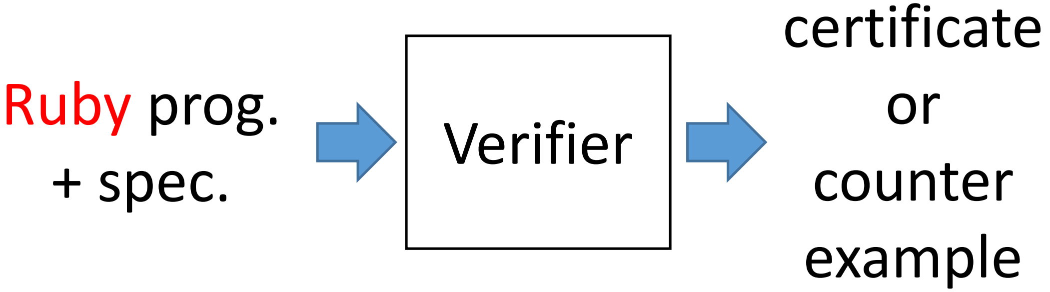
Ongoing Work: Verification of FJ Programs via Transformation to ML Programs

Hiroshi Unno (University of Tsukuba)

Our Ultimate Goal: Path-Sensitive Verification of Ruby Programs

- Ruby is a dynamic OO language designed by ***Yukihiro Matsumoto***, a graduate of U. of Tsukuba



Ruby prog.
+ spec.

Verifier

certificate
or
counter
example

Our Tentative Goal: Path-Sensitive Verification of Java Programs

- Approach: reduction to verification of ML programs with higher-order functions and recursive data types
  - Enable applications of verification techniques recently developed for ML (e.g., refinement types, Horn clause solving, and higher-order model checking)

Java prog.
+ spec.

Translator
(Java to ML)

ML prog.
+ spec.

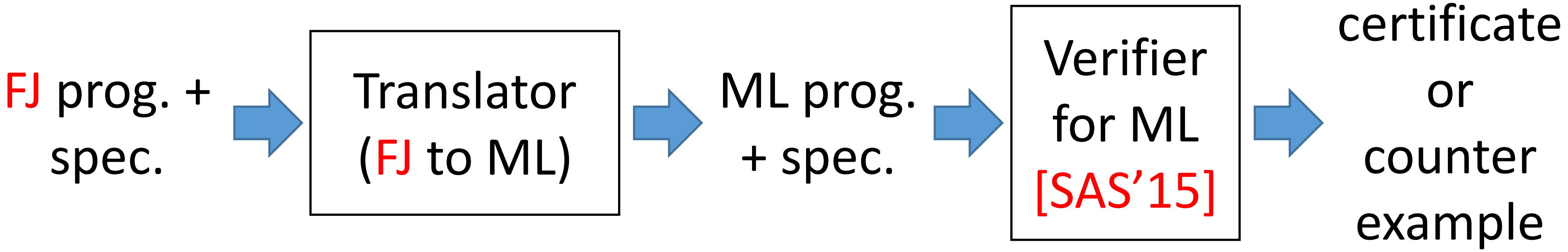
Verifier
for ML

certificate
or
counter
example

This Talk: Path-Sensitive Verification of Featherweight Java (FJ) Programs

- Approach: reduction to ve
ML programs with higher-o
and recursive data types
  - Enable applications of verific
developed for ML (e.g., refine
solving, and higher-order model checking)

A minimal core calculus
for Java [Igarashi, Pierce, Wadler '99]
w/ classes, methods,
fields, and inheritance
w/o assignments


FJ prog. + spec.

Translator (FJ to ML)

ML prog. + spec.

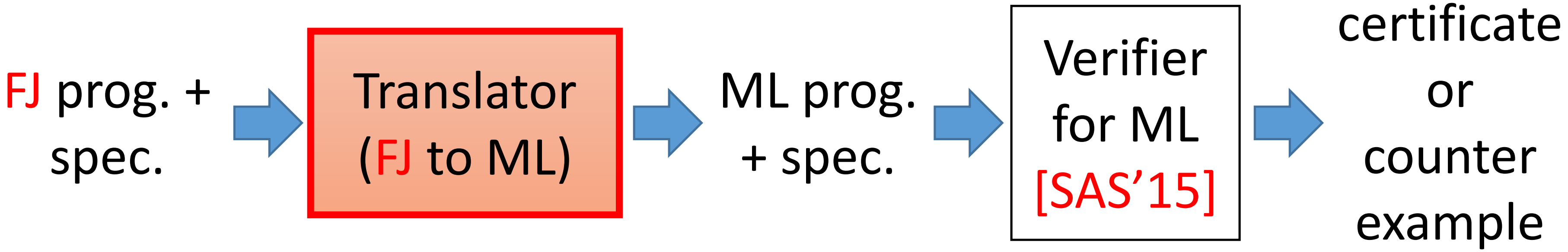
Verifier for ML [SAS'15]

certificate or counter example

This Talk: Path-Sensitive Verification of Featherweight Java (FJ) Programs

- Approach: reduction to ve
ML programs with higher-o
and recursive data types
  - Enable applications of verific
developed for ML (e.g., refine
solving, and higher-order model checking)

A minimal core calculus for Java [Igarashi, Pierce, Wadler '99] w/ classes, methods, fields, and inheritance w/o assignments



FJ to ML Translation
(cf. FJ to μHORS [Kobayashi and Igarashi '13])

- Use higher-order functions and recursive data types in ML to simulate dynamic method dispatch in Java

```
class List { boolean iseven() { assert false; return true; } }
class Nil extends List { boolean iseven() { return true; } }
class Cons extends List { int hd;  List tl;
   Cons(int hd, List tl) { this.hd = hd; this.tl = tl; }
   boolean iseven() { return !this.tl.iseven(); } }
class M {
   void main() { assert (this.mk_elist().iseven()); }
   List mk_elist() { return new Nil() □
                                     new Cons(1, new Cons(0, this.mk_elist())); }}
```

Verify that: new M().main() $\not\longrightarrow^*$ assert false

Example: Method Translation

```
class List { boolean iseven() { assert false; return true; } }
class Nil extends List { boolean iseven() { return true; } }
class Cons extends List { int hd;  List tl;
   Cons(int hd, List tl) { this.hd = hd; this.tl = tl; }
   boolean iseven() { return !this.tl.iseven(); } }
class M { void main() { assert (this.mk_elist().iseven()); }
   List mk_elist() { return new Nil() □
                                       new Cons(1, new Cons(0, this.mk_elist())); }}
```

```
let iseven_List () (this:obj) = assert false; true
let iseven_Nil () (this:obj) = true
let iseven_Cons (hd:int) (tl:obj) () (this:obj) = not (send_iseven () tl)
let main_M () (this:obj) = assert (send_iseven () (send_mk_elist () this))
let mk_elist_M () (this:obj) =
   new_Nil () □ new_Cons (1, new_Cons (0, send_mk_elist () this))
```

Example: Object Translation

```
class List { boolean iseven() { assert false; return true; } }
class Nil extends List { boolean iseven() { return true; } }
class Cons extends List { int hd;  List tl;
  Cons(int hd, List tl) { this.hd = hd; this.tl = tl; }
  boolean iseven() { return !this.tl.iseven(); } }
class M { void main() { assert (this.mk_elist().iseven()); }
  List mk_elist() { return new Nil() □
                                        new Cons(1, new Cons(0, this.mk_elist())); }}
```

```
type obj = Obj of (unit->obj->bool) * (unit->obj->unit) * (unit->obj->obj)
let send_iseven arg (Obj(m, _, _) as o) = m arg o
let send_main arg (Obj(_, m, _) as o) = m arg o
let send_mk_elist arg (Obj(_, _, m) as o) = m arg o
```

Example: Constructor Translation

```
class List { boolean iseven() { assert false; return true; } }
class Nil extends List { boolean iseven() { return true; } }
class Cons extends List { int hd;  List tl;
   Cons(int hd, List tl) { this.hd = hd; this.tl = tl; }
   boolean iseven() { return !this.tl.iseven(); } }
class M { void main() { assert (this.mk_elist().iseven()); }
   List mk_elist() { return new Nil() □
                                        new Cons(1, new Cons(0, this.mk_elist())); }}
```

```
let fail _ _ = assert false
let new_List () = Obj(iseven_List, fail, fail)
let new_Nil () = Obj(iseven_Nil, fail, fail)
let new_Cons (hd, tl) = Obj(iseven_Cons hd tl, fail, fail)
let new_M () = Obj(fail, mk_elist_M, main_M)
```

Obtained ML Verification Problem

Manually verified by a refinement type checker if the following types are provided:

type $objM$= Obj of $(\bot\to\bot\to \text{bool}) \times (\text{unit} \to objM \to \text{unit}) \times (\text{unit} \to objM \to objT)$
and $objT$= Obj of $(\text{unit} \to objT \to \{b : \text{bool} \mid b = \text{true}\}) \times (\bot\to\bot\to \text{unit}) \times (\bot\to\bot\to objT)$
and $objF$= Obj of $(\text{unit} \to objF \to \{b : \text{bool} \mid b = \text{false}\}) \times (\bot\to\bot\to \text{unit}) \times (\bot\to\bot\to objF)$
iseven_Nil : $\text{unit} \to objM \to \{b : \text{bool} \mid b = \text{true}\}$
iseven_Cons: $(\text{int} \to objF \to \text{unit} \to obj \to \{b : \text{bool} \mid b = \text{true}\})$
$\quad\wedge (\text{int} \to objT \to \text{unit} \to obj \to \{b : \text{bool} \mid b = \text{false}\})$
main_M : $(\text{unit} \to objM \to \text{unit})$
mk_elist_M : $(\text{unit} \to objM \to objT)$
send_iseven : $(\text{unit} \to objT \to \{b : \text{bool} \mid b = \text{true}\})$
$\quad\wedge (\text{unit} \to objF \to \{b : \text{bool} \mid b = \text{false}\})$
send_main : $(\text{unit} \to objM \to \text{unit})$
send_mk_elist : $(\text{unit} \to objM \to objT)$
new_Nil : $\text{unit} \to objT$
new_Cons: $(\text{int} \times objT \to objF) \wedge (\text{int} \times objF \to objT)$
new_M : $\text{unit} \to objM$

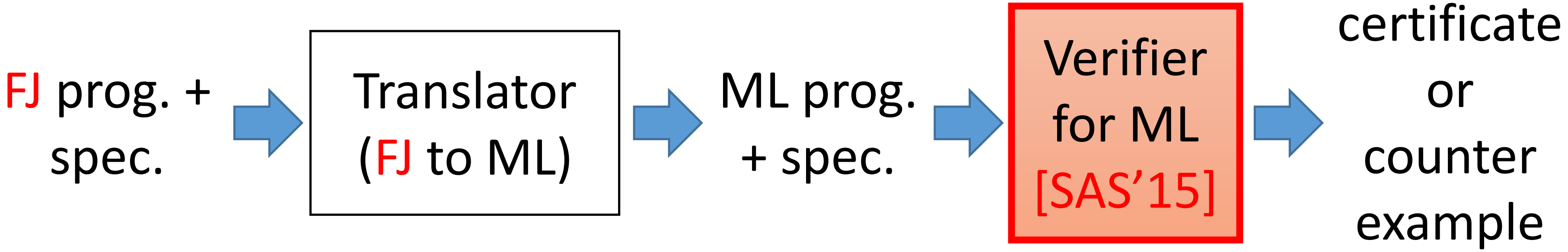
new_Nil () □ new_Cons (1, new_Cons (0, send_mk_elist () this))

Verify that: send_main () (new_M ()) $\not\longrightarrow^{*}$ assert false

This Talk: Path-Sensitive Verification of Featherweight Java (FJ) Programs

- Approach: reduction to ve ML programs with higher-o and recursive data types
  - Enable applications of verifica developed for ML (e.g., refine solving, and higher-order model checking)

A minimal core calculus for Java [Igarashi, Pierce, Wadler ’99] w/ classes, methods, fields, and inheritance w/o assignments



FJ prog. + spec.

Translator (FJ to ML)

ML prog. + spec.

Verifier for ML [SAS’15]

certificate or counter example

Refinement Type Inference via Horn Constraint Optimization

Hiroshi Unno (University of Tsukuba)
Joint work with Kodai Hashimoto

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

Program

let rec sum x =

…

Infer

Refinement Types

sum : $(x\!:\mathrm{int}) \rightarrow \{\, y \mid y \geq 0 \,\}$

…

Refinement types can precisely express program behaviors

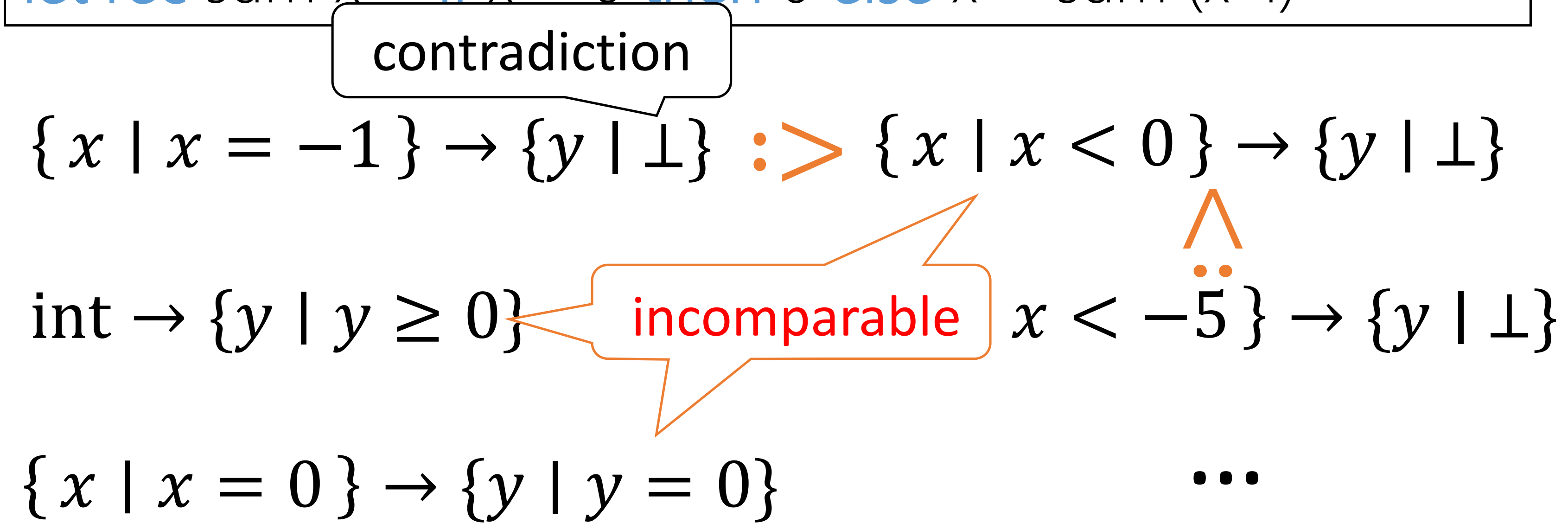
FOL predicates (e.g., QFLIA)

- $\{x : \mathrm{int} \mid x \geq 0\}$
  Non-negative integers
- $(x : \mathrm{int}) \rightarrow \{y : \mathrm{int} \mid y \geq x\}$
  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

$\{\, x \mid x = -1 \,\} \to \{y \mid \bot\} \;:>\; \{\, x \mid x < 0 \,\} \to \{y \mid \bot\}$

$\text{int} \to \{y \mid y \geq 0\}$

incomparable

$x < -5 \,\} \to \{y \mid \bot\}$

$\{\, x \mid x = 0 \,\} \to \{y \mid y = 0\}$

$\cdots$

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

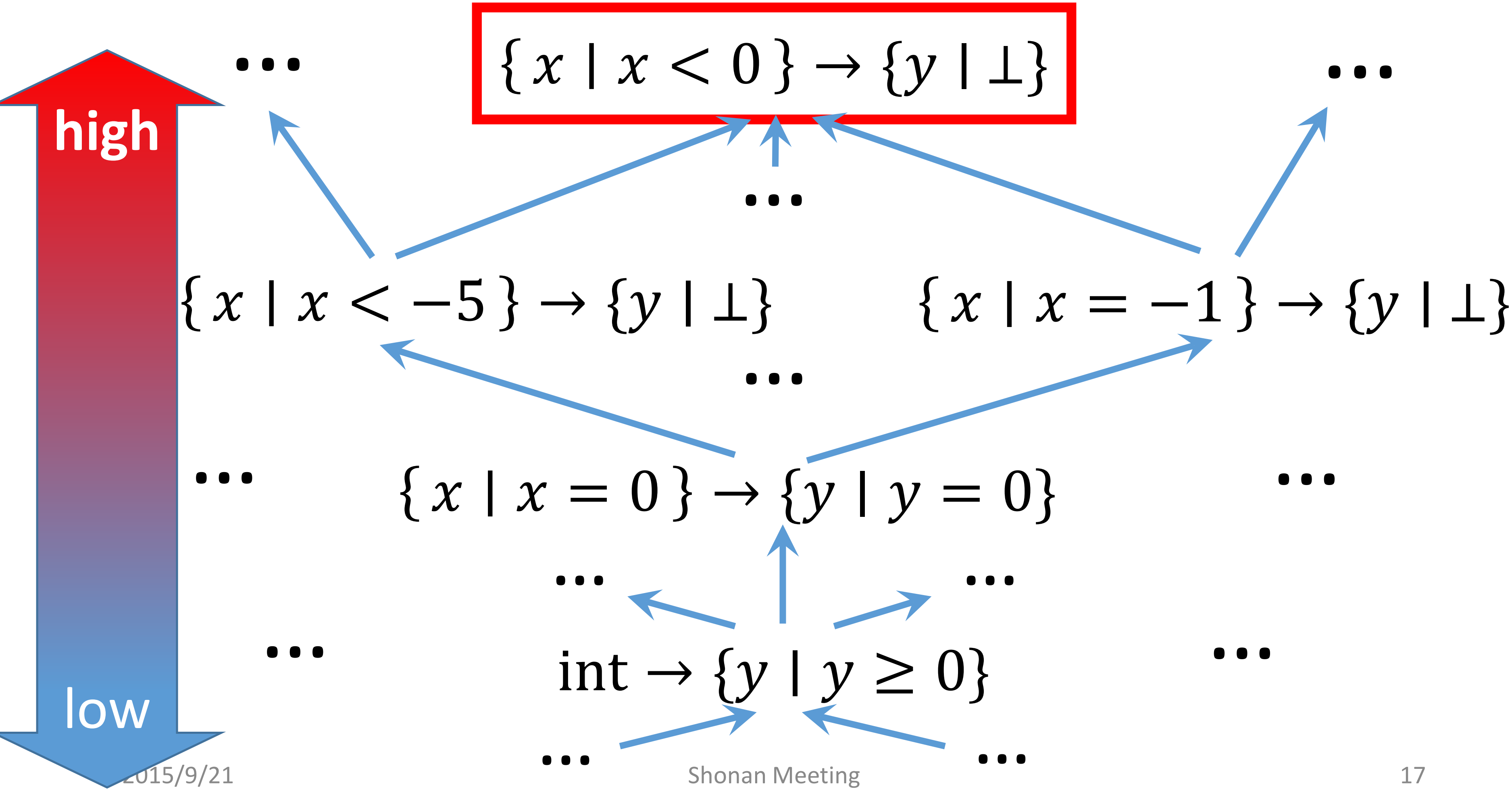
- Refinement Caml [Unno+ ’08, ’09, ’13, ’15]
- Liquid Types [Jhala+ ’08, ’09, …, ’15]
- MoCHi [Kobayashi+ ’11, ’13, ’14, ’15, ’15]
- Depcegar [Terauchi ’10]
- HMC [Jhala+ ’11]
- Popeye [Zhu & Jagannathan ’13]

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**



$$\{\, x \mid x < 0 \,\} \to \{y \mid \bot\}$$

$$\{\, x \mid x < -5 \,\} \to \{y \mid \bot\}$$

$$\{\, x \mid x = -1 \,\} \to \{y \mid \bot\}$$

$$\{\, x \mid x = 0 \,\} \to \{y \mid y = 0\}$$

$$\mathrm{int} \to \{y \mid y \geq 0\}$$

How to Specify Preference Orders (1/3)

Refinement type template

Predicate variables

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

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

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

$$\{x \mid x < 0\} \rightarrow \{y \mid \bot\} \qquad \{x \mid x = 0\} \rightarrow \{y \mid y = 0\}$$

How to Specify Preference Orders (2/3)

max/min optimization constraints

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

Precondition

Postcondition

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

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

$\boldsymbol{max}(\boldsymbol{P})$
$\boldsymbol{min}(\boldsymbol{Q})$

$\{ x \mid x < 0 \} \rightarrow \{y \mid \bot\}$ $\quad$ $\mathrm{int} \rightarrow \{y \mid y \geq 0\}$

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

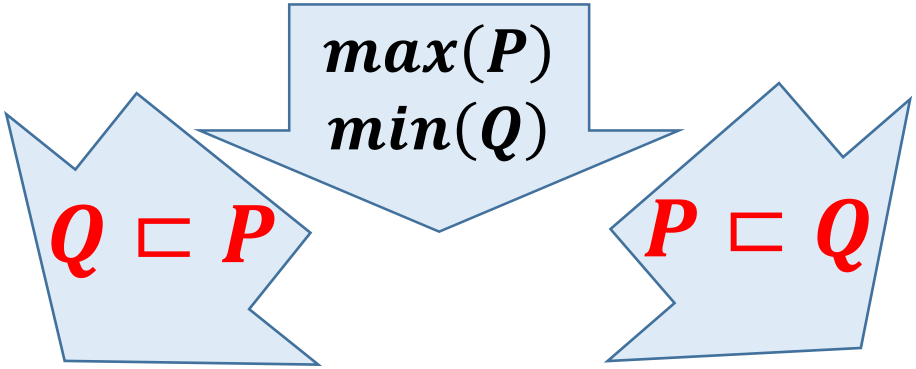
How to Specify Preference Orders (3/3)

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

$\boldsymbol{Q} \sqsubset \boldsymbol{P}$: Q is given higher priority over P

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

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



$\boldsymbol{max(P)}$
$\boldsymbol{min(Q)}$

$\boldsymbol{Q} \sqsubset \boldsymbol{P}$

$\boldsymbol{P} \sqsubset \boldsymbol{Q}$

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

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

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**

$\bot \Leftarrow \boldsymbol{Q(x,y)}$

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

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

infer a maximally-weak precondition $\boldsymbol{P}$

$\boldsymbol{max(P)}$

Existing non-termination analysis tool may infer:
$\{\, x \mid x = -1 \,\} \to \{y \mid \bot\}$

$(x : \{\, x \mid \boldsymbol{x < 0} \,\}) \to \{y \mid \bot\}$

sum never terminates if $\boldsymbol{x < 0}$

Non-Termination Analysis of Non-Deterministic Programs

f : (x: int) → {r | Q(r)}

$\bot \Leftarrow Q(r)$

```
let rec f x =
  let n      = read_int() in
  if n = x then f (x+1) else x
```

non-determinism

infer a maximally-weak condition $\boldsymbol{P}$

$\boldsymbol{max(P)}$

n : $\{\, n \mid \boldsymbol{n = x} \,\}$

f : (x: int) → {r | ⊥ }, …

f never terminates if the user always inputs same value as an argument $\boldsymbol{x}$

Non-Termination Analysis of Higher-Order Programs

main : $(x : \{x \mid \mathrm{P}(x)\}) \to \{y \mid \mathrm{Q}(x, y)\}$, … $\bot \Leftarrow Q(x, y)$

```
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 $\boldsymbol{P}$

$\boldsymbol{max(P)}$

main never terminates if $\boldsymbol{x < 0}$

main : $(x : \{\, \boldsymbol{x} \mid \boldsymbol{x < 0} \,\}) \to \{r \mid \bot\}$,

fix : $(\, f : (a : \{a \mid a < 0\}) \to \{b \mid b < a)\}\,)$
$\qquad \to (x : \{x \mid x < 0)\}) \to \{y \mid \bot\}$,

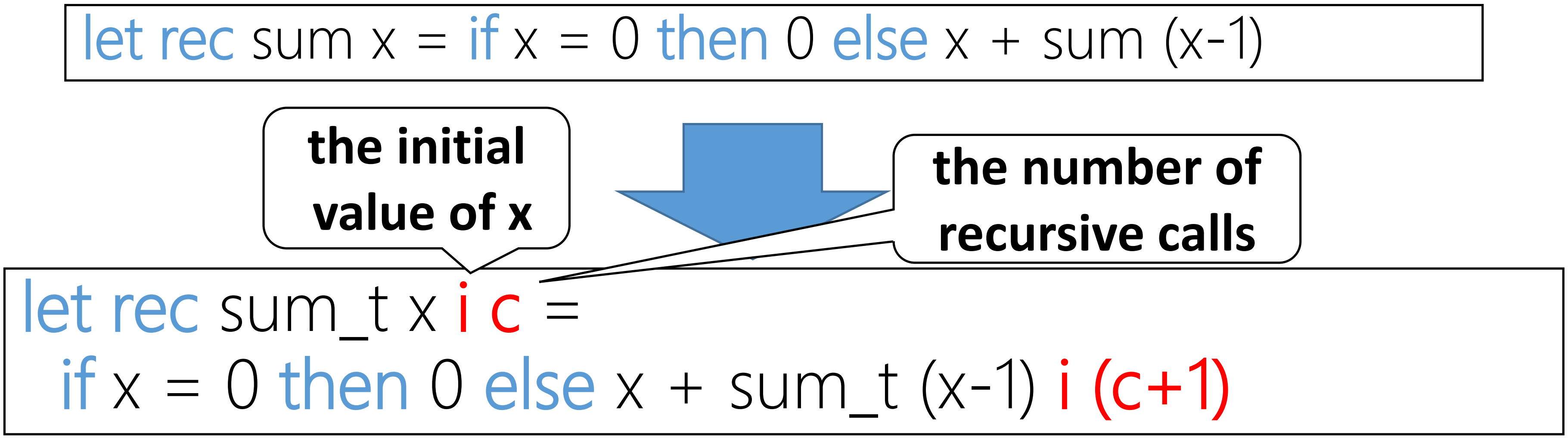
to_zero : $(x : \{x \mid x < 0\}) \to \{y \mid y < x\}$

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

$$\exists f. c \leq f(i) \Leftarrow Bnd(i,c). \quad Bnd(i,c) \Leftarrow P(x) \land Inv(x,i,c)$$

sum_t: $(x : \{ x \mid \boldsymbol{P(x)} \}) \rightarrow (i : \text{int}) \rightarrow (c : \{ c \mid \boldsymbol{Inv(x,i,c)} \}) \rightarrow \text{int}$

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

$Inv(x,i,c)$
$\Leftarrow c = 0 \land i = x$

$\boldsymbol{max(P), min(Bnd)}$

$\boldsymbol{P \sqsubset Bnd}$

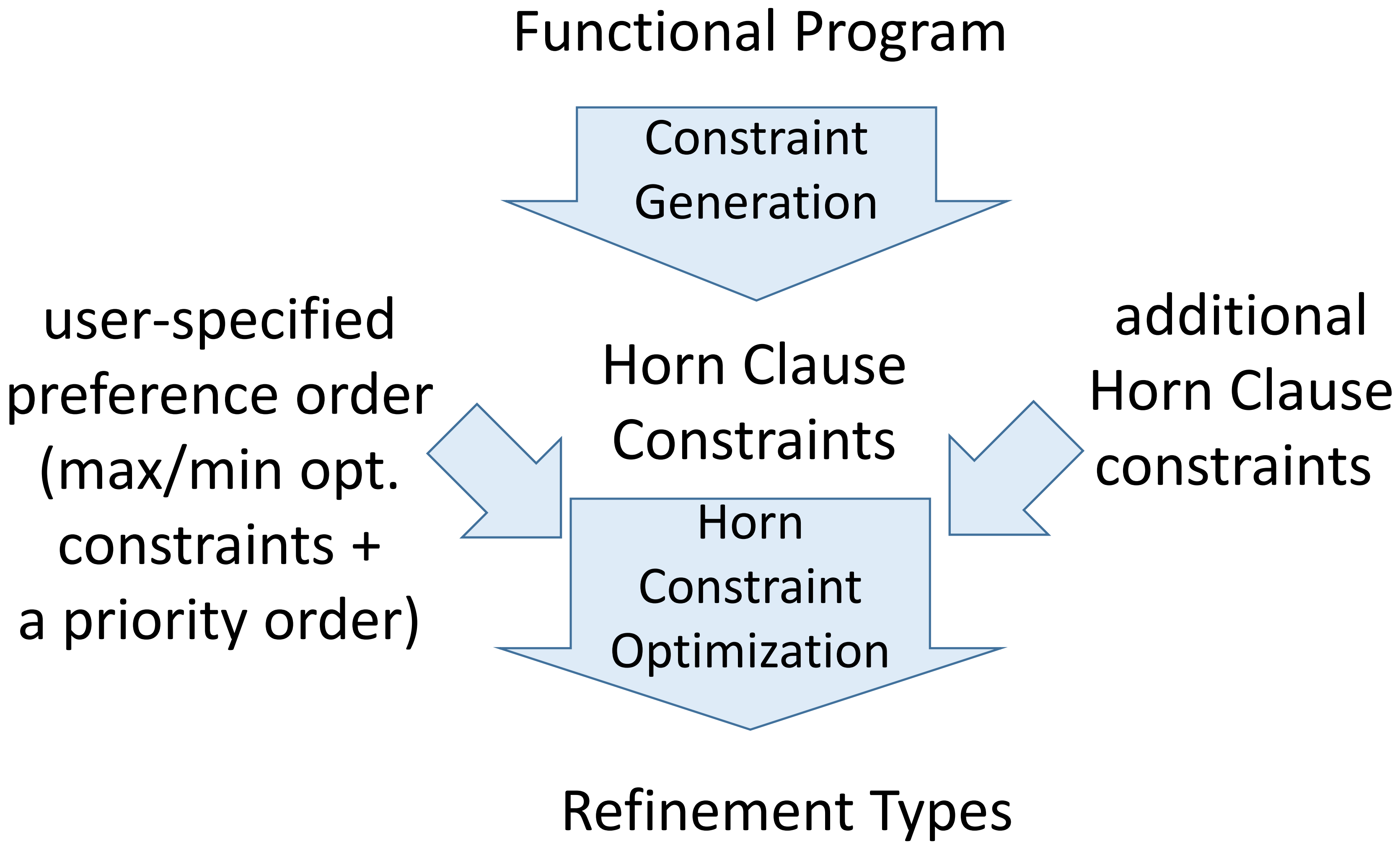
sum_t: $(x : \{ x \mid \boldsymbol{x \geq 0} \}) \rightarrow (i : \text{int}) \rightarrow$
$(c : \{ c \mid x \leq i \land i = x + c \}) \rightarrow \text{int}$
$f(i) = i \qquad Bnd(i,c) \mapsto \boldsymbol{c \leq i}$

sum x
terminates
when $x \geq 0$
because $c \leq i$

Outline

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

Overall Structure



Functional Program

Constraint Generation

Horn Clause Constraints

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

additional Horn Clause constraints

Horn Constraint Optimization

Refinement Types

Example: Type Optimization by Our Method

sum : $(x : \{x \mid P(x)\}) \to \{y \mid \bot\}$

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

Constraint Generation

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

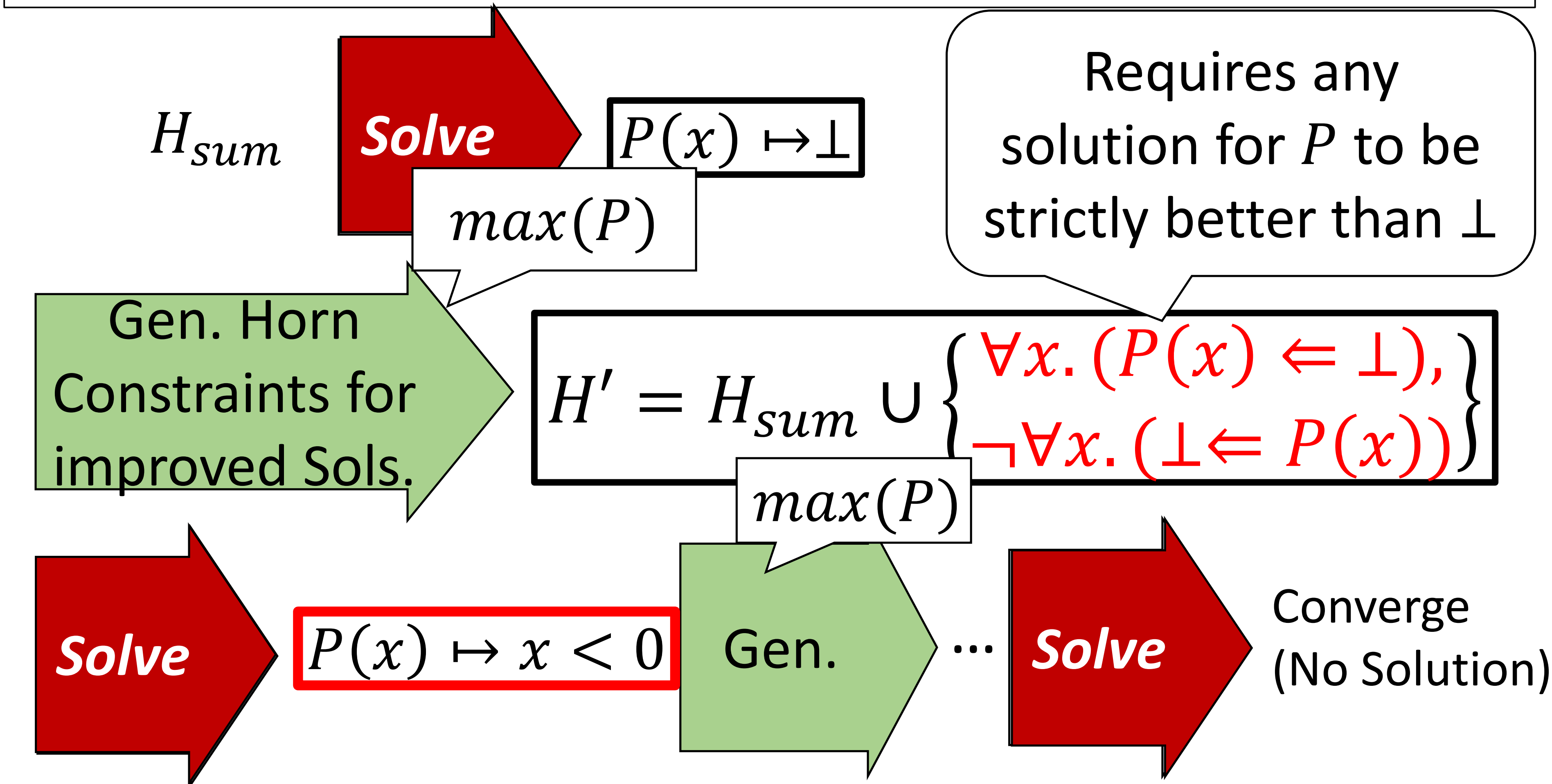
Horn Constraint Optimization

$max(P)$

$$(x : \{x \mid x < 0\}) \to \{y \mid \bot\}$$

Example: Horn Constraint Optimization

repeatedly improves a current solution
until convergence



H_{sum}

Solve

$P(x) \mapsto \bot$

$max(P)$

Requires any solution for P to be strictly better than ⊥

Gen. Horn Constraints for improved Sols.

$$H' = H_{sum} \cup \left\{ \begin{array}{l} \forall x.(P(x) \Leftarrow \bot), \\ \neg\forall x.(\bot \Leftarrow P(x)) \end{array} \right\}$$

$max(P)$

Solve

$P(x) \mapsto x < 0$

Gen.

…

Solve

Converge
(No Solution)

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 [Kuwahara+ ’15]	**Non-termination**	4	**8.04**	✔
fixpoint_nonterm [Kuwahara+ ’15]	**Non-termination**	2	**0.30**	✔
indirectHO_e [Kuwahara+ ’15]	**Non-termination**	2	**0.31**	✔
zip [Kuwahara+ ’14]	**Conditional Termination Analysis**	4	**12.24**	
sum	**Conditional Termination Analysis**	6	**12.02**	✔
append [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	**41**	27	13
CppInv [Larraz+ ’14]	**70**	6	5
T2-TACAS [Chen+ ’14]	**51**	0	30
MoCHi [Kuwahara+ ’15]	**48**	26	7
TNT [Emmes+ ’12]	**19**	3	59

Summary

Refinement type optimization problems

- Infer ***Pareto-optimal*** refinement types with respect to ***a user-specified preference order***
- 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

Thank you!