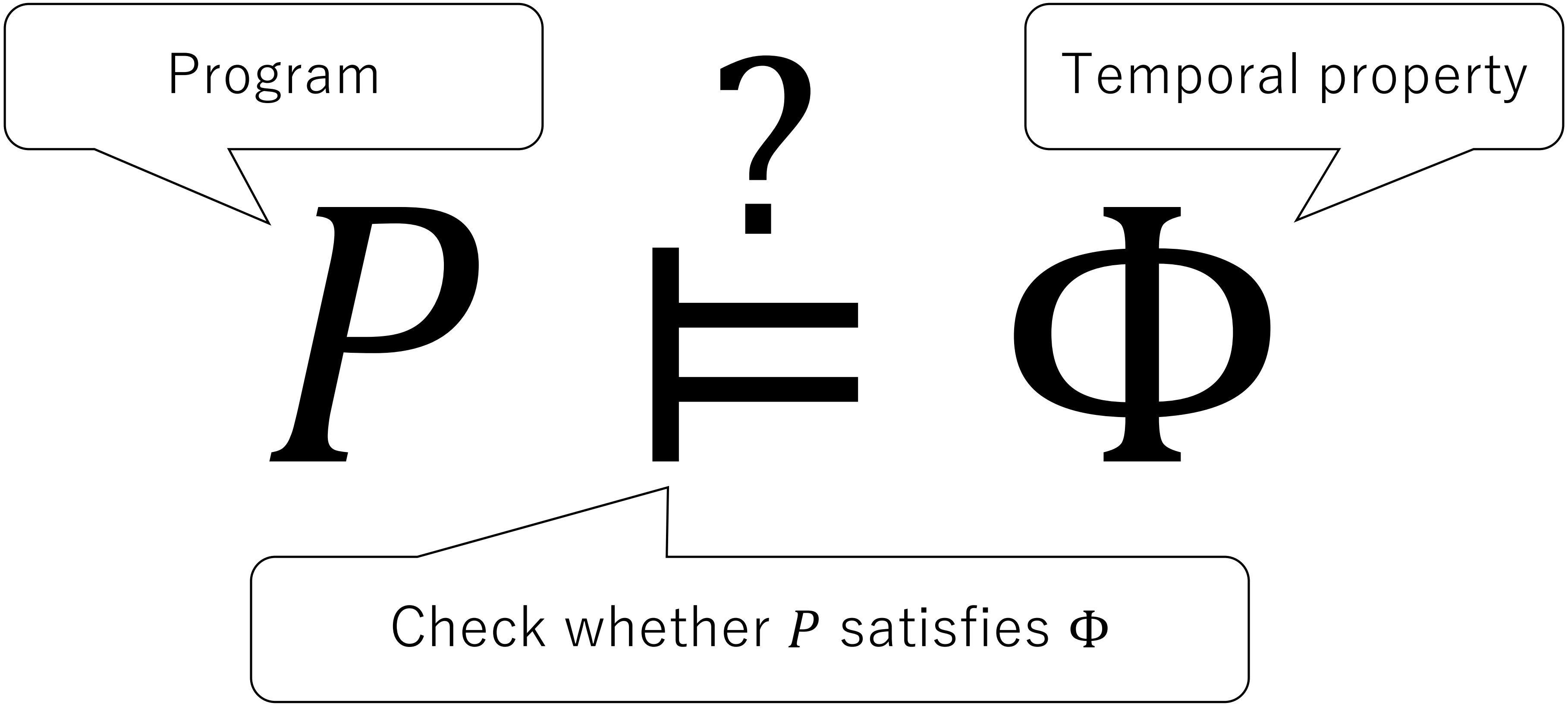
A Fixpoint Logic and Dependent Effects for Temporal Property Verification

Yoji Nanjo[1], **Hiroshi Unno**[1], Eric Koskinen[2], Tachio Terauchi[3]

[1] University of Tsukuba [2] Stevens Institute of Technology [3] Waseda University

Temporal Property Verification



Program

$$P \overset{?}{\models} \Phi$$

Temporal property

Check whether P satisfies Φ

This Work

Higher-order functional program

Value-dependent temporal property

$$P \overset{?}{\models} \Phi$$

Check whether P satisfies Φ 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

$$P \overset{?}{\models} \Phi$$

Check whether P satisfies Φ 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))
```

emit Send event

Generated event sequences:

$\mathsf{n} < 0 : \mathrm{Send}^{\omega}$ (infinite repetition of Send)

$\mathsf{n} = 0 : \epsilon$ (empty sequence)

$\mathsf{n} = 1 : \mathrm{Send}$

$\mathsf{n} = 2 : \mathrm{Send}, \mathrm{Send}$

$\vdots$

This Work

Higher-order functional program

Value-dependent temporal property

$$P \overset{?}{\models} \Phi$$

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

This Work

predicate for *finite* event sequences

predicate for *infinite* event sequences

$$P \overset{?}{\models} (\Phi_\mu, \Phi_\nu)$$

Check whether *finite* event sequences generated by P satisfy Φ_μ and *infinite* event sequences generated by P satisfy Φ_ν

Example: Value-Dependent Temporal Property

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

$\mathsf{n} < 0 : \mathrm{Send}^{\omega}$

$\mathsf{n} = 0 : \epsilon$

$\mathsf{n} = 1 : \mathrm{Send}$

$\mathsf{n} = 2 : \mathrm{Send}, \mathrm{Send}$

$\vdots$

$$\models$$

For terminating executions

n-times repetition of Send

$$\Phi^{\mu} \equiv \lambda x \in \Sigma^{*}. x = \mathrm{Send}^{\mathsf{n}}$$

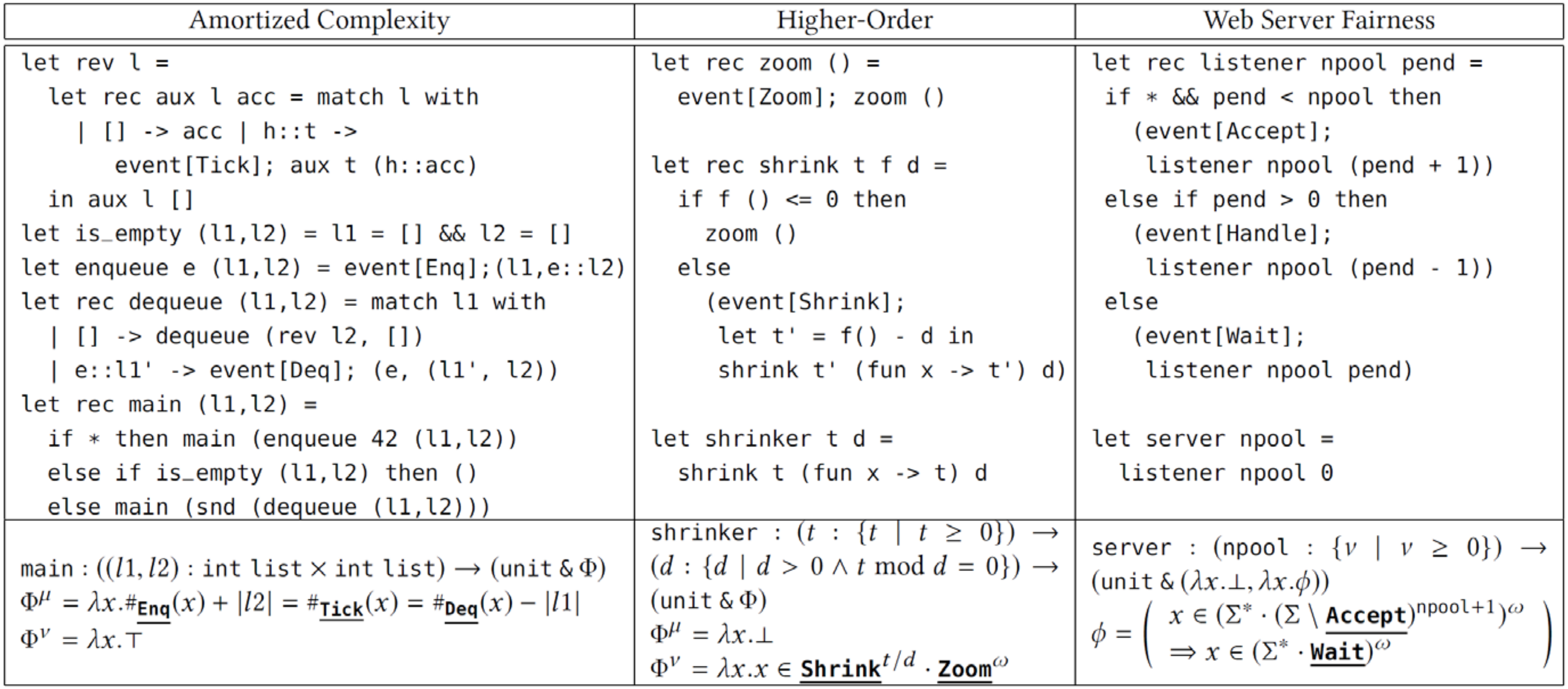
$$\Phi^{\nu} \equiv \lambda x \in \Sigma^{\omega}. x = \mathrm{Send}^{\omega}$$

For diverging executions

infinite repetition of Send

Further Examples

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



Amortized Complexity

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

$\texttt{main} : ((l1, l2) : \texttt{int list} \times \texttt{int list}) \to (\texttt{unit} \,\&\, \Phi)$

$\Phi^{\mu} = \lambda x.\#_{\underline{\texttt{Enq}}}(x) + |l2| = \#_{\underline{\texttt{Tick}}}(x) = \#_{\underline{\texttt{Deq}}}(x) - |l1|$

$\Phi^{\nu} = \lambda x.\top$

Higher-Order

```
let rec zoom () =
  event[Zoom]; zoom ()

let rec shrink t f d =
  if f () <= 0 then
    zoom ()
  else
    (event[Shrink];
     let t' = f() - d in
     shrink t' (fun x -> t') d)

let shrinker t d =
  shrink t (fun x -> t) d
```

$\texttt{shrinker} : (t : \{t \mid t \geq 0\}) \to (d : \{d \mid d > 0 \land t \bmod d = 0\}) \to (\texttt{unit} \,\&\, \Phi)$

$\Phi^{\mu} = \lambda x.\bot$

$\Phi^{\nu} = \lambda x.x \in \underline{\texttt{Shrink}}^{t/d} \cdot \underline{\texttt{Zoom}}^{\omega}$

Web Server Fairness

```
let rec listener npool pend =
 if * && pend < npool then
   (event[Accept];
    listener npool (pend + 1))
 else if pend > 0 then
   (event[Handle];
    listener npool (pend - 1))
 else
   (event[Wait];
    listener npool pend)

let server npool =
  listener npool 0
```

$\texttt{server} : (\texttt{npool} : \{\nu \mid \nu \geq 0\}) \to (\texttt{unit} \,\&\, (\lambda x.\bot, \lambda x.\phi))$

$$\phi = \left(\begin{array}{l} x \in (\Sigma^* \cdot (\Sigma \setminus \underline{\texttt{Accept}})^{\texttt{npool}+1})^{\omega} \\ \Rightarrow x \in (\Sigma^* \cdot \underline{\texttt{Wait}})^{\omega} \end{array} \right)$$

This Work

Higher-order functional program

Value-dependent temporal property

$$P \overset{?}{\models} \Phi$$

Check whether P satisfies Φ 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

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

First-Order Fixpoint Logic $\mathcal{L}$

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

predicate variables

predicate symbols of the background theory

$$\begin{array}{rcl} \text{(formulas) } \phi & ::= & \top \mid \bot \mid A(\widetilde{t}) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall x \in \mathcal{S}.\phi \mid \exists x \in \mathcal{S}.\phi \\ & \mid & X(\widetilde{t}) \mid (\mu X(\widetilde{x} \colon \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \mid (\nu X(\widetilde{x} \colon \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \\ \text{(terms) } t & ::= & x \mid f(\widetilde{t}) \\ \text{(sorts) } \mathcal{S} & ::\in & \{\mathbb{Z}, \Sigma^*, \Sigma^\omega\} \end{array}$$

LFPs (X occurs only positively in ϕ)

GFPs (X occurs only positively in ϕ)

function symbols of the background theory

the set of finite event sequences

the set of infinite event sequences

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

functional program

$e \Rightarrow (\Phi_e^\mu, \Phi_e^\nu)$

dependent temporal effect that describe the temporal behavior of e

Example:

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

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

predicate variable that relates n and the **finite** event sequence x

$$\Phi_e^\mu \equiv \lambda x \in \Sigma^*.\,(\mu X_\mu(n,x).\left(\begin{array}{c} n = 0 \ \wedge \ x = \epsilon \ \vee \\ n \neq 0 \ \wedge \ \left(\exists y.\ x = \mathbf{Send} \cdot y \ \wedge \ X_\mu(n-1,y)\right) \end{array}\right))(n,x)$$

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

predicate variable that relates n and the **infinite** event sequence x

Dependent Refinement Type & Effect System

$$\boxed{\Gamma} \vdash \boxed{e} : (\boxed{\tau} \,\&\, (\boxed{\Phi^{\mu}, \Phi^{\nu}}))$$

Type Environment

Program

Dependent Refinement Type

Dependent Temporal Effect

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 $\Vdash$ enables **algorithmic** type checking

Key typing rules:

$$\frac{x \notin fv(\tau_2) \cup fv(\Phi_2) \quad \Gamma \vdash e_1 : (\tau_1 \,\&\, \Phi_1) \quad \Gamma, x : \tau_1 \vdash e_2 : (\tau_2 \,\&\, \Phi_2)}{\Gamma \vdash \mathtt{let}\; x = e_1 \;\mathtt{in}\; e_2 : (\tau_2 \,\&\, \boxed{\Phi_1 \cdot \Phi_2})}$$

Sequential composition of effects

$$\begin{aligned}\Phi_1 \cdot \Phi_2 = (&\lambda x \in \Sigma^*.\exists x_1, x_2 \in \Sigma^*.x = x_1 \cdot x_2 \wedge \Phi_1^{\mu}(x_1) \wedge \Phi_2^{\mu}(x_2),\\ &\lambda x \in \Sigma^{\omega}.\Phi_1^{\nu}(x) \vee (\exists y \in \Sigma^*, z \in \Sigma^{\omega}.x = y \cdot z \wedge \Phi_1^{\mu}(y) \wedge \Phi_2^{\nu}(z)))\end{aligned}$$

$$\frac{\begin{array}{c}\tau_f' = (\widetilde{x} : \widetilde{\tau}) \rightarrow (\tau \,\&\, (\lambda x \in \Sigma^*.X_{\mu}(\widetilde{x}, x), \lambda x \in \Sigma^{\omega}.X_{\nu}(\widetilde{x}, x)))\\ \Gamma, f : \tau_f', \widetilde{x} : \widetilde{\tau} \vdash e : (\tau \,\&\, \Phi)\\ \boxed{q_{\mu} = \mu X_{\mu}(\widetilde{x}, x).\Phi^{\mu}(x) \quad q_{\nu} = \nu X_{\nu}(\widetilde{x}, x).[q_{\mu}/X_{\mu}]\Phi^{\nu}(x)}\\ \tau_f = (\widetilde{x} : \widetilde{\tau}) \rightarrow (\tau \,\&\, (\lambda x \in \Sigma^*.q_{\mu}(\widetilde{x}, x), \lambda x \in \Sigma^{\omega}.q_{\nu}(\widetilde{x}, x)))\end{array}}{\Gamma \vdash \mathtt{rec}(f, \widetilde{x}, e) : (\tau_f \,\&\, \Phi_{val})}$$

Fixpoints describing a dependent temporal effect of a recursive function

$$\frac{\Gamma \vdash e : \sigma_1 \quad \boxed{\Gamma \vdash \sigma_1 <: \sigma_2}}{\Gamma \vdash e : \sigma_2}$$

Subtyping

$$\frac{\Gamma \vdash \tau_1 <: \tau_2 \quad \boxed{\begin{array}{l}\Vdash \lfloor \Gamma \vdash \forall x \in \Sigma^*.\Phi_1^{\mu}(x) \Rightarrow \Phi_2^{\mu}(x) \rfloor\\ \Vdash \lfloor \Gamma \vdash \forall x \in \Sigma^{\omega}.\Phi_1^{\nu}(x) \Rightarrow \Phi_2^{\nu}(x) \rfloor\end{array}}}{\Gamma \vdash (\tau_1 \,\&\, \Phi_1) <: (\tau_2 \,\&\, \Phi_2)}$$

Check sub-effect relation via fixpoint logic deduction

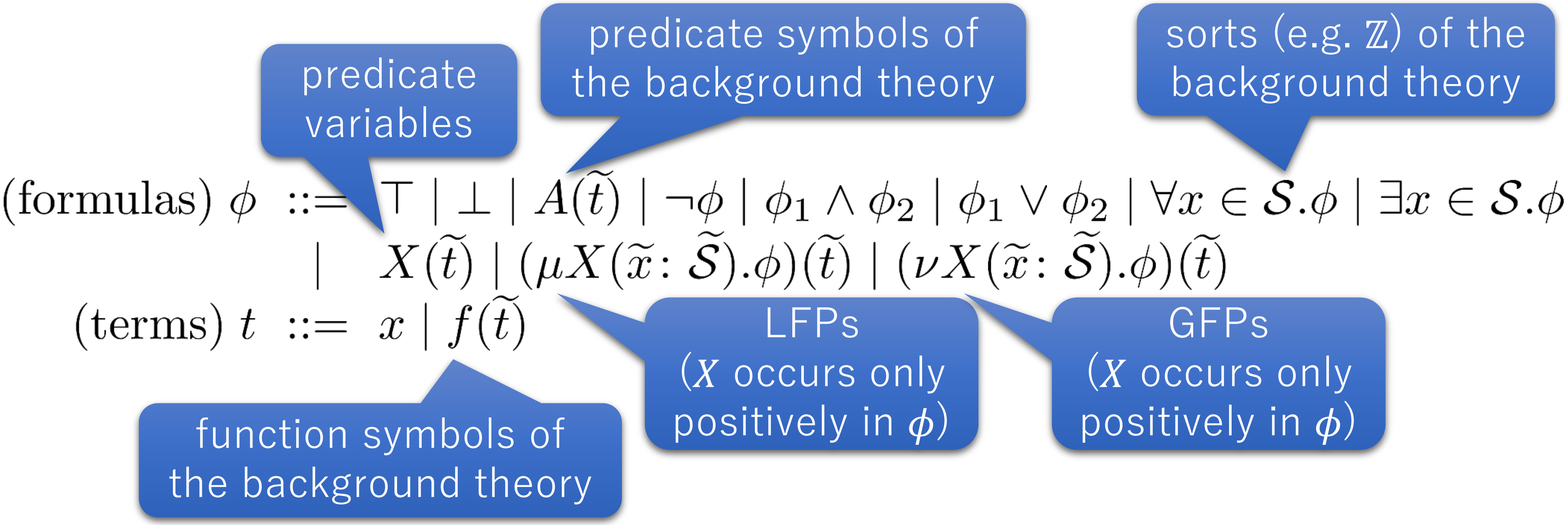
Theorem 1 (Soundness): $\Gamma \vdash e : \big(\tau \,\&\, (\Phi^{\mu}, \Phi^{\nu})\big)$ implies $e \in [\![\Gamma \vdash \tau \,\&\, (\Phi^{\mu}, \Phi^{\nu})]\!]$
(e behaves as specified by $\big(\tau \& (\Phi^{\mu}, \Phi^{\nu})\big)$ under a valuation conforming to Γ)

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

First-Order Fixpoint Logic $\mathcal{L}$ (revisited)

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



predicate variables

predicate symbols of the background theory

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

$$\begin{array}{rcl}
\text{(formulas) } \phi & ::= & \top \mid \bot \mid A(\widetilde{t}) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall x \in \mathcal{S}.\phi \mid \exists x \in \mathcal{S}.\phi \\
 & \mid & X(\widetilde{t}) \mid (\mu X(\widetilde{x} \colon \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \mid (\nu X(\widetilde{x} \colon \widetilde{\mathcal{S}}).\phi)(\widetilde{t}) \\
\text{(terms) } t & ::= & x \mid f(\widetilde{t})
\end{array}$$

function symbols of the background theory

LFPs
(X occurs only positively in ϕ)

GFPs
(X occurs only positively in ϕ)

Deductive System $\Vdash \phi$ for the Validity of $\mathcal{L}$

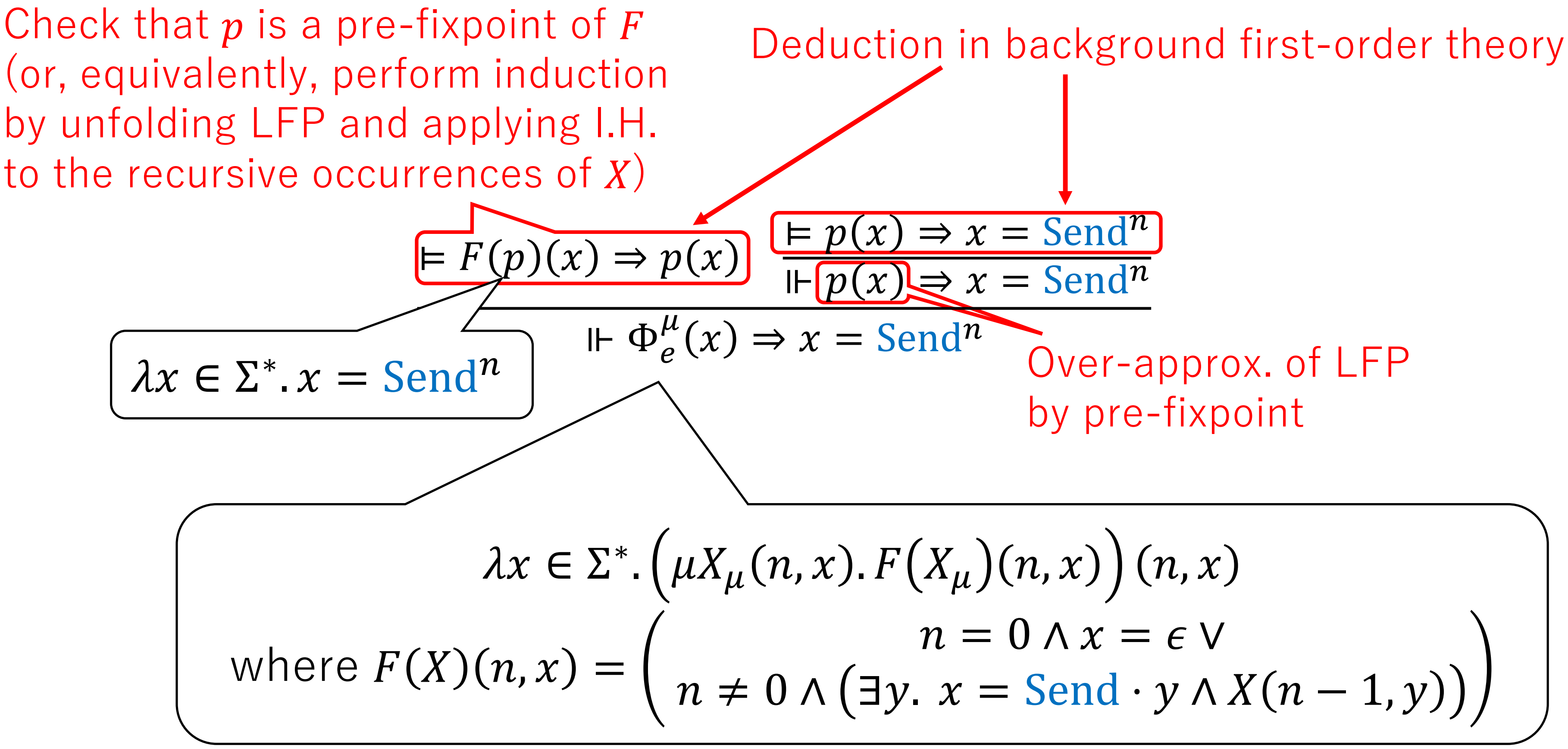
1. Over- and under-approximate fixpoint subformulas of ϕ by non-fixpoint formulas
    - For soundness, subformulas that occur positively and negatively are respectively under- and over-approximated
2. Resulting non-fixpoint formulas are discharged by a solver for the background first-order theory

- Techniques for obtaining approximations:

	Over-Approximation	**Under-Approximation**
LFP	**Invariant (induction)**	**Well-founded relation**
GFP	**Well-founded relation**	**Invariant (co-induction)**

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

$$\dfrac{\boxed{\vDash F(p)(x) \Rightarrow p(x)} \qquad \dfrac{\boxed{\vDash p(x) \Rightarrow x = \mathrm{Send}^n}}{\Vdash \boxed{p(x)} \Rightarrow x = \mathrm{Send}^n}}{\Vdash \Phi_e^{\mu}(x) \Rightarrow x = \mathrm{Send}^n}$$

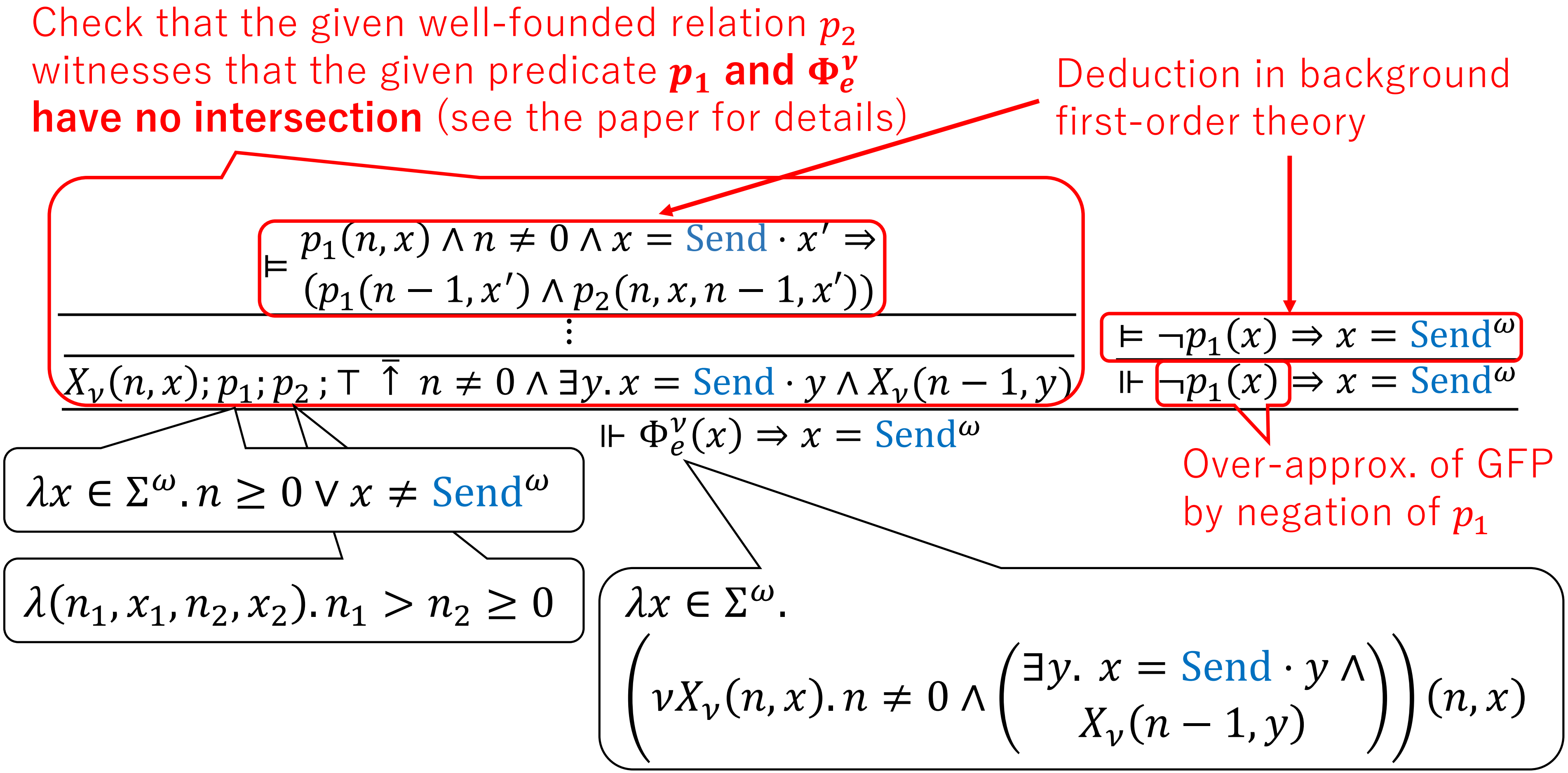
Over-approx. of LFP by pre-fixpoint

$$\lambda x \in \Sigma^*.\, x = \mathrm{Send}^n$$

$$\lambda x \in \Sigma^*. \left(\mu X_\mu(n,x). F\left(X_\mu\right)(n,x)\right)(n,x)$$

where $F(X)(n,x) = \left(\begin{array}{c} n = 0 \land x = \epsilon \lor \\ n \neq 0 \land \left(\exists y.\ x = \mathrm{Send} \cdot y \land X(n-1,y)\right) \end{array}\right)$

Example: Fixpoint Deduction via Over-Approx. of GFP



Check that the given well-founded relation p_2 witnesses that the given predicate $\boldsymbol{p_1}$ **and** $\boldsymbol{\Phi_e^\nu}$ **have no intersection** (see the paper for details)

Deduction in background first-order theory

$$\dfrac{\dfrac{\dfrac{\boxed{\models \begin{array}{l} p_1(n,x) \land n \neq 0 \land x = \mathrm{Send} \cdot x' \Rightarrow \\ (p_1(n-1,x') \land p_2(n,x,n-1,x')) \end{array}}}{\vdots}}{X_\nu(n,x); p_1; p_2; \top \ \bar{\uparrow}\ n \neq 0 \land \exists y. x = \mathrm{Send} \cdot y \land X_\nu(n-1,y)} \qquad \dfrac{\boxed{\models \neg p_1(x) \Rightarrow x = \mathrm{Send}^\omega}}{\Vdash \boxed{\neg p_1(x)} \Rightarrow x = \mathrm{Send}^\omega}}{\Vdash \Phi_e^\nu(x) \Rightarrow x = \mathrm{Send}^\omega}$$

Over-approx. of GFP by negation of p_1

$$\lambda x \in \Sigma^\omega. n \geq 0 \lor x \neq \mathrm{Send}^\omega$$

$$\lambda(n_1,x_1,n_2,x_2). n_1 > n_2 \geq 0$$

$$\lambda x \in \Sigma^\omega. \left(\nu X_\nu(n,x). n \neq 0 \land \begin{pmatrix} \exists y.\ x = \mathrm{Send} \cdot y \land \\ X_\nu(n-1,y) \end{pmatrix}\right)(n,x)$$

Deductive System $\Vdash \phi$ for $\mathcal{L}$

Background first-order theory solver

$$\frac{\models \psi}{\Vdash \psi}\ \textsc{Fp-Valid}$$

Over-approximation of LFP (induction)

$$\frac{\models [(\lambda\widetilde{x}.\psi')/X]\psi \Rightarrow \psi' \quad \Vdash C^-[[\widetilde{t}/\widetilde{x}]\psi']}{\Vdash C^-[(\mu X(\widetilde{x}).\psi)(\widetilde{t})]}\ \textsc{Fp-Lfp}^-$$

$$\frac{\models \psi' \Rightarrow [(\lambda\widetilde{x}.\psi')/X]\psi \quad \Vdash C^+[[\widetilde{t}/\widetilde{x}]\psi']}{\Vdash C^+[(\nu X(\widetilde{x}).\psi)(\widetilde{t})]}\ \textsc{Fp-Gfp}^+$$

Under-approximation of GFP (co-induction)

$$\frac{X(\widetilde{x}); p_1; p_2; \top \downarrow nnf(\psi) \quad \Vdash C^+[p_1(\widetilde{t})] \quad \models WF(p_2)}{\Vdash C^+[(\mu X(\widetilde{x}).\psi)(\widetilde{t})]}\ \textsc{Fp-Lfp}^+$$

$$\frac{X(\widetilde{x}); p_1; p_2; \top \overline{\uparrow}\, nnf(\psi) \quad \Vdash C^-[\neg p_1(\widetilde{t})] \quad \models WF(p_2)}{\Vdash C^-[(\nu X(\widetilde{x}).\psi)(\widetilde{t})]}\ \textsc{Fp-Gfp}^-$$

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

ψ represents a fixpoint-free formula.

$nnf(\psi)$ is negation normal form of ψ.

C^+ (resp. C^-) is positive (resp. negative) context.

Theorem 2 (Soundness of $\Vdash$): $\Vdash \phi$ implies $\models \phi$

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 $\mathcal{L}$

	Over-Approximation	Under-Approximation
LFP	**Invariant (induction)**	**Well-founded relation**
GFP	**Well-founded relation**	**Invariant (co-induction)**

   - 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