LoopInvGen:

A Loop Invariant Generator based on Precondition Inference

SyGuS-COMP 2018

Saswat Padhi
Univ. of California, LA

Rahul Sharma
Microsoft Research, India

Todd Millstein
Univ. of California, LA

Verification

C/C++ Code

```
int n, x = 0, m = 0;

while (x < n) {
  if (rand()) m = x;
  x = x + 1;
}

if(n > 0)
  assert (0 <= m && m <= n );
```

Verification

C/C++ Code

```
int n, x = 0, m = 0;

while (x < n) {
  if (rand()) m = x;
  x = x + 1;
}

if(n > 0)
  assert (0 <= m && m <= n );
```

SyGuS-INV Problem

```
(define-fun pre-f ((x Int) (n Int) (m Int)) Bool
    (and (= x 0) (= m 0)))

(define-fun trans-f ((x Int) (n Int) (m Int)
                     (x! Int) (n! Int) (m! Int)) Bool
    (or (and (and (and (< x n) (= x! (+ x 1)))
                  (= n! n)) (= m! m))
        (and (and (and (< x n) (= x! (+ x 1)))
                  (= n! n)) (= m! x))))

(define-fun post-f ((x Int) (n Int) (m Int)) Bool
    (not (and (and (>= x n) (> n 0))
              (or (<= n m) (< m 0)))))

(inv-constraint inv-f pre-f trans-f post-f)
```

Verification

C/C++ Code

```
int n, x = 0, m = 0;

while (x < n) {
  if (rand()) m = x;
  x = x + 1;
}

if(n > 0)
  assert (0 <= m && m <= n );
```

```
(define-fun inv-f ((x Int) (n Int) (m Int)) Bool
    (and (>= x m)
         (or (and (> n m) (> m 0))
             (= 0 m)))
```

SyGuS-INV Problem

```
(define-fun pre-f ((x Int) (n Int) (m Int)) Bool
    (and (= x 0) (= m 0)))

(define-fun trans-f ((x Int) (n Int) (m Int)
                     (x! Int) (n! Int) (m! Int)) Bool
    (or (and (and (and (< x n) (= x! (+ x 1)))
                  (= n! n)) (= m! m))
        (and (and (and (< x n) (= x! (+ x 1)))
                  (= n! n)) (= m! x))))

(define-fun post-f ((x Int) (n Int) (m Int)) Bool
    (not (and (and (>= x n) (> n 0))
              (or (<= n m) (< m 0)))))

(inv-constraint inv-f pre-f trans-f post-f)
```

LoopInvGen

- Data-driven inference technique
  - Guided by program trace
  - Extensible predicate language

LoopInvGen

- Data-driven inference technique
  - Guided by program trace
  - Extensible predicate language
- Reduces invariant inference to precondition inference problems
  - Based on PIE[PLDI 2016]

LoopInvGen

- Data-driven inference technique
  - Guided by program trace
  - Extensible predicate language
- Reduces invariant inference to precondition inference problems
  - Based on PIE[PLDI 2016]
- Uses Escher[CAV 2013] program synthesizer
- Queries Z3[TACAS 2008] for verification

LoopInvGen

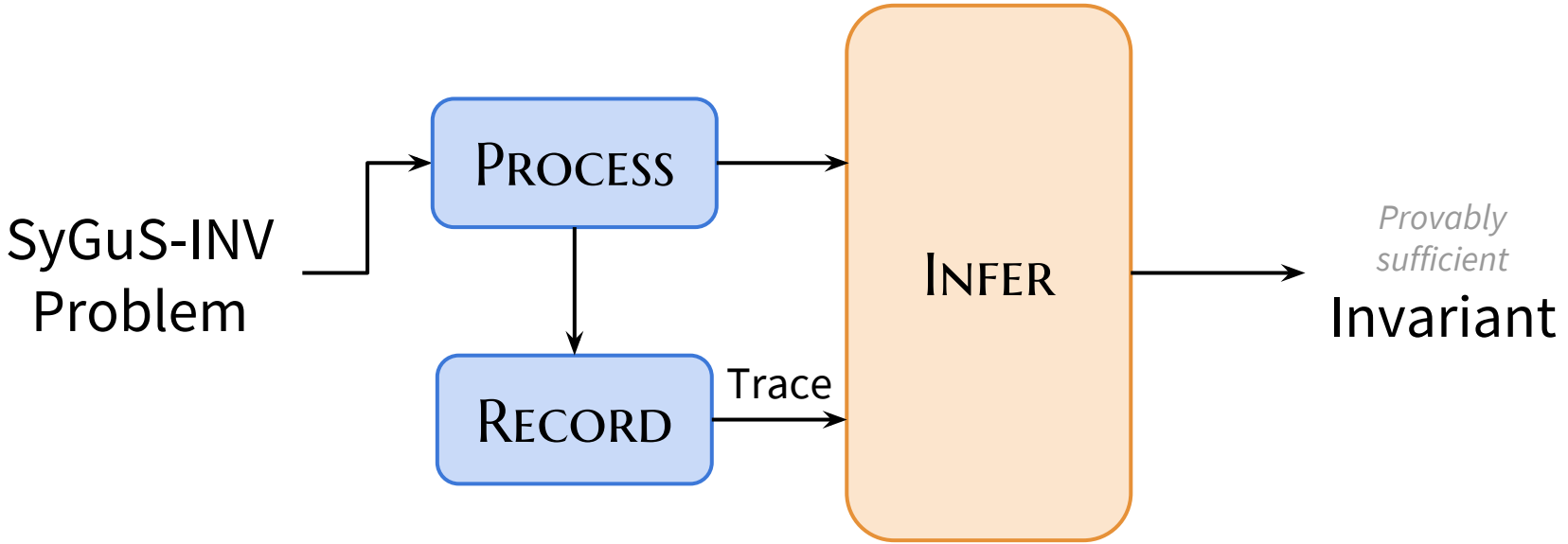
- Data-driven inference technique
  - Guided by program trace
  - Extensible predicate language
- Reduces invariant inference to precondition inference problems
  - Based on PIE[PLDI 2016]
- Uses Escher[CAV 2013] program synthesizer
- Queries Z3[TACAS 2008] for verification
- **Winner** of SyGuS-COMP 2017, 2018 (INV track)

LoopInvGen

- Data-driven inference technique
  - Guided by program trace
  - Extensible predicate language
- Reduces invariant inference to precondition inference problems
  - Based on PIE[PLDI 2016]
- Uses Escher[CAV 2013] program synthesizer
- Queries Z3[TACAS 2008] for verification
- **Winner** of SyGuS-COMP 2017, 2018 (INV track)

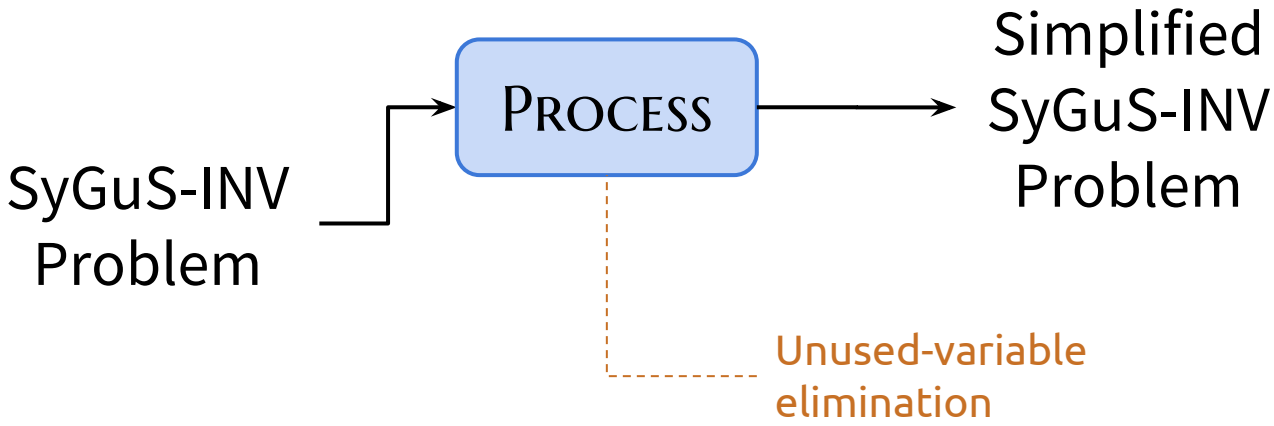
INV	Solved	Fastest	Shortest	Score
CVC4 1.6	109	82	59	850
DryadSynth	103	81	26	784
LoopInvGen	115	102	85	966
Horndini	68	21	25	428
EUSolver	61	15	61	411

Overview


SyGuS-INV
Problem
Process
Record
Trace
Infer
Provably
sufficient
Invariant

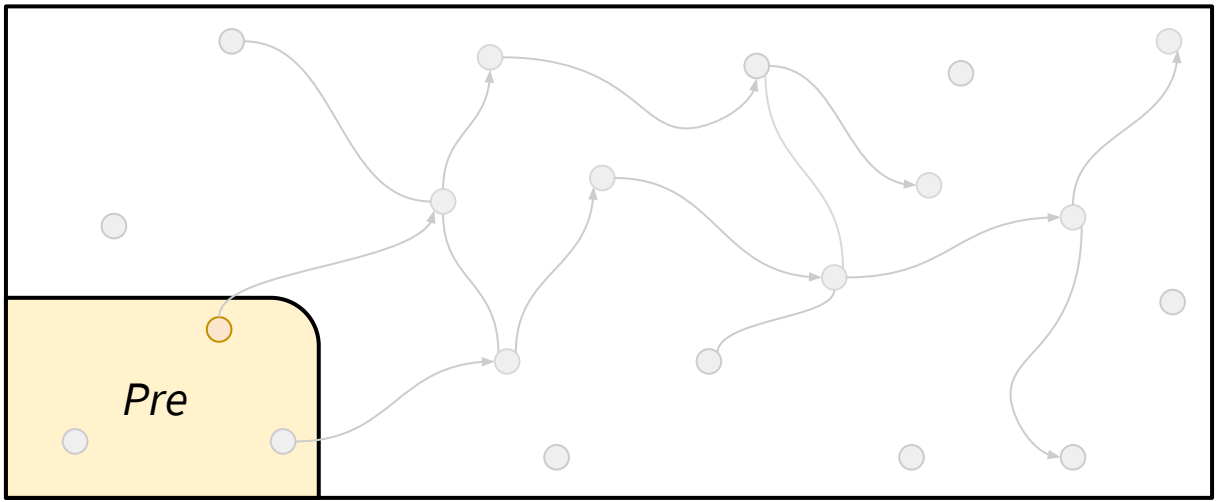
Initial PROCESS-ing

A static analysis pass for straight-forward simplifications



RECORD-ing Reachable States

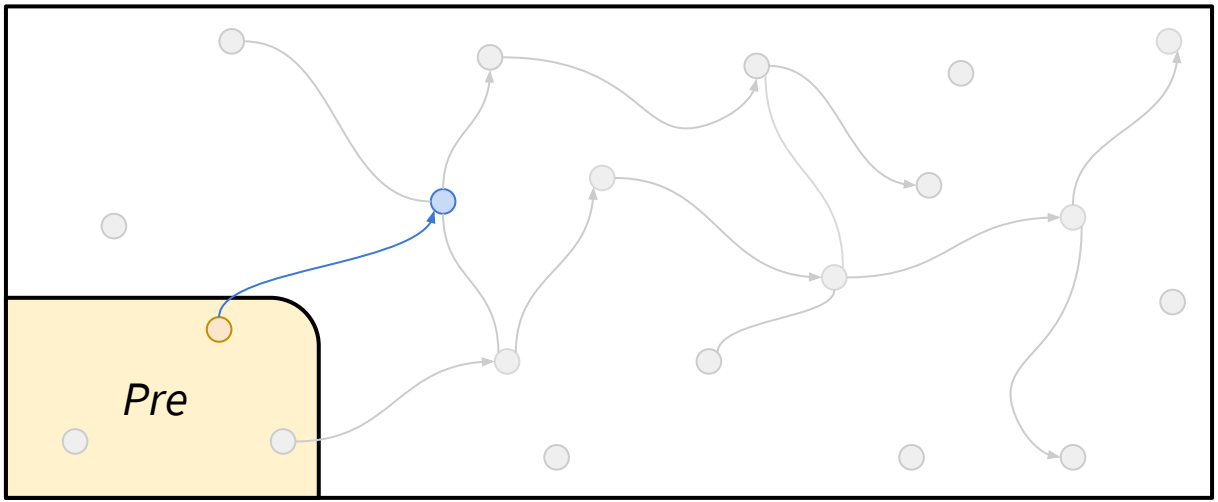
SyGuS Problem (Pre, Trans, Post) → List of variable assignments



1. Pick state s, s.t. Pre(s)

RECORD-ing Reachable States

SyGuS Problem (Pre, Trans, Post) → List of variable assignments

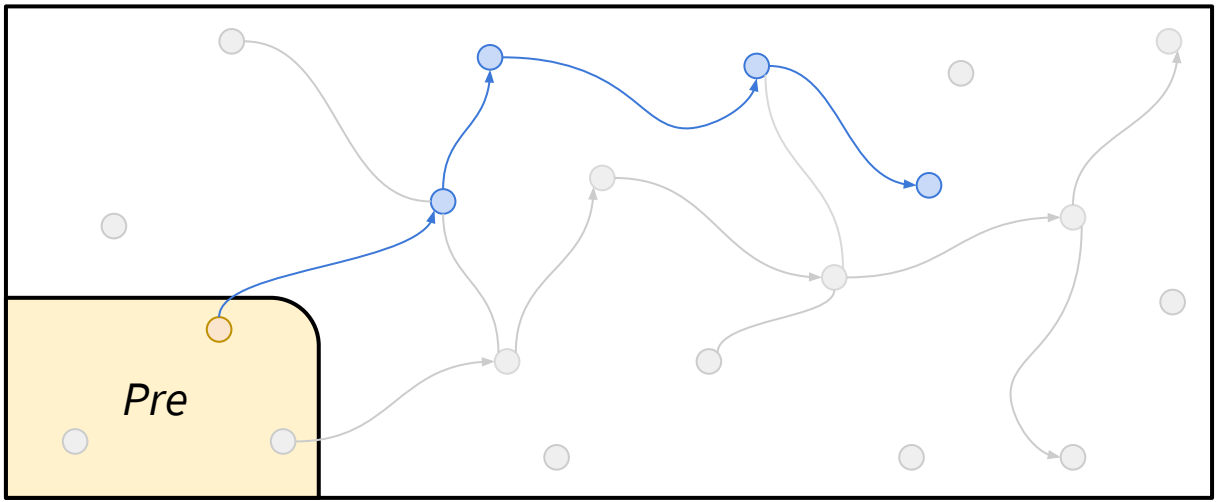


1. Pick state s, s.t. Pre(s)

2. Obtain state t, s.t. Trans(s,t)

RECORD-ing Reachable States

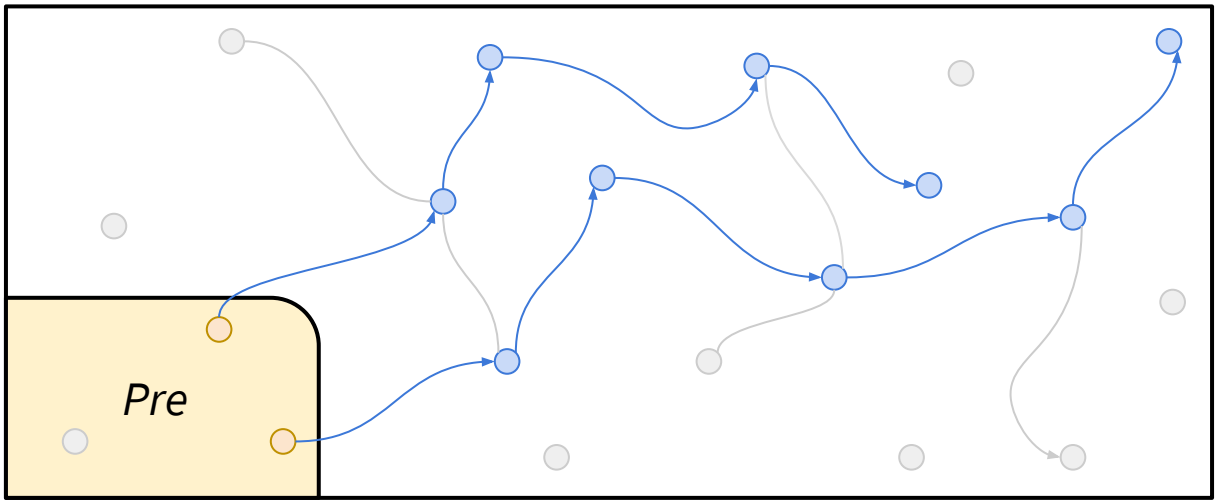
SyGuS Problem (Pre, Trans, Post) → List of variable assignments



1. Pick state s, s.t. Pre(s)
2. Obtain state t, s.t. Trans(s,t)
3. Set s ← t and repeat (2)

RECORD-ing Reachable States

SyGuS Problem (Pre, Trans, Post) → List of variable assignments



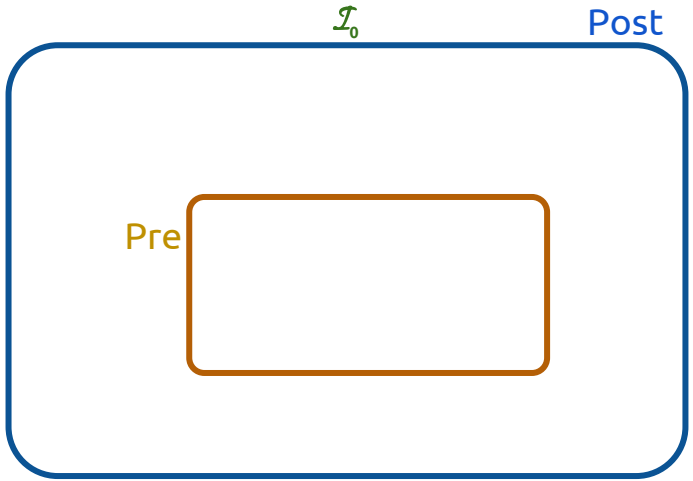
1. Pick state s, s.t. Pre(s) 2. Obtain state t, s.t. Trans(s,t) 3. Set s ← t and repeat (2)

4. Repeat (1,2,3) till the desired number of states has been collected

INFER-ing Sufficient Invariants

- ∀ s: Pre(s) ⇒ $\mathcal{I}$(s)
- ∀ s, t: $\mathcal{I}$(s) ∧ Trans(s,t) ⇒ $\mathcal{I}$(t)
- ∀ s: $\mathcal{I}$(s) ⇒ Post(s)

1. Start with the weakest candidate

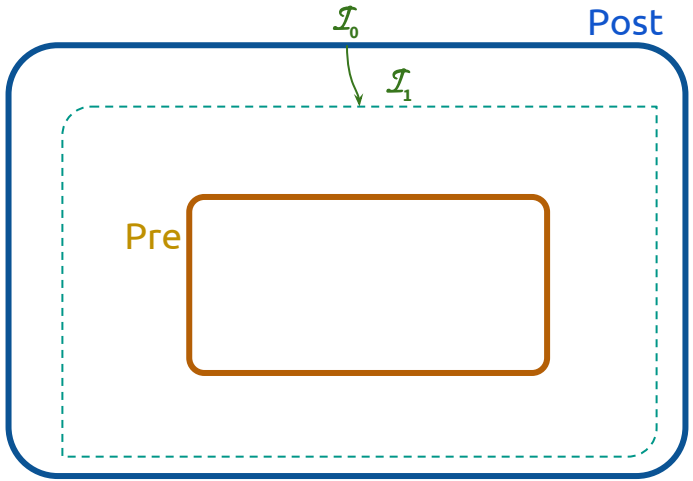


$\mathcal{I}_0$ = Post

INFER-ing Sufficient Invariants

- ∀ s: Pre(s) ⇒ $\mathcal{I}$(s)
- ∀ s,t: $\mathcal{I}$(s) ∧ Trans(s,t) ⇒ $\mathcal{I}$(t)
- ∀ s: $\mathcal{I}$(s) ⇒ Post(s)

1. Start with the weakest candidate
2. Iteratively strengthen for inductiveness (data-driven precondition inference)



$\mathcal{I}_0$ = Post

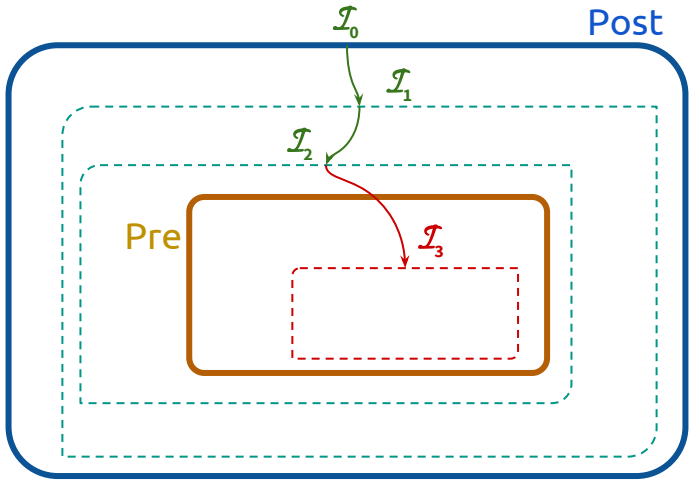
$\mathcal{I}_1 = \delta_0 \wedge \mathcal{I}_0$

$\delta_0 \Rightarrow (\mathcal{I}_0 \wedge \text{Trans} \Rightarrow \mathcal{I}_0')$

INFER-ing Sufficient Invariants

- ∀ s: Pre(s) ⇒ $\mathcal{I}$(s)
- ∀ s,t: $\mathcal{I}$(s) ∧ Trans(s,t) ⇒ $\mathcal{I}$(t)
- ∀ s: $\mathcal{I}$(s) ⇒ Post(s)

1. Start with the weakest candidate
2. Iteratively strengthen for inductiveness (data-driven precondition inference)



$\mathcal{I}_0$ = Post

$\delta_0 \Rightarrow (\mathcal{I}_0 \wedge \text{Trans} \Rightarrow \mathcal{I}_0')$

$\mathcal{I}_1 = \delta_0 \wedge \mathcal{I}_0$

$\delta_1 \Rightarrow (\mathcal{I}_1 \wedge \text{Trans} \Rightarrow \mathcal{I}_1')$

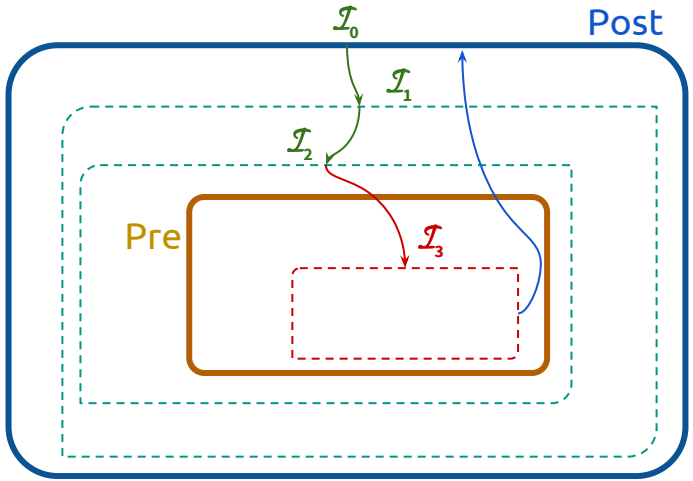
⋮ ⋮ ⋮ ⋮

$\mathcal{I}_n = \delta_{n-1} \wedge \mathcal{I}_{n-1} = \mathcal{I}_{n-1}$

INFER-ing Sufficient Invariants

- ∀ s: Pre(s) ⇒ $\mathcal{I}$(s)
- ∀ s,t: $\mathcal{I}$(s) ∧ Trans(s,t) ⇒ $\mathcal{I}$(t)
- ∀ s: $\mathcal{I}$(s) ⇒ Post(s)

1. Start with the weakest candidate
2. Iteratively strengthen for inductiveness (data-driven precondition inference)
3. If the invariant is too strong, restart from (1) after augmenting the recorded states with appropriate counterexamples



$\mathcal{I}_0$ = Post

$\delta_0 \Rightarrow (\mathcal{I}_0 \wedge \text{Trans} \Rightarrow \mathcal{I}_0')$

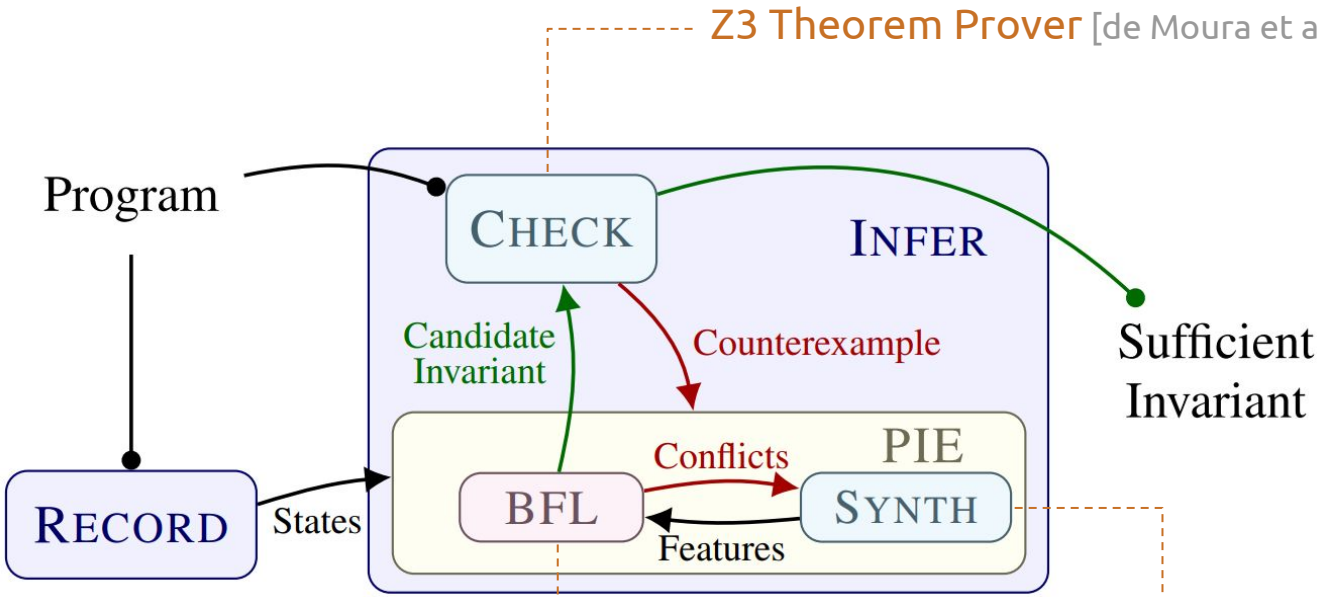
$\mathcal{I}_1 = \delta_0 \wedge \mathcal{I}_0$

$\delta_1 \Rightarrow (\mathcal{I}_1 \wedge \text{Trans} \Rightarrow \mathcal{I}_1')$

⋮ ⋮ ⋮ ⋮

$\mathcal{I}_n = \delta_{n-1} \wedge \mathcal{I}_{n-1} = \mathcal{I}_{n-1}$

LoopInvGen Architecture



Thanks! ☺

Code + Benchmarks:

https://github.com/SaswatPadhi/LoopInvGen

Reach me at:

padhi @ cs . ucla . edu