Generic Refinement Types - Technical Appendix

NICO LEHMANN, University of California at San Diego, USA
COLE KURASHIGE, University of California at San Diego, USA
NIKHIL AKITI, University of California at San Diego, USA
NIROOP KRISHNAKUMAR, University of California at San Diego, USA
RANJIT JHALA, University of California at San Diego, USA

Contents

1 INTRODUCTION

This document is the technical appendix accompanying the paper Generic Refinement Types. It presents the formalization of a core calculus λ_G, an extension of the simply typed λ-calculus with generic refinements. We define the semantics of λ_G via a translation into the polymorphic contract calculus F_H^σ [2]. To facilitate the translation we define an intermediate caculus $\lambda_{\underline{G}}$ with explicit refinement instantiation.

2 SYNTAX

This section shows the syntax of refinements (Section 2.1) and types (Section 2.2) which are shared by λ_G and $\lambda_{\underline{G}}$. Section 2.3 and Section 2.4 show the expressions in λ_G and $\lambda_{\underline{G}}$ respectively.

2.1 Refinements

The language of refinements is built from an underlying logic of linear arithmetic and uninterpreted functions. Refinement contains λ-abstractions which normally fall outside the efficiently SMT decidable fragment. Our algorithmic system ensure λ-abstractions are eliminated before generating constraints.

$$
\begin{array}{rrcll}
\textbf{\textit{Base}} & \sigma_b & := & \mathbb{Z} & \textit{int} \\
 & & \mid & \mathbb{B} & \textit{bool} \\
\textbf{\textit{Sort}} & \sigma & ::= & \sigma_b & \textit{base} \\
 & & \mid & \sigma_b \to \sigma_b & \textit{func.} \\
\textbf{\textit{Reft.}} & r & ::= & z \mid \mathrm{tt} \mid \mathrm{ff} \mid a & \textit{atom} \\
 & & \mid & r \wedge r \mid r \vee r \mid \neg r & \textit{bool} \\
 & & \mid & r \bowtie r \mid r \oplus r & \textit{arith} \\
 & & \mid & \lambda a : \sigma.\, r \mid r\, r & \textit{func.} \\
\textbf{\textit{RCtx.}} & \Theta & := & \cdot \mid \Theta, a :_\mu \sigma \mid \Theta, r &
\end{array}
$$

2.2 Types

The following grammar shows the syntax of types and schemes. Generic refinement abstraction is represented with the scheme $\forall a :_\mu \sigma.\, \eta$, which bind the refinement variables a in any of the two *inference modes* **hdl** or **hrn**.

Though the syntax does not permit the expression $b[r]$, which means indexing b with r, we write it as an alias for $\{b[r] \mid \mathrm{tt}\}$. And in general if a type is refined by tt, we treat it (syntactically or in prose) as if it were not refined.

$$
\begin{array}{llcll}
\textbf{\textit{Type}} & \tau & ::= & \{b[r] \mid r\} & \textit{refinement} \\
& & \mid & \{a.\ b[a] \mid r\} & \textit{existential} \\
& & \mid & \tau \rightarrow \tau & \textit{function} \\
\textbf{\textit{Mode}} & \mu & := & \textbf{hdl} & \textit{hindley} \\
& & \mid & \textbf{hrn} & \textit{horn} \\
\textbf{\textit{Scheme}} & \eta & := & \tau & \textit{type} \\
& & \mid & \forall a :_{\mu} \sigma.\ \eta & \textit{generalization}
\end{array}
$$

2.3 λ_G Expressions

Expressions in λ_G are mostly standard. We highlight the expression $\Lambda a :_{\mu} \sigma.\ e$ used to generalize over a refinement.

$$
\begin{array}{llcll}
\textbf{\textit{Expr.}} & e & ::= & x & \textit{variable} \\
& & \mid & c & \textit{constant} \\
& & \mid & \mathrm{op}(e_1, \dots, e_n) & \textit{operation} \\
& & \mid & \lambda x.\ e & \textit{lambda} \\
& & \mid & e(e_1, \dots, e_n) & \textit{application} \\
& & \mid & e : \tau & \textit{ascription} \\
\textbf{\textit{PCtx.}} & \Gamma & := & \cdot \mid \Gamma, x : \tau &
\end{array}
$$

2.4 $\lambda_{\underline{G}}$ Expressions

Expressions $\underline{e}$ in $\lambda_{\underline{G}}$ mostly mirror the syntax of λ_G, but they have an explicit list of refinement instantiations for applications.

$$
\begin{array}{llcll}
\textbf{\textit{Expressions}} & \underline{e} & ::= & x & \textit{var} \\
& & \mid & c & \textit{constant} \\
& & \mid & \mathrm{op}\langle\overline{r}\rangle(\overline{\underline{e}}) & \textit{operation} \\
& & \mid & \lambda x.\ \underline{e} & \textit{lambda} \\
& & \mid & \underline{e}\langle\overline{r}\rangle(\overline{\underline{e}}) & \textit{application} \\
& & \mid & \underline{e} : \tau & \textit{ascription}
\end{array}
$$

3 ALGORITHMIC SYSTEM

We definen an algorithmic system for λ_G that synthesizes refinement instantiations at application sites. The following grammar shows the extensions to the syntax required for the system. Most notably, refinements are extended with two types of *inference variables*: evars $\hat{a}$ are used to infer generic refinements in mode **hdl** and Horn variables κ for generic refinements in mode **hrn**. Additionally, we define Horn constraints Ψ.

The algorithmic typing judgment outputs a constraint Ψ whose solution yields a valid instantiation of the constraint generic refinement parameters.

$$
\begin{array}{rcll}
\textbf{\textit{Refinements}} & r & ::= & \cdots \mid \hat{a} \mid \kappa(r_1, \ldots, r_n) \\
\textbf{\textit{SMT Terms}} & t & ::= & z \mid \mathrm{tt} \mid \mathrm{ff} \mid a \mid (t_1, \ldots, t_n) \mid \pi_i\, t \\
 & & \mid & t \wedge t \mid t \vee t \mid \neg t \mid t \bowtie t \mid t \oplus t \mid t\; t \\
\textbf{\textit{Predicates}} & p & := & t \mid \kappa(t_1, \ldots, t_n) \mid p \wedge p \\
\textbf{\textit{Constraints}} & \Psi & := & p \mid \Psi \wedge \Psi \mid \forall a : \sigma.\, \Psi \mid \Psi \Rightarrow \Psi \\
\textbf{\textit{Evar context}} & \Delta & ::= & \cdot \mid \Delta, \hat{a} : \sigma \mid \Delta, \hat{a} : \sigma = r
\end{array}
$$

3.1 Well-formedness

The various well-formedness judgments ensure among other things that (1) **hdl**-generic refinements are in position that can be solveds syntactically during subtyping and (2) **hrn**-generic refinements only appear applied in a top-level conjunction guaranteeing they can be instantiated by the Horn solver. In the following, the typing and subtyping judgments presupose well-formedness.

Well-sorted Refinements $\boxed{\Theta \vdash_\ell r : \sigma}$

$$
\textsc{WS-hrn}\ \frac{\Theta \vdash_\bot r : \sigma_b \quad \Theta(a) = (\mathbf{hrn}, \sigma_b \to \mathbb{B})}{\Theta \vdash_\top a\, r : \mathbb{B}}
\qquad
\textsc{WS-hdl}\ \frac{\Theta(a) = (\mathbf{hdl}, \sigma)}{\Theta \vdash_\ell a : \sigma}
\qquad
\textsc{WS-Int}\ \frac{}{\Theta \vdash_\ell z : \mathbb{Z}}
\qquad
\textsc{WS-tt}\ \frac{}{\Theta \vdash_\ell \mathrm{tt} : \mathbb{B}}
\qquad
\textsc{WS-ff}\ \frac{}{\Theta \vdash_\ell \mathrm{ff} : \mathbb{B}}
$$

$$
\textsc{WS-And}\ \frac{\Theta \vdash_\ell r_1 : \mathbb{B} \quad \Theta \vdash_\ell r_2 : \mathbb{B}}{\Theta \vdash_\ell r_1 \wedge r_2 : \mathbb{B}}
\qquad
\textsc{WS-Or}\ \frac{\Theta \vdash_\bot r_1 : \mathbb{B} \quad \Theta \vdash_\bot r_2 : \mathbb{B}}{\Theta \vdash_\ell r_1 \vee r_2 : \mathbb{B}}
\qquad
\textsc{WS-Not}\ \frac{\Theta \vdash_\bot r : \mathbb{B}}{\Theta \vdash_\ell \neg r : \mathbb{B}}
\qquad
\textsc{WS-Rel}\ \frac{\Theta \vdash_\bot r_1 : \sigma_b \quad \Theta \vdash_\bot r_2 : \sigma_b}{\Theta \vdash_\ell r_1 \bowtie r_2 : \mathbb{B}}
$$

$$
\textsc{WS-Op}\ \frac{\Theta \vdash_\bot r_1 : \mathbb{Z} \quad \Theta \vdash_\bot r_2 : \mathbb{Z}}{\Theta \vdash_\ell r_1 \oplus r_2 : \mathbb{Z}}
\qquad
\textsc{WS-App}\ \frac{\Theta \vdash_\bot r_1 : \sigma_b \to \sigma'_b \quad \Theta \vdash_\bot r_2 : \sigma_b}{\Theta \vdash_\ell r_1\, r_2 : \sigma'_b}
$$

Well-sorted refinements $\boxed{\Theta \vdash_\ell r : \sigma}$

$$
\textsc{Wf-hrn}\ \frac{\Theta \vdash_\bot r \quad \Theta(a) = (\mathbf{hrn}, \sigma)}{\Theta \vdash_\top a\, r}
\qquad
\textsc{Wf-hdl}\ \frac{\Theta(a) = (\mathbf{hdl}, \sigma)}{\Theta \vdash_\ell a}
\qquad
\textsc{WS-Int}\ \frac{}{\Theta \vdash_\ell z}
\qquad
\textsc{WS-tt}\ \frac{}{\Theta \vdash_\ell \mathrm{tt}}
\qquad
\textsc{WS-ff}\ \frac{}{\Theta \vdash_\ell \mathrm{ff}}
\qquad
\textsc{WS-And}\ \frac{\Theta \vdash_\ell r_1 \quad \Theta \vdash_\ell r_2}{\Theta \vdash_\ell r_1 \wedge r_2}
$$

$$
\textsc{WS-Or}\ \frac{\Theta \vdash_\bot r_1 \quad \Theta \vdash_\bot r_2}{\Theta \vdash_\ell r_1 \vee r_2}
\qquad
\textsc{WS-Not}\ \frac{\Theta \vdash_\bot r}{\Theta \vdash_\ell \neg r}
\qquad
\textsc{WS-Rel}\ \frac{\Theta \vdash_\bot r_1 \quad \Theta \vdash_\bot r_2}{\Theta \vdash_\ell r_1 \bowtie r_2}
\qquad
\textsc{WS-Op}\ \frac{\Theta \vdash_\bot r_1 \quad \Theta \vdash_\bot r_2}{\Theta \vdash_\ell r_1 \oplus r_2}
\qquad
\textsc{WS-App}\ \frac{\Theta \vdash_\bot r_1 \quad \Theta \vdash_\bot r_2}{\Theta \vdash_\ell r_1\, r_2}
$$

Well-formed refinement context $\boxed{\vdash \Theta \text{ ctx}}$

$$\vdash \cdot \text{ ctx} \qquad \frac{\vdash \Theta \text{ ctx} \qquad a \notin dom(\Theta)}{\vdash \Theta, a :_{\mathbf{hdl}} \sigma \text{ ctx}} \qquad \frac{\vdash \Theta \text{ ctx} \qquad a \notin dom(\Theta)}{\vdash \Theta, a :_{\mathbf{hrn}} \sigma_b \to \mathbb{B} \text{ ctx}} \qquad \frac{\vdash \Theta \text{ ctx} \qquad \Theta \vdash_{\top} r : \mathbb{B}}{\vdash \Theta, r \text{ ctx}}$$

Well-formed Generic Application $\boxed{\Theta \vdash_{\pm} b[r]\ [\Xi]}$

$$\text{WF-VAR-}\ \frac{\Theta(a) = (b_\mu, b_\sigma)}{\Theta \vdash_{-} b[a]\ [a]} \qquad \text{WF-VAR+}\ \frac{\Theta(a) = (b_\mu, b_\sigma)}{\Theta \vdash_{+} b[a]\ [\cdot]} \qquad \text{WF-LAM}\ \frac{b_\sigma = \sigma_b \to \sigma_b' \qquad \Theta, a : \sigma_b \vdash_{\top} r : \sigma_b'}{\Theta \vdash_{\pm} b[\lambda a : \sigma_b.\, r]\ [\cdot]} \qquad \text{WF-OTHER}\ \frac{r \neq a \wedge r \neq \lambda a : \sigma.\, r' \qquad \Theta \vdash_{\top} r : b_\sigma}{\Theta \vdash_{\pm} b[r]\ [\cdot]}$$

Well-formed Types $\boxed{\Theta \vdash_{\pm} \tau \text{ type } [\Xi]}$

$$\text{WF-REFT}\ \frac{\Theta \vdash_{\pm} b[r_1]\ [\Xi] \qquad \Theta \vdash_{\top} r_2 : \mathbb{B}}{\Theta \vdash_{\pm} \{b[r_1] \mid r_2\} \text{ type } [\Xi]} \qquad \text{WF-EXISTS}\ \frac{\Theta, a :_{\mathbf{hdl}} b_\sigma \vdash_{\top} r : \mathbb{B}}{\Theta \vdash_{\pm} \{a.\, b[a] \mid r\} \text{ type } [\cdot]} \qquad \text{WF-FUN}\ \frac{\Theta \vdash_{\neg p} \tau_1 \text{ type } [\Xi_1] \qquad \Theta \vdash_{p} \tau_2 \text{ type } [\Xi_2]}{\Theta \vdash_{p} \tau_1 \to \tau_2 \text{ type } [\Xi_1 \cup \Xi_2]}$$

Well-formed Schemes $\boxed{\Theta \vdash \eta \text{ sch } [\Xi]}$

$$\text{WF-TY}\ \frac{\Theta \vdash_{+} \tau \text{ type } [\Xi]}{\Theta \vdash \tau \text{ sch } [\Xi]} \qquad \text{WF-}\mathbf{HRN}\ \frac{\Theta, a :_{\mathbf{hrn}} \sigma \vdash \eta \text{ sch } [\Xi]}{\Theta \vdash \forall a :_{\mathbf{hrn}} \sigma.\, \eta \text{ sch } [\Xi]} \qquad \text{WF-}\mathbf{HDL}\ \frac{\Theta, a :_{\mathbf{hdl}} \sigma \vdash \eta \text{ sch } [\Xi] \qquad a \in \Xi}{\Theta \vdash \forall a :_{\mathbf{hdl}} \sigma.\, \eta \text{ sch } [\Xi]}$$

Well-formed program context $\boxed{\vdash \Gamma \text{ ctx}}$

$$\text{PrgCxEmpty}\ \frac{}{\Theta \vdash \cdot \text{ ctx}} \qquad \text{PrgCxVar}\ \frac{\Theta \vdash \Gamma \text{ ctx} \qquad \Theta \vdash_{+} \tau \text{ type } [\Xi]}{\Theta \vdash \Gamma, x : \tau \text{ ctx}}$$

Well-formed evar context $\boxed{\vdash \Gamma \text{ ctx}}$

$$\text{EvarCxEmpty}\ \frac{}{\Theta \vdash \cdot \text{ ctx}} \qquad \text{EvarCxUnsolved}\ \frac{\Theta \vdash \Delta \text{ ctx} \qquad \Delta \notin dom(\Delta)}{\Theta \vdash \Delta, \hat{a} : \sigma \text{ ctx}} \qquad \text{EvarCxSolved}\ \frac{\Theta \vdash \Delta \text{ ctx} \qquad \Delta \notin dom(\Delta) \qquad \Theta \vdash_{\top} r : \sigma}{\Theta \vdash \Delta, \hat{a} : \sigma{=}r \text{ ctx}}$$

3.2 Subtyping

During subtyping we solve evars syntactically. The result is a Horn constraint Ψ that may contain unknown Horn variables. The subtyping judgment uses the auxiliary refteq to generate a Horn constraint that implies (extensional) equality between refinements by structurally destructuring them.

Subtyping $\boxed{\Theta; \Delta_1 \vdash \tau_1 <: \tau_2 \dashv \Delta_2; \Psi}$

$$\textsc{<:Eq}\quad \frac{\Theta; \Delta \vdash r_1 \equiv [\Delta] r_2 : b_\sigma \dashv \Delta'; \Psi}{\Theta; \Delta \vdash \{b[r_1] \mid \mathrm{tt}\} <: \{b[r_2] \mid \mathrm{tt}\} \dashv \Delta'; \Psi}$$

$$\textsc{<:Fun}\quad \frac{\Theta; \Delta \vdash \tau_1' <: \tau_1 \dashv \Delta'; \Psi_1 \qquad \Theta; \Delta' \vdash \tau_2 <: \tau_2' \dashv \Delta''; \Psi_2}{\Theta; \Delta \vdash \tau_1 \rightarrow \tau_2 <: \tau_1' \rightarrow \tau_2' \dashv \Delta''; \Psi_1 \wedge \Psi_2}$$

$$\textsc{<:Reft/L}\quad \frac{\Theta, r_2; \Delta \vdash \{b[r_1] \mid \mathrm{tt}\} <: \tau \dashv \Delta'; \Psi}{\Theta; \Delta \vdash \{b[r_1] \mid r_2\} <: \tau \dashv \Delta'; r_2 \Rightarrow \Psi}$$

$$\textsc{<:Reft/R}\quad \frac{\Theta; \Delta \vdash \tau <: \{b[r_1] \mid \mathrm{tt}\} \dashv \Delta'; \Psi}{\Theta; \Delta \vdash \tau <: \{b[r_1] \mid r_2\} \dashv \Delta'; [\Delta'] r_2 \wedge \Psi}$$

$$\textsc{<:Exists/L}\quad \frac{\Theta, a :_{\mathbf{hdl}} b_\sigma; \Delta \vdash \{b[a] \mid r\} <: \tau \dashv \Delta'; \Psi}{\Theta; \Delta \vdash \{a.\, b[a] \mid r\} <: \tau \dashv \Delta'; \forall a : b_\sigma.\, \Psi}$$

$$\textsc{<:Exists/R}\quad \frac{\Theta; \Delta, \hat{a} : b_\sigma \vdash \tau <: \{b[\hat{a}] \mid r[\hat{a}/a]\} \dashv \Delta, \hat{a} : b_\sigma {=} r'; \Psi}{\Theta; \Delta \vdash \tau <: \{a.\, b[a] \mid r\} \dashv \Delta; \Psi}$$

Refinement Equivalence $\boxed{\Theta; \Delta \vdash r_1 \equiv r_2 : \sigma \dashv \Delta'; \Psi}$

$$\equiv\textsc{Inst}\quad \frac{r \text{ ground} \qquad \Delta \triangleq \Delta_1, \hat{a} : \sigma', \Delta_2}{\Theta; \Delta \vdash r \equiv \hat{a} : \sigma \dashv \Delta_1, \hat{a} : \sigma' {=} r, \Delta_2; \mathrm{tt}}$$

$$\equiv\textsc{Base}\quad \frac{\forall \hat{a} \in dom(\Delta).\ r_2 \neq \hat{a}}{\Theta; \Delta \vdash r_1 \equiv r_2 : \sigma_b \dashv \Delta; r_1 = r_2}$$

$$\equiv\kappa\textsc{/L}\quad \frac{\Psi \triangleq \kappa(\overline{r}) \Rightarrow r' \wedge r' \Rightarrow \kappa(\overline{r})}{\Theta; \Delta \vdash \kappa(\overline{r}) \equiv r' : \mathbb{B} \dashv \Delta; \Psi}$$

$$\equiv\kappa\textsc{/R}\quad \frac{\Psi \triangleq \kappa(\overline{r}) \Rightarrow r' \wedge r' \Rightarrow \kappa(\overline{r})}{\Theta; \Delta \vdash r' \equiv \kappa(\overline{r}) : \mathbb{B} \dashv \Delta; \Psi}$$

$$\equiv\textsc{Fun}\quad \frac{\Theta, a : \sigma_b; \Delta \vdash r_1\ a \equiv r_2\ a : \sigma_b' \dashv \Delta'; \Psi}{\Theta; \Delta \vdash r_1 \equiv r_2 : \sigma_b \rightarrow \sigma_b' \dashv \Delta; \forall a : \sigma_b.\, \Psi}$$

3.3 Typing

The type and elaboration judgment synthesizes refinement at applications and generates an expression in $\lambda_{\underline{G}}$. By the structure of the judgment, evars are always solved before elaboration, but the resulting expression $\underline{e}$ may contain Horn variables that must be instantiated by solving the output constraint Ψ. A Horn constraint solution Σ is a list of refinements to be substituted for Horn variables: $\Sigma ::= \cdot \mid \Sigma, r/\kappa$. The meta operation $[\Sigma]\underline{e}$ applies the substitution Σ to an expression $\underline{e}$. Using the algorithm in [1] we can write a procedure $\texttt{Solve} :: \Psi \rightarrow \Sigma \cup \bot$. Hence, we can get a fully elaborated expression by substituting Horn variables after solving the constraint.

Type Checking $\boxed{\Theta;\Gamma;\Delta \vdash e \Leftarrow \eta \dashv \Delta;\Psi \rightsquigarrow \underline{e}}$

$$\textsc{T-Forall}\quad \frac{\Theta, a:_\mu \sigma;\Gamma;\Delta \vdash e \Leftarrow \eta \dashv \Delta';\Psi \rightsquigarrow \underline{e}}{\Theta;\Gamma;\Delta \vdash e \Leftarrow \forall a:_\mu \sigma.\ \eta \dashv \Delta';\forall a:\sigma.\ \Psi \rightsquigarrow \underline{e}}$$

$$\textsc{T-Sub}\quad \frac{\Theta;\Gamma \vdash e \Rightarrow \tau_1 \dashv \Psi_1 \rightsquigarrow \underline{e} \qquad \Theta;\Delta \vdash \tau_1 <: \tau_2 \dashv \Delta';\Psi_2}{\Theta;\Gamma;\Delta \vdash e \Leftarrow \tau_2 \dashv \Delta';\Psi_1 \wedge \Psi_2 \rightsquigarrow \underline{e}}$$

$$\textsc{T-Abs}\quad \frac{\Theta;\Gamma, x:\tau_1;\Delta \vdash e \Leftarrow \tau_2 \dashv \Delta';\Psi \rightsquigarrow \underline{e}}{\Theta;\Gamma;\Delta \vdash \lambda x.\ e \Leftarrow \tau_1 \rightarrow \tau_2 \dashv \Delta';\Psi \rightsquigarrow \lambda x.\ \underline{e}}$$

Type Synthesis $\boxed{\Theta;\Gamma \vdash e \Rightarrow \eta \dashv \Psi \rightsquigarrow \underline{e}}$

$$\textsc{T-Var}\quad \frac{\Gamma(x) = \tau}{\Theta;\Gamma \vdash x \Rightarrow \tau \dashv \mathrm{tt} \rightsquigarrow x}$$

$$\textsc{T-Con}\quad \frac{}{\Theta;\Gamma \vdash c \Rightarrow \mathrm{ty}(c) \dashv \mathrm{tt} \rightsquigarrow c}$$

$$\textsc{T-Asc}\quad \frac{\Theta;\Gamma;\cdot \vdash e \Leftarrow \eta \dashv \cdot;\Psi \rightsquigarrow \underline{e}}{\Theta;\Gamma \vdash e:\eta \Rightarrow \eta \dashv \Psi \rightsquigarrow \underline{e}}$$

$$\textsc{T-App}\quad \frac{\Theta;\Gamma \vdash e \Rightarrow \eta \dashv \Psi_1 \rightsquigarrow \underline{e} \qquad \Theta;\Gamma;\cdot \vdash [\ \eta\ ](e'_1,\ldots,e'_n) \gg \tau \dashv \cdot;\Psi_2 \rightsquigarrow \langle\overline{r}\rangle(\overline{\underline{e}'})}{\Theta;\Gamma \vdash e(e'_1,\ldots,e'_n) \Rightarrow \tau \dashv \Psi_1 \wedge \Psi_2 \rightsquigarrow \underline{e}\langle\overline{r}\rangle(\overline{\underline{e}'})}$$

$$\textsc{T-Op}\quad \frac{\mathrm{scheme}(\mathrm{op}) = \eta \qquad \Theta;\Gamma;\cdot \vdash [\ \eta\ ](e_1,\ldots,e_n) \gg \tau \dashv \cdot;\Psi \rightsquigarrow \langle\overline{r}\rangle(\overline{\underline{e}})}{\Theta;\Gamma \vdash \mathrm{op}(e_1,\ldots,e_n) \Rightarrow \tau \dashv \Psi \rightsquigarrow \mathrm{op}\langle\overline{r}\rangle(\overline{\underline{e}})}$$

$$\begin{array}{lcll}
\mathrm{unpack}(\Theta, \{a.\ b[a] \mid r\}) & = & \Theta, a:b_\sigma; \{b[a'] \mid r\} & \text{fresh } a' \\
\mathrm{unpack}(\Theta, \tau) & = & \Theta; \tau &
\end{array}$$

Typing Applications $\boxed{\Theta; \Gamma; \Delta \vdash [\ \eta\ ](\overline{e}) \gg \tau \dashv \Delta'; \Psi \rightsquigarrow \langle \overline{r} \rangle (\underline{\overline{e}})}$

FA-hdl

$$\frac{\Theta; \Gamma; \Delta, \hat{a} : \sigma \vdash [\ \eta[\hat{a}/a]\ ](\overline{e}) \gg \tau \dashv \Delta', \hat{a} : \sigma = r; \Psi \rightsquigarrow \langle \overline{r'} \rangle (\underline{\overline{e}})}{\Theta; \Gamma; \Delta \vdash [\ \forall a :_{\mathbf{hdl}} \sigma.\ \eta\ ](\overline{e}) \gg \tau \dashv \Delta'; \Psi \rightsquigarrow \langle r, \overline{r'} \rangle (\underline{\overline{e}})}$$

FA-Res

$$\frac{}{\Theta; \Gamma; \Delta \vdash [\ \tau\ ](\cdot) \gg \tau \dashv \Delta; \mathrm{tt} \rightsquigarrow \langle \cdot \rangle (\cdot)}$$

FA-hrn

$$\frac{\kappa \text{ fresh} \quad \mathrm{vars}(\Theta) = \overline{a'} \quad r = \lambda a : \sigma.\ \kappa(a, \overline{a'}) \qquad \Theta; \Gamma; \Delta \vdash [\ \eta[r/a]\ ](\overline{e}) \gg \tau \dashv \Delta'; \Psi \rightsquigarrow \langle \overline{r'} \rangle (\underline{\overline{e}})}{\Theta; \Gamma; \Delta \vdash [\ \forall a :_{\mathbf{hrn}} \sigma.\ \eta\ ](\overline{e}) \gg \tau \dashv \Delta'; \Psi \rightsquigarrow \langle r, \overline{r'} \rangle (\underline{\overline{e}})}$$

FA-Fun

$$\frac{\Theta; \Gamma; \Delta \vdash e \Leftarrow \tau_1 \dashv \Delta'; \Psi_1 \rightsquigarrow \underline{e} \qquad \Theta; \Gamma; \Delta' \vdash [\ [\Delta']\tau_2\ ](\overline{e'}) \gg \tau' \dashv \Delta''; \Psi_2 \rightsquigarrow \langle \cdot \rangle (\underline{\overline{e'}})}{\Theta; \Gamma; \Delta \vdash [\ \tau_1 \to \tau_2\ ](e, \overline{e'}) \gg \tau' \dashv \Delta''; [\Delta'']\Psi_1 \wedge \Psi_2 \rightsquigarrow \langle \cdot \rangle (\underline{e}, \underline{\overline{e'}})}$$

$$\begin{array}{lcl} \mathrm{vars}(\Theta, a :_\mu \sigma) & = & \mathrm{vars}(\Theta), a \\ \mathrm{vars}(\Theta, r) & = & \mathrm{vars}(\Theta) \\ \mathrm{vars}(\cdot) & = & \cdot \end{array}$$

4 THE $\lambda_{\underline{G}}$ SYSTEM

The $\lambda_{\underline{G}}$ calculus is used as an intermediate step to translate λ_G programs into F_H^σ. We give semantics to refinements as expression in F_H^σ where we instantiate the set of base types to include sorts, and the set operations and constants to include those in the language of refinements. Importantly, we assume the existence of an equality operation on base sorts. Note that the equality is only on base sorts where it can be given operational meaning. Additionally, we also include the set of base values in λ_G as base types in F_H^σ. The language can be extended by adding primitive operations and constants to F_H^σ.

The basis of the translation is to interpret an indexed type $b[r]$ as a pair carrying a ghost value that must be (extensionally) equal to r as encoded by the translation $(\!|r|\!)_\sigma$.

$$\begin{array}{rcl} (\!|\forall x : \sigma.\ \eta|\!) & = & (x : \sigma) \to (\!|\eta|\!) \\ (\!|\{b[r_1] \mid r_2\}|\!) & = & b \times \{x : (\!|r|\!)_{b_\sigma} \mid r_2\} \\ (\!|\tau_1 \to \tau_2|\!) & = & (\!|\tau_1|\!) \to (\!|\tau_2|\!) \end{array} \qquad \begin{array}{rcl} (\!|\{a.\ b[a] \mid r\}|\!) & = & b \times \{a : b_\sigma \mid r\} \\ (\!|r|\!)_{\sigma_b \to \sigma_b'} & = & (x : \sigma_b) \to (\!|r\ x|\!)_{\sigma_b'} \\ (\!|r|\!)_{\sigma_b} & = & \{x : \sigma_b \mid x = r\} \end{array}$$

$$\begin{array}{rcl} (\!|\cdot|\!) & = & \cdot \\ (\!|\Theta, a : \sigma|\!) & = & (\!|\Theta|\!), a : \sigma \\ (\!|\Theta, r|\!) & = & (\!|\Theta|\!), \{x : \mathbb{Z} \mid r\} \end{array} \qquad \begin{array}{rcl} (\!|\cdot|\!) & = & \cdot \\ (\!|\Gamma, x : \tau|\!) & = & (\!|\Gamma|\!), x : (\!|\tau|\!) \end{array}$$

We further require that constants and operations in λ_G are typed in a way that translation preservers their types, i.e., we require

Requirement 1 (Typeability of constants and operations).

(1) *If* $ty(c) = \tau$ *then* $ty(c) = (\!|\tau|\!)$.
(2) *If* $scheme(op) = \eta$ *then* $ty(op) = (\!|\eta|\!)$.

4.1 Subtyping

Refinement equivalence $\boxed{\Theta \vdash r_1 \equiv r_2 : \sigma}$

$$\textsc{Dec}\equiv\textsc{base}\quad \frac{\forall \delta.\ (\!|\Theta|\!) \vdash \delta \implies \delta(r_1 = r_2) \longrightarrow^* \mathrm{tt}}{\Theta \vdash r_1 \equiv r_2 : \sigma_b}
\qquad
\textsc{Dec}\equiv\textsc{fun}\quad \frac{\Theta, a : \sigma_b \vdash (r_1\ a) \equiv (r_2\ a) : \sigma_b'}{\Theta \vdash r_1 \equiv r_2 : \sigma_b \to \sigma_b'}
\qquad
\textsc{Dec}\equiv\textsc{prod}\quad \frac{\forall i.\ \Theta \vdash \pi_i\ r_1 \equiv \pi_i\ r_2 : \sigma_i}{\Theta \vdash r_1 \equiv r_2 : \sigma_1 \times \cdots \times \sigma_n}$$

Subtyping $\boxed{\Theta \vdash \tau_1 <: \tau_2}$

$$\textsc{Dec<:eq}\quad \frac{\Theta \vdash r_1 \equiv r_2 : b_\sigma}{\Theta \vdash b[r_1] <: b[r_2]}
\qquad
\textsc{Dec<:constr/l}\quad \frac{\Theta, r_2 \vdash b[r_1] <: \tau}{\Theta \vdash \{b[r_1] \mid r_2\} <: \tau}
\qquad
\textsc{Dec<:constr/r}\quad \frac{\Theta \vdash \tau <: b[r_1] \qquad \Theta \vdash r_2\ \mathrm{true}}{\Theta \vdash \tau <: \{b[r_1] \mid r_2\}}
\qquad
\textsc{Dec<:exists/l}\quad \frac{\Theta, a : b_\sigma \vdash \{b[a] \mid r\} <: \tau}{\Theta \vdash \{a.\ b[a] \mid r\} <: \tau}$$

$$\textsc{Dec<:exists/r}\quad \frac{\Theta \vdash \tau <: \{b[r'] \mid r[r'/a]\}}{\Theta \vdash \tau <: \{a.\ b[a] \mid r\}}
\qquad
\textsc{Dec<:fun}\quad \frac{\Theta \vdash \tau_1' <: \tau_1 \qquad \Theta \vdash \tau_2 <: \tau_2'}{\Theta \vdash \tau_1 \to \tau_2 <: \tau_1' \to \tau_2'}$$

4.2 Typing

The typing rules of $\lambda_{\underline{G}}$ mostly follow that of λ_G, but output a translated F_H^σ expression M that has explicit casts since F_H^σ does not have subsumption.

Type Checking $\boxed{\Theta; \Gamma \vdash e \Leftarrow \eta \rightsquigarrow M}$

$$\textsc{Dec-Forall}\quad \frac{\Theta, a : \sigma; \Gamma \vdash e \Leftarrow \eta \rightsquigarrow M}{\Theta; \Gamma \vdash e \Leftarrow \forall a : \sigma.\ \eta \rightsquigarrow \lambda a : \sigma.\ M}
\qquad
\textsc{Dec-Sub}\quad \frac{\Theta; \Gamma \vdash e \Rightarrow \tau' \rightsquigarrow M \qquad \Theta \vdash \tau' <: \tau}{\Theta; \Gamma \vdash e \Leftarrow \tau \rightsquigarrow \langle (\!|\tau'|\!) \Rightarrow (\!|\tau|\!) \rangle\ M}$$

$$\textsc{Dec-Abs}\quad \frac{\Theta; \Gamma, x : \tau_1 \vdash e \Leftarrow \tau_2 \rightsquigarrow M}{\Theta; \Gamma \vdash \lambda x.\ e \Leftarrow \tau_1 \to \tau_2 \rightsquigarrow \lambda x : \tau_1.\ M}$$

Type Synthesis $\boxed{\Theta; \Gamma \vdash e : \eta \rightsquigarrow M}$

$$\textsc{Dec-Var}\ \frac{\Gamma(x) = \tau}{\Theta; \Gamma \vdash x \Rightarrow \tau \rightsquigarrow x} \qquad \textsc{Dec-Con}\ \frac{}{\Theta; \Gamma \vdash c \Rightarrow \mathrm{ty}(c) \rightsquigarrow c} \qquad \textsc{Dec-Asc}\ \frac{\Theta; \Gamma \vdash e \Leftarrow \eta \rightsquigarrow M}{\Theta; \Gamma \vdash e : \eta \Rightarrow \eta \rightsquigarrow M}$$

$$\textsc{Dec-App}\ \frac{\Theta; \Gamma \vdash e \Rightarrow \eta \rightsquigarrow M \quad \Theta; \Gamma \vdash [\ \eta\ ]\langle \overline{r} \rangle(e_1, \ldots, e_n) \gg \tau \rightsquigarrow \overline{M'}}{\Theta; \Gamma \vdash e\langle \overline{r} \rangle(e_1, \ldots, e_n) \Rightarrow \tau \rightsquigarrow M\ M'_1\ \ldots\ M'_n}$$

$$\textsc{Dec-Op}\ \frac{\eta = \mathrm{scheme}(\mathrm{op}) \quad \Theta; \Gamma \vdash [\ \eta\ ]\langle \overline{r} \rangle(e_1, \ldots, e_n) \gg \tau \rightsquigarrow \overline{M'}}{\Theta; \Gamma \vdash \mathrm{op}\langle \overline{r} \rangle(e_1, \ldots, e_n) \Rightarrow \tau \rightsquigarrow \mathrm{op}(M'_1, \ldots, M'_n)}$$

Well-typed application $\boxed{\Theta; \Gamma \vdash [\ \eta\ ]\langle \overline{r} \rangle(\overline{e}) \gg \tau \rightsquigarrow M\ [M\ ]}$

$$\textsc{Dec-FA-forall}\ \frac{\Theta \vdash_{\top} r' : \sigma \qquad \Theta; \Gamma \vdash [\ \eta[r'/a]\ ]\langle \overline{r} \rangle(\overline{e}) \gg \tau \rightsquigarrow \overline{M}}{\Theta; \Gamma \vdash [\ \forall a : \sigma.\ \eta\ ]\langle r', \overline{r} \rangle(\overline{e}) \gg \tau \rightsquigarrow r', \overline{M}} \qquad \textsc{Dec-FA-App}\ \frac{\Theta; \Gamma \vdash e' \Leftarrow \tau' \rightsquigarrow M' \quad \Theta; \Gamma \vdash [\ \tau_2\ ]\langle \cdot \rangle(\overline{e}) \gg \tau \rightsquigarrow \overline{M}}{\Theta; \Gamma \vdash [\ \tau_1 \to \tau_2\ ]\langle \cdot \rangle(e', \overline{e}) \gg \tau \rightsquigarrow M', \overline{M}}$$

$$\textsc{Dec-FA-Res}\ \frac{}{\Theta; \Gamma \vdash [\ \tau\ ]\langle \cdot \rangle(\cdot) \gg \tau \rightsquigarrow \cdot}$$

5 HORN CONSTRAINT SOLVING

$$\begin{array}{rcl} \mathtt{Solve} & :: & \Theta \times \Psi \to \Sigma \cup \bot \\ \mathtt{Solve}(\Theta, r; \Psi) & = & \mathtt{Solve}(\Theta; r \Rightarrow \Psi) \\ \mathtt{Solve}(\Theta, a : \sigma; \Psi) & = & \mathtt{Solve}(\Theta; \forall a : \sigma.\ \Psi) \\ \mathtt{Solve}(\cdot; \Psi) & = & \mathtt{Solve}(\Psi) \end{array}$$

Requirement 2 (Horn Solving Interface).

(1) *If* $\mathtt{Solve}(\Theta; \Psi) = \Sigma$ *then* $\mathtt{Solve}([\Sigma]\Theta; [\Sigma]\Psi) = \cdot$

(2) $\mathtt{Solve}(\Theta; \Psi_1 \wedge \Psi_2) = \mathtt{Solve}(\Theta; \Psi_2 \wedge \Psi_1)$

(3) *If* $\mathtt{kvars}(\Psi_1 \wedge \Psi_2) = \emptyset$ *and* $\mathtt{Solve}(\Psi_1 \wedge \Psi_2) = \cdot$ *then* $\mathtt{Solve}(\Psi_1) = \cdot$

(4) *If* $\mathtt{kvars}(\Theta) = \mathtt{kvars}(r) = \emptyset$ *and* $\mathtt{Solve}(\Theta; r) = \cdot$ *and* $(\!|\Theta|\!) \vdash \delta$ *then* $\delta(r) \longrightarrow^{*} tt$

6 PROPERTIES

Lemma 6.1 (Output Applied).

(1) *If* $\Theta; \Gamma; \Delta \vdash \boxed{e} \Leftarrow \eta \dashv \Delta'; \Psi \rightsquigarrow \underline{e}$ *then* $[\Delta']\Psi = \Psi$ *and* $[\Delta']\underline{e} = \underline{e}$ *and* $[\Delta']\eta = \eta$.

(2) *If* $\Theta; \Gamma; \Delta \vdash \boxed{[\,\eta\,](\overline{e})} \gg \tau \dashv \Delta'; \Psi \rightsquigarrow \langle\overline{r}\rangle(\overline{\underline{e}})$ *then* $[\Delta']\Psi = \Psi$ *and* $[\Delta']\overline{r} = \overline{r}$ *and* $[\Delta']\overline{\underline{e}} = \overline{\underline{e}}$ *and* $[\Delta']\tau = \tau$

(3) *If* $\Theta; \Delta \vdash \boxed{\tau_1 <: \tau_2} \dashv \Delta'; \Psi$ *then* $[\Delta']\Psi = \Psi$ *and* $[\Delta']\tau_2 = \tau_2$.

(4) *If* $\Theta; \Delta \vdash \boxed{r_1 \equiv r_2 : \sigma} \dashv \Delta'; \Psi$ *then* $[\Delta']\Psi = \Psi$ *and* $[\Delta']r_2 = r_2$

Proof. By induction on the given derivations applying the structural properties of algorithmic extension as appropiate. Parts (1) and (2) are mutually recursive. Part (4) depends on (3). □

Lemma 6.2.

(1) *If* $\Gamma(x) = \tau$ *then* $(\!|\Gamma|\!)(x) = (\!|\tau|\!)$

(2) *If* $\Theta(a) = \sigma$ *then* $(\!|\Theta|\!)(x) = \sigma$

Proof. By induction on the structure of the given contexts. □

Lemma 6.3. *If* $\Theta; \Gamma; \Delta \vdash \boxed{[\,\eta\,](\overline{e})} \gg \tau \dashv \Delta'; \Psi \rightsquigarrow \langle\overline{r}\rangle(\overline{\underline{e}})$ *then* $\eta = \forall a_1 :_{\mu_1} \sigma_1 \ldots a_n :_{\mu_n} \sigma_n.\ \tau_1 \to \cdots \to \tau_m \to \tau$

Proof. By induction on the derivation □

Lemma 6.4. *If* $\Theta \vdash \tau_1 <: \tau_2$ *then* $(\!|\tau_1|\!) \parallel (\!|\tau_2|\!)$

Proof. By induction on the subtyping derivation noting that each subtyping rule matches one of the rules of the type compability judgment in F_H^σ. □

6.1 Evar Context Extension

Evar Context Extension $\boxed{\Theta \vdash \Delta \longrightarrow \Delta'}$

$$\frac{}{\Theta \vdash \cdot \longrightarrow \cdot}\ \longrightarrow\textsc{Empty} \qquad \frac{\Theta \vdash \Delta \longrightarrow \Delta'}{\Theta \vdash \Delta, \hat{a} : \sigma \longrightarrow \Delta', \hat{a} : \sigma}\ \longrightarrow\textsc{Unsolved}$$

$$\frac{\Theta \vdash \Delta \longrightarrow \Delta'}{\Theta \vdash \Delta, \hat{a} : \sigma{=}r \longrightarrow \Delta', \hat{a} : \sigma{=}r}\ \longrightarrow\textsc{Solved} \qquad \frac{\Theta \vdash \Delta \longrightarrow \Delta'}{\Theta \vdash \Delta, \hat{a} : \sigma \longrightarrow \Delta', \hat{a} : \sigma{=}r}\ \longrightarrow\textsc{Solve}$$

Lemma 6.5 (Weaken extension). *If* $\Theta_1, \Theta_2 \vdash \Delta \longrightarrow \Delta'$ *then* $\Theta_1, \Theta_0, \Theta_2 \vdash \Delta \longrightarrow \Delta'$

Proof. By induction on the given context extension derivation. □

Lemma 6.6 (Ext. Reflexivity). *If* $\Theta \vdash \boxed{\Delta}$ *ctx then* $\Theta \vdash \Delta \longrightarrow \Delta$

Proof. Straightforward. □

Lemma 6.7 (Ext. Transitivity). *If* $\Theta \vdash \Delta_1 \longrightarrow \Delta_2$ *and* $\Theta \vdash \Delta_2 \longrightarrow \Delta_3$ *then* $\Theta \vdash \Delta_1 \longrightarrow \Delta_3$

Proof. Straightforward. □

Lemma 6.8 (Ext. Solve Entry). *If* $\Theta \vdash \boxed{\Delta_1, a : \sigma, \Delta_2}$ *ctx and* $\Theta \vdash_\sigma \boxed{\top} : r$ *then* $\Theta \vdash \Delta_1, \hat{a} : \sigma, \Delta_2 \longrightarrow \Delta_1, a : \sigma{=}r, \Delta_2$

Proof. By induction on the structore of Δ_2

- **Case** $\Delta_2 = \cdot$
  By Lemma 6.6 (Ext. Reflexivity) and then by $\longrightarrow$Unsolved.
- **Case** $\Delta_2 = \Delta_2', a' : \sigma$
  By context well-formedness $a' \neq a$. Then by i.h and $\longrightarrow$Solve.
- **Case** $\Delta_2 = \Delta_2', a' : \sigma{=}r''$
  Similar to the previous case.

□

6.2 Algorithmic Subtyping

Theorem 6.9 (Soundness of Alg. Subtyping). *If* $\Theta; \Delta \vdash \boxed{\tau_1 <: \tau_2} \dashv \Delta'; \Psi$ *and* $\texttt{Solve}(\Theta; \Psi \wedge \Psi') = \Sigma$ *and* $\Theta \vdash \Delta' \longrightarrow \Omega$ *and* $[\Delta]\tau_1 = \tau_1$ *then* $\Theta \vdash [\Sigma]\tau_1 <: [\Sigma][\Omega]\tau_2$

Proof. By induction on the algorithmic subtyping derivation.

Case <:eq

$[\Sigma]\Theta \vdash [\Sigma][\Omega]r_1 \equiv [\Sigma][\Omega]r_2 : \sigma$ by Lemma 6.10 (Sound. of Alg. Reft. Equiv)

$[\Sigma]\Theta \vdash [\Sigma][\Omega]\{b[r_1] \mid \mathrm{tt}\} <: [\Sigma][\Omega]\{b[r_2] \mid \mathrm{tt}\}$ conclude by Dec<:base

Case <:fun

$[\Sigma]\Theta \vdash [\Sigma]\tau_1' <: [\Sigma][\Omega]\tau_1'$ and $\Theta \vdash [\Sigma]\tau_2 <: [\Sigma][\Omega]\tau_2'$ by i.h.

$[\Sigma]\Theta \vdash [\Sigma](\tau_1 \to \tau_2) <: [\Sigma][\Omega](\tau_1' \to \tau_2')$ conclude by Dec<:fun and definition

Case <:constr/l

$[\Sigma](\Theta, r_2) \vdash [\Sigma]\{b[r_1] \mid \mathrm{tt}\} <: [\Sigma][\Omega]\tau$ by i.h., Lemma 6.5 (Weaken extension) and def. of `Solve`.

$[\Sigma]\Theta \vdash [\Sigma]\{b[r_1] \mid r_2\} <: [\Sigma][\Omega]\tau$ conclude by Dec<:constr/l

Case <:constr/r

$[\Sigma]\Theta \vdash [\Sigma]\tau <: [\Sigma][\Omega]\{b[r_1] \mid \mathrm{tt}\}$ by i.h.

$[\Sigma]\Theta \vdash [\Sigma]\tau <: [\Sigma][\Omega]\{b[r_1] \mid r_2\}$ conclude by Dec<:constr/r

Case <:exists/l

$[\Sigma]\Theta, a : b_\sigma \vdash [\Sigma]\{b[a] \mid r\} <: [\Sigma][\Omega]\tau$ by i.h., Lemma 6.5 (Weaken extension) and def. of `Solve`.

$[\Sigma]\Theta \vdash [\Sigma]\{a.\ b[a] \mid r\} <: [\Sigma][\Omega]\tau$ conclude by Dec<:exists/l

Case <:exists/r

$[\Sigma]\Theta \vdash [\Sigma]\tau <: [\Sigma][\Omega]\{b[r'] \mid r[r'/a]\}$ by i.h.

$[\Sigma]\Theta \vdash [\Sigma]\tau <: [\Sigma][\Omega]\{a.\ b[a] \mid r\}$ conclude by Dec<:exists/r

□

Lemma 6.10 (Sound. of Alg. Reft. Equiv.). *If* $\Theta; \Delta \vdash r_1 \equiv r_2 : \sigma \dashv \Delta'; \Psi$ *and* $\Theta \vdash \Delta' \longrightarrow \Omega$ *and* $\mathtt{Solve}(\Theta; \Psi \land \Psi') = \Sigma$ *and* $[\Delta]r_1 = r_1$ *then* $[\Sigma]\Theta \vdash [\Sigma][\Omega]r_1 \equiv [\Sigma][\Omega]r_2 : \sigma$

Proof. By induction on the algorithmic refinement equivalence derivation applying Requirement 2 (Horn Solving Interface), Lemma 6.7 (Ext. Transitivity), and Lemma 6.7 (Ext. Transitivity) as needed. □

6.3 Algorithmic Typing Soundness

Theorem 6.11 (Soundness of Algorithmic Typing).

(1) *If* $\Theta; \Gamma \vdash e \Rightarrow \eta \dashv \Psi \rightsquigarrow \underline{e}$ *and* $\mathtt{Solve}(\Theta; \Psi \land \Psi') = \Sigma$ *then* $[\Sigma]\Theta; [\Sigma]\Gamma \vdash [\Sigma][\Omega]\underline{e} \Rightarrow [\Sigma][\Omega]\eta$
(2) *If* $\Theta; \Gamma; \Delta \vdash e \Leftarrow \eta \dashv \Delta'; \Psi \rightsquigarrow \underline{e}$ *and* $\mathtt{Solve}(\Theta; \Psi \land \Psi') = \Sigma$ *and* $\Theta \vdash \Delta' \longrightarrow \Omega$ *then* $[\Sigma]\Theta; [\Sigma]\Gamma \vdash [\Sigma][\Omega]\underline{e} \Leftarrow [\Sigma][\Omega]\eta$
(3) *If* $\Theta; \Gamma; \Delta \vdash [\ \eta\ ](\overline{e}) \gg \tau \dashv \Delta'; \Psi \rightsquigarrow \langle \overline{r} \rangle(\overline{\underline{e}})$ *and* $\mathtt{Solve}(\Theta; \Psi \land \Psi') = \Sigma$ *and* $\Theta \vdash \Delta' \longrightarrow \Omega$ *and* $[\Delta]\eta = \eta$ *then* $[\Sigma]\Theta; [\Sigma]\Gamma \vdash [\ [\Sigma][\Omega]\eta\ ]\langle [\Sigma][\Omega]\overline{r} \rangle([\Sigma][\Omega]\overline{\underline{e}}) \gg [\Sigma][\Omega]\tau$

Proof.

(1) **Case T-var, T-con**
Straightforward

Case T-asc
by i.h. case (2)

Case T-app, T-op
by i.h. case (3)

(2) **Case T-forall**
by i.h. and Lemma 6.5 (Weaken extension)

Case T-sub
by i.h. and Theorem 6.9 (Soundness of Alg. Subtyping)

Case T-abs
by i.h. case (1)

(3) **Case FA-hdl**

Let $\Omega' = \Omega, \hat{a} : \sigma{=}r'$

$[\Sigma]\Theta; [\Sigma]\Gamma \vdash \boxed{[\ [\Sigma][\Omega']\eta[\hat{a}/a]\ ]\langle[\Sigma][\Omega']\overline{r}\rangle([\Sigma][\Omega']\underline{\overline{e}})} \gg [\Sigma][\Omega']\tau$ by i.h. and Lemma 6.8

$[\Sigma]\Theta; [\Sigma]\Gamma \vdash \boxed{[\ [\Sigma][\Omega]\eta[r'/a]\ ]\langle[\Sigma][\Omega]\overline{r}\rangle([\Sigma][\Omega]\underline{\overline{e}})} \gg [\Sigma][\Omega]\tau$ by Lemma 6.1

$[\Sigma]\Theta; [\Sigma]\Gamma \vdash \boxed{[\ [\Sigma][\Omega]\forall a : \sigma.\eta\ ]\langle[\Sigma][\Omega](r', \overline{r})\rangle([\Sigma][\Omega]\underline{\overline{e}})} \gg [\Sigma][\Omega]\tau$ by Dec-FA-forall

Case FA-hrn

$[\Sigma]\Theta; [\Sigma]\Gamma \vdash \boxed{[\ [\Sigma][\Omega]\eta[r/a]\ ]\langle[\Sigma][\Omega]\overline{r'}\rangle([\Sigma][\Omega]\underline{\overline{e}})} \gg [\Sigma][\Omega]\tau$ by i.h.

$[\Sigma]r = \lambda a : \sigma.p(a, \overline{a'})$ for some p.

Let $r'' = \lambda a : \sigma.p(a, \overline{a'})$

$[\Sigma]\Theta; [\Sigma]\Gamma \vdash \boxed{[\ [\Sigma][\Omega]\forall a : \sigma.\eta\ ]\langle[\Sigma][\Omega]\overline{r'}\rangle([\Sigma][\Omega]\underline{\overline{e}})} \gg [\Sigma][\Omega]\tau$ by Dec-FA-forall

Case FA-fun

By i.h. and Lemma 6.7

Case FA-res

Straightforward

□

6.4 Translation into F_H^σ

Theorem 6.12 (Type Preserving Translation).

(1) *If* $\Theta; \Gamma \vdash \boxed{\underline{e}} \Rightarrow \eta \rightsquigarrow M$ *then* $(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{M} : (\!|\eta|\!)$

(2) *If* $\Theta; \Gamma \vdash \boxed{\underline{e}} \Leftarrow \eta \rightsquigarrow M$ *then* $(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{M} : (\!|\eta|\!)$

(3) *If* $\Theta; \Gamma \vdash \boxed{[\ \eta\ ]\langle\overline{r}\rangle(\underline{\overline{e}})} \gg \tau \rightsquigarrow \overline{M}$ *and* $(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{M} : (\!|\eta|\!)$ *then* $(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{M\ \overline{M}} : (\!|\tau|\!)$

Theorem 6.13 (Translation preserves subtyping). *If* $\Theta \vdash \tau_1 <: \tau_2$ *then* $(\!|\Theta|\!) \vdash_H (\!|\tau_1|\!) <: (\!|\tau_2|\!)$.

Proof. By mutual induction on the structure of the derivations

(1) **Case Dec-Var**

By Lemma 6.2

Case Dec-Con

By T_Const

Case Dec-Asc
$(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{M} : (\!|\eta|\!)$ by i.h. case (2)

Case Dec-App
$(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{M} : (\!|\eta|\!)$ by i.h. case (1)
$(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{M\, M_1' \ldots M_n'} : (\!|\tau|\!)$ by i.h. case (3)

Case Dec-op
$\eta = \forall a_1 :_{\mu_1} \sigma_1 \ldots a_n :_{\mu_n} \sigma_n.\ \tau_1 \to \cdots \to \tau_m \to \tau$ by Lemma 6.3
let $M = \lambda a_1{:}\sigma_1, \ldots a_n{:}\sigma_n, x_1{:}\tau_1, \ldots, x_m{:}\tau_m.\ \mathrm{op}(\overline{a}, \overline{x})$
$\cdot \vdash_H \boxed{M} : (a{:}\sigma_1) \to \cdots \to (a{:}\sigma_n) \to (x_1{:}\tau_1) \to \cdots \to \tau_m \to \tau$ by T_Abs, T_Op and Requirement 1
$(\!|\Theta|\!)(\!|\Gamma|\!) \vdash_H \boxed{M} : (a{:}\sigma_1) \to \cdots \to (a{:}\sigma_n) \to (x_1{:}\tau_1) \to \cdots \to \tau_m \to \tau$ by Lemma 4.9 in [2]
$(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{M\, M_1' \ldots M_n'} : \tau$ by i.h. case (3)
$(\!|\Theta|\!), (\!|\Gamma|\!) \vdash_H \boxed{\mathrm{op}(M_1', \ldots, M_n')} : \tau$ by Lema 4.11 in [2]

(2) **Case Dec-Forall, Dec-Abs**
By i.h. and T_Abs

Case Dec-Sub
By Lemma 6.4 and T_Cast

(3) All cases follow by i.h. and the fact that refinement sort checking is preserved in F_H^σ.

□

6.5 Soundness of λ_G

We prove a soudness theorem for λ_G assuming the upcast lemma hold for F_H^σ.

Conjecture 6.14 (Type Soundness). *Suppose* $\cdot;\cdot;\cdot \vdash \boxed{e} \Leftarrow \eta \dashv \cdot;\Psi \rightsquigarrow \underline{e}$ *and* $\mathtt{Solve}(\Psi) = \Sigma$ *and* $\cdot;\cdot \vdash \boxed{[\Sigma]\underline{e}} \Leftarrow [\Sigma]\eta \rightsquigarrow M$. *If* $M \longrightarrow^* M'$ *and* M *does not reduce, then* M' *is a value.*

Proof. By Theorem 6.11 (Soundness of Algorithmic Typing) and Theorem 6.12 (Type Preserving Translation) we have $\cdot \vdash_H \boxed{M} : (\!|[\Sigma]\eta|\!)$. By soudness of F_H^σ, we have that M' must be a result. Finally, assuming the upcast lemma hold and by theorem 6.13 we have that M' cannot be an error and thus it must be a value. □

REFERENCES

[1] Benjamin Cosman and Ranjit Jhala. 2017. Local Refinement Typing. *Proc. ACM Program. Lang.* 1, ICFP, Article 26 (aug 2017), 27 pages. https://doi.org/10.1145/3110270

[2] Taro Sekiyama, Atsushi Igarashi, and Michael Greenberg. 2017. Polymorphic manifest contracts, revised and resolved. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 39, 1 (2017), 1–36. https://doi.org/10.1145/2994594