Property-Directed k-Induction

Dejan Jovanović Bruno Dutertre

SRI International

FMCAD 2016, Mountain View, CA

Thanks to NASA

Outline

1. Introduction
2. Property-Directed k-Induction
3. Experimental Evaluation

Outline

1. Introduction
2. Property-Directed k-Induction
3. Experimental Evaluation

Introduction

the problem

Given a transition system $\mathfrak{S} = \langle I, T \rangle$ with

- $\vec{x}$: state variables,
- $I(\vec{x})$: initial state formula,
- $T(\vec{x}, \vec{x}')$: state transition formula,

check whether all reachable states satisfy a property P.

Example: Zeno

Given $\mathfrak{S} = \langle I, T \rangle$ with

$$I \equiv (x = 0) \wedge (y = 0.5) \quad , \qquad T \equiv (x' = x + y) \wedge (y' = y/2) \quad ,$$

check whether $(x < 1)$.

Introduction

the problem

Given a transition system $\mathfrak{S} = \langle I, T \rangle$ with

- $\vec{x}$: state variables,
- $I(\vec{x})$: initial state formula,
- $T(\vec{x}, \vec{x}')$: state transition formula,

check whether all reachable states satisfy a property P.

Automation goals

1. Find bugs
2. Prove properties

Example: Zeno

Given $\mathfrak{S} = \langle I, T \rangle$ with

$$I \equiv (x = 0) \wedge (y = 0.5) \quad , \qquad T \equiv (x' = x + y) \wedge (y' = y/2) \quad ,$$

check whether $(x < 1)$.

Introduction

bounded model checking

Introduction

bounded model checking

$$I(\vec{x}_0) \wedge \neg P(\vec{x}_0)$$

Introduction

bounded model checking

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge \neg P(\vec{x}_1)$$

Introduction

bounded model checking

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge \neg P(\vec{x}_2)$$

Introduction

bounded model checking

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge T(\vec{x}_2, \vec{x}_3) \wedge \neg P(\vec{x}_3)$$

Introduction

bounded model checking

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge T(\vec{x}_2, \vec{x}_3) \wedge \neg P(\vec{x}_3)$$

- Can find bugs, can not prove properties
- Can use off-the-shelf SAT/SMT solver
- For non-trivial systems unrolling can be expensive

Introduction

bounded model checking

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge T(\vec{x}_2, \vec{x}_3) \wedge \neg P(\vec{x}_3)$$

- Can find bugs, can not prove properties
- Can use off-the-shelf SAT/SMT solver
- For non-trivial systems unrolling can be expensive
- Finite reachability ✓

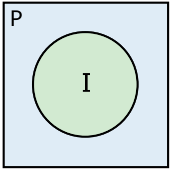
Introduction

induction

Introduction

induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

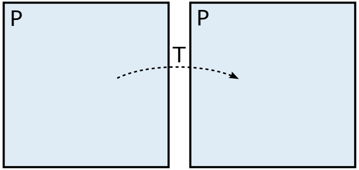


Introduction

induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

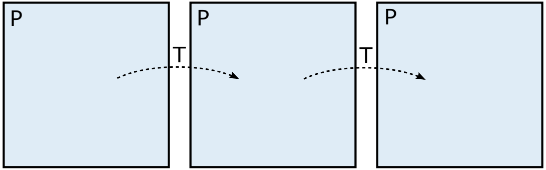


Introduction

induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

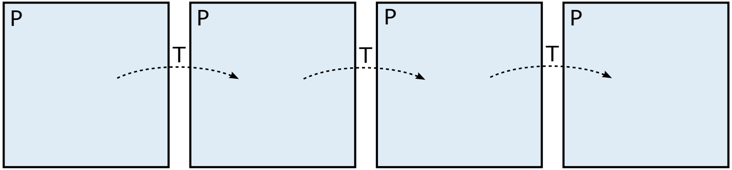


Introduction

induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$



Introduction

induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

- Can prove properties
- Can use off-the-shelf SAT/SMT solver

Introduction

induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

- Can prove properties
- Can use off-the-shelf SAT/SMT solver
- Zeno: property $(x < 1)$ is not inductive

Zeno

$I \equiv (x = 0) \wedge (y = 0.5)$

$T \equiv (x' = x + y) \wedge (y' = y/2)$

$P \equiv (x < 1)$

Introduction

induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

- Can prove properties
- Can use off-the-shelf SAT/SMT solver
- Zeno: property $(x < 1) \wedge (x + 2y \leq 1)$ is inductive

Zeno

$I \equiv (x = 0) \wedge (y = 0.5)$

$T \equiv (x' = x + y) \wedge (y' = y/2)$

$P \equiv (x < 1)$

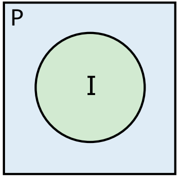
Introduction

k-induction

Introduction

k-induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

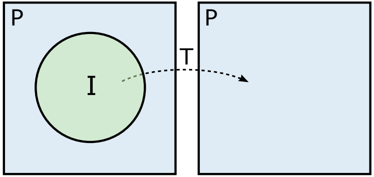


Introduction

k-induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$



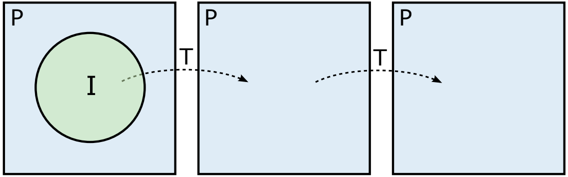
Introduction

k-induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \Rightarrow P(\vec{x}_2)$$



Introduction

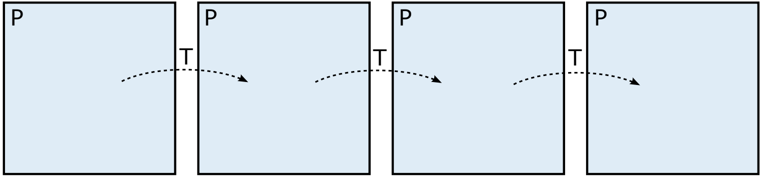
k-induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \Rightarrow P(\vec{x}_2)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge P(\vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge P(\vec{x}_2) \wedge T(\vec{x}_2, \vec{x}_3) \Rightarrow P(\vec{x}_3)$$



Introduction

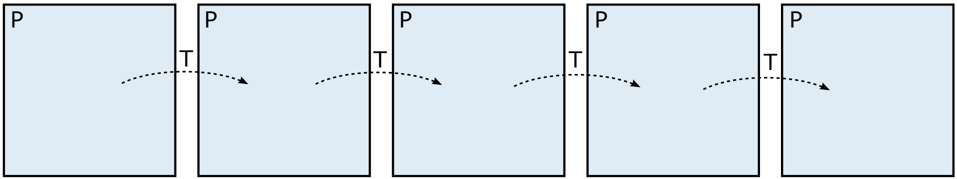
k-induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \Rightarrow P(\vec{x}_2)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge P(\vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge P(\vec{x}_2) \wedge T(\vec{x}_2, \vec{x}_3) \Rightarrow P(\vec{x}_3)$$



Introduction

k-induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \Rightarrow P(\vec{x}_2)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge P(\vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge P(\vec{x}_2) \wedge T(\vec{x}_2, \vec{x}_3) \Rightarrow P(\vec{x}_3)$$

- Can find bugs, can prove properties
- Can use off-the-shelf SAT/SMT solver
- For non-trivial systems unrolling can be expensive

Introduction

k-induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \Rightarrow P(\vec{x}_2)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge P(\vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge P(\vec{x}_2) \wedge T(\vec{x}_2, \vec{x}_3) \Rightarrow P(\vec{x}_3)$$

- Can find bugs, can prove properties
- Can use off-the-shelf SAT/SMT solver
- For non-trivial systems unrolling can be expensive
- Example: property $(|x| < 1)$ is not inductive

Stronger

$$I \equiv (x = 0) \wedge (y = 0)$$

$$T \equiv (x' = \frac{3}{5}x + \frac{2}{5}y) \wedge (|y'| < 1)$$

$$P \equiv (|x| < 1)$$

Introduction

k-induction

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

$$I(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \Rightarrow P(\vec{x}_2)$$

$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \wedge P(\vec{x}_1) \wedge T(\vec{x}_1, \vec{x}_2) \wedge P(\vec{x}_2) \wedge T(\vec{x}_2, \vec{x}_3) \Rightarrow P(\vec{x}_3)$$

- Can find bugs, can prove properties
- Can use off-the-shelf SAT/SMT solver
- For non-trivial systems unrolling can be expensive
- Example: property $(|x| < 1)$ is 2-inductive

Stronger

$I \equiv (x = 0) \wedge (y = 0)$

$T \equiv (x' = \frac{3}{5}x + \frac{2}{5}y) \wedge (|y'| < 1)$

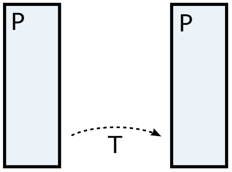
$P \equiv (|x| < 1)$

Introduction

strengthening

Key problem: find a **strengthening** that proves the property

$$I(\vec{x}_0) \Rightarrow P(\vec{x}_0)$$
$$P(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow P(\vec{x}_1)$$

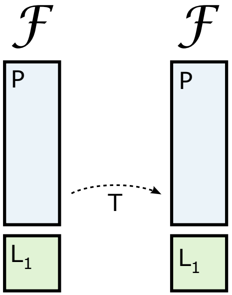


Introduction

strengthening

Key problem: find a **strengthening** that proves the property

$$I(\vec{x}_0) \Rightarrow \mathcal{F}(\vec{x}_0)$$
$$\mathcal{F}(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow \mathcal{F}(\vec{x}_1)$$
$$\mathcal{F}(\vec{x}) \Rightarrow P(\vec{x})$$

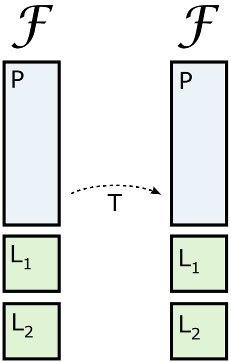


Introduction

strengthening

Key problem: find a **strengthening** that proves the property

$$I(\vec{x}_0) \Rightarrow \mathcal{F}(\vec{x}_0)$$
$$\mathcal{F}(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow \mathcal{F}(\vec{x}_1)$$
$$\mathcal{F}(\vec{x}) \Rightarrow P(\vec{x})$$

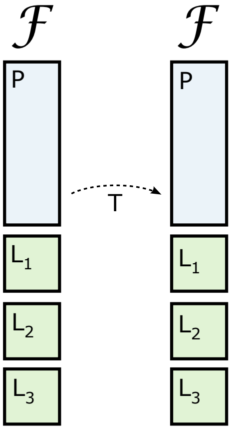


Introduction

strengthening

Key problem: find a **strengthening** that proves the property

$$I(\vec{x}_0) \Rightarrow \mathcal{F}(\vec{x}_0)$$
$$\mathcal{F}(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow \mathcal{F}(\vec{x}_1)$$
$$\mathcal{F}(\vec{x}) \Rightarrow P(\vec{x})$$



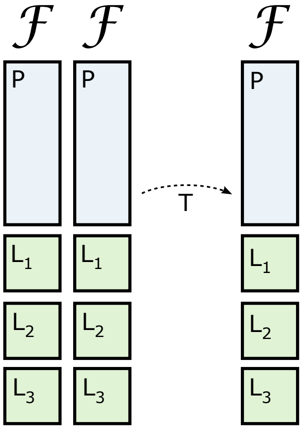
Introduction

strengthening

Key problem: find a **strengthening** that proves the property

$$I(\vec{x}_0) \Rightarrow \mathcal{F}(\vec{x}_0)$$
$$\mathcal{F}(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow \mathcal{F}(\vec{x}_1)$$
$$\mathcal{F}(\vec{x}) \Rightarrow P(\vec{x})$$

- Same for *k*-induction



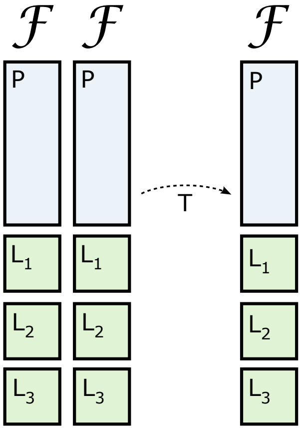
Introduction

strengthening

Key problem: find a **strengthening** that proves the property

$$I(\vec{x}_0) \Rightarrow \mathcal{F}(\vec{x}_0)$$
$$\mathcal{F}(\vec{x}_0) \wedge T(\vec{x}_0, \vec{x}_1) \Rightarrow \mathcal{F}(\vec{x}_1)$$
$$\mathcal{F}(\vec{x}) \Rightarrow P(\vec{x})$$

- Same for *k*-induction
- Is *k*-induction stronger? ✓



Introduction

timeline

- Induction
- Bounded model checking [BCCZ99]
- *k*-induction [SSS00]
- Interpolation-based model checking [McM03]
- IC3/PDR [Bra11]

Introduction

timeline

- Induction
- Bounded model checking [BCCZ99]
- *k*-induction [SSS00]
- Interpolation-based model checking [McM03]
- IC3/PDR [Bra11]
  - based on induction
  - incremental strengthening
  - no unrolling: lots of “easy” queries
  - interpolation-based learning

Introduction

timeline

- Induction
- Bounded model checking [BCCZ99]
- *k*-induction [SSS00]
- Interpolation-based model checking [McM03]
- IC3/PDR [Bra11]
  - based on induction
  - incremental strengthening
  - no unrolling: lots of “easy” queries
  - interpolation-based learning
- Lots of work on SMT-based extensions [HB12, CG12, KGC14, CGMT14]

Outline

1. Introduction
2. Property-Directed k-Induction
3. Experimental Evaluation

Property-Directed *k*-Induction

modules

SMT solving

more than SAT/UNSAT

1-step reachability

more than reachable/unreachable

k-step reachability

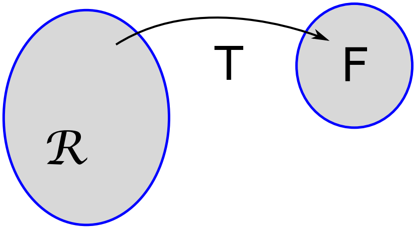
more than reachable/unreachable

***k*-induction**

search for a strengthening and learn from failures

Property-Directed *k*-Induction

1-step reachability

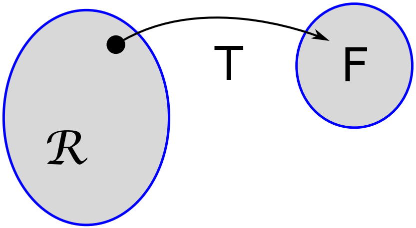


Basic satisfiability query

$$\mathcal{R}(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge F(\vec{x}')$$

Property-Directed k-Induction

1-step reachability



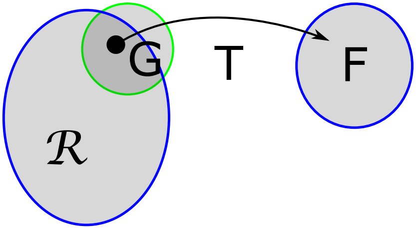
Basic satisfiability query

$$\mathcal{R}(\vec{x}) \land T(\vec{x}, \vec{x}') \land F(\vec{x}')$$

- SAT

Property-Directed *k*-Induction

1-step reachability



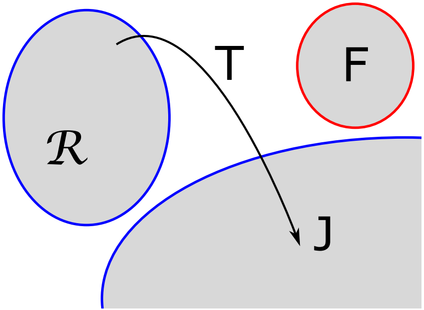
Basic satisfiability query

$$\mathcal{R}(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge F(\vec{x}')$$

- SAT : generalize the counterexample to G — YICES2 with [KGC14]

Property-Directed k-Induction

1-step reachability



Basic satisfiability query

$$\mathcal{R}(\vec{x}) \wedge T(\vec{x}, \vec{x}') \wedge F(\vec{x}')$$

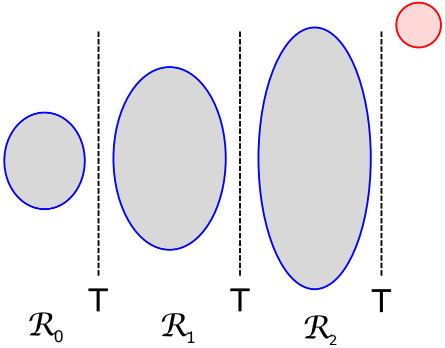
- SAT : generalize the counterexample to G — YICES2 with [KGC14]
- UNSAT: interpolate, with J refuting F — MATHSAT5

Property-Directed k-Induction

k-step reachability

Reachability in k steps

Given F that is not reachable in $< k$ steps, check if it's reachable in k steps.



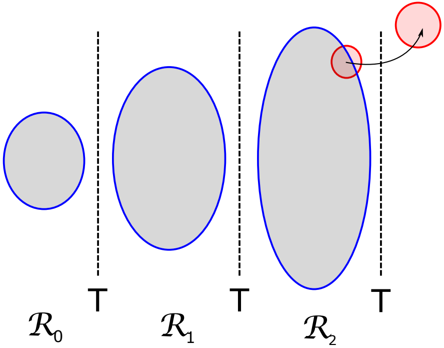
- $\mathcal{R}_i$ valid up to i

Property-Directed k-Induction

k-step reachability

Reachability in k steps

Given F that is not reachable in $< k$ steps, check if it's reachable in k steps.



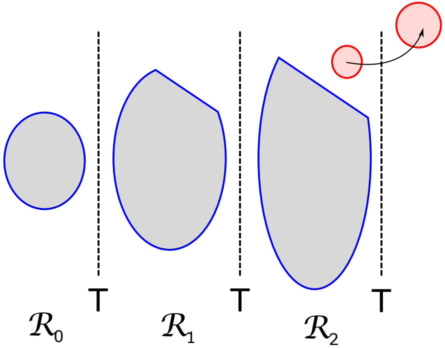
- $\mathcal{R}_i$ valid up to i
- 1-step backward search

Property-Directed k-Induction

k-step reachability

Reachability in k steps

Given F that is not reachable in $< k$ steps, check if it's reachable in k steps.



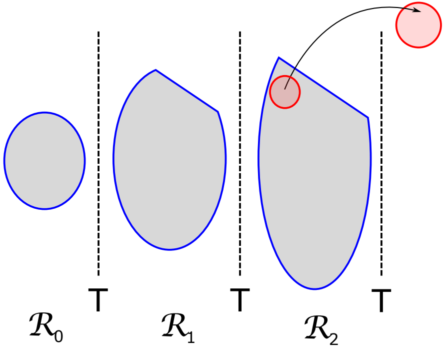
- $\mathcal{R}_i$ valid up to i
- 1-step backward search
- learn and refine $\mathcal{R}_i$

Property-Directed *k*-Induction

k-step reachability

Reachability in *k* steps

Given F that is not reachable in $< k$ steps, check if it's reachable in k steps.



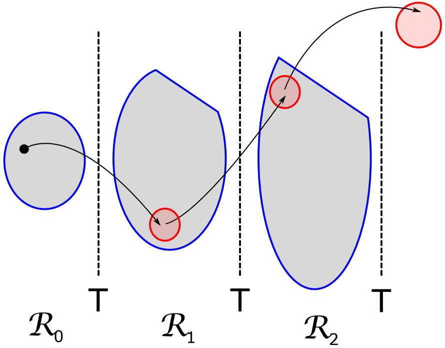
- $\mathcal{R}_i$ valid up to i
- 1-step backward search
- learn and refine $\mathcal{R}_i$

Property-Directed k-Induction

k-step reachability

Reachability in k steps

Given F that is not reachable in $< k$ steps, check if it's reachable in k steps.



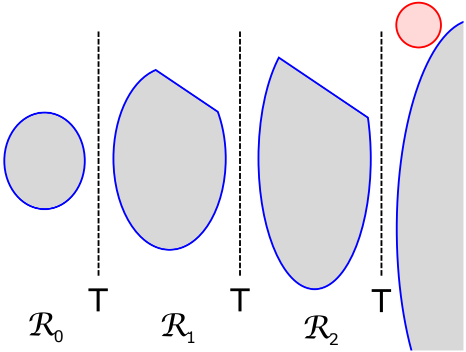
- $\mathcal{R}_i$ valid up to i
- 1-step backward search
- learn and refine $\mathcal{R}_i$
- all the way: reachable

Property-Directed k-Induction

k-step reachability

Reachability in k steps

Given F that is not reachable in $< k$ steps, check if it's reachable in k steps.



- $\mathcal{R}_i$ valid up to i
- 1-step backward search
- learn and refine $\mathcal{R}_i$
- all the way: reachable
- unreachable: learn
- learned fact valid up to k

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

function PD-KIND$(\mathfrak{S}, P)$
    $n \leftarrow 0$
    $\mathcal{F} \leftarrow \{(P, \neg P)\}$
    loop
        pick k-induction depth $1 \leq k \leq n + 1$
        $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH$(\mathfrak{S}, \mathcal{F}, P, n, k)$
        if P marked invalid **then return invalid**
        if $\mathcal{F} = \mathcal{G}$ **then return valid**
        $n \leftarrow n_p$
        $\mathcal{F} \leftarrow \mathcal{G}$

Setup

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

```
function PD-KIND(𝔖, P)
    n ← 0
    𝓕 ← {(P, ¬P)}
    loop
        pick k-induction depth 1 ≤ k ≤ n + 1
        ⟨𝓕, 𝓖, n_p⟩ ← PUSH(𝔖, 𝓕, P, n, k)
        if P marked invalid then return invalid
        if 𝓕 = 𝓖 then return valid
        n ← n_p
        𝓕 ← 𝓖
```

Setup

- single reasoning frame $\mathcal{F}$

Property-Directed *k*-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

function PD-KIND($\mathfrak{S}, P$)

  $n \leftarrow 0$

  $\mathcal{F} \leftarrow \{(P, \neg P)\}$

  loop

    pick k-induction depth $1 \leq k \leq n + 1$

    $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH($\mathfrak{S}, \mathcal{F}, P, n, k$)

    if P marked invalid **then return invalid**

    if $\mathcal{F} = \mathcal{G}$ **then return valid**

    $n \leftarrow n_p$

    $\mathcal{F} \leftarrow \mathcal{G}$

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

```
function PD-KIND(𝔖, P)
    n ← 0
    𝓕 ← {(P, ¬P)}
    loop
        pick k-induction depth 1 ≤ k ≤ n + 1
        ⟨𝓕, 𝓖, n_p⟩ ← PUSH(𝔖, 𝓕, P, n, k)
        if P marked invalid then return invalid
        if 𝓕 = 𝓖 then return valid
        n ← n_p
        𝓕 ← 𝓖
```

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

function PD-KIND($\mathfrak{S}, P$)
  $n \leftarrow 0$
  $\mathcal{F} \leftarrow \{(P, \neg P)\}$
  loop
    pick k-induction depth $1 \leq k \leq n + 1$
    $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH$(\mathfrak{S}, \mathcal{F}, P, n, k)$
    if P marked invalid **then return invalid**
    if $\mathcal{F} = \mathcal{G}$ **then return valid**
    $n \leftarrow n_p$
    $\mathcal{F} \leftarrow \mathcal{G}$

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$
- F_{ABS} is valid up to n

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

function PD-KIND($\mathfrak{S}, P$)
    $n \leftarrow 0$
    $\mathcal{F} \leftarrow \{(P, \neg P)\}$
    loop
        pick k-induction depth $1 \leq k \leq n + 1$
        $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH$(\mathfrak{S}, \mathcal{F}, P, n, k)$
        if P marked invalid **then return invalid**
        if $\mathcal{F} = \mathcal{G}$ **then return valid**
        $n \leftarrow n_p$
        $\mathcal{F} \leftarrow \mathcal{G}$

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$
- F_{ABS} is valid up to n
- $F_{\text{CEX}} \rightsquigarrow \neg P$, F_{ABS} refutes F_{CEX}

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

function PD-KIND($\mathfrak{S}, P$)

- $n \leftarrow 0$
- $\mathcal{F} \leftarrow \{(P, \neg P)\}$
- **loop**
  - pick k-induction depth $1 \leq k \leq n+1$
  - $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH$(\mathfrak{S}, \mathcal{F}, P, n, k)$
  - **if** P marked invalid **then return invalid**
  - **if** $\mathcal{F} = \mathcal{G}$ **then return valid**
  - $n \leftarrow n_p$
  - $\mathcal{F} \leftarrow \mathcal{G}$

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$
- F_{ABS} is valid up to n
- $F_{\text{CEX}} \rightsquigarrow \neg P$, F_{ABS} refutes F_{CEX}

Initially

- P is valid up to $n = 0$
- $\neg P \rightsquigarrow \neg P$, P refutes $\neg P$

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

function PD-KIND($\mathfrak{S}, P$)
    $n \leftarrow 0$
    $\mathcal{F} \leftarrow \{(P, \neg P)\}$
    loop
        pick k-induction depth $1 \leq k \leq n + 1$
        $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH$(\mathfrak{S}, \mathcal{F}, P, n, k)$
        if P marked invalid **then return invalid**
        if $\mathcal{F} = \mathcal{G}$ **then return valid**
        $n \leftarrow n_p$
        $\mathcal{F} \leftarrow \mathcal{G}$

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$
- F_{ABS} is valid up to n
- $F_{\text{CEX}} \rightsquigarrow \neg P$, F_{ABS} refutes F_{CEX}

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

function PD-KIND($\mathfrak{S}, P$)
  $n \leftarrow 0$
  $\mathcal{F} \leftarrow \{(P, \neg P)\}$
  loop
    pick k-induction depth $1 \leq k \leq n + 1$
    $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH($\mathfrak{S}, \mathcal{F}, P, n, k$)
    if P marked invalid **then return invalid**
    if $\mathcal{F} = \mathcal{G}$ **then return valid**
    $n \leftarrow n_p$
    $\mathcal{F} \leftarrow \mathcal{G}$

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$
- F_{ABS} is valid up to n
- $F_{\text{CEX}} \rightsquigarrow \neg P$, F_{ABS} refutes F_{CEX}

Property-Directed *k*-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

function PD-KIND($\mathfrak{S}, P$)
    $n \leftarrow 0$
    $\mathcal{F} \leftarrow \{(P, \neg P)\}$
    loop
        pick k-induction depth $1 \leq k \leq n + 1$
        $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH($\mathfrak{S}, \mathcal{F}, P, n, k$)
        if P marked invalid **then return invalid**
        if $\mathcal{F} = \mathcal{G}$ **then return valid**
        $n \leftarrow n_p$
        $\mathcal{F} \leftarrow \mathcal{G}$

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$
- F_{ABS} is valid up to n
- $F_{\text{CEX}} \rightsquigarrow \neg P$, F_{ABS} refutes F_{CEX}

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T\rangle$ and $I \Rightarrow P$

```
function PD-KIND(𝔖, P)
    n ← 0
    𝓕 ← {(P, ¬P)}
    loop
        pick k-induction depth 1 ≤ k ≤ n + 1
        ⟨𝓕, 𝓖, n_p⟩ ← PUSH(𝔖, 𝓕, P, n, k)
        if P marked invalid then return invalid
        if 𝓕 = 𝓖 then return valid
        n ← n_p
        𝓕 ← 𝓖
```

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$
- F_{ABS} is valid up to n
- $F_{\text{CEX}} \rightsquigarrow \neg P$, F_{ABS} refutes F_{CEX}

Property-Directed k-Induction

main procedure

Require: $\mathfrak{S} = \langle I, T \rangle$ and $I \Rightarrow P$

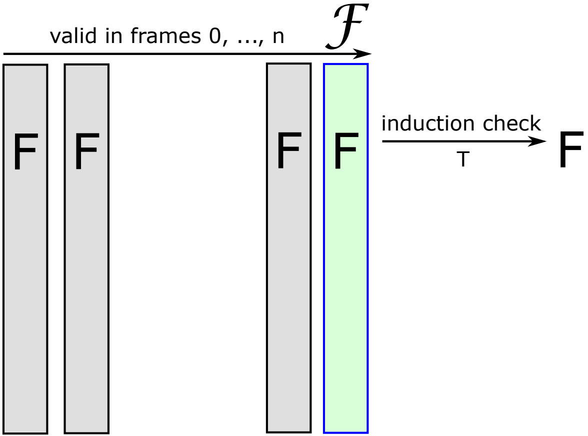
function PD-KIND($\mathfrak{S}, P$)
  $n \leftarrow 0$
  $\mathcal{F} \leftarrow \{(P, \neg P)\}$
  loop
    pick k-induction depth $1 \leq k \leq n + 1$
    $\langle \mathcal{F}, \mathcal{G}, n_p \rangle \leftarrow$ PUSH$(\mathfrak{S}, \mathcal{F}, P, n, k)$
    if P marked invalid **then return invalid**
    if $\mathcal{F} = \mathcal{G}$ **then return valid**
    $n \leftarrow n_p$
    $\mathcal{F} \leftarrow \mathcal{G}$

Setup

- single reasoning frame $\mathcal{F}$
- reasoning index n
- obligations $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$
- F_{ABS} is valid up to n
- $F_{\text{CEX}} \rightsquigarrow \neg P$, F_{ABS} refutes F_{CEX}

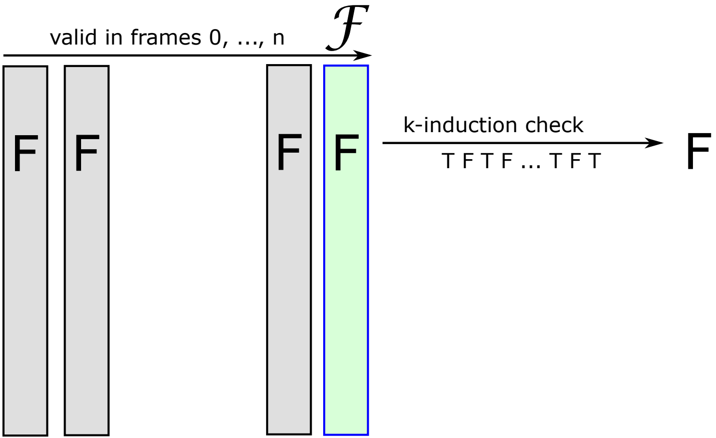
Property-Directed *k*-Induction

main procedure



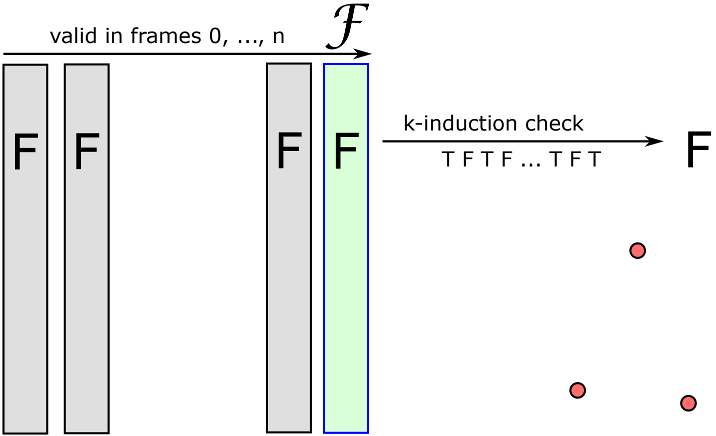
Property-Directed *k*-Induction

main procedure



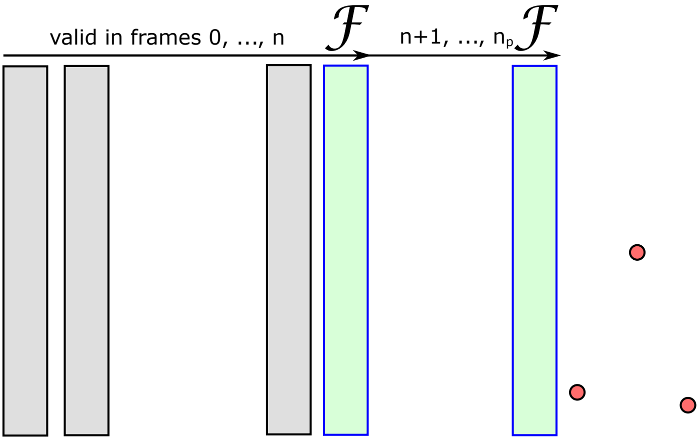
Property-Directed *k*-Induction

main procedure



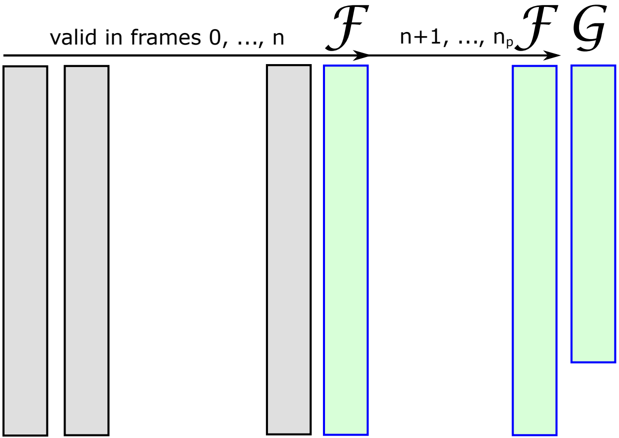
Property-Directed *k*-Induction

main procedure



Property-Directed *k*-Induction

main procedure



Property-Directed k-induction

the PUSH procedure

Pick an obligation $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

Property-Directed *k*-induction

the PUSH procedure

Pick an obligation $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

$$\mathcal{F} \wedge T \wedge \ldots \wedge \mathcal{F} \wedge T \Rightarrow F_{\text{ABS}}$$

Is F_{ABS} k-inductive relative to $\mathcal{F}$?

Property-Directed *k*-induction

the PUSH procedure

Pick an obligation $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

$$\mathcal{F} \wedge T \wedge \ldots \wedge \mathcal{F} \wedge T \Rightarrow F_{\text{ABS}}$$

Is F_{ABS} k-inductive relative to $\mathcal{F}$? If yes, **push** it ✓

Property-Directed *k*-induction

the PUSH procedure

Pick an obligation $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

$$\mathcal{F} \land T \land \ldots \land \mathcal{F} \land T \Rightarrow F_{\text{ABS}}$$

Is F_{ABS} k-inductive relative to $\mathcal{F}$? If no, get the generalization G_{CTI} of the CTI

Property-Directed *k*-induction

the PUSH procedure

Pick an obligation $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

$$\mathcal{F} \wedge T \wedge \ldots \wedge \mathcal{F} \wedge T \wedge F_{\text{CEX}}$$

Can we get to F_{CEX}?

Property-Directed *k*-induction

the PUSH procedure

Pick an obligation $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

$$\mathcal{F} \wedge T \wedge \ldots \wedge \mathcal{F} \wedge T \wedge F_{\text{CEX}}$$

Can we get to F_{CEX}? If yes, then generalize to G_{CEX}

- If G_{CEX} reachable, then we have a **counter-example** to P ✓
- If G_{CEX} not reachable, **learn** lemma to eliminate G_{CEX} ✓

Property-Directed *k*-induction

the PUSH procedure

Pick an obligation $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

$$\mathcal{F} \wedge T \wedge \ldots \wedge \mathcal{F} \wedge T \Rightarrow F_{\text{ABS}}$$
$$\mathcal{F} \wedge T \wedge \ldots \wedge \mathcal{F} \wedge T \wedge F_{\text{CEX}}$$

We have a generalization G_{CTI} of the CTI, and can not get to F_{CEX}

Property-Directed k-induction

the PUSH procedure

Pick an obligation $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$

$$\mathcal{F} \wedge T \wedge \ldots \wedge \mathcal{F} \wedge T \Rightarrow F_{\text{ABS}}$$
$$\mathcal{F} \wedge T \wedge \ldots \wedge \mathcal{F} \wedge T \wedge F_{\text{CEX}}$$

We have a generalization G_{CTI} of the CTI, and can not get to F_{CEX}

- If G_{CTI} reachable, **weaken** F_{ABS} to $\neg F_{\text{CEX}}$ ✓
- If G_{CTI} not reachable, **learn** lemma and **strengthen** F_{ABS} ✓

Outline

1. Introduction
2. Property-Directed k-Induction
3. Experimental Evaluation

Experimental Evaluation

overall

problem set	Z3			SPACER			NUXMV			**PD-KIND**		
	✓	⊤/⊥	time	✓	⊤/⊥	time	✓	⊤/⊥	time	✓	⊤/⊥	time
approximate-agreement (9)	9	8/1	213	7	6/1	1150	9	8/1	2174	**9**	**8/1**	**164**
azadmanesh-kieckhafer (20)	20	17/3	3404	20	17/3	4678	20	17/3	294	**20**	**17/3**	**192**
cav12 (99)	69	48/21	2102	71	49/22	3529	**72**	**50/22**	**7443**	71	49/22	4990
conc (6)	4	4/0	128	4	4/0	655	**6**	**6/0**	**421**	4	4/0	270
ctigar (110)	64	44/20	1683	72	52/20	4249	76	56/20	1342	**77**	**57/20**	**2823**
hacms (5)	1	1/0	11	1	1/0	4	4	3/1	388	**5**	**3/2**	**1661**
lustre (790)	757	421/336	1888	763	427/336	2263	760	424/336	7660	**774**	**438/336**	**3494**
oral-messages (9)	9	7/2	16	9	7/2	44	9	7/2	161	**9**	**7/2**	**2**
tta-startup (3)	1	1/0	9	**1**	**1/0**	**8**	1	1/0	17	**1**	**1/0**	**8**
tte-synchro (6)	6	3/3	969	6	3/3	445	5	2/3	405	**6**	**3/3**	**21**
unified-approx (11)	8	5/3	2928	11	8/3	589	**11**	**8/3**	**139**	11	8/3	217
	948	559/389	13351	965	575/390	17614	973	582/391	20444	**987**	**595/392**	**13842**

timeout of 20 minutes, Z3 [HB12], NUXMV [CGMT14], SPACER [KGC14]

Experimental Evaluation

as a variant of IC3/PDR

	Z3			SPACER			NUXMV			**PD-KIND$_\infty$**			**PD-KIND$_1$**		
problem set	✓	⊤/⊥	time	✓	⊤/⊥	time	✓	⊤/⊥	time	✓	⊤/⊥	time	✓	⊤/⊥	time
approximate-agreement (9)	9	8/1	213	7	6/1	1150	9	8/1	2174	9	8/1	164	**9**	**8/1**	**155**
azadmanesh-kieckhafer (20)	20	17/3	3404	20	17/3	4678	20	17/3	294	20	17/3	192	**20**	**17/3**	**107**
cav12 (99)	69	48/21	2102	71	49/22	3529	72	50/22	7443	71	49/22	4990	**74**	**50/24**	**6404**
conc (6)	4	4/0	128	4	4/0	655	**6**	**6/0**	**421**	4	4/0	270	5	5/0	164
ctigar (110)	64	44/20	1683	72	52/20	4249	76	56/20	1342	**77**	**57/20**	**2823**	73	53/20	4920
hacms (5)	1	1/0	11	1	1/0	4	4	3/1	388	**5**	**3/2**	**1661**	1	1/0	2
lustre (790)	757	421/336	1888	763	427/336	2263	760	424/336	7660	**774**	**438/336**	**3494**	769	431/338	2019
oral-messages (9)	9	7/2	16	9	7/2	44	9	7/2	161	**9**	**7/2**	**2**	9	7/2	74
tta-startup (3)	1	1/0	9	1	1/0	8	1	1/0	17	1	1/0	8	**2**	**1/1**	**742**
tte-synchro (6)	6	3/3	969	6	3/3	445	5	2/3	405	**6**	**3/3**	**21**	6	3/3	60
unified-approx (11)	8	5/3	2928	11	8/3	589	**11**	**8/3**	**139**	11	8/3	217	11	8/3	158
	948	559/389	13351	965	575/390	17614	973	582/391	20444	**987**	**595/392**	**13842**	979	584/395	14805

timeout of 20 minutes, Z3 [HB12], NUXMV [CGMT14], SPACER [KGC14]

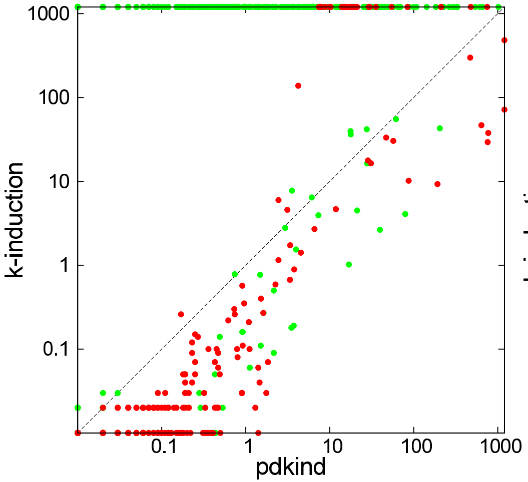
Experimental Evaluation

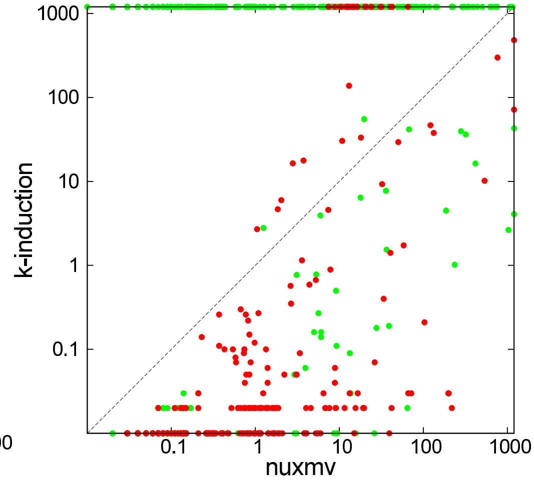
overall

- Effective and robust on real-world problems
- Good at both proving properties and finding bugs
- *k*-induction: can prove properties using a smaller strengthening
- *k*-induction: the only engine that can prove all *k*-inductive properties
- *k*-induction: effective bug-finder due to the longer steps of *k*-induction

Experimental Evaluation

k-induction





Summary

New method for infinite-state systems:

- variant of IC3/PDR based on *k*-induction
- effective in practice: proofs and bugs
- focuses on induction rather than bugs
- no SMT query left behind
- more powerful than *k*-induction
- modular: tunable, amenable to heuristics
- implemented in SALLY (fork me at GitHub)

References I

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu.
Symbolic model checking without BDDs.
Tools and Algorithms for the Construction and Analysis of Systems, pages 193–207, 1999.

[Bra11] Aaron R Bradley.
SAT-based model checking without unrolling.
In *Verification, Model Checking, and Abstract Interpretation*, pages 70–87, 2011.

[CG12] Alessandro Cimatti and Alberto Griggio.
Software model checking via IC3.
In *Computer Aided Verification*, pages 277–293, 2012.

[CGMT14] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta.
IC3 modulo theories via implicit predicate abstraction.
In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 46–61. 2014.

[HB12] Kryštof Hoder and Nikolaj Bjørner.
Generalized property directed reachability.
In *Theory and Applications of Satisfiability Testing*, pages 157–171. 2012.

[KGC14] Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki.
SMT-based model checking for recursive programs.
In *Computer Aided Verification*, pages 17–34, 2014.

References II

[McM03] Kenneth L McMillan.
Interpolation and SAT-based model checking.
In *International Conference on Computer Aided Verification*, pages 1–13, 2003.

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck.
Checking safety properties using induction and a SAT-solver.
In *Formal Methods in Computer-Aided Design*, pages 127–144, 2000.