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

Program
let rec sum x = ...

Infer
Refinement Types
sum : (x: int) → \{y | y ≥ 0\}, ...

Refinement types can precisely express
program behaviors

• \{x : int | x ≥ 0\}
  FOL predicates (e.g., QFLIA)
  Non-negative integers

• (x : int) → \{y : int | y ≥ 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?

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

\[
\begin{align*}
\{ x \mid x = -1 \} & \rightarrow \{ y \mid \bot \} \implies \{ x \mid x < 0 \} \rightarrow \{ y \mid \bot \} \\
\text{int} & \rightarrow \{ y \mid y \geq 0 \} \quad x < -5 \rightarrow \{ y \mid \bot \} \\
\{ x \mid x = 0 \} & \rightarrow \{ y \mid y = 0 \} \\
\end{align*}
\]

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]
- 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

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

\[
\begin{align*}
P(x) & \leftrightarrow x < 0, \\
Q(x, y) & \leftrightarrow \bot
\end{align*}
\]

\[
\begin{align*}
\{ x \mid x < 0 \} & \rightarrow \{ y \mid \bot \} \\
\{ x \mid x = 0 \} & \rightarrow \{ y \mid y = 0 \}
\end{align*}
\]
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**

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

$\text{let rec sum } x = \text{if } x = 0 \text{ then } 0 \text{ else } x + \text{sum } (x-1)$

**Postcondition**

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

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

$\{ x \mid x = 0 \} \rightarrow \{ y \mid y = 0 \}$
How to Specify Preference Orders (3/3)

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

\[ Q \sqsubseteq P : Q \text{ is given higher priority over } P \]

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

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

\[
\begin{align*}
\max(P) & \quad \min(Q) \\
Q \sqsubseteq P & \quad P \sqsubseteq Q
\end{align*}
\]

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

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

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

\[ \text{let rec sum } x = \text{if } x = 0 \text{ then } 0 \text{ else } x + \text{sum} \ (x - 1) \]

\[ \frac{}{\Downarrow \iff Q(x, y)} \]

\[ \frac{}{\max(P)} \]

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

\[ \text{sum never terminates if } x < 0 \]

No return value = **Non-terminating**
Non-Termination Analysis of Non-Deterministic Programs

\[ f : (x : \text{int}) \rightarrow \{ r \mid Q(r) \} \]

\[ \bot \Leftarrow Q(r) \]

let rec \( f \) \( x \) =
  let \( n \) = \text{read_int()} \in
  if \( n = x \) then \( f \) \((x+1)\) else \( x \)

infer a maximally-weak condition \( P \)

\[ \max(P) \]

\[ n : \{ n \mid n = x \} \]

\[ f : (x : \text{int}) \rightarrow \{ r \mid \bot \}, \ldots \]

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)}, .. ⊥ ⊆ 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 | ⊥},
fix : (f : (a : {a | a < 0}) → {b | b < a})
  → (x : {x | x < 0})) → {y | ⊥},
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
- ...

2015/9/10  SAS 2015
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]

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

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

the initial value of x

the number of recursive calls
Conditional Termination Analysis (2/2)

Infer a sufficient condition for termination

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

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

let rec $\sum_t \ x \ i \ c =$

if $x = 0$ then 0 else $x + \sum_t (x-1) i$

$max(P), min(Bnd)$

$P \sqsubseteq Bnd$

$\sum_t: (x : \{x \mid x \geq 0 \}) \to (i : \text{int}) \to (c : \{ c \mid x \leq i \land i = x + c \}) \to \text{int}$

$f(i) = i \quad Bnd(i, c) \iff c \leq i$

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

$\sum x$

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

Horn Constraint Optimization

Refinement Types

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

additional Horn Clause constraints
Example: Type Optimization by Our Method

\[
\sum : (x : \{x \mid P(x)\}) \rightarrow \{y \mid \bot\}
\]

\[
\text{let rec } \sum x = \text{if } x = 0 \text{ then } 0 \text{ else } x + \sum (x-1)
\]

\[
H_{\text{sum}} = \forall x. \begin{cases} 
\bot \iff P(x) \land x = 0, \\
P(x - 1) \iff P(x) \land x \neq 0
\end{cases}
\]

\[
\text{Horn Constraint Optimization}
\]

\[
(x : \{x \mid x < 0\}) \rightarrow \{y \mid \bot\}
\]
Example: Horn Constraint Optimization

repeatedly improves a current solution until convergence

\[ H_{\text{sum}} \]

**Solve**

\[ P(x) \mapsto \bot \]

**max(\( P \))**

- Requires any solution for \( P \) to be strictly better than \( \bot \)

\[ H' = H_{\text{sum}} \cup \{ \forall x. (P(x) \iff \bot), \neg \forall x. (\bot \iff P(x)) \} \]

**Gen.**

- Gen. Horn Constraints for improved Sols.

\[ P(x) \mapsto x < 0 \]

\[ \text{Converge (No Solution)} \]
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

• 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)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>foldr_nonterm [Kuwahara+ ’15]</td>
<td>Non-termination</td>
<td>4</td>
<td>8.04</td>
<td>✔</td>
</tr>
<tr>
<td>fixpoint_nonterm [Kuwahara+ ’15]</td>
<td>Non-termination</td>
<td>2</td>
<td>0.30</td>
<td>✔</td>
</tr>
<tr>
<td>indirectHO_e [Kuwahara+ ’15]</td>
<td>Non-termination</td>
<td>2</td>
<td>0.31</td>
<td>✔</td>
</tr>
<tr>
<td>zip [Kuwahara+ ’14]</td>
<td>Conditional Termination Analysis</td>
<td>4</td>
<td>12.24</td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>Conditional Termination Analysis</td>
<td>6</td>
<td>12.02</td>
<td>✔</td>
</tr>
<tr>
<td>append [Kuwahara+ ’14]</td>
<td>Conditional Termination Analysis</td>
<td>11</td>
<td>10.66</td>
<td>✔</td>
</tr>
</tbody>
</table>

Environment: Intel Core i7-3770 (3.40GHz), 16 GB of RAM
### Results of the First-Order Non-Termination Verification Problems

<table>
<thead>
<tr>
<th>Tool</th>
<th>Verified</th>
<th>Time Out</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our tool</td>
<td>41</td>
<td>27</td>
<td>13</td>
</tr>
<tr>
<td>CppInv [Larraz+ ’14]</td>
<td>70</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>T2-TACAS [Chen+ ’14]</td>
<td>51</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>MoCHi [Kuwahara+ ’15]</td>
<td>48</td>
<td>26</td>
<td>7</td>
</tr>
<tr>
<td>TNT [Emmes+ ’12]</td>
<td>19</td>
<td>3</td>
<td>59</td>
</tr>
</tbody>
</table>

2015/9/10 SAS 2015
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

Thank you!