Monotonicity Checking: Proofs and Algorithm

Kwangkeun Yi

Dept. of Computer Science

http://ropas.kaist.ac.kr

KAIST (Korea Advanced Institute of Science & Technology)
Monotonicity

Monotonic function preserves the order of its arguments.

- \( x \leq y \) implies \( f(x) \leq f(y) \)
- \( (\forall i : x_i \leq y_i) \) implies \( f(x_1, \cdots, x_n) \leq f(y_1, \cdots, y_n) \).

Our goal: a static procedure that can certify whether a function between two lattices is monotonic or not.

Push [here].
Motivation

System Zoo

static analysis specification in Rabbit

abstract interpretation
data-flow analysis
constraint-based analysis
+
model checking for CTL

executable analyzer in C and nML

analysis results: fixpoints

http://ropas.kaist.ac.kr/zoo
Abstract Interpreter Example in Rabbit

analysis ConstantProp =
  ana
  set FnExp = /Ast.expr/
  set Id = /Ast.id/
  set I = {0 ... /Ast.maxRecArgs/}
  lattice C = power FnExp
  lattice L = power Id
  lattice K = L
  lattice Z = power Int
  lattice V = C * L * K * Z
  lattice A = I -> V
  lattice S = Id -> A

eqn Eval(/Ast.Const(n)/, s) = ((__, __, __, numToZ n), s)
  | Eval(/Ast.Apply(e,e')/, s) =
    let
      val (v’, s’) = Eval(/e/, s)
      val (v’’, s’’) = Eval(/e’/, s’)
    in
The $F$ is derived from the $\text{Eval}$ definition and the program to analyze.

Solution: $\sqcap\{\bot, F\bot, F^2\bot, \cdots\}$
Wanted: Not a Blind Zoo

Let Zoo generate analyzers only when the input spec passes

- monotonicity check: to guarantee that the generated analyzers terminate
  - monotonicity over finite lattices implies
  - analysis procedure $\sqcup\{\bot, F\bot, F^2\bot, \cdots\}$
  - that terminates
Related Works

In learning theory: [Vor00, DGL+99, GGLR98, Sch96]

- restricted to boolean functions: \( \{0, 1\}^n \to \{0, 1\} \)
  Zoo supports non-distributive lattices

- probabilistic: err with small probability
  tight bound for general case: seems formidable

- function as black box: extensional tests
  function source is given

Why not try a static analysis?
Language

Core of Rabbit:

\[
\begin{align*}
  e &::= c & \text{constant (lattice point)} \\
  &\mid x & \text{variable} \\
  &\mid \lambda x. e & \text{function} \\
  &\mid \text{fix } f e & \text{recursive definition} \\
  &\mid e e & \text{application} \\
  &\mid e \sqcup e & \text{join operation} \\
  &\mid e \sqcap e & \text{meet operation} \\
  &\mid \text{if } e \sqsubseteq e \text{ then } e \text{ else } e & \text{branching}
\end{align*}
\]

Values are either lattice elements or monotonic functions over lattices.
Monotonicity Check

Estimates the monotonicity behavior of

\[(x_1, \cdots, x_n) \mapsto e(x_1, \cdots, x_n)\]

summarized in a table:
- e.g., \(x \sqsupset c\) has
  \(\{x \mapsto \text{monotonic}, \text{else} \mapsto \text{constant}\}\)
- e.g., if \(x \sqsubseteq c\) then \(\top\) else \(\perp\) has
  \(\{x \mapsto \text{anti-monotonic}, \text{else} \mapsto \text{constant}\}\)
Monotonicity Judgement

\[ \Gamma \vdash e : \tau, me \]

\[
\begin{align*}
me & \in ME = Var^{\text{fin}} \rightarrow M \\
\tau & \in M = \{0, +, -, \top\} \\
\Gamma & \in TE = Var^{\text{fin}} \rightarrow \text{EffectType}
\end{align*}
\]

\[
\begin{align*}
\text{EffectType} & \quad t ::= (\tau, me) \\
\text{Type} & \quad \tau ::= \iota \mid (\tau, me) \rightarrow (\tau, me) \\
& \quad me ::= \vec{0} \mid \vec{+} \mid \vec{-} \mid \{x \mapsto +\} \\
& \quad \mid me[m/x] \mid \@ me me me me \mid me \lor me \\
& \quad \mid \text{if} me me me me \phi \mid \text{ifc} me me me \phi
\end{align*}
\]

Meaning of the monotonicity token \( m \):
The Rules (I)

\[ \Gamma \vdash c : \tau, 0 \quad \text{(CON)} \]

\[ \Gamma \vdash x : \tau, me[+/x] \quad \text{(VAR)} \]

\[ \Gamma \vdash e_1 : \tau, me_1 \quad \Gamma \vdash e_2 : \tau, me_2 \]
\[ \Gamma \vdash e_1 \sqcup e_2 : \tau, me_1 \sqcup me_2 \quad \text{(LUB)} \]

\[ \Gamma \vdash x : (\tau_1, me_1) \vdash e : \tau_2, me'_2 \quad me'_2 \sqsubseteq me_2 \]
\[ \Gamma \vdash \lambda x.e : (\tau_1, me_1) \rightarrow (\tau_2, me_2), me_2[0/x] \quad \text{(LAM)} \]
Case 1: the function to be called is fixed \((me_3 = 0)\)
- if the body and the arg have the same monotonicity, then increasing.
- if different, then decreasing.
- if one remains constant, then constant.
Case 2: the function to be called is changing

- if the body and argument combined have a different monotonicity from the function, then unpredictable
- otherwise, preserve the monotonicity

All captured by:

\[ @ m_{\text{arg}} m_{\text{body}} m_{\text{ftn}} = (m_{\text{arg}} \otimes m_{\text{body}}) \sqcup m_{\text{ftn}} \]
The Rules (III)

\[ \Gamma \vdash e_1 : \tau', me_1 \quad \Gamma \vdash e_2 : \tau', me_2 \quad \Gamma \vdash e_3 : \tau, me_3 \quad \Gamma \vdash e_4 : \tau, me_4 \]
\[ \Gamma \vdash \text{if } e_1 \sqsubseteq e_2 \text{ then } e_3 \text{ else } e_4 : \tau, \textbf{if } me_1 \; me_2 \; me_3 \; me_4 \; \Phi \]

\textbf{if } me_1 \; me_2 \; me_3 \; me_4 \; \Phi = ?

- wrong to join the results of the two branches.
  \[ \text{if } x \sqsubseteq c \text{ then } \top \text{ else } \bot \]

- have to examine:
  - in which direction the \textit{if}-condition changes
  - whether the consequent change of branches preserves the monotonicity
\[
\text{if } me_1(x) \text{ and } me_2(x) \text{ and } me_3(x) \text{ and } me_4(x) \Phi
\]

For “if \( e_1 \sqsubseteq e_2 \) then \( e_3 \) else \( e_4 \)”

<table>
<thead>
<tr>
<th>( me_1(x) )</th>
<th>( me_2(x) )</th>
<th>( me_3(x) )</th>
<th>( me_4(x) )</th>
<th>( \Phi )</th>
<th>( \text{if } me_1(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>( e_3(\bot) \sqsubseteq e_4(\top) )</td>
<td>( e_3(\bot) \sqsubseteq e_4(\top) )</td>
</tr>
<tr>
<td>–</td>
<td>+</td>
<td>–</td>
<td>–</td>
<td>( e_3(\bot) \sqsubseteq e_4(\top) )</td>
<td>( e_3(\bot) \sqsubseteq e_4(\top) )</td>
</tr>
<tr>
<td>+</td>
<td>–</td>
<td>+</td>
<td>+</td>
<td>( e_3(\top) \sqsubseteq e_4(\bot) )</td>
<td>( e_3(\top) \sqsubseteq e_4(\bot) )</td>
</tr>
<tr>
<td>+</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>( e_3(\top) \sqsubseteq e_4(\bot) )</td>
<td>( e_3(\top) \sqsubseteq e_4(\bot) )</td>
</tr>
</tbody>
</table>

- When \( e_1 \sqsubseteq e_2 \) switches from false to true: if the ‘false’ branch does not exceed the ‘true’ branch, then the \( \textit{if} \)-expression is increasing.

- The \( \Phi \) parameter ensures that monotonicity will be preserved at the switching point.
The Rules (IV)

We can sharpen the IF-rule for popular cases in Rabbit programs:

\[
\frac{\Gamma \vdash e_3 : \tau, me_3}{\Gamma \vdash \text{if } x \sqsubseteq c \text{ then } e_3 \text{ else } e_4 : \tau, \text{ifc } x \ me_3 \ me_4 \ \Phi}
\]

(IFC)

\[
\text{ifc } x \ me_3 \ me_4 \ \Phi = \{ y \mapsto \begin{cases} \text{ifc } me_3(y) \ me_4(y) \ \Phi, & \text{if } y = x \\ me_3(y) \sqcup me_4(y), & \text{otherwise} \end{cases}\}
\]

where

<table>
<thead>
<tr>
<th>(me_3(x))</th>
<th>(me_4(x))</th>
<th>(\Phi)</th>
<th>(\text{ifc } me_3(x) \ me_4(x) \ \Phi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\vdash)</td>
<td>(\vdash)</td>
<td>(\forall d \in \hat{\gamma} \ a(x) \sqsubseteq a(d))</td>
<td>(\vdash)</td>
</tr>
</tbody>
</table>
Correctness

Theorem 1  If $\Gamma \vdash e : \tau, me$ then $s \models \Gamma, s' \models \Gamma, s \sqsubseteq s'|_x,$ $s \vdash e \Rightarrow v,$ and $s' \vdash e \Rightarrow v'$ imply $v \operatorname{me}(x) v'.$

Proof. Structural induction on $e.$
Algorithm

Two phases

- derive constraints for types and monotonicity effects
- solve the constraints
  - type constraints by the unification
  - monotonicity constraints by the fixpoint iteration

Observations

- type constraints: equality constraints with variables for the effects
- monotonicity constraints: set constraints with monotonic operators ($\sqcup$, $\oplus$, $\text{if}$, and $\text{ifc}$)
  - the least model for the constraints = the least fixed point of the corresponding equations
Constraints $\rho$:

$$
\rho ::= \tau_1 \leq \tau_2 \mid me_1 \supseteq me_2 \\
\quad \mid \exists \alpha. \rho \mid \exists \beta. \rho \\
\quad \mid \rho_1, \rho_2
$$

where

$$
\tau ::= \text{as before} \mid \alpha \quad (\text{type variable}) \\
me ::= \text{as before} \mid \beta \quad (\text{monotonicity variable})
$$
Validity of the Constraints

\[
\begin{align*}
\vdash \tau = \tau & \quad \vdash \{\tau / \alpha\} \rho \\
& \quad \vdash \exists \alpha. \rho \\
\vdash \{me / \beta\} \rho & \quad \vdash \exists \beta. \rho \\
\end{align*}
\]

\[
\begin{align*}
\vdash me_1 \supseteq me_2 & \quad \vdash \rho_1, \rho_2 \\
\vdash me_1 \supseteq me_2 & \quad \vdash \rho_1, \rho_2 \\
\end{align*}
\]
Constraint Extraction Procedure

\[ C(\Gamma, c, \tau, me) = \tau \Downarrow \iota, me \supseteq \bar{0} \]

\[ C(\Gamma, \lambda x.e, \tau, me) = \exists \alpha_1 \alpha_2 \beta_1 \beta_2 \beta'_2. \]

\[ C(\Gamma + x : (\alpha_1, \beta_1), e, \alpha_2, \beta'_2), \]

\[ \tau \Downarrow (\alpha_1, \beta_1) \rightarrow (\alpha_2, \beta_2), \quad \beta_2 \supseteq \beta'_2, \]

\[ me \supseteq \beta_2[0/x] \]

\[ \Gamma + x : (\tau_1, me_1) \vdash e : \tau_2, me'_2 \quad me'_2 \sqsubseteq me_2 \]

\[ \Gamma \vdash \lambda x.e : (\tau_1, me_1) \rightarrow (\tau_2, me_2), me_2[0/x] \]
The Extractor Is Correct

Theorem 2 \[ \vdash C(\Gamma, e, \tau, me) \iff \Gamma \vdash e : \tau, me' \textbf{ and } me' \sqsubseteq me. \]

Proof. Structural induction on \( e \).
Observations on $C(\Gamma, e, \alpha, \beta)$

- every $me$ in “$(\tau, me)$” is a variable $\beta_i$.
  - can apply the unification procedure
  - $\beta'/\beta$ as $\beta' \supseteq \beta$, $\beta \supseteq \beta'$

- every $me_1$ in “$me_1 \supseteq me_2$” is a variable $\beta_i$.
  - “$\beta \supseteq me_1, \cdots, \beta \supseteq me_k$” as “$\beta = me_1 \sqcup \cdots \sqcup me_k$”
  - can apply the least fixpoint iteration
The constraint extraction procedure $C$ takes $O(n)$.

The number of generated constraints is $O(n)$.

The unification takes $O(n)$.

The fixpoint iteration takes $O(n^3)$.

- $O(n^2)$ iterations: $O(n)$ unknowns and lattice $\text{Var} \rightarrow \{0, +, -, \top\}$'s height is $2 \times |\text{Var}|$
- Each iteration takes $O(n)$: $me_i$ has $O(|\text{Var}|)$ entries
Conclusion

- a monotonicity check for higher-order functions over finite-height lattices.

Go back.
Conclusion

- a monotonicity check for higher-order functions over finite-height lattices.
- it prevents Zoo from generating divergent analyzers, or from generating extra “joining” operations.

Go back.
Conclusion

- a monotonicity check for higher-order functions over finite-height lattices.
- it prevents Zoo from generating divergent analyzers, or from generating extra “joining” operations.
- an effect-type system: mono-variant and flow-insensitive.

Go back.
Conclusion

- a monotonicity check for higher-order functions over finite-height lattices.
- it prevents Zoo from generating divergent analyzers, or from generating extra “joining” operations.
- an effect-type system: mono-variant and flow-insensitive.
- its effectiveness remains to be seen.

Go back.