

A Relational Framework for Higher-Order Shape Analysis

Meta-theory

Calculus λ_R

x, y, z, ν	\in	<i>variables</i>	
C	$::=$	Cons Nil	<i>constructors</i>
v	$::=$	$\lambda(x : \tau). e \mid v : \tau \mid C \bar{v}$	<i>value</i>
e	$::=$	$x \mid e x \mid \text{let } x = e \text{ in } e$ $\mid \text{match } x \text{ with } C \bar{x} \Rightarrow e \text{ else } e$ $\mid e : \tau$	<i>expression</i>
T	$::=$	unit int intlist	<i>datatypes</i>
τ	$::=$	$\{\nu : T \mid \phi\} \mid x : \tau \rightarrow \tau$	<i>dependent types</i>

Specification Language

R	\in	<i>relation names</i>	
ϕ	$::=$	$r = r \mid r \subset r \mid \phi \wedge \phi \mid \phi \vee \phi \mid \top$	<i>type refinement</i>
r	$::=$	$R(x) \mid r \cup r \mid r \times r$	<i>relational expression</i>
Σ_R	$::=$	$\langle R, \tau_R, \overline{C \bar{x} \Rightarrow r} \rangle$ $\mid \langle R, \tau_R, R^* \rangle$	<i>relation definition</i>
θ	$::=$	$T \mid T * \theta$	<i>tuple sort</i>
τ_R	$::=$	$\text{intlist} \rightarrow \{\theta\}$	<i>relation sort</i>

Figure 1: Language

1. Core Language Metatheory

To simplify meta-theory, we use a calculus with propositional atoms as our intermediate language. We refer to the language as λ_ϕ . The language is described in 3.

For a λ_R type environment Γ , we define its λ_ϕ projection (denoted Γ^F) as following:

$$\begin{aligned} (\cdot)^F &= \cdot \\ (\Gamma, x : \{\nu : T \mid \phi\})^F &= \Gamma^F, x : \mathcal{F}(T) \\ (\Gamma, R :: \tau_R)^F &= \Gamma^F, R : \llbracket \tau_R \rrbracket_L \end{aligned}$$

LEMMA 1.1. *If $\Gamma^F \vdash \nu_1^F : \tau_1^F$ and $\Gamma^F \vdash \nu_2^F : \tau_2^F$, then $\gamma_\sqcup(\nu_1^F, \odot, \nu_2^F)$ is a λ_ϕ value ν^F such that $\Gamma^F \vdash \nu^F : \tau^F$, where $\odot \in \{\vee, \Rightarrow, \Leftrightarrow\}$*

Proof By structural induction over τ^F .

- Case **bool** : By inversion on type derivation of ν_1^F and ν_2^F , we know that ν_1^F , and ν_2^F are λ_ϕ propositions ϕ_1^F and ϕ_2^F respectively. From the definition, $\gamma_\sqcup(\phi_1^F, \odot, \phi_2^F) = \phi^F \odot \phi^F$,

where $\odot \in \{\vee, \Rightarrow, \Leftrightarrow\}$. Proof follows from the type rules of λ_ϕ for propositions.

- Case $T^F \rightarrow \tau_1^F$: By inversion on type derivations of ν_1^F and ν_2^F , we know that

$$\nu_1^F = \lambda(k : T^F). e_1, \text{ and } \nu_2^F = \lambda(k : T^F). e_2,$$

for some k, e_1, e_2 , such that

$$\Gamma^F, k : T^F \vdash e_1 : \tau_1^F \text{ and } \Gamma^F, k : T^F \vdash e_2 : \tau_1^F.$$

Now, by structural induction on ν_1^F and ν_2^F , and eliminating inconsistent cases, we have the following inductive hypothesis (IH):

$$\gamma_\sqcup(e_1, \odot, e_2) \text{ is a value } \nu_k^F, \text{ and } \Gamma^F, k : T^F \vdash \nu_k^F : \tau_1^F$$

From first conjunct of IH: $\gamma_\sqcup(\nu_1^F, \odot, \nu_2^F) = \lambda(k : T^F). \nu_k^F$ is a value.

From the second conjunct of IH: Observing that this is the premise of type rules for functions in λ_ϕ , we conclude that:

$$\Gamma^F \vdash \lambda(k : T^F). \gamma_\sqcup(e_1, \odot, e_2) : T^F \rightarrow \tau_1^F$$

Definition (JoinType) *JoinType* of τ_1^F and τ_2^F is defined by structural recursion on the type structure:

- $\text{JoinType}(\text{bool}, \text{bool}) = \text{bool}$
- $\text{JoinType}(\text{bool}, T^F \rightarrow \tau^F) = T^F \rightarrow \text{JoinType}(\text{bool}, \tau^F)$
- $\text{JoinType}(T^F \rightarrow \tau_1^F, \tau_2^F) = T^F \rightarrow \text{JoinType}(\tau_1^F, \tau_2^F)$

LEMMA 1.2. *If $\Gamma^F \vdash \nu_1^F : \tau_1^F$ and $\Gamma^F \vdash \nu_2^F : \tau_2^F$, then $\gamma_{\bowtie}(\nu_1^F, \wedge, \nu_2^F)$ is a λ_ϕ value ν^F , such that $\Gamma^F \vdash \nu^F : \tau^F$: $\text{JoinType}(\tau_1^F, \tau_2^F)$*

Proof By induction on the structure of ν_1^F and ν_2^F , followed by inversion on their typing derivations. ■

LEMMA 1.3. *if $\Gamma \vdash R :: \text{intlist} \rightarrow \{\theta\}$ then, $\Gamma^F \vdash \llbracket R \rrbracket_L : A_1 \rightarrow \llbracket \theta \rrbracket_L \rightarrow \text{bool}$*

Proof $\llbracket \text{intlist} \rightarrow \{\theta\} \rrbracket_L \stackrel{\text{def}}{=} \llbracket \text{intlist} \rrbracket_L \rightarrow \llbracket \theta \rrbracket_L \rightarrow \text{bool}$.

Since $\mathcal{F}(\text{intlist}) = A_1$, we have that:

$$\begin{aligned} \llbracket R \rrbracket_L &= \llbracket R :: \text{intlist} \rightarrow \{\theta\} \rrbracket_L \\ &= \eta_{\text{wrap}}(R, \llbracket \text{intlist} \rightarrow \{\theta\} \rrbracket_L) \\ &= \eta_{\text{wrap}}(R, A_1 \rightarrow \llbracket \{\theta\} \rrbracket_L \rightarrow \text{bool}). \end{aligned}$$

But, when $\Gamma^F \vdash \phi^F : \tau^F$, then $\Gamma^F \vdash \eta_{\text{wrap}}(\phi^F, \tau^F) : \tau^F$.

Therefore, $\Gamma^F \vdash \llbracket R \rrbracket_L : A_1 \rightarrow \llbracket \theta \rrbracket_L \rightarrow \text{bool}$ ■

LEMMA 1.4. *If $\Gamma \vdash r : \theta$, then $\Gamma^F \vdash \llbracket r \rrbracket_L : \llbracket \theta \rrbracket_L \rightarrow \text{bool}$*

Proof By induction on the sort derivation. Cases:

- $r = R(x)$, where

$$\Gamma \vdash R :: \text{intlist} \rightarrow \{\theta\} \quad \Gamma \vdash x : \text{intlist}.$$

From Lemma 1.3, we know that

Sort Checking Specification Language

$$\boxed{\Gamma \vdash r :: \{\theta\}}$$

S-REL

$$\frac{(R :: \tau_R) \in \Gamma}{\Gamma \vdash R :: \tau_R}$$

S-APP

$$\frac{\Gamma \vdash R :: T \rightarrow \{\theta\} \quad \Gamma \vdash x : T}{\Gamma \vdash R(x) :: \{\theta\}}$$

S-UNION

$$\frac{\Gamma \vdash r_1 :: \{\theta\} \quad \Gamma \vdash r_2 :: \{\theta\}}{\Gamma \vdash r_1 \cup r_2 :: \{\theta\}}$$

S-CROSS

$$\frac{\Gamma \vdash r_1 :: \{\theta_1\} \quad \Gamma \vdash r_2 :: \{\theta_2\}}{\Gamma \vdash r_1 \times r_2 :: \{\theta_1 * \theta_2\}}$$

Well-Formedness

$$\boxed{\Gamma \vdash r, \quad \Gamma \vdash \phi, \quad \Gamma \vdash \tau}$$

WF-RPRED

$$\frac{\odot \in \{=, \subset\} \quad \Gamma \vdash r_1 :: \theta \quad \Gamma \vdash r_2 :: \theta}{\Gamma \vdash r_1 \odot r_2}$$

WF-REF

$$\frac{\odot \in \{\wedge, \vee, \Rightarrow\} \quad \Gamma \vdash \phi_1 \quad \Gamma \vdash \phi_2}{\Gamma \vdash \phi_1 \odot \phi_2}$$

WF-BASE

$$\frac{\Gamma, \nu : T \vdash \phi}{\Gamma \vdash \{\nu : T \mid \phi\}}$$

WF-FUN

$$\frac{\Gamma \vdash \tau_1 \quad \Gamma, x : \tau_1 \vdash \tau_2}{\Gamma \vdash x : \tau_1 \rightarrow \tau_2}$$

Figure 2: Static semantics of λ_R specification language

Typed Calculus with Propositions (λ_ϕ)

$$\begin{array}{ll} x & \in \lambda_R \text{ variable} \\ j, k & \in \text{bound variable} \end{array} \quad \begin{array}{ll} R & \in \lambda_R \text{ relation} \end{array}$$

$$\begin{array}{ll} v & ::= x \mid k \mid j \mid R \\ \phi^F & ::= v \mid v = v \mid \phi^F \phi^F \mid \phi^F \Leftrightarrow \phi^F \mid v : \tau^F \\ & \mid \phi^F \Rightarrow \phi^F \mid \phi^F \vee \phi^F \mid \phi^F \wedge \phi^F \\ \nu^F & ::= \lambda(k : T^F). \nu^F \mid \phi^F \\ e & ::= \nu^F \mid e e \\ T^F & ::= A \mid \text{bool} \\ \tau^F & ::= \text{bool} \mid T^F \rightarrow \tau^F \end{array}$$

MSFOL

$$\begin{array}{ll} \phi^L & ::= \forall(k : T^F). \phi^L \mid \phi^F \mid \phi^L \wedge \phi^L \quad \text{proposition} \\ & \mid \phi^L \vee \phi^L \mid \phi^L \Rightarrow \phi^L \end{array}$$

Auxiliary Functions

$$\begin{array}{ll} \text{variable} & Eval \\ \text{atom} & \mathcal{F} \\ \text{value} & \eta_{wrap} \\ \text{expression} & \eta_{wrap}(\phi^F, \tau_1^F \rightarrow \tau_2^F) \\ \text{base type} & \eta_{wrap}(\phi^F, T^F) \\ \text{type} & \eta_{wrap}(\phi^F, T^F) \end{array} \quad \begin{array}{ll} : & e \rightarrow \nu^F \\ : & T \rightarrow A \\ : & \{\phi^F\} \times \{\tau^F\} \rightarrow \{\nu^F \mid \nu^F \equiv_\eta \phi^F\} \\ = & \lambda(k : \tau_1^F). \eta_{wrap}(\phi^F k, \tau_2^F) \\ = & \phi^F \end{array}$$

Semantics of Relational Expressions

$$\boxed{\llbracket r \rrbracket_L}$$

$$\begin{array}{ll} \llbracket T \rrbracket_L & \stackrel{def}{=} \mathcal{F}(T) \\ \llbracket T * \theta \rrbracket_L & \stackrel{def}{=} \llbracket T \rrbracket_L \rightarrow \llbracket \theta \rrbracket_L \\ \llbracket T \rightarrow \{\theta\} \rrbracket_L & \stackrel{def}{=} \llbracket T \rrbracket_L \rightarrow \llbracket \theta \rrbracket_L \rightarrow \text{bool} \\ \llbracket RId \rrbracket_L & \stackrel{def}{=} \lambda(j : \llbracket \text{int} \rrbracket_L). \lambda(k : \llbracket \text{int} \rrbracket_L). j = k \\ \llbracket R :: \tau_R \rrbracket_L & \stackrel{def}{=} \eta_{wrap}(R, \llbracket \tau_R \rrbracket_L) \end{array} \quad \begin{array}{ll} \llbracket R(x) \rrbracket_L & \stackrel{def}{=} Eval(\llbracket R \rrbracket_L x) \\ \llbracket r_1 \cup r_2 \rrbracket_L & \stackrel{def}{=} \gamma_\cup(\llbracket r_1 \rrbracket_L, \vee, \llbracket r_2 \rrbracket_L) \\ \llbracket r_1 \times r_2 \rrbracket_L & \stackrel{def}{=} \gamma_\bowtie(\llbracket r_1 \rrbracket_L, \wedge, \llbracket r_2 \rrbracket_L) \\ \gamma_\cup(\lambda(k : T^F). e_1, \odot, \lambda(k : T^F). e_2) & \hookrightarrow \lambda(k : T^F). \gamma_\cup(e_1, \odot, e_2) \\ \gamma_\cup(\phi_1^F, \odot, \phi_2^F) & \hookrightarrow \phi_1^F \odot \phi_2^F \\ \gamma_\bowtie(\lambda(j : T_j^F). \phi_1^F, \odot, \lambda(k : T_k^F). \phi_1^F) & \hookrightarrow \lambda(j : T_j^F). \lambda(k : T_k^F). \phi_1^F \odot \phi_2^F \end{array}$$

Semantics of Type Refinements

$$\boxed{\llbracket \phi \rrbracket_L}$$

$$\begin{array}{ll} \llbracket r_1 = r_2 \rrbracket_L & \stackrel{def}{=} \gamma_\cup(\llbracket r_1 \rrbracket_L, \Leftrightarrow, \llbracket r_2 \rrbracket_L) \\ \llbracket r_1 \subset r_2 \rrbracket_L & \stackrel{def}{=} \gamma_\cup(\llbracket r_1 \rrbracket_L, \Rightarrow, \llbracket r_2 \rrbracket_L) \\ \llbracket \phi_1 \wedge \phi_2 \rrbracket_L & \stackrel{def}{=} \gamma_\bowtie(\llbracket \phi_1 \rrbracket_L, \wedge, \llbracket \phi_2 \rrbracket_L) \end{array} \quad \begin{array}{ll} \llbracket \phi_1 \vee \phi_2 \rrbracket_L & \stackrel{def}{=} \gamma_\cup(\llbracket \phi_1 \rrbracket_L, \vee, \llbracket \phi_2 \rrbracket_L) \\ \gamma_\forall(\lambda(k : T^F). \nu^F) & \hookrightarrow \forall(k : T^F). \gamma_\forall(\nu^F) \\ \gamma_\forall(\phi^F) & \hookrightarrow \phi^F \end{array}$$

Figure 3: Semantics of Specification Language

$$\Gamma^F \vdash \llbracket R \rrbracket_L : A_1 \rightarrow \llbracket \theta \rrbracket_L \rightarrow \text{bool} \quad \Gamma^F \vdash x : A_1$$

From function application type rule of λ_ϕ , we have:

$$\Gamma^F \vdash Eval(\llbracket R \rrbracket_L x) : \llbracket \theta \rrbracket_L \rightarrow \text{bool}$$

- $r = r_1 \cup r_2$, where

$$\Gamma \vdash r_1 :: \theta, \text{ and } \Gamma \vdash r_1 :: \theta$$

Inductive hypothesis (IH):

$$\Gamma^F \vdash \llbracket r_1 \rrbracket_L : \tau^F, \text{ and } \Gamma^F \vdash \llbracket r_2 \rrbracket_L : \tau^F.$$

Where, $\tau^F = \llbracket \theta \rrbracket_L \rightarrow \text{bool}$.

We know that $\llbracket r_1 \cup r_2 \rrbracket_L \stackrel{def}{=} \gamma_\cup(\llbracket r_1 \rrbracket_L, \vee, \llbracket r_2 \rrbracket_L)$. From lemma 1.1, proof for this case follows.

- $r = r_1 \times r_2$, where

$$\Gamma \vdash r_1 :: \theta_1, \quad \Gamma \vdash r_1 :: \theta_2, \text{ and } \theta = \theta_1 * \theta_2$$

Inductive hypothesis (IH):

$$\Gamma^F \vdash \llbracket r_1 \rrbracket_L : \tau_1^F, \text{ and } \Gamma^F \vdash \llbracket r_2 \rrbracket_L : \tau_2^F.$$

Where, $\tau_1^F = \llbracket \theta_1 \rrbracket_L \rightarrow \text{bool}$, and $\tau_2^F = \llbracket \theta_2 \rrbracket_L \rightarrow \text{bool}$.

We know that $\llbracket r_1 \times r_2 \rrbracket_L \stackrel{def}{=} \gamma_\bowtie(\llbracket r_1 \rrbracket_L, \wedge, \llbracket r_2 \rrbracket_L)$. From Lemma 1.2, we have:

$$\begin{array}{l} \Gamma^F \vdash \llbracket r \rrbracket_L : JoinType(\tau_1^F, \tau_2^F) \Rightarrow \\ \Gamma^F \vdash \llbracket r \rrbracket_L : \llbracket \theta_1 \rrbracket_L \rightarrow \llbracket \theta_2 \rrbracket_L \rightarrow \text{bool} \Rightarrow \end{array}$$

It remains to prove that $\llbracket \theta_1 * \theta_2 \rrbracket_L = \llbracket \theta_1 \rrbracket_L \rightarrow \llbracket \theta_2 \rrbracket_L$, which is trivial to establish by structural induction on θ_1 . ■

LEMMA 1.5. If $\Gamma \vdash \phi$, then $\llbracket \phi \rrbracket_L$ is an λ_ϕ value.

Calculus $\lambda_{\forall R}$

t	\in	$\text{tuple} - \text{sort variables}$	x, y, k	\in	variables
$'a, 'b$	\in	type variables			
ν	$::=$	$C \overline{T} \overline{\theta} \overline{R} \overline{\nu} \mid \Lambda a. e$			<i>value</i>
		$\mid \hat{\Lambda} t. \Lambda(R :: a : \rightarrow t). e \mid \dots$			
e	$::=$	$e T \mid e \theta \mathcal{R} \mid \hat{\Lambda} t. \Lambda(R :: a : \rightarrow t). e$			<i>expression</i>
		$\mid \text{match } x \text{ with } C \overline{T} \overline{\theta} \overline{R} \overline{x} \Rightarrow e$			
		$\mid \text{else } e \mid \dots$			
Tt	$::=$	$T \mid t$			
T	$::=$	$'a \mid 'a \text{ list} \mid \dots$			<i>datatypes</i>
τ	$::=$	$\{\nu : T \mid \Phi\} \mid \dots$			<i>dependent type</i>
δ	$::=$	$\forall t. \forall (R :: 'a : \rightarrow t). \delta \mid \tau$			<i>parametric type</i>
σ	$::=$	$\forall 'a. \sigma \mid \delta$			<i>type scheme</i>

Specification Language

Φ	$::=$	$\rho = \rho \mid \rho \subset \rho \mid \Phi \wedge \Phi$	<i>type refinement</i>
		$\mid \Phi \vee \Phi \mid \top$	
ρ	$::=$	$\mathcal{R}(x) \mid \rho \cup \rho \mid \rho \times \rho$	<i>rel. expression</i>
\mathcal{R}	$::=$	$\mathcal{R} T \mid \mathcal{R} \theta \mathcal{R} \mid R$	<i>instantiation</i>
θ	$::=$	$t \mid t * \theta \mid \dots$	<i>tuple sort</i>
τ_R	$::=$	$\forall t. ('a : \rightarrow t) : \rightarrow ('a \text{ list} : \rightarrow \theta)$	<i>relation sort</i>
		$\mid \dots$	
σ_R	$::=$	$\forall a. \tau_R \mid \tau_R$	<i>sort scheme</i>
Σ_R	$::=$	$\langle R, R_p, \sigma_R, \overline{C \overline{x} \Rightarrow r} \rangle$	<i>rel. definition</i>
		$\mid \langle R, R_p, \sigma_R, \mathcal{R}^* \rangle \mid \dots$	

Figure 4: $\lambda_{\forall R}$ - Complete calculus with parametric relations

r	$::=$	$R(x) \mid r \times r$	
F_R	$::=$	$\lambda(\overline{x : T}). r$	<i>transformer</i>
e_b	$::=$	$\text{bind}(R(x), F_R)$	<i>bind expression</i>
E_b	$::=$	$\lambda(\overline{x : T}). \text{bind}(R(x), F_R)$	<i>bind abstraction</i>
ψ	$::=$	$R = E_b$	<i>bind equation</i>
Σ_R^b	$::=$	$\lambda R. E_b$	<i>bind definition</i>

Figure 6: Bind Syntax

Proof By induction on the derivation of well-formedness judgement. Cases can be proved by straightforward application of Lemmas 1.4, 1.1 and 1.2.

LEMMA 1.6. *If $\Gamma \vdash \phi$, then $\gamma_{\forall}(\llbracket \phi \rrbracket_L)$ is an MSFOL formula.*

Proof From lemma 1.5, we know that $\llbracket \phi \rrbracket_L$ is an λ_{ϕ} value ν^F . Next, by structural induction on ν^F , we prove that $\gamma_{\forall}(\nu^F)$ is an MSFOL formula.

LEMMA 1.7. (Completeness of semantics) *For every type refinement ϕ , if $\Gamma \vdash \phi$, then $\text{compile}(\Gamma, \phi)$ terminates and produces an MSFOL formula.*

Proof From lemma 1.6

THEOREM 1.8. (**Decidability**) *Type checking in λ_R is decidable.*

Proof Follows from Lemma 1.7 and decidability proof of EPR logic. ■

2. Parametric Language Meta-theory

2.1 Decidability of Type Checking

Type refinements (Φ) in $\lambda_{\forall R}$ can be elaborated to a conjunction of bind equations representing semantics of instantiated relations, and

a λ_R type refinement (ϕ). The relevant rewrite rules are shown in figure 5.

THEOREM 2.1. (**Decidability**) *Type checking in $\lambda_{\forall R}$ is decidable.*

Proof Follows from 1. completeness of our rewrite rules for well-formed refinements, 2. Completeness of encoding bind equations in MSFOL, 3. Decidability proof of EPR logic, to which bind equations are compiled to, and decidability result (Theorem 1.8) for λ_R . ■

Kind Rules $\boxed{\Gamma \vdash r :: \delta_R}$

K-PARAM-REL

$$\frac{\Sigma_R^\pi(R) = E_R^\pi \quad \Gamma \vdash E_R^\pi :: \delta_R}{\Gamma \vdash R :: \delta_R}$$

K-BIND

$$\frac{\overline{T_2} = \theta_1 \quad \Gamma, x : T_1 \vdash R(x) :: \{\theta_1\} \quad \Gamma, \overline{k : T_2} \vdash r :: \{\theta_2\}}{\Gamma \vdash \lambda(x : T_1). \mathbf{bind}(R_1(x), \lambda(\overline{k : T_2}). r) :: T_1 \rightarrow \theta_2}$$

K-INST

$$\frac{\Gamma \vdash \mathcal{R}_2 :: \tau_R \quad \Gamma \vdash \mathcal{R}_1 :: \tau_R \rightarrow \tau_R}{\Gamma \vdash \mathcal{R}_1 \mathcal{R}_2 :: \tau_R}$$

Rewrite Rules for Type Refinements

$\boxed{\Phi \hookrightarrow \psi \wedge \phi}$

$\psi ::= \text{true} \mid (R = E_b) \wedge \psi \quad \text{bind equations}$

RW-REF

$$\frac{\Phi_1 \hookrightarrow \psi_1 \wedge \phi_1 \quad \Phi_2 \hookrightarrow \psi_2 \wedge \phi_2 \quad \odot \in \{\wedge, \vee\}}{\Phi_1 \odot \Phi_2 \hookrightarrow (\psi_1 \wedge \psi_2) \wedge (\phi_1 \odot \phi_2)}$$

RW-REF-ATOM

$$\frac{\rho_1 \hookrightarrow \psi_1 \wedge r_1 \quad \rho_2 \hookrightarrow \psi_2 \wedge r_2 \quad \odot \in \{=, \subset\}}{\rho_1 \odot \rho_2 \hookrightarrow (\psi_1 \wedge \psi_2) \wedge (r_1 \odot r_2)}$$

RW-PARAM-REL

$$\frac{\Sigma_R^\pi(R) = E_R^\pi}{R \hookrightarrow E_R^\pi}$$

RW-KINST

$$\frac{\mathcal{R} \hookrightarrow \Lambda t. E_R^\pi}{\mathcal{R} \theta \hookrightarrow [\theta/t] E_R^\pi}$$

RW-RINST

$$\frac{\mathcal{R}_1 \hookrightarrow \lambda(R_1 :: \tau_R). E_R^\pi \quad \mathcal{R}_2 \hookrightarrow R_2}{\mathcal{R}_1 \mathcal{R}_2 \hookrightarrow [R_2/R_1] E_R^\pi}$$

RW-RAPP

$$\frac{\mathcal{R} \hookrightarrow E_b \quad \text{freshVar}(R)}{E_b(x) \hookrightarrow (R = E_b) \wedge R(x)}$$

Semantics of Bind Equations

$\boxed{\llbracket \psi \rrbracket_L}$

$$\begin{aligned} \llbracket R_2 = \lambda(x : T_1). \mathbf{bind}(R_1(x), \lambda(\overline{k : T_2}). r) \rrbracket_L &\stackrel{def}{=} \forall(x : \llbracket T_1 \rrbracket_L). \gamma \Rightarrow (\llbracket R_1(x) \rrbracket_L, \forall(\overline{(k : \llbracket T_2 \rrbracket_L)} \llbracket r \rrbracket_L, \llbracket R_2(x) \rrbracket_L) \\ &\quad \wedge \forall(x : \llbracket T_1 \rrbracket_L). \gamma \Leftarrow (\llbracket R_1(x) \rrbracket_L, \forall(\overline{(k : \llbracket T_2 \rrbracket_L)} \llbracket r \rrbracket_L, \llbracket R_2(x) \rrbracket_L) \\ \gamma \Rightarrow (\forall(\overline{(k : T_1^F)}). \phi_1^F, \forall(\overline{(k : T_1^F)}). \forall(\overline{(j : T_2^F)}). \phi_2^F, \nu^F) &\hookrightarrow \forall(\overline{(k : T_1^F)}). \forall(\overline{(j : T_2^F)}). \phi_1^F \wedge \phi_2^F \Rightarrow \nu^F \bar{j} \\ \gamma \Leftarrow (\forall(\overline{(k : T_1^F)}). \phi_1^F, \forall(\overline{(k : T_1^F)}). \forall(\overline{(j : T_2^F)}). \phi_2^F, \nu^F) &\hookrightarrow \forall(\overline{(j : T_2^F)}). \exists(\overline{(k : T_1^F)}). \nu^F \bar{j} \Rightarrow \phi_1^F \wedge \phi_2^F \end{aligned}$$

Figure 5: Semantics of bind equations for parametric relations in $\lambda_{\forall R}$