Dependent Temporal Effects and Fixpoint Logic for Verification

Yoji Nanjo[1], Hiroshi Unno[1], Eric Koskinen[2], Tachio Terauchi[3]
[1]University of Tsukuba [2]Stevens Institute of Technology [3]Waseda University

Our Goal: Temporal Verification of Higher-Order Functional Programs

functional program: $e \stackrel{?}{\models} (\Phi^\mu, \Phi^\nu)$ dependent temporal spec.

Φ^μ: predicate on event sequences and program values that must be satisfied by finite (i.e., terminating) event sequences produced by e

Φ^ν: predicate on event sequences and program values that must be satisfied by infinite (i.e., diverging) event sequences produced by e

Extend beyond (non-dependent) temporal specs. used in previous work

Example functional program:

raise **Send** event

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

$n < 0$: **Send**, **Send**, **Send**, **Send**, **Send**, ⋯ (infinite)
$n = 0$: ϵ
$n = 1$: **Send**
$n = 2$: **Send**, **Send**
…

Example dependent temporal spec.: (the argument of send_msgs)

$$\Phi^\mu \equiv \lambda x \in \Sigma^*.\, x \in \mathbf{Send}^n$$
$$\Phi^\nu \equiv \lambda x \in \Sigma^\omega.\, x \in \mathbf{Send}^\omega$$

Example: amortized complexity analysis of an implementation of queues

```
let rev l = let rec aux l acc = match l with
  | [] -> acc | h::t -> event[Tick]; aux t (h::acc)
  in aux l []
let enqueue e (l1,l2) = event[Enq]; (l1,e::l2)
let rec dequeue (l1,l2) = match l1 with
  | e::l1' -> event[Deq]; (e, (l1', l2))
  | [] -> dequeue (rev l2, [])
let rec loop (l1,l2) =
  if * then loop (enqueue 42 (l1,l2))
  else if is_empty (l1,l2) then ()
  else loop (snd (dequeue (l1,l2)))
let main () = loop ([], [])
```

Enq, Enq, Tick, Tick, Deq, Deq
Enq, Tick, Deq, Enq, Tick, Deq
Enq, Enq, Tick, Tick, Deq, Deq, Enq, Tick, Deq

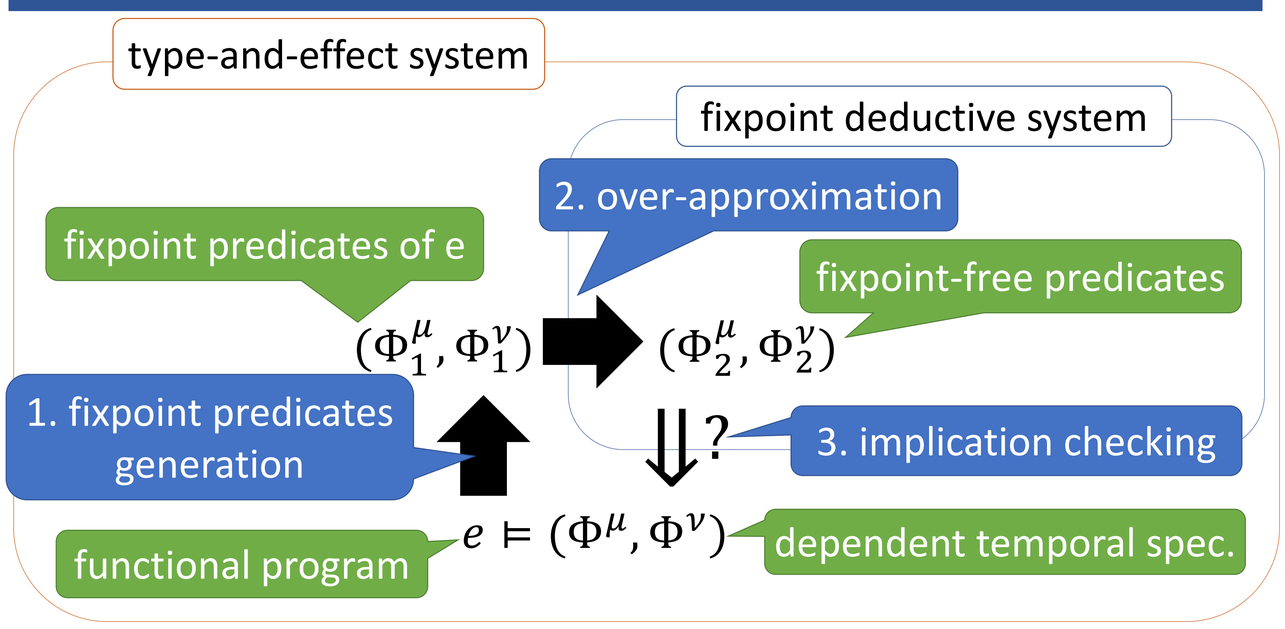
$$\Phi^\mu \equiv \lambda x \in \Sigma^*.\, x \neq \epsilon \Rightarrow \frac{\#_{\mathbf{Tick}}(x)}{\#_{\mathbf{Deq}}(x)} = 1$$

where $\#_a(x)$ is the number of occurrences of the event a in the sequence x

Our Contributions

- A type-and-effect system for dependent temporal verification
- A soundness proof of the type-and-effect system
- A fixpoint deductive system for reasoning about fixpoint predicates
- A soundness proof of the fixpoint deductive system

Overall Verification Flow



1. Fixpoint Predicates Generation

compositionally analyze temporal effects of e via typing rules

functional program e ⇒ (Φ_1^μ, Φ_1^ν) fixpoint predicates of e

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

$$\Phi_1^\mu \equiv \lambda x \in \Sigma^*.\, (\mu X_\mu(n,x).\, (n = 0 \wedge x = \epsilon \vee n \neq 0 \wedge (\exists y.\, x = \mathbf{Send} \cdot y \wedge X_\mu(n-1, y)))(n,x)$$

predicate variable that relates the argument n and the finite sequence x

$$\Phi_1^\nu \equiv \lambda x \in \Sigma^\omega.\, (\nu X_\nu(n,x).\, n \neq 0 \wedge (\exists y.\, x = \mathbf{Send} \cdot y \wedge X_\nu(n-1, y)))(n,x)$$

predicate variable that relates the argument n and the infinite sequence x

2. Over-Approximation

Least Fixpoints

fixpoint predicate: $\Phi_1^\mu \equiv \lambda x \in \Sigma^*.\, (\mu X_\mu.\, F(X_\mu))(n,x)$

check that Φ_2^μ is a *pre*-fixpoint of the function $F \equiv \lambda X.\, \lambda(n,x).\, n = 0 \wedge x = \epsilon \vee n \neq 0 \wedge (\exists y.\, x = \mathbf{Send} \cdot y \wedge X(n-1, y))$

fixpoint-free predicate: $\Phi_2^\mu \equiv \lambda x \in \Sigma^*.\, n > 0 \wedge x \in \mathbf{Send}^n$

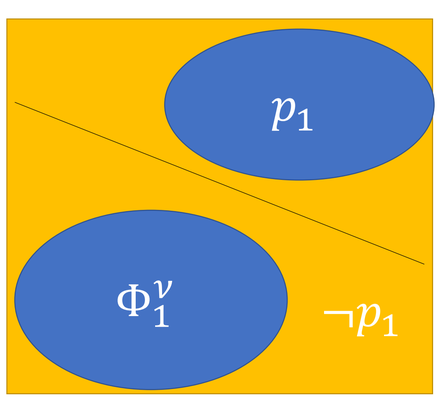
Greatest Fixpoints

fixpoint predicate: $\Phi_1^\nu \equiv \lambda x \in \Sigma^\omega.\, (\nu X_\nu(n,x).\, n \neq 0 \wedge (\exists y.\, x = \mathbf{Send} \cdot y \wedge X_\mu(n-1, y)))(n,x)$

check that a given well-founded relation p_2 witnesses that a given predicate p_1 and Φ_1^ν are separated, and then return $\neg p_1$ as Φ_2^ν

$p_1 = \lambda x \in \Sigma^\omega.\, n \geq 0 \vee x \notin \mathbf{Send}^\omega, p_2 = \lambda(n_1, x_1, n_2, x_2).\, n_1 > n_2 \geq 0$

fixpoint-free predicate: $\Phi_2^\nu \equiv \lambda x \in \Sigma^*.\, n < 0 \wedge x \in \mathbf{Send}^\omega$



Formally, we introduce the relation $X(\tilde{x}); p_1; p_2; \psi' \,\overline{\uparrow}\, \psi$ which means the well-founded relation p_2 witnesses that $p_1(\tilde{x})$ implies $\neg(\nu X_\nu(\tilde{x}).\, \psi' \wedge \psi)(\tilde{x})$ for any $\tilde{x}$

The derivation rules (excerpt):

$$\frac{\models (p_1(\tilde{x}) \wedge \psi) \Rightarrow (\psi_1' \vee \psi_2') \quad fv(\psi_i') \subseteq \{\tilde{x}\} \quad X \notin fpv(\psi_i') \quad X(\tilde{x}); p_1; p_2; \psi \wedge \psi_i' \,\overline{\uparrow}\, \psi_i \quad (i = 1,2)}{X(\tilde{x}); p_1; p_2; \psi \,\overline{\uparrow}\, \psi_1 \wedge \psi_2}$$

$$\frac{X(\tilde{x}); p_1; p_2; \psi' \,\overline{\uparrow}\, [x'/x]\psi \quad x' \notin fv(\psi') \cup fv(\psi) \cup \{\tilde{x}\} \cup fv(p_1) \cup fv(p_2)}{X(\tilde{x}); p_1; p_2; \psi' \,\overline{\uparrow}\, \exists x.\, \psi}$$

Example derivation:

$$\frac{p_1(n,x) \wedge n \neq 0 \wedge x = \mathbf{Send} \cdot x' \Rightarrow (p_1(n-1, x') \wedge p_2(n, x, n-1, x'))}{\dfrac{\dfrac{\dfrac{\dfrac{X(n,x); p_1; p_2; n \neq 0 \wedge x = \mathbf{Send} \cdot x' \,\overline{\uparrow}\, X(n-1, x')}{X(n,x); p_1; p_2; n \neq 0 \,\overline{\uparrow}\, x = \mathbf{Send} \cdot x' \wedge X(n-1, x')}}{X(n,x); p_1; p_2; n \neq 0 \,\overline{\uparrow}\, \exists y.\, x = \mathbf{Send} \cdot y \wedge X(n-1, y)}}{X(n,x); p_1; p_2; \top \,\overline{\uparrow}\, n \neq 0 \wedge \exists y.\, x = \mathbf{Send} \cdot y \wedge X(n-1, y)}}$$