SMT-based Unbounded Model Checking with IC3 and Approximate QE

Cesare Tinelli

The University of Iowa

Acknowledgements

Joint work with Christoph Sticksel and Ruoyu Zhang

Modeling Computational Systems

Software or hardware systems can be often represented as a *state transition system* $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ where

- $\mathcal{S}$ is a set of *states*, the state space
- $\mathcal{I} \subseteq \mathcal{S}$ is a set of *initial states*
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{S}$ is a (right-total) *transition relation*
- $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{P}}$ is a *labeling function* where $\mathcal{P}$ is a set of *state predicates*

Typically, the state predicates denote variable-value pairs $x = v$

Model Checking

Software or hardware systems can be often represented as a state transition system $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$

$\mathcal{M}$ can be seen as a *model* both

1. in an engineering sense:

    an abstraction of the real system

   and

2. in a mathematical logic sense:

    a Kripke structure in some modal logic

Model Checking

The functional properties of a computational system can be expressed as *temporal* properties

- for a suitable model $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ of the system
- in a suitable temporal logic

Model Checking

The functional properties of a computational system can be expressed as *temporal* properties

- for a suitable model $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ of the system
- in a suitable temporal logic

Two main classes of properties:

- *Safety properties*: nothing bad ever happens
- *Liveness properties*: something good eventually happens

Invariance ~~Model~~ Checking

The functional properties of a computational system can be expressed as *temporal* properties

- for a suitable model $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ of the system
- in a suitable temporal logic

Two main classes of properties:

- *Safety properties*: nothing bad ever happens
- *Liveness properties*: something good eventually happens

Safety checking can be reduced to invariance checking

Basic Terminology

Let $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ be a transition system

The set $\mathcal{R}$ of *reachable states (of $\mathcal{M}$)* is the smallest subset of $\mathcal{S}$ such that

1. $\mathcal{I} \subseteq \mathcal{R}$ (initial states are reachable)
2. $(\mathcal{R} \bowtie \mathcal{T}) \subseteq \mathcal{R}$ ($\mathcal{T}$-successors of reachable states are reachable)

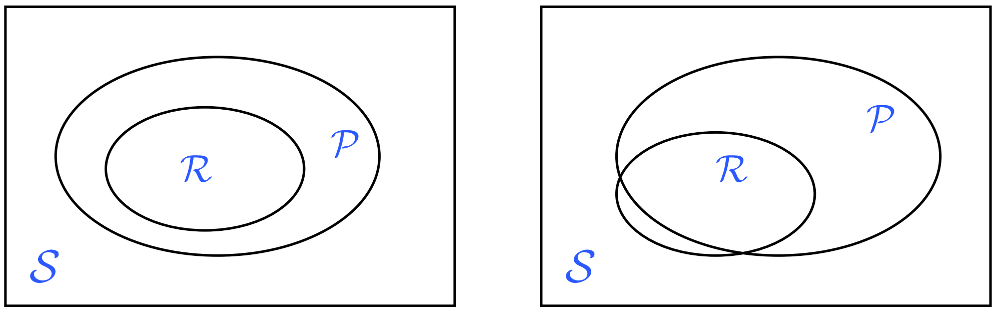
Basic Terminology

Let $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ be a transition system

The set $\mathcal{R}$ of *reachable states (of $\mathcal{M}$)* is the smallest subset of $\mathcal{S}$ such that

1. $\mathcal{I} \subseteq \mathcal{R}$ (initial states are reachable)
2. $(\mathcal{R} \bowtie \mathcal{T}) \subseteq \mathcal{R}$ ($\mathcal{T}$-successors of reachable states are reachable)

A state property $\mathcal{P} \subseteq \mathcal{S}$ is *invariant (for $\mathcal{M}$)* iff $\mathcal{R} \subseteq \mathcal{P}$



invariant not invariant

Checking Invariance

In principle, to check that $\mathcal{P}$ is invariant for $\mathcal{M}$ it suffices to

1. compute $\mathcal{R}$ and
2. check that $\mathcal{R} \subseteq \mathcal{P}$

Checking Invariance

In principle, to check that $\mathcal{P}$ is invariant for $\mathcal{M}$ it suffices to

1. compute $\mathcal{R}$ and
2. check that $\mathcal{R} \subseteq \mathcal{P}$

This can be done explicitly only if $\mathcal{S}$ is finite, and relatively small ($< 10M$ states)

Checking Invariance

In principle, to check that $\mathcal{P}$ is invariant for $\mathcal{M}$ it suffices to

1. compute $\mathcal{R}$ and
2. check that $\mathcal{R} \subseteq \mathcal{P}$

This can be done explicitly only if $\mathcal{S}$ is finite, and relatively small ($< 10M$ states)

Alternatively, we can represent $\mathcal{M}$ symbolically and use

- BDD-based methods, if $\mathcal{S}$ is finite,
- automata-based methods,
- abstract interpretation methods, or
- logic-based methods

Checking Invariance

In principle, to check that $\mathcal{P}$ is invariant for $\mathcal{M}$ it suffices to

1. compute $\mathcal{R}$ and
2. check that $\mathcal{R} \subseteq \mathcal{P}$

This can be done explicitly only if $\mathcal{S}$ is finite, and relatively small ($< 10M$ states)

Alternatively, we can represent $\mathcal{M}$ symbolically and use

- BDD-based methods, if $\mathcal{S}$ is finite,
- automata-based methods,
- abstract interpretation methods, or
- logic-based methods

Logic-based Model Checking

Applicable if we can encode

$$\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$$

in some classical logic $\mathbb{L}$ with decidable entailment $\models_{\mathbb{L}}$ for some large enough class of formulas in $\mathbb{L}$

($\varphi \models_{\mathbb{L}} \psi$ iff $\varphi \wedge \neg\psi$ is unsatisfiable in $\mathbb{L}$)

Logic-based Model Checking

Applicable if we can encode

$$\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$$

in some classical logic $\mathbb{L}$ with decidable entailment $\models_{\mathbb{L}}$ for some large enough class of formulas in $\mathbb{L}$

($\varphi \models_{\mathbb{L}} \psi$ iff $\varphi \wedge \neg\psi$ is unsatisfiable in $\mathbb{L}$)

Some (reasonable) additional requirements on $\mathbb{L}$ are needed

Requirements on $\mathbb{L}$

$\mathbb{L} = (\Sigma, \mathbf{F}, \mathcal{A}, \models_{\mathbb{L}}, \mathbf{V})$ with

- Σ, a many-sorted first-order signature with equality
- $\mathbf{F}$, language of Σ-formulas closed under all Boolean operators and quantifiers
- $\mathcal{A}$, a single Σ-structure with decidable satisfiability for quantifier-free formulas

Requirements on $\mathbb{L}$

$\mathbb{L} = (\Sigma, \mathbf{F}, \mathcal{A}, \models_{\mathbb{L}}, \mathbf{V})$ with

- $\models_{\mathbb{L}}$, same as entailment in $\mathcal{A}$
- $\mathbf{V}$, set of *values* in $\mathcal{A}$,
  variable-free terms with unique interpretation in $\mathcal{A}$
- Quantifier-free formulas satisfied by values:

  for all qffs $F[\mathbf{x}] \in \mathbf{F}$ satisfiable in $\mathcal{A}$,
  there is a $\mathbf{v} \in \mathbf{V}$ such that
  $F[\mathbf{v}]$ is true in $\mathcal{A}$

Examples of $\mathbb{L}$

Any modular combination of the logics of

- Boolean formulas (with variables belonging to a single Boolean sort)
- linear integer, rational or floating point arithmetic
- fixed size bit vectors
- algebraic data types
- strings
- finite sets

... with a suitable choice of function and predicate symbols

Logical encodings of transitions systems

$\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ X: set of *variables* $\mathbf{V}$: *values* in $\mathbb{L}$

Not.: if $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{s} = (v_1, \ldots, v_n)$, $\phi[\mathbf{s}] := \phi[v_1/x_1, \ldots, v_n/x_n]$

Logical encodings of transitions systems

$\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ X: set of *variables* $\mathbf{V}$: *values* in $\mathbb{L}$

Not.: if $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{s} = (v_1, \ldots, v_n)$, $\phi[\mathbf{s}] := \phi[v_1/x_1, \ldots, v_n/x_n]$

- states $\mathbf{s} \in \mathcal{S}$ encoded as n-tuples of $\mathbf{V}^n$

Logical encodings of transitions systems

$\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ X: set of *variables* $\mathbf{V}$: *values* in $\mathbb{L}$

Not.: if $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{s} = (v_1, \ldots, v_n)$, $\phi[\mathbf{s}] := \phi[v_1/x_1, \ldots, v_n/x_n]$

- states $\mathbf{s} \in \mathcal{S}$ encoded as n-tuples of $\mathbf{V}^n$
- $\mathcal{I}$ encoded as a formula $I[\mathbf{x}]$ with free variables $\mathbf{x}$ such that

$$\mathbf{s} \in \mathcal{I} \text{ iff } \models_{\mathbb{L}} I[\mathbf{s}]$$

Logical encodings of transitions systems

$\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ $\quad X$: set of *variables* $\quad \mathbf{V}$: *values* in $\mathbb{L}$

Not.: if $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{s} = (v_1, \ldots, v_n)$, $\phi[\mathbf{s}] := \phi[v_1/x_1, \ldots, v_n/x_n]$

- states $\mathbf{s} \in \mathcal{S}$ encoded as n-tuples of $\mathbf{V}^n$
- $\mathcal{I}$ encoded as a formula $I[\mathbf{x}]$ with free variables $\mathbf{x}$ such that

$$\mathbf{s} \in \mathcal{I} \text{ iff } \models_{\mathbb{L}} I[\mathbf{s}]$$

- $\mathcal{T}$ encoded as a formula $T[\mathbf{x}, \mathbf{x}']$ such that

$$\models_{\mathbb{L}} T[\mathbf{s}, \mathbf{s}'] \text{ for all } (\mathbf{s}, \mathbf{s}') \in \mathcal{T}$$

Logical encodings of transitions systems

$\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ X: set of *variables* $\mathbf{V}$: *values* in $\mathbb{L}$

Not.: if $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{s} = (v_1, \ldots, v_n)$, $\phi[\mathbf{s}] := \phi[v_1/x_1, \ldots, v_n/x_n]$

- states $\mathbf{s} \in \mathcal{S}$ encoded as n-tuples of $\mathbf{V}^n$
- $\mathcal{I}$ encoded as a formula $I[\mathbf{x}]$ with free variables $\mathbf{x}$ such that

$$\mathbf{s} \in \mathcal{I} \text{ iff } \models_{\mathbb{L}} I[\mathbf{s}]$$

- $\mathcal{T}$ encoded as a formula $T[\mathbf{x}, \mathbf{x}']$ such that

$$\models_{\mathbb{L}} T[\mathbf{s}, \mathbf{s}'] \text{ for all } (\mathbf{s}, \mathbf{s}') \in \mathcal{T}$$

- State properties encoded as formulas $P[\mathbf{x}]$

Strongest Inductive Invariant

The *strongest inductive invariant (for $\mathcal{M}$ in $\mathbb{L}$)* is a formula $R[\mathbf{x}]$ such that $\models_{\mathbb{L}} R[\mathbf{s}]$ iff $\mathbf{s} \in \mathcal{R}$

Strongest Inductive Invariant

The *strongest inductive invariant (for $\mathcal{M}$ in $\mathbb{L}$)* is a formula $R[\mathbf{x}]$ such that $\models_{\mathbb{L}} R[\mathbf{s}]$ iff $\mathbf{s} \in \mathcal{R}$

Suppose we can compute R from I and T. Then,

checking that a property $P[\mathbf{x}]$ is invariant for $\mathcal{M}$ reduces to checking that $R[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$

Strongest Inductive Invariant

The *strongest inductive invariant (for $\mathcal{M}$ in $\mathbb{L}$)* is a formula $R[\mathbf{x}]$ such that $\models_{\mathbb{L}} R[\mathbf{s}]$ iff $\mathbf{s} \in \mathcal{R}$

Problem: R may be very expensive or impossible to compute, or not even representable in $\mathbb{L}$

Strongest Inductive Invariant

The *strongest inductive invariant (for $\mathcal{M}$ in $\mathbb{L}$)* is a formula $R[\mathbf{x}]$ such that $\models_{\mathbb{L}} R[\mathbf{s}]$ iff $\mathbf{s} \in \mathcal{R}$

Problem: R may be very expensive or impossible to compute, or not even representable in $\mathbb{L}$

One Strategy: *Property-Directed Reachability*. Try to construct an over-approximation $\hat{R}$ of R that entails P in $\mathbb{L}$

Property Directed Reachability

Two main methods:

- Interpolation-based model checking [McMillan'03]
- Incremental Construction of
  Inductive Clauses for
  Indubitable Correctness (IC3) [Bradley'10]

Property Directed Reachability

Two main methods:

- Interpolation-based model checking [McMillan'03]
- Incremental Construction of
  Inductive Clauses for
  Indubitable Correctness (IC3) [Bradley'10]

Property Directed Reachability

Two main methods:

- Interpolation-based model checking [McMillan'03]
- Incremental Construction of
  Inductive Clauses for
  Indubitable Correctness (IC3) [Bradley'10]

Note: PDR is used typically to refer to IC3

IC3's Main Idea

Given $M = (I[\mathbf{x}], T[\mathbf{x}, \mathbf{x}'])$ and $P[\mathbf{x}]$, construct $\hat{R}$ incrementally

Maintain list $\hat{R}_0\ \hat{R}_1\ \cdots\ \hat{R}_k\ \hat{R}_{k+1}$ where

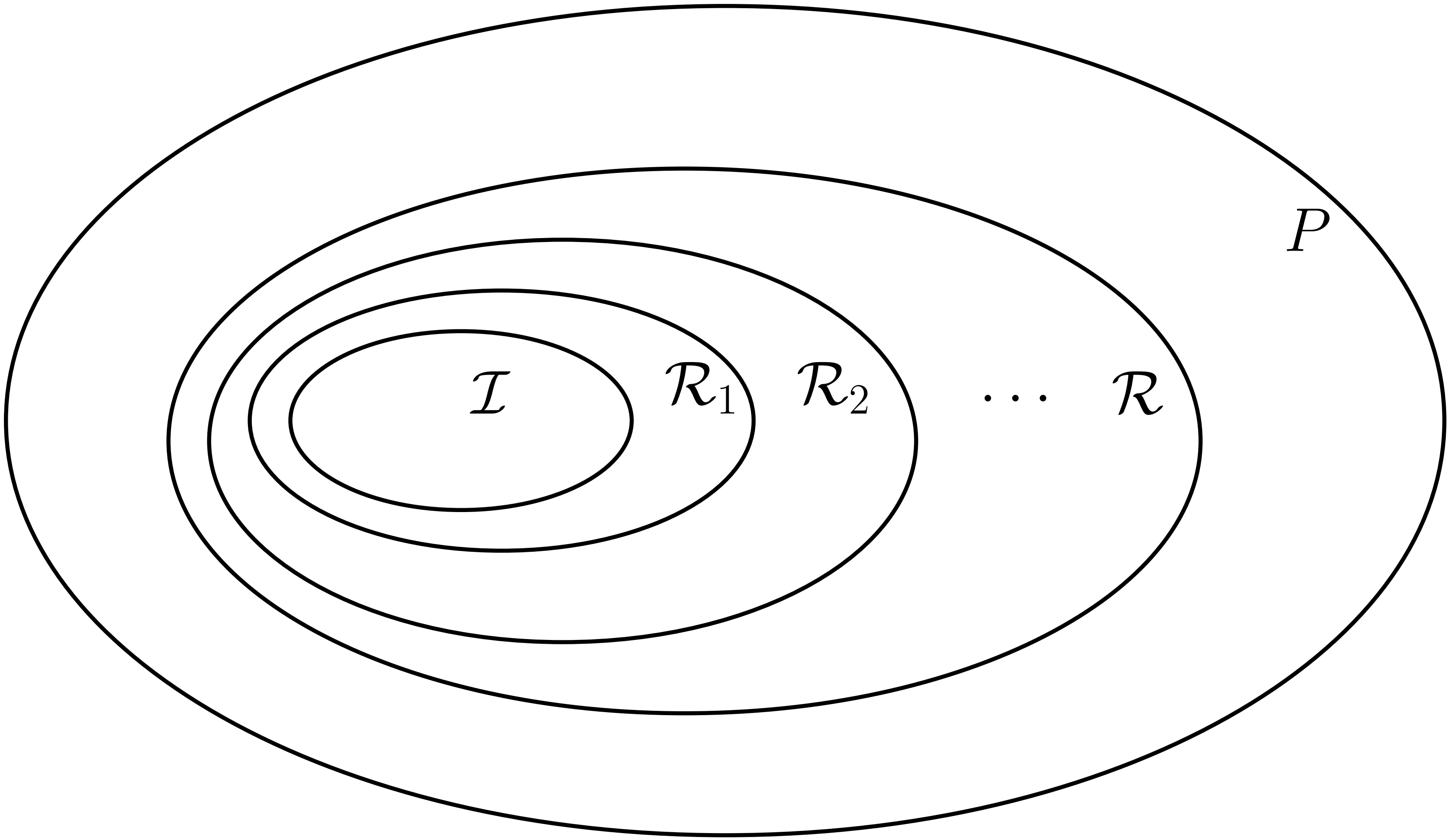
- $$\begin{array}{lcl} \hat{R}_0 & = & \{I[\mathbf{x}]\} \\ \hat{R}_{k+1} & = & \{P[\mathbf{x}]\} \end{array}$$
- for each $i = 1, \ldots, k$

  $\hat{R}_i$ is a set of one-state formulas over $\mathbf{x}$

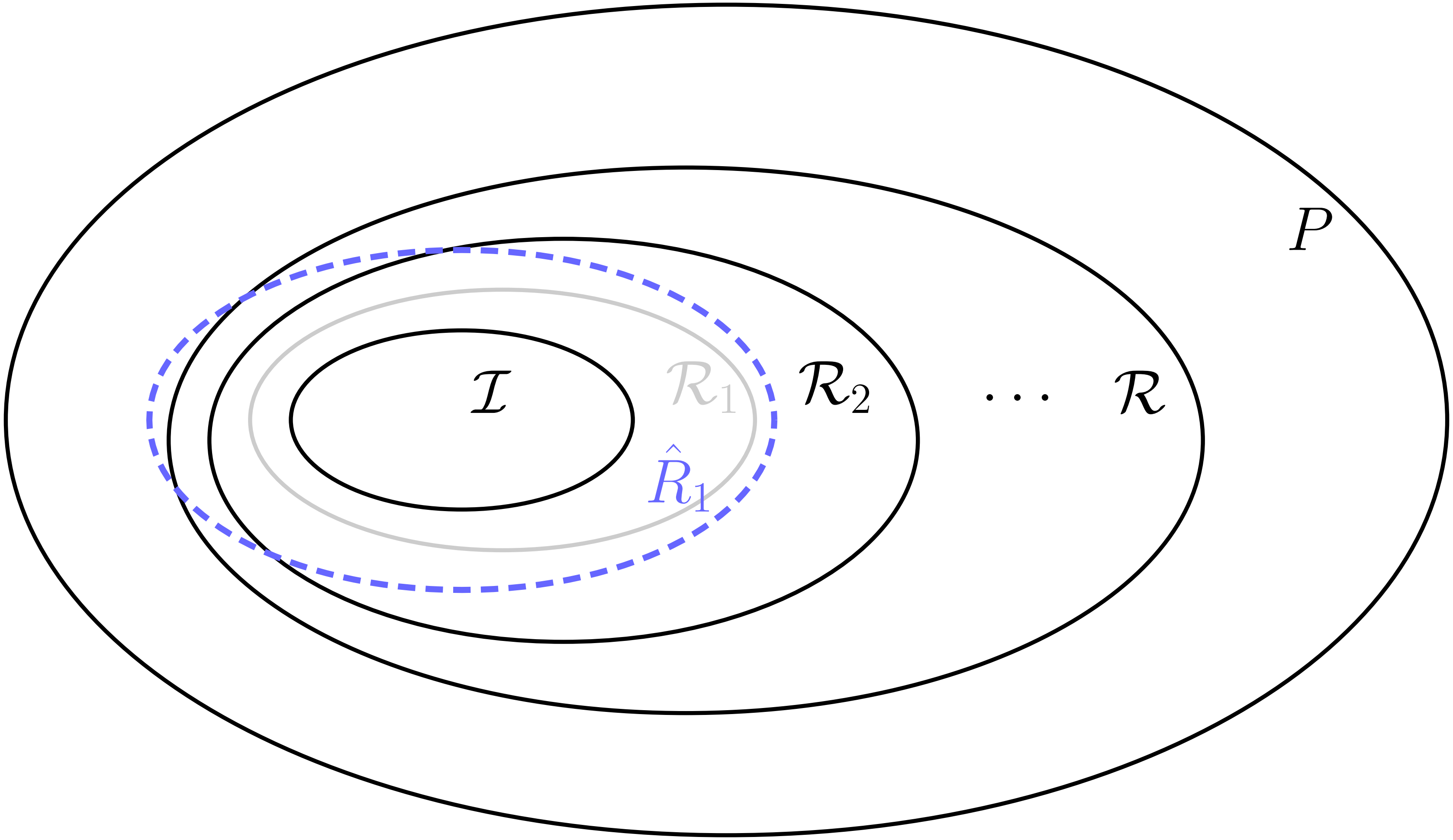
  $\hat{R}_i$ over-approximates the states reachable in i-steps

  $\hat{R}_i$ under-approximates $\hat{R}_{i+1}$

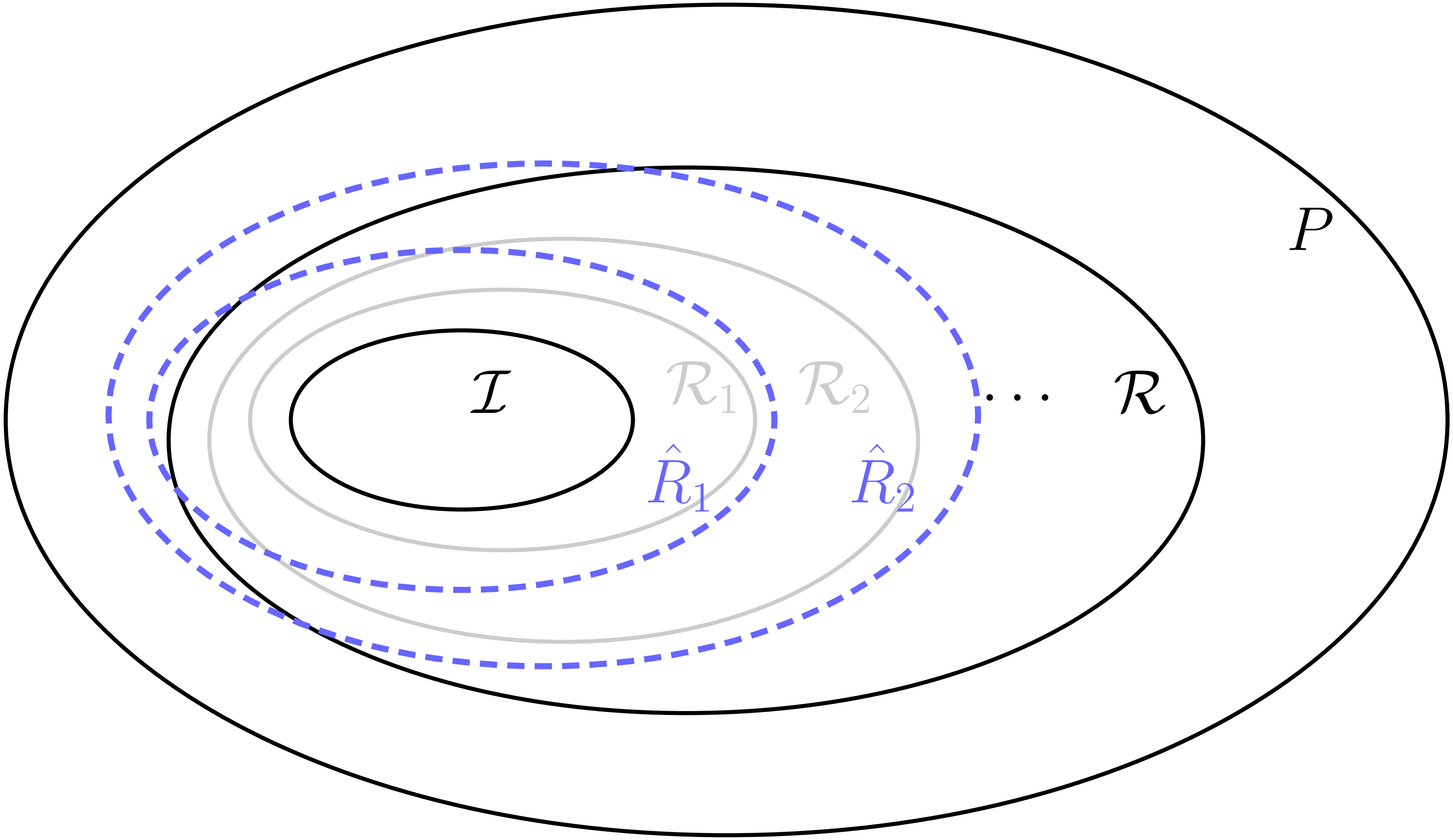
IC3's Main Idea



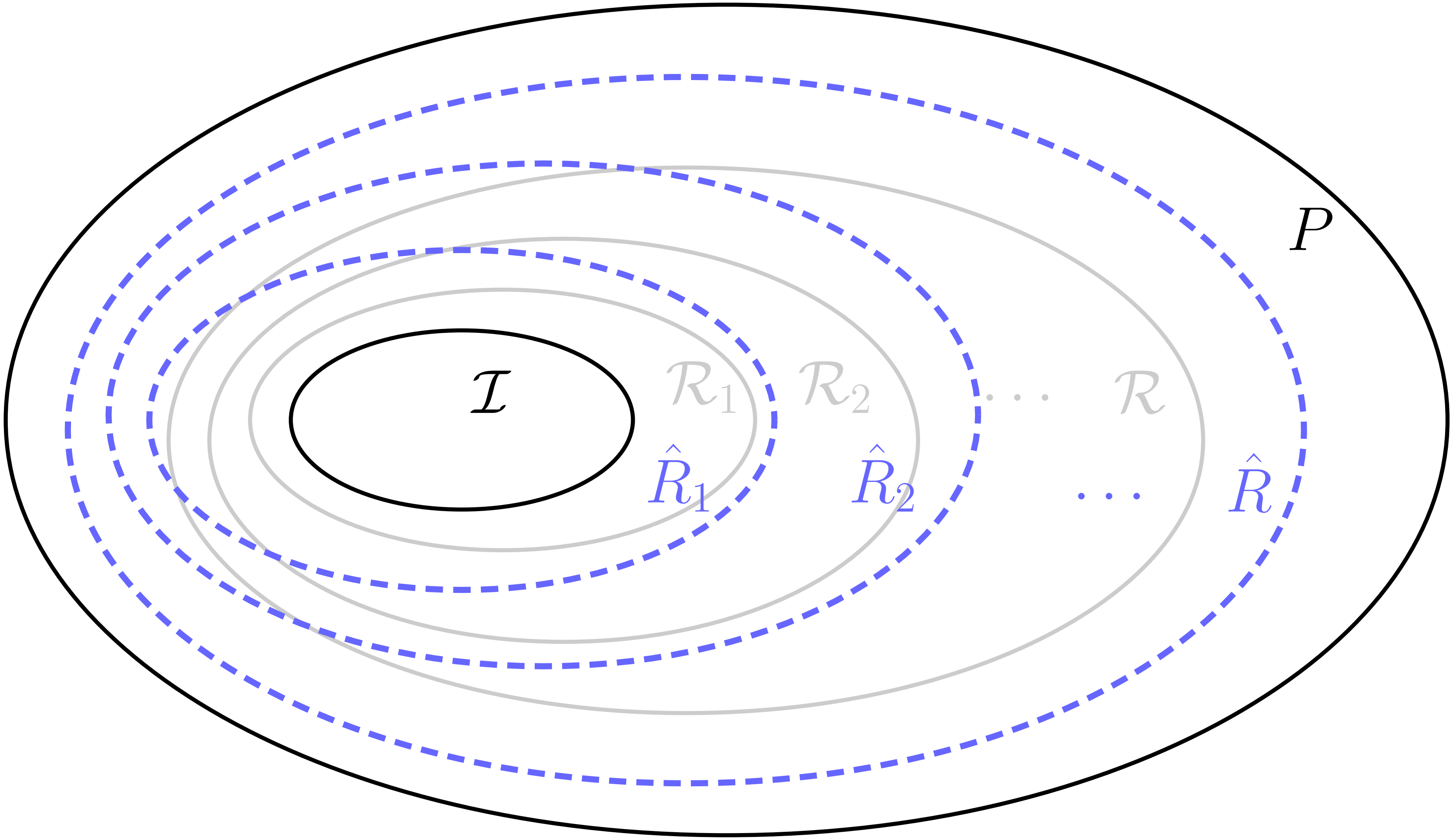
IC3's Main Idea



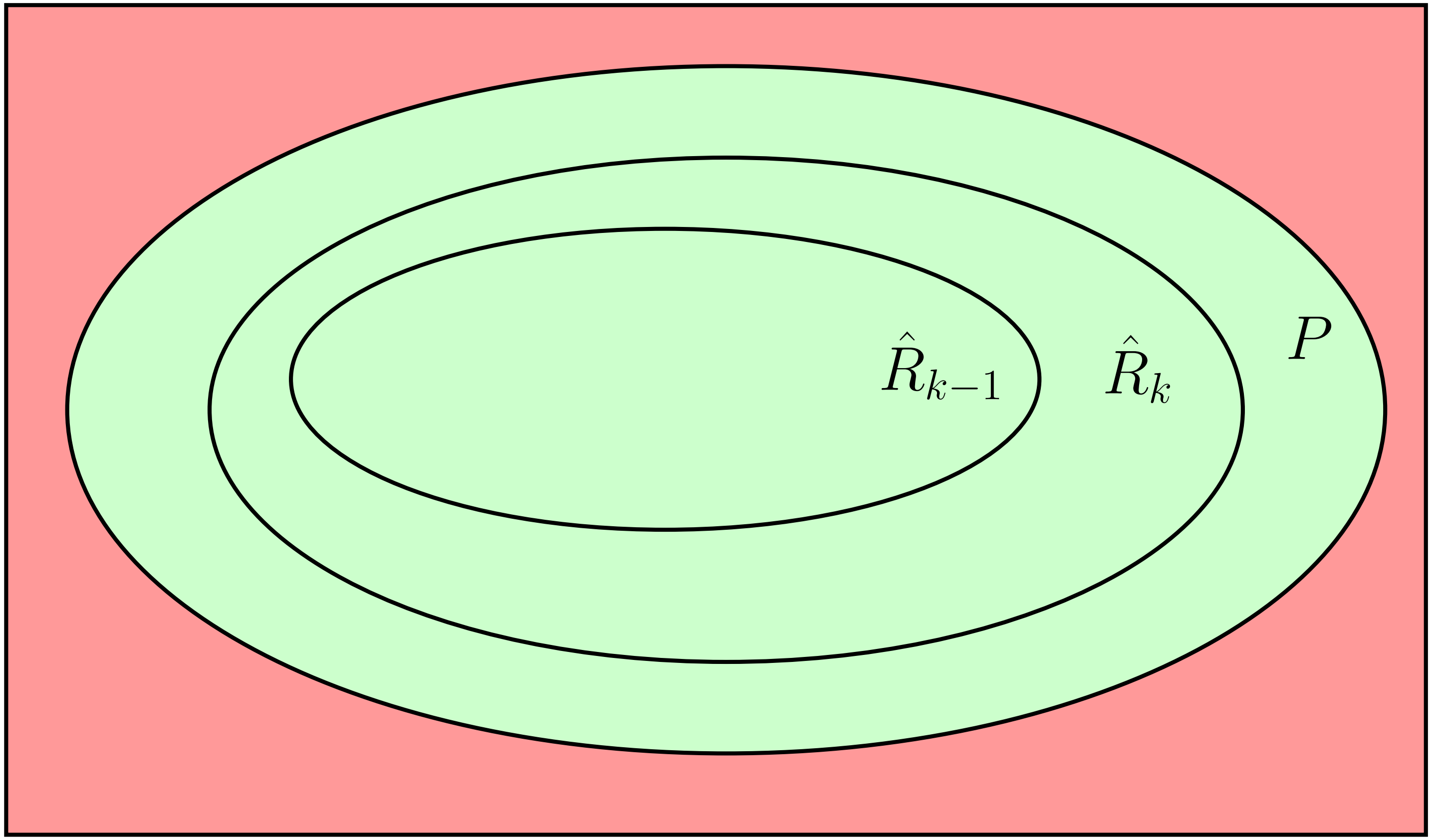
IC3's Main Idea



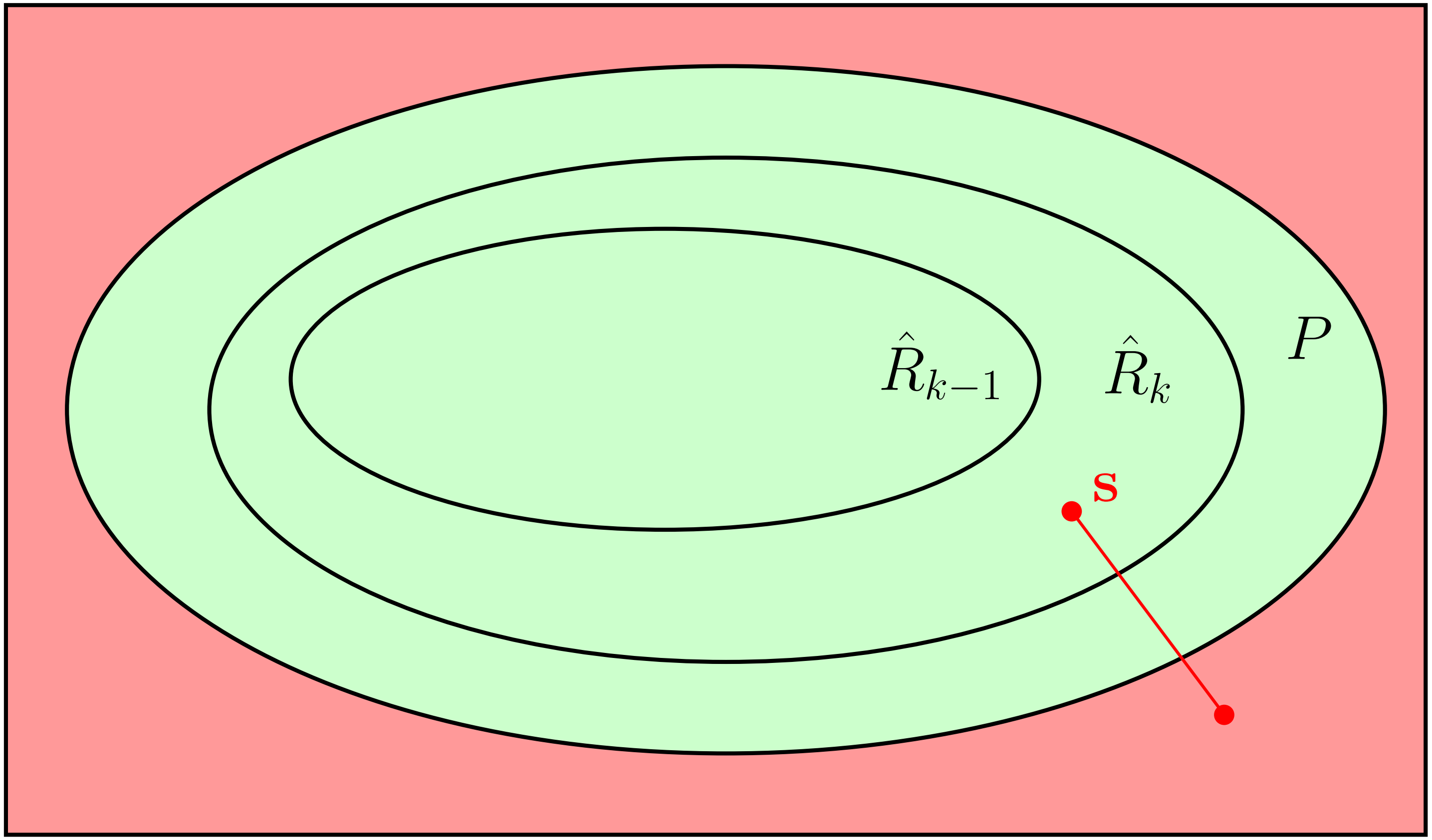
IC3's Main Idea



Recursively Refining $\hat{R}_i$

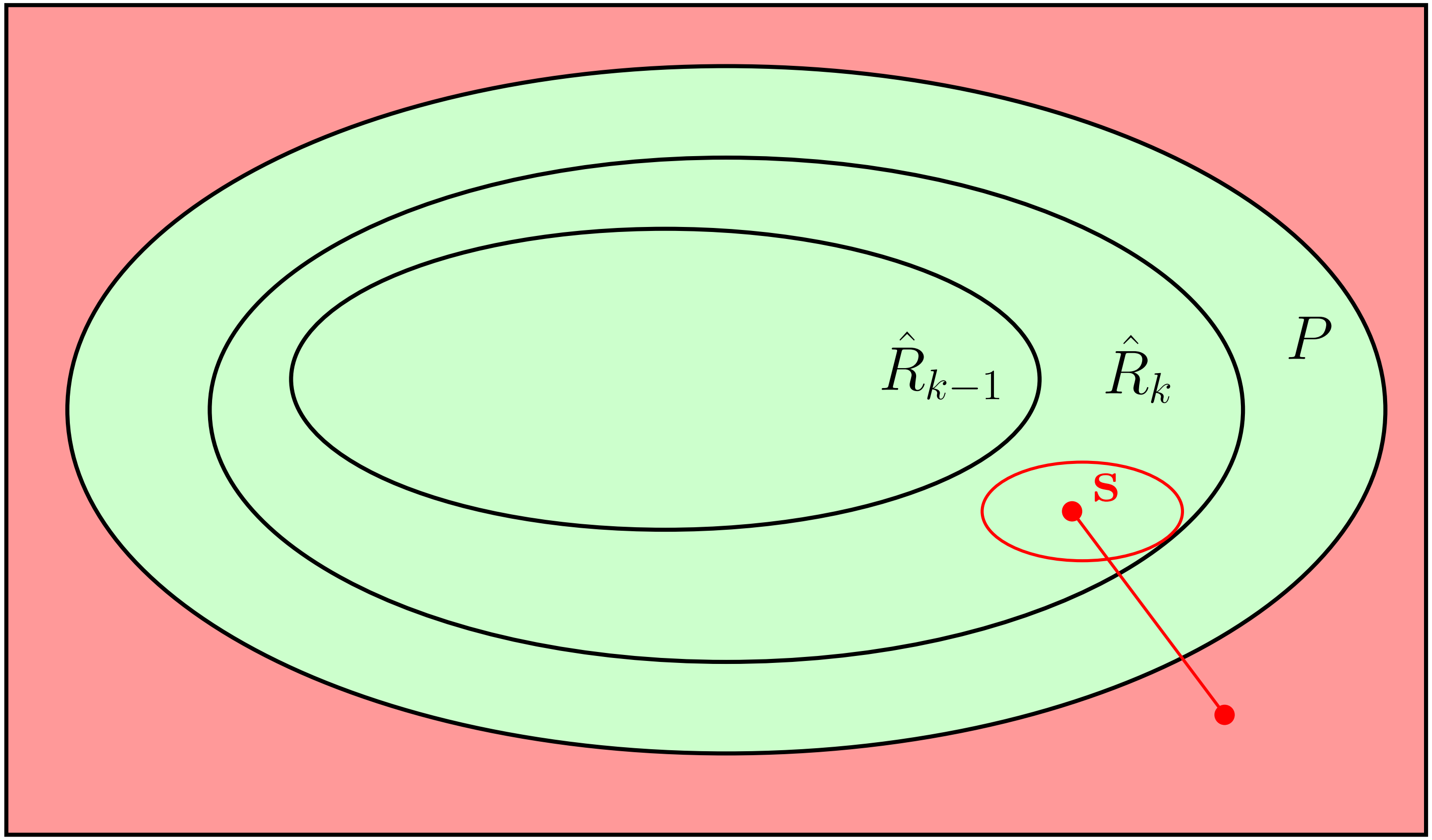


Recursively Refining $\hat{R}_i$



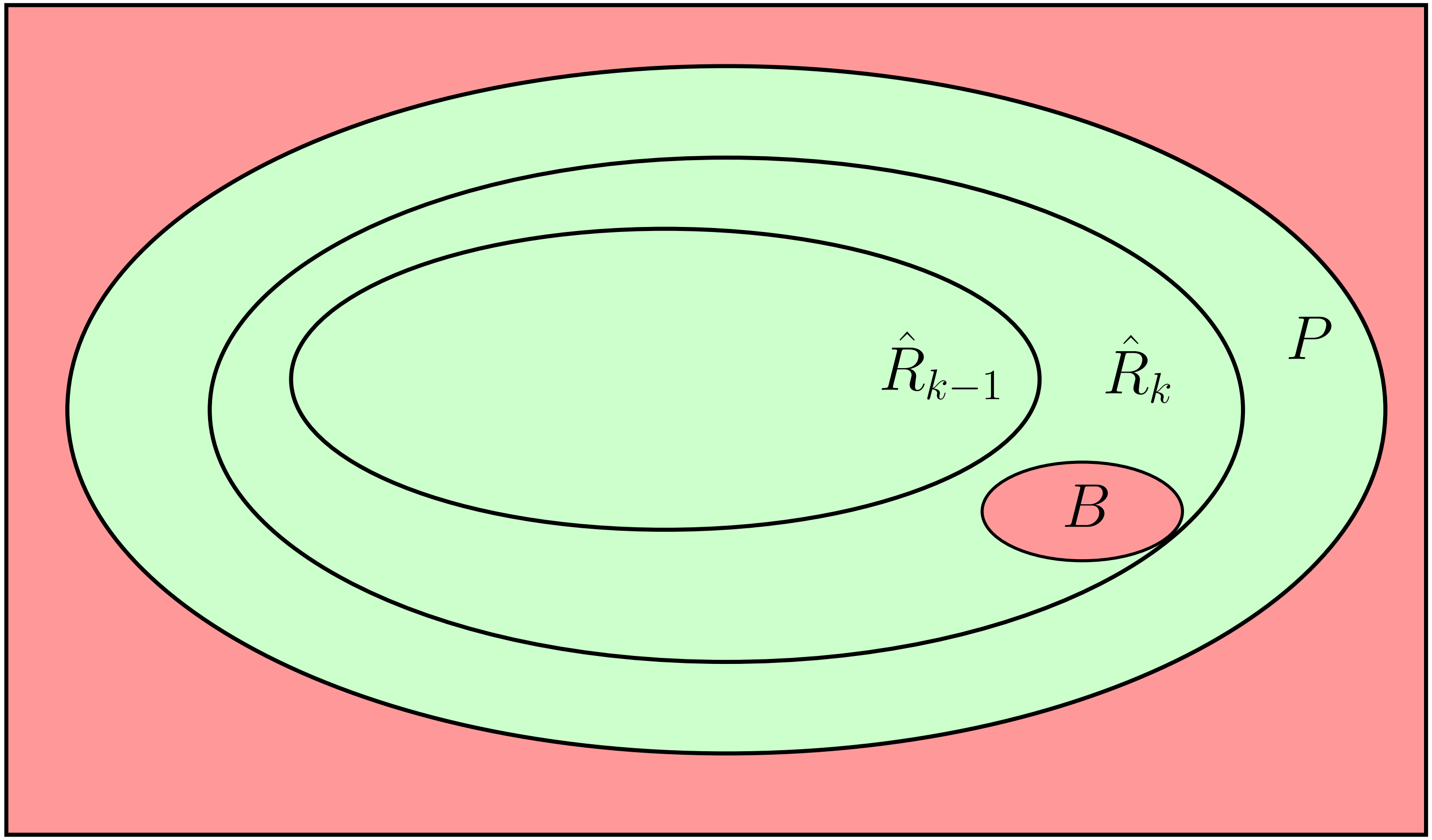
Suppose there are $\mathbf{s}, \mathbf{s}'$ s.t. $\hat{R}_k[\mathbf{s}] \wedge T[\mathbf{s}, \mathbf{s}'] \wedge \neg P[\mathbf{s}']$ is satisfiable

Recursively Refining $\hat{R}_i$



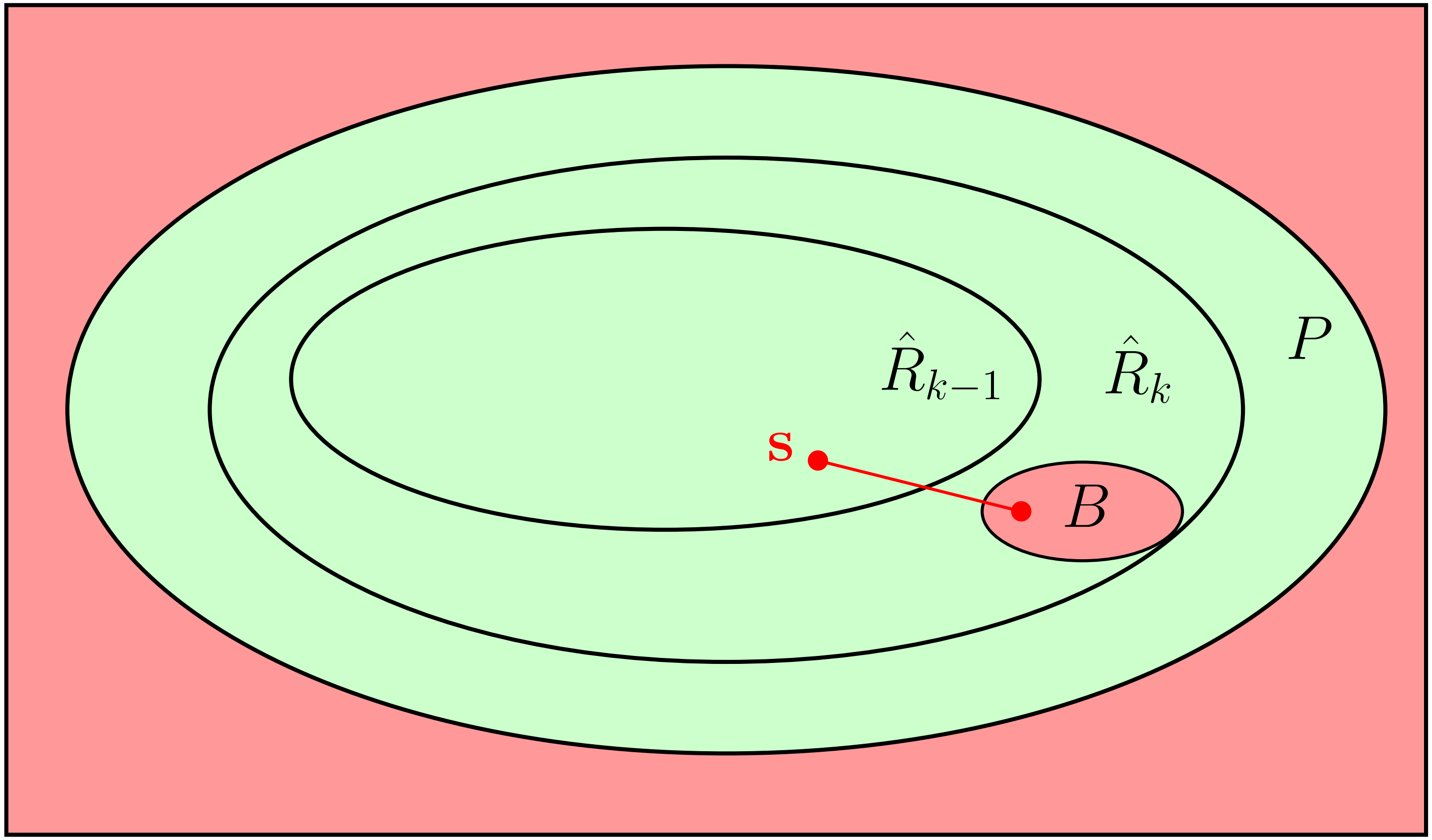
Find $B[\mathbf{x}]$ s.t. $B[\mathbf{s}]$ is satisfiable and $B[\mathbf{x}],\ T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} \neg P[\mathbf{x}']$

Recursively Refining $\hat{R}_i$



If $\hat{R}_{k-1}[\mathbf{x}], T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} \neg B[\mathbf{x}']$ let $\hat{R}_k := \hat{R}_{k-1} \cup \{\neg B\}$

Recursively Refining $\hat{R}_i$



Else there are $\mathbf{s}, \mathbf{s}'$ s.t. $\hat{R}_{k-1}[\mathbf{s}] \wedge T[\mathbf{s}, \mathbf{s}'] \wedge \neg P[\mathbf{s}']$ is satisfiable.
Refine $\hat{R}_{k-1}$

Frame Sequences

IC3 constructs (initial segments of) sequences $(R_i)_{i \geq 0}$ of *frames*, sets of one-state formulas, satisfying the following

Frame Conditions

(1) $R_0 = \{I\}$

(2) $R_i \supseteq R_{i+1}$ for all $i > 0$

(3) $R_i \supseteq \{P\}$ for all $i > 0$

(4) $R_i[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_{i+1}[\mathbf{x}']$ for all $i \geq 0$

Extension of a Formula

The *extension* of an m-state formula $F[\mathbf{y}_1, \ldots, \mathbf{y}_m]$ of $\mathbb{L}$ is the following subset of $\mathcal{S}^m$:

$$[\![F]\!] \stackrel{\mathrm{def}}{=} \{(\mathbf{s}_1, \ldots, \mathbf{s}_m) \in \mathcal{S}^m \mid F[\mathbf{s}_1, \ldots, \mathbf{s}_m] \text{ is satisfiable in } \mathbb{L}\}$$

Note: I will sometimes identify a state formula F with its extension $[\![F]\!]$

Properties of Frame Sequences

Frame Conditions

(1) $R_0 = I$

(2) $R_i \supseteq R_{i+1}$ for all $i > 0$

(3) $R_i \supseteq \{P\}$ for all $i > 0$

(4) $R_i[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_{i+1}[\mathbf{x}']$ for all $i \geq 0$

Properties of Frame Sequences

Frame Conditions

(1) $R_0 = I$

(2) $R_i \supseteq R_{i+1}$ for all $i > 0$

(3) $R_i \supseteq \{P\}$ for all $i > 0$

(4) $R_i[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_{i+1}[\mathbf{x}']$ for all $i \geq 0$

Lemma 1 [Soundness] Suppose $(R_i)_{i \geq 0}$ satisfies the frame conditions and $R_0[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$. If there is an $i > 0$ such that $R_i = R_{i+1}$, then P is invariant.

Properties of Frame Sequences

Frame Conditions

(1) $R_0 = I$

(2) $R_i \supseteq R_{i+1}$ for all $i > 0$

(3) $R_i \supseteq \{P\}$ for all $i > 0$

(4) $R_i[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_{i+1}[\mathbf{x}']$ for all $i \geq 0$

Lemma 2 [Termination, non-invariant case] If P is not invariant, there is a $k \geq 0$ such that for all frame sequences $(R_i)_{i \geq 0}$ satisfying the frame conditions, $[\![R_k]\!]$ contains a k-reachable error state.

Properties of Frame Sequences

Frame Conditions

(1) $R_0 = I$

(2) $R_i \supseteq R_{i+1}$ for all $i > 0$

(3) $R_i \supseteq \{P\}$ for all $i > 0$

(4) $R_i[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_{i+1}[\mathbf{x}']$ for all $i \geq 0$

Lemma 2 [Termination, non-invariant case] If P is not invariant, there is a $k \geq 0$ such that for all frame sequences $(R_i)_{i \geq 0}$ satisfying the frame conditions, $[\![R_k]\!]$ contains a k-reachable error state.

Lemma 3 [Termination, invariant case] If $[\![P]\!]$ is finite, there is no frame sequence $(R_i)_{i \geq 0}$ satisfying the frame conditions such that $[\![R_i]\!] \subsetneq [\![R_{i+1}]\!]$ for all $i \geq 0$.

The IC3 Procedure: Our Version

Defined by *verify*$(R_0\, R_1)$

where

$R_0 = \{I\}$, $R_1 = \{P\}$

$I[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$

$I[\mathbf{x}],\, T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$

Require: $R_{i-1}[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_i[\mathbf{x}']$
for $i = 1, \ldots, k$ with $R_k = P$

1: **function** *verify*$(R_0 \cdots R_k)$
2: **let** $R_0 \cdots R_k =$ *strengthen*$(R_0 \cdots R_k)$ **in**
3: **let** $R_0 \cdots R_k =$ *propagate*$(R_0,\ R_1 \cdots R_k)$ **in**
4: *verify*$(R_0 \cdots R_k\, \{P\})$

Backward Pass

Require: $R_{i-1}[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_i[\mathbf{x}']$ for $i = 1, \ldots, k$

Ensure: $R_{i-1}[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_i[\mathbf{x}']$ for $i = 1, \ldots, k+1$
with $R_{k+1} = \{P\}$

function *strengthen*$(R_0 \cdots R_k)$
  if $R_k[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$ **then**
    $R_0 \cdots R_k$
  else
    let $B =$ *generalize*$(R_k, \neg P)$ **in**
    let $R_0 \cdots R_k =$ *block*$(R_0 \cdots R_{k-1}, (\{B\}, R_k))$ **in**
    strengthen$(R_0 \cdots R_k)$

Not. $A :: R$ denotes $\{A\} \cup R$

Blocking Bad States (simplified)

Require: $\mathbf{v} \in [\![R_i]\!]$, $\mathbf{v}$ reaches $\neg P$ in $k-i+1$ steps
for each $\mathbf{v} \in [\![B]\!]$, $B \in Q_j$, $i = j, \ldots, k$

Invariant: $R_{i-1}[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_i[\mathbf{x}']$ for $i = 1, \ldots, k$

function *block*$(R_0 \cdots R_{j-1},\ (Q_j, R_j) \cdots (Q_k, R_k))$
    let $B \in Q_j,\ Q_j = Q_j \setminus \{B\}$ **in**
    if $\neg B[\mathbf{x}] \wedge R_{j-1}[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} \neg B[\mathbf{x}']$ **then**
        let $R_0 \cdots R_k = R_0\,(\neg B :: R_1) \cdots (\neg B :: R_j)\,R_{j+1} \cdots R_k$ **in**
        if $Q_j \neq \emptyset$ **then**
            block$(R_0 \cdots R_{j-1},\ (Q_j, R_j)(B :: Q_{j+1}, R_{j+1}) \cdots (B :: Q_k, R_k))$
        else if $j = k$ **then** $R_0 \cdots R_k$
        else *block*$(R_0 \cdots R_j,\ (B :: Q_{j+1}, R_{j+1}) \cdots (B :: Q_k, R_k))$
    else
        let $\bar{B} =$ *generalize*$(R_{j-1} \wedge C_j,\ B)$ **in**
        block$(R_0 \cdots R_{j-2},\ (\{\bar{B}\}, R_{j-1})\,(Q_j, R_j) \cdots (Q_k, R_k))$

The IC3 Procedure

Require: $0 \leq j < k$

Invariant: $R_{i-1}[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} R_i[\mathbf{x}']$ for $i = 1, \ldots, k$

function *propagate*$(R_0 \cdots R_j,\ R_{j+1} \cdots R_k)$
  if $\left(\begin{array}{l} \text{there is } C \in R_j \setminus R_{j+1} \text{ s.t.} \\ R_j[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} C[\mathbf{x}'] \end{array} \right)$ **then**
    propagate$(R_0 \cdots R_j,\ (C :: R_{j+1}) \cdots R_k)$
  else if $R_j = R_{j+1}$ **then**
    raise Success
  else if $j + 1 < k$ **then**
    propagate$(R_0 \cdots R_{j+1},\ R_{j+2} \cdots R_k)$
  else
    $R_0 \cdots R_k$

The IC3 Procedure

Require: $[\![F[\mathbf{x}] \land T[\mathbf{x},\mathbf{x}'] \land B[\mathbf{x}']]\!] \neq \emptyset$

function *generalize*$(F,\ B)$

  let $(\mathrm{s},\mathrm{s}') \in [\![F[\mathbf{x}] \land T[\mathbf{x},\mathbf{x}'] \land B[\mathbf{x}']]\!]$ **in**

  let $\bar{B}[\mathbf{x}] =$ *extrapolate*$(\mathrm{s},\ \mathrm{s}',\ F[\mathbf{x}] \land T[\mathbf{x},\mathbf{x}'] \land B[\mathbf{x}'])$ **in**

  if $I[\mathbf{x}],\ \bar{B}[\mathbf{x}]\ \models_{\mathbb{L}}\ \bot$ **then**

    $\bar{B}[\mathbf{x}]$

  else

    raise Counterexample

Key Point of non-Boolean IC3

The critical component in generalizing IC3 beyond propositional logic is *extrapolate*

extrapolate encapsulates IC3's idea of generalizing *induction counterexamples*

Producing lemmas that eliminate whole sets of induction counterexamples is crucial for refining the frame sequence

Eliminating these states one by one is either impractical or even impossible

It is imperative to find a finite number of lemmas that eliminate all induction counterexamples from a frame

Generalizing Induction Conterexamples

The set of all induction counterexamples in a frame F wrt bad states B' has an exact and compact representation:

$$G[\mathbf{x}] := \exists \mathbf{x}'(F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B[\mathbf{x}'])$$

Generalizing Induction Conterexamples

The set of all induction counterexamples in a frame F wrt bad states B' has an exact and compact representation:

$$G[\mathbf{x}] := \exists \mathbf{x}' (F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B[\mathbf{x}'])$$

This formula typically cannot be used because it is not quantifier-free

Generalizing Induction Conterexamples

The set of all induction counterexamples in a frame F wrt bad states B' has an exact and compact representation:

$$G[\mathbf{x}] := \exists \mathbf{x}' (F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B[\mathbf{x}'])$$

This formula typically cannot be used because it is not quantifier-free

Quantifier elimination, when possible at all, can be very expensive (e.g., doubly exponential in the length of x')

Generalizing Induction Conterexamples

The set of all induction counterexamples in a frame F wrt bad states B' has an exact and compact representation:

$$G[\mathbf{x}] := \exists \mathbf{x}' (F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B[\mathbf{x}'])$$

This formula typically cannot be used because it is not quantifier-free

Quantifier elimination, when possible at all, can be very expensive (e.g., doubly exponential in the length of x')

Our approach: compute quantifier-free under-approximations of G driven by specific counterexamples

Our Approach

Additional requirement: $\mathbb{L}$ has quantifier elimination

Our Approach

Additional requirement: $\mathbb{L}$ has quantifier elimination

Given $E[\mathbf{x}, \mathbf{x}'] := F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B'[\mathbf{x}']$ and $(\mathbf{s}, \mathbf{s}') \in [\![E]\!]$,

Our Approach

Additional requirement: $\mathbb{L}$ has quantifier elimination

Given $E[\mathbf{x}, \mathbf{x}'] := F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B'[\mathbf{x}']$ and $(\mathbf{s}, \mathbf{s}') \in [\![E]\!]$,

Step 1 Extract from E a conjunction $H[\mathbf{x}, \mathbf{x}']$ of literals s.t.

$(\mathbf{s}, \mathbf{s}') \in [\![H]\!]$ and $H[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B'[\mathbf{x}']$

Our Approach

Additional requirement: $\mathbb{L}$ has quantifier elimination

Given $E[\mathbf{x}, \mathbf{x}'] := F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B'[\mathbf{x}']$ and $(\mathbf{s}, \mathbf{s}') \in [\![E]\!]$,

Step 1 Extract from E a conjunction $H[\mathbf{x}, \mathbf{x}']$ of literals s.t.

$$(\mathbf{s}, \mathbf{s}') \in [\![H]\!] \quad \text{and} \quad H[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B'[\mathbf{x}']$$

Step 2 Compute a conjuction $B[\mathbf{x}]$ of literals s.t.

$$\mathbf{s} \in [\![B]\!] \quad \text{and} \quad B[\mathbf{x}] \models_{\mathbb{L}} \exists \mathbf{x}'\, H[\mathbf{x}, \mathbf{x}']$$

Extracting a Conjunctive Implicant

$$H[\mathbf{x}, \mathbf{x}'] := e^+(F[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge B'[\mathbf{x}'])$$

where

$$e^+(F) := \begin{cases} e^+(F_1) & \text{if } F = F_1 \vee \cdots \vee F_n \text{ and } \models_{\mathbb{L}} \ F_1[\mathbf{s}, \mathbf{s}'] \\ e^+(F_1) \wedge \cdots \wedge e^+(F_n) & \text{if } F = F_1 \wedge \cdots \wedge F_n \\ e^-(F_1) & \text{if } F = \neg F_1 \\ F & \text{if } F \text{ is an atom} \end{cases}$$

$$e^-(F) := \begin{cases} e^-(F_1) \wedge \cdots \wedge e^-(F_n) & \text{if } F = F_1 \vee \cdots \vee F_n \\ e^-(F_1) & \text{if } F = F_1 \wedge \cdots \wedge F_n \text{ and } \models_{\mathbb{L}} \ \neg F_1[\mathbf{s}, \mathbf{s}'] \\ e^+(F_1) & \text{if } (3)\text{if } F = \neg F_1 \\ \neg F & \text{if } (4)\text{if } F \text{ is an atom} \end{cases}$$

Computing One-state Cube

Use a under-approximating version of QE to compute $B[\mathbf{x}]$ from $H[\mathbf{x}, \mathbf{x}']$

Currently done for linear integer arithmetic

Based on Cooper's QE procedure for LIA

Idea applies similarly to other logics with QE (e.g., real arithmetic)

Experimental Evaluation

Implementation in Kind 2 model checker with $\mathbb{L} = \text{LIA}$

Kind 2 is written in OCaml and uses several SMT solvers as reasoning engines

Used Z3 in this case (as it has does QE)

Step 2 of *extrapolate* can be configured to use either

- our approximate QE for LIA or
- precise QE provided by Z3

Experimental Evaluation

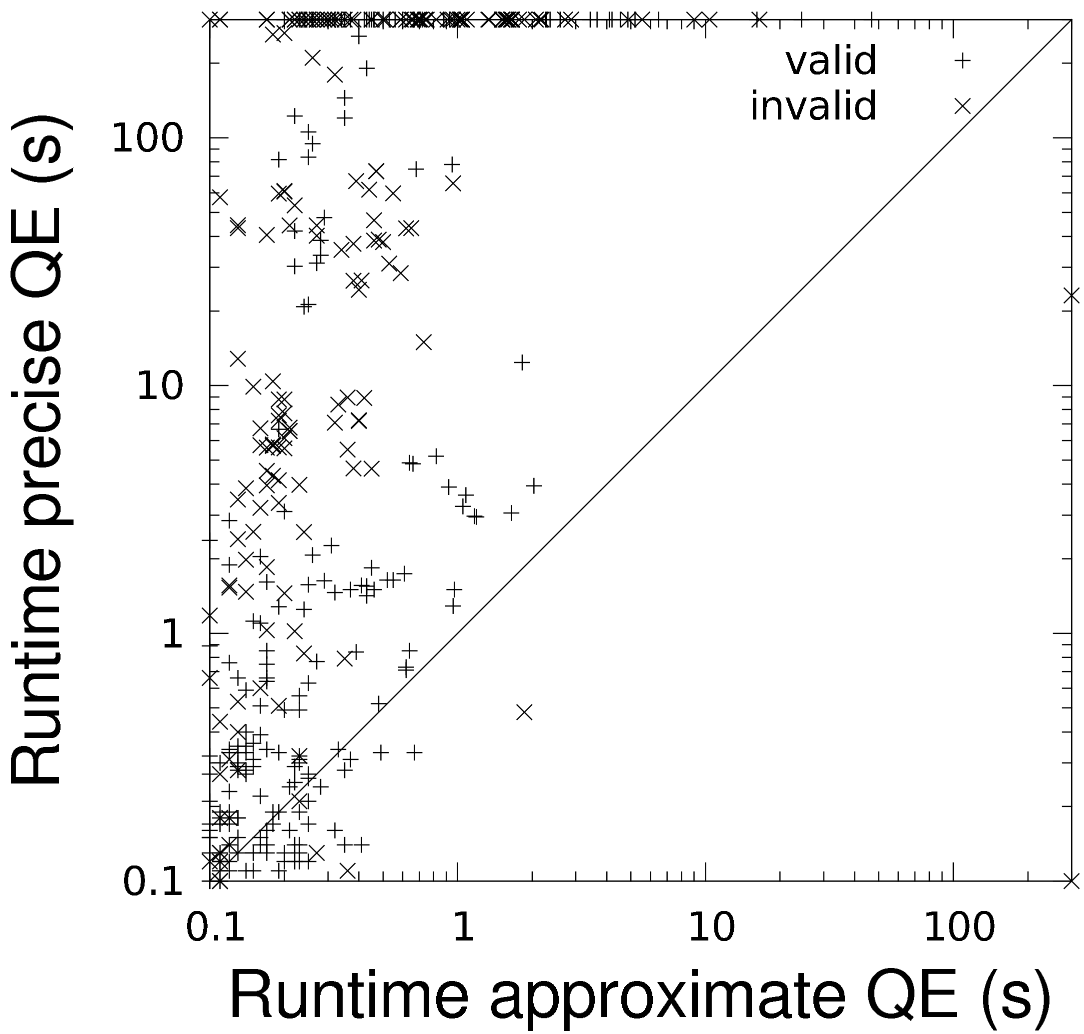
883 benchmark problems, each containing a transition system specified in Lustre and a single property

About half are *valid*, i.e., their property is invariant

Timeout: 300s of wall clock time

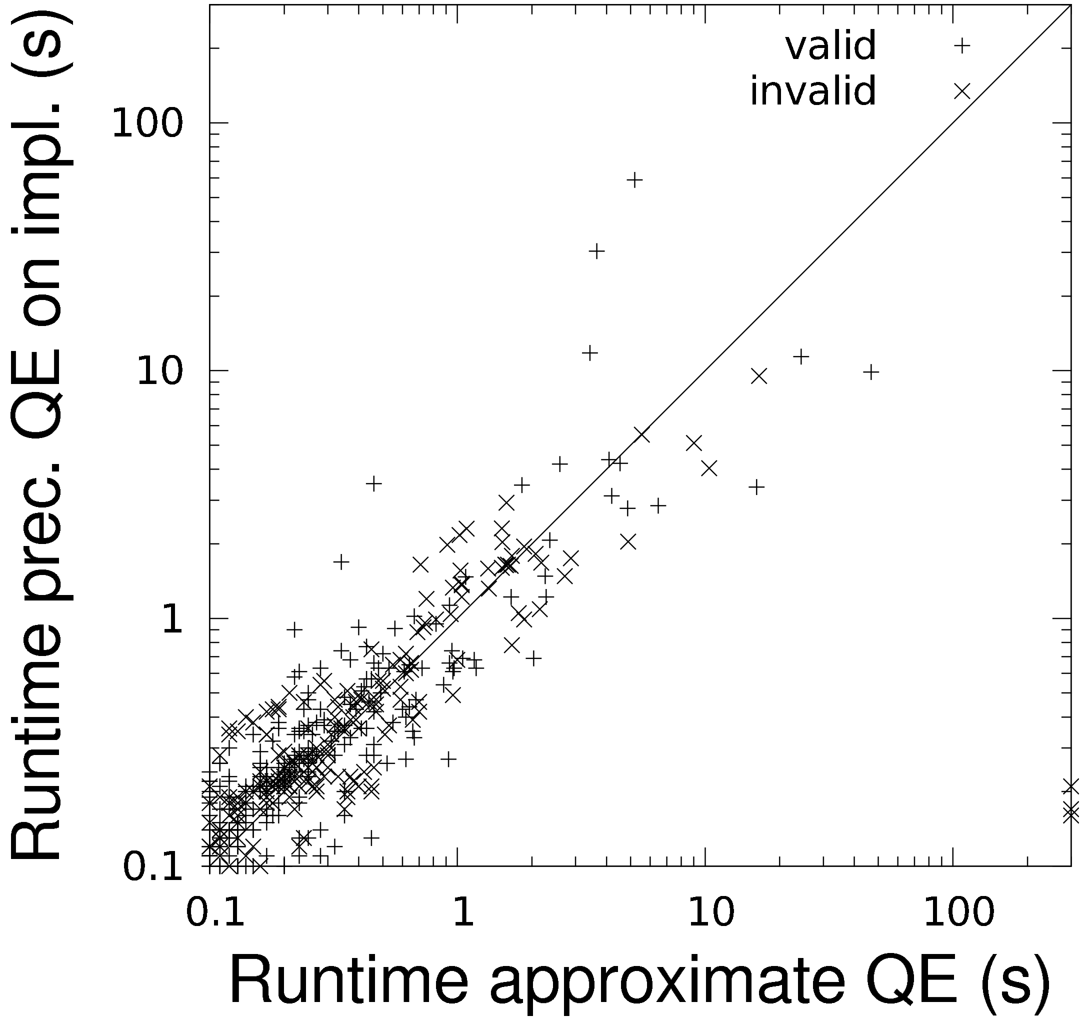
Hardware: AMD Opteron 24-core 2.1GHz with 32GB RAM

Precise vs. Approximate QE in Kind 2



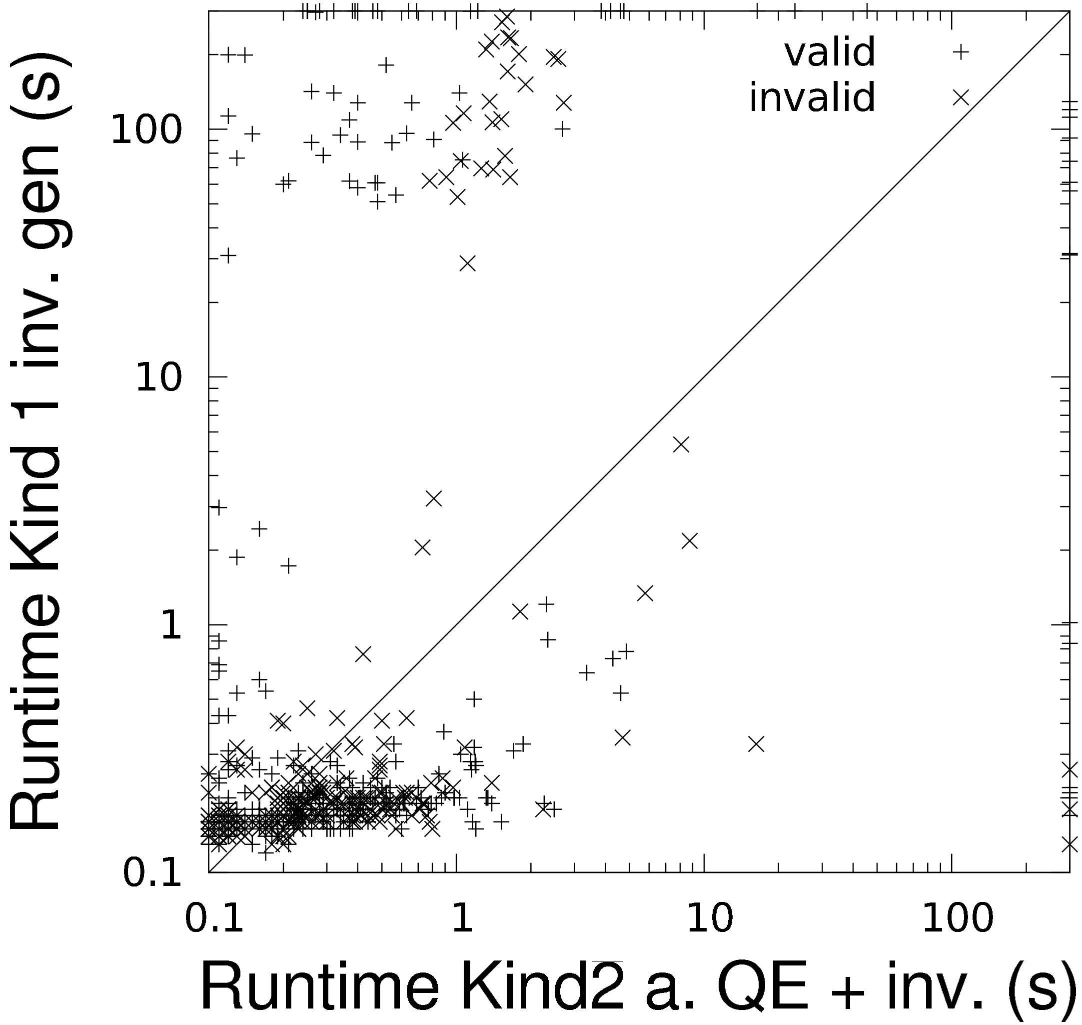
THE UNIVERSITY OF IOWA

Precise QE on Implicants

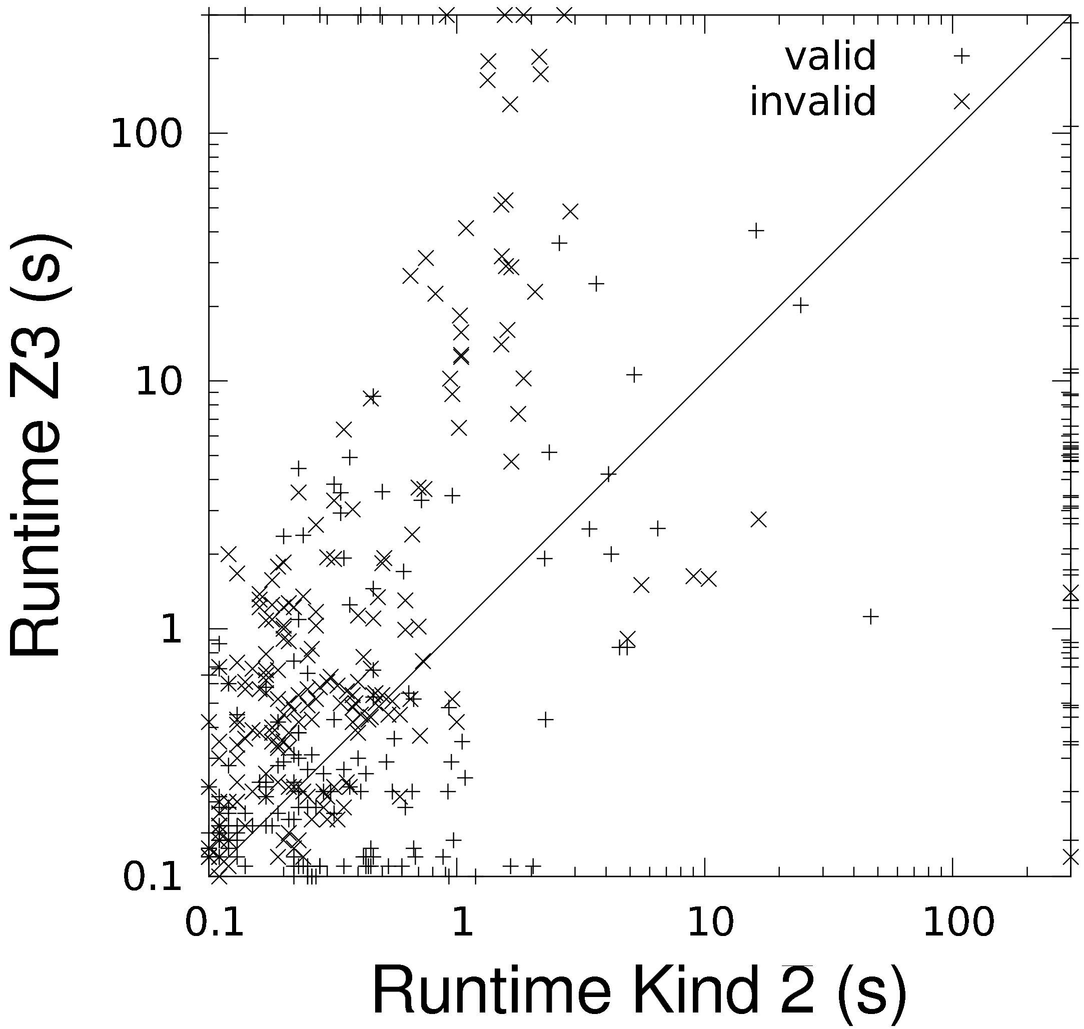


THE UNIVERSITY OF IOWA

Kind 2 vs. Kind 1 with Invariants



Kind 2 vs. Z3's PDR



THE UNIVERSITY OF IOWA

Conclusions

General version of IC3 procedure applying beyond propositional logic

A QE-based method for generalizing induction counterexamples for frame refinement

Explicit use of the counterexamples to guide approximate QE

Developed simple under-approximate QE method for LIA
IC3 procedure and QE mentor implemented within a new, multi-engine version of Kind model checker

Implementation competitive with other IC3-based system for same logic

Future Work

- Develop and integrate approximate QE methods for logics besides LIA
- Developing methods akin to ternary simulation in the propositional case to generalize approximate QE further
- In general, find new methods to weaken refinement lemmas to include more reachable states so as to enable or accelerate convergence in logics of interest