

# Refinement Type Inference via Horn Constraint Optimization

Kodai Hashimoto and Hiroshi Unno  
(University of Tsukuba, Japan)

# Our Goal: Path-Sensitive Program Analysis of Higher-order Non-det. Functional Programs

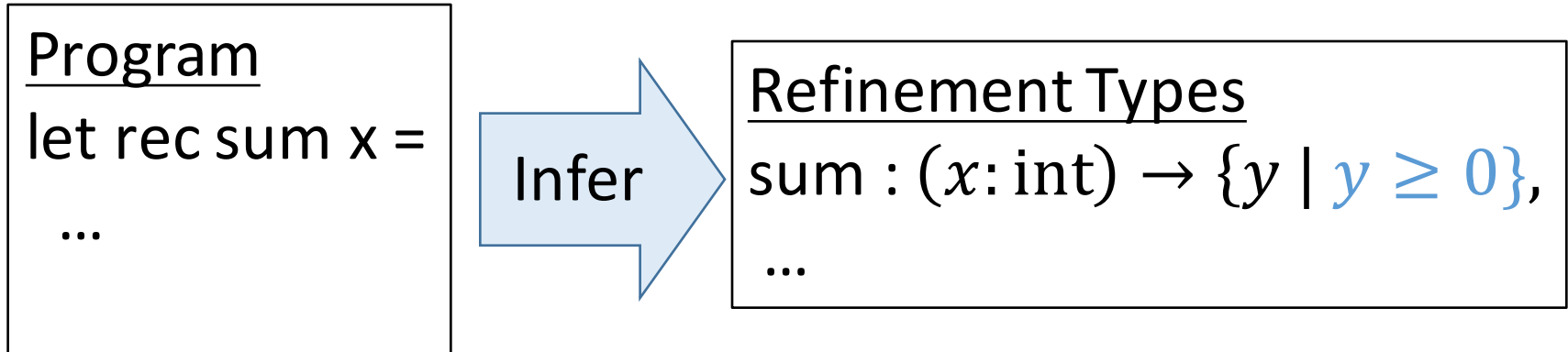
- Precondition inference
- Bug finding
- (Conditional) termination analysis
- Non-termination analysis
- Modular verification
- ...



## **Refinement type optimization**

a generalization of ordinary  
refinement type inference

# Refinement Type Inference



Refinement types can precisely express  
**program behaviors**

- $\{x : \text{int} \mid x \geq 0\}$  FOL predicates (e.g., QFLIA)  
Non-negative integers
- $(x : \text{int}) \rightarrow \{y : \text{int} \mid y \geq x\}$   
Functions that take an integer  $x$  and return an integer  $y$  not less than  $x$

# A Challenge in Refinement Type Inference

Which refinement type should be inferred?

```
let rec sum x = if x = 0 then 0 else x + sum (x-1)
```

contradiction

$\{x \mid x = -1\} \rightarrow \{y \mid \perp\} \text{ :> } \{x \mid x < 0\} \rightarrow \{y \mid \perp\}$

$\text{int} \rightarrow \{y \mid y \geq 0\}$  incomparable  $x < -5 \rightarrow \{y \mid \perp\}$

$\{x \mid x = 0\} \rightarrow \{y \mid y = 0\}$  ...

The most general types are often not expressible in the underlying logic (e.g., QFLIA)

# Existing Refinement Type Inference Tools

## **Infer refinement types precise enough to verify a given safety specification**

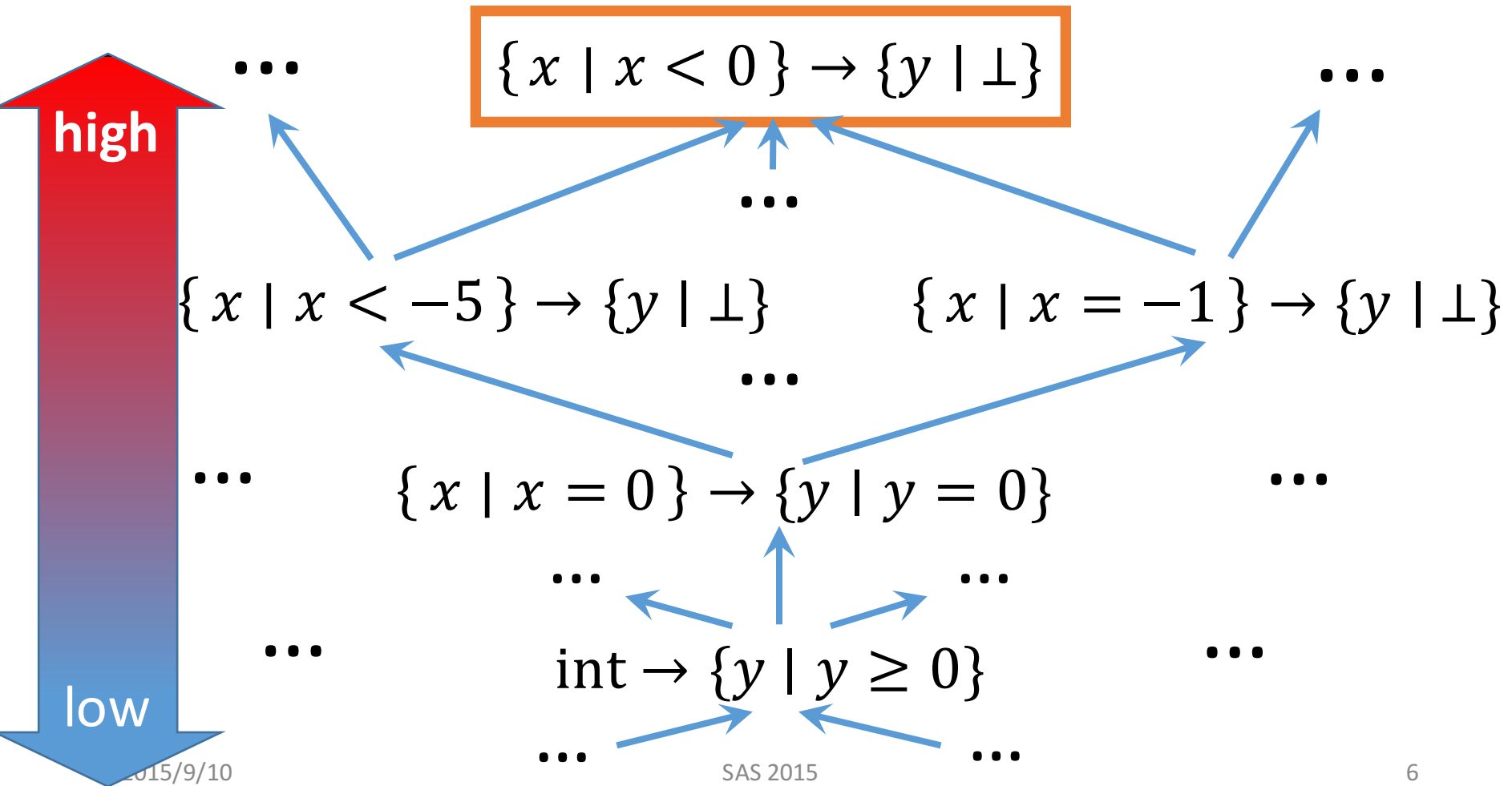
- Refinement Caml [Unno+ '08, '09, '13, '15]
- Liquid Types [Jhala+ '08, '09, ..., '15]
- MoCHi [Kobayashi+ '11, '13, '14, '15, '15]
- Depcegar [Terauchi '10]
- HMC [Jhala+ '11]
- Popeye [Zhu & Jagannathan '13]

Inferred types are often too specific to the spec.

→ Limited applications

# Our Approach: Refinement Type Optimization

Infer maximally preferred (i.e. **Pareto optimal**) refinement types with respect to **a user-specified preference order**



# How to Specify Preference Orders (1/3)

- Refinement type template

**Predicate variables**

$$(x : \{x \mid P(x)\}) \rightarrow \{y \mid Q(x, y)\}$$

$$P(x) \mapsto x < 0, \\ Q(x, y) \mapsto \perp$$

$$P(x) \mapsto x = 0, \\ Q(x, y) \mapsto y = 0$$

$$\{x \mid x < 0\} \rightarrow \{y \mid \perp\} \quad \{x \mid x = 0\} \rightarrow \{y \mid y = 0\}$$

# How to Specify Preference Orders (2/3)

## *max/min* optimization constraints

***max*( $P$ )**: infer a maximally-**weak** predicate for  $P$   
***min*( $Q$ )**: infer a maximally-**strong** predicate for  $Q$

**Precondition**

**Postcondition**

sum : ( $x : \{x \mid P(x)\}$ )  $\rightarrow$   $\{y \mid Q(x, y)\}$

**let rec** sum x = **if** x = 0 **then** 0 **else** x + sum (x-1)

***max*( $P$ )**  
***min*( $Q$ )**

$\{x \mid x < 0\} \rightarrow \{y \mid \perp\}$        $\text{int} \rightarrow \{y \mid y \geq 0\}$

$\{x \mid x = 0\} \rightarrow \{y \mid y = 0\}$



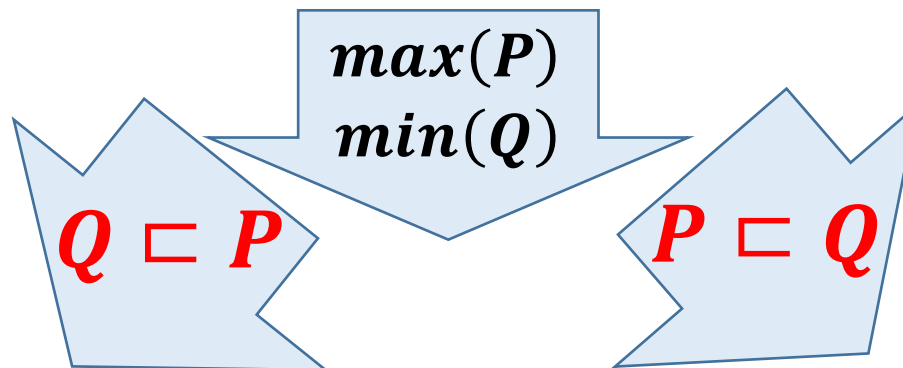
# How to Specify Preference Orders (3/3)

**a priority order  $\sqsubset$  on predicate variables**

$Q \sqsubset P$ :  $Q$  is given higher priority over  $P$

$\text{sum} : (x : \{x \mid P(x)\}) \rightarrow \{y \mid Q(x, y)\}$

**let rec** sum x = **if** x = 0 **then** 0 **else** x + sum (x-1)



$(x : \{x \mid x < 0\}) \rightarrow \{y \mid \perp\}$

$(x : \text{int}) \rightarrow \{y \mid y \geq 0\}$

# Outline

- Refinement Type Optimization
  - Applications
  - Our Type Optimization Method
- Implementation & Experiments
- Summary

# Applications of Refinement Type Optimization

- Non-termination analysis
- Conditional termination analysis
- Precondition inference
- Bug finding
- Modular verification
- ...

# Non-Termination Analysis

Find a program input that violates the termination property

No return value = **Non-terminating**

$\text{sum} : (x : \{x \mid P(x)\}) \rightarrow \{y \mid Q(x, y)\}$   $\perp \Leftarrow Q(x, y)$   
`let rec sum x = if x = 0 then 0 else x + sum (x-1)`

infer a  
maximally-weak  
precondition  $P$

$\max(P)$

Existing non-termination  
analysis tool may infer:  
 $\{x \mid x = -1\} \rightarrow \{y \mid \perp\}$

$(x : \{x \mid x < 0\}) \rightarrow \{y \mid \perp\}$

**sum never terminates if  $x < 0$**

# Non-Termination Analysis of Non-Deterministic Programs

```
f : (x:int) → {r | Q(r)}  
let rec f x =  
  let n = read_int() in  
  if n = x then f (x+1) else x
```

$\perp \Leftarrow Q(r)$

non-determinism

infer a  
maximally-weak  
condition ***P***

$\max(P)$

$n : \{n \mid \mathbf{n} = x\}$

$f : (x:int) \rightarrow \{r \mid \perp\}, \dots$

**f never terminates  
if the user always  
inputs same value  
as an argument  $x$**

# Non-Termination Analysis of Higher-Order Programs

```
main : (x : {x | P(x)}) → {y | Q(x, y)}, ...  $\perp \Leftarrow Q(x, y)$   
let rec fix (f:int -> int) x =  
  let x' = f x in  
  if x' = x then x else fix f x'  
let to_zero x = if x = 0 then 0 else x - 1  
let main x = fix to_zero x
```

infer a  
maximally-weak  
precondition **P**

$\max(P)$

**main never terminates**  
**if  $x < 0$**

```
main : (x: {x |  $x < 0$ }) → {r |  $\perp$ },  
fix : (f: (a : {a | a < 0}) → {b | b < a})  
      → (x : {x | x < 0}) → {y |  $\perp$ },  
to_zero : (x : {x | x < 0}) → {y | y < x}
```

# Applications of Refinement Type Optimization

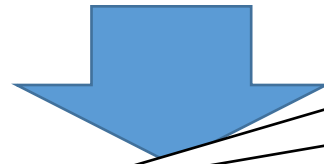
- Non-termination analysis
- Conditional termination analysis
- Precondition inference
- Bug finding
- Modular verification
- ...

# Conditional Termination Analysis (1/2)

- Infer a sufficient condition for termination
- Our approach is inspired by a program transformation approach to termination analysis of imperative programs [Gulwani+ '08, '09]

```
let rec sum x = if x = 0 then 0 else x + sum (x-1)
```

the initial  
value of x



the number of  
recursive calls

```
let rec sum_t x i c =  
  if x = 0 then 0 else x + sum_t (x-1) i (c+1)
```



# Conditional Termination Analysis (2/2)

Infer a sufficient condition for termination

$$\exists f. c \leq f(i) \Leftarrow \mathit{Bnd}(i, c). \quad \mathit{Bnd}(i, c) \Leftarrow P(x) \wedge \mathit{Inv}(x, i, c)$$

```
sum_t: (x : { x | P(x) }) → (i : int) → (c : { c | Inv(x, i, c) }) → int
let rec sum_t x i c =
  if x = 0 then 0 else x + sum_t (x-1) i c
```

$$\begin{aligned} \mathit{Inv}(x, i, c) \\ \Leftarrow c = 0 \wedge i = x \end{aligned}$$

$$\begin{aligned} \mathit{max}(P), \mathit{min}(\mathit{Bnd}) \\ P \sqsubset \mathit{Bnd} \end{aligned}$$

```
sum_t: (x : { x | x ≥ 0 }) → (i : int) →
(c : { c | x ≤ i ∧ i = x + c }) → int
```

$$f(i) = i \quad \mathit{Bnd}(i, c) \mapsto c \leq i$$

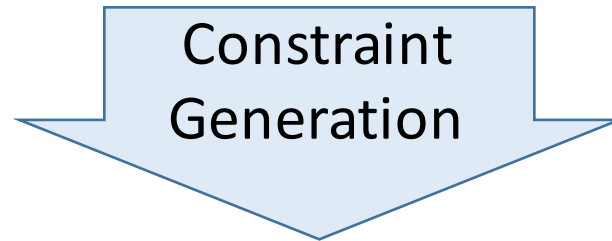
```
sum x
terminates
when x ≥ 0
because c ≤ i
```

# Outline

- Refinement Type Optimization
  - Applications
  - **Our Type Optimization Method**
- Implementation & Experiments
- Summary

# Overall Structure

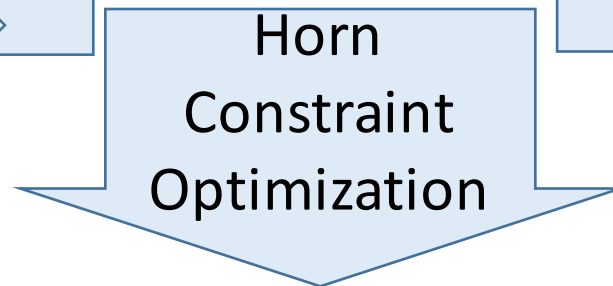
Functional Program



user-specified  
preference order  
(max/min opt.  
constraints +  
a priority order)

Horn Clause  
Constraints

additional  
Horn Clause  
constraints



Refinement Types

# Example: Type Optimization by Our Method

```
sum : (x : {x | P(x)}) → {y | ⊥}  
let rec sum x = if x = 0 then 0 else x + sum (x-1)
```

Constraint  
Generation

$$H_{sum} = \forall x. \left\{ \begin{array}{l} \perp \Leftarrow P(x) \wedge x = 0, \\ P(x-1) \Leftarrow P(x) \wedge x \neq 0 \end{array} \right\}$$

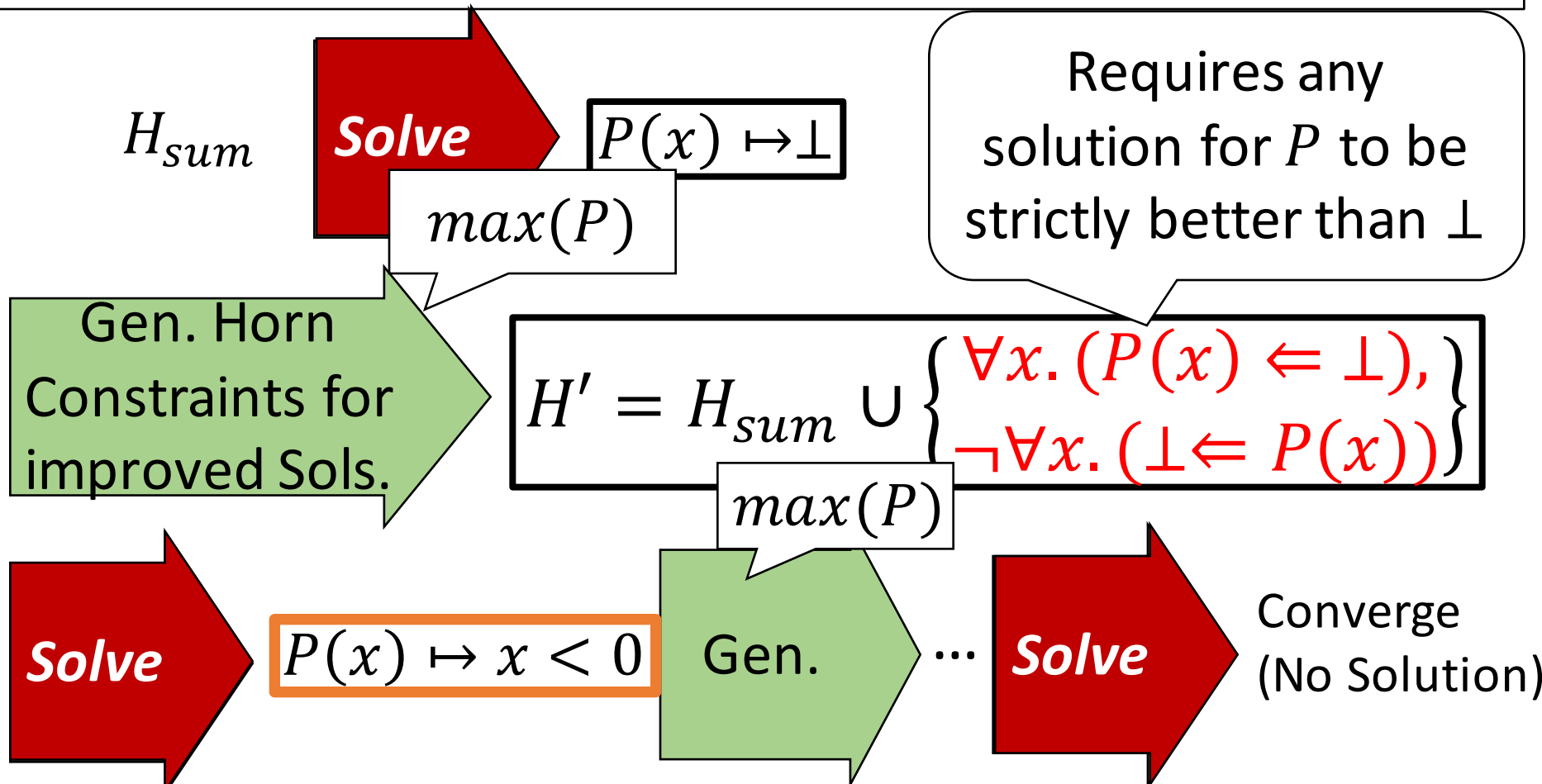
Horn  
Constraint  
Optimization

$max(P)$

```
(x : {x | x < 0}) → {y | ⊥}
```

# Example: Horn Constraint Optimization

repeatedly improves a current solution  
until convergence



# Horn Constraint Solver *Solve*

- Extended template-based invariant generation techniques [Colon+ '03, Gulwani+ '08] to solve ***existentially-quantified Horn clause constraints***
  - Extend the reach from imperative programs w/o recursion to higher-order non-det. programs
- Any other solver for the class of constraints can be used instead [Unno+ '13, Beyene+ '14, Kuwahara+ '15]

# Outline

- Refinement Type Optimization
  - Applications
  - Our Type Optimization Method
- Implementation & Experiments
- Summary

# Implementation & Experiments

A refinement type checking  
and inference tool for OCaml

- Implemented in ***Refinement Caml*** [Unno+ '08, '09, ...]
  - Z3 [Moura+ '08] as a backend SMT solver
- Two preliminary experiments:
  - Various program analysis problems for higher-order non-deterministic programs (partly obtained from [Kuwahara+ '14, Kuwahara+ '15])
  - Non-termination verification problems for first-order non-deterministic programs (obtained from [Chen+ '14, Larraz+ '14, Kuwahara+ '15, ...])



# Results of the Various Program Analysis Problems (excerpt)

| Program                             | Application                      | #Iter. | Time (sec) | Opt. |
|-------------------------------------|----------------------------------|--------|------------|------|
| foldr_nonterm<br>[Kuwahara+ '15]    | Non-termination                  | 4      | 8.04       | ✓    |
| fixpoint_nonterm<br>[Kuwahara+ '15] | Non-termination                  | 2      | 0.30       | ✓    |
| indirectHO_e<br>[Kuwahara+ '15]     | Non-termination                  | 2      | 0.31       | ✓    |
| zip<br>[Kuwahara+ '14]              | Conditional Termination Analysis | 4      | 12.24      |      |
| sum                                 | Conditional Termination Analysis | 6      | 12.02      | ✓    |
| append<br>[Kuwahara+ '14]           | Conditional Termination Analysis | 11     | 10.66      | ✓    |

Environment: Intel Core i7-3770 (3.40GHz), 16 GB of RAM

# Results of the First-Order Non-Termination Verification Problems

|                          | Verified  | Time Out | Other |
|--------------------------|-----------|----------|-------|
| Our tool                 | <b>41</b> | 27       | 13    |
| CppInv<br>[Larraz+ '14]  | <b>70</b> | 6        | 5     |
| T2-TACAS<br>[Chen+ '14]  | <b>51</b> | 0        | 30    |
| MoCHi<br>[Kuwahara+ '15] | <b>48</b> | 26       | 7     |
| TNT<br>[Emmes+ '12]      | <b>19</b> | 3        | 59    |

# Summary

## Refinement type optimization problems

- Infer Pareto-optimal refinement types with respect to a **user-specified preference order**
- Has applications to various program analysis problems of higher-order and non-deterministic functional programs

## Refinement type optimization method

- Reduction to a Horn constraint optimization problem
- Horn constraint optimization method
  - Repeatedly improve the current solution until convergence

## Prototype implementation and preliminary experiments