A Fixpoint Logic and Dependent Effects for Temporal Property Verification

Yoji Nanjo\textsuperscript{1}, Hiroshi Unno\textsuperscript{1}, Eric Koskinen\textsuperscript{2}, Tachio Terauchi\textsuperscript{3}

\textsuperscript{1} University of Tsukuba \textsuperscript{2} Stevens Institute of Technology \textsuperscript{3} Waseda University
Temporal Property Verification

Program $P$ ? Temporal property $\Phi$

Check whether $P$ satisfies $\Phi$
This Work

Higher-order functional program

Value-dependent temporal property

Check whether $P$ satisfies $\Phi$ by using
(1) a dependent refinement type & effect system and
(2) a deductive system for a first-order fixpoint logic
Main Contribution

• Foundation for **compositional & algorithmic** verification of **value-dependent temporal** properties of higher-order programs
  • cf. previous proposals are:
    • fully automated but whole program analysis [Kobayashi+ PLDI’11], [U.+ POPL’13], [Kuwahara+ ESOP’14], [Kuwahara+ CAV’15], [Murase+ POPL’16]
    • compositional but no support of the class of properties [Koskinen+ CSL-LICS’14], [U.+ POPL’18]
This Work

Higher-order functional program

Value-dependent temporal property

\[ P \models \Phi \]

Check whether $P$ satisfies $\Phi$ by using
(1) a dependent refinement type & effect system and
(2) a deductive system for a first-order fixpoint logic
Example: Functional Program

```
let rec send_msgs n =
  if n = 0 then ()
  else (event[Send]; send_msgs (n-1))
```

Generated event sequences:
- $n < 0 : \text{Send}^\omega$ (infinite repetition of \textit{Send})
- $n = 0 : \epsilon$ (empty sequence)
- $n = 1 : \text{Send}$
- $n = 2 : \text{Send}, \text{Send}$
  \[\vdots\]
This Work

Higher-order functional program

\[ P \models \Phi \]

Value-dependent temporal property

Check whether \( P \) satisfies \( \Phi \) by using

1. a dependent refinement type & effect system and
2. a deductive system for a first-order fixpoint logic
This Work

\[ P \models (\Phi_\mu, \Phi_\nu) \]

Check whether \textit{finite} event sequences generated by \( P \) satisfy \( \Phi_\mu \) and \textit{infinite} event sequences generated by \( P \) satisfy \( \Phi_\nu \).
Example: Value-Dependent Temporal Property

let rec send_msgs n =
  if n = 0 then
    ()
  else
    (event[Send]; send_msgs (n-1))

For terminating executions

\[ \Phi^\mu \equiv \lambda x \in \Sigma^*. x = \text{Send}^n \]

For diverging executions

\[ \Phi^\nu \equiv \lambda x \in \Sigma^{\omega}. x = \text{Send}^\omega \]

For diverging executions

\[ \Phi^\mu \equiv \lambda x \in \Sigma^*. x = \text{Send}^n \]

n-times repetition of Send

\[ \Phi^\nu \equiv \lambda x \in \Sigma^{\omega}. x = \text{Send}^\omega \]

infinite repetition of Send

\[ n < 0 : \text{Send}^\omega \]
\[ n = 0 : \epsilon \]
\[ n = 1 : \text{Send} \]
\[ n = 2 : \text{Send, Send} \]
\[ \vdots \]
Further Examples

- See the paper for further examples that demonstrate the range of applications

<table>
<thead>
<tr>
<th>Amortized Complexity</th>
<th>Higher-Order</th>
<th>Web Server Fairness</th>
</tr>
</thead>
<tbody>
<tr>
<td>let rev l =</td>
<td>let rec zoom () =</td>
<td></td>
</tr>
<tr>
<td>let rec aux l acc = match l with</td>
<td>event[Zoom]; zoom ()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[] -&gt; acc</td>
<td>h::t -&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>event[Tick]; aux t (h::acc)</td>
</tr>
<tr>
<td></td>
<td>in aux l []</td>
<td></td>
</tr>
<tr>
<td>let is_empty (l1,l2) = l1 = [] &amp;&amp; l2 = []</td>
<td>let rec shrink t f d =</td>
<td></td>
</tr>
<tr>
<td>let enqueue e (l1,l2) = event[Enq];(l1,e::l2)</td>
<td>if f () &lt;= 0 then</td>
<td></td>
</tr>
<tr>
<td>let rec dequeue (l1,l2) = match l1 with</td>
<td>zoom ()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[] -&gt; dequeue (rev l2, [])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e::l1' -&gt; event[Deq]; (e, (l1', l2))</td>
<td></td>
</tr>
<tr>
<td>let rec main (l1,l2) =</td>
<td>else</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if * then main (enqueue 42 (l1,l2))</td>
<td>(event[Shrink];</td>
</tr>
<tr>
<td></td>
<td>else if is_empty (l1,l2) then ()</td>
<td>let t' = f() - d in</td>
</tr>
<tr>
<td></td>
<td>else main (snd (dequeue (l1,l2)))</td>
<td>shrink t' (fun x -&gt; t') d)</td>
</tr>
<tr>
<td>main : ((l1,l2) : int list × int list) → (unit &amp; Φ)</td>
<td>let shrinker t d =</td>
<td></td>
</tr>
<tr>
<td>Φ̃ = λx.λEnq(x) +</td>
<td>l1</td>
<td>shrinker t (fun x -&gt; t) d</td>
</tr>
<tr>
<td>l2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Φ̃ = λx.T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t : {t \mid t ≥ 0}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d : {d \mid d &gt; 0 \land t mod d = 0}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(unit &amp; Φ)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Φ̃ = λx.⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Φ̃ = λx.x ∈ Shrink^{t/d} . Zoom^{0}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>server : (npool : {\nu \mid \nu ≥ 0}) → (unit &amp; (λx.⊥, λx.φ))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>φ = \left( x ∈ (Σ^* \setminus \text{Accept})^{npool+1}_ω \right)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\Rightarrow x ∈ (Σ^* \cdot \text{Wait})^{ω}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Check whether $P$ satisfies $\Phi$ via

(1) a dependent refinement type & effect system and

(2) a deductive system for a first-order fixpoint logic
Contributions

1. A dependent refinement type & effect system for compositional & algorithmic temporal verification
   • Compositional analysis of dependent temporal effects represented by predicates of first-order fixpoint logic $\mathcal{L}$
   • Algorithmic type checking via validity checking for $\mathcal{L}$

2. A deductive system for the validity of $\mathcal{L}$
   • Use invariants and well-founded relations to over- and under-approximate fixpoints
     • Designed by transferring ideas from verification research
   • Can be used with any background first-order theory
     • Enable other applications to program verification, which will be presented at the HCVS workshop on 13th 2018/7/11
Contributions

1. A dependent refinement type & effect system for \textit{compositional} & \textit{algorithmic} temporal verification
   - \textbf{Compositional} analysis of dependent temporal effects represented by predicates of \textit{first-order fixpoint logic} $\mathcal{L}$
   - \textbf{Algorithmic} type checking via validity checking for $\mathcal{L}$

2. A deductive system for the validity of $\mathcal{L}$
   - Use \textit{invariants} and \textit{well-founded relations} to over- and under-approximate fixpoints
     - Designed by transferring ideas from verification research
   - Can be used with \textit{any} background \textit{first-order theory}
     - Enable other applications to program verification, which will be presented at the HCVS workshop on 13th
First-Order Fixpoint Logic $\mathcal{L}$

- First-order logic extended with least fixpoints (LFPs) and greatest fixpoints (GFPs)

We here fix the theory as the one above for *temporal effect analysis*, though we could choose any background first-order theory.
Temporal Effect Analysis

Let rec send_msgs n =
if n = 0 then ()
else (event[Send]; send_msgs (n-1))

Example:
The use of first-order fixpoint logic allows precise representation (cf. previous work only allowed \((\omega-)\)regular expressions [Skalka+'08, Hofmann+'14] or did not specify the effect language [Koskinen+'14])

\[ \Phi_e^\mu \equiv \lambda x \in \Sigma^*. (\mu X_\mu (n, x)). \]
\[ \Phi_e^\nu \equiv \lambda x \in \Sigma^\omega. (\nu X_\nu (n, x)). n \neq 0 \land (\exists y. x = \text{Send} \cdot y \land X_\mu(n-1, y)) \]
Dependent Refinement Type & Effect System

Key typing rules:

\[
\begin{align*}
\Gamma \vdash e_1 : (\tau_1 & \& \Phi_1) & \quad \Gamma, x : \tau_1 \vdash e_2 : (\tau_2 & \& \Phi_2) \\
\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : (\tau_2 & \& \Phi_1 \cdot \Phi_2)
\end{align*}
\]

Sequential composition of effects

\[
\Phi_1 \cdot \Phi_2 = (\lambda x \in \Sigma^*. \exists x_1, x_2 \in \Sigma^*. x = x_1 \cdot x_2 \land \Phi_1(x_1) \land \Phi_2(x_2), \lambda x \in \Sigma^*. \Phi_1(x) \lor (\exists y \in \Sigma^*, z \in \Sigma^*. x = y \cdot z \land \Phi_1(y) \land \Phi_2(z)))
\]

Fixpoints describing a dependent temporal effect of a recursive function

\[
\begin{align*}
\Gamma \vdash \text{rec}(f, \tilde{x}, e) : (\tau_f & \& \Phi_{\text{val}})
\end{align*}
\]

Check sub-effect relation via fixpoint logic deduction

Theorem 1 (Soundness): \(\Gamma \vdash e : (\tau & (\Phi^\mu, \Phi^\nu))\) implies \(e \in [\Gamma \vdash \tau & (\Phi^\mu, \Phi^\nu)]\)

(e behaves as specified by \((\tau & (\Phi^\mu, \Phi^\nu))\) under a valuation conforming to \(\Gamma\))

Extends existing refinement type systems [Koskinen+'14, Rondon+'08, U.+'09, Terauchi'10, …]

• Types & effects facilitate compositional analysis of dependent temporal effects
• Fixpoint logic deduction enables algorithmic type checking
Contributions

1. A dependent refinement type & effect system for compositional & algorithmic temporal verification
   • Compositional analysis of dependent temporal effects represented by predicates of first-order fixpoint logic $\mathcal{L}$
   • Algorithmic type checking via validity checking for $\mathcal{L}$

2. A deductive system for the validity of $\mathcal{L}$
   • Use invariants and well-founded relations to over- and under-approximate fixpoints
     • Designed by transferring ideas from verification research
   • Can be used with any background first-order theory
     • Enable other applications to program verification, which will be presented at the HCVS workshop on 13th

2018/7/11 LICS'18, Oxford, UK 17
First-Order Fixpoint Logic $\mathcal{L}$ (revisited)

• First-order logic extended with least fixpoints (LFPs) and greatest fixpoints (GFPs)

(formulas) $\phi ::= \top \mid \bot \mid A(\tilde{t}) \mid \neg \phi \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid \forall x \in S. \phi \mid \exists x \in S. \phi \mid X(\tilde{t}) \mid (\mu X(\tilde{x}: \tilde{S}). \phi)(\tilde{t}) \mid (\nu X(\tilde{x}: \tilde{S}). \phi)(\tilde{t})$

(terms) $t ::= x \mid f(\tilde{t})$

predicate symbols of the background theory

sorts (e.g. $\mathbb{Z}$) of the background theory

LFPs ($X$ occurs only positively in $\phi$)

GFPs ($X$ occurs only positively in $\phi$)

function symbols of the background theory

predicate variables
Deductive System $\vdash \phi$ for the Validity of $\mathcal{L}$

1. Over- and under-approximate fixpoint subformulas of $\phi$ by non-fixpoint formulas
   - For soundness, subformulas that occur positively and negatively are respectively under- and over-approximated

2. Resulting non-fixpoint formulas are discharged by a solver for the background first-order theory

- Techniques for obtaining approximations:

<table>
<thead>
<tr>
<th>Over-Approximation</th>
<th>Under-Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFP</td>
<td>Invariant (induction)</td>
</tr>
<tr>
<td>GFP</td>
<td>Well-founded relation</td>
</tr>
<tr>
<td></td>
<td>Invariant (co-induction)</td>
</tr>
</tbody>
</table>

Analogous to techniques in safety and liveness property verification
Example: Fixpoint Deduction via Over-Approx. of LFP

Check that $p$ is a pre-fixpoint of $F$ (or, equivalently, perform induction by unfolding LFP and applying I.H. to the recursive occurrences of $X$)

Deduction in background first-order theory

Over-approx. of LFP by pre-fixpoint

\[ \lambda x \in \Sigma^*. x = \text{Send}^n \]

where $F(X)(n, x) = \left( \begin{array}{l}
  n = 0 \land x = \epsilon \\
  n \neq 0 \land (\exists y. x = \text{Send} \cdot y \land X(n - 1, y))
\end{array} \right) \]
Example: Fixpoint Deduction via Over-Approx. of GFP

Check that the given well-founded relation $p_2$ witnesses that the given predicate $p_1$ and $\Phi_e^n$ have no intersection (see the paper for details).

Deduction in background first-order theory

Over-approx. of GFP by negation of $p_1$
Theorem 2 (Soundness of $\vdash$): $\vdash \phi$ implies $\models \phi$

\[ \models \psi \vdash \phi \quad \text{FP-VALID} \]
\[ \vdash [(\lambda \bar{x}.\psi')/X] \psi \Rightarrow \psi' \quad \vdash C^-\left[\dfrac{\bar{t}/\bar{x}}{\psi'}\right] \]
\[ \vdash C^-\left[(\mu X(\bar{x}).\psi)(\bar{t})\right] \]

\[ \models \psi' \Rightarrow [(\lambda \bar{x}.\psi')/X] \psi \quad \vdash C^+\left[\dfrac{\bar{t}/\bar{x}}{\psi'}\right] \]
\[ \models \vdash C^+\left[(\nu X(\bar{x}).\psi)(\bar{t})\right] \quad \text{FP-GFP}^+ \]

\[ X(\bar{x}); p_1; p_2; \top \downarrow \text{nnf}(\psi) \quad \vdash C^+\left[p_1(\bar{t})\right] \quad \models \text{WF}(p_2) \]
\[ \models \vdash C^+\left[(\mu X(\bar{x}).\psi)(\bar{t})\right] \quad \text{FP-LFP}^+ \]

\[ X(\bar{x}); p_1; p_2; \top \uparrow \text{nnf}(\psi) \quad \vdash C^-\left[\neg p_1(\bar{t})\right] \quad \models \text{WF}(p_2) \]
\[ \models \vdash C^-\left[(\nu X(\bar{x}).\psi)(\bar{t})\right] \quad \text{FP-GFP}^- \]

\[ \text{Over-approximation of LFP (induction)} \]
\[ \text{Under-approximation of GFP (co-induction)} \]

\[ \psi \text{ represents a fixpoint-free formula.} \]
\[ \text{nnf}(\psi) \text{ is negation normal form of } \psi. \]
\[ C^+ \text{ (resp. } C^-\text{) is positive (resp. negative) context.} \]

Background first-order theory solver

Approximation of fixpoints using well-founded relation (see the paper for details)
Conclusion

• Foundation for **compositional & algorithmic** verification of **value-dependent temporal** properties of higher-order programs

1. Dependent refinement type & effect system
   • **Compositional** analysis of **dependent temporal effects** represented by predicates of **first-order fixpoint logic** $\mathcal{L}$
   • **Algorithmic** type checking via validity checking for $\mathcal{L}$

2. Deductive system for the validity of $\mathcal{L}$

<table>
<thead>
<tr>
<th></th>
<th>Over-Approximation</th>
<th>Under-Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFP</td>
<td>Invariant (induction)</td>
<td>Well-founded relation</td>
</tr>
<tr>
<td>GFP</td>
<td>Well-founded relation</td>
<td>Invariant (co-induction)</td>
</tr>
</tbody>
</table>

• Can be used with **any background first-order theory**
  • Enable other applications to program verification, which will be presented at the HCVS workshop on 13th