A New Refinement Type System for Automated $\nu\text{HFL}_{\mathbb{Z}}$ Validity Checking

Hiroyuki Katsura[1], Naoki Iwayama[1], Naoki Kobayashi[1], and Takeshi Tsukada[2]

[1] The University of Tokyo, {katsura,iwayama,koba}@kb.is.s.u-tokyo.ac.jp
[2] Chiba University, tsukada@math.s.chiba-u.ac.jp

Abstract. Kobayashi et al. have recently shown that various verification problems for higher-order functional programs can naturally be reduced to the validity checking problem for $\text{HFL}_{\mathbb{Z}}$, a higher-order fixpoint logic extended with integers. We propose a refinement type system for checking the validity of $\nu\text{HFL}_{\mathbb{Z}}$ formulas, where $\nu\text{HFL}_{\mathbb{Z}}$ is a fragment of $\text{HFL}_{\mathbb{Z}}$ without least fixpoint operators, but sufficiently expressive for encoding safety property verification problems. Our type system has been inspired by the type system of Burn et al. for solving the satisfiability problem for HoCHC, which is essentially equivalent to the $\nu\text{HFL}_{\mathbb{Z}}$ validity checking problem. Our type system is more expressive, however, due to a more sophisticated subtyping relation. We have implemented a type-based $\nu\text{HFL}_{\mathbb{Z}}$ validity checker ReTHFL based on the proposed type system, and confirmed through experiments that ReTHFL can solve more instances than Horus, the tool based on Burn et al.'s type system.

1 Introduction

Kobayashi et al. [7,16] have recently shown that various verification problems for higher-order functional programs can naturally be reduced to the validity checking problem for $\text{HFL}_{\mathbb{Z}}$, an extension of HFL [15] with integers. In this paper, we focus on a fragment of $\text{HFL}_{\mathbb{Z}}$ called $\nu\text{HFL}_{\mathbb{Z}}$, which is a fragment of $\text{HFL}_{\mathbb{Z}}$ without least fixpoint operators, and propose an automated method for solving the validity checking problem (which, in turn, serves as an automated method for higher-order program verification, thanks to the reduction mentioned above). The fragment $\nu\text{HFL}_{\mathbb{Z}}$ is sufficiently expressive for encoding safety properties of programs. A validity checker for $\nu\text{HFL}_{\mathbb{Z}}$ can also be used as a building block for a validity checker for full $\text{HFL}_{\mathbb{Z}}$, as briefly discussed in [16], and worked out for the first-order fixpoint logic [6].

To see the connection between program verification and $\nu\text{HFL}_{\mathbb{Z}}$ validity checking, let us consider the following ML program.

```
let rec sum n k =
    if n <= 0 then k n
    else sum (n - 1) (fun r -> k (n + r))
let main m = sum m (fun r -> assert(r >= m))
```

This program calculates the sum of integers from 1 to n, and then asserts that the value is no less than n. Suppose that we wish to verify that the assertion never fails for any integer n. By using the reduction of Kobayashi et al. [7], the verification problem can be reduced to the validity checking problem for the following $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula.

$$\begin{aligned}\psi \ := \forall m.(&\nu\mathrm{Sum}.\lambda n.\lambda k.\\ &(n \leq 0 \Rightarrow k\ n) \wedge\\ &(n > 0 \Rightarrow \mathrm{Sum}\ (n-1)\ (\lambda r.k\ (n+r)))\\ )\ &m\ (\lambda r.r \geq m)\end{aligned} \tag{1}$$

Here, the part $\nu\mathrm{Sum}.\lambda n.\cdots$ denotes the greatest predicate such that $\mathrm{Sum} = \lambda n.\cdots$. A detailed explanation is deferred to Section 2, but the reader should be able to notice the close correspondence between the program and the formula above: for example, the part $(n \leq 0 \Rightarrow \cdots) \wedge (n > 0 \Rightarrow \cdots)$ corresponds to the conditional expression in the program.

In this paper, we propose a refinement type system for proving the validity of a $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula, and develop an automated procedure for refinement type inference. In our refinement type system, the type of propositions is refined to a type of the form $\bullet\langle\theta\rangle$, which is the type of propositions that hold whenever θ holds; in other words, if a proposition ψ has type $\bullet\langle\theta\rangle$, then θ is an underapproximation of ψ (with respect to the order $\texttt{false} < \texttt{true}$). For example, $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula $x \geq 0$ has type $\bullet\langle x > 0\rangle$ because $x > 0 \Rightarrow x \geq 0$ holds.

Our type system has been inspired by that of Burn et al. [2] for proving the satisfiability of Higher-order Constrained Horn Clauses (HoCHC), a higher-order extension of Constrained Horn Clauses (CHC) [1]. In fact, the HoCHC satisfiability problem[3] is essentially the same as the $\nu\mathrm{HFL}_{\mathbb{Z}}$ validity checking problem (in the sense that for any HoCHC C, there exists a $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula ψ_C such that C is satisfiable if and only if ψ_C is valid, and vice versa). The main difference between our type system and theirs is in the subtyping relation. We introduce more sophisticated subtyping relations, which makes the resulting subtyping relation complete with respect to the semantic subtyping relation. In contrast, the subtyping relation in Burn et al.'s system is too conservative, which makes their type system too weak; in fact, as confirmed through experiments, there are many $\nu\mathrm{HFL}_{\mathbb{Z}}$ formulas whose validity can be proved in our type system but the satisfiability of the corresponding HoCHC cannot be proved in Burn et al.'s type system.

An alternative existing approach to automatically proving the validity of a $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula is a combination of (pure) HFL model checking and predicate abstraction [5]. Though our type-based approach is less powerful in theory than the model checking approach, ours tends to be faster, as confirmed by our experiments. Thus, we consider that the two approaches are complementary.

The rest of this paper is structured as follows. Section 2 reviews the definition of $\nu\mathrm{HFL}_{\mathbb{Z}}$. Section 3 presents our refinement type system for $\nu\mathrm{HFL}_{\mathbb{Z}}$ and proves

[3] Throughout the paper, we assume integer arithmetic as the underlying constraint language of HoCHC.

the soundness of the type system and the relative completeness of the subtyping relation. Section 4 discusses the relationship between our type system for $\nu\mathrm{HFL}_{\mathbb{Z}}$ and Burn et al.'s one for HoCHC. Section 5 presents an automated method for $\nu\mathrm{HFL}_{\mathbb{Z}}$ validity checking based on our type system. Section 6 reports an implementation and experimental results. Section 7 discusses related work, and Section 8 concludes the paper.

2 Preliminaries: $\nu\mathbf{HFL}_{\mathbb{Z}}$

We review the syntax and semantics of $\nu\mathrm{HFL}_{\mathbb{Z}}$ [7], which is a simply-typed higher-order logic with arithmetic operations and the greatest fixed-point operator.

2.1 Syntax

The logic $\nu\mathrm{HFL}_{\mathbb{Z}}$ is simply typed. The syntax of *simple types* is given by:

$$\rho ::= \bullet \mid \eta \to \rho \qquad \text{and} \qquad \eta ::= \rho \mid \mathbf{Int}.$$

The type $\bullet$ is for propositions and **Int** is for integers. The types are constructed from these atomic types and the function type constructor $\to$. The above syntax restricts occurrences of **Int** only to argument positions. The reason will be explained in the next subsection.

The syntax of $\nu\mathrm{HFL}_{\mathbb{Z}}$ formulas is given by:

$$\begin{array}{rcl} \psi & ::= & n \mid \psi_1 \,\mathbf{op}\, \psi_2 \mid \mathbf{p}(\psi_1, \cdots, \psi_n) \mid \mathbf{tt} \mid \mathbf{ff} \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \forall X : \mathbf{Int}.\psi \\ & \mid & X \mid \lambda X : \eta.\psi \mid \psi_1\, \psi_2 \mid \nu X : \rho.\psi \end{array}$$

where n ranges over integers, **op** over basic binary operations on integers (such as summation and multiplication), **p** over basic predicates on integers (such as equality), and X over variables. The constructors in the first line are standard; those in the second line are those from the simply-typed λ-calculus (i.e. variable X, abstraction $\lambda X : \eta.\psi$ and application $\psi_1\, \psi_2$) and the greatest fixed-point operator $\nu X : \rho.\psi$. The occurrences of X in $\forall X : \mathbf{Int}.\psi$, $\lambda X : \eta.\psi$ and $\nu X : \rho.\psi$ are binding occurrences. We shall not distinguish α-equivalent terms. We shall often omit the type annotations. Lower case letters such as x, y and z are sometimes used as variables of type **Int**.

The typing rules are straightforward. A *judgment* is a triple $\Gamma \vdash_H \psi : \eta$, where Γ is a *(simple) type environment* (i.e. finite map from variables to simple types). The type system is basically the simply-typed λ-calculus with typed constants

$$n : \mathbf{Int} \qquad \mathbf{op} : \mathbf{Int} \to \mathbf{Int} \to \mathbf{Int} \qquad \mathbf{p} : \mathbf{Int} \to \cdots \to \mathbf{Int} \to \bullet$$
$$\mathbf{tt}, \mathbf{ff} : \bullet \qquad \vee, \wedge : \bullet \to \bullet \to \bullet$$

and the following additional typing rules:

$$\frac{\Gamma, X : \mathbf{Int} \vdash_H \psi : \bullet}{\Gamma \vdash_H \forall X : \mathbf{Int}.\psi : \bullet} \quad \text{and} \quad \frac{\Gamma, X : \rho \vdash_H \psi : \rho}{\Gamma \vdash_H \nu X : \rho.\psi : \rho}\ .$$

The complete list of typing rules can be found in Appendix A. In the sequel, we shall consider only well-typed formulas.

A closed formula of type $\bullet$ is called a *sentence*.

Example 1. Let ψ be the $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula defined by

$$\psi \quad := \quad \nu X : \mathbf{Int} \to \bullet.\, \lambda y : \mathbf{Int}.\, y \neq 0 \wedge X\,(y+1).$$

The meaning of this formula can be intuitively understood as follows. Since it is a fixed-point, (the meaning of) this formula must be a solution of the equation

$$X \quad = \quad \lambda y.\, y \neq 0 \wedge X\,(y+1).$$

More specifically it is the greatest solution, where a predicate A is greater than B if $\forall n \in \mathbf{Z}.(A\,n \Rightarrow B\,n)$.

A more intuitive way to guess the greatest solution is to iteratively apply the equation. Since (the meaning of) ψ satisfies the above equation, one has

$$\begin{aligned}\psi\,n \quad &= \quad (n \neq 0) \wedge \psi\,(n+1) \quad = \quad (n \neq 0) \wedge (n+1 \neq 0) \wedge \psi\,(n+2) \quad = \quad \cdots \\ &= \quad (n \neq 0) \wedge (n+1 \neq 0) \wedge \cdots \wedge (n+k \neq 0) \wedge \ldots.\end{aligned}$$

This informal argument shows that $\psi\,n$ must be false for every $n \leq 0$. The greatest solution is obtained by letting $\psi\,n$ be true if $\psi\,n$ does not have to be false by this argument based on expansion of the definition. Hence $\psi\,n$ is true for every $n > 0$. □

2.2 Semantics

A type η is interpreted as a poset $\mathcal{D}_\eta$ and a formula ψ of type η as an element of $\mathcal{D}_\eta$. The formal definition is as follows.

The poset $\mathcal{D}_\eta = (\mathcal{D}_\eta, \sqsubseteq_\eta)$ is defined by induction on η:

$$\begin{aligned}&\mathcal{D}_\bullet = \{\top, \bot\} \quad \sqsubseteq_\bullet = \{(\bot,\bot),(\bot,\top),(\top,\top)\} \\ &\mathcal{D}_{\mathbf{Int}} = \mathbb{Z} \quad \sqsubseteq_{\mathbf{Int}} = \{(n,n) \mid n \in \mathbb{Z}\} \\ &\mathcal{D}_{\eta\to\rho} = \{f \in \mathcal{D}_\eta \to \mathcal{D}_\rho \mid \forall x, y.(x \sqsubseteq_\eta y \Rightarrow f(x) \sqsubseteq_\rho f(y))\} \\ &\sqsubseteq_{\eta\to\rho} = \{(f,g) \mid \forall x \in \mathcal{D}_\eta.f(x) \sqsubseteq_\rho g(x)\}.\end{aligned}$$

We note that $\mathcal{D}_{\eta\to\rho}$ is not the set of all functions but *monotone* functions. Observe that $\mathcal{D}_\rho$ is a complete lattice (i.e., for each subset $A \subseteq \mathcal{D}_\rho$, the greatest lower bound $\bigsqcap A$ of A exists). The interpretation $\mathcal{D}_{\mathbf{Int}}$ is not a complete lattice, and this is why we distinguish **Int** from other simple types.

For a simple type environment Γ, we write $[\![\Gamma]\!]$ for the set of functions that maps a variable X in (the domain of) Γ to an element of $\mathcal{D}_{\Gamma(X)}$. We call an element of $[\![\Gamma]\!]$ a *valuation*. Valuations are ordered by the point-wise ordering.

The interpretation $[\![\psi]\!]$ of a formula $\Gamma \vdash_H \psi : \eta$ is a *monotone* function from $[\![\Gamma]\!]$ to $\mathcal{D}_\eta$. It is defined by induction on ψ. For example,

$$[\![\nu X : \rho.\psi]\!](\chi) \quad := \quad \bigsqcap \{\, v \in \mathcal{D}_\rho \mid v \sqsubseteq_\rho [\![\psi]\!](\chi[X \mapsto v]) \,\}$$

where $\chi[X \mapsto v]$ is the valuation defined by $\chi[X \mapsto v](X) = v$ and $\chi[X \mapsto v](Y) = \chi(Y)$ $(X \neq Y)$. The right-hand-side of the above definition is an explicit formula that calculates the greatest fixed-point of the mapping $v \mapsto [\![\psi]\!](\chi[X \mapsto v])$. The well-definedness and correctness of this explicit formula is ensured by the facts that $\mathcal{D}_\rho$ is a complete lattice and that $v \mapsto [\![\psi]\!](\chi[X \mapsto v])$ is monotone. We omit other cases since they are straightforward; see Appendix B for the complete definition.

We write the interpretation of a sentence ψ as $[\![\psi]\!]$ since it is independent of a valuation (as a sentence has no free variable). If $[\![\psi]\!](\emptyset) = \top$, then the sentence ψ is *valid* and we write $\models \psi$. The *νHFL$_{\mathbb{Z}}$ validity checking problem* is the problem of checking whether a given sentence is valid. Note that this problem is undecidable in general.

Example 2. Let us consider the following formula $\nu\text{HFL}_{\mathbb{Z}}$ formula:

$$\begin{aligned}\phi \;:= \forall m.(&\nu \mathrm{Sum}.\lambda n.\lambda k.\\ &(n > 0 \vee k\ n) \wedge\\ &(n \leq 0 \vee \mathrm{Sum}\ (n-1)\ (\lambda r.k\ (n+r)))\\ )\ m\ &(\lambda r.r \geq m).\end{aligned}$$

This formula is essentially the same as the example in Introduction (Section 1) except that $\Rightarrow$ is replaced with with other connectives (since $\Rightarrow$ is not in $\nu\text{HFL}_{\mathbb{Z}}$). The relationship between this formula and the safety verification of the program at the beginning of Introduction can be now explained as follows.

The reduction of the program corresponds to the β-reduction, the expansion of Sum (cf. Example 1), and some trivial rewriting of formulas such as $(0 \neq 0) \vee \delta \longrightarrow \delta$. The safety verification asks whether the program fails in some finite steps. If the program fails, then the corresponding rewriting of the formula shows that the formula is false. If there is no such rewriting, the formula is true as expected since the greatest fixed-point is true "by default" (cf. Example 1). □

3 Refinement Type System

This section introduces a refinement type system, which our validity checker is based on. The refinement type system introduced in this section is inspired by and closely related to that of Burn et al. [2]. This section focuses on our refinement type system; a comparison of the two systems is the topic of the next section.

3.1 Syntax of Refinement Types

Our type system uses refinement types to describe properties of formulas. Here we define the syntax and semantics of refinement types.

The syntax of *refinement types* is given by the following grammar:

$$
\begin{array}{ll}
\textit{arithmetic expressions} & \mathbf{a} ::= n \mid x \mid \mathbf{op}(\mathbf{a}_1, \cdots, \mathbf{a}_n) \\
\textit{constraint formulas} & \theta ::= \mathbf{tt} \mid \mathbf{ff} \mid \mathbf{p}(\mathbf{a}_1, \cdots, \mathbf{a}_n) \mid \theta_1 \wedge \theta_2 \mid \theta_1 \vee \theta_2 \\
\textit{extended constraint formulas} & \Theta ::= \theta \mid \Theta_1 \wedge \Theta_2 \mid \exists x.\Theta \\
\textit{refinement types} & \tau ::= \bullet\langle\theta\rangle \mid x : \mathbf{Int} \to \tau \mid \tau_1 \to \tau_2.
\end{array}
$$

The occurrence of x in $x : \mathbf{Int} \to \tau$ is a binding occurrence. We shall not distinguish between α-equivalent refinement types.

Each refinement type τ describes a property on formulas and semantic elements of a simple type ρ. This relationship is formalized as the *refinement relation*, which is defined by the following rules:

$$
\frac{}{\bullet\langle\theta\rangle :: \bullet} \qquad \frac{\tau :: \rho}{(x : \mathbf{Int} \to \tau) :: (\mathbf{Int} \to \rho)} \qquad \frac{\tau_1 :: \rho_1 \quad \tau_2 :: \rho_2}{(\tau_1 \to \tau_2) :: (\rho_1 \to \rho_2)}.
$$

For every refinement type τ, there exists a unique simple type ρ such that $\tau :: \rho$. We write $\Gamma \vdash \tau :: \rho$ if $\tau :: \rho$ and $\mathrm{fv}(\tau) \subseteq \{\, x \mid \Gamma(x) = \mathbf{Int} \,\}$.

The meaning of arithmetic expressions and constraint formulas should be obvious. We explain the intuitive meaning of refinement types. If $\tau :: \rho$, then τ is for formulas of simple type ρ that satisfies a certain property.

A formula ψ of type $\bullet$ has the refinement type $\bullet\langle\theta\rangle$ if θ implies ψ. More precisely, the type judgement $\psi : \bullet\langle\theta\rangle$ means "if θ holds, then the interpretation of ψ is $\top$." The simplest example is $\bullet\langle\mathbf{tt}\rangle$; if $\psi : \bullet\langle\mathbf{tt}\rangle$, then the interpretation of ψ is $\top$. Another extreme example is $\bullet\langle\mathbf{ff}\rangle$; $\psi : \bullet\langle\mathbf{ff}\rangle$ holds for every formula ψ of simple type $\bullet$ since the condition $\mathbf{ff}$ never holds. Both ψ and θ may contain free variables. For example, $\psi : \bullet\langle x > 0\rangle$ holds if the interpretation of $\psi[n/x]$ is $\top$ for every $n > 0$.

The meaning of the refinement type $\tau_1 \to \tau_2$ is similar to the standard function type. A formula ψ has type $\tau_1 \to \tau_2$ just if $\psi\,\phi : \tau_2$ for every formula ϕ of type τ_1.

The meaning of $x : \mathbf{Int} \to \tau$ is similar to the above case, but τ can refer to the argument x in this case. For example, $x : \mathbf{Int} \to \bullet\langle x > 0\rangle$ is for formulas ψ of simple type $\mathbf{Int} \to \bullet$ such that $\psi\, n : \bullet\langle n > 0\rangle$ for every n.[4] In other words, it is a type for predicates that are true on every positive integer.

It is worth emphasising that a refinement type describes a situation in which a formula should be *true*. It does not say anything about a situation in which a formula should be *false*. Therefore the constantly true function $\lambda X : \rho.\mathbf{tt}$ has all refinement type τ such that $\tau :: \rho \to \bullet$. So a (valid) refinement type judgement $\psi : \tau$ gives an underapproximation of ψ.

[4] Equivalently, $\psi\, x : \bullet\langle x > 0\rangle$, provided that ψ has no free occurrence of x.

3.2 Semantics of Refinement Types

In order to clarify the informal definition of the meaning of refinement types given above, we formalize the semantics of refinement types. For a refinement type $\tau :: \rho$, we give two interpretations. In the first interpretation, the refinement type is interpreted as the subset $(\!|\tau|\!) \subseteq \mathcal{D}_\rho$ of semantic elements that satisfies τ. This is a direct formarization of the above discussed meaning of refinement types. In the second interpretation, the refinement type is seen as an element $\gamma_\tau \in \mathcal{D}_\rho$. As expected, the two interpretations are closely related: we have $(\!|\tau|\!) = \{\, v \in \mathcal{D}_\rho \mid \gamma_\tau \sqsubseteq_\rho v \,\}$.

We give some auxiliary definitions. The interpretation $[\![\theta]\!]$ of constraint formulas θ is straightforward as constraint formulas can be seen as $\nu\mathrm{HFL}_{\mathbb{Z}}$ formulas. It is a map from valuations α on free variables of θ to $\mathcal{D}_\bullet = \{\, \bot, \top \,\}$. The interpretation can be naturally extended to extended constraint formulas by $[\![\exists x.\Theta]\!](\alpha) := \bigsqcup_{v \in \mathbb{Z}} [\![\Theta]\!](\alpha[x \mapsto v])$.

The first interpretation $(\!|\tau|\!)$ of a refinement type $\Gamma \vdash \tau :: \rho$ is a function from valuations $\alpha \in [\![\Gamma]\!]$ to subsets $(\!|\tau|\!)(\alpha) \subseteq \mathcal{D}_\rho$ of the interpretation of ρ. It is defined by induction on the structure as follows:

$$
\begin{aligned}
(\!|\bullet \langle\theta\rangle|\!)(\alpha) &:= \begin{cases} \{\, \top \,\} & (\text{if } \alpha \models \theta) \\ \{\, \bot, \top \,\} & (\text{if } \alpha \not\models \theta) \end{cases} \\
(\!|x : \mathbf{Int} \to \tau|\!)(\alpha) &:= \{\, f \in \mathcal{D}_{\mathbf{Int} \to \rho} \mid \forall v \in \mathcal{D}_{\mathbf{Int}}.\, f(v) \in (\!|\tau|\!)(\alpha[x \mapsto v]) \,\} \\
(\!|\tau_1 \to \tau_2|\!)(\alpha) &:= \{\, f \in \mathcal{D}_{\rho_1 \to \rho_2} \mid \forall v \in (\!|\tau_1|\!)(\alpha).\, f(v) \in (\!|\tau_2|\!)(\alpha) \,\}.
\end{aligned}
$$

This is basically a direct translation of the informal semantics discussed in the previous subsection.

The second interpretation γ_τ is a map from $[\![\Gamma]\!]$ to $\mathcal{D}_\rho$, inductively defined by

$$
\begin{aligned}
\gamma_{\bullet\langle\theta\rangle}(\alpha) &:= \begin{cases} \top_\bullet & (\text{if } \alpha \models \theta) \\ \bot_\bullet & (\text{if } \alpha \not\models \theta) \end{cases} \\
\gamma_{x:\mathbf{Int}\to\tau}(\alpha) &:= \Big[\mathcal{D}_{\mathbf{Int}} \ni v \;\mapsto\; \gamma_\tau(\alpha[x \mapsto v])\Big] \\
\gamma_{\tau_1 \to \tau_2}(\alpha) &:= \left[\mathcal{D}_{\rho_1} \ni v \;\mapsto\; \begin{cases} \gamma_{\tau_2}(\alpha) & (\text{if } \gamma_{\tau_1}(\alpha) \sqsubseteq_{\rho_1} v) \\ \bot_{\rho_2} & (\text{otherwise}) \end{cases}\right]
\end{aligned}
$$

where we assume $(\tau_1 \to \tau_2) :: (\rho_1 \to \rho_2)$ in the last case. Here $\top_\rho$ and $\bot_\rho$ are the greatest and least element of $\mathcal{D}_\rho$. The element $\gamma_\tau(\alpha)$ is the minimum element in $(\!|\tau|\!)(\alpha)$.

Lemma 1. *Assume $\Gamma \vdash \tau :: \rho$ and $\alpha \in [\![\Gamma]\!]$. Then*

$$
\forall v \in \mathcal{D}_\rho. \Big[v \in (\!|\tau|\!)(\alpha) \quad\Longleftrightarrow\quad \gamma_\tau(\alpha) \sqsubseteq_\rho v\Big].
$$

$$\frac{\Delta(x) = \tau}{\Delta \vdash x : \tau} \text{ (RVar)} \qquad \frac{\Delta, x : \mathbf{Int} \vdash \psi : \bullet\langle\theta\rangle \quad x \notin \mathbf{fv}(\theta)}{\Delta \vdash \forall x^{\mathbf{Int}}.\psi : \bullet\langle\theta\rangle} \text{ (RAllI)}$$

$$\frac{\Delta \vdash \psi : x : \mathbf{Int} \to \tau}{\Delta \vdash \psi\, \mathbf{a} : [\mathbf{a}/x]\tau} \text{ (RAppI)} \qquad \frac{}{\Delta \vdash \mathbf{p}(\mathbf{a}_1, \cdots, \mathbf{a}_n) : \bullet\langle \mathbf{p}(\mathbf{a}_1, \cdots, \mathbf{a}_n)\rangle} \text{ (RPred)}$$

$$\frac{\Delta \vdash \psi_1 : \bullet\langle\theta_1\rangle \qquad \Delta \vdash \psi_2 : \bullet\langle\theta_2\rangle}{\Delta \vdash \psi_1 \wedge \psi_2 : \bullet\langle\theta_1 \wedge \theta_2\rangle} \text{ (RAnd)} \qquad \frac{\Delta \vdash \psi_1 : \tau_1 \to \tau_2 \qquad \Delta \vdash \psi_2 : \tau_1}{\Delta \vdash \psi_1\, \psi_2 : \tau_2} \text{ (RApp)}$$

$$\frac{\Delta \vdash \psi_1 : \bullet\langle\theta_1\rangle \qquad \Delta \vdash \psi_2 : \bullet\langle\theta_2\rangle}{\Delta \vdash \psi_1 \vee \psi_2 : \bullet\langle\theta_1 \vee \theta_2\rangle} \text{ (ROr)}$$

$$\frac{\Delta, X : \tau \vdash \psi : \tau}{\Delta \vdash \nu X.\psi : \tau} \text{ (RGfp)} \qquad \frac{\Delta \vdash \psi : \tau_1 \qquad \Delta; \mathbf{tt} \vdash \tau_1 \prec \tau_2}{\Delta \vdash \psi : \tau_2} \text{ (RSub)}$$

$$\frac{\Delta, x : \mathbf{Int} \vdash \psi : \tau}{\Delta \vdash \lambda x.\psi : x : \mathbf{Int} \to \tau} \text{ (RAbsI)} \qquad \frac{\Delta, x : \tau_1 \vdash \psi : \tau_2}{\Delta \vdash \lambda x.\psi : \tau_1 \to \tau_2} \text{ (RAbs)}$$

Fig. 1. Refinement typing rules

$$\frac{\Delta \models \Theta \wedge \theta_2 \Rightarrow \theta_1}{\Delta; \Theta \vdash \bullet\langle\theta_1\rangle \prec \bullet\langle\theta_2\rangle} \text{ (S-Bool)} \qquad \frac{\Delta, x : \mathbf{Int}; \Theta \vdash \tau_1 \prec \tau_2}{\Delta; \Theta \vdash x : \mathbf{Int} \to \tau_1 \prec x : \mathbf{Int} \to \tau_2} \text{ (S-IntFun)}$$

$$\frac{\Delta; \Theta \wedge \mathbf{rty}(\tau_2') \vdash \tau_1' \prec \tau_1 \qquad \Delta; \Theta \vdash \tau_2 \prec \tau_2'}{\Delta; \Theta \vdash \tau_1 \to \tau_2 \prec \tau_1' \to \tau_2'} \text{ (S-Fun)}$$

Fig. 2. Subtyping rules

3.3 Typing Rules

Now we define our refinement type system by giving the typing rules.

A *refinement type environment* Δ is a finite map from a subset of variables to refinement types or **Int**. We write $\Delta :: \Gamma$ if the domains of Δ and Γ coincide and $\Delta(X) :: \Gamma(X)$ for every X in the domain. Here we assume $\mathbf{Int} :: \mathbf{Int}$.

A *refinement type judgement* is a triple $\Delta \vdash \psi : \tau$. We shall only consider a refinement type judgement that refines a simple type judgement. That means, when we consider $\Delta \vdash \psi : \tau$, we implicitly assume a simple type judgement $\Gamma \vdash_H \psi : \rho$ and refinement relations $\Delta :: \Gamma$ and $\Gamma \vdash \tau :: \rho$.

Figure 1 shows typing rules of the refinement type system. We explain some key rules. The rule RAnd says that $\theta_1 \wedge \theta_2$ is an underapproximation of $\psi_1 \wedge \psi_2$ if θ_i is an underapproximation of ψ_i for $i = 1, 2$. The rule RAppI substitutes the actual argument $\mathbf{a}$ for x in τ. The rule RGfp is the standard coinductive (i.e. greatest) fixed-point rule, saying that the fixed-point $\nu X.\psi$ has type τ if ψ has type τ under the assumption that X has type τ. The most important rule for this paper is RSub, which allows us to construct a derivation of $\Delta \vdash \psi : \tau_2$ from that of $\Delta \vdash \psi : \tau_1$ under a certain assumption. We explain this rule in more detail.

The rule RSub refers to the subtyping judgement $\Delta; \Theta \vdash \tau_1 \prec \tau_2$, defined by the *subtyping rules* listed in Fig. 2. Among the rules in Fig. 2, S-Fun is the only nontrivial rule. Similar to the standard subtyping rule for function types, it concludes $\tau_1 \to \tau_2 \prec \tau_1' \to \tau_2'$ from $\tau_1' \prec \tau_1$ and $\tau_2 \prec \tau_2'$. A notable point is

that the assumption for $\tau_1' \prec \tau_1$ is strengthened by $\mathbf{rty}(\tau_2')$, which is defined by the following equations:

$$\mathbf{rty}(\bullet\langle\theta\rangle) := \theta \qquad \mathbf{rty}(x{:}\mathbf{Int} \to \tau) := \exists x.\mathbf{rty}(\tau) \quad \text{and} \quad \mathbf{rty}(\tau_1 \to \tau_2) := \mathbf{rty}(\tau_2).$$

A key property of $\mathbf{rty}(\tau)$ is the following lemma.

Lemma 2. *Assume $\Gamma \vdash \tau :: \rho$ and $\alpha \in [\![\Gamma]\!]$. If $\alpha \not\models \mathbf{rty}(\tau)$, then $(\!|\tau|\!)(\alpha) = \mathcal{D}_\rho$.*

This means that, if $\mathbf{rty}(\tau)$ is false, then τ_2 is the trivial property that all elements satisfy. Therefore, to show that $\tau \prec \tau'$, we can assume without loss of generality that $\mathbf{rty}(\tau')$ holds because otherwise $\tau \prec \tau'$ trivially holds. This explains why we can assume $\mathbf{rty}(\tau_2')$ in the premise of S-Fun.[5]

The significance of the assumption $\mathbf{rty}(\tau_2')$ in S-Fun is demonstrated by the next example.

Example 3. Recall the formula ψ in Introduction (Section 1) and Example 2:

$$\forall m.(\nu\mathrm{Sum}.\lambda n.\lambda k.(n > 0 \vee k\, n) \wedge (n \leq 0 \vee \mathrm{Sum}\,(n-1)\,(\lambda r.k(r+n))))\; m\; (\lambda r.r \geq m).$$

We would like to show that $\vdash \psi : \bullet\langle\mathbf{tt}\rangle$, which implies the validity of ψ as we shall see. The most interesting part is the typing of $(\nu\mathrm{Sum}.\ldots)$:

$$\vdash (\nu\mathrm{Sum}.\ldots) \;:\; n : \mathbf{Int} \to (x : \mathbf{Int} \to \bullet\langle x \geq n\rangle) \to \bullet\langle\mathbf{tt}\rangle.$$

Let Δ be the refinement type environment:

$$\mathrm{Sum} : \big(n{:}\mathbf{Int} \to (x{:}\mathbf{Int} \to \bullet\langle x \geq n\rangle) \to \bullet\langle\mathbf{tt}\rangle\big),\quad n{:}\mathbf{Int},\quad k : \big(x{:}\mathbf{Int} \to \bullet\langle x \geq n\rangle\big).$$

It suffices to show:

$$\Delta \vdash (n > 0 \vee k\, n) \wedge (n \leq 0 \vee \mathrm{Sum}\,(n-1)\,(\lambda r.k(r+n))) \;:\; \bullet\,\langle\mathbf{tt}\rangle.$$

We have:

$$\dfrac{\dfrac{\vdots}{(n > 0 \vee k\, n) : \bullet\langle\mathbf{tt}\rangle} \qquad \dfrac{\dfrac{}{n \leq 0 : \bullet\langle n \leq 0\rangle} \quad \dfrac{\vdots}{\mathrm{Sum}\,(n-1)\,(\lambda r.k(r+n)) : \bullet\langle n > 0\rangle}}{(n \leq 0 \vee \mathrm{Sum}\,(n-1)\,(\lambda r.k(r+n))) : \bullet\langle\mathbf{tt}\rangle}}{(n > 0 \vee k\, n) \wedge (n \leq 0 \vee \mathrm{Sum}\,(n-1)\,(\lambda r.k(r+n))) \;:\; \bullet\,\langle\mathbf{tt}\rangle}$$

where we omit $\Delta \vdash$ from each judgement and implicitly rewrite $\bullet\langle n \leq 0 \vee n > 0\rangle$ to $\bullet\langle\mathbf{tt}\rangle$. Since the left judgement is easy to show, we focus on the right judgement.

We have

$$\Delta \vdash \mathrm{Sum}\,(n-1) \;:\; (r : \mathbf{Int} \to \bullet\langle r \geq n-1\rangle) \to \bullet\langle\mathbf{tt}\rangle$$

[5] A reader may wonder why we do not assume $\mathbf{rty}(\tau_2')$ in the other premise. This is because the subtyping judgements $\Delta; \Theta \vdash \tau_2 \prec \tau_2'$ and $\Delta; \Theta \wedge \mathbf{rty}(\tau_2') \vdash \tau_2 \prec \tau_2'$ are equivalent in the sense that the derivability of one of them implies the other's. We chose the simpler judgement.

$$\dfrac{\dfrac{\dfrac{\Delta, r : \mathbf{Int} \models n > 0 \wedge r \geq n - 1 \Rightarrow r \geq 0}{\Delta, r : \mathbf{Int}; n > 0 \vdash \bullet\langle r \geq 0\rangle \prec \bullet\langle r \geq n - 1\rangle}}{\Delta; n > 0 \vdash r : \mathbf{Int} \to \bullet\langle r \geq 0\rangle \prec r : \mathbf{Int} \to \bullet\langle r \geq n - 1\rangle} \qquad \dfrac{\Delta \models n > 0 \Rightarrow \mathbf{tt}}{\Delta; \mathbf{tt} \vdash \bullet\langle \mathbf{tt}\rangle \prec \bullet\langle n > 0\rangle}}{\Delta; \mathbf{tt} \vdash (r : \mathbf{Int} \to \bullet\langle r \geq n - 1\rangle) \to \bullet\langle \mathbf{tt}\rangle \prec (r : \mathbf{Int} \to \bullet\langle r \geq 0\rangle) \to \bullet\langle n > 0\rangle}$$

Fig. 3. A derivation of a subtyping judgement used in Example 3

but this is not immediately usable since

$$\Delta \nvdash (\lambda r.k(r + n))) \ : \ r : \mathbf{Int} \to \bullet\langle r \geq n - 1\rangle.$$

Actually this judgement is *invalid*[6]: the type of k requires that $r + n \geq n$ but $r \geq n - 1$ is not sufficient for this when $n \leq 0$. Therefore one needs subtyping.

Figure 3 proves a subtyping judgement. Note that the assumption $n > 0$ plays a crucial role in the left branch of the derivation. Since $\Delta \vdash (\lambda r.k(r+n))) : (r : \mathbf{Int} \to \bullet\langle r \geq 0\rangle)$ is easily provable, we have completed the proof. □

3.4 Soundness and Completeness

This subsection defines the semantic counterpart of (sub)typing judgements, and discuss soundness and completeness of the refinement type system.

The interpretation of a refinement type environment $\Delta :: \Gamma$ is the subset $[\![\Delta]\!] \subseteq [\![\Gamma]\!]$ defined by

$$[\![\Delta]\!] \quad := \quad \{\, \alpha \in [\![\Gamma]\!] \mid \forall X \in \mathrm{dom}(\Gamma).\, \alpha(X) \in [\![\Delta(X)]\!](\alpha) \,\}.$$

We write $[\![\Delta; \Theta]\!]$ for the set of valuations $\{\alpha \in [\![\Delta]\!] \mid \alpha \models \Theta\}$.

The semantic counterpart of (sub)typing judgements are defined as follows:

$$\begin{array}{rcl} \Delta; \Theta \models \tau \prec \tau' & :\Longleftrightarrow & (\!|\tau|\!)(\alpha) \subseteq (\!|\tau'|\!)(\alpha) \text{ for every } \alpha \in [\![\Delta; \Theta]\!] \\ \Delta \models \psi : \tau & :\Longleftrightarrow & [\![\psi]\!](\alpha) \in (\!|\tau|\!)(\alpha) \text{ for every } \alpha \in [\![\Delta]\!]. \end{array}$$

The (sub)typing rules are sound with respect to the semantics of judgements.

Theorem 1 (Soundness).

- *If* $\Delta; \Theta \vdash \tau_1 \prec \tau_2$*, then* $\Delta; \Theta \models \tau_1 \prec \tau_2$.
- *If* $\Delta \vdash \psi : \tau$*, then* $\Delta \models \psi : \tau$.

Proof. By induction on the derivations. See Appendix D. □

By applying Soundness to sentences, one can show that a derivation in the refinement type system witnesses the validity of a sentence.

[6] The formal definition of the validity of a refinement type judgement will be defined in the next subsection.

Corollary 1. *Let ψ be a $\nu HFL_{\mathbb{Z}}$ sentence. If $\vdash \psi : \bullet\langle \boldsymbol{tt} \rangle$, then $\models \psi$.*

A remarkable feature is completeness. Although the type system is not complete for typing judgements, it is complete for subtyping judgements.

Theorem 2 (Completeness of subtyping). *If $\Delta; \Theta \models \tau_1 \prec_\rho \tau_2$, then $\Delta; \Theta \vdash \tau_1 \prec_\rho \tau_2$.*

Proof (Sketch). By induction on the structure of simple type ρ. Here we prove only the case $\rho = \rho_1 \to \rho_2$. A complete proof can be found in Appendix E.

In this case $\tau = \tau_1 \to \tau_2$ and $\tau' = \tau_1' \to \tau_2'$. Assume that $\Delta; \Theta \models \tau \prec \tau'$. We prove $\Delta; \Theta \models \tau_2 \prec \tau_2'$ and $\Delta; \Theta \wedge \mathbf{rty}(\tau_2') \models \tau_1' \prec \tau_1$. Then $\Delta; \Theta \vdash \tau \prec \tau'$ follows from the induction hypothesis and S-Fun.

We prove $\Delta; \Theta \models \tau_2 \prec \tau_2'$. Let $\alpha \in [\![\Delta; \Theta]\!]$ and $v \in (\!|\tau_2|\!)(\alpha)$ and define $f \in (\!|\tau_1 \to \tau_2|\!)(\alpha)$ by $f(x) := v$. By the assumption, $f \in (\!|\tau_1' \to \tau_2'|\!)(\alpha)$. Since $\top_{\rho_1} \in (\!|\tau_1'|\!)(\alpha)$, we have $f(\top_{\rho_1}) = v \in (\!|\tau_2'|\!)(\alpha)$. Since $v \in (\!|\tau_2|\!)(\alpha)$ is arbitrary, we obtain $(\!|\tau_2|\!)(\alpha) \subset (\!|\tau_2'|\!)(\alpha)$.

We prove $\Delta; \Theta \wedge \mathbf{rty}(\tau_2') \models \tau_1' \prec \tau_1$. Assume for contradiction that $\Delta; \Theta \wedge \mathbf{rty}(\tau_2') \not\models \tau_1' \prec \tau_1$. Then, there exist $\alpha \in [\![\Delta; \Theta \wedge \mathbf{rty}(\tau_2')]\!]$ and $g \in (\!|\tau_1'|\!)(\alpha)$ such that $g \notin (\!|\tau_1|\!)(\alpha)$. By Lemma 1, we have the minimal element $\gamma_{\tau_1 \to \tau_2}(\alpha)$ in $(\!|\tau_1 \to \tau_2|\!)(\alpha)$, which belongs to $(\!|\tau_1' \to \tau_2'|\!)(\alpha)$ by the assumption. Since $g \in (\!|\tau_1'|\!)(\alpha)$, we have $\gamma_{\tau_1 \to \tau_2}(\alpha)(g) \in (\!|\tau_2'|\!)(\alpha)$. One can prove that $\alpha \models \mathbf{rty}(\tau_2')$ implies $\bot_{\rho_2} \notin (\!|\tau_2'|\!)(\alpha)$ and thus $\gamma_{\tau_1 \to \tau_2}(\alpha)(g) \neq \bot_{\rho_2}$. On the other hand, from the definition of the minimal element $\gamma_{\tau_1 \to \tau_2}(\alpha)$ and the assumption $g \notin (\!|\tau_1|\!)(\alpha)$, we have $\gamma_{\tau_1 \to \tau_2}(\alpha)(g) = \bot_{\rho_2}$, a contradiction. □

4 Relationship with Higher-Order Constrained Horn Clauses

Our work is closely related to the work on *Higher-order constrained Horn clauses* (HoCHC for short) [2]. HoCHC has been introduced by Burn et al. [2] as a higher-order extension of the standard notion of constrained Horn clauses. They also gave a refinement type system that proves the satisfiability of higher-order constrained Horn clauses. The satisfiability problem of higher-order constrained Horn clauses is equivalent to the validity problem of $\nu\mathrm{HFL}_{\mathbb{Z}}$, and the refinement type system of Burn et al. [2] is almost identical to ours, *except for the crucial difference in the subtyping rules*. Below we discuss the connection and the difference between our work on their work in more detail; readers who are not familiar with HoCHC may safely skip the rest of this section.

4.1 The Duality of $\nu\mathbf{HFL}_{\mathbb{Z}}$ and HoCHC

A HoCHC is of the form[7] $\psi \implies Z$, where ψ is a $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula that does not contain the fixed-point operator ν and Z is a variable X or the constant

[7] The syntax of HoCHC is modified in a way that emphasises the relationship to $\nu\mathrm{HFL}_{\mathbb{Z}}$.

ff whose simple type is the same as ψ. The formula ψ in HoCHC may have free variables that possibly include X. A valuation α *satisfies* the HoCHC if $[\![\psi]\!](\alpha) \sqsubseteq [\![Z]\!](\alpha)$. A *solution* of a set of HoCHCs is a valuation that satisfies all given HoCHCs. Burn et al. [2] studied the *HoCHC satisfiability problem*, which asks whether a given finite set of HoCHC has a solution.

The HoCHC satisfiability problem can be characterized by using the least fixed-points. Assume a set of HoCHCs $\mathcal{C} = \{\psi_0 \Longrightarrow \mathbf{ff}, \psi_1 \Longrightarrow X_1, \ldots, \psi_n \Longrightarrow X_n\}$, where $X_1, \ldots, X_n$ are pairwise distinct variables. The HoCHCs $\{\, \psi_1 \Longrightarrow X_1,\ \ldots,\ \psi_n \Longrightarrow X_n \,\}$ has the minimum solution, say α, and $\mathcal{C}$ has a solution if and only if $[\![\psi_0]\!](\alpha) = \bot$ for the minimum solution α.

The connection to the $\nu\mathrm{HFL}_{\mathbb{Z}}$ validity problem becomes apparent when we consider the dual problem. Given a ν-free formula ψ, we write $\overline{\psi}$ for the *dual* of ψ obtained by replacing $\wedge$ with $\vee$, **ff** with **tt**, atomic predicates $\mathbf{p}(\vec{\mathbf{a}})$ with its negation $\neg\mathbf{p}(\vec{\mathbf{a}})$ and a variable X with the dual variable $\overline{X}$. Then $\mathcal{C}$ has a solution if and only if so does

$$\{\, \overline{\psi}_0 \Longleftarrow \mathbf{tt},\ \overline{\psi}_1 \Longleftarrow \overline{X}_1,\ \ldots,\ \overline{\psi}_n \Longleftarrow \overline{X}_n \,\}.$$

This dual problem has a characterisation using the greatest fixed-points: it has a solution if and only if $[\![\overline{\psi}_0]\!](\alpha) = \top$ where α is the *greatest* solution α of $\{\, \overline{\psi}_1 \Longleftarrow \overline{X}_1,\ \ldots,\ \overline{\psi}_n \Longleftarrow \overline{X}_n \,\}$. Since the greatest solution satisfies $\overline{\psi}_i = \overline{X}_i$ for every i, it can be represented by using the greatest fixed-point operator ν of $\nu\mathrm{HFL}_{\mathbb{Z}}$. By substituting $\overline{X}_i$ in $\overline{\psi}_0$ with the $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula representation of the greatest solution α, one obtains a $\nu\mathrm{HFL}_{\mathbb{Z}}$ formula ϕ. Now $\mathcal{C}$ has a solution if and only if $[\![\phi]\!] = \top$, that means, ϕ is valid.

4.2 The similarity and difference between two refinement type systems

The connection between HoCHC and $\nu\mathrm{HFL}_{\mathbb{Z}}$ allows us to compare the refinement type system for HoCHC of Burn et al. [2] with our refinement type system for $\nu\mathrm{HFL}_{\mathbb{Z}}$. In fact, as mentioned in Introduction, this work is inspired by their work. Our refinement type system is almost identical to that of Burn et al. [2], but there is a significant difference. The subtyping rule for function types in their type system corresponds to:

$$\frac{\Delta; \Theta \vdash \tau_1' \prec \tau_1 \qquad \Delta; \Theta \vdash \tau_2 \prec \tau_2'}{\Delta; \Theta \vdash \tau_1 \to \tau_2 \prec \tau_1' \to \tau_2'}.$$

The difference from S-Fun is that $\mathbf{rty}(\tau_2')$ cannot be used to prove $\tau_1' \prec \tau_1$. Because of this difference, our refinement type system is strictly more expressive than that of Burn et al. [2]. Their refinement type system cannot prove the (judgement corresponding to the) subtyping judgement in Example 3, namely,

$$\Delta; \mathbf{tt} \vdash \big((r\!:\!\mathbf{Int} \to \bullet\langle r \geq n-1\rangle) \to \bullet\langle \mathbf{tt}\rangle\big) \ \prec\ \big((r\!:\!\mathbf{Int} \to \bullet\langle r \geq 0\rangle) \to \bullet\langle n > 0\rangle\big);$$

recall that $\mathbf{rty}((r : \mathbf{Int} \to \bullet\langle r \geq 0\rangle) \to \bullet\langle n > 0\rangle) = (n > 0)$ is crucial in the derivation of the subtyping judgement in Example 3. In fact, their type system cannot prove that the sentence in Example 3 is valid.

The difference is significant from both theoretical and practical view points. Theoretically our change makes the subtyping rules complete (Theorem 2). Practically this change is needed to prove the validity of higher-order instances. We will confirm this claim by experiments in Section 6.

5 Type Inference

This section discusses a type inference algorithm for our refinement type system in Section 3. The type system is based on constraint generation and solving. The constraint solving procedure simply invokes external solvers such as Spacer [8], HoIce [3] and PCSat [11]. In what follows, we describe the constraint generation algorithm and discuss the shape of generated constraints.

5.1 Constraint generation

The constraint generation algorithm adopts the template-based approach. For each subformula $\Gamma \vdash \phi : \rho$ of a given sentence $\vdash \psi : \bullet$, we prepare a refinement type template, which is a refinement type with predicate variables. For example, if $\Gamma = (X : \rho', y : \mathbf{Int}, Z : \rho'')$ and $\rho = \mathbf{Int} \to (\mathbf{Int} \to \bullet) \to \mathbf{Int} \to \bullet$, then the template is $a : \mathbf{Int} \to (b : \mathbf{Int} \to \bullet\langle P(y, a, b)\rangle) \to c : \mathbf{Int} \to \bullet\langle Q(y, a, c)\rangle$. The ideas are: (i) for each occurrence of type $\mathbf{Int}$, we give a fresh variable of type $\mathbf{Int}$ (in the above example, a, b and c), and (ii) for each occurrence of type $\bullet$, we give a fresh predicate variable (in the above example, P and Q). The arity of each predicate variable is the number of integer variables available at the position. Recall that the scope of x in $(x : \mathbf{Int} \to \tau)$ is τ.

Then we extract constraints. For example, assume that

$$\frac{x : \mathbf{Int} \vdash \phi_1 : (\mathbf{Int} \to \bullet) \to \bullet \qquad x : \mathbf{Int} \vdash \phi_2 : \mathbf{Int} \to \bullet}{x : \mathbf{Int} \vdash \phi_1\, \phi_2 : \bullet}$$

is a part of the simple type derivation of the input sentence. Then the refinement type templates for ϕ_1 and ϕ_2 are

$$(y : \mathbf{Int} \to \bullet\langle P(x, y)\rangle) \to \bullet\langle Q(x)\rangle \quad \text{and} \quad z : \mathbf{Int} \to \bullet\langle R(x, z)\rangle,$$

respectively. The refinement type system requires that

$$x : \mathbf{Int}; \mathbf{tt} \vdash (z : \mathbf{Int} \to \bullet\langle R(x, z)\rangle) \prec (y : \mathbf{Int} \to \bullet\langle P(x, y)\rangle),$$

from which one obtains a constraint $x : \mathbf{Int}, z : \mathbf{Int}; \mathbf{tt} \models P(x, z) \Rightarrow R(x, z)$, or more simply $\forall x, z. \big[P(x, z) \Longrightarrow R(x, z)\big]$.

Example 4. Recall the formula ψ in Example 1:

$$\psi \quad := \quad \nu X.\, \lambda y.\, y \neq 0 \wedge X\,(y + 1) \quad : \quad \mathbf{Int} \to \bullet.$$

We generate constraints for the sentence $\forall z.\, (z \leq 0) \vee \psi\, z$. The refinement type template for ψ is $y : \mathbf{Int} \to \bullet\langle P(z, y)\rangle$.

The first constraint comes from the subtyping judgement filling the gap between

$$\dfrac{\dfrac{}{z : \mathbf{Int} \vdash z \leq 0 : \bullet\langle z \leq 0\rangle} \quad \dfrac{z : \mathbf{Int} \vdash \psi : y : \mathbf{Int} \to \bullet\langle P(z,y)\rangle}{z : \mathbf{Int} \vdash \psi\, z : \bullet\langle P(z,z)\rangle}}{z : \mathbf{Int} \vdash (z \leq 0) \vee \psi\, z : \bullet\langle (z \leq 0) \vee P(z,z)\rangle}$$

and $z : \mathbf{Int} \vdash (z \leq 0) \vee \psi\, z : \bullet\langle \mathbf{tt}\rangle$. The required subtyping judgement is $z : \mathbf{Int}; \mathbf{tt} \vdash \bullet\langle (z \leq 0) \vee P(z,z)\rangle \prec \bullet\langle \mathbf{tt}\rangle$, from which one obtains

$$\forall z \in \mathcal{D}_{\mathbf{Int}}.\quad \mathbf{tt} \implies z \leq 0 \vee P(z,z).$$

The second constraint comes from the gap between

$$\dfrac{\dfrac{}{\cdots \vdash y \neq 0 : \bullet\langle y \neq 0\rangle} \quad \dfrac{\cdots \vdash X : (y' : \mathbf{Int} \to \bullet\langle P(z,y')\rangle)}{\cdots \vdash X\,(y+1) : \bullet\langle P(z,(y+1))\rangle}}{\begin{array}{l} z : \mathbf{Int}, X : (y' : \mathbf{Int} \to \bullet\langle P(z,y')\rangle), y : \mathbf{Int} \vdash \\ \qquad (y \neq 0 \wedge X\,(y+1)) : \bullet\langle (y \neq 0) \wedge P(z,(y+1))\rangle \end{array}}$$

and the requirement $z : \mathbf{Int}, X : y{:}\mathbf{Int} \to \bullet\langle P(z,y)\rangle, y : \mathbf{Int} \vdash (y \neq 0 \wedge X\,(y+1)) : \bullet\langle P(z,y)\rangle$. The second constraint is

$$\forall y, z \in \mathcal{D}_{\mathbf{Int}}.\quad P(z,y) \implies P(z,y+1).$$

These two constraints are sufficient for the validity of $\forall z.\, (z \leq 0) \vee \psi\, z$. □

Remark 1. The constraint generation procedure is *complete* with respect to the typability: $\vdash \psi : \bullet\langle \mathbf{tt}\rangle$ is derivable for the input sentence if and only if the generated constraints are satisfiable. However it is not complete with respect to the validity since the refinement type system is not complete with respect to the validity. □

5.2 Shape of generated constraints

Constraints obtained by the above procedure are of the from

$$\forall \tilde{x}.\quad P_1(\tilde{x}_1) \wedge \cdots \wedge P_n(\tilde{x}_n) \wedge \theta \implies Q_1(\tilde{y}_1) \vee \cdots \vee Q_m(\tilde{y}_m).$$

Here P_i and Q_j are predicate variables and θ is a constraint formula. If $m \leq 1$, then this is called a *constrained Horn clause* (*CHC* for short). Following [11], we call the general form pCSP. We invoke external solvers such as Spacer [8], HoIce [3] and PCSat [11] to solve the satisfiability of generated constraints.

PCSat [11] accepts the constraints of the above form, so it can be used as a backend solver of the type inference. However PCSat is immature at present compared with CHC solvers, some of which are quite efficient. By this reason, we use CHC solvers such as Spacer [8] and HoIce [3] as the backend solver if the constraints are CHCs.

It is natural to ask when generated constraints are CHCs. We give a convenient sufficient condition on input $\nu\mathrm{HFL}_{\mathbb{Z}}$ formulas. We say a formula is *tractable*

if for every occurrence of disjunctions $(\psi_1 \vee \psi_2)$, at least one of ψ_1 and ψ_2 is an atomic formula. For example, $((F\,x) \wedge (G\,y)) \vee (b = 2)$ is tractable because $b = 2$ is atomic, and $((F\,x) \wedge (b = 2)) \vee (G\,y)$ is not. If the input formula is tractable, the constraint generation algorithm generates CHCs.

In the context of program verification, the safety property verification of higher-order programs are reducible to the validity problem of tractable formulas. In fact, the reduction given in [7] satisfies this condition. Therefore the translation in [7] followed by our type-based validity checking reduces the safety property verification to CHCs, for which efficient solvers are available.

6 Implementation and Experiments

6.1 Implementation

We have implemented a $\nu\text{HFL}_{\mathbb{Z}}$ validity checker ReTHFL based on the inference on the proposed refinement type system. ReTHFL uses, as its backend, CHC solvers HoIce [3] and Spacer [8], and pCSP solver PCSat [11]. In the experiments reported below, unless explicitly mentioned, HoIce is used as the backend solver. We have also implemented a functionality to disprove the validity when a given formula is untypable, as discussed below. For this functionality, Eldarica [4] is used to obtain a resolution proof of the unsatisfiability of CHC.

A method to disprove the validity of a $\nu\text{HFL}_{\mathbb{Z}}$ formula. Since our reduction from the typability of a $\nu\text{HFL}_{\mathbb{Z}}$ formula ψ to the satisfiability of CHC or pCSP is complete, we can conclude that ψ is untypable if the CHC or pCSP obtained by the reduction is unsatisfiable. That does not imply, however, that the original formula ψ is invalid, due to the incompleteness of the type system. Therefore, when a CHC solver returns "unsat", we try to disprove the validity of the original formula. To this end, we first use Eldarica [4] to obtain a resolution proof of the unsatisfiability of CHC, and estimate how many times each fixpoint formula should be unfolded to disprove the validity of the $\nu\text{HFL}_{\mathbb{Z}}$ formula. Below we briefly explain this idea through an example.

Example 5. Let us consider the following formula:

$$\forall n.n < 0 \vee (\nu X.(\lambda y.y = 1 \vee (y \geq 1 \wedge X\ (y-1))))\ n.$$

By preparing a refinement type template $y : \mathbf{Int} \to \bullet\langle P_X(y)\rangle$ for X, we obtain the following constraints:

$$\forall x \in \mathcal{D}_{\mathbf{Int}}.\ \mathbf{tt} \Rightarrow P_X(x) \vee x < 0$$
$$\forall x \in \mathcal{D}_{\mathbf{Int}}.\ P_X(x) \Rightarrow x = 1 \vee (x \geq 1 \wedge P_X(x-1)),$$

which correspond to the CHC:

$$\forall x \in \mathcal{D}_{\mathbf{Int}}.\ x \geq 0 \Rightarrow P_X(x) \qquad \forall x \in \mathcal{D}_{\mathbf{Int}}.\ P_X(x) \wedge x \neq 1 \wedge x < 1 \Rightarrow \mathbf{ff}$$
$$\forall x \in \mathcal{D}_{\mathbf{Int}}.\ P_X(x) \wedge x \neq 1 \Rightarrow P_X(x-1)$$

This set of CHC is unsatisfiable, having the following resolution proof:

$$\frac{0 \geq 0 \Rightarrow P_X(0) \quad P_X(0) \wedge 0 \neq 1 \wedge 0 < 1 \Rightarrow \mathbf{ff}}{0 \geq 0 \wedge 0 \neq 1 \wedge 0 < 1 \Rightarrow \mathbf{ff}\ (= \mathbf{ff})}$$

Here, the two leaves of the proof have been obtained from the first two clauses by instantiating x to 0. Since the second clause is used just once in the proof, we can estimate that a single unfolding of X is sufficient for disproving the validity of the formula. We thus expand the fixpoint formula for X once and check whether the following resulting formula holds by using an SMT solver:

$$\forall n.n < 0 \vee (n = 1 \vee (n \geq 1 \wedge \mathbf{tt})).$$

The SMT solver returns 'No' in this case; hence we can conclude that the original $\nu\text{HFL}_{\mathbb{Z}}$ formula is invalid.

6.2 Experiments

We have conducted experiments to compare ReTHFL with:

- Horus [2]: a HoCHC solver based on refinement type inference [2].
- PaHFL [5]: a $\nu\text{HFL}_{\mathbb{Z}}$ validity checker [5] based on HFL model checking and predicate abstraction.

The experiments were conducted on a Linux server with Intel Xeon CPU E5-2680 v3 and 64 GB of RAM. We set the timeout as 180 seconds in all the experiments below.

Comparison with Horus [2]. We prepared two sets of benchmarks A and B. Both benchmark sets A and B consist of $\nu\text{HFL}_{\mathbb{Z}}$ validity checking problems and the corresponding HoCHC problems. Benchmark set A comes from the HoCHC benchmark for Horus [2], and we prepared $\nu\text{HFL}_{\mathbb{Z}}$ versions based on the correspondence between HoCHC and $\nu\text{HFL}_{\mathbb{Z}}$ discussed in Section 4. Benchmark set B has been obtained from safety verification problems for OCaml programs. Benchmark set A has 8 instances, and benchmark set B has 56 instances. In the experiments, we used Spacer as the common backend CHC solver of ReTHFL and Horus.

The result is shown in Fig. 4. In the figure, "Unknown" means that Horus returned "unsat", which implies that it is unknown whether the program is safe, due to the incompleteness of the underlying refinement type system. ReTHFL could solve 8 instances correctly for benchmark set A, and 46 instances for benchmark set B. In contrast, Horus could solve 7 instances correctly for benchmark set A, and only 18 instances for benchmark set B; as already discussed, this is mainly due to the difference of the subtyping relations of the underlying type systems. The running times were comparable for the instances solved by both ReTHFL and Horus,

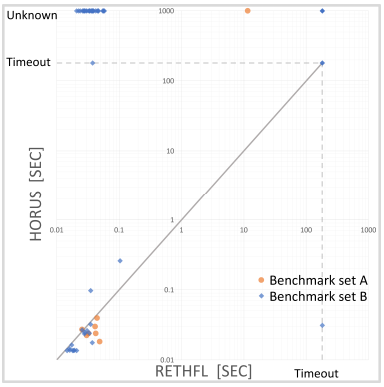

Fig. 4. Comparison with Horus [2].

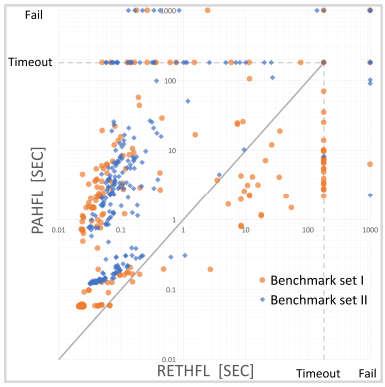

Fig. 5. Comparison with PaHFL [5].

Comparison with PaHFL [5]. We used two benchmark sets I and II. Benchmark set I is the benchmark set of PaHFL [5] consisting of $\nu\text{HFL}_{\mathbb{Z}}$ validity checking problems, which have been obtained from the safety property verification problems for OCaml programs [12]. Since the translation used to obtain $\nu\text{HFL}_{\mathbb{Z}}$ formulas is tailor-made for and works favorably for PaHFL, we also used benchmark set II, which consists of the original program verification problems [12]; for this benchmark set, ReTHFL and PaHFL use their own translations to $\nu\text{HFL}_{\mathbb{Z}}$ formulas.

The results of the two experiments are shown in Fig. 5. In the figure, "Fail" means that the tool terminated abnormally, due to a problem of the backend solvers, or a limitation of our current translator from OCaml programs to $\nu\text{HFL}_{\mathbb{Z}}$ formulas. For benchmark set I, ReTHFL and PaHFL solved 205 and 217 instances respectively. For benchmark set II, ReTHFL and PaHFL solved 247 and 217 instances respectively. Thus, both systems are comparable in terms of the number of solved instances. As for the running times, our solver outperformed PaHFL for most of the instances.

We also compared our solver with PaHFL by using 10 problems reduced from higher-order non-termination problems, which were used in [9]. While PaHFL could solve 4 instances, our solver could not solve any of them in 180 seconds. This is mainly due to the bottleneck of the underlying pCSP solver; developing a better pCSP solver is left for future work.

7 Related work

Burn et al. [2] introduced a higher-order extension of CHC (HoCHC) and proposed a refinement type system for proving the satisfiability of HoCHC. As already discussed in Section 4, the HoCHC satisfiability problem is essentially equivalent to the $\nu\text{HFL}_{\mathbb{Z}}$ validity problem. Our type system is more expressive than Burn et al.'s type system due to more sophisticated subtyping rules. We

have confirmed through experiments that our $\nu\text{HFL}_{\mathbb{Z}}$ solver ReTHFL outperforms their HoCHC solver Horus in terms of the number of solved instances.

Iwayama et al. [5] have recently proposed an alternative approach to $\nu\text{HFL}_{\mathbb{Z}}$ validity checking, which is based on a combination of (pure) HFL model checking, predicate abstraction, and counterexample guided abstraction refinement. In theory, their method is more powerful than ours, since theirs can be viewed as a method for inferring refinement *intersection* types. In practice, however, their solver PaHFL is often slower and times out for some of the instances which ReTHFL can solve. Thus, both approaches can be considered complementary.

Kobayashi et al. [6] have shown that a validity checker for a first-order fixpoint logic can be constructed on top of the validity checker for the ν-only fragment of the first-order logic. We expect that the same technique can be used to construct a validity checker for full $\text{HFL}_{\mathbb{Z}}$ on top of our $\nu\text{HFL}_{\mathbb{Z}}$ validity checker ReTHFL.

There are other refinement type-based approach to program verification, such as Liquid types [10,14] and F* [13]. They are not fully automated in the sense that users must provide either refinement type annotations or qualifiers [10] as hints for verification, while our method is fully automatic. Also, our $\nu\text{HFL}_{\mathbb{Z}}$-based verification method can deal with (un)reachability in the presence of both demonic and angelic branches, while most of the type-based verification methods including those mentioned above can deal with reachability in the presence of only demonic branches.

8 Conclusion

We have proposed a refinement type system for $\nu\text{HFL}_{\mathbb{Z}}$ validity checking, and developed an automated procedure for refinement type inference. Our refinement type system is more expressive than the system by Burn et al. [2] thanks to the refined subtyping relation, which is sound and relative complete with respect to the semantic subtyping relation. We have confirmed the effectiveness of our approach through experiments. Future work includes an improvement of the backend pCSP solver (which is the current main bottleneck of our approach), and an extension of the method to deal with full $\text{HFL}_{\mathbb{Z}}$, based on the method for the first-order case [6].

Acknowledgments

We would like to thank anonymous referees for useful comments. This work was supported by JSPS Kakenhi JP15H05706, JP20H00577, and JP20H05703.

References

1. Bjørner, N., Gurfinkel, A., McMillan, K.L., Rybalchenko, A.: Horn clause solvers for program verification. In: Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday. LNCS, vol. 9300, pp. 24–51. Springer (2015). https://doi.org/10.1007/978-3-319-23534-9_2

2. Burn, T.C., Ong, C.L., Ramsay, S.J.: Higher-order constrained horn clauses for verification. Proc. ACM Program. Lang. **2**(POPL), 11:1–11:28 (2018). https://doi.org/10.1145/3158099
3. Champion, A., Chiba, T., Kobayashi, N., Sato, R.: Ice-based refinement type discovery for higher-order functional programs. In: Proceedings of TACAS 2018. LNCS, vol. 10805, pp. 365–384. Springer (2018). https://doi.org/10.1007/978-3-319-89960-2_20
4. Hojjat, H., Rümmer, P.: The ELDARICA horn solver. In: Proceedings of FMCAD 2018. pp. 1–7. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8603013
5. Iwayama, N., Kobayashi, N., Tsukada, T.: Predicate abstraction and CEGAR for nu-HFLZ validity checking (2020), draft
6. Kobayashi, N., Nishikawa, T., Igarashi, A., Unno, H.: Temporal verification of programs via first-order fixpoint logic. In: Proceedings of SAS 2019. pp. 413–436 (2019). https://doi.org/10.1007/978-3-030-32304-2_20
7. Kobayashi, N., Tsukada, T., Watanabe, K.: Higher-order program verification via HFL model checking. In: Proceedings of ESOP 2018. LNCS, vol. 10801, pp. 711–738. Springer (2018). https://doi.org/10.1007/978-3-319-89884-1_25
8. Komuravelli, A., Gurfinkel, A., Chaki, S.: Smt-based model checking for recursive programs. Formal Methods in System Design **48**(3), 175–205 (2016). https://doi.org/10.1007/s10703-016-0249-4
9. Kuwahara, T., Sato, R., Unno, H., Kobayashi, N.: Predicate abstraction and CEGAR for disproving termination of higher-order functional programs. In: Proceedings of CAV 2015. LNCS, vol. 8410, pp. 287–303. Springer (2015). https://doi.org/10.1007/978-3-319-21668-3_17
10. Rondon, P.M., Kawaguchi, M., Jhala, R.: Liquid types. In: Gupta, R., Amarasinghe, S.P. (eds.) Proceedings of the PLDI 2008. pp. 159–169. ACM (2008). https://doi.org/10.1145/1375581.1375602
11. Satake, Y., Unno, H., Yanagi, H.: Probabilistic inference for predicate constraint satisfaction. Proceedings of the AAAI **34**, 1644–1651 (04 2020). https://doi.org/10.1609/aaai.v34i02.5526
12. Sato, R., Iwayama, N., Kobayashi, N.: Combining higher-order model checking with refinement type inference. In: Proceedings of PEPM 2019. pp. 47–53 (2019). https://doi.org/10.1145/3294032.3294081
13. Swamy, N., Hritcu, C., Keller, C., Rastogi, A., Delignat-Lavaud, A., Forest, S., Bhargavan, K., Fournet, C., Strub, P.Y., Kohlweiss, M., Zinzindohoué, J.K., Zanella-Béguelin, S.: Dependent types and multi-monadic effects in F*. In: 43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL). pp. 256–270. ACM (Jan 2016), https://www.fstar-lang.org/papers/mumon/
14. Vazou, N., Seidel, E.L., Jhala, R., Vytiniotis, D., Jones, S.L.P.: Refinement types for haskell. In: Jeuring, J., Chakravarty, M.M.T. (eds.) Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014. pp. 269–282. ACM (2014). https://doi.org/10.1145/2628136.2628161, https://doi.org/10.1145/2628136.2628161
15. Viswanathan, M., Viswanathan, R.: A higher order modal fixed point logic. In: Proceedings of CONCUR 2004. LNCS, vol. 3170, pp. 512–528. Springer (2004). https://doi.org/10.1007/978-3-540-28644-8_33
16. Watanabe, K., Tsukada, T., Oshikawa, H., Kobayashi, N.: Reduction from branching-time property verification of higher-order programs to

HFL validity checking. In: Proceedings of PEPM 19. pp. 22–34 (2019). https://doi.org/10.1145/3294032.3294077

Appendix

A Simple type system for $\nu\text{HFL}_{\mathbb{Z}}$

$$\frac{}{\Gamma \vdash_H n : \mathbf{Int}} \qquad \frac{\Gamma \vdash_H \psi_i : \bullet \text{ for each } i \in \{1,2\}}{\Gamma \vdash_H \psi_1 \vee \psi_2 : \bullet}$$

$$\frac{\Gamma \vdash_H \psi_i : \mathbf{Int} \text{ for each } i \in \{1,2\}}{\Gamma \vdash_H \psi_1 \,\mathbf{op}\, \psi_2 : \mathbf{Int}} \qquad \frac{\Gamma \vdash_H \psi_i : \bullet \text{ for each } i \in \{1,2\}}{\Gamma \vdash_H \psi_1 \wedge \psi_2 : \bullet}$$

$$\frac{}{\Gamma \vdash_H \mathbf{tt} : \bullet} \qquad \frac{\Gamma, X : \rho \vdash_H \psi : \rho}{\Gamma \vdash_H \nu X : \rho.\psi : \rho}$$

$$\frac{}{\Gamma \vdash_H \mathbf{ff} : \bullet} \qquad \frac{\Gamma, X : \eta \vdash_H \psi : \rho}{\Gamma \vdash_H \lambda X : \eta.\psi : \eta \to \rho}$$

$$\frac{\mathbf{Arity}(p) = k \qquad \Gamma \vdash_H \psi_i : \mathbf{Int} \text{ for each } i \in \{1, \cdots, k\}}{\Gamma \vdash_H \mathbf{p}(\psi_1, \cdots, \psi_k) : \bullet} \qquad \frac{\Gamma \vdash_H \psi_1 : \eta \to \rho \qquad \Gamma \vdash_H \psi_2 : \eta}{\Gamma \vdash_H \psi_1\, \psi_2 : \rho}$$

$$\frac{}{\Gamma, X : \eta \vdash_H X : \eta} \qquad \frac{\Gamma, X : \mathbf{Int} \vdash_H \psi : \bullet}{\Gamma \vdash_H \forall X : \mathbf{Int}.\psi : \bullet}$$

Fig. 6. Simple type judgement of $\nu\text{HFL}_{\mathbb{Z}}$

B Semantics of $\nu\text{HFL}_{\mathbb{Z}}$ formulas

The semantics of a well-typed $\nu\text{HFL}_{\mathbb{Z}}$ formula $[\![\Gamma \vdash_H \psi : \rho]\!]$ is defined as a map from $[\![\Gamma]\!]$ to $\mathcal{D}_\rho$ by induction as shown in Fig. 7. Note that $(\mathcal{D}_\rho, \sqsubseteq_\rho)$ forms a complete lattice; therefore, we can define that $\sqcup_\rho$ and $\sqcap_\rho$ are respectively the least upper bound and greatest lower bound with respect to $\sqsubseteq_\rho$.

Note that $[\![\mathbf{op}]\!]$ is the function over integers denoted by $\mathbf{op}$. Also, $[\![\mathbf{p}]\!]$ is the k-ary relation on integers denoted by $\mathbf{p}$.

The greatest fixpoint operators $\mathbf{gfp}_\rho$ are defined as follows.

$$\mathbf{gfp}_\rho(f) = \bigsqcup_\rho \{x \in \mathcal{D}_\rho \mid x \sqsubseteq_\rho f(x)\}$$

C Proof of Minor Lemmas

C.1 Well-definedness of the minimum element

One has to check that $\gamma_{\tau_1 \to \tau_2}(\alpha)$ is monotone. We first prove an auxiliary lemma.

$$
\begin{array}{l}
[\![\Gamma \vdash_H n : \mathbf{Int}]\!](\alpha) = n \quad [\![\Gamma \vdash_H \mathbf{tt} : \bullet]\!](\alpha) = \top \quad [\![\Gamma \vdash_H \mathbf{ff} : \bullet]\!](\alpha) = \bot \\
[\![\Gamma \vdash_H \psi_1 \ \mathbf{op} \ \psi_2 : \mathbf{Int}]\!](\alpha) = ([\![\Gamma \vdash_H \psi_1 : \mathbf{Int}]\!](\alpha))[\![\mathbf{op}]\!]([\![\Gamma \vdash_H \psi_2 : \mathbf{Int}]\!](\alpha)) \\
[\![\Gamma \vdash_H \mathbf{p}(\psi_1, \cdots, \psi_k) : \bullet]\!](\alpha) = \\
\begin{cases} \top \text{ if } ([\![\Gamma \vdash_H \psi_1 : \mathbf{Int}]\!](\alpha), \cdots, [\![\Gamma \vdash_H \psi_k : \mathbf{Int}]\!](\alpha)) \in [\![\mathbf{p}]\!] \\ \bot \text{ otherwise} \end{cases} \\
[\![\Gamma, X : \eta \vdash_H X : \eta]\!](\alpha) = \alpha(X) \\
[\![\Gamma \vdash_H \psi_1 \vee \psi_2 : \bullet]\!](\alpha) = [\![\Gamma \vdash_H \psi_1 : \bullet]\!](\alpha) \sqcup_\bullet [\![\Gamma \vdash_H \psi_2 : \bullet]\!](\alpha) \\
[\![\Gamma \vdash_H \psi_1 \wedge \psi_2 : \bullet]\!](\alpha) = [\![\Gamma \vdash_H \psi_1 : \bullet]\!](\alpha) \sqcap_\bullet [\![\Gamma \vdash_H \psi_2 : \bullet]\!](\alpha) \\
[\![\Gamma \vdash_H \nu X^\rho.\psi : \rho]\!](\alpha) = \mathbf{gfp}_\rho([\![\Gamma \vdash_H \lambda X : \rho.\psi : \rho \to \rho]\!](\alpha)) \\
[\![\Gamma \vdash_H \lambda X : \eta.\psi : \eta \to \rho]\!](\alpha) = \{(v, [\![\Gamma, X : \eta \vdash_H \psi : \rho]\!](\alpha[X \mapsto v])) \mid v \in \mathcal{D}_\eta\} \\
[\![\Gamma \vdash_H \psi_1 \ \psi_2 : \rho]\!](\alpha) = [\![\Gamma \vdash_H \psi_1 : \eta \to \rho]\!](\alpha)([\![\Gamma \vdash_H \psi_2 : \eta]\!](\alpha)) \\
[\![\Gamma \vdash_H \forall X^\eta.\psi : \bullet]\!](\alpha) = \bigsqcap_\bullet \{[\![\Gamma, X : \eta \vdash_H \psi : \bullet]\!](\alpha[X \mapsto v]) \mid v \in \mathcal{D}_\eta\}
\end{array}
$$

Fig. 7. Semantics of $\nu\mathrm{HFL}_{\mathbb{Z}}$

Lemma 3. *Let $\Gamma \vdash \tau :: \rho$ be a refinement type and α be a valuation in $[\![\Gamma]\!]$. For any $x, y \in \mathcal{D}_\rho$, if $x \sqsubseteq_\rho y$ and $x \in (\!|\tau|\!)(\alpha)$, then $y \in (\!|\tau|\!)(\alpha)$.*

Proof. By induction on the structure of the refinement type $\Gamma \vdash \tau :: \rho$.

- Case $\Gamma \vdash \bullet\langle\theta\rangle :: \bullet$:
  If $y = \top$, then $y \in (\!|\tau|\!)(\alpha)$ for every τ and α (provided that τ refines $\bullet$). Otherwise, $x = y = \bot$ and thus $y \in (\!|\tau|\!)(\alpha)$ follows from the assumption $x \in (\!|\tau|\!)(\alpha)$.
- Case $\Gamma \vdash z : \mathbf{Int} \to \tau :: \mathbf{Int} \to \rho$:
  It suffices to show that $y(v) \in (\!|\tau|\!)(\alpha[z \mapsto v])$ for every $v \in \mathcal{D}_{\mathbf{Int}}$. Let $v \in \mathcal{D}_{\mathbf{Int}}$. Then (a) $x(v) \in (\!|\tau|\!)(\alpha[z \mapsto v])$ from the assumption, and (b) $x(v) \sqsubseteq_\rho y(v)$ from $x \sqsubseteq_{\mathbf{Int}\to\rho} y$. So by the induction hypothesis, we have $y(v) \in (\!|\tau|\!)(\alpha[z \mapsto v])$.
- Case $\Gamma \vdash \tau_1 \to \tau_2 :: \rho_1 \to \rho_2$:
  Similar to the previous case.

□

The monotonicity of $\gamma_{\tau_1\to\tau_2}(\alpha)$ is a consequence of this lemma. Assume that $\tau_1 \to \tau_2 :: \rho_1 \to \rho_2$. Let x and y be elements of $\mathcal{D}_{\rho_1}$ and assume that $x \sqsubseteq y$. If $x \in [\![\tau_1]\!](\alpha)$, then $y \in [\![\tau_1]\!](\alpha)$ by the previous lemma. Then

$$\gamma_{\tau_1\to\tau_2}(\alpha)(x) \quad = \quad \gamma_{\tau_2}(\alpha) \quad = \quad \gamma_{\tau_1\to\tau_2}(\alpha)(y).$$

If $x \notin [\![\tau_1]\!](\alpha)$, then

$$\gamma_{\tau_1\to\tau_2}(\alpha)(x) \quad = \quad \bot_{\rho_2} \quad \sqsubseteq \quad \gamma_{\tau_1\to\tau_2}(\alpha)(y).$$

C.2 Proof for Lemma 1

We prove the claim by induction on ρ. Assume $\Gamma \vdash \tau :: \rho$ and $\alpha \in [\![\Gamma]\!]$.

- Case $\bullet$:
  Then $\tau = \bullet\langle\theta\rangle$. If $\alpha \models \theta$, then $\gamma_\tau(\alpha) = \top$ and $(\!|\tau|\!)(\alpha) = \{\top\}$. If $\alpha \not\models \theta$, then $\gamma_\tau(\alpha) = \bot$ and $(\!|\tau|\!)(\alpha) = \{\bot, \top\}$.
- Case $\mathbf{Int} \to \rho_1$:
  Then $\tau = x : \mathbf{Int} \to \tau_1$. Let $v \in \mathcal{D}_{\mathbf{Int}\to\rho_1}$. Then, by using the induction hypothesis,

$$\begin{aligned}
\gamma_{x:\mathbf{Int}\to\tau_1}(\alpha) \sqsubseteq v \quad &\iff \quad \forall n \in \mathcal{D}_{\mathbf{Int}}.\, \gamma_{x:\mathbf{Int}\to\tau_1}(\alpha)(n) \sqsubseteq v(n) \\
&\iff \quad \forall n \in \mathcal{D}_{\mathbf{Int}}.\, \gamma_{\tau_1}(\alpha[x \mapsto n]) \sqsubseteq v(n) \\
&\iff \quad \forall n \in \mathcal{D}_{\mathbf{Int}}.\, v(n) \in (\!|\tau_1|\!)(\alpha[x \mapsto n]) \\
&\iff \quad v \in (\!|x : \mathbf{Int} \to \tau_1|\!)(\alpha).
\end{aligned}$$

- Case $\rho_1 \to \rho_2$:
  Then $\tau = \tau_1 \to \tau_2$. Let $v \in \mathcal{D}_{\rho_1\to\rho_2}$. Then, by using the induction hypothesis,

$$\begin{aligned}
\gamma_{\tau_1\to\tau_2}(\alpha) \sqsubseteq v \quad &\iff \quad \forall w \in \mathcal{D}_{\rho_1}.\, \gamma_{\tau_1\to\tau_2}(\alpha)(w) \sqsubseteq v(w) \\
&\iff \quad \forall w \in (\!|\tau_1|\!)(\alpha).\, \gamma_{\tau_2}(\alpha) \sqsubseteq v(w) \\
&\iff \quad \forall w \in (\!|\tau_1|\!)(\alpha).\, v(w) \in (\!|\tau_2|\!)(\alpha) \\
&\iff \quad v \in (\!|\tau_1 \to \tau_2|\!)(\alpha).
\end{aligned}$$

C.3 Proof of Lemma 2

By induction on the structure of τ.

- Case $\tau = \bullet\langle\theta\rangle$:
  Trivial.
- Case $\tau = x : \mathbf{Int} \to \tau'$:
  Then $\rho = \mathbf{Int} \to \rho'$ for some ρ'. Since $\mathbf{rty}(x : \mathbf{Int} \to \tau') = \exists x.\mathbf{rty}(\tau')$, the assumption is $\alpha \not\models \exists x.\mathbf{rty}(\tau')$, which is equivalent to $\alpha \models \forall x.\neg\mathbf{rty}(\tau')$ and to
  $$\forall v \in \mathcal{D}_{\mathbf{Int}}.\, \left[\, \alpha[x \mapsto v] \not\models \mathbf{rty}(\tau') \,\right].$$
  Let f be an arbitrary element in $\mathcal{D}_{\mathbf{Int}\to\rho'}$. By definition, $f \in (\!|x{:}\mathbf{Int} \to \tau'|\!)(\alpha)$ if $f(v) \in (\!|\tau'|\!)(\alpha[x \mapsto v])$ for every $v \in \mathcal{D}_{\mathbf{Int}}$. So let v be an arbitrary element in $v \in \mathcal{D}_{\mathbf{Int}}$. By the above proposition and the induction hypothesis, we have $(\!|\tau'|\!)(\alpha[x \mapsto v]) = \mathcal{D}_{\rho'}$. In particular, $f(v) \in \mathcal{D}_{\rho'}$. Since v is arbitrary, we obtain $f \in \mathcal{D}_{x:\mathbf{Int}\to\tau'}(\alpha)$.
- Case $\tau = \tau_1 \to \tau_2$:
  Then $\rho = \rho_1 \to \rho_2$ and $\mathbf{rty}(\tau_1 \to \tau_2) = \mathbf{rty}(\tau_2)$. So the assumption is equivalent to $\alpha \not\models \mathbf{rty}(\tau_2)$. By the induction hypothesis, we have $(\!|\tau_2|\!)(\alpha) = \mathcal{D}_{\rho_2}$. Let f be an arbitrary element in $\mathcal{D}_{\rho_1\to\rho_2}$. By definition, $f \in (\!|\tau_1 \to \tau_2|\!)(\alpha)$ if $f(v) \in (\!|\tau_2|\!)(\alpha)$ for every $v \in \mathcal{D}_{\rho_1}$. This follows from the induction hypothesis $(\!|\tau_2|\!)(\alpha) = \mathcal{D}_{\rho_2}$.

D Soundness (Theorem 1)

We prove the soundness of the subtyping rules.

Lemma 4. *If* $\Delta;\Theta \vdash \tau \prec_\rho \tau'$*, then* $\Delta;\Theta \models \tau \prec_\rho \tau'$.

Proof. By induction on the the derivation $\Delta;\Theta \vdash \tau \prec_\rho \tau'$. We appeal to the case analysis on the shape of the judgement $\Delta;\Theta \vdash \tau \prec_\rho \tau'$.

- Case $\Delta;\Theta \vdash \bullet\langle\theta\rangle \prec_\bullet \bullet\langle\theta'\rangle$:
  Since $\Delta;\Theta \vdash \bullet\langle\theta\rangle \prec_\bullet \bullet\langle\theta'\rangle$ is derivable, we have
  $$\Delta \models \Theta \wedge \theta' \Rightarrow \theta.$$
  Let α be a valuation in $[\![\Delta;\Theta]\!]$. We show that $(\!|\bullet\langle\theta\rangle|\!)(\alpha) \subseteq (\!|\bullet\langle\theta'\rangle|\!)(\alpha)$.
  - If $\alpha \models \theta'$, then $\alpha \models \theta$ by the assumptions. In this case, we have $(\!|\bullet\langle\theta\rangle|\!)(\alpha) = (\!|\bullet\langle\theta'\rangle|\!)(\alpha) = \{\top\}$.
  - If $\alpha \not\models \theta'$, then $(\!|\bullet\langle\theta'\rangle|\!)(\alpha) = \{\bot,\top\} = \mathcal{D}_\bullet$. Hence $(\!|\bullet\langle\theta\rangle|\!)(\alpha) \subseteq (\!|\bullet\langle\theta'\rangle|\!)(\alpha)$ holds.
- Case $\Delta;\Theta \vdash x:\mathbf{Int} \to \tau \prec_{\mathbf{Int}\to\rho} x:\mathbf{Int} \to \tau'$:
  The premise of the derivation is $\Gamma, x:\mathbf{Int};\Theta \vdash \tau \prec_\rho \tau'$. Assume that $\alpha \in [\![\Delta;\Theta]\!]$ and $f \in (\!|x{:}\mathbf{Int} \to \tau|\!)(\alpha)$. By definition, $f(v) \in (\!|\tau|\!)(\alpha[x \mapsto v])$ for every $v \in \mathcal{D}_{\mathbf{Int}}$. Since $(\!|\tau|\!)(\alpha[x \mapsto v]) \subseteq (\!|\tau'|\!)(\alpha[x \mapsto v])$ from the induction hypothesis, we have $f(v) \in (\!|\tau'|\!)(\alpha[x \mapsto v])$ for every $v \in \mathcal{D}_{\mathbf{Int}}$. Therefore $f \in (\!|x:\mathbf{Int} \to \tau'|\!)(\alpha)$.
- Case $\Delta;\Theta \vdash \tau_1 \to \tau_2 \prec_{\rho_1\to\rho_2} \tau_1' \to \tau_2'$:
  The premises of the derivation are
  $$\Gamma;\Theta \wedge \mathbf{rty}(\tau_2') \vdash \tau_1' \prec_{\rho_1} \tau_1$$
  and
  $$\Gamma;\Theta \vdash \tau_2 \prec_{\rho_2} \tau_2'.$$
  Assume $\alpha \in [\![\Delta;\Theta]\!]$, $f \in (\!|\tau_1 \to \tau_2|\!)(\alpha)$ and $v \in (\!|\tau_1'|\!)(\alpha)$. It suffices to show that $f(v) \in (\!|\tau_2'|\!)(\alpha)$.
  - Assume that $\alpha \models \mathbf{rty}(\tau_2')$. Then $\alpha \in [\![\Delta;\Theta \wedge \mathbf{rty}(\tau_2')]\!]$, and thus $(\!|\tau_1'|\!)(\alpha) \subseteq (\!|\tau_1|\!)(\alpha)$ by the induction hypothesis. So the assumption $v \in (\!|\tau_1'|\!)(\alpha)$ implies $v \in (\!|\tau_1|\!)(\alpha)$. Since $f \in (\!|\tau_1 \to \tau_2|\!)(\alpha)$, we have $f(v) \in (\!|\tau_2|\!)(\alpha)$. Then $f(v) \in (\!|\tau_2'|\!)(\alpha)$ since $(\!|\tau_2|\!)(\alpha) \subseteq (\!|\tau_2'|\!)(\alpha)$ by the induction hypothesis.
  - Assume that $\alpha \not\models \mathbf{rty}(\tau_2')$. Then, by Lemma 2, we have $(\!|\tau_2'|\!)(\alpha) = \mathcal{D}_{\rho_2}$. In particular, $f(v) \in (\!|\tau_2'|\!)(\alpha)$.

□

We then prove the soundness of the refinement type system.

Lemma 5. *If* $\Delta \vdash \psi : \tau$*, then* $\Delta \models \psi : \tau$.

Proof. By induction on the structure of derivation. Assume that $\Delta \vdash \psi : \tau$ and let $\alpha \in [\![\Delta]\!]$.

- Case RAbs:
  Then $\tau = \tau_1 \to \tau_2$ and $\psi = \lambda X.\phi$. We have $\Delta, X : \tau_1 \vdash \phi : \tau_2$ as the premise. Let v be an arbitrary element in $(\!|\tau_1|\!)(\alpha)$. Since $\alpha[X \mapsto v] \in [\![\Delta, X : \tau_1]\!]$, we have $[\![\phi]\!](\alpha[X \mapsto v]) \in (\!|\tau_2|\!)(\alpha[X \mapsto v])$ by the induction hypothesis. Note that $(\!|\tau_2|\!)(\alpha[X \mapsto v]) = (\!|\tau_2|\!)(\alpha)$ because X, which is not of simple type **Int**, does not appear in τ_2. Therefore $v \in (\!|\tau_1|\!)(\alpha)$ implies $[\![\phi]\!](\alpha[x \mapsto v]) \in (\!|\tau_2|\!)(\alpha)$. This means that $[\![\lambda X.\phi]\!](\alpha) \in (\!|\tau_1 \to \tau_2|\!)(\alpha)$ since $[\![\lambda X.\phi]\!](\alpha)(v) = [\![\phi]\!](\alpha[X \mapsto v]) \in (\!|\tau_2|\!)(\alpha)$ for every $v \in (\!|\tau_1|\!)(\alpha)$.
- Case RGfp:
  Then $\psi = \nu X : \tau.\phi$ and we have $\Delta, X : \tau \vdash \phi : \tau$ as the premise. Let $v = \gamma_\tau(\alpha)$ be the minimum element in $(\!|\tau|\!)(\alpha)$ (cf. Lemma 1). Since $v \in (\!|\tau|\!)(\alpha)$, we have $\alpha[X \mapsto v] \in [\![\Delta, X : \tau]\!]$. By the induction hypothesis, $[\![\phi]\!](\alpha[X \mapsto v]) \in (\!|\tau|\!)(\alpha[X \mapsto v])$. Since X does not appear in τ, we have $(\!|\tau|\!)(\alpha[X \mapsto v]) = (\!|\tau|\!)(\alpha)$ and thus $[\![\phi]\!](\alpha[X \mapsto v]) \in (\!|\tau|\!)(\alpha)$. By Lemma 1, we have $v \sqsubseteq [\![\phi]\!](\alpha[X \mapsto v])$.
  Recall that the goal is $[\![\nu X : \tau.\phi]\!](\alpha) \in (\!|\tau|\!)(\alpha)$, which is equivalent to $v \sqsubseteq [\![\nu X : \tau.\phi]\!](\alpha)$. By definition,

$$[\![\nu X : \tau.\phi]\!](\alpha) \quad = \quad \bigsqcup \{\, w \mid w \sqsubseteq [\![\phi]\!](\alpha[X \mapsto w]) \,\}.$$

  We have $v \sqsubseteq [\![\nu X : \tau.\phi]\!](\alpha)$ since v belongs to the set in the right-hand-side.
- Case RSub: Immediate from Lemma 4.

Other cases are easy. □

Proof (Theorem 1). The first claim is Lemma 4 and the second claim is Lemma 5. □

E Completeness of Subtyping Rules (Theorem 2)

We start from auxiliary lemmas. We write $\bot_\rho$ for the minimum element in $\mathcal{D}_\rho$.

Lemma 6. *Let $\Gamma \vdash \tau :: \rho$ be a refinement type and α be a valuation in $[\![\Gamma]\!]$. If $\alpha \models \boldsymbol{rty}(\tau)$, then $\bot_\rho \notin (\!|\tau|\!)(\alpha)$.*

Proof. By induction on the structure of the refinement type τ.

- Case $\bullet\langle\theta\rangle$:
  From the assumption, $\alpha \models \theta$ and thus $(\!|\bullet\langle\theta\rangle|\!)(\alpha) = \{\top\}$.
- Case $(x : \mathbf{Int} \to \tau)$:
  Since $\mathbf{rty}(x : \mathbf{Int} \to \tau) = \exists x.\mathbf{rty}(\tau)$, the assumption is equivalent to $\alpha \models \exists x.\mathbf{rty}(\tau)$. So $\alpha[x \mapsto v] \models \mathbf{rty}(\tau)$ for some v. We fix v that satisfies this condition. Then, by the induction hypothesis, we have $\bot_\rho \notin (\!|\tau|\!)(\alpha[x \mapsto v])$. Assume for contradiction that $\bot_{\mathbf{Int}\to\rho} \in (\!|x{:}\mathbf{Int} \to \tau|\!)(\alpha)$. Then $\bot_{\mathbf{Int}\to\rho}(v) = \bot_\rho \in (\!|\tau|\!)(\alpha[x \mapsto v])$, a contradiction.

– Case $\Gamma \vdash \tau_1 \to \tau_2 :: \rho_1 \to \rho_2$
Since $\mathbf{rty}(\tau_1 \to \tau_2) = \mathbf{rty}(\tau_2)$, the assumption is equivalent to $\alpha \models \mathbf{rty}(\tau_2)$. By the induction hypothesis, $\bot_{\rho_2} \notin (\!|\tau_2|\!)(\alpha)$.
Assume for contradiction that $\bot_{\rho_1 \to \rho_2} \in (\!|\tau_1 \to \tau_2|\!)(\alpha)$. Since $\top_{\rho_1} \in (\!|\tau_1|\!)(\alpha)$,[8] we have $\bot_{\rho_1 \to \rho_2}(\top_{\rho_1}) = \bot_{\rho_2} \in (\!|\tau_2|\!)(\alpha)$, a contradiction.

□

Proof (Theorem 2). Assume that $\Delta; \Theta \models \tau \prec_\rho \tau'$. We prove the claim by induction on ρ.

– Case $\rho = \bullet$:
Then $\tau = \bullet\langle\theta\rangle$ and $\tau' = \bullet\langle\theta'\rangle$. Let $\alpha \in [\![\Delta; \Theta]\!]$. If $\alpha \not\models \theta$, then

$$\{\bot, \top\} \quad \subseteq \quad (\!|\bullet\langle\theta\rangle|\!)(\alpha) \quad \subseteq \quad (\!|\bullet\langle\theta'\rangle|\!)(\alpha),$$

which implies $\alpha \not\models \theta'$ and thus $\alpha \models \theta' \wedge \Theta \Longrightarrow \theta$. If $\alpha \models \theta$, then $\alpha \models \theta' \wedge \Theta \Longrightarrow \theta$ as well.
– Case $\rho = \mathbf{Int} \to \rho_1$:
Then $\tau = (x : \mathbf{Int} \to \tau_1)$ and $\tau' = (x : \mathbf{Int} \to \tau_1')$. Let $\alpha \in [\![\Delta; \Theta]\!]$ and $n \in \mathcal{D}_{\mathbf{Int}}$. We have

$$\gamma_{x:\mathbf{Int}\to\tau_1}(\alpha) \quad \in \quad (\!|x : \mathbf{Int} \to \tau_1|\!)(\alpha) \quad \subseteq \quad (\!|x : \mathbf{Int} \to \tau_1'|\!)(\alpha)$$

by Lemma 1 and the assumption. Hence, for every $n \in \mathcal{D}_{\mathbf{Int}}$,

$$\gamma_{\tau_1}(\alpha[x \mapsto n]) \quad = \quad \gamma_{x:\mathbf{Int}\to\tau_1}(\alpha)(n) \quad \in \quad (\!|\tau_1'|\!)(\alpha[x \mapsto n]).$$

By using Lemmas 1 and 3,

$$\begin{array}{rcl} v \in (\!|\tau_1|\!)(\alpha[x \mapsto n]) & \Longleftrightarrow & \gamma_{\tau_1}(\alpha[x \mapsto n]) \sqsubseteq v \\ & \Longrightarrow & v \in (\!|\tau_1'|\!)(\alpha[x \mapsto n]). \end{array}$$

Since $\alpha \in [\![\Delta; \Theta]\!]$ and $v \in \mathcal{D}_{\mathbf{Int}}$ are arbitrary and x does not appear freely in Θ, the above proposition says that $v \in (\!|\tau_1|\!)(\alpha')$ implies $v \in (\!|\tau_1'|\!)(\alpha')$ for every $\alpha' \in [\![\Delta, x : \mathbf{Int}; \Theta]\!]$. In other words, $\Delta, x : \mathbf{Int}; \Theta \models \tau_1 \prec \tau_1'$. Hence, by the induction hypothesis, $\Delta, x : \mathbf{Int}; \Theta \vdash \tau_1 \prec \tau_1'$, from which $\Delta; \Theta \vdash (x : \mathbf{Int} \to \tau_1) \prec (x : \mathbf{Int} \to \tau_1')$ follows.
– Case $\rho = \rho_1 \to \rho_2$:
In this case $\tau = \tau_1 \to \tau_2$ and $\tau' = \tau_1' \to \tau_2'$. Assume that $\Delta; \Theta \models \tau \prec \tau'$. We prove $\Delta; \Theta \models \tau_2 \prec \tau_2'$ and $\Delta; \Theta \wedge \mathbf{rty}(\tau_2') \models \tau_1' \prec \tau_1$. Then $\Delta; \Theta \vdash \tau \prec \tau'$ follows from the induction hypothesis and S-Fun.
We prove $\Delta; \Theta \models \tau_2 \prec \tau_2'$. Let $\alpha \in [\![\Delta; \Theta]\!]$ and $v \in (\!|\tau_2|\!)(\alpha)$ and define $f \in (\!|\tau_1 \to \tau_2|\!)(\alpha)$ by $f(x) := v$. By the assumption, $f \in (\!|\tau_1' \to \tau_2'|\!)(\alpha)$. Since $\top_{\rho_1} \in (\!|\tau_1'|\!)(\alpha)$, we have $f(\top_{\rho_1}) = v \in (\!|\tau_2'|\!)(\alpha)$. Since $v \in (\!|\tau_2|\!)(\alpha)$ is arbitrary, we obtain $(\!|\tau_2|\!)(\alpha) \subset (\!|\tau_2'|\!)(\alpha)$.

[8] One can easily prove that $\top_\rho \in (\!|\tau|\!)(\alpha)$ if $I \vdash \tau :: \rho$ and $\alpha \in [\![I]\!]$, by induction on the structure of τ.

We prove $\Delta; \Theta \wedge \mathbf{rty}(\tau_2') \models \tau_1' \prec \tau_1$. Assume for contradiction that $\Delta; \Theta \wedge \mathbf{rty}(\tau_2') \not\models \tau_1' \prec \tau_1$. Then, there exist $\alpha \in [\![\Delta; \Theta \wedge \mathbf{rty}(\tau_2')]\!]$ and $g \in (\!|\tau_1'|\!)(\alpha)$ such that $g \notin (\!|\tau_1|\!)(\alpha)$. By Lemma 1, we have the minimal element $\gamma_{\tau_1 \to \tau_2}(\alpha)$ in $(\!|\tau_1 \to \tau_2|\!)(\alpha)$, which belongs to $(\!|\tau_1' \to \tau_2'|\!)(\alpha)$ by the assumption. Since $g \in (\!|\tau_1'|\!)(\alpha)$, we have $\gamma_{\tau_1 \to \tau_2}(\alpha)(g) \in (\!|\tau_2'|\!)(\alpha)$. By Lemma 6, we have $\bot_{\rho_2} \notin (\!|\tau_2'|\!)(\alpha)$ and thus $\gamma_{\tau_1 \to \tau_2}(\alpha)(g) \neq \bot_{\rho_2}$. On the other hand, from the definition of the minimal element $\gamma_{\tau_1 \to \tau_2}(\alpha)$ and the assumption $g \notin (\!|\tau_1|\!)(\alpha)$, we have $\gamma_{\tau_1 \to \tau_2}(\alpha)(g) = \bot_{\rho_2}$, a contradiction.