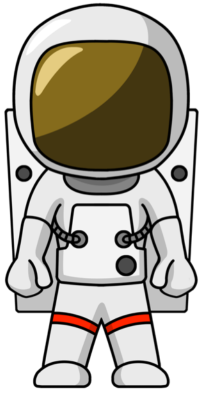
The Curse of Interpolation

Arie Gurfinkel

ANDREI-60
May 21, 2019

UNIVERSITY OF WATERLOO

SPACER: The Final Frontier



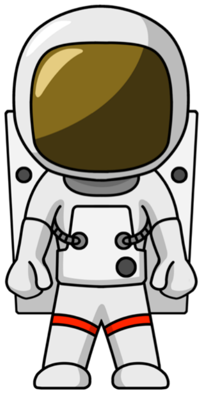
joint work with Nikolaj Bjorner, Anvesh Komuraveli, Sharon Shoham, Yakir Vizel, Hari Govind, Yu-Ting (Jeff) Chen, …

UNIVERSITY OF WATERLOO

Safety Property Verification of Programs / Transitions Systems / Push-down Systems

=

Satisfiability of Constrained Horn Logic (CHC) fragment of First Order Logic



Reduce Model Checking to FOL Satisfiability

UNIVERSITY OF WATERLOO

Constrained Horn Clauses (CHC)

A Constrained Horn Clause (CHC) is a FOL formula of the form

$$\forall V \cdot (\varphi \land p_1[X_1] \land \cdots \land p_n[X_n]) \to h[X]$$

where

- $\mathcal{T}$ is a background theory (e.g., Linear Arithmetic, Arrays, Bit-Vectors, or combinations of the above)
- V are variables, and X_i are terms over V
- φ is a constraint in the background theory $\mathcal{T}$
- p_1, …, p_n, h are n-ary predicates
- $p_i[X]$ is an application of a predicate to first-order terms

UNIVERSITY OF WATERLOO

CHC Satisfiability

A $\mathcal{T}$-**model** of a set of a CHCs Π is an extension of the model M of $\mathcal{T}$ with a first-order interpretation of each predicate p_i that makes all clauses in Π true in M

A set of clauses is **satisfiable** if and only if it has a model

- This is the usual FOL satisfiability

A $\boldsymbol{\mathcal{T}}$**-solution** of a set of CHCs Π is a substitution σ from predicates p_i to $\mathcal{T}$-formulas such that $\Pi\sigma$ is $\mathcal{T}$-valid

In the context of program verification

- a program satisfies a property iff corresponding CHCs are satisfiable
- **solutions** are **inductive invariants**
- refutation proofs are counterexample traces

Procedures for Solving CHC(T)

Predicate abstraction by lifting Model Checking to HORN

- QARMC, Eldarica, …

Maximal Inductive Subset from a finite Candidate space (Houdini)

- TACAS'18: hoice, FreqHorn

Machine Learning

- PLDI'18: sample, ML to guess predicates, DT to guess combinations

Abstract Interpretation (Poly, intervals, boxes, arrays…)

- Approximate least model by an abstract domain (SeaHorn, …)

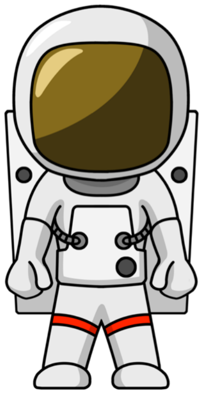
Interpolation-based Model Checking

- Duality, QARMC, …

SMT-based Unbounded Model Checking (IC3/PDR)

- Spacer, Implicit Predicate Abstraction

Spacer: Solving SMT-constrained CHC



Spacer: SAT procedure for SMT-constrained Horn Clauses

- now the default CHC solver in Z3
  - https://github.com/Z3Prover/z3
  - dev branch at https://github.com/agurfinkel/z3

Supported SMT-Theories

- Linear Real and Integer Arithmetic
- Quantifier-free theory of arrays
- Universally quantified theory of arrays + arithmetic
- Best-effort support for many other SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic

Support for Non-Linear CHC

- for procedure summaries in inter-procedural verification conditions
- for compositional reasoning: abstraction, assume-guarantee, thread modular, etc.

Program Verification with HORN(LIA)

```
z = x; i = 0;
assume (y > 0);
while (i < y) {
  z = z + 1;
  i = i + 1;
}
assert(z == x + y);
```

IS SAT?

```
z = x & i = 0 & y > 0                          → Inv(x, y, z, i)
Inv(x, y, z, i) & i < y & z1=z+1 & i1=i+1      → Inv(x, y, z1, i1)
Inv(x, y, z, i) & i >= y & z != x+y            → false
```

UNIVERSITY OF WATERLOO

In SMT-LIB

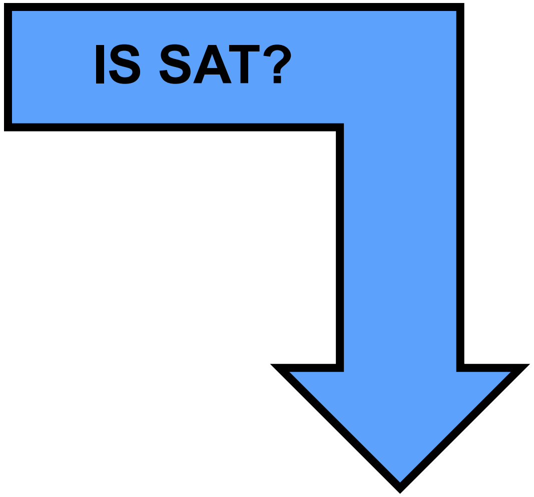
```
(set-logic HORN)

;; Inv(x, y, z, i)
(declare-fun Inv ( Int Int Int Int) Bool)

(assert
 (forall ( (A Int) (B Int) (C Int) (D Int))
        (=> (and (> B 0) (= C A) (= D 0))
           (Inv A B C D)))
 )
(assert
 (forall ( (A Int) (B Int) (C Int) (D Int) (C1 Int) (D1 Int) )
        (=>
         (and (Inv A B C D) (< D B) (= C1 (+ C 1)) (= D1 (+ D
1)))
         (Inv A B C1 D1)
         )
        )
 )
(assert
 (forall ( (A Int) (B Int) (C Int) (D Int))
        (=> (and (Inv A B C D) (>= D B) (not (= C (+ A B))))
           false
           )
        )
 )

(check-sat)
(get-model)
```

```
$ z3 add-by-one.smt2
sat
(model
  (define-fun Inv ((x!0 Int) (x!1 Int) (x!2 Int) (x!3 Int)) Bool
    (and (<= (+ x!2 (* (- 1) x!0) (* (- 1) x!3)) 0)
         (<= (+ x!2 (* (- 1) x!0) (* (- 1) x!1)) 0)
         (<= (+ x!0 x!3 (* (- 1) x!2)) 0)))
)
```

```
Inv(x, y, z, i)
    z  = x + i
    z <= x + y
```

HORN(ALIA): Arrays + LIA

```
int A[N];
for (int i = 0; i < N; ++i)
   A[i] = 0;
int j = nd();
assume(0 <= j < N);
assert(A[j] == 0);
```




```
Inv(A, N, 0)
Inv(A, N, i) & i < N ➔ Inv(A[i := 0], N, i+1)
Inv(A, N, i) & i >= N & 0 <= j < N & A[j] != 0 ➔ false
```

In SMT-LIB

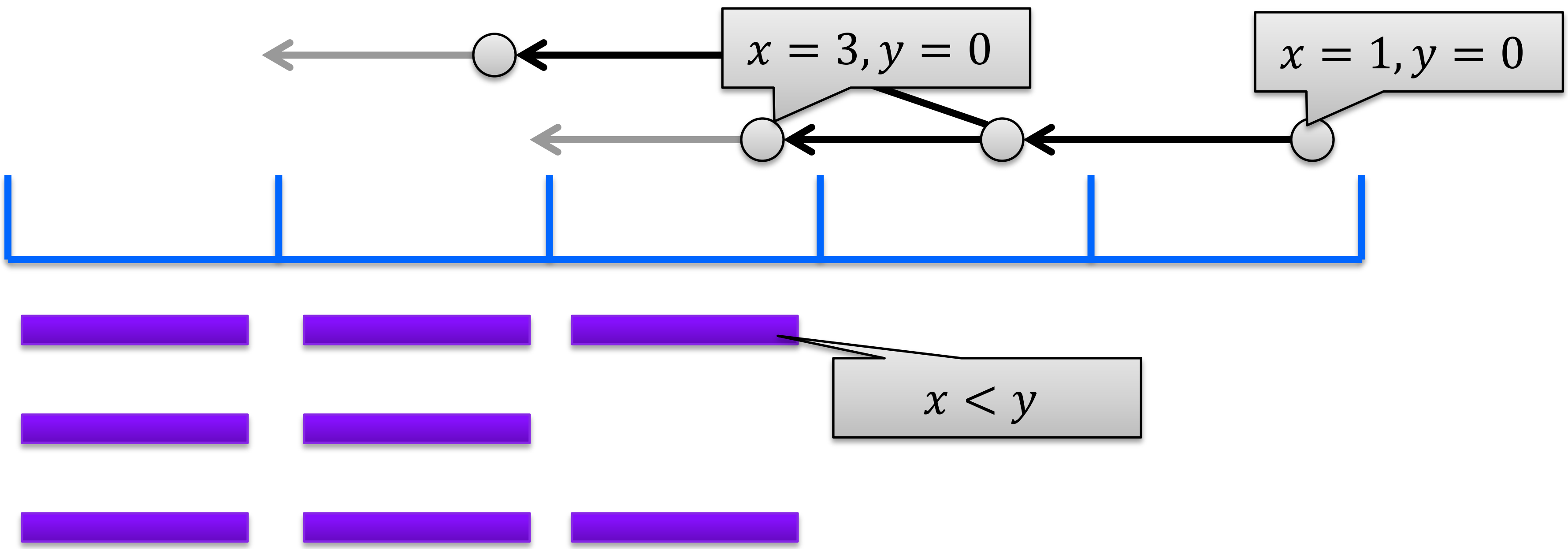
```
(set-logic HORN)

;; Inv(A, N, i)
(declare-fun Inv ( (Array Int Int) Int Int ) Bool)

(assert
 (forall ( (A (Array Int Int)) (N Int) (C Int)) (Inv A N 0)))

(assert
 (forall ( (A (Array Int Int)) (N Int) (i Int) )
         (=>
          (and (Inv A N i) (< i N) )
          (Inv (store A i 0) N (+ i 1))
          )
         )
 )
(assert
 (forall ( (A (Array Int Int)) (N Int) (i Int) (j Int))
         (=> (and (Inv A N i )
                 (>= i N) (<= 0 j) (< j N) (not (= (select A
j) 0)))
           false
           )
         )
 )

(check-sat)
(get-model)
```

```
$ z3 array-zero.smt2

sat
(model
  (define-fun Inv ((x!0 (Array Int Int)) (x!1 Int) (x!2 Int)) Bool
    (let ((a!1 (forall ((sk!0 Int))
                  (! (or (not (>= sk!0 0))
                         (>= (select x!0 sk!0) 0)
                         (<= (+ x!2 (* (- 1) sk!0)) 0))
                     :weight 15)))
          (a!2 (forall ((sk!0 Int))
                  (! (or (not (>= sk!0 0))
                         (<= (select x!0 sk!0) 0)
                         (<= (+ x!2 (* (- 1) sk!0)) 0))
                     :weight 15)))
)
```

```
Inv(A, N, i)

 ∀ 0 <= j < i < N →
                    A[j] = 0
```

IC3/PDR In Pictures: MkSafe

MkSafe



Predecessor

$$\text{find } M \text{ s.t. } M \models F_i \wedge Tr \wedge m'$$

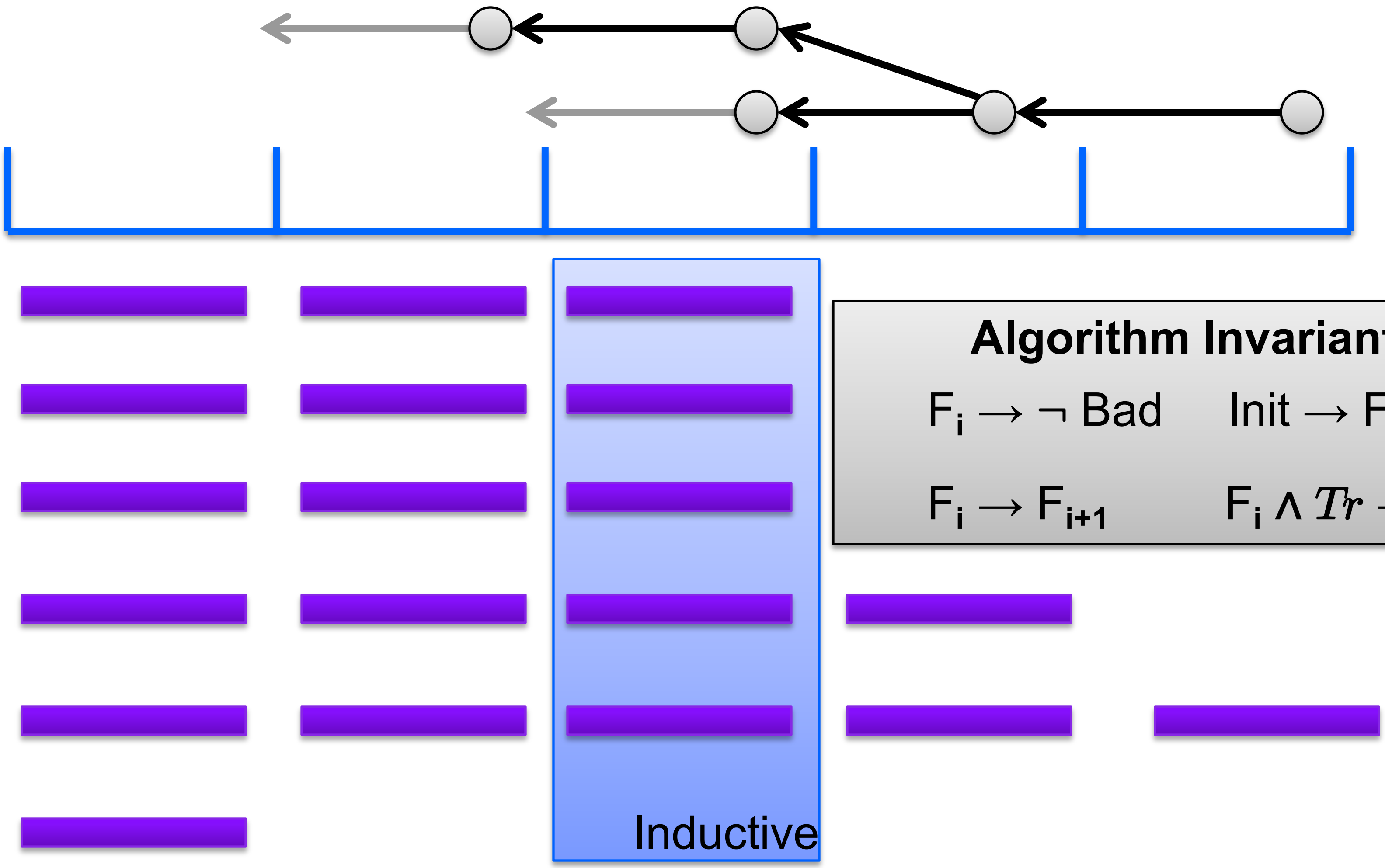
$$\text{find } m \text{ s.t. } (M \models m) \wedge (m \implies \exists V' \cdot Tr \wedge m')$$

NewLemma

$$\text{find } \ell \text{ s.t. } (F_i \wedge Tr \implies \ell') \wedge (\ell \implies \neg m)$$

IC3/PDR in Pictures: Push

Push



Algorithm Invariants

$F_i \rightarrow \neg$ Bad   Init $\rightarrow F_i$

$F_i \rightarrow F_{i+1}$ $F_i \wedge Tr \rightarrow F_{i+1}$

$$\text{SMT-query: } \vdash \ell \wedge F_i \wedge Tr \implies \ell'$$

UNIVERSITY OF WATERLOO

Predecessor and NewLemma rules in Spacer

Predecessor – generate a new predecessor of a given POB *m*

- Use SMT to check satisfiability of a transition relation with given pre- and post-conditions
- Use **Model-based Projection** to construct new POB over pre-variables only

$$\text{find } M \text{ s.t. } M \models F_i \wedge Tr \wedge m'$$

$$\text{find } m \text{ s.t. } (M \models m) \wedge (m \implies \exists V' \cdot Tr \wedge m')$$

NewLemma – create a new lemma that blocks a given POB *m*

- Use SMT to check unsatisfiability of a transition relation with a given pre- and post-conditions
- Use **Interpolation** to construct a new lemma

$$\text{find } \ell \text{ s.t. } (F_i \wedge Tr \implies \ell') \wedge (\ell \implies \neg m)$$

THE CURSE OF INTERPOLATION

UNIVERSITY OF WATERLOO

current work with Hari Govind and Yu-Ting (Jeff) Chen

The Curse of Interpolation

Interpolation is capable of generating many interesting terms

- (almost) any inductive invariant is an interpolant of something under the right conditions!

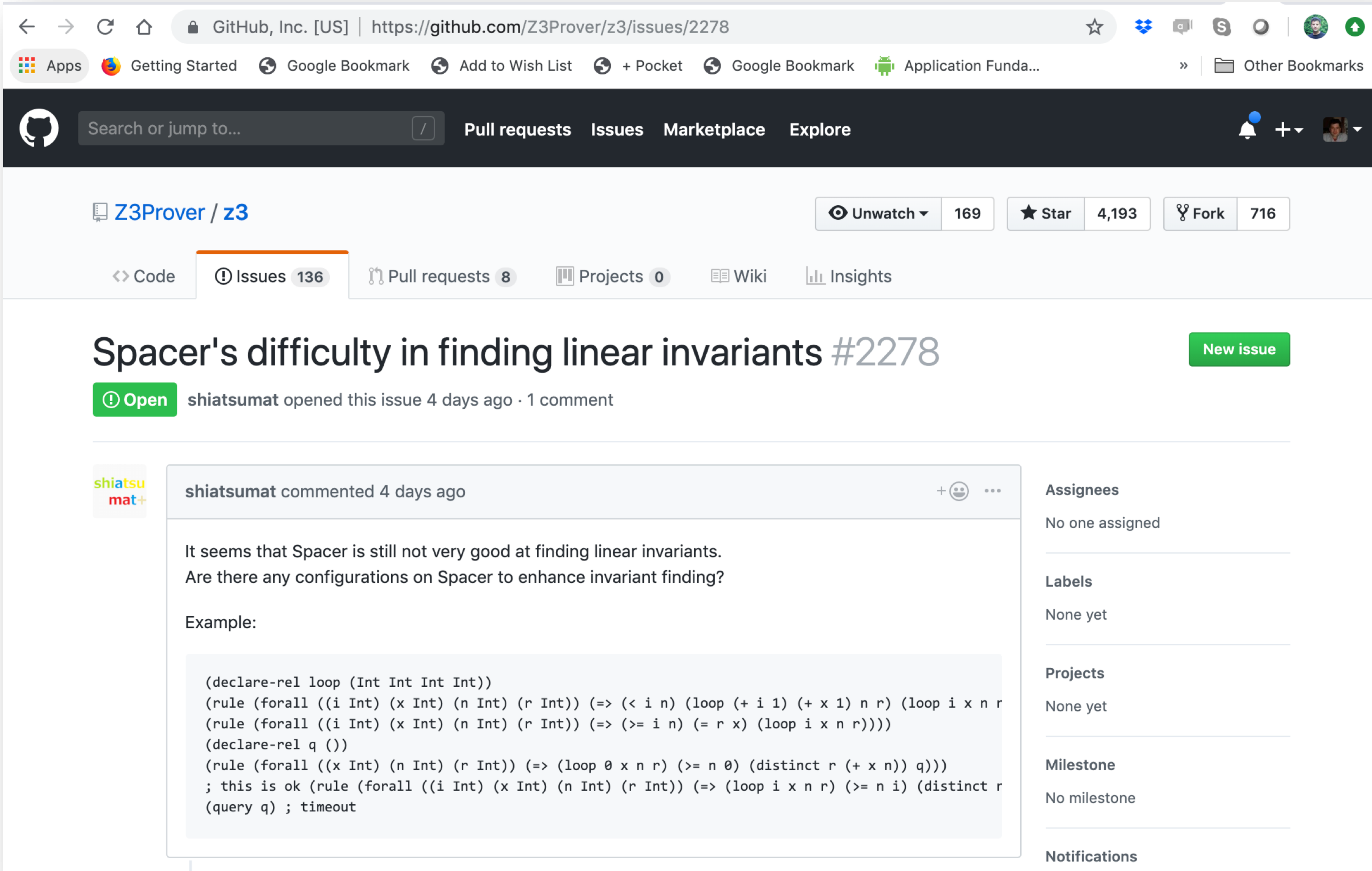
Interpolation often works in practice

- creates false sense of security
- predicate / term generation is a solved problem

But, interpolation is very hard to control!

- Small changes to input result in big change in interpolants
- Small changes to solver parameter result in big change in interpolants
- Works well overall (i.e., large benchmark set), but poorly for any given user problem!

UNIVERSITY OF WATERLOO



Z3Prover / z3

Unwatch 169 | Star 4,193 | Fork 716

Code | Issues 136 | Pull requests 8 | Projects 0 | Wiki | Insights

Spacer's difficulty in finding linear invariants #2278

New issue

Open **shiatsumat** opened this issue 4 days ago · 1 comment

shiatsumat commented 4 days ago

It seems that Spacer is still not very good at finding linear invariants.
Are there any configurations on Spacer to enhance invariant finding?

Example:

```
(declare-rel loop (Int Int Int Int))
(rule (forall ((i Int) (x Int) (n Int) (r Int)) (=> (< i n) (loop (+ i 1) (+ x 1) n r) (loop i x n r
(rule (forall ((i Int) (x Int) (n Int) (r Int)) (=> (>= i n) (= r x) (loop i x n r))))
(declare-rel q ())
(rule (forall ((x Int) (n Int) (r Int)) (=> (loop 0 x n r) (>= n 0) (distinct r (+ x n)) q)))
; this is ok (rule (forall ((i Int) (x Int) (n Int) (r Int)) (=> (loop i x n r) (>= n i) (distinct r
(query q) ; timeout
```

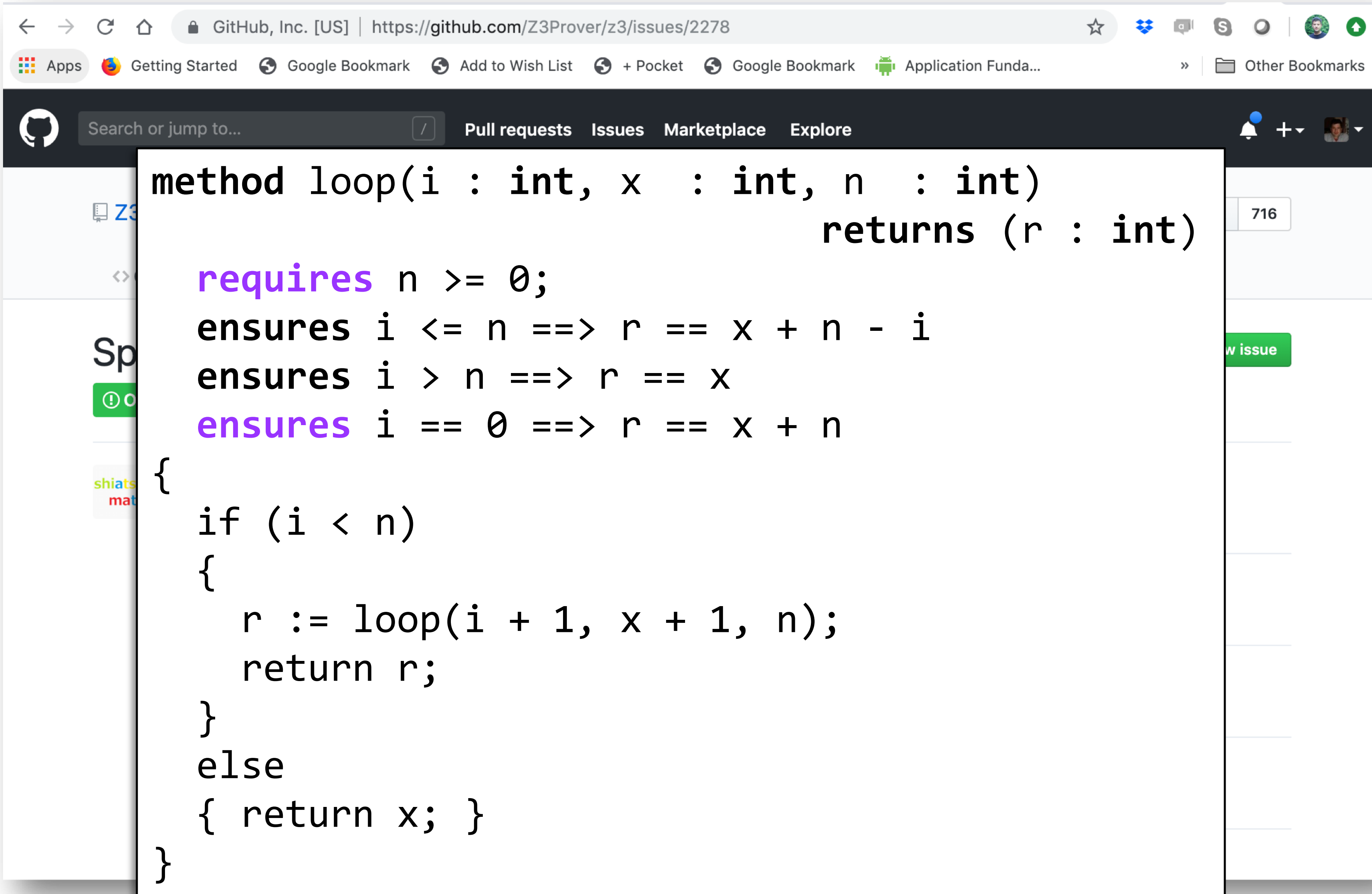
Assignees: No one assigned

Labels: None yet

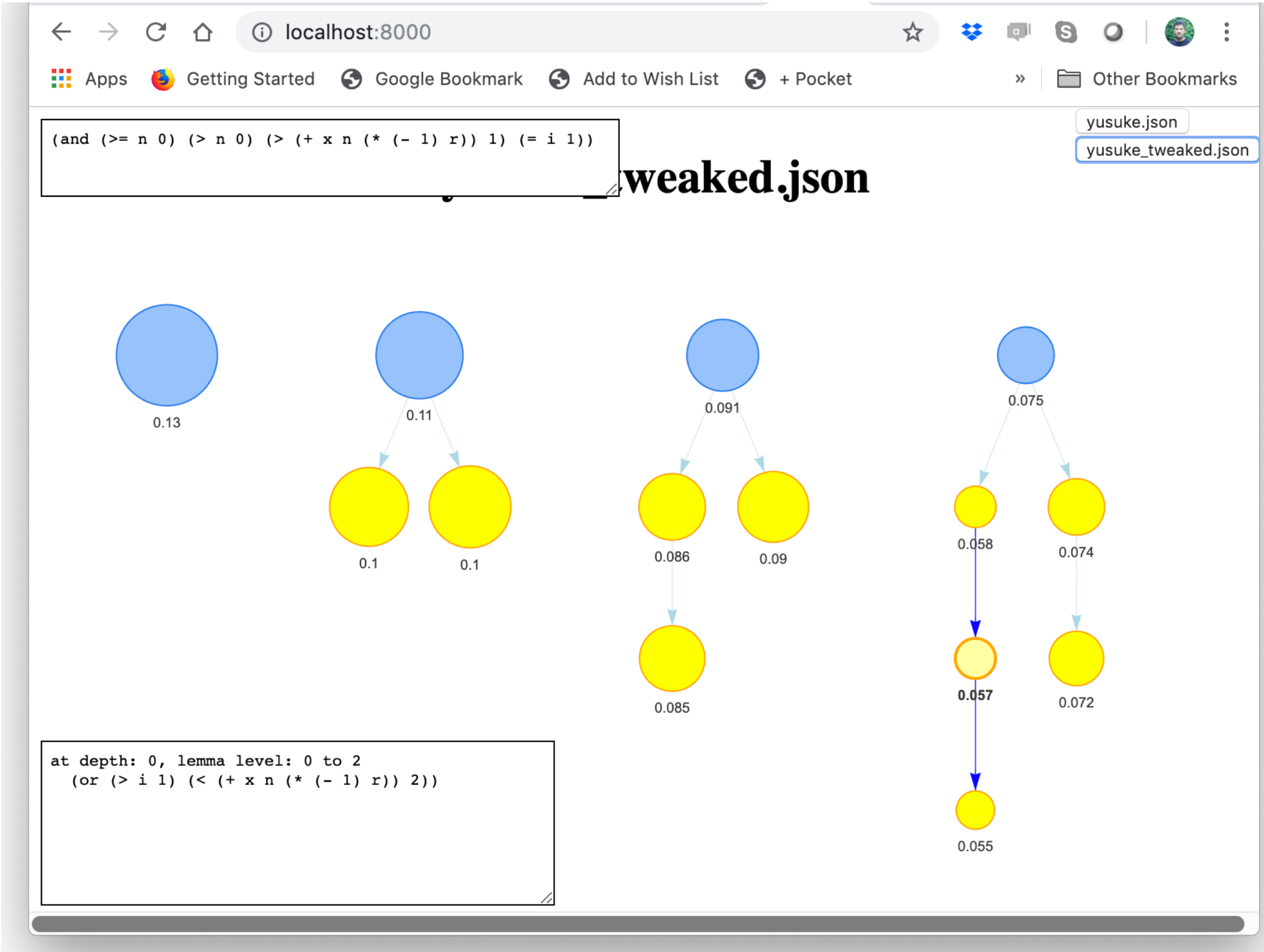
Projects: None yet

Milestone: No milestone

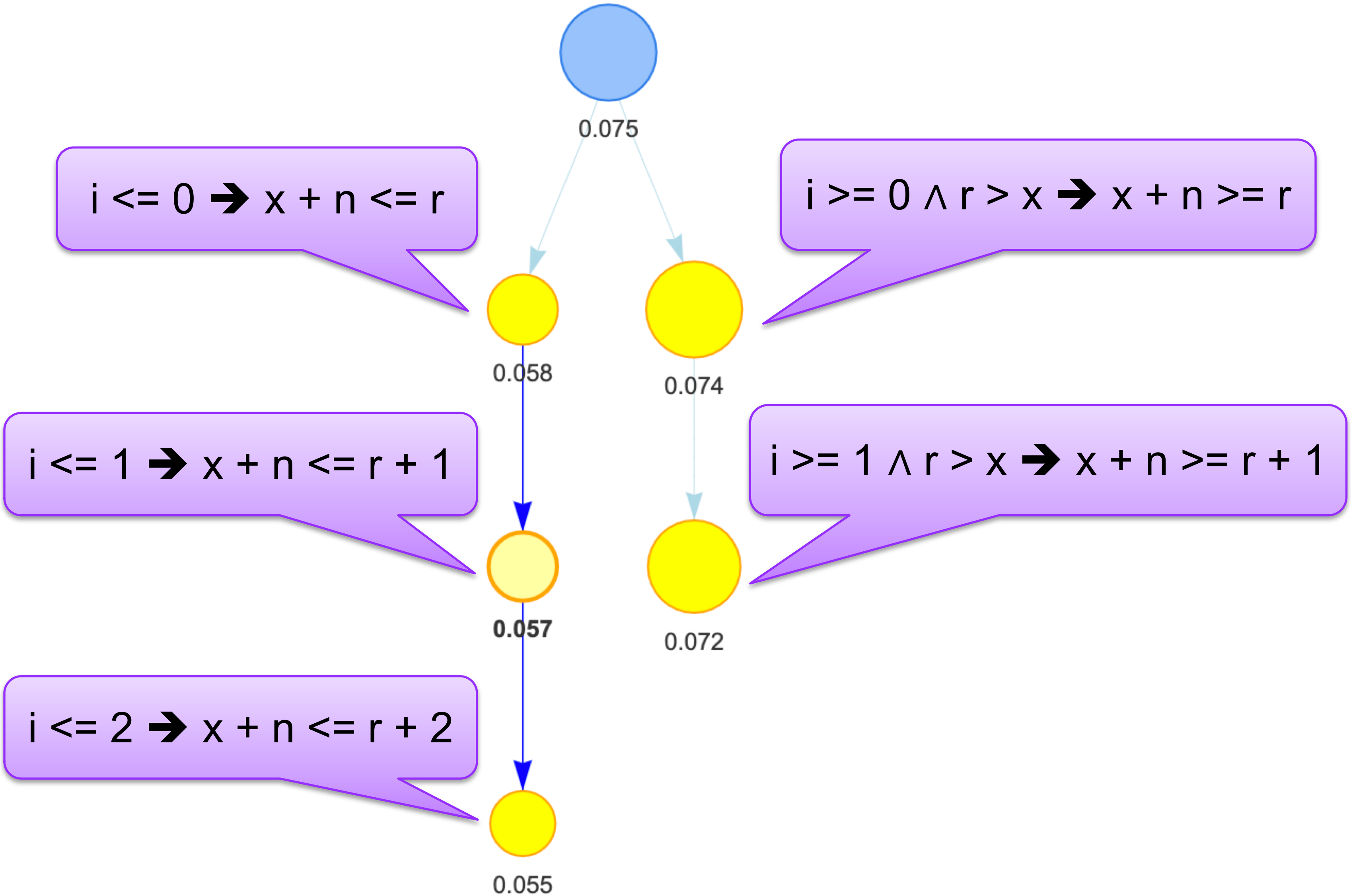
Notifications




```
method loop(i : int, x  : int, n  : int)
                                  returns (r : int)
  requires n >= 0;
  ensures i <= n ==> r == x + n - i
  ensures i > n ==> r == x
  ensures i == 0 ==> r == x + n
{
  if (i < n)
  {
    r := loop(i + 1, x + 1, n);
    return r;
  }
  else
  { return x; }
}
```



UNIVERSITY OF WATERLOO



0.075
i <= 0 ➔ x + n <= r
i >= 0 ∧ r > x ➔ x + n >= r
0.058
0.074
i <= 1 ➔ x + n <= r + 1
i >= 1 ∧ r > x ➔ x + n >= r + 1
0.057
0.072
i <= 2 ➔ x + n <= r + 2
0.055

Data Driven Generalization & Lemma Discovery

Global view of the current solver state

- **group** lemmas (and pobs) based on syntactic/semantic similarity
  - we currently use anti-unification on interpreted constants
- detect whenever global proof is diverging and mitigate

One lemma to rule them all

- **merge** lemmas in group to form a single *universal* lemma
- interpolation and inductive generalization can be applied to generalize further
- new lemma reduces the global proof by blocking all POBs in its group

Reduce, reuse, recycle

- under-approximate groups that cannot be merged in current theory
- learn multiple (simple) lemmas to block a (complex) pob

$i < 0 \rightarrow x + n <= r + 0$

Lemma 1

$i < 1 \rightarrow x + n <= r + 1$

Lemma 2

$0 <= \boldsymbol{v} <= 1 \rightarrow (i < \boldsymbol{v} \rightarrow x + n <= r + \boldsymbol{v})$

Group 1

$x + n <= r + i$

Generalized Lemma

UNIVERSITY OF WATERLOO

$i < 0 \Rightarrow x + n <= r + 0$

$r > x \wedge i >= 0 \Rightarrow r + 0 <= x + n$

$i < 1 \Rightarrow x + n <= r + 1$

$r > x \wedge i >= 1 \Rightarrow r + 1 <= x + n$

$0 <= \mathbf{\textit{v}} <= 1 \Rightarrow$

$(i < \mathbf{\textit{v}} \Rightarrow x + n <= r +$

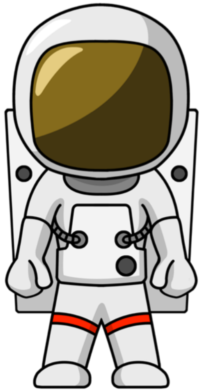
$0 <= \mathbf{\textit{v}} <= 1 \Rightarrow$

$r > x \wedge i >= \mathbf{\textit{v}} \Rightarrow r + \mathbf{\textit{v}} <= x + n$

$x + n <= r + i$

$r > x \Rightarrow r + i <= x + n$

Conclusion



Verification of Safety Properties is FOL satisfiability

- Logic: Constrained Horn Clauses (CHC)
- “Decision” procedure: Spacer

The Curse of Interpolation

- Interpolation can be amazing at guessing required terms
- but, is hard to control and masks the underlying problem!

Data driven generalization

- supplement interpolation with data-driven learning
- global view of the overall proof process
- identify diverging patterns / groups
- generalize lemmas based on groups

?

?

?

?

?

?

THE END