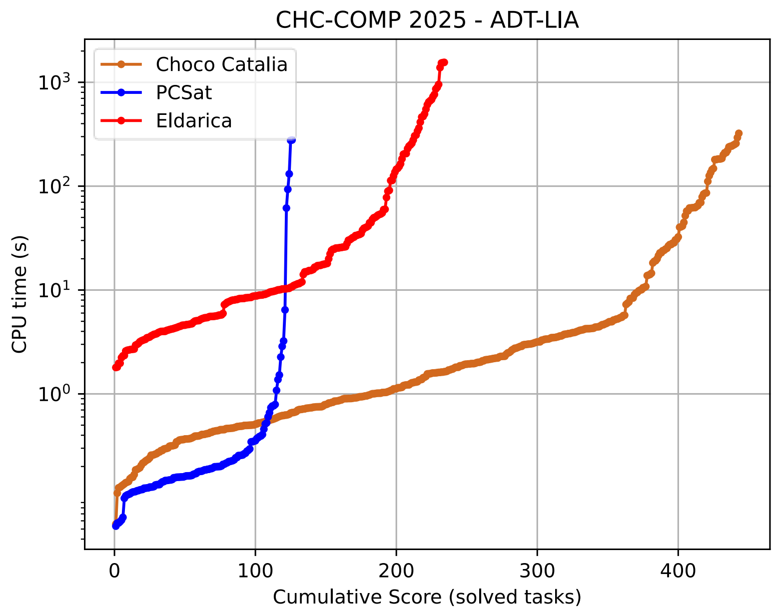
Solvable Tuple Patterns and Their Applications to Program Verification

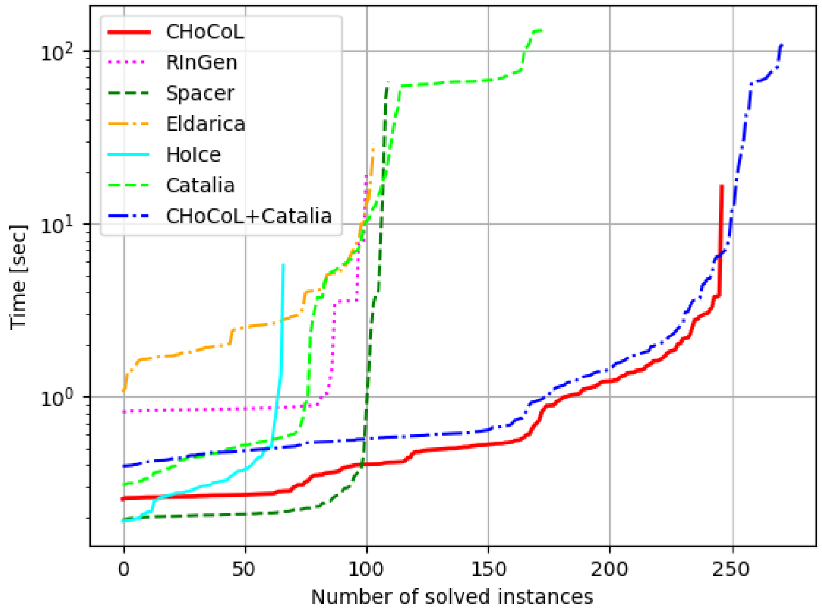
Naoki Kobayashi
The University of Tokyo
(joint work with Ryosuke Sato, Ayumi Shinohara, and Ryo Yoshinaka)

This Work

- **Solvable Tuple Patterns (STPs) for Lightweight Inference of Relations on Sequences**
  (demo page: https://www-kb.is.s.u-tokyo.ac.jp/~koba/tupinf)
  - **Learnable from only a few positive samples**
  - **Efficient inference algorithm**
- **Applications to CHCs over List-like Data Structures**



For List-only benchmark from CHC-COMP2025



Outline

- ♦ Motivations
- ♦ Solvable Tuple Patterns
- ♦ Applications to Program Verification
- ♦ Implementation and Experiments
- ♦ Related Work
- ♦ Conclusion

Motivating Example

```
let rec reva l1 l2 =
    match l1 with  [] -> l2
                 | x::l1' -> reva l1' (x::l2)
let main l1 l2 =
    assert(reva (reva l1 l2) [] = reva l2 l1)
```

How to automatically verify that the assertion never fails?

- Naïve induction (on l1 or l2) would not work.
- The invariant "reva l1 l2 = $l1^R$ l2" would be useful,

  but how can we find it automatically?

  => data-driven approach

Lightweight Inference of Invariants?

- **Learning Numerical Equality Invariants** [Sharma+ ESOP13][Ikeda+ APLAS 23]

```
int x = 0, y = 0, z = 0;
while(x < 500) {
  y += x; z += x + 2; x += 1;
};
assert(z >= y + 1000);
```

x	y	z
0	0	0
1	0	2
2	1	5
3	3	9
…	…	…

Loop invariant among x, y, z?

Lightweight Inference of Invariants?

- **Learning Numerical Equality Invariants [Sharma+ ESOP13][Ikeda+ APLAS 23]**

```
int x = 0, y = 0, z = 0;
while(x < 500) {
  y += x; z += x + 2; x += 1;
};
assert(z >= y + 1000);
```

x	y	z
0	0	0
1	0	2
2	1	5
3	3	9

Loop invariant among x, y, z?

Lightweight Inference of Invariants?

- **Learning Numerical Equality Invariants** **[Sharma+ ESOP13][Ikeda+ APLAS 23]**

```
int x = 0, y = 0, z = 0;
while(x < 500) {
  y += x; z += x + 2; x += 1;
};
assert(z >= y + 1000);
```

x	y	z-2x
0	0	0
1	0	0
2	1	1
3	3	3

Loop invariant among x, y, z?

Lightweight Inference of Invariants?

- **Learning Numerical Equality Invariants** [Sharma+ ESOP13][Ikeda+ APLAS 23]

```
int x = 0, y = 0, z = 0;
while(x < 500) {
   y += x; z += x + 2; x += 1;
};
assert(z >= y + 1000);
```

Loop invariant among x, y, z?

x	y	z-2x-y
0	0	0
1	0	0
2	1	0
3	3	0

Invariant:
$z-2x-y = 0$

At the assertion,
$z=2x+y \geq 2*500+y = y+1000$

Lightweight Inference of Invariants?

- **Learning Numerical Equality Invariants** **[Sharma+ ESOP13][Ikeda+ APLAS 23]**

```
int x = 0, y = 0, z = 0;
while(x < 500) {
  y += x; z += x + 2; x += 1;
};
assert(z >= y + 1000);
```

x	y	z-2x-y
0	0	0
1	0	0
2	1	0
3	3	0

Advantages:
- Learnable from only a few positive samples
- Positive samples are easy to collect

=> Is a similar method possible for inference of relations on recursive data structures?

Invariant:
$z-2x-y = 0$

At the assertion,
$z=2x+y \geq 2*500+y = y+1000$

Outline

- ♦ Motivations
- ♦ Solvable Tuple Patterns
  - – tuple patterns
  - – tuple pattern inference
  - – solvable tuple patterns
  - – extensions
- ♦ Applications to Program Verification
- ♦ Implementation and Experiments
- ♦ Related Work
- ♦ Conclusion

Tuple Patterns

t (tuple patterns) ::= $(p_1, \ldots, p_n)$ $\qquad$ **$p_i \in (\Sigma \cup \text{Vars})^*$**

$L(t) = \{[s_1/x_1, \ldots, s_k/x_k]t \mid s_i \in \Sigma^*\}$

Examples:

- **(x, y, xy) : append (or concatenation) relation**
- **(xy, yz, xyz) :**

  the first (second, rep.) element is a prefix (postfix, resp.) of the third element, with y being the overlapping part.

cf. Anguluin's pattern languages [1980] ("identifiable in the limit" [Gold67] from only positive samples) and its variants [Shinohara 1983]

Tuple Pattern Inference

x	y
aa	aaa
abb	abbab
ba	ba

Tuple Pattern Inference

x	y		x	y′
aa	aaa	$y = xy'$	aa	a
abb	abbab	$\longrightarrow$	abb	ab
ba	ba		ba	ε

Tuple Pattern Inference

x	y		x	y'		x'	y'
aa	aaa	$\xrightarrow{y = xy'}$	aa	a	$\xrightarrow{x = y'x'}$	a	a
abb	abbab		abb	ab		b	ab
ba	ba		ba	ε		ba	ε

$$(x, y) = (x, xy') = (y'x', y'x'y')$$

$$\frac{M[*][i] \neq \widetilde{\epsilon} \qquad M[*][j] = M[*][i] \cdot \widetilde{s} \qquad x'_j \text{ fresh}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([x_i \cdot x'_j/x_j]t, [M\{j \mapsto \widetilde{s}\}/(x_1, \ldots, x_{j-1}, x'_j, x_{j+1}, \ldots, x_m)])}$$

$$\frac{M[*][j] = a \cdot \widetilde{s} \qquad a \in \Sigma \qquad x'_j \text{ fresh}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([ax'_j/x_j]t, [M\{j \mapsto \widetilde{s}\}/(x_1, \ldots, x_{j-1}, x'_j, x_{j+1}, \ldots, x_m)])}$$

$$\frac{M[*][j] = \widetilde{\epsilon}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([\epsilon/x_j]t, [M \uparrow_j /(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_m)])}$$

Non-determinism of Inference Algorithm

$$((x, y, z), \begin{pmatrix} x & y & z \\ aa & a & aac \\ b & bb & bbd. \end{pmatrix}) \longrightarrow ((x, y, xz'), \begin{pmatrix} x & y & z' \\ aa & a & c \\ b & bb & bd. \end{pmatrix})$$

$$((x, y, z), \begin{pmatrix} x & y & z \\ aa & a & aac \\ b & bb & bbd. \end{pmatrix}) \longrightarrow ((x, y, yz'), \begin{pmatrix} x & y & z' \\ aa & a & ac \\ b & bb & d \end{pmatrix}).$$

Conjunctive tuple patterns? (left for future work): $(x, y, xz) \wedge (x, y, yz)$

Properties of the Algorithm

THEOREM 1.1 (SOUNDNESS). *If* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta)$, *then* $M, \Theta \models t$.

THEOREM 1.2. *Given* M *as input,* t *such that* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta) \not\longrightarrow$ *can be computed in polynomial time.*

Note: Not all tuple patterns can be inferred.
For example, the singleton pattern (xx) cannot be inferred.

Outline

- ♦ Motivations
- ♦ Solvable Tuple Patterns
  - tuple patterns
  - tuple pattern inference
  - solvable tuple patterns
  - extensions
- ♦ Applications to Program Verification
- ♦ Implementation and Experiments
- ♦ Related Work
- ♦ Conclusion

Solvable Tuple Patterns

Definition 1.3. A tuple pattern t is solvable if $t \rightsquigarrow^* (x_1, \ldots, x_k)$.

$$\frac{p_j = p_i \cdot p'_j \qquad p_i \neq \epsilon}{(p_1, \ldots, p_n) \rightsquigarrow (p_1, \ldots, p_{j-1}, p'_j, p_{j+1}, \ldots, p_n)}$$

$$\frac{p_j = a \cdot p'_j \qquad a \in \Sigma}{(p_1, \ldots, p_n) \rightsquigarrow (p_1, \ldots, p_{j-1}, p'_j, p_{j+1}, \ldots, p_n)}$$

$$\frac{p_j = \epsilon}{(p_1, \ldots, p_n) \rightsquigarrow (p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_n)}$$

Example:
(xyx) is not solvable,
but (xyx, xy) is solvable,
because:
(xyx,xy) -> (x, xy) -> (x,y)

Why “Solvable”?

- Equation $t = (s_1, \ldots, s_n)$ can be “solved” if t is an STP (cf. solvable polynomial equations)

  e.g.

  – $(xy, xyx) = (s_1, s_2)$ has a general solution:

  $$x = s_1 \diagdown s_2, \quad y = (s_1 \diagdown s_2) \diagdown s_1$$

  – $(x, yy, xy) = (s_1, s_2, s_3)$ has a general solution:

  $$x = s_1, \quad y = s_1 \diagdown s_3$$

  just if $s_2 = (s_1 \diagdown s_3)\,(s_1 \diagdown s_3)$

Properties of the Algorithm (2)

THEOREM 1.4. *If* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta)$, *then* t *is solvable.*

Properties of the Algorithm (2)

Theorem 1.4. *If* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta)$, *then* t *is solvable.*

Theorem 1.5 (completeness). *If* $M \models_s t$ *and* t *is solvable, then* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta)$ *for some* Θ.

Properties of the Algorithm (2)

THEOREM 1.4. *If* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta)$, *then* t *is solvable.*

THEOREM 1.5 (COMPLETENESS). *If* $M \models_s t$ *and* t *is solvable, then* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta)$ *for some* Θ.

THEOREM 1.6 (MINIMALITY). *Suppose* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t_1, \Theta_1) \not\longrightarrow$. *If* $M \models t_0$ *and* $\mathcal{L}(t_0) \subseteq \mathcal{L}(t_1)$ *for an STP* t_0, *then* $\mathcal{L}(t_0) = \mathcal{L}(t_1)$.

Properties of the Algorithm (2)

THEOREM 1.4. *If* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta)$, *then* t *is solvable.*

THEOREM 1.5 (COMPLETENESS). *If* $M \models_s t$ *and* t *is solvable, then* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta)$ *for some* Θ.

THEOREM 1.6 (MINIMALITY). *Suppose* $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t_1, \Theta_1) \not\longrightarrow$. *If* $M \models t_0$ *and* $\mathcal{L}(t_0) \subseteq \mathcal{L}(t_1)$ *for an STP* t_0, *then* $\mathcal{L}(t_0) = \mathcal{L}(t_1)$.

THEOREM 1.7. *If* t *be an STP, then there exists* M *such that*
(i) The size of M *is* $O(n \log n)$ *where* n *is the size of* t;
(ii) $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t, \Theta) \not\longrightarrow$ *for some* Θ; *and*
(iii) $(\widetilde{x}, [M/\widetilde{x}]) \longrightarrow^* (t', \Theta') \not\longrightarrow$ *implies* $\mathcal{L}(t) = \mathcal{L}(t')$.

Corollary: STPs are “identifiable in the limit [Gold67]” from positive samples

Learnability and Complexity Results

	STP	NE-patterns [Angluin 80]	E-patterns [Shinohara 83]
Learnability	Yes	Yes	No for $\lvert\Sigma\rvert$=2,3,4 Yes for $\lvert\Sigma\rvert$=1,∞ Open for other cases
Membership	P	NP-complete	NP-complete
Inclusion	P	Undecidable	Undecidable
Equivalence	P	P	Open

Outline

- ♦ Motivations
- ♦ Solvable Tuple Patterns
  - – tuple patterns
  - – tuple pattern inference
  - – solvable tuple patterns
  - – extensions
- ♦ Applications to Program Verification
- ♦ Implementation and Experiments
- ♦ Related Work
- ♦ Conclusion

Postfix

$$\frac{M[*][i] \neq \widetilde{\epsilon} \qquad M[*][j] = \widetilde{s} \cdot M[*][i] \qquad x_j' \text{ fresh}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([x_j' \cdot x_i/x_j]t, [M\{j \mapsto \widetilde{s}\}/(x_1, \ldots, x_{j-1}, x_j', x_{j+1}, \ldots, x_m)])}$$

$$\frac{M[*][j] = \widetilde{s} \cdot a \qquad a \in \Sigma \qquad x_j' \text{ fresh}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([x_j' a/x_j]t, [M\{j \mapsto \widetilde{s}\}/(x_1, \ldots, x_{j-1}, x_j', x_{j+1}, \ldots, x_m)])}$$

$$\frac{p_j = p_j' \cdot p_i \qquad p_i \neq \epsilon}{(p_1, \ldots, p_n) \rightsquigarrow (p_1, \ldots, p_{j-1}, p_j', p_{j+1}, \ldots, p_n)}$$

$$\frac{p_j = p_j' \cdot a \qquad a \in \Sigma}{(p_1, \ldots, p_n) \rightsquigarrow (p_1, \ldots, p_{j-1}, p_j', p_{j+1}, \ldots, p_n)}$$

Reverse

$$p ::= a \mid x \mid p_1 p_2 \mid x^R.$$

$$\frac{M[*][i] \neq \widetilde{\epsilon} \qquad M[*][j] = M[*][i]^R \cdot \widetilde{s} \qquad x'_j \text{ fresh}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([x_i^R \cdot x'_j / x_j]t, [M\{j \mapsto \widetilde{s}\}/(x_1, \ldots, x_{j-1}, x'_j, x_{j+1}, \ldots, x_m)])}$$

$$\frac{M[*][i] \neq \widetilde{\epsilon} \qquad M[*][j] = \widetilde{s} \cdot M[*][i]^R \qquad x'_j \text{ fresh}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([x'_j \cdot x_i^R / x_j]t, [M\{j \mapsto \widetilde{s}\}/(x_1, \ldots, x_{j-1}, x'_j, x_{j+1}, \ldots, x_m)])}$$

$$\frac{p_j = p_i^R \cdot p'_j \qquad p_i \neq \epsilon}{(p_1, \ldots, p_n) \rightsquigarrow (p_1, \ldots, p_{j-1}, p'_j, p_{j+1}, \ldots, p_n)}$$

$$\frac{p_j = p'_j \cdot p_i^R \qquad p_i \neq \epsilon}{(p_1, \ldots, p_n) \rightsquigarrow (p_1, \ldots, p_{j-1}, p'_j, p_{j+1}, \ldots, p_n)}$$

Set/Multiset Patterns

$$\frac{M[*][i] \neq \widetilde{\epsilon} \qquad \forall k. M[k][j] \supseteq M[k][i] \qquad \widetilde{s} = M[*][j] \setminus M[*][i] \qquad x'_j \text{ fresh}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([x_i x'_j / x_j]t, [M\{j \mapsto \widetilde{s}\}/(x_1, \ldots, x_{j-1}, x'_j, x_{j+1}, \ldots, x_m)])}$$

$$\frac{M[*][j] = \widetilde{\emptyset}}{(t, [M/(x_1, \ldots, x_m)]) \longrightarrow ([\epsilon/x_j]t, [M\uparrow_j /(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_m)])}$$

$$((x_1, x_2, x_3), \begin{pmatrix} x_1 & x_2 & x_3 \\ \{a\} & \{b\} & \{a, b\} \\ \{b\} & \{b, c\} & \{b, c\} \end{pmatrix}) \longrightarrow ((x_1, x_2, x_1 x'_3), \begin{pmatrix} x_1 & x_2 & x'_3 \\ \{a\} & \{b\} & \{b\} \\ \{b\} & \{b, c\} & \{c\} \end{pmatrix})$$

$$\longrightarrow ((x_1, x'_3 x'_2, x_1 x'_3), \begin{pmatrix} x_1 & x'_2 & x'_3 \\ \{a\} & \emptyset & \{b\} \\ \{b\} & \{b\} & \{c\} \end{pmatrix}) \longrightarrow ((x'_2 x'_1, x'_3 x'_2, x'_2 x'_1 x'_3), \begin{pmatrix} x'_1 & x'_2 & x'_3 \\ \{a\} & \emptyset & \{b\} \\ \emptyset & \{b\} & \{c\} \end{pmatrix})$$

Outline

- ♦ Motivations
- ♦ Solvable Tuple Patterns
- ♦ Applications to Automated Program Verification
  - – Program Verification via CHCs
  - – CHC solving via SPT inference
  - – Combination with CHC solving over integer arithmetic
  - – Applications of Set/Multiset Patterns
- ♦ Implementation and Experiments
- ♦ Related Work
- ♦ Conclusion

Program Verification via CHC Solving

```
let rec reva l1 l2 =
  match l1 with
       [] -> l2
     | x::l1' -> reva l1' (x::l2)
let main l1 l2 =
 assert
  (reva (reva l1 l2) [] = reva l2 l1)
```

May the assertion fail?

$$Reva(\epsilon, l_2, l_2).$$

$$Reva(l'_1, x \cdot l_2, l_3) \Rightarrow Reva(x \cdot l'_1, l_2, l_3).$$

$$Reva(l_1, l_2, l_3) \land Reva(l_3, \epsilon, l_4) \land Reva(l_2, l_1, l_5) \Rightarrow l_4 = l_5.$$

($Reva(l_1,l_2,l_3)$ ⇔ reva l_1 l_2 may return l_3)

Are the CHCs satisfiable?

Yes: $Reva(l_1,l_2,l_3) \equiv l_3 = l_1^R\, l_2$

CHC Solving via STP Inference (Example)

Samples:

l_1	l_2	l_3
ε	a	a
ε	bc	bc

↓ STP inference

(ε, l, l)

↓

$$\forall x, l_1', l_2, l_3. l_1' = \epsilon \land x \cdot l_2 = l_3 \Rightarrow x \cdot l_1' = \epsilon \land l_2 = l_3?$$

↓

counterexample:

x=a, l1' = l_2= l_3 = ε, … Reva(xl1', l2, l3) = Reva(a, ε, a) does not hold

$$Reva(\epsilon, l_2, l_2).$$

$$Reva(l_1', x \cdot l_2, l_3) \Rightarrow Reva(x \cdot l_1', l_2, l_3).$$

$$Reva(l_1, l_2, l_3) \land Reva(l_3, \epsilon, l_4)$$

$$\land Reva(l_2, l_1, l_5) \Rightarrow l_4 = l_5.$$

CHC Solving via STP Inference (Example)

Samples:

l_1	l_2	l_3
ε	a	a
ε	bc	bc
a	ε	a

↓

(x, y, xy)

↓

$$\forall x, l_1', l_2, l_3. l_1' x l_2 = l_3 \Rightarrow x l_1' l_2 = l_3?$$

↓

counterexample:

x=a, l1'=b, l_2= ε, l_3 = ba   Reva(xl1', l2, l3) = Reva(ab, ε, ba) does not hold

$$Reva(\epsilon, l_2, l_2).$$

$$Reva(l_1', x \cdot l_2, l_3) \Rightarrow Reva(x \cdot l_1', l_2, l_3).$$

$$Reva(l_1, l_2, l_3) \wedge Reva(l_3, \epsilon, l_4)$$

$$\wedge\ Reva(l_2, l_1, l_5) \Rightarrow l_4 = l_5.$$

CHC Solving via STP Inference (Example)

Samples:

l_1	l_2	l_3
ε	a	a
ε	bc	bc
a	ε	a
ab	ε	ba

⬇

(x, y, x^Ry)

⬇

$$\forall x, l_1', l_2, l_3 . {l_1'}^R x l_2 = l_3 \Rightarrow (x l_1')^R l_2 = l_3 ?$$

✓

$$Reva(\epsilon, l_2, l_2).$$

$$Reva(l_1', x \cdot l_2, l_3) \Rightarrow Reva(x \cdot l_1', l_2, l_3).$$

$$Reva(l_1, l_2, l_3) \land Reva(l_3, \epsilon, l_4)$$

$$\land\ Reva(l_2, l_1, l_5) \Rightarrow l_4 = l_5.$$

CHC Solving Procedure via STP Inference

function SOLVE(D, G)
$T := \emptyset$;
$M :=$ `collect_samples`(D);
while true **do**
if `true_samples`$(M) \models \neg G$ **then** return UNSAT;
end if
$t_{new} :=$ `STPinf`(M);
if $t_{new} \notin T$ and $T \cup \{t_{new}\} \models D$ **then**
$T := T \cup \{t_{new}\}$;
if $T \models G$ **then** return SAT(T)
end if;
else
$M := M \cup$ `collect_more_samples`(D, t_{new});
end if

Outline

- ♦ Motivations
- ♦ Solvable Tuple Patterns
- ♦ Applications to Automated Program Verification
  - Program Verification via CHCs
  - CHC solving via SPT inference
  - Combination with CHC solving over integer arithmetic
  - Applications of Set/Multiset Patterns
- ♦ Implementation and Experiments
- ♦ Related Work
- ♦ Conclusion

Motivating Example

```
let rec take n l =
   if n=0 then []
   else
    match l with
       [] -> []
     | x::l' -> x::(take (n-1) l')
let rec drop n l =
   if n=0 then l
   else
      match l with
         [] -> []
      | _::l' -> drop (n-1) l'
let main n l =
   assert((take n l)@(drop n l)=l)
```

With STPs, we can only express:
  take(n,l) returns a prefix of l
  drop(n,l) returns a postfix of l

Length Abstraction

```
let rec take n l =
   if n=0 then []
   else
    match l with
      [] -> []
     | x::l' -> x::(take (n-1) l')
let rec drop n l =
   if n=0 then l
   else
     match l with
       [] -> []
     | _::l' -> drop (n-1) l'
let main n l =
   assert((take n l)@(drop n l)=l)
```

```
let rec take' n l =
  if n=0 then 0
  else
     if l=0 then 0
     else
       let l'=l-1 in 1+(take' (n-1) l')
let rec drop' n l =
  if n=0 then l
  else
    if l=0 then 0
    else
      let l'=l-1 in drop' (n-1) l'
let main n l =
  assert((take' n l)+(drop' n l)=l)
```

Length Abstraction

```
let rec take' n l =
  if n=0 then 0
  else
     if l=0 then 0
     else
       let l'=l-1 in 1+(take' (n-1) l')
let rec drop' n l =
  if n=0 then l
  else
    if l=0 then 0
    else
      let l'=l-1 in drop' (n-1) l'
let main n l =
   assert((take' n l)+(drop' n l)=l)
```

Output of CHC solvers for LIA

$$Take'(n, l, r) \equiv (l < n \land r = l) \lor (l \geq n \land r = n)$$

$$Drop'(n, l, r) \equiv (l < n \land r = 0) \lor (l \geq n \land r = l - n).$$

Length Abstraction

```
let rec take' n l =
  if n=0 then 0
  else
     if l=0 then 0
     else
       let l'=l-1 in 1+(take' (n-1) l')
let rec drop' n l =
  if n=0 then l
  else
    if l=0 then 0
    else
      let l'=l-1 in drop' (n-1) l'
let main n l =
   assert((take' n l)+(drop' n l)=l)
```

Output of CHC solvers for LIA

$$Take'(n,l,r) \equiv (l < n \land r = l) \lor (l \geq n \land r = n)$$

$$Drop'(n,l,r) \equiv (l < n \land r = 0) \lor (l \geq n \land r = l - n).$$

Combination with the result of STP inference

$$Take(n,l,r) \equiv \exists s.rs = l \land ((len(l) < n \land len(r) = len(l)) \lor (len(l) \geq n \land len(r) = n))$$

$$Drop(n,l,r) \equiv \exists s.sr = l \land ((len(l) < n \land len(r) = 0) \lor (len(l) \geq n \land len(r) = len(l) - n)).$$

Applications of Multiset Tuple Patterns

$$Insert(x, [], [x]). \qquad x \leq y \Rightarrow Insert(x, y :: l_2, x :: y :: l_2).$$

$$x > y \wedge Insert(x, l_2, l_3) \Rightarrow Insert(x, y :: l_2, y :: l_3).$$

$$Sort([], []). \qquad Sort(l_1, l_2) \wedge Insert(x, l_2, l_3) \Rightarrow Sort(x :: l_1, l_3).$$

$$Sort(l_1, l_2) \wedge Count(x, l_1, z_1) \wedge Count(x, l_2, z_2) \Rightarrow z_1 = z_2.$$

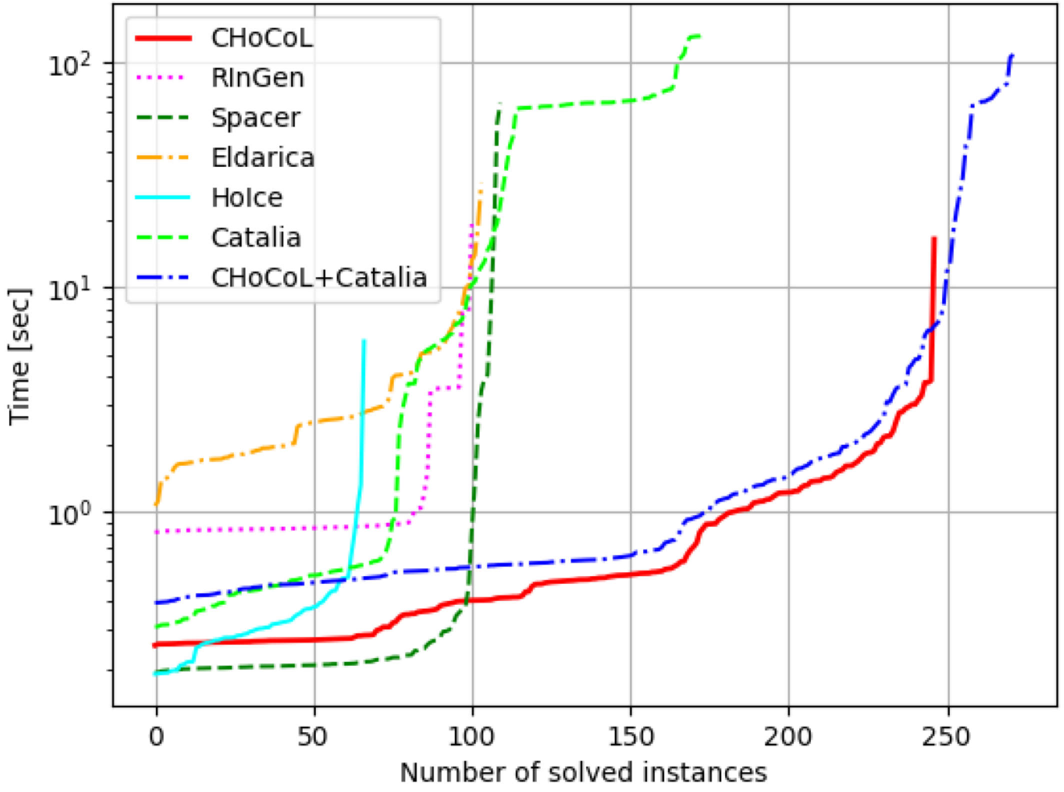
Multiset abstraction with STP inference for multisets yield:

$Insert(x, l_1, l_2) \equiv \{x\} \cup ms(l_1) = ms(l_2)$

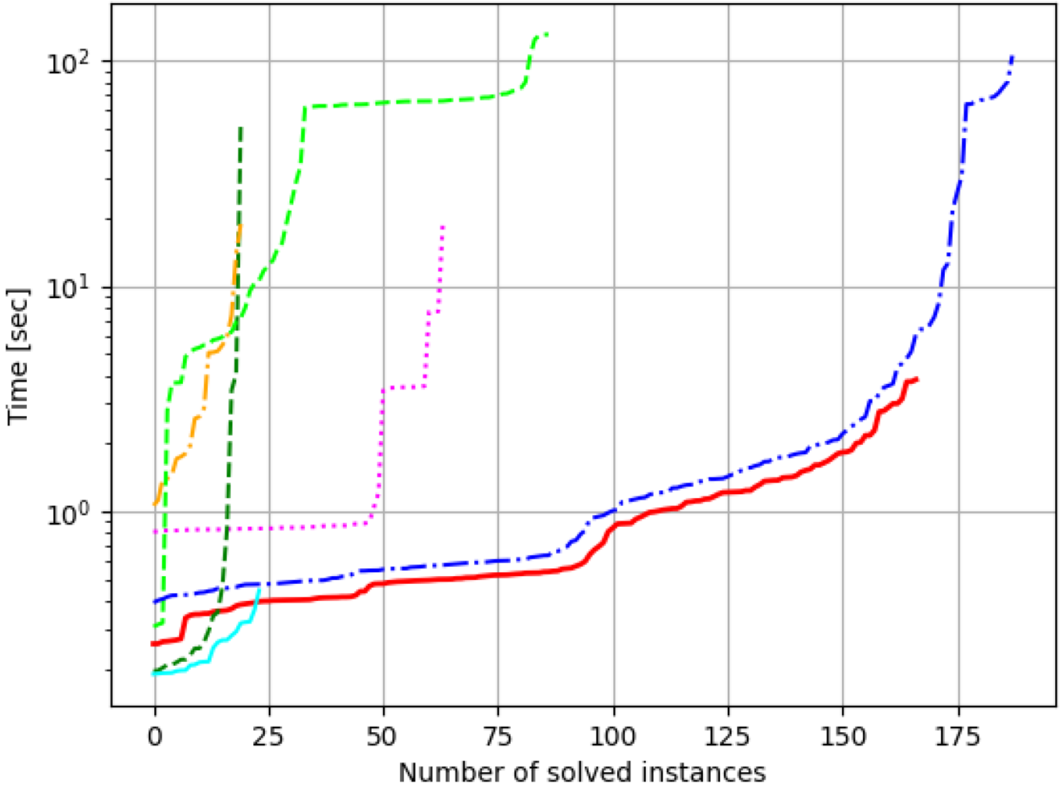
$Sort(l_1, l_2) \equiv ms(l_1) = ms(l_2)$

Experimental Results

for CHCs over Lists from CHC-COMP 2025 Benchmark



(a) All instances



(b) SAT instances

The Numbers of Solved Instances

Solver	Solved (SAT)	Solved (UNSAT)	Solved (all)
CHoCoL	167 (84)	80 (0)	247 (84)
RInGen	64 (23)	37 (4)	101 (27)
Spacer	20 (2)	90 (0)	110 (2)
ELdarica	20 (0)	84 (0)	104 (0)
HoIce	24 (6)	43 (0)	67 (6)
Catalia	87 (12)	87 (0)	174 (12)
CHoCoL+Catalia	188	84	272

Related Work

- **Pattern Languages [Angluin 80, Shinohara 83, ...]**
  - No efficient algorithm known for the full class of pattern languages
  - Previously known subclasses (e.g. regular patterns [Shinohara 94]) do not seem very useful for program verification
- **Data-driven approaches to invariant inference**
  - Algebraic methods: [Sharma+ ESOP13][Ikeda+, APLAS23]
  - Decision trees: [Garg+][Champion+ TACAS18]...
  - SVM: [Zhu+ ICFP15]...
  - Neural networks: CLN2INV [Ryan+ 20], NeuGuS [K+ SAS21] ...

  Mostly for inference of numerical relations

Related Work

- ♦ **Solving CHCs over ADTs**
  - **Induction [Unno CAV17], Unfold/fold transformation [Angelis+ 18], ...**
    - **require heuristics and/or hints**
  - **Abstractions: RinGen [Koskyukov PLDI21], Racer [Govind+ POPL22], Catalia [SAS25], ...**
    - **unable to prove the equality of lists**

Conclusion

♦ Proposed Solvable Tuple Patterns (STPs) for Lightweight Representation/Inference of Relations among Sequences

♦ Applied STPs to CHC Solving for List-like Data Structures

Future Work

- Supporting more patterns (such as sort/fold functions)
- Extension to tree patterns
- Other applications