Data-Driven Precondition Inference with Learned Features

PLDI 2016

Saswat Padhi
University of California Los Angeles

Rahul Sharma
Stanford University

Todd Millstein
University of California Los Angeles

"Data-Driven" Precondition Inference

$\{0 < x < 5\}$

```
y := x + 1
```

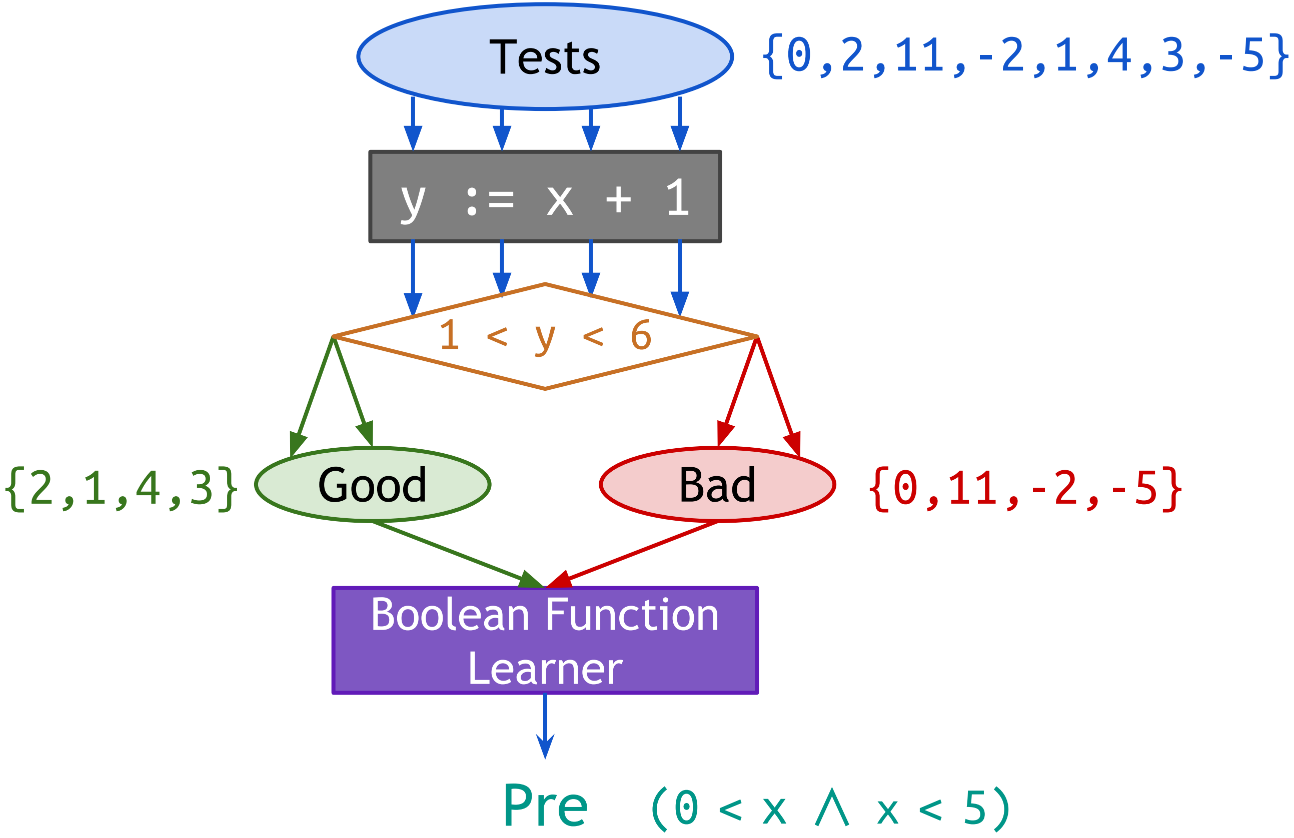
$\{1 < y < 6\}$

Find *Pre* such that $\{Pre\}\ C\ \{Post\}$

Learn likely precondition from test executions

- Likely Specifications
  - Likely precondition inference [2008 Sankaranarayanan et al]
  - Learning commutativity specs [2015 Gehr et al]
- Program Verification
  - Randomized search [2014 Sharma et al]
  - ICE, ICE-DT [2014 Garg et al], [2016 Garg et al]

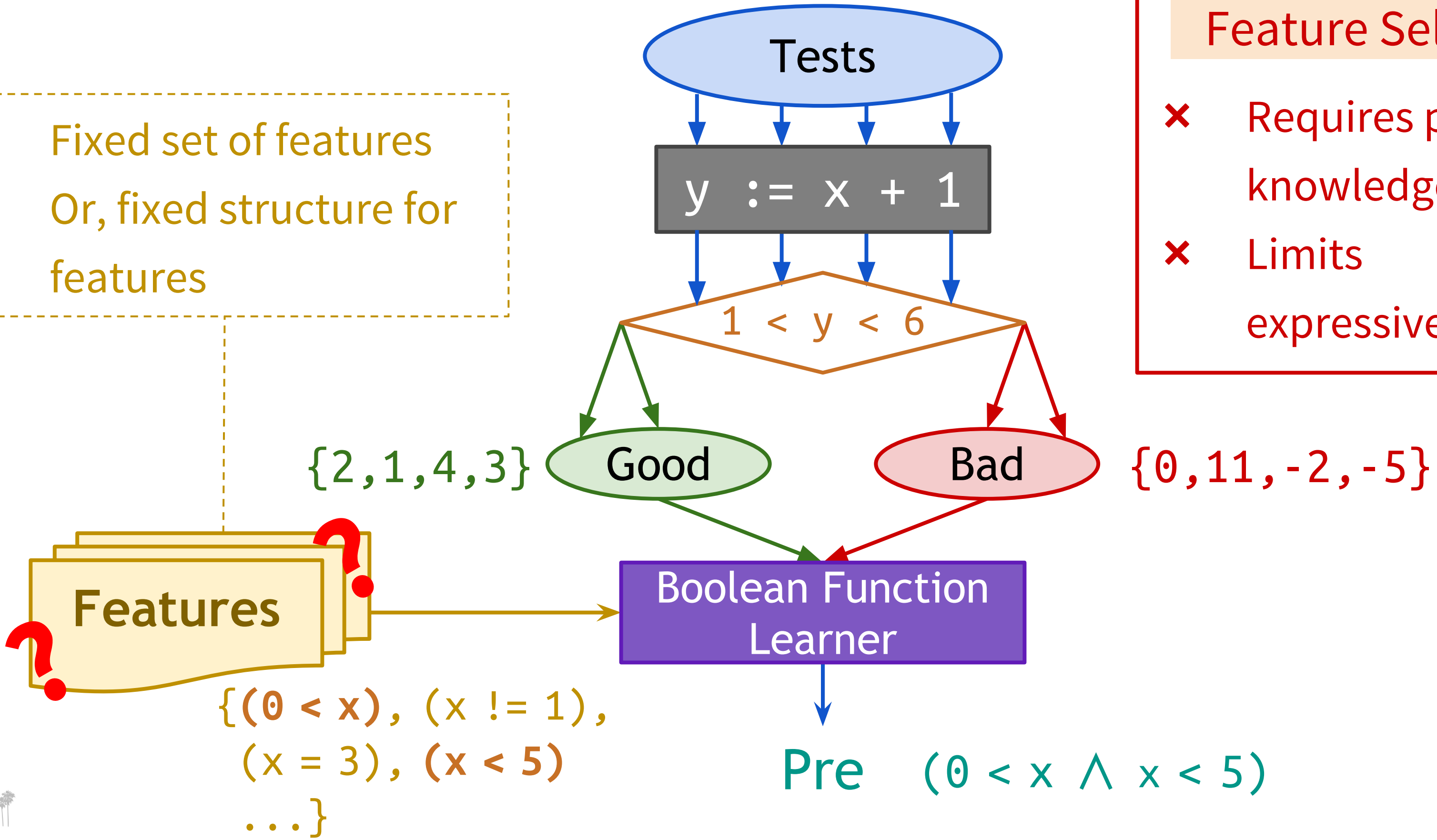
General Data-Driven Approach



PLDI 2016

Issue with Prior Approaches

- ➢ Fixed set of features
- ➢ Or, fixed structure for features



Feature Selection

- × Requires prior knowledge
- × Limits expressiveness

Contributions

On-demand feature generation for data-driven invariant inference

⇒ Using program synthesis

A. Likely Precondition Inference
   - ✓ Precondition Inference Engine (PIE)
   - ✓ No initial features - automatic, on-demand feature learning
   - ✓ Sound & sufficient precondition, up to the tests

B. Program Verification
   - ✓ Sound loop invariant inference
   - ✓ Automatic, on-demand feature learning

Sources available at https://github.com/SaswatPadhi/PIE

Likely Precondition Inference

OCaml's `String.sub(s, i, l)`:

⇒ Postcondition = `no exceptions`

⇒ Precondition inferred by PIE =

$$(i \geq 0) \land (l \geq 0) \land ((\text{length } s) \geq i + l)$$

Data-Driven Inference

Input (s, i, l)	Features	
	`i ≥ 0`	`l ≥ 0`
("pie", 1, -1)	T	F
("xy", 2, 3)	T	T
("a", -1, 3)	F	T
("pqrs", 1, 2)	T	T
("bcd", 2, 1)	T	T

Fixed feature set = `{i ≥ 0, l ≥ 0}`

Separate good and bad inputs by *learning* a predicate

(over the fixed set of features)

Issue with Prior Work

Input (s, i, l)	Features $i \geq 0$	Features $l \geq 0$
("pie", 1, -1)	T	F
("xy", 2, 3)	T	T
("a", -1, 3)	F	T
("pqrs", 1, 2)	T	T
("bcd", 2, 1)	T	T

No such boolean formula!

Prior work *must* violate *at least* one input

$(i \geq 0) \wedge (l \geq 0)$

Insufficient!

Key Insight: *Conflicts*

Input (s, i, l)	Features i ≥ 0	Features l ≥ 0
(“pie”, 1, -1)	T	F
(“xy”, 2, 3)	T	T
(“a”, -1, 3)	F	T
(“pqrs”, 1, 2)	T	T
(“bcd”, 2, 1)	T	T

Conflict

When a feature vector appears both in the good set and the bad set

Conflicts cannot be resolved with a fixed set of features

PIE: Conflict Resolution

Input (s, i, l)	Features		
	i ≥ 0	l ≥ 0	?
("pie", 1, -1)	T	F	
("xy", 2, 3)	T	T	F
("a", -1, 3)	F	T	
("pqrs", 1, 2)	T	T	T
("bcd", 2, 1)	T	T	T

Add a new feature to resolve a conflict

(s, i, l)	?
("xy", 2, 3)	F
("pqrs", 1, 2)	T
("bcd", 2, 1)	T

PIE: Learned Precondition

Input (s, i, l)	Features		
	i ≥ 0	l ≥ 0	(length s) ≥ i + l
("pie", 1, -1)	T	F	T
("xy", 2, 3)	T	T	F
("a", -1, 3)	F	T	F
("pqrs", 1, 2)	T	T	T
("bcd", 2, 1)	T	T	T

(i ≥ 0) ∧
(l ≥ 0) ∧
((length s) ≥ i + l)

(s, i, l)	?
("xy", 2, 3)	F
("pqrs", 1, 2)	T
("bcd", 2, 1)	T

Feature Learning using Program Synthesis

Synthesize features using a grammar

```
expr := var
      | expr + expr
      | ...      /* arithmetic operations */
      | expr > expr
      | ...      /* relational operations */
      | (length expr)
      | ...      /* string operations */
      | ...      /* other operations */
```

Standard set of operations for several data-types

User may extend with additional operations

(s, i, l)	?
(“xy”, 2, 3)	F
(“pqrs”, 1, 2)	T
(“bcd”, 2, 1)	T

Enumerative Synthesis

Enumerate *expr* in grammar by size

A smaller *expr* is likely to be more general

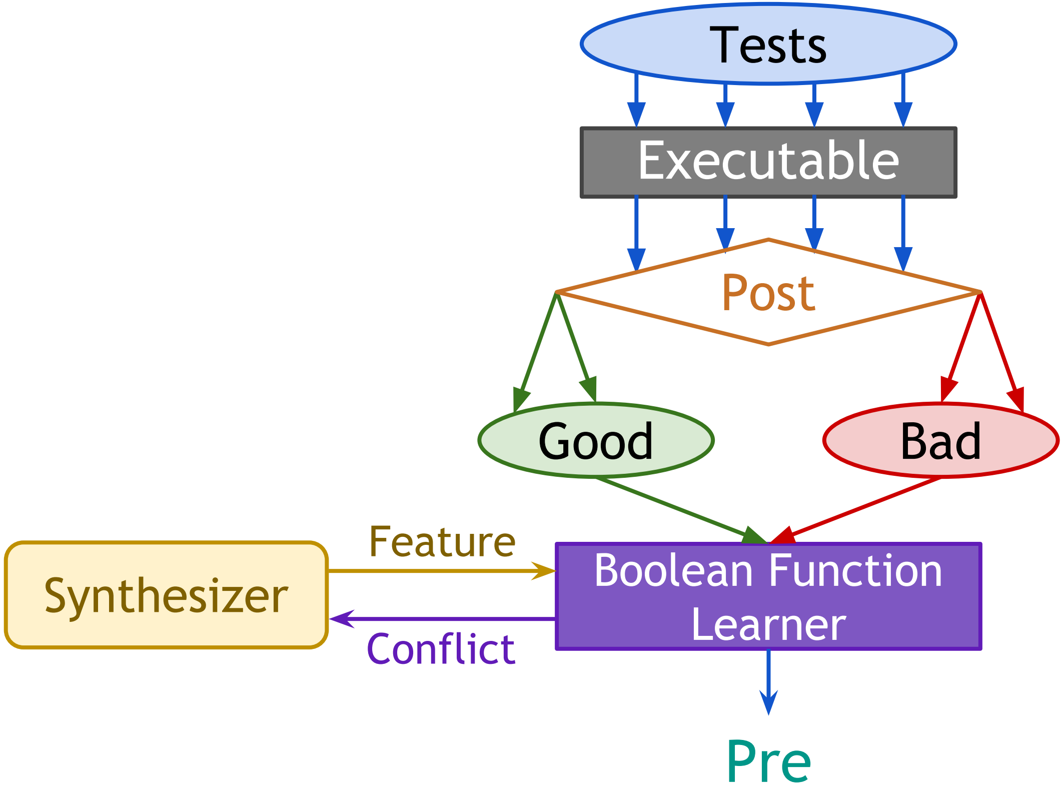
Learned feature = `(length s) ≥ i + l`

More tests avoid overly specific features

```
i = 1
. . .
. . . .
. . . . .
(length s) ≥ i + l
```

(s, i, l)	?
("xy", 2, 3)	F
("pqrs", 1, 2)	T
("bcd", 2, 1)	T

Summary of PIE



PIE: Properties

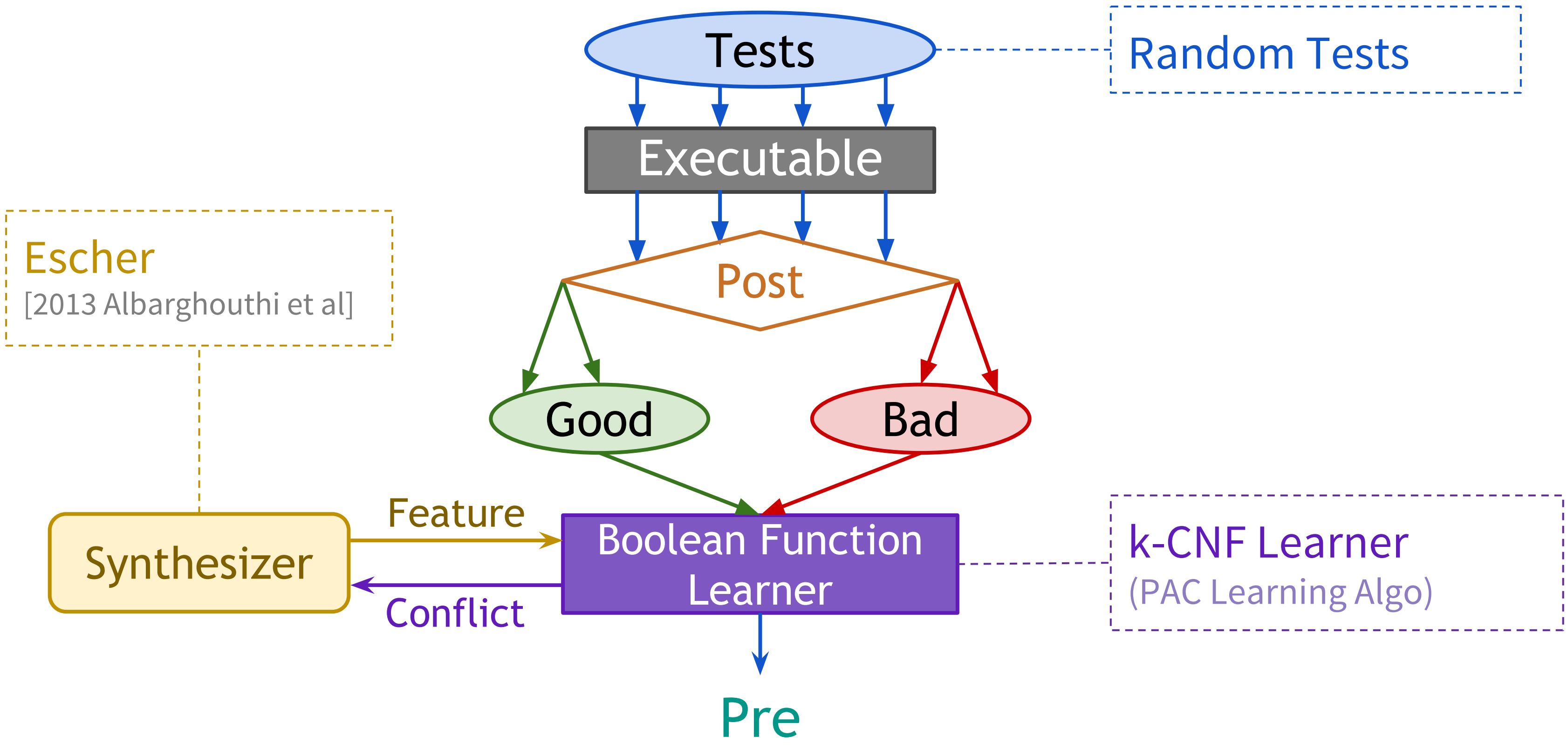
✓ Sufficiency & Necessity

Preconditions inferred by PIE are sufficient & necessary, up to the set of test inputs.

✓ Strong Convergence

If there is a precondition (sufficient & necessary up to the tests) expressible in the provided grammar, PIE would find it in finite time.

PIE: Implementation



PIE: Evaluation

⇒ All first-order functions in

→ `List`

→ `String`

→ `BatAvlTree (AVL Tree)`

⇒ Correct preconditions: 87 / 101

→ Incomplete library documentation

⇢ `BatAvlTree.split_leftmost`

⇢ `BatAvlTree.split_rightmost`

→ Failure causes

⇢ Inadequate test coverage

⇢ Lack of quantifier support

$$\dfrac{(i < 0) \lor (i > \mathrm{len}(s)) \lor \neg\, \mathrm{has}(\mathrm{sub}(s,\ i,\ \mathrm{len}(s) - i),\ c)}{\texttt{String.index\_from}(s, i, c) : \text{throws exception}}$$

Benchmarks + logs available at
https://github.com/SaswatPadhi/PIE

PLDI 2016

Contributions

On-demand feature generation for data-driven invariant inference

A. Likely Precondition Inference (PIE)

B. Program Verification

Sources available at https://github.com/SaswatPadhi/PIE

Program Verification

```
function foo(...) {
  assume P
  while B do
    S
  done
  assert Q
}
```

Infer loop invariant for verification

➡ Data-Driven Loop Invariant Inference

- × Randomized search
  [2014 Sharma et al]
- × ICE, ICE-DT
  [2014 Garg et al], [2016 Garg et al]

Fixed set of features
Or, fixed template for features

PIE for Loop Invariant Inference

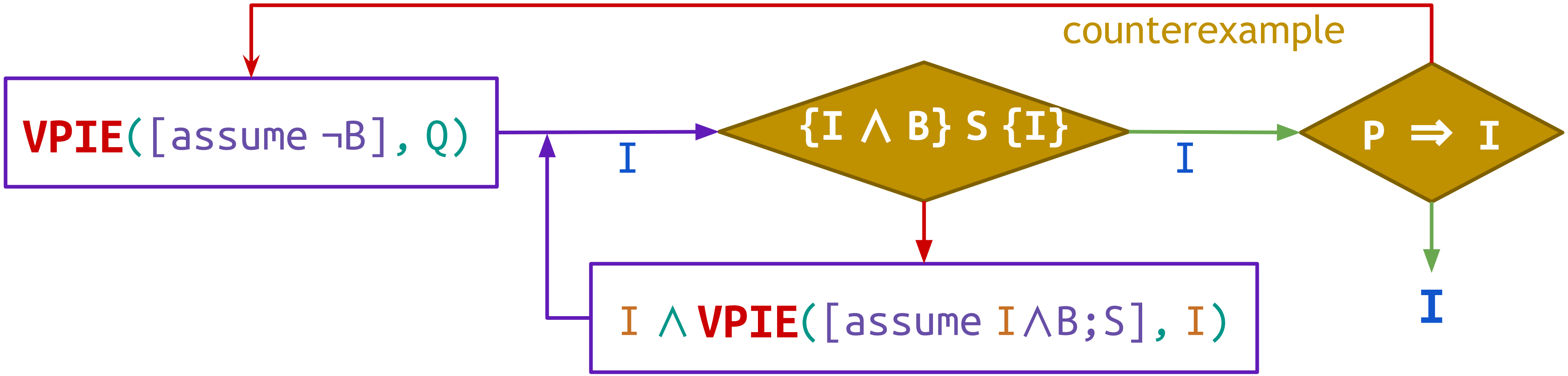
```
assume P
while B do S done
assert Q
```

Obtain loop invariant
using ~~logical abduction~~

HOLA [2013 Dillig et al]

Precondition Inference

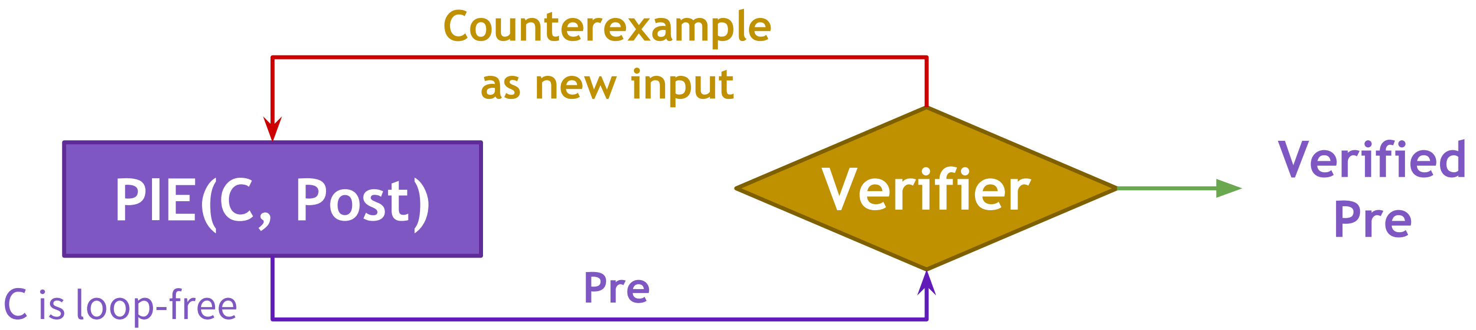
$I \land \neg B \Rightarrow Q$ $\{I \land B\}\ S\ \{I\}$ $P \Rightarrow I$



VPIE: Verified PIE

Guess and check

CEGIS [2006 Solar-Lezama et al]



Evaluation

⇒ Benchmarks from prior static & data-driven program verifiers

- → *HOLA* [1] [43/46]: linear integer arithmetic
  - × Static: Only for theories supporting quantifier elimination
- → *ICE-DT* [2] [37/39]: linear, non-linear integer arithmetic
  - × Fixed template for features, requires specialized learner
- → *Randomized Search* [3] [4/4]: combination of theories: string, integer
  - × Pre-defined features, structure for invariants

[1] [2013 Dillig et al] [2] [2016 Garg et al] [3] [2014 Sharma et al]

Related Works

⇒ Likely Precondition Inference

→ Dynamic Inference of Likely Preconditions
[2008 Sankaranarayanan et al]

→ Learning Commutativity Specifications
[2015 Gehr et al]

Fixed set of features

Lack of sufficiency *and* necessity

⇒ Loop Invariant Inference

→ ICE, ICE-DT
[2014 Garg et al], [2016 Garg et al]

Fixed template for features, Requires specialized learner

→ Randomized Search
[2014 Sharma et al]

Fixed set of features, fixed structure for invariants

Conclusion

A novel *feature generation* technique
for data-driven precondition inference

A. Inferring necessary & sufficient preconditions

B. Program verification by inferring sound loop invariants

Sources available at https://github.com/SaswatPadhi/PIE

Thanks!

PIE is an open-source project.

[https://github.com/SaswatPadhi/PIE]

Questions

Comments

Suggestions?

padhi@cs.ucla.edu