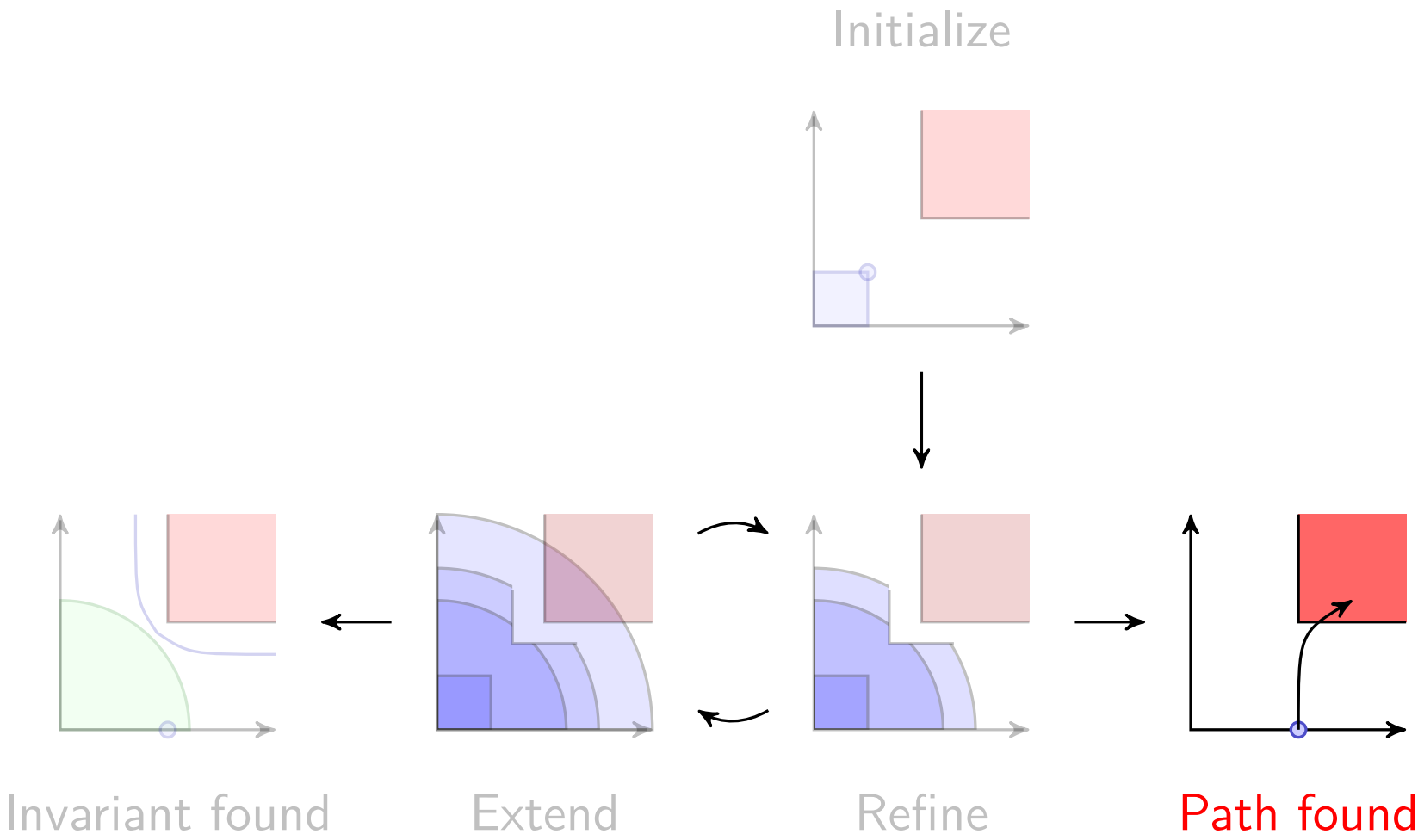
IC3 and IIC in a nutshell



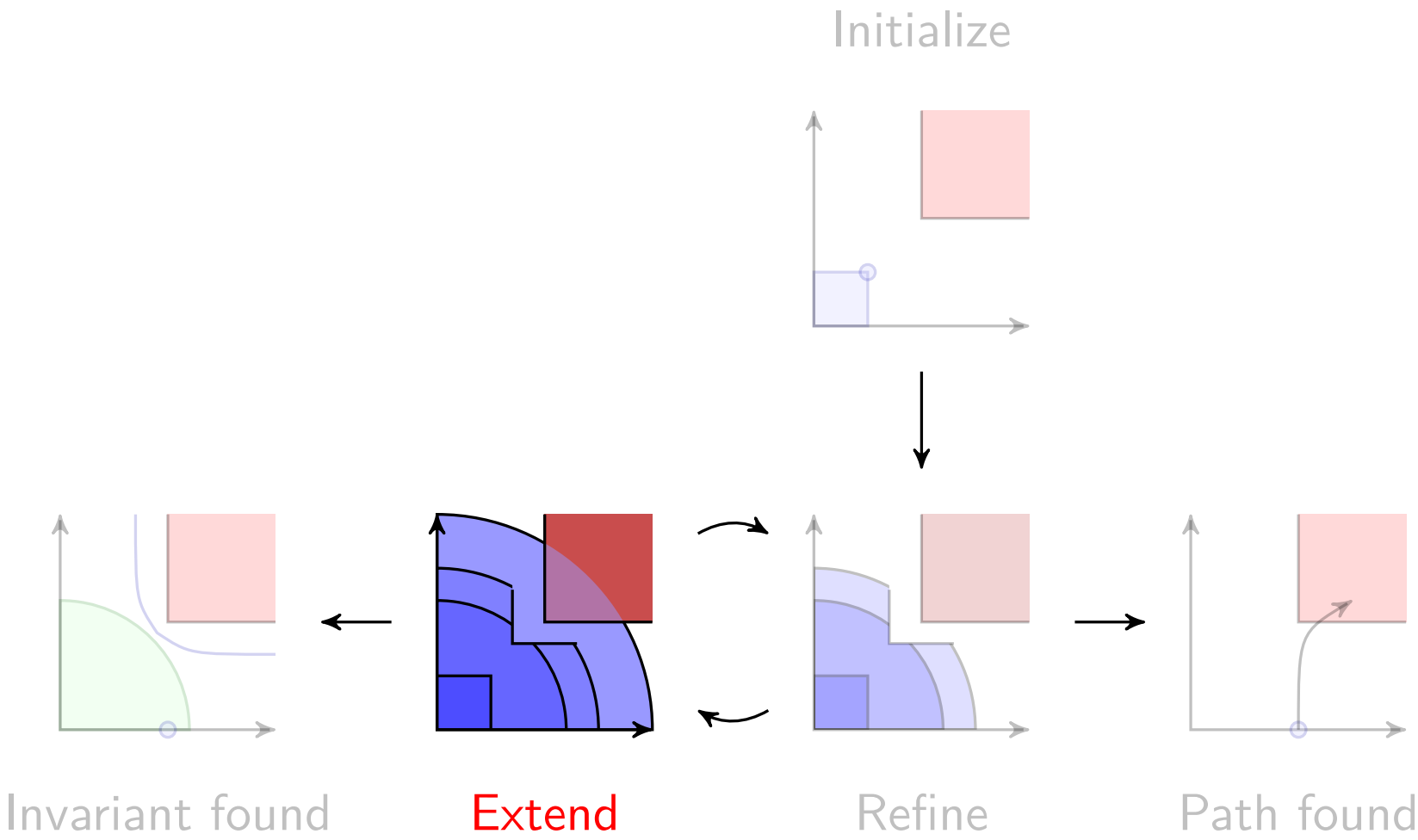
IC3 and IIC in a nutshell



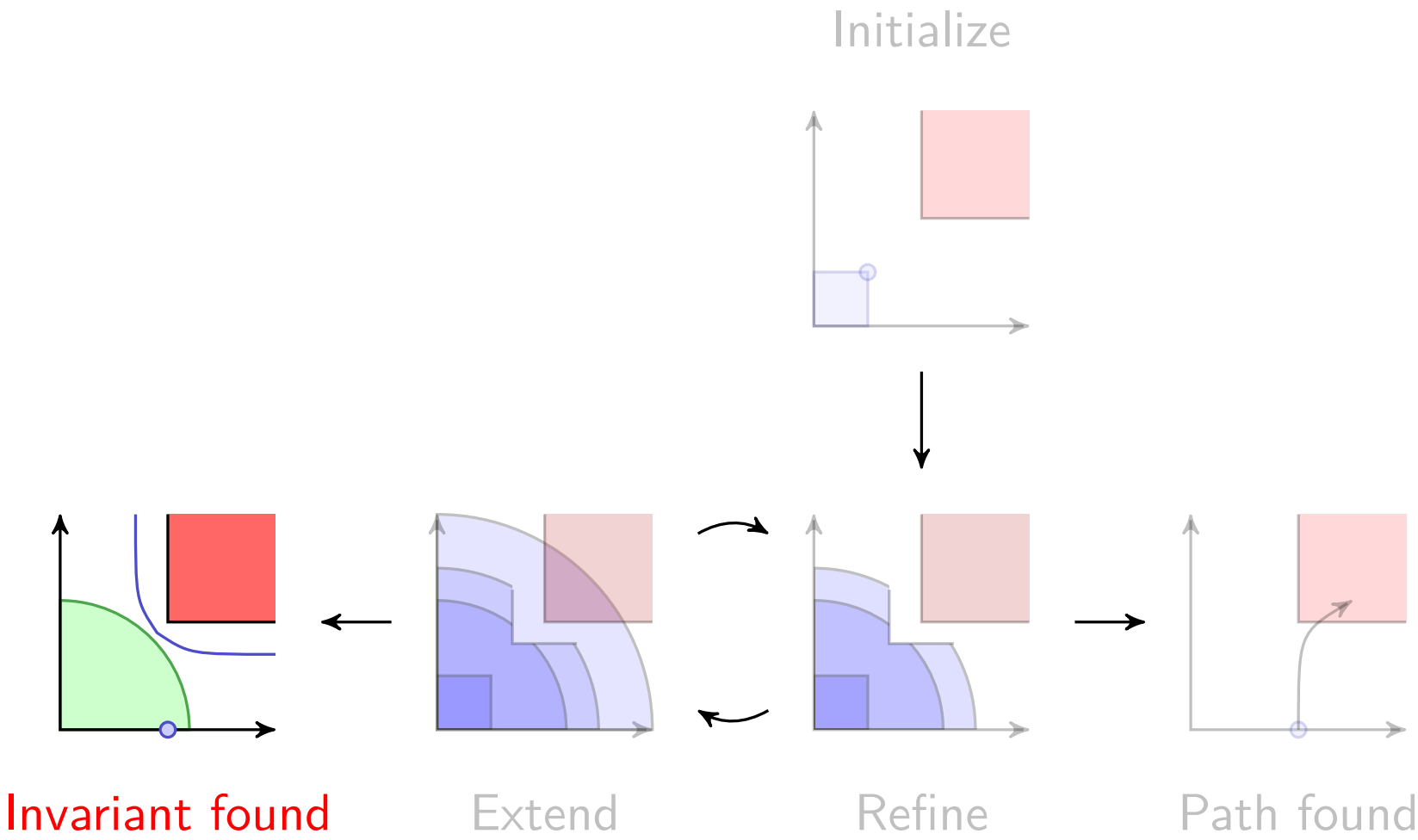
IC3 and IIC in a nutshell



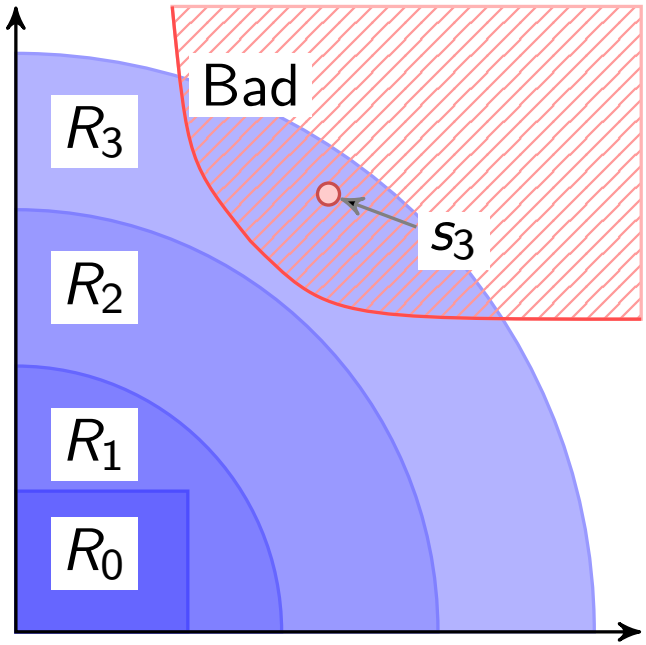
IC3 and IIC in a nutshell



IC3 and IIC in a nutshell

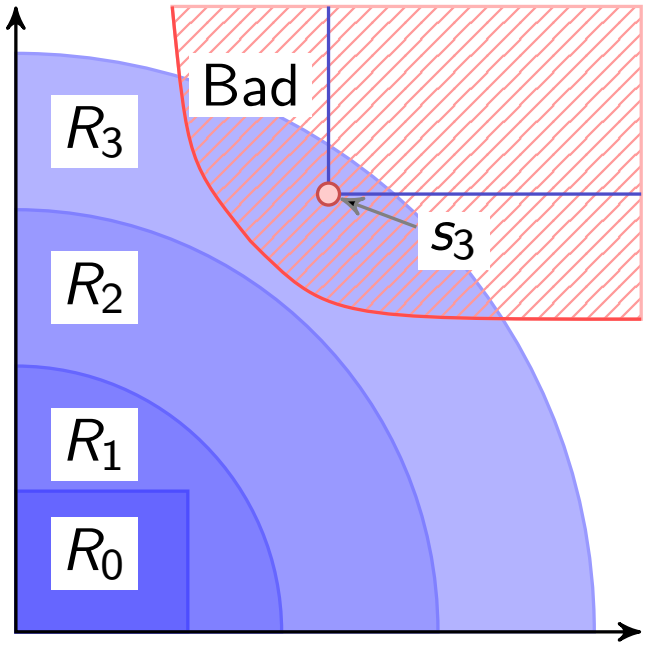


How to rule out bad states



- Try finding a path to a bad state by backward search.
- $s_{i+1} \in R_{i+1} \implies s_i \in \text{minpre}(s_{i+1}\uparrow) \cap R_i$.
- $s_0 \in R_0 \implies$ path from initial to bad states.

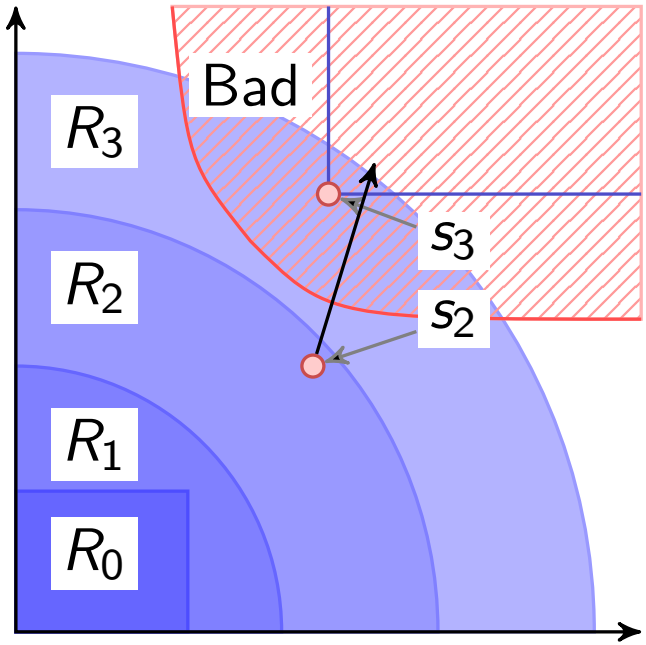
How to rule out bad states



$i = 2$

- Try finding a path to a bad state by backward search.
- $s_{i+1} \in R_{i+1} \implies$
  $s_i \in \text{minpre}(s_{i+1}\uparrow) \cap R_i$.
- $s_0 \in R_0 \implies$ path from initial to bad states.

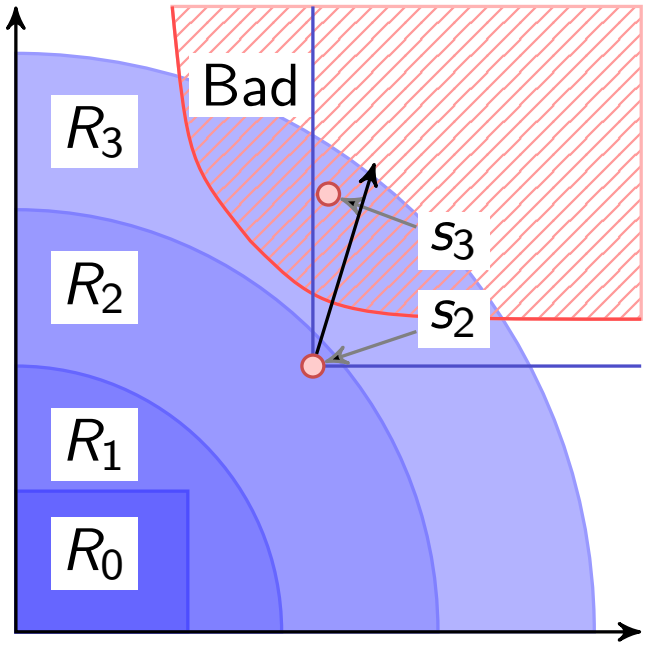
How to rule out bad states



$i = 2$

- Try finding a path to a bad state by backward search.
- $s_{i+1} \in R_{i+1} \implies$
  $s_i \in \text{minpre}(s_{i+1}\uparrow) \cap R_i$.
- $s_0 \in R_0 \implies$ path from initial to bad states.

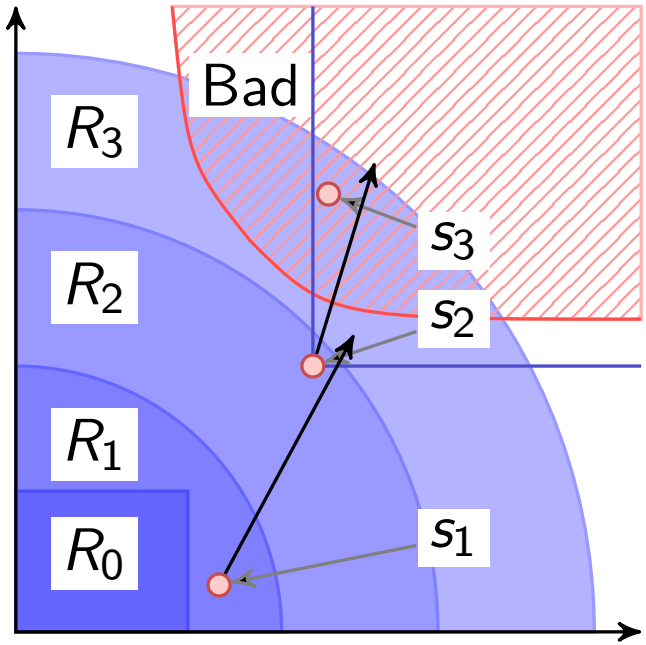
How to rule out bad states



$i = 1$

- Try finding a path to a bad state by backward search.
- $s_{i+1} \in R_{i+1} \implies$
  $s_i \in \text{minpre}(s_{i+1}\uparrow) \cap R_i$.
- $s_0 \in R_0 \implies$ path from initial to bad states.

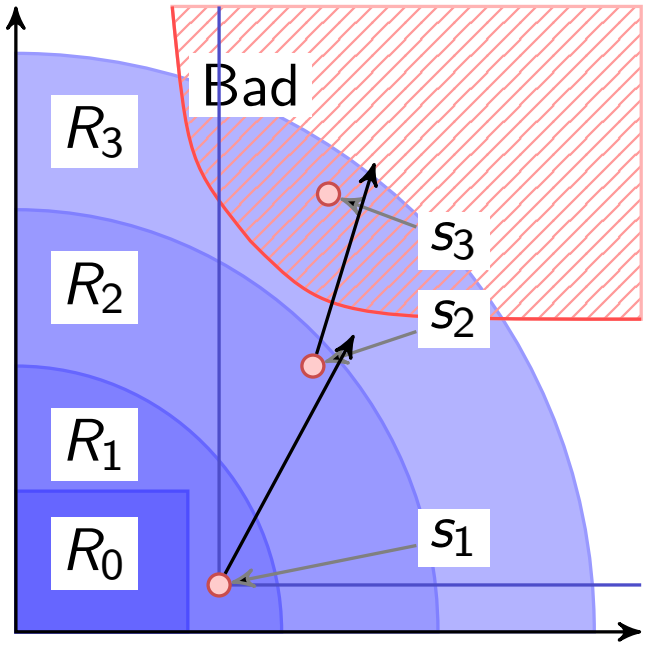
How to rule out bad states



$i = 1$

- Try finding a path to a bad state by backward search.
- $s_{i+1} \in R_{i+1} \implies$
  $s_i \in \text{minpre}(s_{i+1}\uparrow) \cap R_i$.
- $s_0 \in R_0 \implies$ path from initial to bad states.

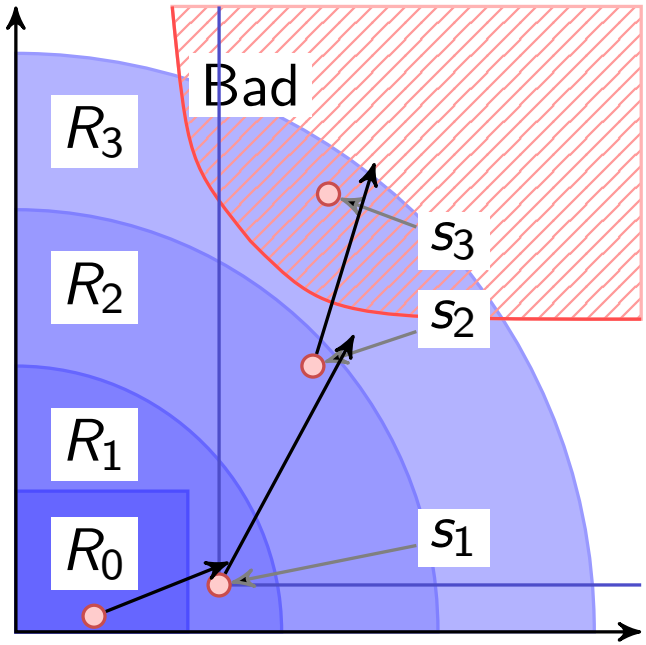
How to rule out bad states



$i = 0$

- Try finding a path to a bad state by backward search.
- $s_{i+1} \in R_{i+1} \implies$
  $s_i \in \text{minpre}(s_{i+1}\uparrow) \cap R_i$.
- $s_0 \in R_0 \implies$ path from initial to bad states.

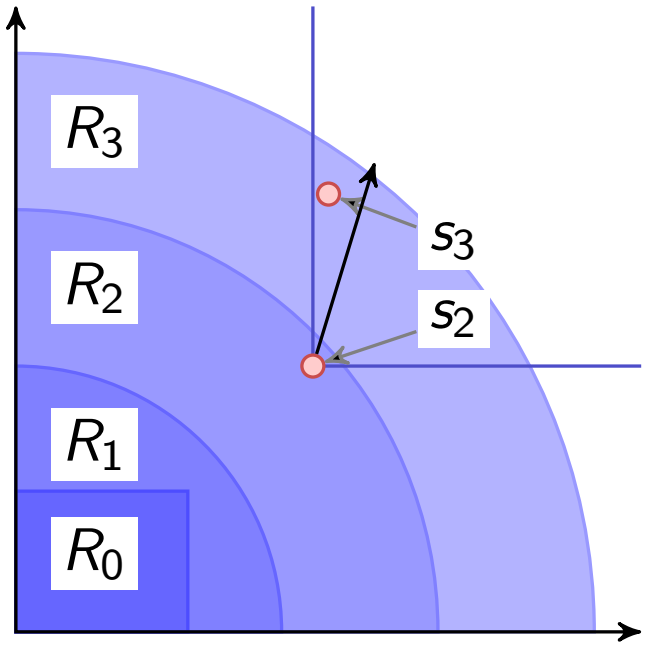
How to rule out bad states



$i = 0$

- Try finding a path to a bad state by backward search.
- $s_{i+1} \in R_{i+1} \implies$ $s_i \in \text{minpre}(s_{i+1}\uparrow) \cap R_i$.
- $s_0 \in R_0 \implies$ path from initial to bad states.

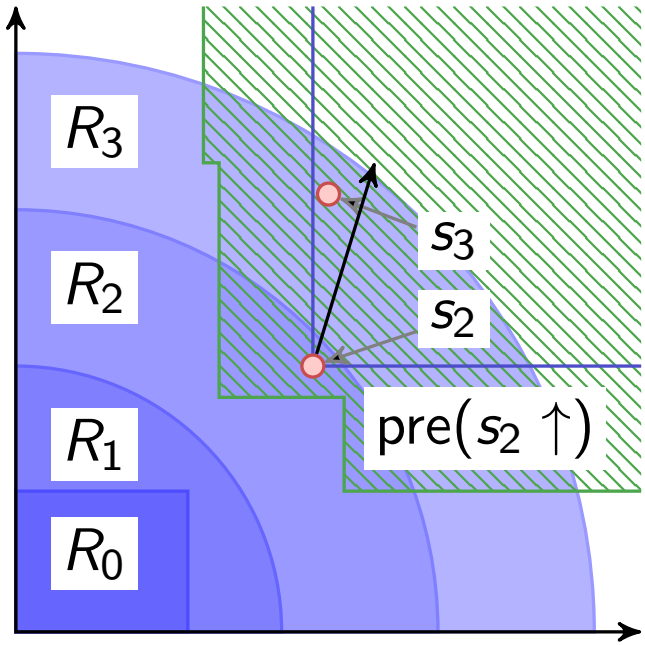
How to rule out bad states



$i = 1$

If $\text{minpre}(s_{i+1}\uparrow) \cap R_i = \varnothing$, overapproximation R_{i+1} is too coarse.

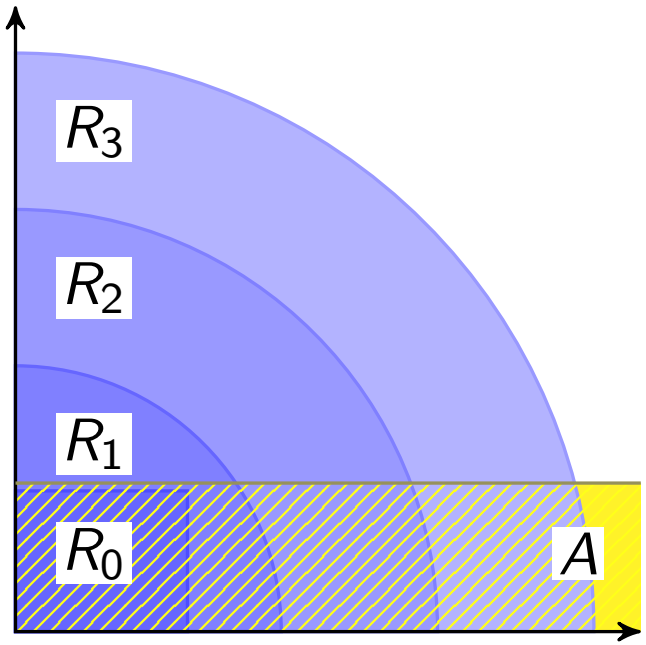
How to rule out bad states



$i = 1$

If $\mathrm{minpre}(s_{i+1} \uparrow) \cap R_i = \varnothing$, overapproximation R_{i+1} is too coarse.

We can use relative inductive sets to refine approximations

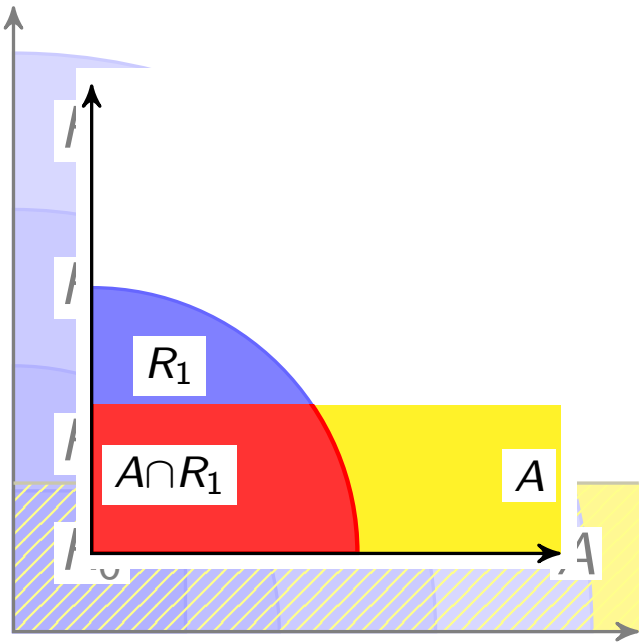


Assume

- A downward-closed, $R_0 \subseteq A$,
- A is *inductive relative* to R_i: $\mathrm{Post}(A \cap R_i) \subseteq A$.

Then replace $R_1, \ldots, R_{i+1}$ by $R_1 \cap A, \ldots, R_{i+1} \cap A$.

We can use relative inductive sets to refine approximations



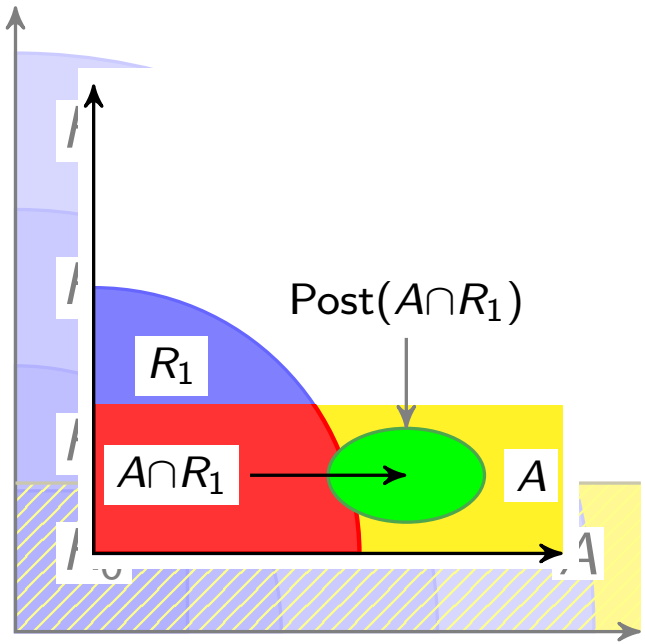
$i = 1$

Assume

- A downward-closed, $R_0 \subseteq A$,
- A is *inductive relative* to R_i: $\mathrm{Post}(A \cap R_i) \subseteq A$.

Then replace $R_1, \ldots, R_{i+1}$ by $R_1 \cap A, \ldots, R_{i+1} \cap A$.

We can use relative inductive sets to refine approximations



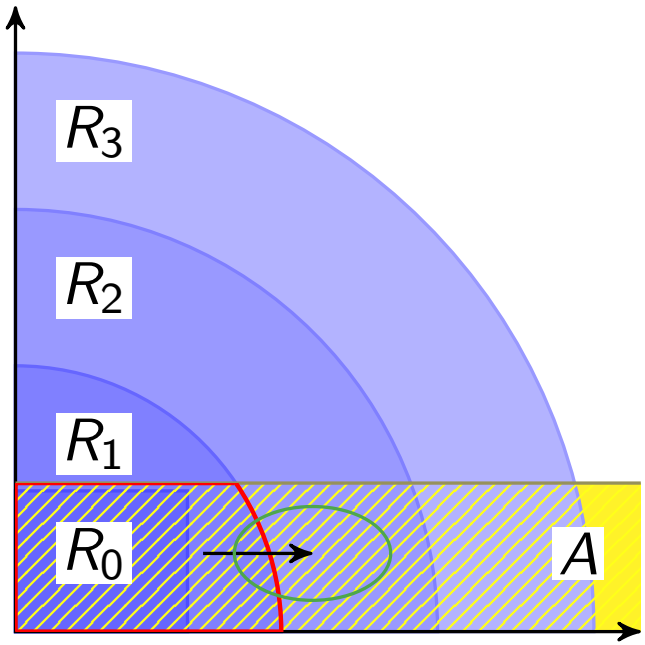
$i = 1$

Assume

- A downward-closed, $R_0 \subseteq A$,
- A is *inductive relative* to R_i: $\text{Post}(A \cap R_i) \subseteq A$.

Then replace $R_1, \ldots, R_{i+1}$ by $R_1 \cap A, \ldots, R_{i+1} \cap A$.

We can use relative inductive sets to refine approximations



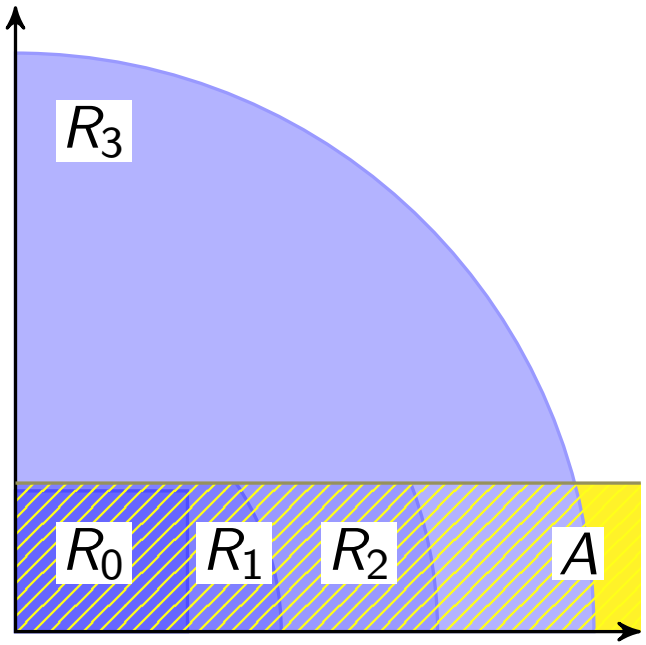
$i = 1$

Assume

- A downward-closed, $R_0 \subseteq A$,
- A is *inductive relative* to R_i: $\mathrm{Post}(A \cap R_i) \subseteq A$.

Then replace $R_1, \ldots, R_{i+1}$ by $R_1 \cap A, \ldots, R_{i+1} \cap A$.

We can use relative inductive sets to refine approximations



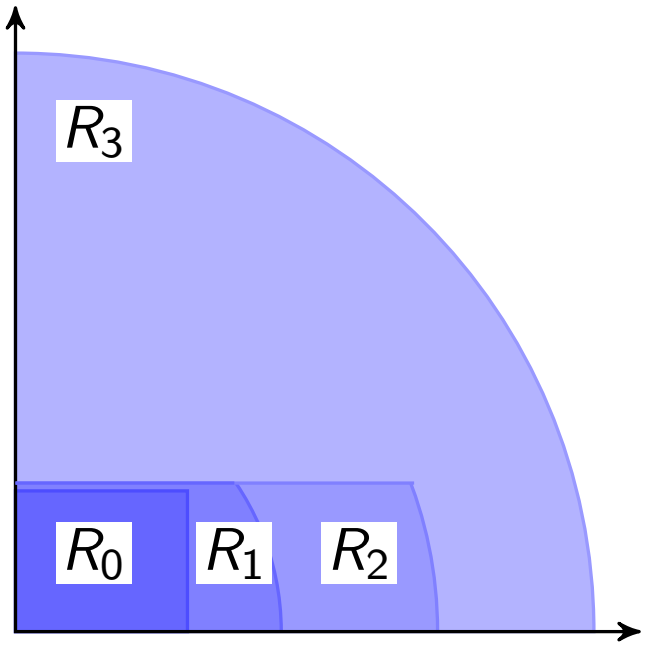
$i = 1$

Assume

- A downward-closed, $R_0 \subseteq A$,
- A is *inductive relative* to R_i: $\mathrm{Post}(A \cap R_i) \subseteq A$.

Then replace $R_1, \ldots, R_{i+1}$ by $R_1 \cap A, \ldots, R_{i+1} \cap A$.

We can use relative inductive sets to refine approximations



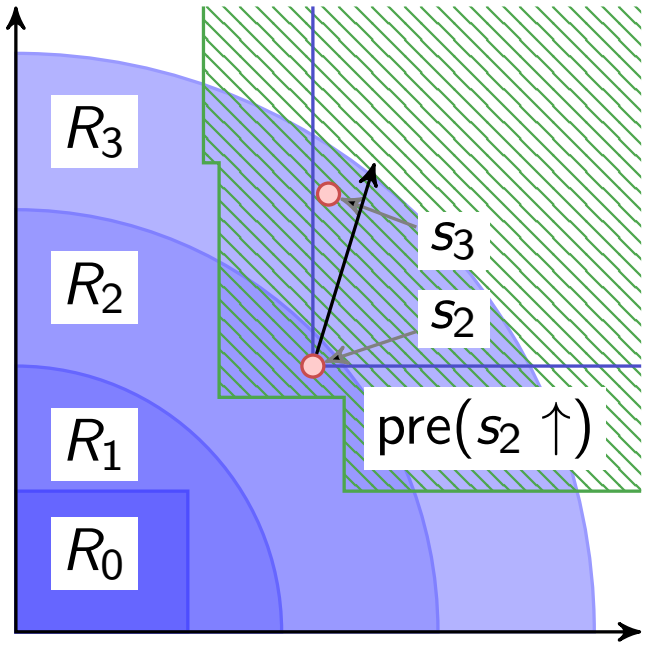
$i = 1$

Assume

- A downward-closed, $R_0 \subseteq A$,
- A is *inductive relative* to R_i: $\text{Post}(A \cap R_i) \subseteq A$.

Then replace $R_1, \ldots, R_{i+1}$ by $R_1 \cap A, \ldots, R_{i+1} \cap A$.

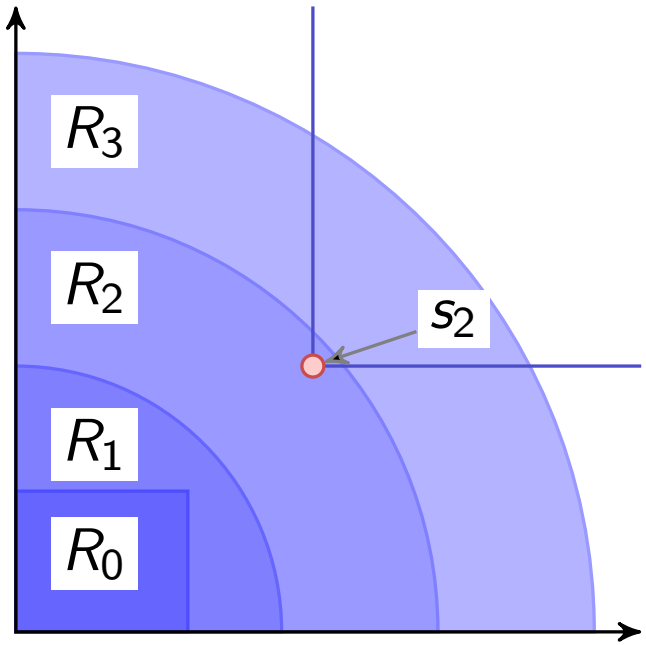
Spurious states can be removed via relative inductiveness



$i = 1$

- If $\mathrm{minpre}(s_{i+1} \uparrow) \cap R_i = \varnothing$, overapproximation R_{i+1} is too coarse.
- Set $C := (s_{i+1} \uparrow)^C$. C is downward-closed, $R_0 \subseteq C$, C inductive relative to R_i.

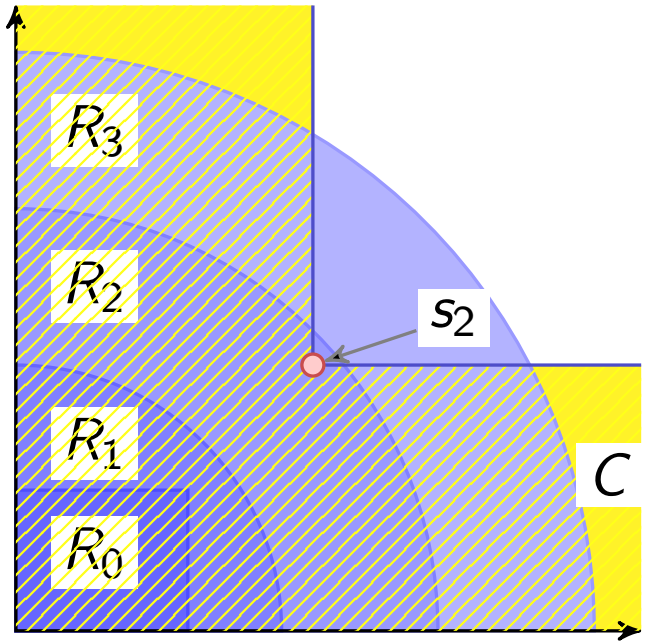
Spurious states can be removed via relative inductiveness



$i = 1$

- If $\text{minpre}(s_{i+1}\uparrow) \cap R_i = \varnothing$, overapproximation R_{i+1} is too coarse.
- Set $C := (s_{i+1}\uparrow)^C$.
  C is downward-closed, $R_0 \subseteq C$, C inductive relative to R_i.

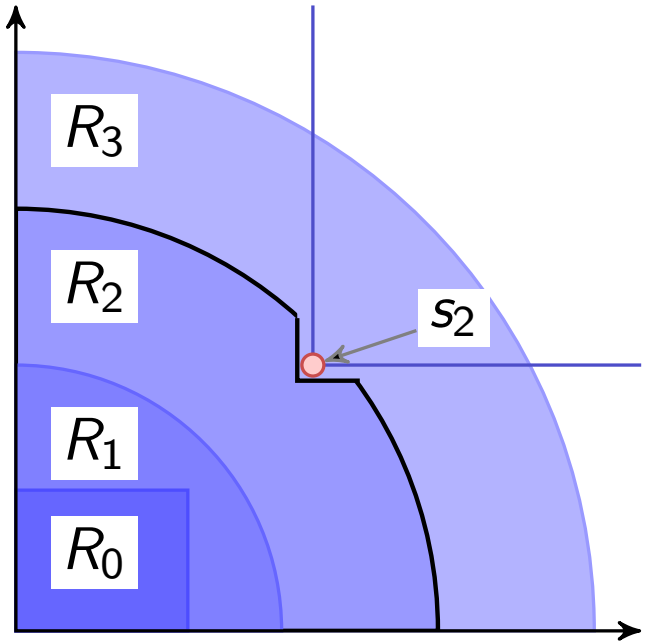
Spurious states can be removed via relative inductiveness



$i = 1$

- If $\text{minpre}(s_{i+1}\uparrow) \cap R_i = \varnothing$, overapproximation R_{i+1} is too coarse.
- Set $C := (s_{i+1}\uparrow)^C$.
  C is downward-closed,
  $R_0 \subseteq C$, C inductive relative to R_i.

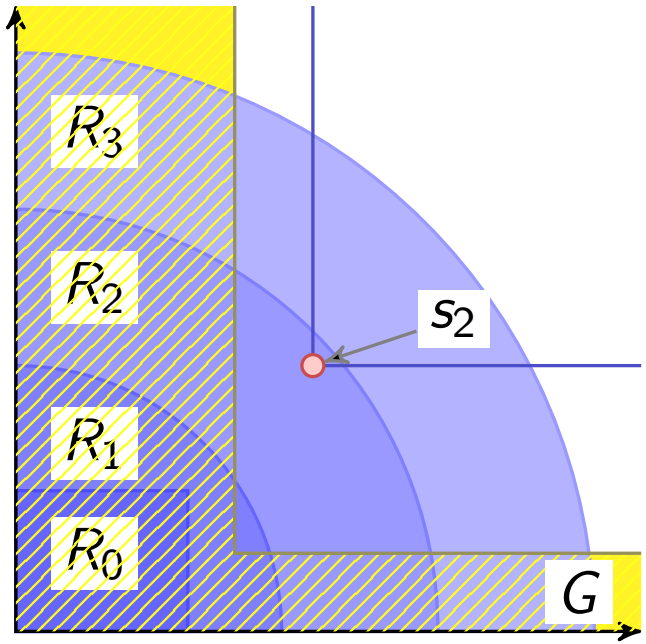
Spurious states can be removed via relative inductiveness



$i = 1$

- If $\text{minpre}(s_{i+1}\uparrow) \cap R_i = \varnothing$, overapproximation R_{i+1} is too coarse.
- Set $C := (s_{i+1}\uparrow)^C$.
  C is downward-closed,
  $R_0 \subseteq C$, C inductive relative to R_i.

Generalization gives better refinements



$i = 1$

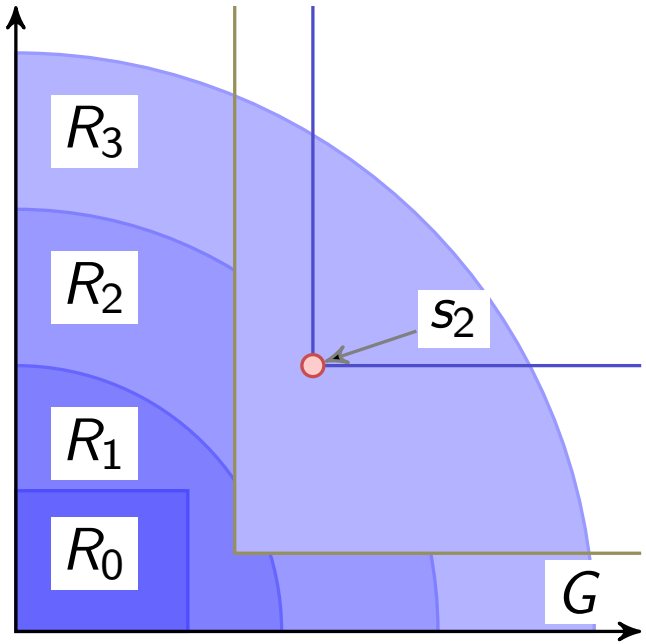
We can often find a smaller set G that

1. G is downward-closed,
2. $R_0 \subseteq G$,
3. G inductive relative to R_i,
4. $s_{i+1} \notin G$.

Refining with such a set rules out more spurious states.

The process is known as *generalization*.

Generalization gives better refinements



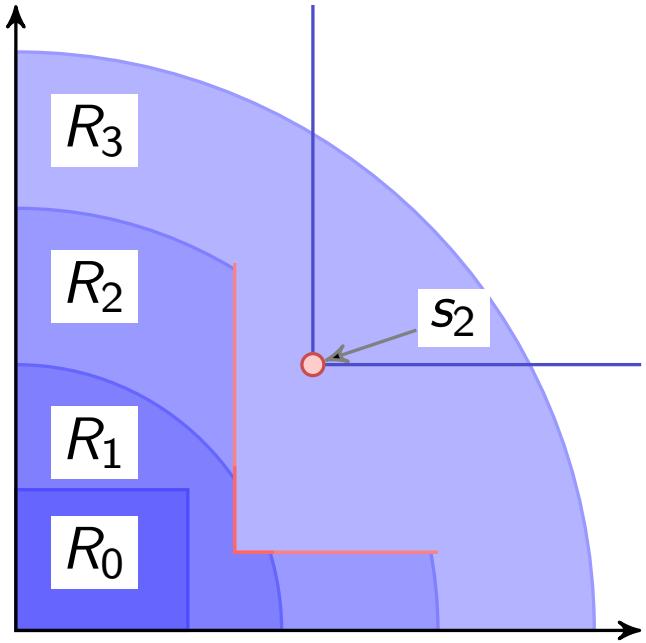
$i = 1$

We can often find a smaller set G that

1. G is downward-closed,
2. $R_0 \subseteq G$,
3. G inductive relative to R_i,
4. $s_{i+1} \notin G$.

Refining with such a set rules out more spurious states.
The process is known as *generalization*.

Generalization gives better refinements



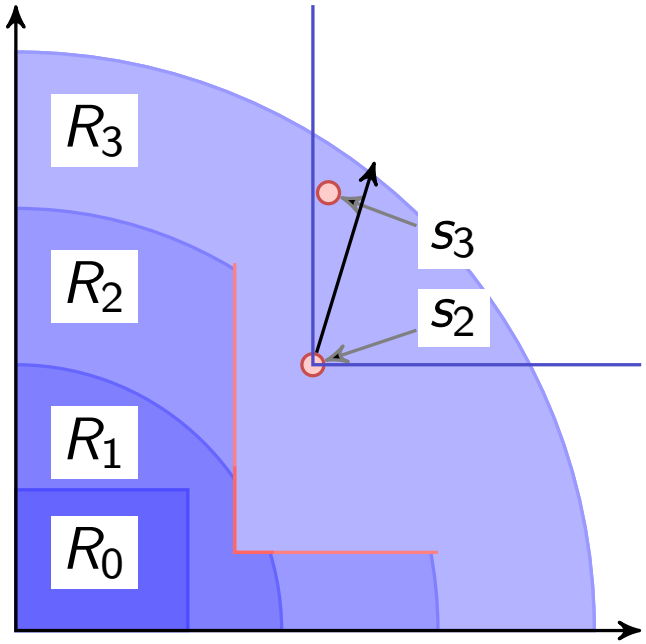
$i = 1$

We can often find a smaller set G that

1. G is downward-closed,
2. $R_0 \subseteq G$,
3. G inductive relative to R_i,
4. $s_{i+1} \notin G$.

Refining with such a set rules out more spurious states.
The process is known as *generalization*.

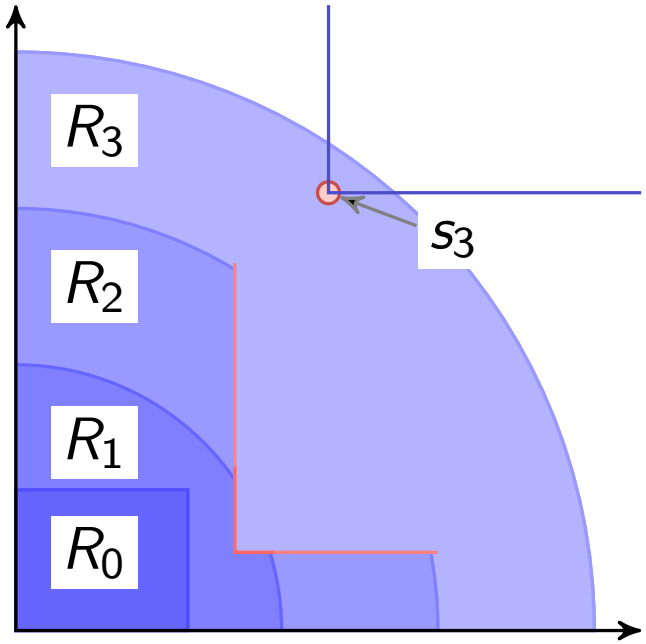
Completing the refinement phase



Repeat refinement steps as long as possible.

Choose a new bad state to start from, if required.

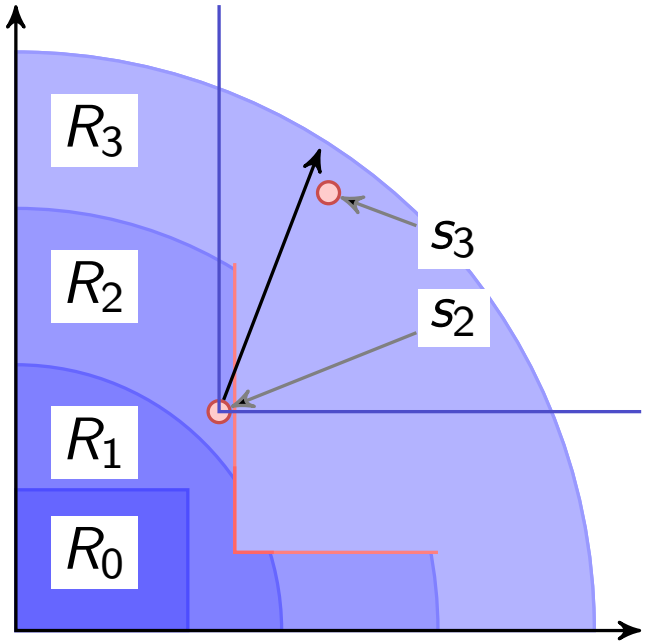
Completing the refinement phase



Repeat refinement steps as long as possible.

Choose a new bad state to start from, if required.

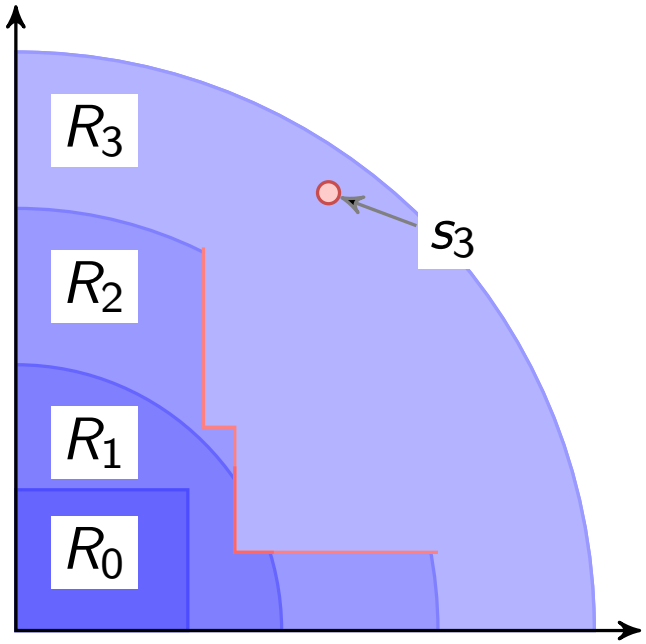
Completing the refinement phase



Repeat refinement steps as long as possible.

Choose a new bad state to start from, if required.

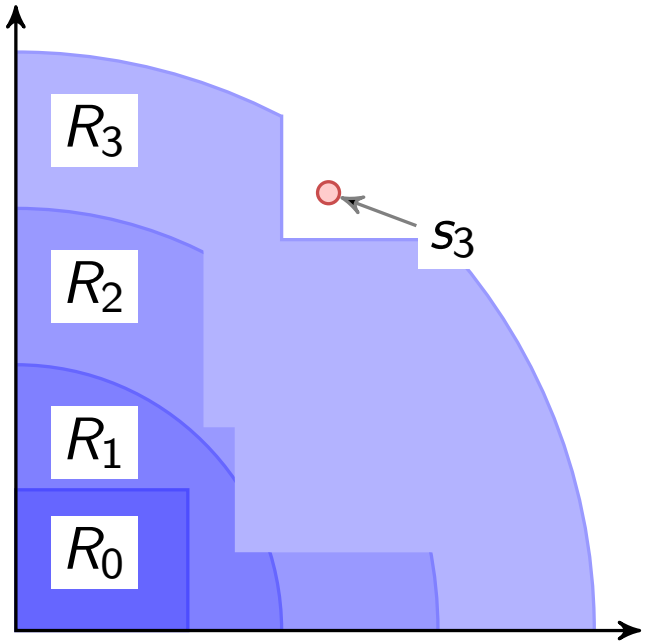
Completing the refinement phase



Repeat refinement steps as long as possible.

Choose a new bad state to start from, if required.

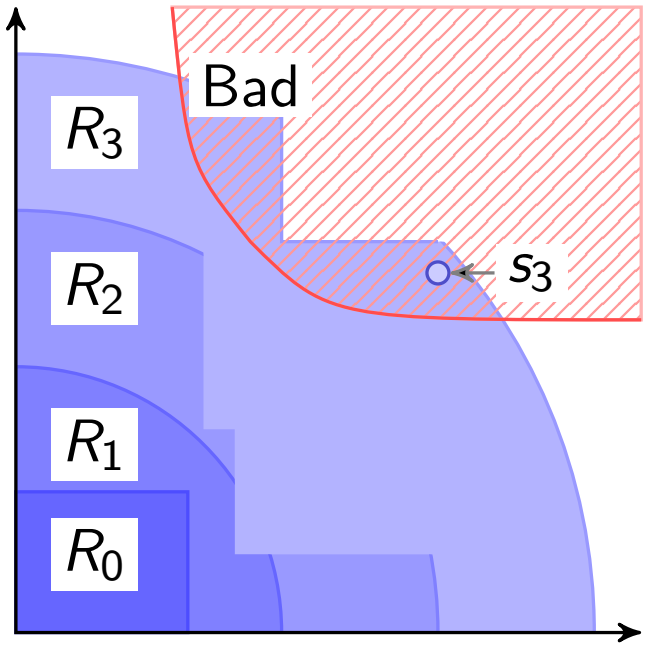
Completing the refinement phase



Repeat refinement steps as long as possible.

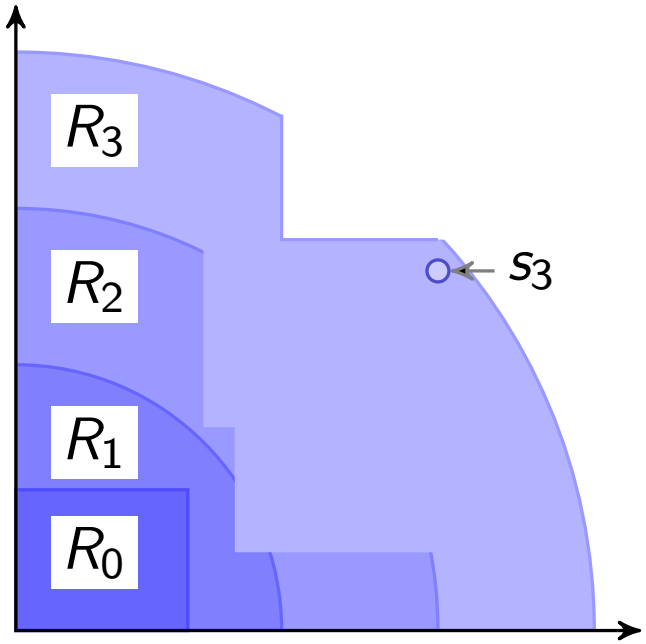
Choose a new bad state to start from, if required.

Completing the refinement phase



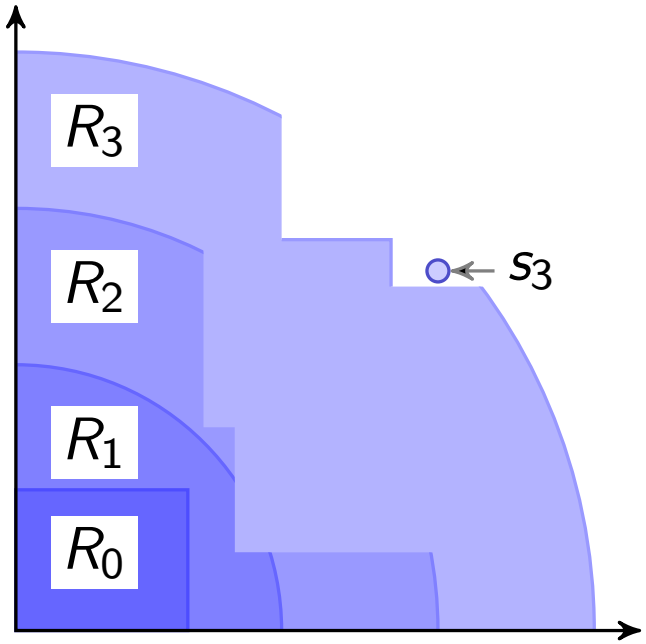
Repeat refinement steps as long as possible.
Choose a new bad state to start from, if required.

Completing the refinement phase



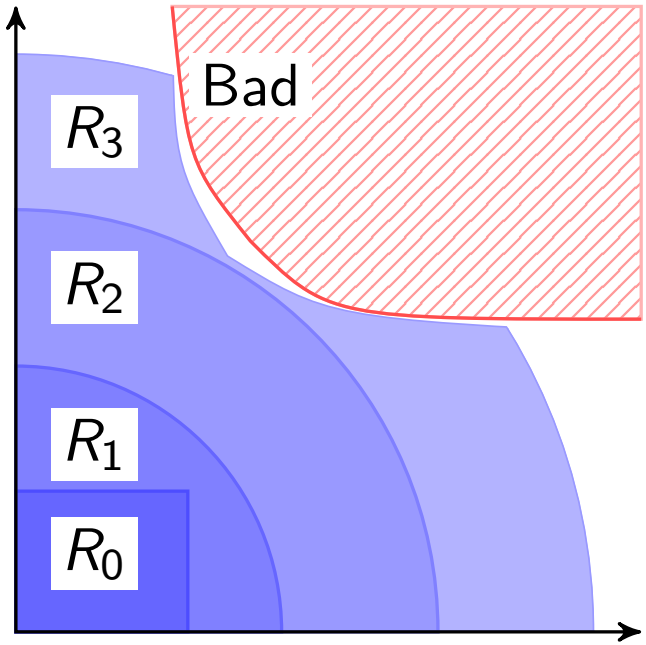
Repeat refinement steps as long as possible.
Choose a new bad state to start from, if required.

Completing the refinement phase



Repeat refinement steps as long as possible.
Choose a new bad state to start from, if required.

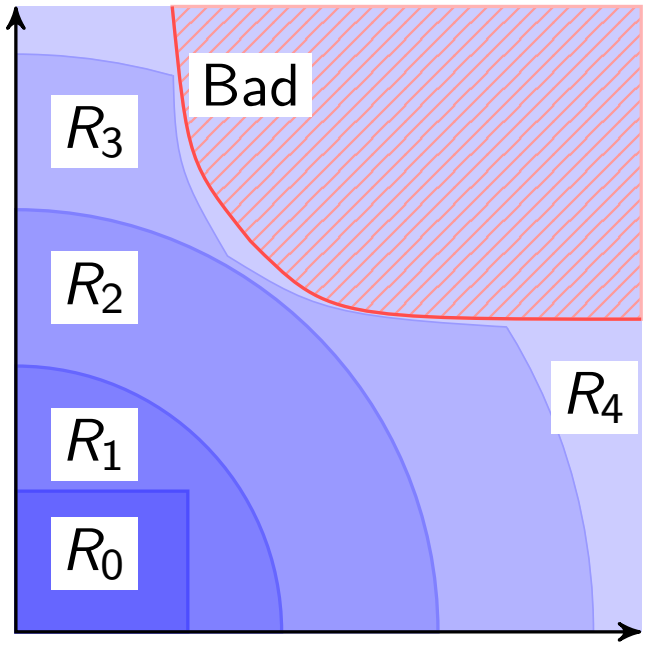
The second phase enlarges the approximation sequence



$N = 3$

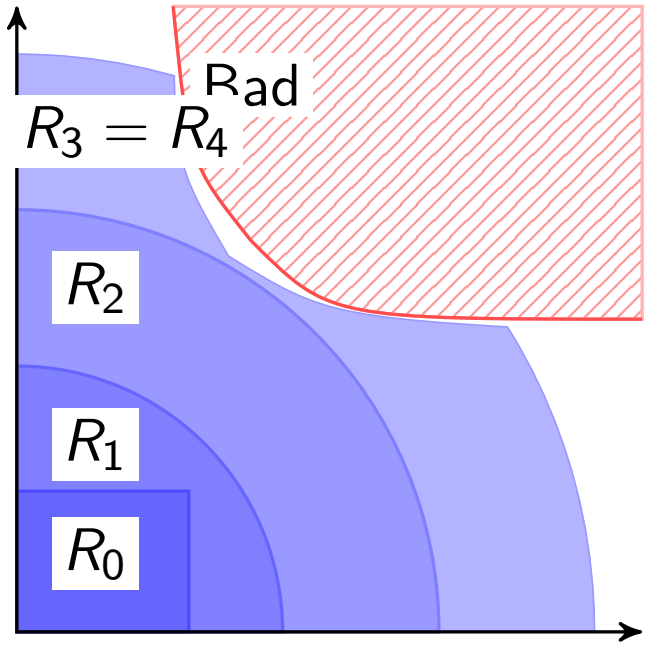
At the end of the first phase, R_N and Bad are disjoint.
Check if two R_i sets are equal.
If not, extend and repeat the refinement phase.
If $R_i = R_{i+1}$, an invariant has been found.

The second phase enlarges the approximation sequence



At the end of the first phase, R_N and Bad are disjoint.
Check if two R_i sets are equal.
If not, extend and repeat the refinement phase.
If $R_i = R_{i+1}$, an invariant has been found.

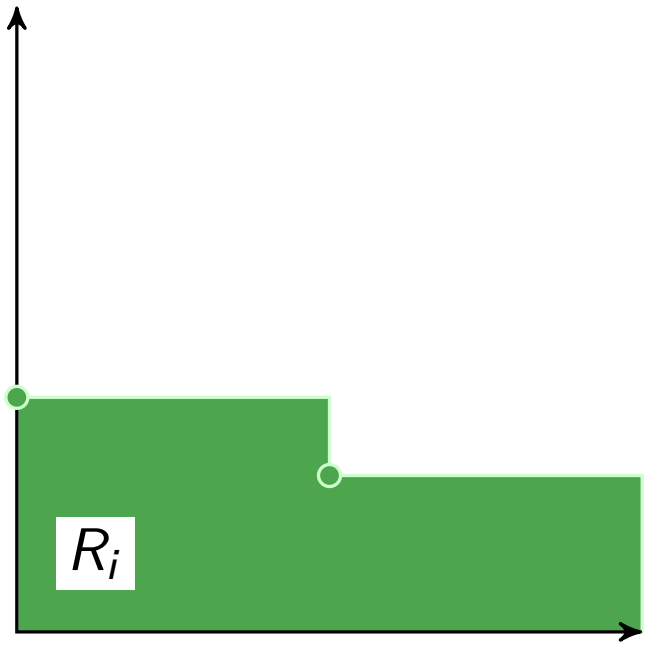
The second phase enlarges the approximation sequence



$i = 3$

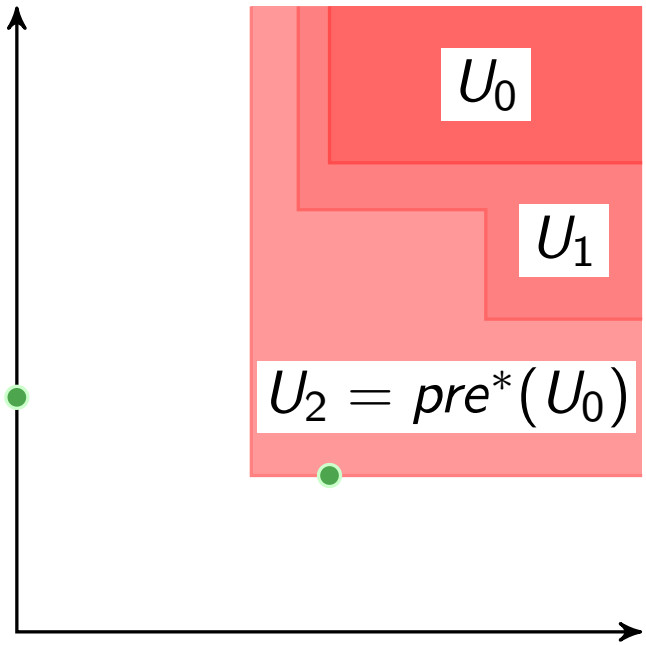
At the end of the first phase, R_N and Bad are disjoint.
Check if two R_i sets are equal.
If not, extend and repeat the refinement phase.
If $R_i = R_{i+1}$, an invariant has been found.

Termination



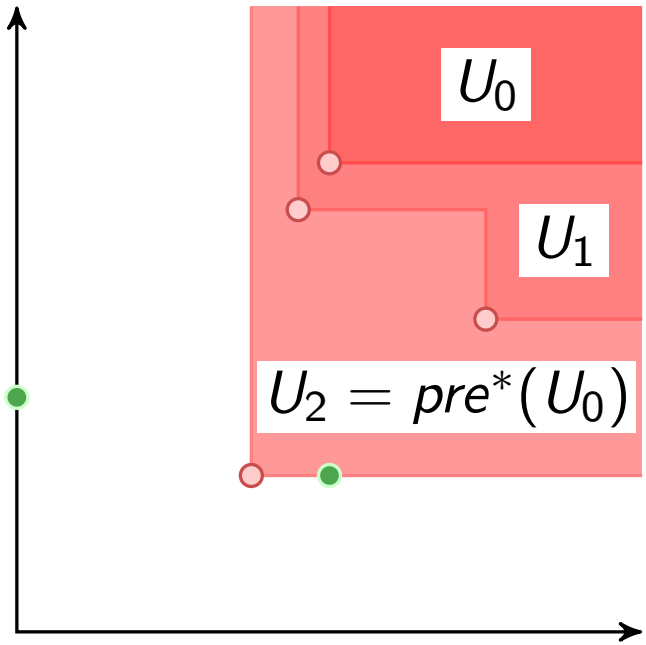
- Claim: There can only be finitely many different R_i.
- $R_i \overset{1:1}{\longleftrightarrow} \{r_{i,1}, \ldots, r_{i,k}\}\uparrow$
- $r_{i,j}$ generated by generalization.
- Relation to backward search: $M := \left(\bigcup_{i \geq 0} \min U_i\right)\downarrow$. For all $i, j : r_{i,j} \in M$.
- M is finite.

Termination



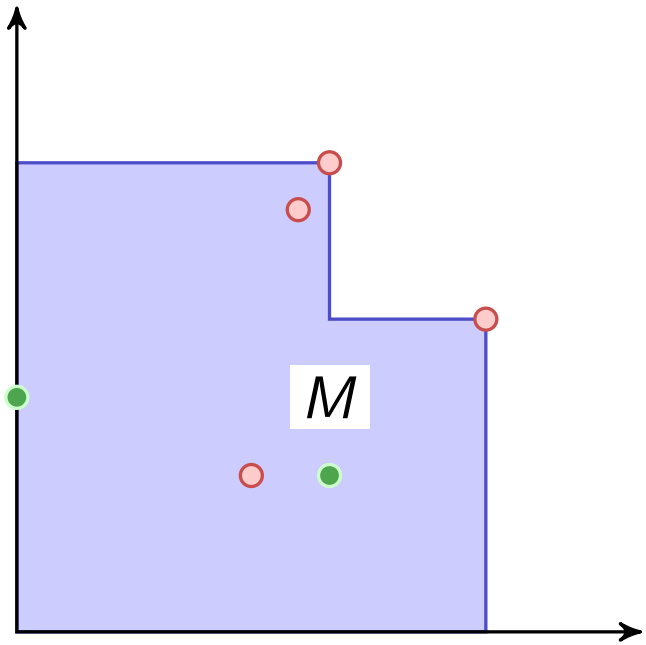
- Claim: There can only be finitely many different R_i.
- $R_i \overset{1:1}{\longleftrightarrow} \{r_{i,1}, \ldots, r_{i,k}\}\uparrow$
- $r_{i,j}$ generated by generalization.
- Relation to backward search:
  $M := \left(\bigcup_{i \geq 0} \min U_i\right)\downarrow$.
  For all $i, j : r_{i,j} \in M$.
- M is finite.

Termination



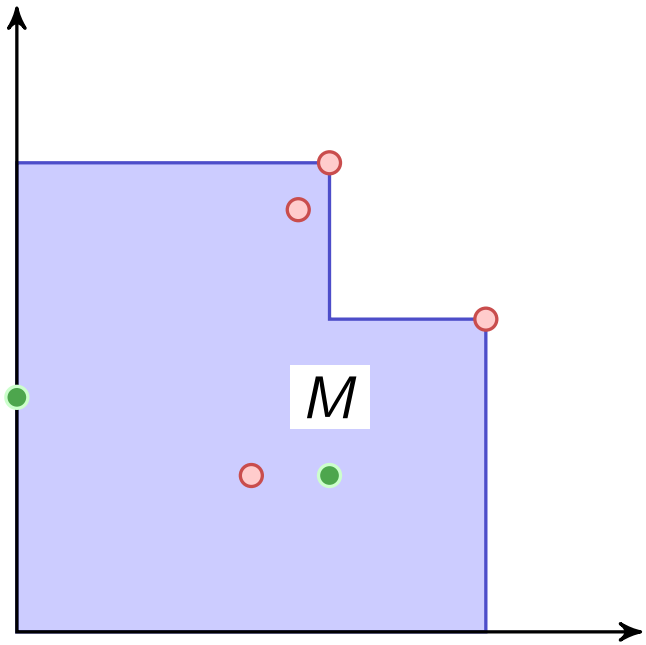
- Claim: There can only be finitely many different R_i.
- $R_i \overset{1:1}{\longleftrightarrow} \{r_{i,1}, \ldots, r_{i,k}\}\uparrow$
- $r_{i,j}$ generated by generalization.
- Relation to backward search:
  $M := \left(\bigcup_{i \geq 0} \min U_i\right)\downarrow$.
  For all $i, j : r_{i,j} \in M$.
- M is finite.

Termination



- Claim: There can only be finitely many different R_i.
- $R_i \overset{1:1}{\longleftrightarrow} \{r_{i,1}, \ldots, r_{i,k}\}\uparrow$
- $r_{i,j}$ generated by generalization.
- Relation to backward search: $M := \left(\bigcup_{i \geq 0} \min U_i\right)\downarrow$. For all $i, j : r_{i,j} \in M$.
- M is finite.

Termination



- Claim: There can only be finitely many different R_i.
- $R_i \overset{1:1}{\longleftrightarrow} \{r_{i,1}, \ldots, r_{i,k}\}\uparrow$
- $r_{i,j}$ generated by generalization.
- Relation to backward search: $M := \left(\bigcup_{i \geq 0} \min U_i\right)\downarrow$. For all $i, j : r_{i,j} \in M$.
- M is finite.

Experimental results

Problem Instance	IIC Time (s)	Backward Time (s)	EEC Time (s)	MCOV Time (s)
Uncoverable instances				
Bingham ($h = 250$)	**0.2**	Timeout	9.6	**0.2**
Ext. ReadWrite (small consts)	**0.0**	0.1	Timeout	Timeout
Ext. ReadWrite	**0.3**	216.3	Timeout	0.6
Mesh2x2	$<$ **0.1**	0.3	266.9	$<$ **0.1**
Mesh3x2	$<$ **0.1**	4.1	Timeout	$<$ **0.1**
Multipoll	1.5	0.5	21.8	$<$ **0.1**
MedAR1	**0.8**	8.77	Timeout	3.7
MedAR2	33.2	15.7	Timeout	**13.7**
MedAR5	128.1	26.6	Timeout	**12.9**
Coverable instances				
Kanban	$<$ **0.1**	804.7	Timeout	0.1
pncsacover	2.8	7.9	36.5	**1.0**
pncsasemiliv	0.1	0.2	32.1	$<$ **0.1**

Summary: Incremental, Inductive Coverability

- adapts Bradley's IC3 to solve the coverability problem for well-structured transition systems (including Petri nets)
- terminates for Petri nets, more generally: for downward-finite well-structured transition systems.
- has an efficient implementation for Petri nets that outperforms backward search and EEC.

Copyright etc.

These slides are based on the paper "Incremental, Inductive Coverability" by Johannes Kloos, Rupak Majumdar, Filip Niksic and Ruzica Piskac.