



Predicate Abstraction and CEGAR for $\nu\text{HFL}_{\mathbb{Z}}$ Validity Checking

Naoki Iwayama¹, Naoki Kobayashi¹, Ryota Suzuki¹, and Takeshi Tsukada²

¹ The University of Tokyo, {iwayama,koba,rsuzuki}@kb.is.s.u-tokyo.ac.jp

² Chiba University, tsukada@math.s.chiba-u.ac.jp

Abstract. We propose an automated method for $\nu\text{HFL}_{\mathbb{Z}}$ validity checking. $\text{HFL}_{\mathbb{Z}}$ is an extension of the higher-order fixpoint logic HFL with integers, and $\nu\text{HFL}_{\mathbb{Z}}$ is a restriction of it to the fragment without the least fixpoint operator. The validity checking problem for $\text{HFL}_{\mathbb{Z}}$ has recently been shown to provide a uniform approach to higher-order program verification. The restriction to $\nu\text{HFL}_{\mathbb{Z}}$ studied in this paper already provides an automated method for a large class of program verification problems including safety and non-termination verification, and also serves as a key building block for solving the validity checking problem for full $\text{HFL}_{\mathbb{Z}}$. Our approach is based on predicate abstraction and counterexample-guided abstraction refinement (CEGAR). We have implemented the proposed method, and applied it to program verification. According to experiments, our tool outperforms a closely related tool called Horus in terms of precision, and is competitive with a more specialized program verification tool called MOCHI despite the generality of our approach.

1 Introduction

$\text{HFL}_{\mathbb{Z}}$ [12] is an extension of the higher-order fixpoint logic HFL [21] with integers. Kobayashi et al. [12,22] have shown that various program verification problems for functional programs can be reduced to $\text{HFL}_{\mathbb{Z}}$ validity checking problems.³ For example, consider the following OCaml program.

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

The main function takes an integer n as an argument, computes the sum $r = \sum_{x=1}^n (x+x)$, and asserts that $r \geq n$ (here, the function `sum` is represented in the continuation-passing style to make the correspondence with the formula below clear). By using the reduction of [12], the property that the assertion never fails

³ Kobayashi et al. [12] actually considered *model* checking problems, but it is actually sufficient to consider validity checking problems for formulas without modal operators, as shown in a follow-up paper [22]; thus, throughout this paper, we shall consider only validity checking for formulas without modal operators.

for any integer n can be expressed by the $\text{HFL}_{\mathbb{Z}}$ formula $\forall n.\text{main } n$, where main is defined by:

$$\begin{aligned} \text{main } n &=_{\nu} \text{sum } (\lambda x.\lambda k.k(x+x)) \ n \ (\lambda r.r \geq n) \\ \text{sum } f \ n \ k &=_{\nu} (n \leq 0 \Rightarrow k \ 0) \wedge (n > 0 \Rightarrow f \ n \ (\lambda x.\text{sum } f \ (n-1) \ \lambda y.k(x+y))). \end{aligned}$$

Here, the subscript ν of each equality symbol indicates that main and sum are the *greatest* predicates that satisfy the equations. Notice that the formulas above (whose precise semantics will be introduced later) directly correspond to the definitions of the `main` and `sum` functions; for example, the part $n \leq 0 \Rightarrow k \ 0$ in the equation for the sum predicate corresponds to the then-part of the `sum` function. Watanabe et al. [22] have shown that verification of arbitrary regular properties (i.e., those expressible in the modal μ -calculus) of simply-typed, higher-order recursive functional programs can be reduced to $\text{HFL}_{\mathbb{Z}}$ validity checking in a similar (but a little more elaborated) manner: more precisely, given a closed program P and a regular property A , one can construct a closed $\text{HFL}_{\mathbb{Z}}$ formula $\varphi_{P,A}$ such that P satisfies A just if $\varphi_{P,A}$ is valid. Thus, an automated $\text{HFL}_{\mathbb{Z}}$ validity checker would yield a very general automated verification tool for higher-order functional programs.

As the first step towards the development of an automated $\text{HFL}_{\mathbb{Z}}$ validity checker, in the present paper, we focus on a fragment of $\text{HFL}_{\mathbb{Z}}$ called $\nu\text{HFL}_{\mathbb{Z}}$, and develop an automated method for validity checking of $\nu\text{HFL}_{\mathbb{Z}}$ formulas (note that our method is sound but necessarily incomplete, as the problem is undecidable). The fragment $\nu\text{HFL}_{\mathbb{Z}}$ is obtained by removing the least fixpoint operator from $\text{HFL}_{\mathbb{Z}}$. A $\nu\text{HFL}_{\mathbb{Z}}$ validity checker can be used for verifying various properties of higher-order functional programs, such as safety and non-termination properties. In fact, the verification of properties expressible in the ν -only fragment of the modal μ -calculus can be reduced to validity checking of a $\nu\text{HFL}_{\mathbb{Z}}$ formula. This fragment is powerful enough to verify the (un)reachability problem in the presence of both angelic and demonic branches; in contrast, most of the previous automated verification tools for higher-order programs (such as `MOCHI` [10]) only deal with demonic branches. A $\nu\text{HFL}_{\mathbb{Z}}$ validity checker can also be used as a building block for a (forthcoming) full $\text{HFL}_{\mathbb{Z}}$ validity checker (which can then be used for verification of arbitrary properties expressive in the full the modal μ -calculus, like “an event A occurs infinitely often”), by using the technique developed for a first-order fixpoint logic [9].

Our method is based on predicate abstraction and counterexample-guided abstraction refinement (CEGAR). The techniques of predicate abstraction and CEGAR have been used in the context of model checking. In this paper, we adapt them for proving the validity of a $\nu\text{HFL}_{\mathbb{Z}}$ formula. Given a $\nu\text{HFL}_{\mathbb{Z}}$ formula φ and a set of predicates on integers, we compute a *pure* HFL formula φ' without integers, as an underapproximation of φ , so that if φ' is valid, so is φ . The validity of the pure HFL formula φ' is decidable; one can use either an HFL model checker (such as `HOMUSAT` [6]) or a `HORS` model checker (such as [7]) based on the reduction from `HORS` to HFL model checking [8]. For example, suppose that we have chosen the predicate $\lambda x.x > 0$ for abstracting integers. Then, the integer

predicate $\lambda x.\lambda y.x + y > 0$ can be abstracted to $\lambda b_{x>0}.\lambda b'_{y>0}.b_{x>0} \wedge b'_{y>0}$, where $b_{x>0}$ ($b'_{y>0}$, resp.) is instantiated to `true` just if the value of the original argument x (y , resp.) is positive. The formula $\lambda b_{x>0}.\lambda b'_{y>0}.b_{x>0} \wedge b'_{y>0}$ semantically represents the predicate $\lambda x.\lambda y.x > 0 \wedge y > 0$, which is an underapproximation of the original predicate $\lambda x.\lambda y.x + y > 0$. As in the ordinary predicate abstraction technique for model checking, the success of validity checking heavily depends on the choice of the predicates used for abstraction; we thus use CEGAR to refine the set of predicates in an on-demand manner, based on counterexamples. Due to the generality of $\nu\text{HFL}_{\mathbb{Z}}$ validity checking (which, as mentioned earlier, can deal with the reachability in the presence of both angelic and demonic branches), we need a more elaborate method for CEGAR than the previous methods for CEGAR for higher-order program verification.

We have implemented an automated $\nu\text{HFL}_{\mathbb{Z}}$ validity checker PAHFL based on the method above, and compared through experiments with two related tools: Horus [2] and MOCHI [10]. Horus is a satisfiability checker for HoCHC, higher-order constrained Horn clauses. As we discuss in Section 2, the validity checking problem for $\nu\text{HFL}_{\mathbb{Z}}$ and the satisfiability problem for HoCHC are reducible to each other; thus Horus can also be used as a $\nu\text{HFL}_{\mathbb{Z}}$ validity checker. As demonstrated through the experiments, however, Horus is not powerful enough to prove the validity of many formulas obtained from higher-order program verification problems, although Horus often terminates quickly when it succeeds. MOCHI [10] is an automated program verification tool for OCaml, developed based on HORS model checking. The original version of MOCHI is tailor-made for (non-)reachability verification, although various extensions for proving and disproving termination and fair termination have been developed later. In contrast, our $\nu\text{HFL}_{\mathbb{Z}}$ validity checker can deal with a wider class of properties than the original version of MOCHI, in a more uniform and general manner than the various extensions of MOCHI mentioned above. According to our experiments, PAHFL is competitive with MOCHI, despite the generality.

The rest of this paper is structured as follows. Section 2 reviews the definition of $\text{HFL}_{\mathbb{Z}}$ and its validity checking problem. Sections 3 and 4 formalize our method. Section 5 reports our implementation and experiments. Section 6 discusses related work and Section 7 concludes the paper.

2 Preliminaries: Higher-Order Fixed-Point Logic $\nu\text{HFL}_{\mathbb{Z}}$

This section reviews (modal-free) $\nu\text{HFL}_{\mathbb{Z}}$ and its *validity checking problem*. The logic $\nu\text{HFL}_{\mathbb{Z}}$ is a higher-order logic with arithmetic (over integers) and greatest fixed-point operators $\nu x.\psi$, hence the name. It is a fragment of $\text{HFL}_{\mathbb{Z}}$ [12], which is an extension of HFL [21] with arithmetic (over integers).⁴

The set of *types*, ranged over by τ , is defined by:

$$\tau \text{ (types)} ::= \bullet \mid \bar{\tau} \rightarrow \tau \quad \bar{\tau} \text{ (extended types)} ::= \tau \mid \mathbf{int}.$$

⁴ It is possible to further extend $\text{HFL}_{\mathbb{Z}}$ with other data structures such as lists and trees, and extend our predicate abstraction method accordingly, as long as the background solvers (such as SMT and CHC solvers) support them.

$$\begin{array}{c}
\frac{}{\Gamma, x : \tau \vdash_{\text{ST}} x : \tau} \text{ (S-VAR)} \quad \frac{b \in \{\mathbf{true}, \mathbf{false}\}}{\Gamma \vdash_{\text{ST}} b : \bullet} \text{ (S-BOOL)} \quad \frac{}{\Gamma \vdash_{\text{ST}} n : \mathbf{int}} \text{ (S-INT)} \\
\frac{\Gamma \vdash_{\text{ST}} a_i : \mathbf{int} \text{ for each } i}{\Gamma \vdash_{\text{ST}} p(a_1, \dots, a_n) : \bullet} \text{ (S-PRED)} \quad \frac{\Gamma \vdash_{\text{ST}} a_i : \mathbf{int} \text{ for each } i}{\Gamma \vdash_{\text{ST}} op(a_1, \dots, a_n) : \mathbf{int}} \text{ (S-OP)} \\
\frac{\Gamma \vdash_{\text{ST}} \psi : \bar{\tau} \rightarrow \tau \quad \Gamma \vdash_{\text{ST}} \bar{\psi} : \bar{\tau}}{\Gamma \vdash_{\text{ST}} \psi \bar{\psi} : \tau} \text{ (S-APP)} \quad \frac{\Gamma, x : \bar{\tau} \vdash_{\text{ST}} \psi : \tau}{\Gamma \vdash_{\text{ST}} \lambda x^{\bar{\tau}}. \psi : \bar{\tau} \rightarrow \tau} \text{ (S-ABS)} \\
\frac{\Gamma \vdash_{\text{ST}} \psi_1 : \bullet \quad \Gamma \vdash_{\text{ST}} \psi_2 : \bullet}{\Gamma \vdash_{\text{ST}} \psi_1 \wedge \psi_2 : \bullet} \text{ (S-AND)} \quad \frac{\Gamma \vdash_{\text{ST}} \psi_1 : \bullet \quad \Gamma \vdash_{\text{ST}} \psi_2 : \bullet}{\Gamma \vdash_{\text{ST}} \psi_1 \vee \psi_2 : \bullet} \text{ (S-OR)} \\
\frac{\Gamma, x : \tau \vdash_{\text{ST}} \psi : \tau}{\Gamma \vdash_{\text{ST}} \nu x^{\tau}. \psi : \tau} \text{ (S-NU)}
\end{array}$$

Fig. 1. Simple typing of $\nu\text{HFL}_{\mathbb{Z}}$

The type \bullet describes propositions. Note that \mathbf{int} can occur only on the lefthand side of \rightarrow ; for example, $\mathbf{int} \rightarrow \mathbf{int}$ is invalid. Every type τ can be written in the form $\bar{\tau}_1 \rightarrow \dots \rightarrow \bar{\tau}_k \rightarrow \bullet$.

The set of $\nu\text{HFL}_{\mathbb{Z}}$ formulas, ranged over ψ , is defined by:

$$\begin{array}{ll}
\psi \text{ (formula)} & ::= x^{\tau} \mid \mathbf{true} \mid \mathbf{false} \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \\
& \quad \mid \nu x^{\tau}. \psi \mid \lambda x^{\bar{\tau}}. \psi \mid \psi \bar{\psi} \mid p(\bar{a}) \\
a \text{ (arithmetic expression)} & ::= n \mid x^{\mathbf{int}} \mid op(\bar{a}) \\
\bar{\psi} \text{ (extended formula)} & ::= \psi \mid a
\end{array}$$

Here, the metavariables x , n , p , and op respectively range over the sets of variables, integers, integer predicates (such as $<$), and integer operators (such as $+$). We often use infix notations for predicates and operators. The formula $\nu x^{\tau}. \psi$ denotes the greatest fixpoint of $\lambda x^{\tau}. \psi$. We often omit the type annotation on the shoulder of a variable.

We use a standard simple type system for the λ -calculus to restrict the shape of formulas. As usual, a *type environment*, denoted by the metavariable Γ , is a finite map from a set of variables to the set of extended types. The typing relation $\Gamma \vdash_{\text{ST}} \bar{\psi} : \bar{\tau}$ is defined by the rules in Figure 1. Henceforth, we consider only well-typed formulas.

The interpretation of a type $\bar{\tau}$ is a poset $(\mathbb{D}_{\bar{\tau}}, \sqsubseteq_{\bar{\tau}})$, inductively defined by:

$$\begin{array}{ll}
\mathbb{D}_{\bullet} := \{\mathbf{t}, \mathbf{ff}\} & \sqsubseteq_{\bullet} := \{(\mathbf{ff}, \mathbf{ff}), (\mathbf{ff}, \mathbf{t}), (\mathbf{t}, \mathbf{t})\} \\
\mathbb{D}_{\mathbf{int}} := \mathbb{Z} & \sqsubseteq_{\mathbf{int}} := \{(n, n) \mid n \in \mathbb{Z}\} \\
\mathbb{D}_{\bar{\tau} \rightarrow \tau} := \mathbb{D}_{\bar{\tau}} \rightarrow \mathbb{D}_{\tau} & \sqsubseteq_{\bar{\tau} \rightarrow \tau} := \{(f, g) \mid \forall v \in \mathbb{D}_{\bar{\tau}}. f(v) \sqsubseteq_{\tau} g(v)\}.
\end{array}$$

Here, $\mathbb{D}_{\bar{\tau}} \rightarrow \mathbb{D}_{\tau}$ denotes the set of *monotone* functions from $\mathbb{D}_{\bar{\tau}}$ to \mathbb{D}_{τ} . Note that $(\mathbb{D}_{\tau}, \sqsubseteq_{\tau})$ is a complete lattice (although $(\mathbb{D}_{\mathbf{int}}, \sqsubseteq_{\mathbf{int}})$ is not). Hence, for every function $f \in \mathbb{D}_{\bar{\tau} \rightarrow \tau}$, there exists a greatest fixpoint. $\text{gfp}(f)$ of f , given by: $\text{gfp}(f) = \bigsqcup \{v \in \mathbb{D}_{\tau} \mid v \sqsubseteq_{\tau} f(v)\}$.

We define the interpretation of formulas. Given a type environment Γ , a *valuation for Γ* is a mapping ρ such that $\rho(x) \in \mathbb{D}_\tau$ for each $(x : \tau) \in \Gamma$. We assume that the interpretations $\llbracket op \rrbracket$ and $\llbracket p \rrbracket$ of operators and atomic predicates are given a priori. For a formula $\Gamma \vdash_{\text{ST}} \psi : \tau$ and a valuation ρ for Γ , the interpretation $\llbracket \psi \rrbracket_\rho$ is defined by:

$$\begin{aligned} \llbracket x \rrbracket_\rho &:= \rho(x) & \llbracket \text{true} \rrbracket_\rho &:= \# & \llbracket \text{false} \rrbracket_\rho &:= \# & \llbracket n \rrbracket_\rho &:= n \\ \llbracket op(a_1, \dots, a_n) \rrbracket_\rho &:= \llbracket op \rrbracket(\llbracket a_1 \rrbracket_\rho, \dots, \llbracket a_n \rrbracket_\rho) \\ \llbracket p(a_1, \dots, a_n) \rrbracket_\rho &:= \llbracket p \rrbracket(\llbracket a_1 \rrbracket_\rho, \dots, \llbracket a_n \rrbracket_\rho) \\ \llbracket \psi \bar{\psi} \rrbracket_\rho &:= \llbracket \psi \rrbracket_\rho(\llbracket \bar{\psi} \rrbracket_\rho) & \llbracket \lambda x^\tau. \psi \rrbracket_\rho &= \{ v \mapsto \llbracket \psi \rrbracket_{\rho \cup \{x \mapsto v\}} \mid v \in \mathbb{D}_{\bar{\tau}} \} \\ \llbracket \nu x^\tau. \psi \rrbracket_\rho &:= \text{gfp}(\llbracket \lambda x^\tau. \psi \rrbracket_\rho). \end{aligned}$$

For a formula ψ of type \bullet , we write $\rho \models \psi$ to mean $\llbracket \psi \rrbracket_\rho = \#$. If $\rho \models \psi$ for every valuation ρ , we write $\models \psi$ and say that ψ is *valid*. The $\nu\text{HFL}_{\mathbb{Z}}$ *validity checking problem* asks if a formula of type \bullet is valid. Since the universal quantifiers are definable (see Examples 3 and 4 below), we can assume without loss of generality that an input of the validity checking problem is a closed formula. The validity checking problem for $\nu\text{HFL}_{\mathbb{Z}}$ is undecidable.

Example 1. Let ψ be $\nu X. \lambda n. (n = 0 \vee (n > 0 \wedge X(n - 2)))$. Then $\models \psi(n)$ holds just if n is a non-negative even number. \square

Example 2. The example in Section 1 is expressed by $\forall n. \text{main } n$, where *main* is defined by:

$$\begin{aligned} \text{main} &:= \lambda n^{\text{int}}. \text{sum} (\lambda x. \lambda k. k(x + x)) n (\lambda r. r \geq n) \\ \text{sum} &:= \nu \text{sum}^\tau. \lambda f^{\text{int} \rightarrow (\text{int} \rightarrow \bullet) \rightarrow \bullet}. \lambda n^{\text{int}}. \lambda k^{\text{int} \rightarrow \bullet}. (n > 0 \vee k 0) \\ &\quad \wedge (n \leq 0 \vee f n (\lambda x^{\text{int}}. \text{sum } f (n - 1) \lambda y^{\text{int}}. k(x + y))) \\ \tau &:= (\text{int} \rightarrow (\text{int} \rightarrow \bullet) \rightarrow \bullet) \rightarrow \text{int} \rightarrow (\text{int} \rightarrow \bullet) \rightarrow \bullet. \end{aligned}$$

Note that the subformula $n > 0 \vee k 0$ is equivalent to $n \leq 0 \Rightarrow k 0$, which in turn corresponds to the then-part of the sum function in the source program. We emphasize again that the formula above mimics the structure of the source program in the continuation passing style, where the answer type of the source program corresponds to the type \bullet of formulas. \square

Example 3. The universal quantifier over integers is definable. Let $\text{forall}_{\text{int}} := \lambda f^{\text{int} \rightarrow \bullet}. ((\nu X^{\text{int} \rightarrow \bullet}. \lambda y^{\text{int}}. (f y) \wedge (X(y + 1)) \wedge (X(y - 1))) 0)$. Then, given a formula of type $\text{int} \rightarrow \bullet$, one has $\llbracket \text{forall}_{\text{int}} \psi \rrbracket_\rho = \#$ iff $\forall n \in \mathbb{Z}. \llbracket \psi \rrbracket_\rho(n) = \#$. \square

Example 4. The universal quantifiers over predicates can be given without using the fixed-point operators. Since the interpretation of a formula is monotone, $\llbracket \psi \rrbracket_\rho(\perp_\tau) = \#$ if and only if $\forall v \in \mathbb{D}_\tau. \llbracket \psi \rrbracket_\rho(v) = \#$, where \perp_τ is the least element of \mathbb{D}_τ . Since $\perp_\tau = \llbracket \lambda x_1. \dots. \lambda x_k. \text{false} \rrbracket$ (where $\tau = \bar{\tau}_1 \rightarrow \dots \rightarrow \bar{\tau}_k \rightarrow \bullet$), the universal quantifier can be defined by $\text{forall}_\tau := \lambda f^{\tau \rightarrow \bullet}. f(\lambda x_1. \dots. \lambda x_k. \text{false})$. One can define existential quantifiers on predicates by the same technique, using the greatest element $\lambda x_1. \dots. \lambda x_k. \text{true}$ instead of the least one.⁵ \square

⁵ In contrast, the existential quantifier over integers is not definable in this logic.

A νHFL *formula* is a $\nu\text{HFL}_{\mathbb{Z}}$ formula that has no arithmetic subformula. It is known that the validity checking of closed νHFL formulas is decidable. We shall use νHFL as the target of the predicate abstraction in Section 3.

Remark 1. The $\nu\text{HFL}_{\mathbb{Z}}$ validity checking problem is polynomial-time equivalent to the HoCHC satisfiability checking problem [2] with arithmetic as the underlying constraint language. The mutual reductions between the two problems are obtained in essentially the same way as those between the validity checking problem for the first-order fragment of $\nu\text{HFL}_{\mathbb{Z}}$ and the satisfiability problem for CHC [9]. Existential quantifiers in HoCHC correspond to universal quantifiers in $\nu\text{HFL}_{\mathbb{Z}}$, which can be expressed as discussed in Examples 3 and 4 above. \square

3 Predicate Abstraction

This section formalizes a predicate abstraction method for $\nu\text{HFL}_{\mathbb{Z}}$. It computes a pure νHFL formula φ (for which validity checking is decidable) as an underapproximation of an input $\nu\text{HFL}_{\mathbb{Z}}$ formula ψ , by abstracting information about integers. We can then check the validity of φ by using either a (pure) νHFL model checker [6], or using a reduction to HORS model checking [8]. If φ is valid, we can conclude that the original formula ψ is also valid; otherwise, we proceed to the CEGAR phase described in Section 4.

Following the predicate abstraction method of Kobayashi et al. [10] for higher-order functional programs, we use *abstraction types* to express how each subformula should be abstracted. The syntax of *abstraction type* is given by:

$$\begin{aligned} \text{(abstraction type)} \quad \sigma &::= \bullet \mid x : \text{int}[P_1, \dots, P_k] \rightarrow \sigma \mid \sigma_1 \rightarrow \sigma_2 \\ \text{(predicate)} \quad P, Q &::= \text{true} \mid \text{false} \mid p(\tilde{a}) \mid P_1 \wedge P_2 \mid P_1 \vee P_2 \\ \text{(environment)} \quad \Sigma &::= \emptyset \mid \Sigma, x : \text{int} \mid \Sigma, x : \sigma \end{aligned}$$

Here x in $x : \text{int}[P_1, \dots, P_k] \rightarrow \sigma$ is a binding variable whose scope is P_1, \dots, P_k and σ . The type $(x : \text{int}[P_1, \dots, P_n] \rightarrow \sigma)$ describes predicates whose first integer argument x should be abstracted by using the predicates P_1, \dots, P_n . For example, given an abstraction type $(x : \text{int}[x = 0, 1 < x, x < 5] \rightarrow \bullet)$, the predicate $\lambda x.(0 \leq x \wedge x \leq 10)$ on integers is abstracted to the predicate $\lambda b_{x=0} b_{1 < x} b_{x < 5}.(b_{x=0} \vee (b_{1 < x} \wedge b_{x < 5}))$ on Booleans $b_{x=0}, b_{1 < x}$, and $b_{x < 5}$, which respectively represent underapproximations of the values of $x = 0$, $1 < x$, and $x < 5$. Thus, $(\lambda x.(0 \leq x \wedge x \leq 10))2$ is abstracted to $(\lambda b_{x=0} b_{1 < x} b_{x < 5}.b_{x=0} \vee (b_{1 < x} \wedge b_{x < 5}))\text{false true true}$, which evaluates to $\#$. Intuitively, the abstract Boolean predicate above corresponds to $\lambda x.(x = 0 \vee (1 < x \wedge x < 5))$, which is an underapproximation of the original predicate $\lambda x.(0 \leq x \wedge x \leq 10)$. As another example, let us consider the higher-order predicate $\lambda x.\lambda k.k(x + x)$. Given the abstraction type $x : \text{int}[x \geq 0] \rightarrow (r : \text{int}[r \geq x] \rightarrow \bullet) \rightarrow \bullet$, $\lambda x.\lambda k.k(x + x)$ is abstracted to $\lambda b_{x \geq 0}.\lambda k'.k' b_{x \geq 0}$. Here, k' expects as its argument an underapproximation of $r \geq x$, where r refers to the argument $x + x$ of k in the original expression. Since $x \geq 0$ implies $x + x \geq x$, we can pass $b_{x \geq 0}$ to k' as an underapproximation of $r \geq x$. The formula $(\lambda x.\lambda k.k(x + x))1(\lambda r.r \geq 1)$ is then

abstracted to $(\lambda b_{x \geq 0}. \lambda k'. k' b_{x \geq 0}) \mathbf{true} (\lambda b_{r \geq 1}. b_{r \geq 1})$, which evaluates to **true**. In contrast, by using the same abstraction type, $(\lambda x. \lambda k. k(x+x))1 (\lambda r. r \geq 2)$ is abstracted to $(\lambda b_{x \geq 0}. \lambda k'. k' b_{x \geq 0}) \mathbf{true} (\lambda b_{r \geq 1}. \mathbf{false})$, which is equivalent to **false**; note that $b_{r \geq 1}$ only gives an underapproximation of $r \geq 1$, which is not useful to conclude $r \geq 2$, although r in the original formula evaluates to 2.⁶ As the last example shows, the result of predicate abstraction only provides an underapproximation of the original formula (which is equivalent to **true** in the last example).

In order to clarify the shapes of input and output formulas of predicate abstraction, let us define the following two translations from abstraction types to simple types:

$$\begin{aligned} \bullet^\# &:= \bullet & (x : \mathbf{int}[P_1, \dots, P_k] \rightarrow \sigma)^\# &= \mathbf{int} \rightarrow \sigma^\# & (\sigma_1 \rightarrow \sigma_2)^\# &= \sigma_1^\# \rightarrow \sigma_2^\# \\ \bullet^\flat &:= \bullet & (x : \mathbf{int}[P_1, \dots, P_k] \rightarrow \sigma)^\flat &= \overbrace{\bullet \rightarrow \dots \rightarrow \bullet}^k \rightarrow \sigma^\flat & (\sigma_1 \rightarrow \sigma_2)^\flat &= \sigma_1^\flat \rightarrow \sigma_2^\flat. \end{aligned}$$

Given an abstraction type σ , our predicate abstraction converts a $\nu\text{HFL}_{\mathbb{Z}}$ formula of type $\sigma^\#$ to a νHFL formula of type σ^\flat ; for instance, as in the above examples, the abstraction type $x : \mathbf{int}[x = 0, 1 < x, x < 5] \rightarrow \bullet$ is used to abstract a formula of type $(x : \mathbf{int}[x = 0, 1 < x, x < 5] \rightarrow \bullet)^\# = \mathbf{int} \rightarrow \bullet$ to a formula of type $(x : \mathbf{int}[x = 0, 1 < x, x < 5] \rightarrow \bullet)^\flat = \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet$.

Our predicate abstraction is formalized as the *predicate abstraction relation* $\Sigma \mid \Theta \vdash \psi : \sigma \rightsquigarrow \varphi$ (where the metavariable Θ denotes a sequence of predicates P_1, \dots, P_k ; we sometimes use set notations for Θ when the order is not important) given in Figure 2. It means that assuming that the free variables in ψ are abstracted according to the abstraction type environment Σ , and that underapproximations of $\Theta = P_1, \dots, P_k$ are available as special Boolean variables b_{P_1}, \dots, b_{P_k} , ψ can be abstracted to ϕ according to the abstraction type σ . The abstraction relation $\Sigma \mid P_1, \dots, P_k \vdash \psi : \sigma \rightsquigarrow \varphi$ is used to convert a $\nu\text{HFL}_{\mathbb{Z}}$ formula ψ such that $\Sigma^\# \vdash \psi : \sigma^\#$ to a νHFL formula φ such that $\Sigma^\flat, b_{P_1} : \bullet, \dots, b_{P_k} : \bullet \vdash \varphi : \sigma^\flat$, where $\Sigma^\#$ and Σ^\flat are pointwise extensions of the corresponding translations for types, defined by:

$$\begin{aligned} \emptyset^\# &:= \emptyset & (\Sigma, x : \mathbf{int})^\# &:= \Sigma^\#, x : \mathbf{int} & (\Sigma, x : \sigma)^\# &:= \Sigma^\#, x : \sigma^\# \\ \emptyset^\flat &:= \emptyset & (\Sigma, x : \mathbf{int})^\flat &:= \Sigma^\flat & (\Sigma, x : \sigma)^\flat &:= \Sigma^\flat, x : \sigma^\flat \end{aligned}$$

We explain the main rules in Figure 2. (A-PRED) translates a predicate into a corresponding Boolean variable provided that the predicate is currently available. In (A-INTABS), the integer variable x of $\lambda x. \psi$ is translated into Boolean variables b_{P_1}, \dots, b_{P_k} where P_1, \dots, P_k is predicates attached to x ; the predicates P_1, \dots, P_k , as well as corresponding variables b_{P_i} , are available in the body ψ . These Boolean variables are supplied in (A-INTAPP), which applies Boolean variables \widetilde{b}_P to the abstraction φ of the function. The (A-COERCE) rule is used

⁶ We can prove the validity of $(\lambda x. \lambda k. k(x+x))1 (\lambda r. r \geq 2)$ if we use a different abstraction type, like $x : \mathbf{int}[] \rightarrow (r : \mathbf{int}[r \geq 2x] \rightarrow \bullet) \rightarrow \bullet$ for $\lambda x. \lambda k. k(x+x)$.

$$\begin{array}{c}
\frac{}{\Sigma, x : \sigma \mid \Theta \vdash x : \sigma \rightsquigarrow x} \quad (\text{A-VAR}) \\
\\
\frac{P \in \Theta}{\Sigma \mid \Theta \vdash P : \bullet \rightsquigarrow b_P} \quad (\text{A-PRED}) \\
\\
\frac{\Sigma \mid \Theta \vdash \psi : (x : \mathbf{int}[P_1, \dots, P_k] \rightarrow \sigma) \rightsquigarrow \varphi \quad \Sigma \vdash_{\text{ST}} a : \mathbf{int}}{\Sigma \mid \Theta, [a/x]\tilde{P} \vdash \psi a : [a/x]\sigma \rightsquigarrow \varphi \tilde{b}_{[a/x]P}} \quad (\text{A-INTAPP}) \\
\\
\frac{\Sigma, x : \mathbf{int} \mid \Theta, \tilde{P} \vdash \psi : \sigma \rightsquigarrow \varphi}{\Sigma \mid \Theta \vdash \lambda x. \psi : (x : \mathbf{int}[\tilde{P}] \rightarrow \sigma) \rightsquigarrow \lambda \tilde{b}_P. \varphi} \quad (\text{A-INTABS}) \\
\\
\frac{\Sigma \mid \Theta \vdash \psi_1 : \sigma_1 \rightarrow \sigma_2 \rightsquigarrow \varphi_1 \quad \Sigma \mid \Theta \vdash \psi_2 : \sigma_1 \rightsquigarrow \varphi_2}{\Sigma \mid \Theta \vdash \psi_1 \psi_2 : \sigma_2 \rightsquigarrow \varphi_1 \varphi_2} \quad (\text{A-APP}) \\
\\
\frac{\Sigma, x : \sigma_1 \mid \Theta \vdash \psi : \sigma_2 \rightsquigarrow \varphi}{\Sigma \mid \Theta \vdash \lambda x. \psi : \sigma_1 \rightarrow \sigma_2 \rightsquigarrow \lambda x. \varphi} \quad (\text{A-ABS}) \\
\\
\frac{\Sigma \mid \Theta \vdash \psi_1 : \bullet \rightsquigarrow \varphi_1 \quad \Sigma \mid \Theta \vdash \psi_2 : \bullet \rightsquigarrow \varphi_2}{\Sigma \mid \Theta \vdash \psi_1 \wedge \psi_2 \rightsquigarrow \varphi_1 \wedge \varphi_2} \quad (\text{A-AND}) \\
\\
\frac{\Sigma \mid \Theta \vdash \psi_1 : \bullet \rightsquigarrow \varphi_1 \quad \Sigma \mid \Theta \vdash \psi_2 : \bullet \rightsquigarrow \varphi_2}{\Sigma \mid \Theta \vdash \psi_1 \vee \psi_2 \rightsquigarrow \varphi_1 \vee \varphi_2} \quad (\text{A-OR}) \\
\\
\frac{\Sigma, x : \sigma \mid \Theta \vdash \psi : \sigma \rightsquigarrow \varphi}{\Sigma \mid \Theta \vdash \nu x. \psi : \sigma \rightsquigarrow \nu x. \varphi} \quad (\text{A-NU}) \\
\\
\frac{\Sigma \mid \Theta \vdash \psi : \sigma \rightsquigarrow \varphi \quad \Sigma \vdash \varphi : (\Theta, \sigma) \preceq (\Theta', \sigma') \rightsquigarrow \varphi'}{\Sigma \mid \Theta' \vdash \psi : \sigma' \rightsquigarrow \varphi'} \quad (\text{A-COERCE}) \\
\\
\frac{X : \bullet^k \rightarrow \bullet \vdash_{\text{ST}} \xi : \bullet^l \rightarrow \bullet \quad \llbracket X P_1 \dots P_k \rrbracket_\rho \sqsupseteq \bullet \llbracket \xi Q_1 \dots Q_l \rrbracket_\rho \text{ for all } \rho \text{ s.t. } \text{dom}(\rho) = \{X\} \cup \{x \mid x : \mathbf{int} \in \Sigma\}}{\Sigma \vdash \varphi : (P_1, \dots, P_k, \bullet) \preceq (Q_1, \dots, Q_l, \bullet) \rightsquigarrow [\lambda b_{P_1} \dots b_{P_k}. \varphi / X] \xi b_{Q_1} \dots b_{Q_l}} \quad (\text{AC-BASE}) \\
\\
\frac{\Sigma, x : \mathbf{int} \vdash \varphi b_{P_1} \dots b_{P_k} : ((\Theta_1, P_1, \dots, P_k), \sigma_1) \preceq ((\Theta_2, Q_1, \dots, Q_l), \sigma_2) \rightsquigarrow \varphi'}{\Sigma \vdash \varphi : (\Theta_1, x : \mathbf{int}[P_1, \dots, P_k] \rightarrow \sigma_1) \preceq (\Theta_2, x : \mathbf{int}[Q_1, \dots, Q_l] \rightarrow \sigma_2) \rightsquigarrow \lambda b_{Q_1} \dots b_{Q_l}. \varphi'} \quad (\text{AC-INTARROW}) \\
\\
\frac{\Sigma, x : \sigma'_1 \vdash x : (\varepsilon, \sigma'_1) \preceq (\Theta', \sigma_1) \rightsquigarrow \varphi_1 \quad \Sigma, y : \sigma_1 \vdash \varphi y : (\Theta, \sigma_2) \preceq (\Theta', \sigma'_2) \rightsquigarrow \varphi'_2}{\Sigma \vdash \varphi : (\Theta, \sigma_1 \rightarrow \sigma_2) \preceq (\Theta', \sigma'_1 \rightarrow \sigma'_2) \rightsquigarrow \lambda x. [\varphi_1 / y] \varphi'_2} \quad (\text{AC-ARROW})
\end{array}$$

Fig. 2. Abstraction relation

to change the abstraction type σ and predicates \tilde{P} in a judgment. Its major premise is the coercion relation $\Sigma \vdash \varphi : (\Theta, \sigma) \preceq (\Theta', \sigma') \rightsquigarrow \varphi'$, which we shall explain below.

The *coercion relation* $\Sigma \vdash \varphi : (\Theta, \sigma) \preceq (\Theta', \sigma') \rightsquigarrow \varphi'$ transforms an abstraction φ following (Θ, σ) into another abstraction φ' following (Θ', σ') . For example, let $\Theta = (x = 0, \tilde{P})$ and $\Theta' = (x \leq 0, x \geq 0, \tilde{P})$. Then an abstraction φ following (Θ, σ) can be rewritten to another abstraction $[(b_{x \leq 0} \wedge b_{x \geq 0})/b_{x=0}]\varphi$ following (Θ', σ) since $\models x = 0 \iff (x \leq 0 \wedge x \geq 0)$; hence $\Sigma \vdash \varphi : (\Theta, \sigma) \preceq (\Theta', \sigma) \rightsquigarrow [(b_{x \leq 0} \wedge b_{x \geq 0})/b_{x=0}]\varphi$.⁷ Another interesting example is

$$\begin{aligned} \Sigma \vdash \varphi : ((x \leq 0, x \geq 0), \bullet) \preceq ((\), \bullet) \\ \rightsquigarrow ([\mathbf{true}/b_{x \leq 0}, \mathbf{false}/b_{x \geq 0}]\varphi) \wedge ([\mathbf{false}/b_{x \leq 0}, \mathbf{true}/b_{x \geq 0}]\varphi). \end{aligned}$$

Although an abstraction following $((\), \bullet)$ has no information on x , we know that $\models (x \leq 0) \vee (x \geq 0)$ and the above coercion means that it suffices to check the two cases, namely the cases that $x \leq 0$ and that $x \geq 0$.⁸ The most important rule is (AC-BASE), which is a generalization of the above argument. Since $\llbracket XP_1 \cdots P_k \rrbracket_\rho \sqsupseteq \bullet \llbracket \xi Q_1 \cdots Q_l \rrbracket_\rho$ for arbitrary X , substituting $\lambda \tilde{b}_P. \varphi$ for X results in the judgment of the conclusion. Note that (AC-BASE) is non-deterministic in the choice of ξ . In general, the most precise ξ is given by the following formula.

$$\xi = \lambda \tilde{b}_Q. \bigvee_{\substack{\Phi, \Psi \\ \models \Psi(\tilde{Q}) \implies \Phi(\tilde{P})}} \left(\Psi(\tilde{b}_Q) \wedge \left(\bigwedge_{\substack{\tilde{v} \in \{\mathbf{true}, \mathbf{false}\}^n \\ \models \Phi(\tilde{v})}} X \tilde{v} \right) \right)$$

where n and m are the lengths of \tilde{P} and \tilde{Q} , and Φ and Ψ range over positive Boolean formulas over n and m variables. In practice, since computing the best abstraction is too costly, we restrict the shape and size of Ψ and Φ (for example, Ψ is restricted to conjunctive formulas of a certain size, as in [1]).

We present basic properties of the proposed abstraction. The first property is soundness, as stated in the following theorem; see Appendix A for a proof.

Theorem 1 (Soundness of predicate abstraction). *Suppose $\vdash \psi : \bullet \rightsquigarrow \varphi$. If φ is valid, so is ψ .*

The second property is about expressivity. The theorem below (see Appendix C for a proof) says that the proposed predicate abstraction (followed by ν HFL validity checking) is at least as expressive as a *refinement intersection type system* (see Appendix C for the definition). In particular, this result

⁷ More precisely there exists a formula φ' such that $\Sigma \vdash \varphi : (\Theta, \sigma) \preceq (\Theta', \sigma) \rightsquigarrow \varphi'$ and $\varphi' =_{\beta\eta} [(b_{x \leq 0} \wedge b_{x \geq 0})/b_{x=0}]\varphi$.

⁸ Note that the case where both $x \leq 0$ and $x \geq 0$ hold (i.e. $x = 0$) is not problematic because of monotonicity: if $[\mathbf{true}/b_{x \leq 0}, \mathbf{false}/b_{x \geq 0}]\varphi$ is true, then $[\mathbf{true}/b_{x \leq 0}, \mathbf{true}/b_{x \geq 0}]\varphi$ is true as well.

indicates that our approach is more powerful than the approach of Horus for HoCHC [2], because their approach is based on a refinement type system *without intersection types*.

Theorem 2 (Completeness with respect to refinement intersection system). *If $\vdash \psi : \#$ is provable in the refinement dependent intersection type system, then there exists a ν HFL formula φ such that $\vdash \psi : \bullet \rightsquigarrow \varphi$ and φ is valid.*

As indicated in the example below, an abstraction of a valid formula is not necessarily valid. One needs to find a good abstraction type to obtain an abstraction that is valid.

Example 5. Let $\psi := \psi_0 \text{ } \# \text{ } \psi_1$ where

$$\psi_0 := (\lambda n. \lambda f. (f \text{ } 5) \wedge (f \text{ } n)) \quad \text{and} \quad \psi_1 := (\lambda y. (0 \leq y \wedge y \leq 10)).$$

It is easy to see that ψ is valid. For the abstraction type $\sigma_0 := (n : \text{int}[0 \leq n] \rightarrow (y : \text{int}[0 \leq y, y \leq 5] \rightarrow \bullet) \rightarrow \bullet)$, we obtain:

$$\vdash \psi_0 : \sigma_0 \rightsquigarrow \lambda b_{0 \leq n}. \lambda f.$$

$$f \text{ } \text{true} \text{ } \text{true} \wedge ((b_{0 \leq n} \wedge f \text{ } \text{true} \text{ } \text{false}) \vee (f \text{ } \text{true} \text{ } \text{false} \wedge f \text{ } \text{false} \text{ } \text{true})).$$

Here $f \text{ } \text{true} \text{ } \text{true}$ is the abstraction of $f \text{ } 5$ and the remaining part of the body of the function is the abstraction of $f \text{ } n$. This is indeed the “most precise” abstraction of ψ_0 following σ_0 . An abstraction of ψ_1 following $\sigma_2 := (y : \text{int}[0 \leq y, y \leq 5] \rightarrow \bullet)$ is $\lambda b_{0 \leq y} b_{y \leq 5}. b_{0 \leq y} \wedge b_{y \leq 5}$. Hence

$$\vdash \psi_0 \text{ } \# \text{ } \psi_1 : \bullet \rightsquigarrow$$

$$\left(\lambda b_{0 \leq n}. \lambda f. f \text{ } \text{true} \text{ } \text{true} \wedge ((b_{0 \leq n} \wedge f \text{ } \text{true} \text{ } \text{false}) \vee (f \text{ } \text{true} \text{ } \text{false} \wedge f \text{ } \text{false} \text{ } \text{true})) \right) \text{true} (\lambda b_{0 \leq y} b_{y \leq 5}. b_{0 \leq y} \wedge b_{y \leq 5}).$$

The result of the abstraction is invalid. This suggests that a better abstraction type is required for ψ_0 . \square

Example 6. Recall Example 2. Let the abstraction type σ_{sum} for *sum* be:

$$(x : \text{int}[x \geq 0] \rightarrow (y : \text{int}[y \geq x] \rightarrow \bullet) \rightarrow \bullet) \rightarrow n : \text{int}[] \rightarrow (r : \text{int}[r \geq n] \rightarrow \bullet) \rightarrow \bullet.$$

The body ψ_{sum} of *sum* can be abstracted as follows.

$$\begin{aligned} \Gamma; n > 0, n \leq 0 \vdash \psi_{sum} : \bullet \rightsquigarrow \\ (b_{n > 0} \vee k \text{ } b_{n \leq 0}) \\ \wedge (b_{n \leq 0} \vee f \text{ } b_{n > 0} (\lambda b_{x \geq n}. \text{sum} \text{ } f (\lambda b_{y \geq n-1}. k(b_{x \geq n} \wedge b_{y \geq n-1} \wedge b_{n \leq 0}))))). \end{aligned}$$

Here, $\Gamma = \text{sum} : \sigma_{sum}, x : \text{int}, f : (y : \text{int}[y \geq x] \rightarrow \bullet) \rightarrow \bullet, n : \text{int}, k : r : \text{int}[r \geq n] \rightarrow \bullet$. By (A-COERCE) (let ξ in (AC-BASE) be $(\lambda X. X \text{ } \text{false} \text{ } \text{true} \wedge X \text{ } \text{true} \text{ } \text{false})X$), we get:

$$\begin{aligned} \Gamma \vdash \psi_{sum} : \sigma_{sum} \rightsquigarrow (\lambda X. X \text{ } \text{false} \text{ } \text{true} \wedge X \text{ } \text{true} \text{ } \text{false}) \\ (\lambda b_{n > 0} b_{n \leq 0}. (b_{n > 0} \vee k \text{ } b_{n \leq 0}) \\ \wedge (b_{n \leq 0} \vee f \text{ } b_{n > 0} (\lambda b_{x \geq n}. \text{sum} \text{ } f (\lambda b_{y \geq n-1}. k(b_{x \geq n} \wedge b_{y \geq n-1} \wedge b_{n > 0}))))). \end{aligned}$$

The output formula can be simplified to:

$$k \text{ true} \wedge f \text{ true} \lambda b_{x \geq n}. \text{sum } f \lambda b_{y \geq n-1}. k(b_{x \geq n} \wedge b_{y \geq n-1}).$$

Thus, the whole formula is abstracted to $\text{sum}' (\lambda b_{x \geq 0}. \lambda k. k b_{x \geq 0}) \lambda b_{r \geq n}. b_{r \geq n}$, where

$$\text{sum}' := \nu \text{sum}. \lambda f. \lambda k. k \text{ true} \wedge f \text{ true} \lambda b_{x \geq n}. \text{sum } f \lambda b_{y \geq n-1}. k(b_{x \geq n} \wedge b_{y \geq n-1}),$$

which is equivalent to **true** (as can be confirmed by a HFL model checker); hence, we can conclude that the original formula is also valid. \square

4 Counterexample-Guided Abstraction Refinement

This section describes the second component of our method, counterexample-guided abstraction refinement (CEGAR). Let φ be an abstraction of ψ , i.e. $\vdash \psi : \bullet \rightsquigarrow \varphi$. If φ is valid, then ψ is valid and we are done, as discussed in the previous section. Otherwise either ψ is invalid or the abstraction is too coarse (or both). Below we first introduce the notion of a counterexample (which shows why φ is invalid) in our context in Section 4.1. We then discuss, in Section 4.2, a way to determine whether the counterexample also implies the invalidity of ψ . If that is the case, we can conclude that ψ is invalid; otherwise, we refine the abstraction by finding new predicates, as discussed in Section 4.3.

4.1 Counterexample

In the context of our $\nu\text{HFL}_{\mathbb{Z}}$ validity checking, a counterexample is a witness of the invalidity of a closed proposition. Formally the set of *candidate counterexamples* used in this paper is given by the following grammar:

$$c ::= \text{false} \mid c \wedge * \mid * \wedge c \mid c \vee c.$$

Intuitively a candidate counterexample is a sufficiently large part of a formula, which ensures the invalidity of the formula; replacing each $*$ in a counterexample to an arbitrary formula results in a false formula.

A candidate counterexample c is a *counterexample of ψ* , written $c \triangleright \psi$, if c witnesses the invalidity of ψ . Intuitively $c \triangleright \psi$ means that ψ matches the pattern of c . For example, counterexamples of

$$((1 = 0) \wedge (10 > 1)) \vee ((4 \neq 2 + 2) \wedge (3 < 0))$$

are

$$(\text{false} \wedge *) \vee (\text{false} \wedge *) \quad \text{and} \quad (\text{false} \wedge *) \vee (* \wedge \text{false}).$$

Here, the subformulas $1 = 0$, $4 \neq 2 + 2$, and $3 < 0$ “match” **false**, since they are equivalent to **false**. The counterexamples above evaluate to **false** irrespectively of which formula we substitute for $*$.

The relation $c \triangleright \psi$ is formally defined inductively by the rules in Fig. 3.

$$\begin{array}{c}
\frac{\models \neg p(\tilde{a})}{\mathbf{false} \triangleright p(\tilde{a})} \qquad \frac{c_1 \triangleright \varphi_1}{c_1 \wedge * \triangleright \varphi_1 \wedge \varphi_2} \qquad \frac{c_2 \triangleright \varphi_2}{* \wedge c_2 \triangleright \varphi_1 \wedge \varphi_2} \\
\\
\frac{c_1 \triangleright \varphi_1 \quad c_2 \triangleright \varphi_2}{c_1 \vee c_2 \triangleright \varphi_1 \vee \varphi_2} \qquad \frac{c \triangleright ([\varphi'/x]\varphi) \tilde{\psi}}{c \triangleright (\lambda x. \varphi) \varphi' \tilde{\psi}} \qquad \frac{c \triangleright ([\nu x. \varphi/x]\varphi) \tilde{\psi}}{c \triangleright (\nu x. \varphi) \tilde{\psi}}
\end{array}$$

Fig. 3. Rules for the counterexample relation $c \triangleright \psi$

Example 7. Let φ be the result of the abstraction in Example 5, i.e.,

$$\begin{aligned}
& \vdash \psi_0 \not\triangleright \psi_1 : \bullet \rightsquigarrow \\
& \left(\lambda b_{0 \leq n}. \lambda f. f \mathbf{true} \mathbf{true} \wedge ((b_{0 \leq n} \wedge f \mathbf{true} \mathbf{false}) \right. \\
& \quad \left. \vee (f \mathbf{true} \mathbf{false} \wedge f \mathbf{false} \mathbf{true})) \right) \mathbf{true} (\lambda b_{0 \leq y} b_{y \leq 5}. b_{0 \leq y} \wedge b_{y \leq 5}).
\end{aligned}$$

Then $c \triangleright \varphi$ for

$$c = * \wedge (* \wedge (* \wedge \mathbf{false})) \vee (* \wedge (\mathbf{false} \wedge *)).$$

In fact, we have

$$c \triangleright (\mathbf{true} \wedge \mathbf{true}) \wedge \left((\mathbf{true} \wedge (\mathbf{true} \wedge \mathbf{false})) \vee ((\mathbf{true} \wedge \mathbf{false}) \wedge (\mathbf{false} \wedge \mathbf{true})) \right),$$

from which we can derive $c \triangleright \varphi$ by using the rule for β -redexes in Figure 3. \square

Every invalid formula has a counterexample. This follows easily from the fact that, for co-continuous arguments, $\llbracket \nu x^\tau. \psi \rrbracket_\rho$ coincides with $\prod_{i \in \omega} \llbracket \lambda x^\tau. \psi \rrbracket_\rho^i(\top^\tau)$ in the semantics of $\nu\text{HFL}_{\mathbb{Z}}$ formulas⁹; see, e.g. Lemma 14 of [11]:

Proposition 1. *For any closed $\nu\text{HFL}_{\mathbb{Z}}$ formula ψ of type \bullet , $\llbracket \psi \rrbracket = \mathbf{ff}$ if and only if $c \triangleright \psi$ for some c .*

Let ψ be a $\nu\text{HFL}_{\mathbb{Z}}$ formula, and φ be a νHFL formula obtained by applying predicate abstraction to ψ , and suppose that φ is invalid. The goal of the rest of this subsection is to compute a set of candidate counterexamples for ψ .

By using a model-checker for νHFL (without arithmetic), we can obtain a counterexample c for φ (i.e., c such that $c \triangleright \varphi$). Note, however, that the shape of c does not necessarily match that of ψ , due to the conjunctions and disjunctions that may have been introduced in the predicate abstraction phase. For example, recall φ in Example 7, which is an abstraction of the formula ψ in Example 5. The counterexample c in Example 7 is not a counterexample of ψ . In fact, the original formula has just conjunctions, and the disjunctions (and also some of the conjunctions) in c have been introduced by the abstraction.

To address the issue above, we distinguish between boolean connectives in an original formula and those introduced in the abstraction phase, by writing

⁹ This is not the case for full $\text{HFL}_{\mathbb{Z}}$.

$\bar{\wedge}$ and $\bar{\vee}$ for the latter. We assume that the boolean connectives in (AC-BASE) (used in ξ) are $\bar{\wedge}$ and $\bar{\vee}$. A counterexample with $\bar{\wedge}$ or $\bar{\vee}$ is called an *abstract counterexample*.

An abstract counterexample c induces a set of (candidate) counterexamples $\mathcal{C}(c)$ defined by:

$$\begin{aligned} \mathcal{C}(\mathbf{false}) &:= \{\mathbf{false}\} & \mathcal{C}(c_1 \vee c_2) &:= \{c'_1 \vee c'_2 \mid c'_i \in \mathcal{C}(c_i)\} \\ \mathcal{C}(* \wedge c) &:= \{* \wedge c' \mid c' \in \mathcal{C}(c)\} & \mathcal{C}(c \wedge *) &:= \{c' \wedge * \mid c' \in \mathcal{C}(c)\} \\ \mathcal{C}(c_1 \bar{\vee} c_2) &:= \mathcal{C}(c_1) \cup \mathcal{C}(c_2) & \mathcal{C}(* \bar{\wedge} c) &:= \mathcal{C}(c) & \mathcal{C}(c \bar{\wedge} *) &:= \mathcal{C}(c). \end{aligned}$$

Example 8. The formula in Example 7 should be written as

$$\begin{aligned} & \left(\lambda b_{0 \leq n}. \lambda f. f \mathbf{true} \mathbf{true} \wedge ((b_{0 \leq n} \bar{\wedge} f \mathbf{true} \mathbf{false}) \right. \\ & \quad \left. \bar{\vee} (f \mathbf{true} \mathbf{false} \bar{\wedge} f \mathbf{false} \mathbf{true})) \right) \mathbf{true} (\lambda b_{0 \leq y} b_{y \leq 5}. b_{0 \leq y} \wedge b_{y \leq 5}). \end{aligned}$$

and an abstract counterexample is $c = * \wedge \left((* \bar{\wedge} (* \wedge \mathbf{false})) \bar{\vee} (* \bar{\wedge} (\mathbf{false} \wedge *)) \right)$. Then $\mathcal{C}(c) = \{* \wedge (* \wedge \mathbf{false}), * \wedge (\mathbf{false} \wedge *)\}$. \square

Assume that $\vdash \psi : \bullet \rightsquigarrow \varphi$ and φ is invalid. Then a model-checker generates an abstract counterexample c of φ . We randomly pick a candidate counterexample from $\mathcal{C}(c)$ and proceed to feasibility checking.

4.2 Feasibility Check

Let ψ be a closed formula, and c be a candidate counterexample of ψ . We would like to check whether $c \triangleright \psi$.

The rules in Fig. 3 can be seen as the definition of a procedure for checking $c \triangleright \psi$. For example, to check whether $c_1 \vee c_2 \triangleright \psi_1 \vee \psi_2$, the procedure checks whether $c_1 \triangleright \psi_1$ and $c_2 \triangleright \psi_2$. If the candidate counterexample c comes from an abstract counterexample of an abstraction of ψ , the procedure to check $c \triangleright \psi$ terminates.¹⁰

If $c \triangleright \psi$, then ψ is invalid. Otherwise we refine the abstraction by using c , as discussed in the next subsection.

Remark 2. In the actual implementation, we allow ψ to contain free integer variables x_1, \dots, x_k , and judge the validity of $\forall x_1, \dots, x_k. \psi$. In this case, the feasibility check is a little more involved.

4.3 Predicate Discovery and Abstraction Refinement

Let c be an infeasible candidate counterexample c of ψ . To improve the precision of abstraction, we would like to find predicates that are useful to show the validity of ψ . Our approach is based on a refinement dependent intersection type system.

¹⁰ This procedure does not terminate in general. An example is $\mathbf{false} \triangleright (\nu f. \lambda x. f x) 0$.

Our type system is a variant of the refinement intersection type system for higher-order functional programs [18]. An important feature of our type system is a type of the form $\neg c$, for each candidate counterexample c . Intuitively, a formula ψ has type $\neg c$ if ψ does not have c as a counterexample (i.e. if $c \not\vdash \psi$). The typing rules include:

$$\frac{\Delta \mid \Phi \vdash \psi_2 : \neg c}{\Delta \mid \Phi \vdash \psi_1 \wedge \psi_2 : \neg(* \wedge c)} \quad \text{and} \quad \frac{\Delta \mid \Phi \vdash \psi_i : \neg c_i \quad (i = 1 \text{ or } 2)}{\Delta \mid \Phi \vdash \psi_1 \vee \psi_2 : \neg(c_1 \vee c_2)}$$

where Δ is a type environment and Φ is a precondition. The former rule checks the specified branch, ignoring the discarded part (i.e. ψ_1) corresponding to $*$. So $\vdash \mathbf{false} \wedge \mathbf{true} : \neg(* \wedge \mathbf{false})$ is a valid type judgment although $\mathbf{false} \wedge \mathbf{true}$ is an invalid formula. The latter rule says that, to show $(c_1 \vee c_2) \not\vdash (\psi_1 \vee \psi_2)$, it suffices to prove either $c_1 \not\vdash \psi$ or $c_2 \not\vdash \psi$. The complete list of typing rules can be found in Appendix C.

Example 9. Here is an example of a derivation in the refinement intersection type system.

$$\frac{\frac{y : \mathbf{int} \mid y \leq 9 \vdash (y \leq 10) : \neg(\mathbf{false})}{y : \mathbf{int} \mid y \leq 9 \vdash (0 \leq y) \wedge (y \leq 10) : \neg(* \wedge \mathbf{false})}}{\epsilon \mid \mathbf{true} \vdash \lambda y.(0 \leq y) \wedge (y \leq 10) : y : \{\mathbf{int} \mid y \leq 9\} \rightarrow \neg(* \wedge \mathbf{false})}$$

Although the condition $y \leq 9$ does not imply that the body is true, $y \leq 9$ implies that $* \wedge \mathbf{false}$ is not a valid counterexample for the body (since the right branch of the conjunction $y \leq 10$ is actually true). \square

The abstraction refinement phase proceeds as follows. Given an infeasible candidate counterexample c of ψ , we construct a derivation $\vdash \psi : \neg c$, a proof of $c \not\vdash \psi$. Then we extract predicates from the derivation.

We can use a template-based dependent type inference [18,19] to find a derivation for $\vdash \psi : \neg c$. We first prepare a template of each refinement type, which is completely determined by ψ and c . For example, the template for the formula ψ in Example 5 with candidate counterexample $* \wedge (* \wedge \mathbf{false})$ is

$$\begin{aligned} & (\lambda n. \lambda f. fn) : \\ & n : \{\mathbf{int} \mid P(n)\} \rightarrow (y : \{\mathbf{int} \mid Q(n, y)\} \rightarrow \neg(* \wedge \mathbf{false})) \rightarrow \neg(* \wedge \mathbf{false}) \\ & (\lambda y. (0 \leq y) \wedge (y \leq 10)) : y : \{\mathbf{int} \mid R(y)\} \rightarrow \neg(* \wedge \mathbf{false}) \end{aligned}$$

where P , Q and R are predicate variables. We then generate constraints on predicate variables so that the formula has type $\neg c$ if and only if the constraints are satisfied, in a manner similar to [10]. We then solve the constraints by using a CHC solver [19,4] (see the remark below) and obtain a derivation for $\vdash \psi : \neg c$.

Remark 3. The constraints generated are actually more complex than those generated in the CEGAR phase of [10]. The constraints generated in our CEGAR phase are conjunctions of clauses of the form:

$$P_1(\tilde{x}_1) \vee \cdots \vee P_k(\tilde{x}_k) \Leftarrow A_1 \wedge \cdots \wedge A_m,$$

where $k \geq 0$, and each A_i is an atom of the form $P_\ell(\tilde{y})$ or $p(\tilde{a})$. The constraints are acyclic (in the sense that there is no circular dependency like $P(x) \vee \dots \Leftarrow Q(x) \wedge \dots$ and $Q(x) \vee \dots \Leftarrow P(x) \wedge \dots$). In contrast, the constraints generated in the CEGAR phase of [10] are acyclic CHCs, obtained by imposing the restriction $k \leq 1$ to the form of constraints above. Thanks to the acyclicity, we can solve the extended constraints by invoking a CHC solver multiple times. \square

The extraction of predicates from a derivation for $\vdash \psi : \neg c$ is rather straightforward. For example, suppose a subformula $\lambda x. \psi_0$ of ψ that has simple type $\mathbf{int} \rightarrow \tau$. Because our type system has intersection types, this subformula may have several types, say $x : \{\mathbf{int} \mid P_i(x)\} \rightarrow \delta_i$ for each $i = 1, \dots, k$. Then the extracted abstraction type for this subformula is $x : \mathbf{int}[P_1, \dots, P_k] \rightarrow \sigma$ for some σ .

Example 10. Let ψ be a formula in Example 5, i.e. $\psi = \psi_0 \text{ } \tau \text{ } \psi_1$ where $\psi_0 = (\lambda n. \lambda f. f5 \wedge fn)$ and $\psi_1 = (\lambda y. (0 \leq y) \wedge (y \leq 10))$. Assume that a candidate counterexample is $* \wedge (* \wedge \mathbf{false})$ (cf. Example 8). A derivation of $\vdash \psi : \neg(* \wedge (* \wedge \mathbf{false}))$ contains the following typing to subformulas:

$$\begin{aligned} \psi_0 &: n : \{\mathbf{int} \mid n \leq 7\} \rightarrow (y : \{\mathbf{int} \mid y \leq 8\} \rightarrow \neg(* \wedge \mathbf{false})) \rightarrow \neg(* \wedge (* \wedge \mathbf{false})) \\ \psi_1 &: y : \{\mathbf{int} \mid y \leq 9\} \rightarrow \neg(* \wedge \mathbf{false}) \end{aligned}$$

The abstraction types for ψ_0 and ψ_1 extracted from this proof are

$$\sigma'_0 := n : \mathbf{int}[n \leq 7] \rightarrow (y : \mathbf{int}[y \leq 8] \rightarrow \bullet) \rightarrow \bullet \quad \sigma'_1 := y : \mathbf{int}[y \leq 9] \rightarrow \bullet.$$

By adding this information to the abstraction types in Example 5, one obtains

$$\begin{aligned} \sigma''_0 &:= n : \mathbf{int}[0 \leq n, n \leq 7] \rightarrow (y : \mathbf{int}[0 \leq y, y \leq 5, y \leq 8] \rightarrow \bullet) \rightarrow \bullet \\ \sigma''_2 &:= y : \mathbf{int}[0 \leq y, y \leq 5, y \leq 9] \rightarrow \bullet. \end{aligned}$$

After this refinement, there exists an abstraction that is true. \square

The refinement process enjoys the progress property in the following sense.

Theorem 3 (Progress). *Assume a derivation of $\vdash \psi : \neg c$ in the dependent intersection type system. Then there exists an abstraction $\vdash \psi : \bullet \rightsquigarrow \varphi$ following the abstraction types extracted from the derivation such that, for any abstract counterexample c' of φ , the set $\mathcal{C}(c')$ contains a candidate counterexample that differs from c .*

5 Implementation and Evaluation

We have implemented a $\nu\text{HFL}_{\mathbb{Z}}$ validity checker PAHFL based on the method described so far. PAHFL uses the following tools as backend solvers:

- Z3 [14], as a backend SMT solver for computing predicate abstraction.

- HorSat2, as a backend model checker for checking the validity of (pure) ν HFL formulas and extracting an abstract counterexample if there is any. (HorSat2 is a HORS model checker, but it can also be used as a ν HFL model checker by using the reduction of Kobayashi et al. [8].)
- RCaml [19] and HoIce [4] as backend constraint solvers, for the predicate discovery phase.

Given a ν HFL $_{\mathbb{Z}}$ formula, PAHFL starts with the empty set of predicates, and repeats the CEGAR loop until it succeeds to prove or disprove the validity of the formula; of course, it may run forever since the validity checking problem is undecidable.

We have conducted experiments to compare our tool with two related tools:

- A HoCHC solver called Horus [2], which solves the satisfiability of HoCHC (a higher-order extension of CHC, constrained Horn Clauses). Since the HoCHC satisfiability problem and ν HFL $_{\mathbb{Z}}$ can be mutually reducible (recall Remark 1), Horus is a direct competitor of our tool.
- An automated higher-order program verification tool MOCHI, developed by Kobayashi et al. [10,16,13]. Since the main application of our ν HFL $_{\mathbb{Z}}$ validity checking is higher-order program verification, MOCHI is also an (indirect) competitor of our tool. Like PAHFL, MOCHI uses predicate abstraction and CEGAR, though its main building block is HORS model checking (rather than HFL validity checking. Actually, the goal of our project has been to replace the HORS-based approach of MOCHI with the HFL-based approach, where the latter provides a more uniform approach to program verification. Thus the goal of the comparison with MOCHI is to confirm that our new tool PAHFL works at least as effectively as MOCHI, for program verification problems. There are other fully automated verification tools for functional programs, including those based on refinement type inference [23,5,24,25,3]. We have chosen MOCHI as the target of the experimental comparison, as the underlying technique is directly related; other tools can be indirectly compared through their experimental comparison with MOCHI found in the respective papers [24,25,3].

Both experiments were conducted on a Linux server with Intel Xeon CPU E5-2680 v3 and 64 GB of RAM with a timeout set to 180 seconds.

Comparison with Horus We used two benchmark sets in this experiment: Benchmarks A and B. Benchmark A has been taken from that of Horus [2], which in turn comes from a benchmark set of MOCHI for safety property verification [10]. Benchmark B has been obtained from another benchmark set of MOCHI, by using our own translation (based on [12,22]). Benchmark A has 10 instances and B has 58. We used Z3 as a backend CHC solver of Horus.

The result is shown in Table 1 and Figure 4. In the table, the row ‘Unknown’ means that Horus could not prove the validity of ν HFL $_{\mathbb{Z}}$ formula due to its incompleteness; as mentioned in Section 1, Horus reduces HoCHC problems

		Benchmark A	Benchmark B
Ours	Solved	8	50
	Timeout	0	8
Horus	Solved	7	19
	Timeout	0	3
	Unknown	1	35

Table 1. Comparison of PAHFL with Horus

to CHC problems in a sound but *incomplete* manner. The “unknown” case of Horus in Benchmark A is attributed to the lack of intersection types in the type system used by Horus; recall our remark before Theorem 2. As is clear from Figure 4, Horus is much faster than PAHFL when Horus succeeds. In fact, Horus terminated within 0.1 seconds in those cases. In contrast, PAHFL clearly outperforms Horus in the number of solved instances. That is more apparent for the subset of benchmarks consisting of only higher-order inputs: see Appendix E.

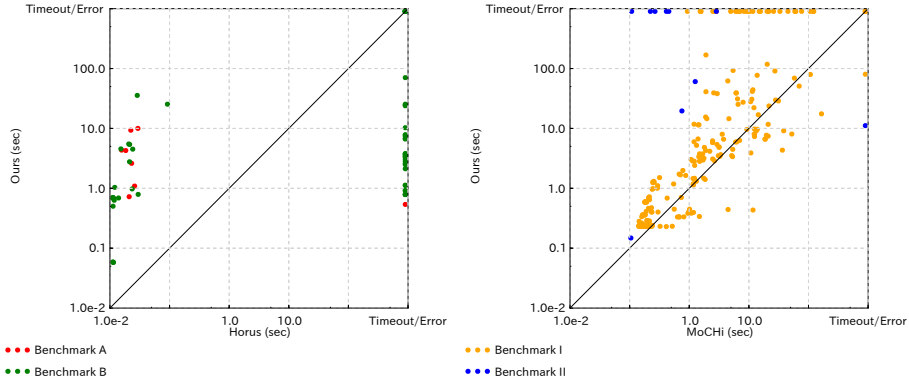


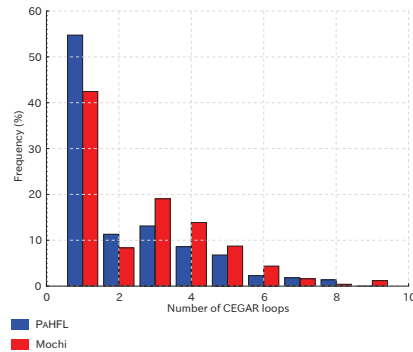
Fig. 4. Comparison of PAHFL with Horus (left) and MoCHI (right)

Comparison with MoCHI Figure 4 shows the result of the comparison with MoCHI. We used two benchmark sets; Benchmark I consists of 258 of 262 safety property verification problems of OCaml programs used in [16] and Benchmark II consists of a modified version¹¹ of 10 non-termination property verification problems used in [13]. All the benchmark programs are small (each around 10 lines of OCaml programs), but many of them are tricky. The $\nu\text{HFL}_{\mathbb{Z}}$ formulas

¹¹ Existential quantifiers that arise from the original programs have been replaced with finite disjunctions.

in both benchmark sets have been obtained by using the translation used for obtaining Benchmark B. Different modes of MOCHI have been used for the two benchmark sets: the reachability verification mode [10] for Benchmark I, and the non-termination verification mode [13] for Benchmark II. PAHFL solved 221 instances for Benchmark I and 4 for II while MOCHI solved 252 for Benchmark I and 9 for II. Although PAHFL is a little inferior to MOCHI in the number of solved instances, PAHFL is competitive in terms of the running times for solved instances, as shown in Figure 4.

Figure 5 and Table 2 respectively show the distribution of the number of CEGAR loops and the average percentage of the time spent in each phase for solved instances in Benchmark I and II. PAHFL and MOCHI have similar tendencies in both the figure and the table.



	PAHFL	MOCHI
Preprocess	31.0	39.2
Abstraction	47.7	45.4
Refine	19.1	13.9
Model Checking	2.21	1.49

Table 2. Average percentage of time spent in each phase

Fig. 5. Distribution of the number of CEGAR loops

6 Related Work

Burn et al. [2] introduced higher-order constrained Horn clauses (HoCHC), and developed Horus, a type-based HoCHC satisfiability checker. As already mentioned, Horus is a direct competitor of our tool PAHFL, since the satisfiability problem for HoCHC is essentially equivalent to the validity problem for $\nu\text{HFL}_{\mathbb{Z}}$. As confirmed by experiments, our tool is often slower than Horus, but can solve more problem instances. Ong and Wagner [15] also studied some theoretical aspects of HoCHC, but have not developed an actual verification tool, to our knowledge.

Our technique of predicate abstraction has been inspired by the corresponding techniques developed for MOCHI [10,13]; in particular, we have borrowed the notion of abstraction types from their work. Our predicate abstraction technique is, however, different from theirs, in that our technique is used for abstracting

formulas, while their technique is for abstracting programs. Our technique can also be considered more general since the techniques of MOCHI [10,13] are specialized for the verification of either reachability or non-termination, whereas our technique can be used for the verification of arbitrary branching-time safety properties, including the reachability and non-termination properties.

Higher-order fixpoint logic (HFL) has originally been proposed by Viswanathan and Viswanathan [21]. Kobayashi et al. [12,22] introduced $\text{HFL}_{\mathbb{Z}}$, an extension of HFL with integers, and showed its applications to higher-order program verification. Although a pure HFL model checker has already been developed by Hosoi et al. [6], there is no automated validity checker for $\text{HFL}_{\mathbb{Z}}$, to our knowledge. Kobayashi et al. [9] have recently developed a validity checker for the first-order fragment of $\text{HFL}_{\mathbb{Z}}$, which reduces (in a sound but incomplete manner) the validity problem for the first-order fragment of $\text{HFL}_{\mathbb{Z}}$ to that for the ν -only, first-order fragment of $\text{HFL}_{\mathbb{Z}}$. We expect that the same technique can be used to obtain a full $\text{HFL}_{\mathbb{Z}}$ validity checker from our $\nu\text{HFL}_{\mathbb{Z}}$ validity checker.

Another major approach to automated verification of higher-order programs is a type-based one [23,5,24,25,3,18], some of which incorporates counterexample-guided refinement [23,18]. Most of them are restricted to verification of the (un)reachability problem in the presence of only demonic branches (except [20]), and do not support intersection types (except [18]). Our support of both kinds of (i.e., angelic and demonic) branches is crucial for building the full $\text{HFL}_{\mathbb{Z}}$ validity checker as mentioned above. Intersection types are also crucial for high precision. To mitigate the high cost of predicate abstraction and discovery, Sato et al. [16] combined higher-order model checking and type inference.

7 Conclusion

We have proposed a new method for automated $\nu\text{HFL}_{\mathbb{Z}}$ validity checking based on predicate abstraction and CEGAR, and developed a tool based on the proposed technique. According to our experiments on applications to program verification, our tool outperformed Horus in terms of precision, and was competitive with a more specialized program verification tool MOCHI. We plan to develop a full $\text{HFL}_{\mathbb{Z}}$ validity checker, by combining the present work with the work of Kobayashi et al’s [9] for the first-order $\text{HFL}_{\mathbb{Z}}$ validity checker. It is also left for future work to extending the logic with data structures (such as lists and trees), which is important for verification of higher-order functional programs that manipulate data structures. To this end, we plan to exploit two approaches: one is to encode data structures as higher-order functions as in MOCHI [17], and the other is to directly handle data structures, as in the refinement-type-based approach [5,25].

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

References

1. Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Burke, M., Soffa, M.L. (eds.) Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Snowbird, Utah, USA, June 20-22, 2001. pp. 203–213. ACM (2001). <https://doi.org/10.1145/378795.378846>
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. *Journal of Automated Reasoning* (2010), to appear. A preliminary summary appeared in Proceedings of TACAS 2018.
4. Champion, A., Chiba, T., Kobayashi, N., Sato, R.: Ice-based refinement type discovery for higher-order functional programs. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10805, pp. 365–384. Springer (2018). https://doi.org/10.1007/978-3-319-89960-2_20
5. Hashimoto, K., Unno, H.: Refinement type inference via horn constraint optimization. In: Blazy, S., Jensen, T.P. (eds.) Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9291, pp. 199–216. Springer (2015). https://doi.org/10.1007/978-3-662-48288-9_12
6. Hosoi, Y., Kobayashi, N., Tsukada, T.: A type-based HFL model checking algorithm. In: Lin, A.W. (ed.) Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11893, pp. 136–155. Springer (2019). https://doi.org/10.1007/978-3-030-34175-6_8
7. Kobayashi, N.: HorSat2: A saturation-based model checker for higher-order recursion schemes (2015), <https://www.kb.is.s.u-tokyo.ac.jp/~koba/horsat2/>
8. Kobayashi, N., Lozes, É., Bruse, F.: On the relationship between higher-order recursion schemes and higher-order fixpoint logic. In: Castagna, G., Gordon, A.D. (eds.) Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 246–259. ACM (2017)
9. Kobayashi, N., Nishikawa, T., Igarashi, A., Unno, H.: Temporal verification of programs via first-order fixpoint logic. In: Chang, B.E. (ed.) Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11822, pp. 413–436. Springer (2019). https://doi.org/10.1007/978-3-030-32304-2_20
10. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. In: Hall, M.W., Padua, D.A. (eds.) Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011. pp. 222–233. ACM (2011). <https://doi.org/10.1145/1993498.1993525>
11. Kobayashi, N., Tsukada, T., Watanabe, K.: Higher-order program verification via HFL model checking. *CoRR* **abs/1710.08614** (2017), <http://arxiv.org/abs/1710.08614>

12. Kobayashi, N., Tsukada, T., Watanabe, K.: Higher-order program verification via HFL model checking. In: Ahmed, A. (ed.) Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10801, pp. 711–738. Springer (2018). https://doi.org/10.1007/978-3-319-89884-1_25
13. Kuwahara, T., Sato, R., Unno, H., Kobayashi, N.: Predicate abstraction and CEGAR for disproving termination of higher-order functional programs. In: Kroening, D., Pasareanu, C.S. (eds.) Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9207, pp. 287–303. Springer (2015). https://doi.org/10.1007/978-3-319-21668-3_17
14. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. pp. 337–340 (2008)
15. Ong, C.L., Wagner, D.: Hochc: A refutationally complete and semantically invariant system of higher-order logic modulo theories. In: 34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019. pp. 1–14. IEEE (2019). <https://doi.org/10.1109/LICS.2019.8785784>
16. Sato, R., Iwayama, N., Kobayashi, N.: Combining higher-order model checking with refinement type inference. In: Hermenegildo, M.V., Igarashi, A. (eds.) Proceedings of the 2019 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM@POPL 2019, Cascais, Portugal, January 14-15, 2019. pp. 47–53. ACM (2019). <https://doi.org/10.1145/3294032.3294081>
17. Sato, R., Unno, H., Kobayashi, N.: Towards a scalable software model checker for higher-order programs. In: Albert, E., Mu, S. (eds.) Proceedings of the ACM SIGPLAN 2013 Workshop on Partial Evaluation and Program Manipulation, PEPM 2013, Rome, Italy, January 21-22, 2013. pp. 53–62. ACM (2013). <https://doi.org/10.1145/2426890.2426900>
18. Terauchi, T.: Dependent types from counterexamples. In: Hermenegildo, M.V., Palsberg, J. (eds.) Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010. pp. 119–130. ACM (2010). <https://doi.org/10.1145/1706299.1706315>
19. Unno, H., Kobayashi, N.: Dependent type inference with interpolants. In: Porto, A., López-Fraguas, F.J. (eds.) Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal. pp. 277–288. ACM (2009). <https://doi.org/10.1145/1599410.1599445>
20. Unno, H., Satake, Y., Terauchi, T.: Relatively complete refinement type system for verification of higher-order non-deterministic programs. Proc. ACM Program. Lang. **2**(POPL), 12:1–12:29 (2018). <https://doi.org/10.1145/3158100>
21. Viswanathan, M., Viswanathan, R.: A higher order modal fixed point logic. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3170, pp. 512–528. Springer (2004). https://doi.org/10.1007/978-3-540-28644-8_33

22. Watanabe, K., Tsukada, T., Oshikawa, H., Kobayashi, N.: Reduction from branching-time property verification of higher-order programs to HFL validity checking. In: Hermenegildo, M.V., Igarashi, A. (eds.) Proceedings of the 2019 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM@POPL 2019, Cascais, Portugal, January 14-15, 2019. pp. 22–34. ACM (2019). <https://doi.org/10.1145/3294032.3294077>
23. Zhu, H., Jagannathan, S.: Compositional and lightweight dependent type inference for ML. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7737, pp. 295–314. Springer (2013). https://doi.org/10.1007/978-3-642-35873-9_19
24. Zhu, H., Nori, A.V., Jagannathan, S.: Learning refinement types. In: Fisher, K., Reppy, J.H. (eds.) Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015. pp. 400–411. ACM (2015). <https://doi.org/10.1145/2784731.2784766>
25. Zhu, H., Petri, G., Jagannathan, S.: Automatically learning shape specifications. In: Krintz, C., Berger, E. (eds.) Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016. pp. 491–507. ACM (2016). <https://doi.org/10.1145/2908080.2908125>

A Soundness of Predicate Abstraction

As usual, the correctness of the predicate abstraction is ensured by a semantic characterization of the abstraction. For each closed abstraction type σ , we define the abstraction function α_σ and the concretization function γ_σ , which give us translations between the concrete domain σ^\sharp and the abstract domain σ^\flat .

Definition 1 (abstraction function and concretization function). *Given a closed abstraction type σ , the abstraction function $\alpha_\sigma^\flat : \mathbb{D}_{\sigma^\sharp} \rightarrow \mathbb{D}_{\sigma^\flat}$ and the concretization function $\gamma_\sigma^\flat : \mathbb{D}_{\sigma^\flat} \rightarrow \mathbb{D}_{\sigma^\sharp}$ are defined as follows:*

$$\begin{aligned} \alpha_\bullet(v) &:= v & \gamma_\bullet(v) &:= v \\ \alpha_{\sigma_1 \rightarrow \sigma_2}(v)(x) &:= \alpha_{\sigma_2}(v(\gamma_{\sigma_1}(x))) & \gamma_{\sigma_1 \rightarrow \sigma_2}(v)(x) &:= \gamma_{\sigma_2}(v(\alpha_{\sigma_1}(x))) \end{aligned}$$

and, for $\text{int}[\tilde{P}] \rightarrow \sigma$,

$$\begin{aligned} \alpha_{x:\text{int}[\tilde{P}] \rightarrow \sigma}(v)(\tilde{b}) &:= \prod_{\substack{n \in \mathbb{Z} \\ \llbracket \tilde{P}(n) \rrbracket \geq \tilde{b}}} \alpha_{[n/x]\sigma}(v(n)) \\ \gamma_{x:\text{int}[\tilde{P}] \rightarrow \sigma}(v)(n) &:= \gamma_{[n/x]\sigma}(v(\llbracket \tilde{P}(n) \rrbracket)). \end{aligned}$$

□

We first check well-definedness.

Lemma 1. *Let σ be a closed abstraction type and assume $\sigma = \sigma_1 \rightarrow \sigma_2$ or $x : \text{int}[\tilde{P}] \rightarrow \sigma_0$. Then $\alpha_\sigma(v)$ and $\gamma_\sigma(v)$ are monotone.*

Proof. The only nontrivial case is $\alpha_{x:\text{int}[\tilde{P}] \rightarrow \sigma_0}(v)$. Assume $\tilde{b} \leq \tilde{b}'$. It suffices to show that

$$\prod_{\substack{n \in \mathbb{Z} \\ \llbracket \tilde{P}(n) \rrbracket \geq \tilde{b}}} \alpha_{[n/x]\sigma}(v(n)) \leq \prod_{\substack{n \in \mathbb{Z} \\ \llbracket \tilde{P}(n) \rrbracket \geq \tilde{b}'}} \alpha_{[n/x]\sigma}(v(n)),$$

which follows from the fact that $\llbracket \tilde{P}(n) \rrbracket \geq \tilde{b}'$ implies $\llbracket \tilde{P}(n) \rrbracket \geq \tilde{b}$. □

We prove some basic properties.

Lemma 2. *α_σ and γ_σ are monotonic for every closed abstraction type σ .*

Proof. Easy induction on the structure of σ . □

Lemma 3. *For every closed abstraction type σ ,*

$$(\forall v \in \mathbb{D}_{\sigma^\flat})(\forall w \in \mathbb{D}_{\sigma^\sharp}) [v \sqsubseteq_{\sigma^\flat} \alpha_\sigma(w) \iff \gamma_\sigma(v) \sqsubseteq_{\sigma^\sharp} w].$$

Proof. By induction on σ .

– Case $\sigma = \bullet$: Trivial.

– Case $\sigma = \sigma_1 \rightarrow \sigma_2$: Let $v \in \mathbb{D}_{\sigma_1^b \rightarrow \sigma_2^b}$ and $w \in \mathbb{D}_{\sigma_1^\# \rightarrow \sigma_2^\#}$. Then

$$\begin{aligned}
v \sqsubseteq_{\sigma^b} \alpha_\sigma(w) &\iff (\forall x \in \mathbb{D}_{\sigma_1^b}) [v(x) \sqsubseteq_{\sigma_2^b} \alpha_{\sigma_2}(w(\gamma_{\sigma_1}(x)))] \\
&\iff (\forall x \in \mathbb{D}_{\sigma_1^b}) [\gamma_{\sigma_2}(v(x)) \sqsubseteq_{\sigma_2^\#} w(\gamma_{\sigma_1}(x))] \\
&\implies (\forall y \in \mathbb{D}_{\sigma_1^\#}) [\gamma_{\sigma_2}(v(\alpha_{\sigma_1}(y))) \sqsubseteq_{\sigma_2^\#} w(\gamma_{\sigma_1}(\alpha_{\sigma_1}(y)))] \\
&\implies (\forall y \in \mathbb{D}_{\sigma_1^\#}) [\gamma_{\sigma_2}(v(\alpha_{\sigma_1}(y))) \sqsubseteq_{\sigma_2^\#} w(y)] \\
&\iff \gamma_\sigma(v) \sqsubseteq_{\sigma^\#} w,
\end{aligned}$$

where we use $\gamma_{\sigma_1}(\alpha_{\sigma_1}(x)) \sqsubseteq_{\sigma_1^\#} y$, which follows from the induction hypothesis. The other direction is similar.

– Case $\sigma = x : \text{int}[\tilde{P}] \rightarrow \sigma_0$: Assume that $v \sqsubseteq_{\sigma^b} \alpha_\sigma(w)$. Let n be an arbitrary integer. Then

$$\begin{aligned}
v(\llbracket \tilde{P}(n) \rrbracket) &\sqsubseteq_{\sigma_0^b} \alpha_\sigma(w)(\llbracket \tilde{P}(n) \rrbracket) \\
&= \prod_{\substack{m \in \mathbb{Z} \\ \llbracket \tilde{P}(n) \rrbracket \geq \llbracket \tilde{P}m \rrbracket}} \alpha_{[m/x]\sigma_0}(w(m)) \\
&\sqsubseteq_{\sigma_0^b} \alpha_{[n/x]\sigma_0}(w(n)).
\end{aligned}$$

By the induction hypothesis, we have

$$\gamma_\sigma(v)(n) = \gamma_{[n/x]\sigma_0}(v(\llbracket \tilde{P}(n) \rrbracket)) \sqsubseteq_{\sigma_0^\#} w(n).$$

Since n is arbitrary, we obtain $\gamma_\sigma(v) \sqsubseteq_{\sigma^\#} w$ as desired.

To prove the converse, assume $\gamma_\sigma(v) \sqsubseteq_{\sigma^\#} w$ and let \tilde{b} be a sequence of Boolean values. Let n be an integer such that $\tilde{b} \sqsubseteq \llbracket \tilde{P}(n) \rrbracket$. Then

$$\gamma_{[n/x]\sigma_0}(v(\tilde{b})) \sqsubseteq_{\sigma_0^\#} \gamma_{[n/x]\sigma_0}(v(\llbracket \tilde{P}(n) \rrbracket)) = \gamma_\sigma(v)(n) \sqsubseteq_{\sigma_0^\#} w(n)$$

since $\gamma_{[n/x]\sigma_0}$ and v are monotone. By the induction hypothesis,

$$v(\tilde{b}) \sqsubseteq_{\sigma_0^b} \alpha_{\sigma_0}(w(n)).$$

Since n is an arbitrary integer such that $\tilde{b} \sqsubseteq \llbracket \tilde{P}(n) \rrbracket$,

$$v(\tilde{b}) \sqsubseteq_{\sigma_0^b} \prod_{\substack{n \in \mathbb{Z} \\ \llbracket \tilde{P}(n) \rrbracket \geq \tilde{b}}} \alpha_{[n/x]\sigma_0}(w(n)) = \alpha_\sigma(w)(\tilde{b}).$$

Since \tilde{b} is arbitrary, we obtain

$$v \sqsubseteq_{\sigma^b} \alpha_\sigma(w)$$

as required. □

Let $(\dot{\sqsubseteq}_{\sigma}) \in \mathbb{D}_{\sigma^b} \times \mathbb{D}_{\sigma^\#}$ be a relation defined by

$$\begin{aligned} v \dot{\sqsubseteq}_{\sigma} w & \iff v \sqsubseteq_{\sigma^b} \alpha_{\sigma}(w) \\ & \iff \gamma_{\sigma}(v) \sqsubseteq_{\sigma^\#} w. \end{aligned}$$

This is the criterion of the soundness of the abstraction: we shall show that $\vdash \psi : \sigma \rightsquigarrow \varphi$ implies $\llbracket \varphi \rrbracket \dot{\sqsubseteq}_{\sigma} \llbracket \psi \rrbracket$, which immediately leads to the soundness of the predicate abstraction (i.e. $\llbracket \varphi \rrbracket = \#$ implies $\llbracket \psi \rrbracket = \#$).

We extend the relation $\dot{\sqsubseteq}$ to valuations. Let Σ be an abstraction type environment and Θ be a sequence of predicates. For valuations χ and ρ of $(\Sigma \mid \Theta)^b$ and $(\Sigma \mid \Theta)^\#$, respectively, we write $\chi \dot{\sqsubseteq}_{\Sigma, \Theta} \rho$ if

- $\chi(x) \dot{\sqsubseteq}_{\rho(\sigma)} \rho(x)$ for all $x : \sigma \in \Sigma$, and
- $\chi(b_P) \sqsubseteq_{\bullet} \llbracket P \rrbracket_{\rho}$ for all $P \in \Theta$.

Here $\rho(\sigma)$ is obtained by replacing all free variables x in σ with $\rho(x)$ (note that free variables in σ are integers).

Lemma 4. $f \dot{\sqsubseteq}_{\sigma_1 \rightarrow \sigma_2} g$ if and only if $f(x) \dot{\sqsubseteq}_{\sigma_2} g(y)$ for every $x \dot{\sqsubseteq}_{\sigma_1} y$.

Proof. Assume that $f \dot{\sqsubseteq}_{\sigma_1 \rightarrow \sigma_2} g$ and $x \dot{\sqsubseteq}_{\sigma_1} y$. Hence

$$\gamma_{\sigma_1 \rightarrow \sigma_2}(f) \sqsubseteq_{(\sigma_1 \rightarrow \sigma_2)^\#} g \quad \text{and} \quad \gamma_{\sigma_1}(x) \sqsubseteq_{\sigma_1^\#} y.$$

We have

$$\begin{aligned} \gamma_{\sigma_2}(f(x)) & \sqsubseteq \gamma_{\sigma_2}(f(\alpha_{\sigma_1}(\gamma_{\sigma_1}(x)))) \\ & = \gamma_{\sigma_1 \rightarrow \sigma_2}(f)(\gamma_{\sigma_1}(x)) \\ & \sqsubseteq g(\gamma_{\sigma_1}(x)) \\ & \sqsubseteq g(y). \end{aligned}$$

To prove the converse, assume that $x \dot{\sqsubseteq}_{\sigma_1} y$ implies $f(x) \dot{\sqsubseteq}_{\sigma_2} g(y)$. For every $y \in \mathbb{D}_{\sigma_1^\#}$, one has $\alpha_{\sigma_1}(y) \sqsubseteq_{\sigma_1^b} \alpha_{\sigma_1}(y)$ and thus $\alpha_{\sigma_1}(y) \dot{\sqsubseteq}_{\sigma_1} y$. So $f(\alpha_{\sigma_1}(y)) \dot{\sqsubseteq}_{\sigma_2} g(y)$. Hence

$$\gamma_{\sigma_1 \rightarrow \sigma_2}(f)(y) = \gamma_{\sigma_2}(f(\alpha_{\sigma_1}(y))) \sqsubseteq_{\sigma_2^\#} g(y).$$

Since y is arbitrary, $\gamma_{\sigma_1 \rightarrow \sigma_2}(f) \sqsubseteq_{(\sigma_1 \rightarrow \sigma_2)^\#} g$ as required. \square

Before the proof of the soundness, we prepare lemmas to prove soundness of (A-NU) and (A-COERCE), respectively.

Lemma 5. If $f \dot{\sqsubseteq}_{\sigma \rightarrow \sigma} g$, then $\text{gfp}(f) \dot{\sqsubseteq}_{\sigma} \text{gfp}(g)$.

Proof. Let $f \in \mathbb{D}_{\sigma^b \rightarrow \sigma^b}$ and $g \in \mathbb{D}_{\sigma^\# \rightarrow \sigma^\#}$ and assume that $f \dot{\sqsubseteq}_{\sigma \rightarrow \sigma} g$. Since $\mathbb{D}_{\sigma^\#}$ is a complete lattice, we have

$$\text{gfp}(g) = \bigsqcup \{ y \mid y \sqsubseteq_{\sigma^\#} g(y) \}$$

by the Knaster-Tarski theorem.

Let x be an arbitrary fixed-point of f , i.e. $f(x) = x$. Then

$$x = f(x) \sqsubseteq_{\sigma^b} \alpha_{\sigma \rightarrow \sigma}(g)(x) = \alpha_{\sigma}(g(\gamma_{\sigma}(x))).$$

Hence

$$\gamma_{\sigma}(x) \sqsubseteq_{\sigma^{\#}} g(\gamma_{\sigma}(x)).$$

So

$$\gamma_{\sigma}(x) \sqsubseteq_{\sigma^{\#}} \bigsqcup \{y \mid y \sqsubseteq_{\sigma^{\#}} g(y)\} = \text{gfp}(g),$$

that means, $x \dot{\sqsubseteq}_{\sigma} \text{gfp}(g)$. Since x is an arbitrary fixed-point of f , we have $\text{gfp}(f) \dot{\sqsubseteq}_{\sigma} \text{gfp}(g)$. \square

Lemma 6. *Assume that $\Sigma \vdash \varphi_1 : (\Theta_1, \sigma_1) \preceq (\Theta_2, \sigma_2) \rightsquigarrow \varphi_2$ and that $\chi_0 \sqsubseteq_{\Sigma, \varepsilon} \rho$. If $\chi_0 \uplus \chi_2 \sqsubseteq_{\Sigma, \Theta_2} \rho$, there exists χ_1 such that $\chi_0 \uplus \chi_1 \dot{\sqsubseteq}_{\Sigma, \Theta_1} \rho$ and $\gamma_{\rho(\sigma_2)}(\llbracket \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2}) \sqsubseteq \gamma_{\rho(\sigma_1)}(\llbracket \varphi_1 \rrbracket_{\chi_0 \uplus \chi_1})$. Furthermore, for every $P \in \Theta_1$, the value $\chi_1(b_P)$ depends only on the restriction of ρ to the free variables of P (in particular, on the restriction of ρ to integer variables).*

Proof. By induction on the structure of the derivation $\Sigma \vdash \varphi_1 : (\Theta_1, \sigma_1) \preceq (\Theta_2, \sigma_2) \rightsquigarrow \varphi_2$.

– (AC-BASE). Then the last rule is the following.

$$\frac{X : \bullet^k \rightarrow \bullet \vdash \xi : \bullet^l \rightarrow \bullet \quad \llbracket X P_1 \dots P_k \rrbracket_{\rho \uplus \{X \mapsto v\}} \sqsupseteq \llbracket \xi Q_1 \dots Q_l \rrbracket_{\rho \uplus \{X \mapsto v\}} \text{ for all } \rho \text{ for } \Sigma^{\#} \text{ and } v \in \mathbb{D}_{\bullet^k \rightarrow \bullet}}{\Sigma \vdash \varphi : (P_1, \dots, P_k, \bullet) \preceq (Q_1, \dots, Q_l, \bullet) \rightsquigarrow [\lambda \widetilde{b}_P. \varphi / X] \xi \widetilde{b}_Q}$$

Assume $\chi_0 \dot{\sqsubseteq}_{\Sigma, \varepsilon} \rho$ and $\chi_0 \uplus \chi_2 \dot{\sqsubseteq}_{\Sigma, \widetilde{Q}} \rho$. Let χ_1 be a valuation defined by

$$\chi_1(b_{P_i}) := \llbracket P_i \rrbracket_{\rho}.$$

Then $\chi_0 \uplus \chi_1 \dot{\sqsubseteq}_{\Sigma, \widetilde{P}} \rho$ and χ_1 satisfies the dependency requirement. We have

$$\begin{aligned} \gamma_{\bullet}(\llbracket [\lambda \widetilde{b}_P. \varphi / X] \xi \widetilde{b}_Q \rrbracket_{\chi_0 \uplus \chi_2}) &= \llbracket [\lambda \widetilde{b}_P. \varphi / X] \xi \widetilde{b}_Q \rrbracket_{\chi_0 \uplus \chi_2} \\ &= \llbracket [\lambda \widetilde{b}_P. \varphi / X] \xi \rrbracket_{\chi_0}(\chi_2(\widetilde{b}_Q)) \\ &\sqsubseteq \llbracket [\lambda \widetilde{b}_P. \varphi / X] \xi \rrbracket_{\chi_0}(\llbracket \widetilde{Q} \rrbracket_{\rho}) \\ &\sqsubseteq \llbracket \xi \rrbracket_{\{X \mapsto \llbracket \lambda \widetilde{b}_P. \varphi \rrbracket_{\chi_0}\}}(\llbracket \widetilde{Q} \rrbracket_{\rho}) \\ &= \llbracket \xi \widetilde{Q} \rrbracket_{\rho \uplus \{X \mapsto \llbracket \lambda \widetilde{b}_P. \varphi \rrbracket_{\chi_0}\}} \\ &= \llbracket X \widetilde{P} \rrbracket_{\rho \uplus \{X \mapsto \llbracket \lambda \widetilde{b}_P. \varphi \rrbracket_{\chi_0}\}} \\ &= \llbracket \lambda \widetilde{b}_P. \varphi \rrbracket_{\chi_0}(\llbracket \widetilde{P} \rrbracket_{\rho}) \\ &= \llbracket \varphi \rrbracket_{\chi_0 \uplus \chi_1} \\ &= \gamma_{\bullet}(\llbracket \varphi \rrbracket_{\chi_0 \uplus \chi_1}). \end{aligned}$$

– (AC-INTARROW). The last rule is the following.

$$\frac{\Sigma, x : \text{int} \vdash \varphi_1 \widetilde{b}_P : ((\Theta_1, \widetilde{P}), \sigma_1) \preceq ((\Theta_2, \widetilde{Q}), \sigma_2) \rightsquigarrow \varphi_2}{\Sigma \vdash \varphi_1 : (\Theta_1, x : \text{int}[\widetilde{P}] \rightarrow \sigma_1) \preceq (\Theta_2, x : \text{int}[\widetilde{Q}] \rightarrow \sigma_2) \rightsquigarrow \lambda \widetilde{b}_Q \cdot \varphi_2}$$

Assume that $\chi_0 \dot{\sqsubseteq}_{\Sigma, \varepsilon} \rho$ and $\chi_0 \uplus \chi_2 \dot{\sqsubseteq}_{\Sigma, \Theta_2} \rho$. Let n be an arbitrary integer. Then $\chi_0 \dot{\sqsubseteq}_{(\Sigma, x : \text{int}), \varepsilon} \rho \uplus \{x \mapsto n\}$

$$\chi_0 \uplus \chi_2 \uplus \{b_Q \mapsto \llbracket Q \rrbracket_{\rho \uplus \{x \mapsto n\}} \mid Q \in \widetilde{Q}\} \dot{\sqsubseteq}_{(\Sigma, x : \text{int}), (\Theta_2, \widetilde{Q})} \rho \uplus \{x \mapsto n\}.$$

By the induction hypothesis, there exists χ_1 such that $\chi_0 \uplus \chi_1 \dot{\sqsubseteq}_{(\Sigma, x : \text{int}), (\Theta_2, \widetilde{Q})} \rho \uplus \{x \mapsto n\}$ and

$$\gamma_{\rho(\sigma_2)}(\llbracket \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2 \uplus \{b_Q \mapsto \llbracket Q \rrbracket_{\rho \uplus \{x \mapsto n\}} \mid Q \in \widetilde{Q}\}}) \sqsubseteq \gamma_{\rho(\sigma_1)}(\llbracket \varphi_1 \widetilde{b}_P \rrbracket_{\chi_0 \uplus \chi_1}).$$

Let $\chi_{1,1}$ and $\chi_{1,2}$ be the restriction of χ_1 to Θ_1 and \widetilde{P} , respectively. We claim that $\chi_{1,1}$ satisfies the requirements. Obviously $\chi_0 \uplus \chi_{1,1} \dot{\sqsubseteq}_{\Sigma, \Theta_2} \rho$ and $\chi_{1,1}$ satisfies the requirement on the dependency. It suffices to show that

$$\gamma_{\rho(x : \text{int}[\widetilde{Q}] \rightarrow \sigma_2)}(\llbracket \lambda \widetilde{b}_P \cdot \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2})(n) \sqsubseteq \gamma_{\rho(x : \text{int}[\widetilde{Q}] \rightarrow \sigma_1)}(\llbracket \varphi_1 \rrbracket_{\chi_0 \uplus \chi_{1,1}})(n)$$

since n is arbitrary and χ_1 is independent of n . We actually have

$$\begin{aligned} & \gamma_{\rho(x : \text{int}[\widetilde{Q}] \rightarrow \sigma_2)}(\llbracket \lambda \widetilde{b}_P \cdot \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2})(n) \\ &= \gamma_{\rho(\sigma_2)}\left(\llbracket \lambda \widetilde{b}_Q \cdot \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2}(\llbracket Q \rrbracket_{\rho \uplus \{x \mapsto n\}})\right) \\ &= \gamma_{\rho(\sigma_2)}\left(\llbracket \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2 \uplus \{b_Q \mapsto \llbracket Q \rrbracket_{\rho \uplus \{x \mapsto n\}} \mid Q \in \widetilde{Q}\}}\right) \\ &\sqsubseteq \gamma_{\rho(\sigma_1)}\left(\llbracket \varphi_1 \widetilde{b}_P \rrbracket_{\chi_0 \uplus \chi_{1,1} \uplus \chi_{1,2}}\right) \\ &\sqsubseteq \gamma_{\rho(\sigma_1)}\left(\llbracket \varphi_1 \widetilde{b}_P \rrbracket_{\chi_0 \uplus \chi_{1,1} \uplus \{b_P \mapsto \llbracket P \rrbracket_{\rho \uplus \{x \mapsto n\}} \mid P \in \widetilde{P}\}}\right) \\ &= \gamma_{\rho(\sigma_1)}\left(\llbracket \varphi_1 \rrbracket_{\chi_0 \uplus \chi_{1,1}}(\llbracket \widetilde{P} \rrbracket_{\rho \uplus \{x \mapsto n\}})\right) \\ &= \gamma_{\rho(x : \text{int}[\widetilde{P}] \rightarrow \sigma_1)}(\llbracket \varphi_1 \rrbracket_{\chi_0 \uplus \chi_{1,1}})(n) \end{aligned}$$

as required.

– (AC-ARROW). Then the last rule is the following.

$$\frac{\begin{array}{l} \Sigma, x : \sigma_{2,1} \vdash x : (\varepsilon, \sigma_{2,1}) \preceq (\Theta_2, \sigma_{1,1}) \rightsquigarrow \xi \\ \Sigma, y : \sigma_{1,1} \vdash \varphi_1 y : (\Theta_1, \sigma_{1,2}) \preceq (\Theta_2, \sigma_{2,2}) \rightsquigarrow \varphi_2 \end{array}}{\Sigma \vdash \varphi_1 : (\Theta_1, \sigma_{1,1} \rightarrow \sigma_{1,2}) \preceq (\Theta_2, \sigma_{2,1} \rightarrow \sigma_{2,2}) \rightsquigarrow \lambda x. [\xi/y] \varphi_2}$$

Assume that $\chi_0 \dot{\sqsubseteq}_{\Sigma, \varepsilon} \rho$ and $\chi_0 \uplus \chi_2 \dot{\sqsubseteq}_{\Sigma, \Theta_2} \rho$. Let v be an arbitrary element in $\mathbb{D}_{\sigma_{1,2}}^{\#} = \mathbb{D}_{\sigma_{2,1}}^{\#}$. Then

$$\chi_0 \uplus \{y \mapsto \alpha_{\rho(\sigma_{1,1})}(v)\} \dot{\sqsubseteq}_{(\Sigma, y : \sigma_{1,1}), \varepsilon} \rho \uplus \{y \mapsto v\}$$

and

$$(\chi_0 \uplus \{y \mapsto \alpha_{\rho(\sigma_{1,1})}(v)\}) \uplus \chi_2 \stackrel{\dot{\sqsubseteq}}{\sqsubseteq} (\Sigma, y: \sigma_{1,1}, \Theta_2) \rho \uplus \{y \mapsto v\}.$$

By the induction hypothesis, there exists χ_1 such that

- $(\chi_0 \uplus \{y \mapsto \alpha_{\rho(\sigma_{1,1})}(v)\}) \uplus \chi_1 \stackrel{\dot{\sqsubseteq}}{\sqsubseteq} (\Sigma, y: \sigma_{1,1}, \Theta_1) \rho \uplus \{y \mapsto w\}$, and
- $\gamma_{\rho(\sigma_{2,2})}(\llbracket \varphi_2 \rrbracket_{\chi_0 \uplus \{y \mapsto \alpha_{\rho(\sigma_{1,1})}(v)\}} \uplus \chi_2) \sqsubseteq \gamma_{\rho(\sigma_{1,2})}(\llbracket \varphi_1 y \rrbracket_{\chi_0 \uplus \{y \mapsto \alpha_{\rho(\sigma_{1,1})}(v)\}} \uplus \chi_1)$.

By a similar argument using the induction hypothesis to the first premise, we also have

$$\begin{aligned} \gamma_{\rho(\sigma_{1,1})}(\llbracket \xi \rrbracket_{\chi_0 \uplus \{x \mapsto \alpha_{\rho(\sigma_{2,1})}(v)\}} \uplus \chi_2) &\sqsubseteq \gamma_{\rho(\sigma_{2,1})}(\llbracket x \rrbracket_{\chi_0 \uplus \{x \mapsto \alpha_{\rho(\sigma_{2,1})}(v)\}}) \\ &= \gamma_{\rho(\sigma_{2,1})}(\alpha_{\rho(\sigma_{2,1})}(v)) \\ &\sqsubseteq v \end{aligned}$$

and thus

$$\llbracket \xi \rrbracket_{\chi_0 \uplus \{x \mapsto \alpha_{\rho(\sigma_{2,1})}(v)\}} \uplus \chi_2 \sqsubseteq \alpha_{\rho(\sigma_{1,1})}(v).$$

Obviously $\chi_0 \uplus \chi_1 \stackrel{\dot{\sqsubseteq}}{\sqsubseteq} \Sigma, \Theta_1 \rho$ and the dependency requirement for χ_1 follows from the same condition for the induction hypothesis. It suffices to show that

$$\gamma_{\rho(\sigma_{2,1} \rightarrow \sigma_{2,2})}(\llbracket \lambda x. [\xi/y] \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2})(v) \sqsubseteq \gamma_{\rho(\sigma_{1,1} \rightarrow \sigma_{1,2})}(\llbracket \varphi_1 \rrbracket_{\chi_0 \uplus \chi_1})(v)$$

since v is arbitrary and χ_1 is independent of v . We actually have

$$\begin{aligned} &\gamma_{\rho(\sigma_{2,1} \rightarrow \sigma_{2,2})}(\llbracket \lambda x. [\xi/y] \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2})(v) \\ &= \gamma_{\rho(\sigma_{2,2})}(\llbracket \lambda x. [\xi/y] \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2}(\alpha_{\rho(\sigma_{2,1})}(v))) \\ &= \gamma_{\rho(\sigma_{2,2})}(\llbracket [\xi/y] \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2 \uplus \{x \mapsto \alpha_{\rho(\sigma_{2,1})}(v)\}}) \\ &= \gamma_{\rho(\sigma_{2,2})}(\llbracket \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2 \uplus \{y \mapsto \llbracket \xi \rrbracket_{\chi_0 \uplus \chi_2 \uplus \{x \mapsto \alpha_{\rho(\sigma_{2,1})}(v)\}}) \\ &\sqsubseteq \gamma_{\rho(\sigma_{2,2})}(\llbracket \varphi_2 \rrbracket_{\chi_0 \uplus \chi_2 \uplus \{y \mapsto \alpha_{\rho(\sigma_{1,1})}(v)\}}) \\ &\sqsubseteq \gamma_{\rho(\sigma_{1,2})}(\llbracket \varphi_1 y \rrbracket_{\chi_0 \uplus \{y \mapsto \alpha_{\rho(\sigma_{1,1})}(v)\}} \uplus \chi_1) \\ &= \gamma_{\rho(\sigma_{1,2})}(\llbracket \varphi_1 \rrbracket_{\chi_0 \uplus \chi_1}(\alpha_{\rho(\sigma_{1,1})}(v))) \\ &= \gamma_{\rho(\sigma_{1,1} \rightarrow \sigma_{2,2})}(\llbracket \varphi_1 \rrbracket_{\chi_0 \uplus \chi_1})(v) \end{aligned}$$

as desired. □

Lemma 7. *Suppose that $\Sigma \vdash \varphi : (\tilde{P}, \sigma) \preceq (\tilde{Q}, \sigma') \rightsquigarrow \varphi'$. Let $v'_1, \dots, v'_i \in \{t, ff\}$ and assume $v'_i \sqsubseteq_{\bullet} \llbracket Q_i \rrbracket_{\rho}$ for each i . There exist $v_1, \dots, v_k \in \{t, ff\}$ such that*

- $v_i \sqsubseteq_{\bullet} \llbracket P_i \rrbracket_{\rho}$ for each i and
- $\gamma_{\sigma'} \left(\llbracket (\Sigma \mid \tilde{Q})^b \vdash \varphi' : \sigma'^b \rrbracket_{\rho \cup \{\tilde{b}_Q \mapsto \tilde{v}'\}} \right) \sqsubseteq_{\sigma^\#} \gamma_{\sigma} \left(\llbracket (\Sigma \mid \tilde{P})^b \vdash \varphi : \sigma^b \rrbracket_{\rho \cup \{\tilde{b}_P \mapsto \tilde{v}\}} \right)$.

Proof. Induction on the derivation with case analysis on the last rule used. We show only the case of (AC-BASE) since other cases are easy. Thus, required conditions hold. □

Now we can describe the main lemma.

Lemma 8. *If $\Sigma \mid \Theta \vdash \psi : \sigma \rightsquigarrow \varphi$ and $\chi \dot{\sqsubseteq}_{\Sigma, \Theta} \rho$, then $\llbracket \varphi \rrbracket_{\chi} \dot{\sqsubseteq}_{\rho(\sigma)} \llbracket \psi \rrbracket_{\rho}$.*

Proof. By induction on the structure of the derivation $\Sigma \mid \Theta \vdash \psi : \sigma \rightsquigarrow \varphi$.

- (A-VAR) and (A-PRED). By the definition of $\dot{\sqsubseteq}_{\Sigma, \Theta}$.
- (A-APP), (A-ABS), (A-AND), (A-OR). By straightforward application of the induction hypothesis. We use Lemma 4 for (A-APP) and (A-ABS).
- (A-NU). By Lemma 5.
- (A-INTAPP). The last rule of the derivation is the following:

$$\frac{\Sigma \mid \Theta \vdash \psi : (x : \text{int}[P_1, \dots, P_k] \rightarrow \sigma) \rightsquigarrow \varphi \quad \Sigma \vdash_{\text{ST}} a : \text{int}}{\Sigma \mid \Theta, [a/x]P_1, \dots, [a/x]P_k \vdash \psi a : [a/x]\sigma \rightsquigarrow \varphi b_{[a/x]P_1} \dots b_{[a/x]P_k}}$$

Assume that $\chi \dot{\sqsubseteq}_{\Sigma, (\Theta, [a/x]P_1, \dots, [a/x]P_k)} \rho$. Let χ' be the restriction of χ to $(\Sigma \mid \Theta)^{\flat}$; then $\chi' \dot{\sqsubseteq}_{\Sigma, \Theta} \rho$. By the induction hypothesis, $\llbracket \varphi \rrbracket_{\chi'} \dot{\sqsubseteq}_{\rho(x:\text{int}[\tilde{P}] \rightarrow \sigma)} \llbracket \psi \rrbracket_{\rho}$, i.e. $\gamma_{\rho(x:\text{int}[\tilde{P}] \rightarrow \sigma)}(\llbracket \varphi \rrbracket_{\chi'}) \sqsubseteq \llbracket \psi \rrbracket_{\rho}$. Therefore

$$\begin{aligned} \gamma_{\rho([a/x]\sigma)}(\llbracket \varphi b_{[a/x]P_1} \dots b_{[a/x]P_k} \rrbracket_{\chi}) &= \gamma_{\rho([a/x]\sigma)}(\llbracket \varphi \rrbracket_{\chi'}(\widetilde{\llbracket b_{[a/x]P} \rrbracket}_{\chi})) \\ &\sqsubseteq \gamma_{\rho([a/x]\sigma)}(\llbracket \varphi \rrbracket_{\chi'}(\llbracket [a/x]\tilde{P} \rrbracket_{\rho})) \\ &= \gamma_{\rho(x:\text{int}[\tilde{P}] \rightarrow \sigma)}(\llbracket \varphi \rrbracket_{\chi'})(\llbracket a \rrbracket_{\rho}) \\ &\sqsubseteq \llbracket \psi \rrbracket_{\rho}(\llbracket a \rrbracket_{\rho}) \\ &= \llbracket \psi a \rrbracket_{\rho}. \end{aligned}$$

- (A-INTABS). The last rule is the following:

$$\frac{\Sigma, x : \text{int} \mid \Theta, \tilde{P} \vdash \psi : \sigma \rightsquigarrow \varphi}{\Sigma \mid \Theta \vdash \lambda x. \psi : (x : \text{int}[\tilde{P}] \rightarrow \sigma) \rightsquigarrow \widetilde{\lambda b_P. \varphi}}$$

Assume $\chi \dot{\sqsubseteq}_{\Sigma, \Theta} \rho$. The goal is to show

$$\gamma_{\rho(x:\text{int}[\tilde{P}] \rightarrow \sigma)}(\llbracket \widetilde{\lambda b_P. \varphi} \rrbracket_{\chi}) \sqsubseteq_{\text{int} \rightarrow \sigma^{\sharp}} \llbracket \lambda x. \psi \rrbracket_{\rho}.$$

Let n be an arbitrary integer. Then

$$\begin{aligned} \gamma_{\rho(x:\text{int}[\tilde{P}] \rightarrow \sigma)}(\llbracket \widetilde{\lambda b_P. \varphi} \rrbracket_{\chi})(n) &= \gamma_{\rho([n/x]\sigma)}(\llbracket \widetilde{\lambda b_P. \varphi} \rrbracket_{\chi}(\llbracket [n/x]\rho(\tilde{P}) \rrbracket)) \\ &= \gamma_{\rho([n/x]\sigma)}(\llbracket \widetilde{\lambda b_P. \varphi} \rrbracket_{\chi}(\llbracket \tilde{P} \rrbracket_{\rho \cup \{x \mapsto n\}})) \\ &= \gamma_{\rho([n/x]\sigma)}(\llbracket \varphi \rrbracket_{\chi \cup \{b_P \mapsto \llbracket P \rrbracket_{\rho \cup \{x \mapsto n\}}\}}). \end{aligned}$$

Since

$$\chi \cup \{b_P \mapsto \llbracket P \rrbracket_{\rho \cup \{x \mapsto n\}}\} \dot{\sqsubseteq}_{\Sigma, (\Theta, \tilde{P})} \rho \cup \{x \mapsto n\},$$

by applying the induction hypothesis, we obtain

$$\gamma_{\rho([n/x]\sigma)}\left(\llbracket\varphi\rrbracket_{\chi\cup\{\widetilde{b}_P\mapsto\llbracket P\rrbracket_{\rho\cup\{x\mapsto n}\}}}\right) \sqsubseteq \llbracket\psi\rrbracket_{\rho\cup\{x\mapsto n\}}.$$

Therefore

$$\begin{aligned} \gamma_{\rho(x:\text{int}[\widetilde{P}]\rightarrow\sigma)}\left(\llbracket\lambda\widetilde{b}_P.\varphi\rrbracket_{\chi}\right)(n) &\sqsubseteq \gamma_{\rho([n/x]\sigma)}\left(\llbracket\varphi\rrbracket_{\chi\cup\{\widetilde{b}_P\mapsto\llbracket P\rrbracket_{\rho\cup\{x\mapsto n}\}}}\right) \\ &\sqsubseteq \llbracket\psi\rrbracket_{\rho\cup\{x\mapsto n\}} \\ &= \llbracket\lambda x.\psi\rrbracket_{\rho}(n). \end{aligned}$$

Since n is arbitrary, $\gamma_{\rho(x:\text{int}[\widetilde{P}]\rightarrow\sigma)}\left(\llbracket\lambda\widetilde{b}_P.\varphi\rrbracket_{\chi}\right) \sqsubseteq \llbracket\lambda x.\psi\rrbracket_{\rho}$ as required.

– (A-COERCE). The last rule of the derivation is

$$\frac{\Sigma \mid \Theta' \vdash \psi : \sigma' \rightsquigarrow \varphi' \quad \Sigma \vdash \varphi' : (\Theta', \sigma') \preceq (\Theta, \sigma) \rightsquigarrow \varphi}{\Sigma \mid \Theta \vdash \psi : \sigma \rightsquigarrow \varphi}$$

Let χ and ρ be valuations with $\chi \dot{\sqsubseteq}_{\Sigma, \Theta} \rho$. By Lemma 6, there exists χ' such that $\chi' \dot{\sqsubseteq}_{\Sigma, \Theta'} \rho$ and $\gamma_{\rho(\sigma)}(\llbracket\varphi\rrbracket_{\chi}) \sqsubseteq_{\sigma} \gamma_{\rho(\sigma')}(\llbracket\varphi'\rrbracket_{\chi'})$. By induction hypothesis, we have $\llbracket\varphi'\rrbracket_{\chi'} \dot{\sqsubseteq}_{\sigma'} \llbracket\psi\rrbracket_{\rho}$. Thus $\llbracket\varphi\rrbracket_{\chi} \dot{\sqsubseteq}_{\sigma} \llbracket\psi\rrbracket_{\rho}$. \square

Soundness of the predicate abstraction (Theorem 1) is an immediate consequence of the previous lemma.

B Counterexample interpretation

This section introduces a new interpretation of formulas, in which the interpretation of a proposition is the counterexamples of the proposition.

The *counterexample interpretation* is obtained by replacing the interpretation of \bullet , which is $\{\text{ff}, \#\}$ in the standard interpretation, with the set of sets of counterexamples:

$$\mathbb{D}_{\bullet}^C := \mathcal{P}(\{\text{counterexamples}\}), \quad (\sqsubseteq_{\bullet}^C) := \{(X, Y) \mid X \supseteq Y\}.$$

Note that the maximum element with respect to \sqsubseteq_{\bullet}^C is the empty set, which is the denotation of formulas with no counterexample (i.e. valid formulas). The interpretation of other type constructors are unchanged, e.g. $\mathbb{D}_{\widetilde{\tau}\rightarrow\tau}^C$ is the set of monotone functions from $\mathbb{D}_{\widetilde{\tau}}^C$ to \mathbb{D}_{τ}^C . The interpretation of formulas is essentially the same as in the standard interpretation, but the interpretation of boolean connectives are changed:

$$\begin{aligned} \llbracket\varphi_1 \vee \varphi_2\rrbracket_{\rho}^C &:= \{c_1 \vee c_2 \mid c_1 \in \llbracket\varphi_1\rrbracket_{\rho}^C, c_2 \in \llbracket\varphi_2\rrbracket_{\rho}^C\} \\ \llbracket\varphi_1 \wedge \varphi_2\rrbracket_{\rho}^C &:= \{c_1 \wedge * \mid c_1 \in \llbracket\varphi_1\rrbracket_{\rho}^C\} \cup \{* \wedge c_2 \mid c_2 \in \llbracket\varphi_2\rrbracket_{\rho}^C\}. \end{aligned}$$

Proposition 2. *Let φ be a closed proposition. Then $c \in \llbracket\varphi\rrbracket^C$ if and only if $c \triangleright \varphi$.*

Proof (Sketch). The right-to-left direction can be easily shown by induction on the derivation $c \triangleright \varphi$; note that the rewriting rules $(\lambda x.\varphi) \varphi' \tilde{\psi} \longrightarrow [\varphi'/x]\varphi \tilde{\psi}$ and $(\nu x.\varphi) \tilde{\psi} \longrightarrow [\nu x.\varphi/x]\varphi \tilde{\psi}$ preserve the interpretation of propositions.

The proof of the left-to-right direction is a consequence of a game-theoretic characterization of $\llbracket \varphi \rrbracket$; since the fixed-point operations in φ are greatest fixed-points, a refutation of a formula should be finite. That means, if $\llbracket \varphi \rrbracket = \text{ff}$, then its falsehood should be made obvious by unfolding ν finitely many times. Such a refutation induces a counterexample. \square

Recall Proposition 1, which says that a closed formula ψ of type \bullet is valid if and only if ψ has no counterexample. Hence the above proposition leads to the following.

Proposition 3. *Let φ be a closed proposition. Then $\llbracket \varphi \rrbracket = \text{tt}$ if and only if $\llbracket \varphi \rrbracket^C = \emptyset$.*

By this reason, we shall write tt for \emptyset in the sequel.

Recall that a result of predicate abstraction of a proposition φ contains two kinds of boolean operations: those originate from φ and those introduced in the abstraction phase. In order to distinguish them, we shall use $\bar{\wedge}$ and $\bar{\vee}$ for boolean operations introduced in the abstraction phase. A formula possibly having $\bar{\wedge}$ or $\bar{\vee}$ is called an *abstracted formula*. The interpretation of abstracted formulas are given by the rules for the standard formulas and

$$\begin{aligned} \llbracket \varphi \bar{\wedge} \psi \rrbracket_{\rho}^C &:= \llbracket \varphi \rrbracket_{\rho}^C \cup \llbracket \psi \rrbracket_{\rho}^C \\ \llbracket \varphi \bar{\vee} \psi \rrbracket_{\rho}^C &:= \llbracket \varphi \rrbracket_{\rho}^C \cap \llbracket \psi \rrbracket_{\rho}^C. \end{aligned}$$

The corresponding rules is given by:

$$\begin{aligned} &\frac{c \triangleright \varphi_1}{c \triangleright \varphi_1 \bar{\wedge} \varphi_2} \\ &\frac{c \triangleright \varphi_2}{c \triangleright \varphi_1 \bar{\wedge} \varphi_2} \\ &\frac{c \triangleright \varphi_1 \quad c \triangleright \varphi_2}{c \triangleright \varphi_1 \bar{\vee} \varphi_2} \end{aligned}$$

Lemma 9. *If a closed abstracted formula φ has a counterexample c such that $\mathcal{C}(c) = \{d\}$, then $d \in \llbracket \varphi \rrbracket^C$.*

Proof. By induction of the derivation of $c \triangleright \varphi$. \square

C Refinement intersection type system for $\nu\text{HFL}_{\mathbb{Z}}$

We define a refinement type system for $\nu\text{HFL}_{\mathbb{Z}}$, which is a variant of a refinement intersection type system for higher-order functional language [18], and show that the soundness of the type system.

C.1 Type System

The set of *refinement types* is defined by the following grammar:

(prime type)	$\delta ::= t \mid \neg c \mid x : \{v : \mathbf{int} \mid \Phi\} \rightarrow \delta \mid \theta \rightarrow \delta$
(intersection type)	$\theta ::= \delta_1 \wedge \cdots \wedge \delta_k$
(extended type)	$\bar{\theta} ::= \mathbf{int} \mid \theta$
(formula)	$\Phi ::= \mathbf{true} \mid \mathbf{false} \mid p(\bar{a}) \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2$
(environment)	$\Delta ::= t \mid \Delta, x : \bar{\theta}$

We regard types defined above as refinements of abstraction types.¹² We write $\Phi ::= [P_1, \dots, P_n]$ if atomic propositions in Φ are in $\{P_1, \dots, P_n\}$. For a dependent refinement type δ and an abstraction type σ , the *refinement relation* $\delta :: \sigma$ is defined by the following rules:

$$\frac{}{t :: \bullet}$$

$$\frac{}{\neg c :: \bullet}$$

$$\frac{\Phi \text{ is a positive boolean combination of } \{P_1, \dots, P_n\} \quad \delta :: \sigma}{(x : \{v : \mathbf{int} \mid \Phi\} \rightarrow \delta) :: (x : \mathbf{int}[P_1, \dots, P_n] \rightarrow \sigma)}$$

$$\frac{\theta :: \sigma \quad \delta' :: \sigma'}{(\theta \rightarrow \delta') :: (\sigma \rightarrow \sigma')}$$

$$\frac{\delta_1 :: \sigma \quad \dots \quad \delta_n :: \sigma}{(\delta_1 \wedge \cdots \wedge \delta_n) :: \sigma}$$

We write $x : \Phi \rightarrow \delta$ for the abbreviation of $x : \{v : \mathbf{int} \mid [v/x]\Phi\} \rightarrow \delta$. Note that the syntax of formula Φ is the same as that of the predicate P and thus compatible. In the intersection $\delta_1 \wedge \cdots \wedge \delta_k$, each δ_i is a refinement of the same abstraction type.

The typing relation $\Sigma \mid \Phi \vdash \psi : \delta$ is defined in Figure 6. This relation means that ψ has refinement type δ under the environment Σ if formula Φ holds. Each rule is self-explanatory.

¹² Since each abstraction type follows the structure of a simple type, one can regard refinement types defined above refine simple types as usual.

$$\begin{array}{c}
 \frac{x : \delta \in \Delta}{\Delta \mid \Phi \vdash x : \delta} \quad (\text{R-VAR}) \\
 \\
 \frac{\models \Phi \implies p(\tilde{a})}{\Delta \mid \Phi \vdash p(\tilde{a}) : \#} \quad (\text{R-PRED}) \\
 \\
 \frac{\Delta \mid \Phi \vdash \psi : (x : \Phi' \rightarrow \delta)}{\Delta \mid \Phi \wedge [a/x]\Phi' \vdash \psi a : [a/x]\delta} \quad (\text{R-INTAPP}) \\
 \\
 \frac{\Delta, x : \mathbf{int} \mid \Phi \wedge \Phi' \vdash \psi : \delta}{\Delta \mid \Phi \vdash \lambda x. \psi : (x : \Phi' \rightarrow \delta)} \quad (\text{R-INTABS}) \\
 \\
 \frac{\Delta \mid \Phi \vdash \psi_1 : \delta_1 \wedge \dots \wedge \delta_k \rightarrow \delta \quad \Delta \mid \Phi \vdash \psi_2 : \delta_i \text{ (for each } i)}{\Delta \mid \Phi \vdash \psi_1 \psi_2 : \delta} \quad (\text{R-APP}) \\
 \\
 \frac{\Delta, x : \delta_1, \dots, x : \delta_k \mid \Phi \vdash \psi : \delta}{\Delta \mid \Phi \vdash \lambda x. \psi : \delta_1 \wedge \dots \wedge \delta_k \rightarrow \delta} \quad (\text{R-ABS}) \\
 \\
 \frac{\Delta \mid \Phi_1 \vdash \psi_1 : \# \quad \Delta \mid \Phi_2 \vdash \psi_2 : \#}{\Delta \mid \Phi_1 \wedge \Phi_2 \vdash \psi_1 \wedge \psi_2 : \#} \quad (\text{R-ANDT}) \\
 \\
 \frac{\Delta \mid \Phi \vdash \psi_1 : c_1}{\Delta \mid \Phi \vdash \psi_1 \wedge \psi_2 : \neg(c_1 \wedge *)} \quad (\text{R-ANDC1}) \\
 \\
 \frac{\Delta \mid \Phi \vdash \psi_2 : \neg c_2}{\Delta \mid \Phi \vdash \psi_1 \wedge \psi_2 : \neg(* \wedge c_2)} \quad (\text{R-ANDC1}) \\
 \\
 \frac{\Delta \mid \Phi_1 \vdash \psi_1 : \# \quad \Delta \mid \Phi_2 \vdash \psi_2 : \#}{\Delta \mid \Phi_1 \vee \Phi_2 \vdash \psi_1 \vee \psi_2 : \#} \quad (\text{R-ORT}) \\
 \\
 \frac{\Delta \mid \Phi_1 \vdash \psi_1 : \neg c_1 \quad \Delta \mid \Phi_2 \vdash \psi_2 : \neg c_2}{\Delta \mid \Phi_1 \vee \Phi_2 \vdash \psi_1 \vee \psi_2 : \neg(c_1 \vee c_2)} \quad (\text{R-ORC}) \\
 \\
 \frac{\Delta, x : \delta_0, x : \delta_1, \dots, x : \delta_k \mid \Phi \vdash \psi : \delta_i \text{ for every } i = 0, \dots, k}{\Delta \mid \Phi \vdash \nu x. \psi : \delta_0} \quad (\text{R-NU}) \\
 \\
 \frac{\Delta \mid \Phi \vdash \psi : \delta \quad \Delta \mid \Phi \vdash \delta \preceq \delta'}{\Delta \mid \Phi \vdash \psi : \delta'} \quad (\text{R-COERCE}) \\
 \\
 \frac{\Delta \mid \Phi \vdash \psi : \delta_i \text{ (for each } i)}{\Delta \mid \Phi \vdash \psi : \delta_1 \wedge \dots \wedge \delta_k} \quad (\text{R-INTERSECTION}) \\
 \\
 \frac{\delta_1 = \# \text{ or } \delta_1 = \delta_2 = \neg c}{\Delta \mid \Phi \vdash \delta_1 \preceq \delta_2} \quad (\text{RSUB-BASE}) \\
 \\
 \frac{\models \Phi \wedge \Phi'_x \implies \Phi_x \quad \Delta, x : \mathbf{int} \mid \Phi \wedge \Phi'_x \vdash \delta \preceq \delta'}{\Delta \mid \Phi \vdash (x : \Phi_x \rightarrow \delta) \preceq (x : \Phi'_x \rightarrow \delta')} \quad (\text{RSUB-INTARROW}) \\
 \\
 \frac{\Delta \mid \Phi \vdash \theta' \preceq \theta \quad \Delta \mid \Phi \vdash \delta \preceq \delta'}{\Delta \mid \Phi \vdash \theta \rightarrow \delta \preceq \theta' \rightarrow \delta'} \quad (\text{RSUB-ARROW}) \\
 \\
 \frac{\forall i'. \exists i. \Delta \mid \Phi \vdash \delta_i \preceq \delta'_i}{\Delta \mid \Phi \vdash \delta_1 \wedge \dots \wedge \delta_k \preceq \delta'_1 \wedge \dots \wedge \delta'_k} \quad (\text{RSUB-INTERSECTION})
 \end{array}$$

Fig. 6. Typing rules of the refinement type system

C.2 Soundness of refinement intersection type system

We prove the soundness of the type system, i.e. $\vdash \varphi : \neg c$ implies $c \notin \llbracket \varphi \rrbracket^C$ for every closed proposition φ . The proof is based on semantics. For a refinement type environment Δ which refines Σ , a *valuation for Δ* means a valuation for (the corresponding simple-type environment of) Σ .

Definition 2 (Semantics of types). Let δ be a refinement type δ with $\delta :: \tau$ and ρ be a valuation that assigns integers to free variables of δ . The semantics $\llbracket \delta \rrbracket \in \mathbb{D}_\tau^C$ is defined by following equations:

$$\begin{aligned} \llbracket \# \rrbracket_\rho &:= \# \\ \llbracket \neg c \rrbracket_\rho &:= \{ \text{counterexamples} \} \setminus \{ c \} \\ \llbracket x : \Phi \rightarrow \delta \rrbracket_\rho(n) &= \begin{cases} \llbracket \delta \rrbracket_{\rho[x \mapsto n]} & \text{if } \rho[x \mapsto n] \models \Phi \\ \perp & \text{otherwise} \end{cases} \\ \llbracket \theta \rightarrow \delta \rrbracket_\rho(v) &:= \begin{cases} \llbracket \delta \rrbracket_\rho & \text{if } \llbracket \theta \rrbracket_\rho \sqsubseteq v \\ \perp & \text{otherwise} \end{cases} \\ \llbracket \delta_1 \wedge \cdots \wedge \delta_k \rrbracket_\rho &:= \llbracket \delta_1 \rrbracket_\rho \sqcap \cdots \sqcap \llbracket \delta_k \rrbracket_\rho \end{aligned}$$

The set of valuations $\langle \Delta; \Phi \rangle$ is defined by

$$\langle \Delta; \Phi \rangle := \{ \rho \mid (\forall (x : \delta) \in \Delta. \rho(x) \sqsupseteq \llbracket \delta \rrbracket_\rho) \text{ and } \rho \models \Phi \}$$

Lemmas 10 and 11 are justifications of the semantics, relating type judgments $\varphi : \delta$ to a semantic inequation.

Lemma 10. If $\Delta \mid \Phi \vdash \delta \preceq \delta'$ and $\rho \in \langle \Delta; \Phi \rangle$, then $\llbracket \delta \rrbracket_\rho \sqsupseteq \llbracket \delta' \rrbracket_\rho$.

Proof. By induction on the structure of derivation.

- (RSUB-BASE): Easy.
- (RSUB-ARROW): Let $\rho \in \langle \Delta; \Phi \rangle$. By the induction hypothesis, $\llbracket \theta' \rrbracket_\rho \sqsupseteq \llbracket \theta \rrbracket_\rho$ and $\llbracket \delta \rrbracket_\rho \sqsupseteq \llbracket \delta' \rrbracket_\rho$. We prove $\llbracket \theta \rightarrow \delta \rrbracket_\rho \sqsupseteq \llbracket \theta' \rightarrow \delta \rrbracket_\rho$, i.e. $\llbracket \theta \rightarrow \delta \rrbracket_\rho(x) \sqsupseteq \llbracket \theta' \rightarrow \delta \rrbracket_\rho(x)$ for every x . If $x \not\sqsupseteq \llbracket \theta' \rrbracket_\rho$, the inequation trivially holds since $\llbracket \theta' \rightarrow \delta' \rrbracket_\rho(x) = \perp$. Otherwise $x \sqsupseteq \llbracket \theta' \rrbracket_\rho \sqsupseteq \llbracket \theta \rrbracket_\rho$ and thus $\llbracket \theta \rightarrow \delta \rrbracket_\rho(x) = \llbracket \delta \rrbracket_\rho \sqsupseteq \llbracket \delta' \rrbracket_\rho = \llbracket \theta' \rightarrow \delta \rrbracket_\rho(x)$.
- (RSUB-INTARROW):

$$\frac{\models \Phi \wedge \Phi'_x \implies \Phi_x \quad \Delta, x : \mathbf{int} \mid \Phi \wedge \Phi'_x \vdash \delta \preceq \delta'}{\Delta \mid \Phi \vdash (x : \Phi_x \rightarrow \delta) \preceq (x : \Phi'_x \rightarrow \delta')} \text{(RSUB-INTARROW)}$$

Let $\rho \in \langle \Delta; \Phi \rangle$ and n be an integer. We prove that $\llbracket x : \Phi_x \rightarrow \delta \rrbracket_\rho(n) \sqsupseteq \llbracket x : \Phi'_x \rightarrow \delta' \rrbracket_\rho(n)$. If $\rho[x \mapsto n] \not\models \Phi'_x$, the inequation holds since $\llbracket x : \Phi'_x \rightarrow \delta' \rrbracket_\rho(n) = \perp$. Assume $\rho[x \mapsto n] \models \Phi'_x$; then $\rho[x \mapsto n] \in \langle \Delta; x : \mathbf{int}, \Phi \wedge \Phi'_x \rangle$. By the first premise, we have $\rho[x \mapsto n] \models \Phi_x$. By the induction hypothesis, we have

$$\llbracket x : \Phi_x \rightarrow \delta \rrbracket_\rho(n) = \llbracket \delta \rrbracket_{\rho[x \mapsto n]} \sqsupseteq \llbracket \delta' \rrbracket_{\rho[x \mapsto n]} = \llbracket x : \Phi'_x \rightarrow \delta' \rrbracket_\rho(n).$$

- (R_{SUB-INTERSECTION}): Let $\rho \in \langle \Delta; \Phi \rangle$. By the induction hypothesis, for each i' , there exists i such that $\langle \delta_i \rangle_\rho \sqsupseteq \langle \delta'_{i'} \rangle_\rho$. Hence

$$\langle \delta_1 \wedge \dots \wedge \delta_k \rangle_\rho = \langle \delta_1 \rangle_\rho \sqcup \dots \sqcup \langle \delta_k \rangle_\rho \sqsupseteq \langle \delta_i \rangle_\rho \sqsupseteq \langle \delta'_{i'} \rangle_\rho$$

for each i' (where i depends on i'). Since i' is arbitrary, we have

$$\langle \delta_1 \wedge \dots \wedge \delta_k \rangle_\rho \sqsupseteq \langle \delta'_1 \rangle_\rho \sqcup \dots \sqcup \langle \delta'_{k'} \rangle_\rho = \langle \delta'_1 \wedge \dots \wedge \delta'_{k'} \rangle_\rho$$

□

Lemma 11. *If $\Delta \mid \Phi \vdash \psi : \delta$ and $\rho \in \langle \Delta; \Phi \rangle$, then $\llbracket \psi \rrbracket_\rho^C \sqsupseteq \langle \delta \rangle_\rho$.*

Proof. By induction on the derivation with case analysis on the last rule used.

- (R-VAR), (R-PRED). By definition of $\langle \Delta; \Phi \rangle$.
- (R-AND), (R-OR). By simple application of the induction hypothesis.
- (R-APP). Let $\rho \in \langle \Delta; \Phi \rangle$. By the induction hypothesis,

$$\begin{aligned} \llbracket \psi_1 \rrbracket_\rho^C &\sqsupseteq \langle \delta_1 \wedge \dots \wedge \delta_k \rightarrow \delta \rangle_\rho \\ \llbracket \psi_2 \rrbracket_\rho^C &\sqsupseteq \langle \delta_i \rangle_\rho \text{ for each } i. \end{aligned}$$

We have $\llbracket \psi_1 \rrbracket_\rho^C (\llbracket \psi_2 \rrbracket_\rho^C) \sqsupseteq \langle \delta \rangle_\rho$ since $\llbracket \psi_2 \rrbracket_\rho^C \sqsupseteq \langle \delta_1 \wedge \dots \wedge \delta_k \rangle_\rho$.

- (R-ABS).

$$\frac{\Delta, x : \delta_1, \dots, x : \delta_k \mid \Phi \vdash \psi : \delta}{\Delta \mid \Phi \vdash \lambda x. \psi : \delta_1 \wedge \dots \wedge \delta_k \rightarrow \delta} \quad (\text{R-ABS})$$

Let $\rho \in \langle \Delta; \Phi \rangle$ and v be a value with $v \sqsupseteq \langle \delta_1 \wedge \dots \wedge \delta_k \rangle_\rho$. We have

$$\llbracket \lambda x. \psi \rrbracket_\rho^C (v) = \llbracket \psi \rrbracket_{\rho \cup \{x \mapsto v\}}^C \sqsupseteq \langle \delta \rangle_\rho$$

by the induction hypothesis. Since $v \sqsupseteq \langle \delta_1 \wedge \dots \wedge \delta_k \rangle_\rho$ is arbitrary, we have $\llbracket \lambda x. \psi \rrbracket_\rho^C \sqsupseteq \langle \delta_1 \wedge \dots \wedge \delta_k \rightarrow \delta \rangle_\rho$.

- (R-INTAPP). Let $\rho \in \langle \Delta; \Phi \wedge [a/x]\Phi' \rangle \subseteq \langle \Delta; \Phi \rangle$ and $n = \llbracket a \rrbracket_\rho^C$. By the induction hypothesis, we have $\llbracket \psi \rrbracket_\rho^C \sqsupseteq \langle x : \Phi' \rightarrow \delta \rangle_\rho$. Since $\llbracket [a/x]\Phi' \rrbracket_\rho = \llbracket \Phi' \rrbracket_{\rho \cup \{x \mapsto n\}} = \mathbf{true}$, we have $\llbracket \psi a \rrbracket_\rho^C = \llbracket \psi \rrbracket_\rho^C (n) \sqsupseteq \langle \delta \rangle_{\rho[x \mapsto n]} = \langle [a/x]\delta \rangle_\rho$.
- (R-INTABS). Let $\rho \in \langle \Delta; \Phi \rangle$ and n be an integer with $\llbracket \Phi' \rrbracket_{\rho \cup \{x \mapsto n\}}^C = \mathbf{t}$. By the induction hypothesis, we have

$$\llbracket \lambda x. \psi \rrbracket_\rho^C (n) = \llbracket \psi \rrbracket_{\rho[x \mapsto n]}^C \sqsupseteq \langle \delta \rangle_{\rho[x \mapsto n]}.$$

- (R-NU). Let $\rho \in \langle \Delta; \Phi \rangle$ and $v = \langle \delta_0 \wedge \dots \wedge \delta_k \rangle_\rho$. By the induction hypothesis, we have $\llbracket \psi \rrbracket_{\rho[x \mapsto v]}^C \sqsupseteq \langle \delta_i \rangle_\rho$ for every $i = 1, \dots, k$. Hence $\llbracket \psi \rrbracket_{\rho[x \mapsto v]}^C \sqsupseteq \langle \delta_0 \wedge \dots \wedge \delta_k \rangle_\rho = v$. This means that $v \sqsubseteq \llbracket \lambda x. \psi \rrbracket_\rho^C (v)$, and thus $v \sqsubseteq \text{gfp}(\llbracket \lambda x. \psi \rrbracket_\rho^C) = \llbracket \nu x. \psi \rrbracket_\rho^C$.
- (R-COERCE). By Lemma 10.

□

The soundness theorem is an immediate consequence of the previous lemma.

Theorem 4 (Soundness of refinement type system). *Let φ be a closed proposition φ . If $\mathbf{t} \mid \mathbf{true} \vdash \varphi : \neg c$, then $c \notin \llbracket \varphi \rrbracket^C$. If $\mathbf{t} \mid \mathbf{true} \vdash \varphi : \mathbf{t}$, then $\llbracket \varphi \rrbracket^C = \mathbf{t}$, i.e. $\llbracket \varphi \rrbracket = \mathbf{t}$.*

D Proofs of Theorems 2 and 3

Theorem 2 states that, if the validity of ψ is provable in the refinement type system, then there exists an abstraction (following the abstraction types extracted from the refinement type derivation) that is valid. The proof is an adaptation of the proof of a similar statement in Kobayashi et al. [10] for functional programs. Interestingly Progress (Theorem 3) can be proved by the same technique. This is because Theorem 3 can be rephrased as follows: if $c \not\vdash \psi$ (i.e. $c \notin \llbracket \psi \rrbracket^C$) is provable in the refinement intersection type system, then there exists an abstraction φ (following the abstraction types extracted from the derivation) such that $c \notin \llbracket \varphi \rrbracket^C$.

A *formula with abstraction type annotation* is a formula of which each binding variable (i.e. x in $\lambda x.\varphi$ or $\nu x.\varphi$) is annotated by an abstraction type as in $\lambda x^\sigma.\varphi$ and $\nu x^\sigma.\varphi$. We assume an abstraction type σ annotated to a variable x is a refinement of the simple type τ of x . An *abstraction type judgment* is of the form $\Sigma \mid \Theta \vdash \varphi : \sigma$ where φ is a formula with abstract type annotation. We assume that an annotated judgment respects the corresponding simple-type judgment, but do not require further conditions: for example, $x : \sigma \mid \cdot \vdash x : \sigma'$ is a valid annotated judgment even if $\sigma \neq \sigma'$ (provided that both σ and σ' are refinements of the simple type x).

Let $\Theta = (P_1, \dots, P_k)$ be a sequence of predicates. We often abbreviate the sequence $b_{P_1} b_{P_2} \dots b_{P_k}$ of variables as $\widetilde{b_P}$ or b_Θ .

Definition 3 (Type template).

(<i>template</i>)	$C ::= \mathbf{true} \mid \mathbf{false} \mid []_i \mid C_1 \wedge C_2 \mid C_1 \vee C_2$
(<i>type</i>)	$\xi ::= \bullet \mid x : C \rightarrow \xi \mid \chi \rightarrow \xi$
(<i>intersection type</i>)	$\chi ::= \xi_1 \wedge \dots \wedge \xi_k$
(<i>extended type</i>)	$\bar{\chi} ::= \mathbf{int} \mid \chi$
(<i>environment</i>)	$\Xi ::= \emptyset \mid \Xi, x : \bar{\chi}$

Type templates are used to obtain refinement type by combination with abstraction types (template application).

Definition 4 (Template application). *When template C does not have any hole $[]_i$ with $i > k$, the template application $C[\Phi_1, \dots, \Phi_k]$ denotes formula obtained by replacing all $[]_i$ in C with Φ_i . The refinement type $\xi[\sigma]$ and the refinement type environment $\Xi[\Sigma]$ are also defined by recursively applying this*

template applications. Formally,

$$\begin{aligned}
 \bullet[\bullet] &= \bullet \\
 (x : C \rightarrow \xi)[x : \mathbf{int}[\tilde{P}] \rightarrow \sigma] &= x : C[\tilde{P}] \rightarrow \xi[\sigma] \\
 (\chi \rightarrow \xi)[\sigma' \rightarrow \sigma] &= \chi[\sigma'] \rightarrow \xi[\sigma] \\
 (\xi_1 \wedge \dots \wedge \xi_k)[\sigma] &= \xi_1[\sigma] \wedge \dots \wedge \xi_k[\sigma] \\
 \emptyset[\emptyset] &= \emptyset \\
 (\Xi, x : \mathbf{int})[\Sigma, x : \mathbf{int}] &= \Xi[\Sigma], x : \mathbf{int} \\
 (\Xi, x : \xi)[\Sigma, x : \sigma] &= \Xi[\Sigma], x : \xi[\sigma]
 \end{aligned}$$

Definition 5. For a closed template type ξ and a closed abstraction type σ , a set of abstract values $\llbracket \xi \rrbracket \subseteq \mathbb{D}_{\sigma^b}$ is defined by

$$\begin{aligned}
 \llbracket \bullet \rrbracket &:= \{\#\} \\
 \llbracket \emptyset \rrbracket &:= \{\emptyset\} \\
 \llbracket \neg c \rrbracket &:= \{X \subseteq \{\text{counterexamples}\} \mid c \notin X\} \\
 \llbracket C \rightarrow \xi \rrbracket &:= \{g \mid \forall \tilde{v} \in \llbracket C \rrbracket. (g \tilde{v}) \in \llbracket \xi \rrbracket\} \\
 \llbracket \chi \rightarrow \xi \rrbracket &:= \{g \mid \forall v \in \llbracket \chi \rrbracket. (g v) \in \llbracket \xi \rrbracket\} \\
 \llbracket \xi_1 \wedge \dots \wedge \xi_k \rrbracket &:= \llbracket \xi_1 \rrbracket \cap \dots \cap \llbracket \xi_k \rrbracket \\
 \llbracket C \rrbracket &:= \{(v_1, \dots, v_k) \in \{\#, ff\}^k \mid \models C[\tilde{v}]\}.
 \end{aligned}$$

A subset of abstract valuations $\llbracket \Xi \mid C \rrbracket^\Theta$ is defined as

$$\{\rho \mid \models C[\rho(b_\Theta)] \text{ and } \forall (x : \chi) \in \Xi. \rho(x) \in \llbracket \chi \rrbracket\}.$$

We often omit Θ from $\llbracket \Xi \mid C \rrbracket^\Theta$. We write $\Xi \mid C \models e : \xi$ if $\rho \in \llbracket \Xi \mid C \rrbracket$ implies $\llbracket e \rrbracket_\rho^C \in \llbracket \xi \rrbracket$. \square

We shall prove that, if $\Xi[\Sigma] \mid C[\Theta] \vdash \psi : \xi[\sigma]$, then there exists φ that is an abstraction of ψ , i.e.,

$$\Sigma \mid \Theta \vdash \psi : \sigma \rightsquigarrow \varphi$$

such that

$$\Xi \mid C \vdash \psi : \xi.$$

Theorems 2 and 3 are corollaries of this result, applied to refinement intersection type judgments $\vdash \psi : \emptyset$ and $\vdash \psi : \neg c$, respectively.

The proof is by induction on the structure of ψ . Since a subformula ψ' of ψ may appear several times in a derivation of $\Xi[\Sigma] \mid C[\Theta] \vdash \psi : \xi[\sigma]$, associated to several different judgments, say $(\Xi'_i[\Sigma'] \mid C'_i[\Theta'] \vdash \psi' : \xi'_i[\sigma'])_{i \in I}$. This family of judgments that share the same subject and the same abstraction type is the main object in the proof.

We first prepare some lemmas.

Lemma 12. For each ξ , $\llbracket \xi \rrbracket$ is upward closed and closed under meets. That means

- $a \in \llbracket \xi \rrbracket$ and $a \sqsubseteq b$ implies $b \in \llbracket \xi \rrbracket$, and
- $X \subseteq \llbracket \xi \rrbracket$ implies $\sqcup X \in \llbracket \xi \rrbracket$.

A similar statement holds for $\llbracket \chi \rrbracket$.

Proof. Easy. □

Lemma 13. For each ξ , the set $\llbracket \xi \rrbracket$ has the minimum element. A similar statement holds for χ .

Proof. By induction on ξ .

- $\xi = (\neg c)$: Recall that $\llbracket \neg c \rrbracket$ is the set of subsets X of counterexamples such that $c \notin X$. The minimum element is the largest subset, which is $\{\text{counterexamples}\} \setminus \{c\}$.
- $\xi = \chi_1 \rightarrow \xi_2$: Let w be the minimum element of $\llbracket \xi_2 \rrbracket$. Then the minimum element in $\llbracket \chi_1 \rightarrow \xi_2 \rrbracket$ is the mapping

$$v \mapsto \begin{cases} w, & \text{if } v \in \llbracket \chi_1 \rrbracket \\ \perp, & \text{otherwise,} \end{cases}$$

where \perp is the minimum element of the whole domain.

Other cases are similar. □

The following lemma is used to handle the subsumption rule:

$$\frac{\Xi_i[\Sigma] \mid C_i[\Theta] \vdash \psi : \xi_i[\sigma] \quad \models C'_i[\Theta'] \implies C_i[\Theta] \quad C'_i[\Theta'] \vdash \xi_i[\sigma] \preceq \xi'_i[\sigma']}{\Xi_i[\Sigma] \mid C_i[\Theta] \vdash \psi : \xi_i[\sigma]}$$

Lemma 14. Assume that

- $\Xi_i \mid C_i \models \varphi : \xi_i$ for every $i = 1, \dots, k$,
- $C'_i[\Theta'] \vdash \xi_i[\sigma] \preceq \xi'_i[\sigma']$ for every $i = 1, \dots, k$, and
- $\models C'_i[\Theta'] \implies C_i[\Theta]$.

Then there exists φ' such that

- $\Xi_i \mid C'_i \models \varphi' : \xi'_i$ for every $i = 1, \dots, k$, and
- $\Sigma \vdash \varphi : (\Theta, \sigma) \preceq (\Theta', \sigma') \rightsquigarrow \varphi'$.

A similar statement holds for intersections.

Proof. By induction on the structure of σ .

- $\sigma = \bullet$: Then $C'_i[\Theta'] \vdash \xi_i[\sigma] \preceq \xi'_i[\sigma']$ implies $\xi_i = \#$ or $\xi_i = \xi'_i$. Therefore $\llbracket \xi_i \rrbracket \subseteq \llbracket \xi'_i \rrbracket$. Let $\zeta(X)$ be the following formula with free variable X :

$$\zeta(X) = \lambda b_{\Theta'} . \bigvee_{1 \leq i \leq k} \left(\bar{C}'_i[b_{\Theta'}] \bar{\wedge} \bigwedge_{\tilde{v} \text{ such that } \models C_i[\tilde{v}]} X \tilde{v} \right),$$

where \bar{C} is obtained by replacing \wedge and \vee in C with $\bar{\wedge}$ and $\bar{\vee}$. Because $\models C'[\Theta'] \implies C[\Theta]$, we have $\llbracket X \Theta \rrbracket_{\rho} \supseteq \llbracket \zeta \Theta' \rrbracket_{\rho}$ for every valuation ρ for X and free integer variables in Θ, Θ' . Hence, by applying (AC-BASE) to the assumption, we obtain

$$\Sigma \vdash \varphi : (\Theta, \bullet) \preceq (\Theta', \bullet) \rightsquigarrow \zeta(\lambda b_{\Theta}.\psi).$$

We show that $\Xi_i \mid C'_i \models \zeta(\lambda b_{\Theta}.\psi) : \xi'_i$ for every $i = 1, \dots, k$. Assume $1 \leq i \leq k$ and $\rho \in \llbracket \Xi_i \mid C'_i \rrbracket^{\Theta'}$. Then

$$\begin{aligned} \llbracket \zeta(\lambda b_{\Theta}.\psi) \rrbracket_{\rho}^C &\supseteq \llbracket \bar{C}'_i[b_{\Theta'}] \bar{\wedge} \bigwedge_{\tilde{v} \text{ such that } \models_{C_i}[\tilde{v}]} (\lambda b_{\Theta}.\psi) \tilde{v} \rrbracket_{\rho}^C \\ &= \llbracket \bigwedge_{\tilde{v} \text{ such that } \models_{C_i}[\tilde{v}]} (\lambda b_{\Theta}.\psi) \tilde{v} \rrbracket_{\rho}^C \\ &= \prod_{\tilde{v} \text{ such that } \models_{C_i}[\tilde{v}]} \llbracket (\lambda b_{\Theta}.\psi) \tilde{v} \rrbracket_{\rho}^C \\ &= \prod_{\tilde{v} \text{ such that } \models_{C_i}[\tilde{v}]} \llbracket \psi \rrbracket_{\rho[b_{\Theta} \mapsto \tilde{v}]}^C \\ &\in \llbracket \xi_i \rrbracket \\ &\subseteq \llbracket \xi'_i \rrbracket. \end{aligned}$$

– $\sigma = \sigma_1 \rightarrow \sigma_2$: For every i , $\xi_i = (\chi_{i,1} \rightarrow \xi_{i,2})$ and $\xi'_i = (\chi'_{i,1} \rightarrow \xi_{i,2})$. We have

$$C'_i[\Theta'] \vdash \chi'_{i,1}[\sigma'_1] \preceq \chi_{i,1}[\sigma_1]$$

and

$$C'_i[\Theta'] \vdash \xi_{i,2}[\sigma_2] \preceq \xi'_{i,2}[\sigma'_2].$$

Since

$$\Xi_i, x : \chi'_{i,1} \mid \mathbf{true} \models x : \chi'_{i,1}$$

for each i , by applying the induction hypothesis, there exists φ' such that

$$\Xi_i, x : \chi'_{i,1} \mid C'_i \models \varphi' : \chi_{i,1}$$

for every i and

$$\Sigma, x : \sigma'_1 \vdash x : (\epsilon, \sigma'_1) \preceq (\Theta, \sigma_1) \rightsquigarrow \varphi'.$$

Since $\Sigma_i \mid C_i \vdash \varphi : \chi_{i,1} \rightarrow \xi_{i,2}$,

$$\Sigma, y : \chi_{i,1} \mid C_i \vdash \varphi y : \xi_{i,2}$$

for each i . By the induction hypothesis, there exists φ'' such that

$$\Xi_i, y : \chi_{i,1} \mid C'_i \models \varphi'' : \xi'_{i,2}$$

for every i and

$$\Sigma, y : \sigma_1 \vdash \varphi y : (\Theta, \sigma_2) \preceq (\Theta', \sigma'_2) \rightsquigarrow \varphi''.$$

By (AC-INTARROW),

$$\Sigma \vdash \varphi : (\Theta, \sigma_1 \rightarrow \sigma_2) \preceq (\Theta', \sigma'_1 \rightarrow \sigma'_2) \rightsquigarrow \lambda x. [\varphi' / y] \varphi''.$$

Then

$$\Xi_i \mid C'_i \models \lambda x. [\varphi' / y] \varphi'' : \chi'_{i,1} \rightarrow \xi'_{i,2}$$

follows from $\Xi_i, y : \chi_{i,1} \mid C'_i \models \varphi'' : \xi'_{i,2}$ and $\Xi_i, x : \chi'_{i,1} \mid C'_i \models \varphi' : \chi_{i,1}$.

– $\sigma = (x : \text{int}[\tilde{P}] \rightarrow \sigma_0)$: Then $\sigma' = (x : \text{int}[\tilde{P}'] \rightarrow \sigma'_0)$, $\xi_i = (D_i \rightarrow \xi_{i,0})$ and $\xi'_i = (D'_i \rightarrow \xi'_{i,0})$. For every i , we have

$$\models C'_i[\Theta] \wedge D'_i[\tilde{P}'] \Longrightarrow D_i[\tilde{P}]$$

and

$$C'_i[\Theta] \wedge D'_i[\tilde{P}'] \vdash \xi_{i,0} \preceq \xi'_{i,0}.$$

Since $\models C'_i[\Theta'] \Longrightarrow C_i[\Theta]$, we have

$$\models C'_i[\Theta'] \wedge D'_i[\tilde{P}'] \Longrightarrow C_i[\Theta] \wedge D[\tilde{P}].$$

Because

$$\Xi_i, x : \text{int} \mid C_i \wedge D \models \varphi b_\Theta : \xi_{i,0},$$

by the induction hypothesis, there exists φ'' such that

$$\Xi_i, x : \text{int} \mid C'_i \wedge D' \models \varphi' : \xi'_{i,0}$$

for every i and

$$\Sigma, x : \text{int} \vdash \varphi \widetilde{b}_P : ((\Theta, \tilde{P}), \sigma_0) \preceq ((\Theta', \tilde{P}'), \sigma'_0) \rightsquigarrow \varphi'.$$

Hence

$$\Sigma \vdash \varphi : (\Theta, (x : \text{int}[\tilde{P}] \rightarrow \sigma_0)) \preceq (\Theta', (x : \text{int}[\tilde{P}'] \rightarrow \sigma'_0)) \rightsquigarrow \lambda \widetilde{b}_{P'} . \varphi'.$$

Then $\Xi_i \mid C'_i \models \lambda \widetilde{b}_{P'} . \varphi' : D' \rightarrow \xi'_{i,0}$ follows from $\Xi_i, x : \text{int} \mid C'_i \wedge D' \models \varphi' : \xi'_{i,0}$.

We prove the case of intersections. Assume that $\chi_i = \bigwedge_{j \in J_i} \xi_{i,j}$ and $\chi'_i = \bigwedge_{k \in K_i} \xi'_{i,k}$. Then $C'_i[\Theta'] \vdash \chi_i \preceq \chi'_i$ implies that, for each i and $k \in K_i$, there exists $j \in J_i$ such that $C'_i[\Theta'] \vdash \xi_{i,j} \preceq \xi'_{i,k}$. Let us write the mapping $k \mapsto j$ by f_i . Therefore, for each i and $k \in K_i$,

$$C'_i[\Theta'] \vdash \xi_{i,f_i(k)} \preceq \xi'_{i,k}.$$

Since $\Xi_i \mid C'_i \models \varphi : \chi_i$, we have

$$\Xi_i \mid C'_i \models \varphi : \xi_{i,f(k)}$$

for every i and $k \in K_i$. By the induction hypothesis, there exists φ' such that

$$\Xi_i \mid C'_i \models \varphi : \xi'_{i,k}$$

for every i and $k \in K_i$ and

$$\Sigma \vdash \varphi : (\Theta, \sigma) \preceq (\Theta', \sigma') \rightsquigarrow \varphi'.$$

The former implies

$$\Xi_i \mid C'_i \models \varphi : \bigwedge_{k \in K_i} \xi'_{i,k}$$

as desired. \square

The following lemma is mentioned above, relating intersection type derivations and abstractions.

Lemma 15. *Let $(\Xi_i[\Sigma] \mid C_i[\Theta] \vdash \psi : \xi_i[\sigma])_{i \in I}$ be a family of refinement intersection type derivations. Then there exists φ such that*

$$\Sigma \mid \Theta \vdash \psi : \sigma \rightsquigarrow \varphi$$

and

$$\Xi_i \mid C_i \models \varphi : \xi_i$$

for every $i \in I$. Furthermore, if ψ has abstraction type annotation and all derivations $\Xi_i[\Sigma] \mid C_i[\Theta] \vdash \psi : \xi_i[\sigma]$ respects the annotation, then the above abstraction respects the abstraction type as well.

Proof. By induction on the structure of families of derivations.

Assume that at least one of derivations ends with (R-COERCE). One can assume without loss of generality that all derivations in the family end with (R-COERCE), by applying the trivial coercion $\xi_i[\sigma] \preceq \xi_i[\sigma]$ if necessary. The claim follows from the induction hypothesis and Lemma 14.

Otherwise the last rule of a derivation is completely determined by ψ .

- Case $\psi = x$: Then $\varphi = x$ satisfies the requirements.
- Case $\psi = p(\tilde{a})$: Because $\models C_i[\Theta] \implies p(\tilde{a})$ for every $i \in I$, we have

$$\models \left(\bigvee_i C_i[\Theta] \right) \implies p(\tilde{a}).$$

Therefore

$$\frac{\Sigma \mid \Theta, p(\tilde{a}) \vdash p(\tilde{a}) : \bullet \rightsquigarrow b_{p(\tilde{a})} \quad \Sigma \vdash b_{p(\tilde{a})} : ((\Theta, p(\tilde{a})), \bullet) \preceq (\Theta, \bullet) \rightsquigarrow \bigvee_{i \in I} \hat{C}_i[b_\Theta]}{\Sigma \mid \Theta, p(\tilde{a}) \vdash p(\tilde{a}) : \bullet \rightsquigarrow \bigvee_{i \in I} \hat{C}_i[b_\Theta]}$$

(More precisely, the result of the abstraction is a formula that is $\beta\eta$ -equivalent to $\bigvee_{i \in I} \hat{C}_i[b_\Theta]$.) It is easy to see that this satisfies the requirement.

- Case $\psi = \psi_0 a$: Then the premises are

$$\Xi_i[\Sigma] \mid C'_i[\Theta'] \vdash \psi_0 : (D_i \rightarrow \xi_i)[(x : \text{int}[\tilde{P}] \rightarrow \sigma')]$$

for some $C'_i, \Theta', \xi'_i, \tilde{P}$ and σ' . Then

$$\begin{aligned}\sigma &= [a/x]\sigma' \\ \xi_i[\sigma] &= \xi_i[[a/x]\sigma'] \\ \Theta &= (\Theta', [a/x]\tilde{P}) \\ C_i[\Theta] &= C''_i[\Theta'] \wedge D_i[[a/x]\tilde{P}].\end{aligned}$$

By the induction hypothesis, there exists φ such that

$$\Sigma \mid \Theta' \vdash \psi_0 : (x : \text{int}[\tilde{P}] \rightarrow \sigma') \rightsquigarrow \varphi$$

and

$$\Xi_i \mid C'_i \models \varphi : (D_i \rightarrow \xi'_i)$$

for every $i \in I$. Therefore $\varphi \widetilde{b_{[a/x]P}}$ satisfies the requirements.

– Case $\psi = \nu x.\psi_0$: The premises for the i -th derivation is

$$\forall i \in J_i. \quad (\Xi_i, x : \bigwedge_{j \in J_i} \xi_{i,j})[\Sigma, x : \sigma] \mid C_i[\Theta] \vdash \psi_0 : \xi_{i,j}[\sigma]$$

where $\xi_i = \xi_{i,j}$ for some $j \in J_i$. By applying the induction hypothesis to the family indexed by $\{(i, j) \mid i \in I, j \in J_i\}$, we have φ such that

$$\Sigma, x : \sigma \mid \Theta \vdash \psi_0 : \sigma \rightsquigarrow \varphi$$

and for every $i \in I$ and $j \in J_i$,

$$\Xi_i, x : \bigwedge_{j \in J_i} \xi_{i,j} \mid C_i \models \varphi : \xi_{i,j}.$$

Let $\rho \in \llbracket \Xi_i \mid C_i \rrbracket$ and v be the minimum element of $\llbracket \bigwedge_{j \in J_i} \xi_{i,j} \rrbracket$ (cf. Lemma 13). Then $\rho[x \mapsto v] \in \llbracket \Xi_i, x : \bigwedge_{j \in J_i} \xi_{i,j} \mid C_i \rrbracket$. So $\llbracket \varphi \rrbracket_{\rho[x \mapsto v]}^C \in \llbracket \xi_{i,j} \rrbracket$ for every $j \in J_i$, and thus

$$\llbracket \varphi \rrbracket_{\rho[x \mapsto v]}^C \in \llbracket \bigwedge_{i \in J_i} \xi_{i,j} \rrbracket,$$

that means, $\llbracket \varphi \rrbracket_{\rho[x \mapsto v]}^C \sqsupseteq v$. So

$$\llbracket \nu x.\varphi \rrbracket_{\rho}^C(v) \sqsupseteq v,$$

that means, $\llbracket \nu x.\varphi \rrbracket_{\rho}^C \in \llbracket \bigwedge_{j \in J_i} \xi_{i,j} \rrbracket \subseteq \llbracket \xi_{i,j} \rrbracket$.

Other cases are similar. □

We prove the theorems.

Proof of Theorem 2

Assume that $\vdash \psi : \#$. By Lemma 15, there exists φ such that $\vdash \psi : \bullet \rightsquigarrow \varphi$ and $\models \varphi : \#$. By definition, $\llbracket \varphi \rrbracket^C = \#$. By Proposition 3, we have $\llbracket \varphi \rrbracket = \#$.

Proof of Theorem 3

Assume $\vdash \psi : \neg c$. Let $\check{\psi}$ be the formula with abstract type annotation extracted from the derivation. The derivation respects the annotation $\check{\psi}$. So by Lemma 15, there exists φ such that $\vdash \check{\psi} : \bullet \rightsquigarrow \varphi$ and $\models \varphi : \neg c$, i.e. $c \notin \llbracket \varphi \rrbracket^{\mathcal{C}}$. By (the contraposition of) Lemma 9, for every counterexample c' of φ , one has $\mathcal{C}(c') \neq \{c\}$. Since $\mathcal{C}(c') \neq \emptyset$, we have $\mathcal{C}(c') \setminus \{c\} \neq \emptyset$.

E Additional Information about Experimental Results

Figure 7 shows the breakdown of the result reported in the lefthand side of Figure 4 into the results for first-order inputs and higher-order inputs.

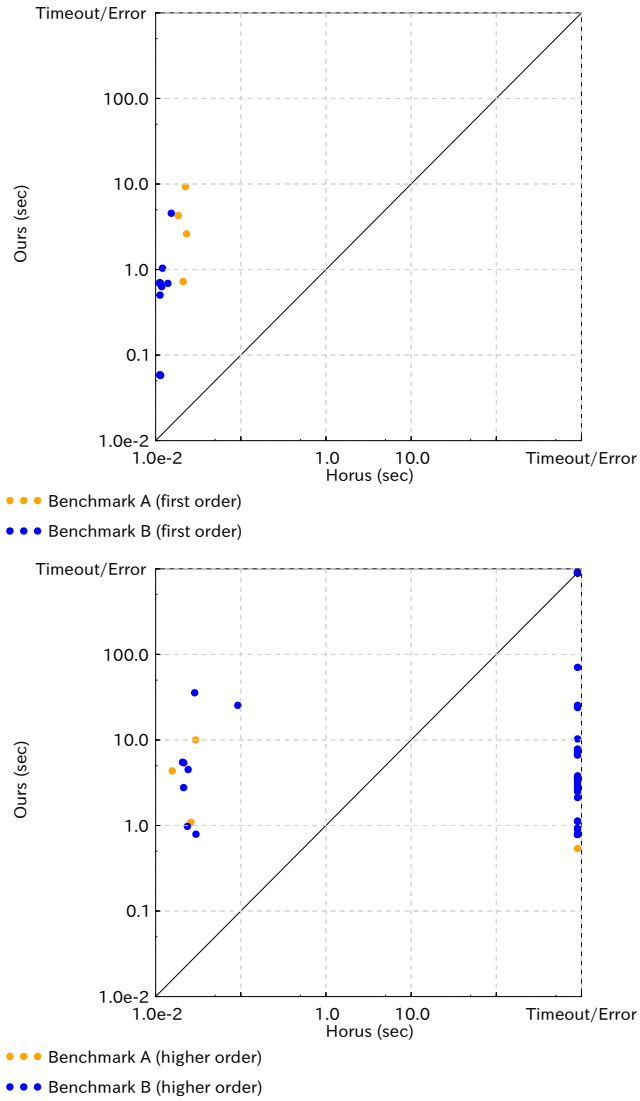


Fig. 7. Comparison with Horus for first-order and higher-order benchmarks