Optimal Inference of Fields in Row-Polymorphic Records (Proof Appendix)

Axel Simon

Technische Universität München
Lehrstuhl für Informatik 2
Garching b. München, Germany
Axel.Simon@in.tum.de

$$
\begin{array}{rcl}
\mathit{own} & : & \mathbb{X} \times \mathbb{E} \to \mathscr{P}(\mathbb{X} \backslash \mathbb{X}^\lambda \times \mathbb{X}^\lambda) \\
\mathit{own}(f, x) & = & \emptyset \\
\mathit{own}(f, \lambda x \,.\, e) & = & \{\langle f, x\rangle\} \cup \mathit{own}(f, e) \\
\mathit{own}(f, e_1\, e_2) & = & \mathit{own}(f, e_1) \cup \mathit{own}(f, e_2) \\
\mathit{own}(f, \mathbf{let}\ x = e\ \mathbf{in}\ e') & = & \mathit{own}(x, e) \cup \mathit{own}(f, e') \\
\mathit{own}(f, c) & = & \emptyset \\
\mathit{own}(f, \mathbf{if}\ e_s\ \mathbf{then}\ e_t\ \mathbf{else}\ e_e) & = & \mathit{own}(f, e_s) \cup \\
 & & \mathit{own}(f, e_t) \cup \mathit{own}(f, e_e)
\end{array}
$$

Figure 1. calculating the owner of a variable

Abstract

This report contains the derivation details of the paper of the same name [7].

A. Additional Preliminaries

This section contains proves and more on the derivation process. It is an extension of [6].

Without loss of generality we assume that all **let**- and λ-bound variables $x \in \mathbb{X}$ are pair-wise different. Let $\mathbb{X}^\lambda \subseteq \mathbb{X}$ denote the set of variables that are λ-bound.

Let $x \prec_{\mathbb{X}} y$ iff x is in scope of y. We write environments as $[x_1 \mapsto t_1, \ldots x_n \mapsto t_n]$ where $x_i \prec_{\mathbb{X}} x_{i+1}$. Define $\mathit{args} \subseteq \mathscr{P}(\mathbb{X})$ such that $m \in \mathit{args}$ contains a function $f \in \mathbb{X} \setminus \mathbb{X}^\lambda$ and $m \setminus \{f\}$ are the λ-bound arguments of f. For instance, $\mathit{args} = \{\{f, s\}, \{s'\}, \{v\}\}$ for the introductory example.

We give a formal definition of how to compute *args* from a program $e \in \mathbb{E}$. Let $\xi \in \mathbb{X} \setminus \mathbb{X}^\lambda$ be a function symbol that represents the owner of outermost λ-variables. Each function symbol and its owned λ-bound variables form a so-called *monomorphic group*, a notion we use to specify what variables to instantiate when accessing **let**- and λ-bound variables.

DEFINITION 1. *A variable set $V \subseteq \mathbb{X}$ forms a monomorphic group in program P iff $V \in \mathit{args}(P)$ where $\mathit{args}(P) := \{V \subseteq \mathbb{X} \mid \exists f \,.\, \{f\} = (V \setminus \mathbb{X}^\lambda) \wedge \forall x \in (V \cap \mathbb{X}^\lambda) \,.\, \langle f, x\rangle \in \mathit{own}(\xi, P)\}$.*

As informally described above, we let *args* denote the set of all monomorphic groups of the current program. Observe that *args* is a partitioning of λ-bound variables:

PROPOSITION 1. *Let $\{V_1, \ldots V_n\} = \mathit{args}(P)$ and $\mathbb{X}$ the be the set of variables in P. Then $V_i \cap V_j = \emptyset$ for all $i \neq j$ and $\mathbb{X}^\lambda \subseteq \bigcup_{i=1}^n V_i$.*

Proof. Let e_x denote the expression $e_x \equiv \lambda x \,.\, e'$ that binds x. Show $i \neq j$ implies $V_i \cap V_j = \emptyset$. By construction, exactly one function symbol $f \in \mathbb{X} \setminus \mathbb{X}^\lambda$ exists in V_i. Let $x \in V_i \cap V_j \cap \mathbb{X}^\lambda$ and thus $\langle f, x\rangle \in \mathit{own}(\xi, P)$. Then there exists a function symbol $g \in V_j$ with $\langle g, x\rangle \in \mathit{own}(\xi, P)$. Since all variable bindings feature different variables, the same expression e_x must have been visited by *own*, once with $\mathit{own}(f, e_x)$ and once with $\mathit{own}(g, e_x)$ which is a contradiction since *own* visits every sub-expression of P exactly once. Now show $\mathbb{X}^\lambda \subseteq \bigcup_{i=1}^n V_i$. By definition of *own*, for each $x \in \mathbb{X}^\lambda$, e_x is visited exactly once, returning $\langle f, x\rangle$. Thus there exists V_i with $f, x \in V_i$ as required. ■

Intuitively, each monomorphic group V_i contains one **let**-bound function f and its arguments $V_i \backslash \{f\}$. Since $\mathit{args}(P)$ partitions $\mathbb{X}^\lambda$, each λ-bound variable is argument to only one function.

B. Concrete Semantics

In this section we present the concrete semantic of the language in Fig. 1 in [7] by closely following a definition by Milner [5] but adding the operations on records.

Let $S_\perp := S \cup \{\perp_S\}$ where $\perp_S \notin S$ denotes an undefined value or a non-terminating evaluation. The denotational standard semantics evaluates $e \in \mathbb{E}$ to a program value $\mathbb{U}_\perp$ where $\mathbb{U}$ is a sum of constructors terms $\mathbb{A}$, records $\mathbb{R}$, functions $\mathbb{F}$ and the wrong value ω, that denotes a type error. A record $r \in \mathbb{R}$ is a partial map from names to values. They are constructed from the empty map $[]$ and extended using $r[\mathrm{N} \mapsto v]$. Define values $\mathbb{U}$ as follows:

$$
\begin{array}{rcl}
\mathbb{U} & := & \mathbb{F} + \mathbb{A} + \mathbb{R} + \mathbb{W} \\
\mathbb{F} & := & \mathbb{U} \to \mathbb{U}_\perp \\
\mathbb{A} & := & \mathbb{Z} \cup \ldots \\
\mathbb{R} & := & \mathbb{L} \dashrightarrow \mathbb{U} \\
\mathbb{W} & := & \{\omega\}
\end{array}
$$

We construct values of type $\mathbb{U}$ from a value in $\mathbb{D} = \mathbb{F}, \mathbb{A}, \mathbb{R}, \mathbb{W}$ using the injection functions $\uparrow^{\mathbb{U}}_{\mathbb{D}}$: $\mathbb{D} \to \mathbb{U}$ (thus, $\uparrow^{\mathbb{U}}_{\mathbb{D}}$ is analogous to a Haskell/ML constructor of the algebraic data type $\mathbb{U}$ that takes a single value of type $\mathbb{D}$). Conversely, define $\downarrow^{\mathbb{U}}_{\mathbb{D}}$: $\mathbb{U}_\perp \to \mathbb{D}_\perp$ with $\downarrow^{\mathbb{U}}_{\mathbb{D}}(\uparrow^{\mathbb{U}}_{\mathbb{D}}(v)) = v$ for all $v \in \mathbb{D}$ and $\downarrow^{\mathbb{U}}_{\mathbb{D}}(\cdot) = \perp_{\mathbb{U}}$ otherwise. For brevity, let $\Omega = \uparrow^{\mathbb{U}}_{\mathbb{W}}(\omega) \in \mathbb{U}$ define the error value. In order to test if $v \in \mathbb{U}$ originated in $\mathbb{D}$ we define the following C-like conditional:

$$
(v \underline{\in} \mathbb{D}\ \underline{?}\ v_1 \,\underline{:}\, v_2) = \begin{cases} v_1 & \text{if } v = \uparrow^{\mathbb{U}}_{\mathbb{D}}(d) \text{ for some } d \in \mathbb{D} \\ \perp_{\mathbb{U}} & \text{if } v = \perp_{\mathbb{U}} \\ v_2 & \text{otherwise} \end{cases}
$$

In order to evaluate an expression, it is necessary to track the values of variables. To this end, we write program environments as

$$
\begin{array}{rcl}
\mathcal{S}[\![\cdot]\!] & : & \mathbb{E} \to (\mathbb{X} \to \mathbb{U}_\bot) \to \mathbb{U}_\bot \\
\mathcal{S}[\![x]\!]\ \rho & = & \rho(x) \\
\mathcal{S}[\![\lambda x\,.\,e]\!]\ \rho & = & \uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda v\,.\,\exists_x(\mathcal{S}[\![e]\!]\ \rho[x \mapsto v])) \\
\mathcal{S}[\![e_1\ e_2]\!]\ \rho & = & (v_1 \underline{\in} \mathbb{F}\ \underline{?}\ (v_2 \underline{\in} \mathbb{W}\ \underline{?}\ \Omega\ \underline{:}\ \downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1)v_2)\ \underline{:}\ \Omega) \text{ where } v_i = \mathcal{S}[\![e_i]\!]\ \rho \text{ for } i = 1,2 \\
\mathcal{S}[\![\mathbf{let}\ x = e\ \mathbf{in}\ e']\!]\ \rho & = & (v \underline{\in} \mathbb{W}\ \underline{?}\ \Omega\ \underline{:}\ \exists_x(\mathcal{S}[\![e']\!]\ \rho[x \mapsto v])) \text{ where } v = \mathrm{lfp}^{\preceq}_{\bot_{\mathbb{U}}} \lambda v\,.\,\mathcal{S}[\![e]\!]\ \rho[x \mapsto v] \\
\mathcal{S}[\![c]\!]\ \rho & = & \uparrow^{\mathbb{U}}_{\mathbb{A}}(c) \text{ for } c \in \mathbb{Z} \qquad \mathcal{S}[\![\underline{\{\}}]\!]\ \rho\ =\ \uparrow^{\mathbb{U}}_{\mathbb{R}}([]) \\
\mathcal{S}[\![@\underline{\{}\textsc{n} = e\underline{\}}]\!]\ \rho & = & \uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda r\,.\,(r \underline{\in} \mathbb{R}\ \underline{?}\ (v \underline{\in} \mathbb{W}\ \underline{?}\ \Omega\ \underline{:}\ \uparrow^{\mathbb{U}}_{\mathbb{R}}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(r)[\textsc{n} \mapsto v]))\ \underline{:}\ \Omega)) \text{ where } v = \mathcal{S}[\![e]\!]\ \rho \\
\mathcal{S}[\![\#\textsc{n}]\!]\ \rho & = & \uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda r\,.\,(r \underline{\in} \mathbb{R}\ \underline{?}\ \text{if } \textsc{n} \in \mathbf{dom}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(r)) \text{ then } \downarrow^{\mathbb{U}}_{\mathbb{R}}(r)(\textsc{n}) \text{ else } \Omega\ \underline{:}\ \Omega)) \\
\mathcal{S}[\![\mathbf{if}\ e_s\ \mathbf{then}\ e_t\ \mathbf{else}\ e_e]\!]\ \rho & = & (v_s \underline{\in} \mathbb{A}\ \underline{?}\ \text{if } \downarrow^{\mathbb{U}}_{\mathbb{A}}(v_s) \subseteq \mathbb{Z} \text{ then } (\text{if } \downarrow^{\mathbb{U}}_{\mathbb{A}}(v_s) \neq 0 \text{ then } \mathcal{S}[\![e_t]\!]\ \rho \text{ else } \mathcal{S}[\![e_e]\!]\ \rho) \text{ else } \Omega\ \underline{:}\ \Omega) \text{ where } v_s = \mathcal{S}[\![e_s]\!]\ \rho
\end{array}
$$

Figure 2. standard denotational semantics for $e \in \mathbb{E}$

$$
\begin{array}{rcl}
\mathcal{C}_1[\![\cdot]\!] & : & \mathbb{E} \to \mathscr{P}((\mathbb{X} \to \mathbb{U}_\bot) \to \mathbb{U}_\bot) \\
\mathcal{C}_1[\![x]\!] & = & \{\lambda\rho\,.\,\rho(x)\} \\
\mathcal{C}_1[\![\lambda x\,.\,e]\!] & = & \{\lambda\rho\,.\ \uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda v\,.\,S(\rho[x \mapsto v])) \mid S \in \mathcal{C}_1[\![e]\!]\} \\
\mathcal{C}_1[\![e_1\ e_2]\!] & = & \{\lambda\rho\,.\,(S_1\rho \underline{\in} \mathbb{F}\ \underline{?}\ (S_2\rho \underline{\in} \mathbb{W}\ \underline{?}\ \Omega\ \underline{:}\ \downarrow^{\mathbb{U}}_{\mathbb{F}}(S_1\rho)S_2\rho)\ \underline{:}\ \Omega) \mid S_i \in \mathcal{C}_1[\![e_i]\!], i = 1,2\} \\
\mathcal{C}_1[\![\mathbf{let}\ x = e\ \mathbf{in}\ e']\!] & = & \{\lambda\rho\,.\,(v \underline{\in} \mathbb{W}\ \underline{?}\ \Omega\ \underline{:}\ S'(\rho[x \mapsto v])) \mid S' \in \mathcal{C}_1[\![e']\!] \wedge v \in V\} \\
 & & \text{where } V = \{\mathrm{lfp}^{\preceq}_{\bot_{\mathbb{U}}} \lambda v\,.\,S(\rho[x \mapsto v]) \mid S \in \mathcal{C}_1[\![e]\!]\} \\
\mathcal{C}_1[\![c]\!] & = & \{\lambda\rho\,.\ \uparrow^{\mathbb{U}}_{\mathbb{A}}(c)\} \qquad \mathcal{C}_1[\![\underline{\{\}}]\!]\ =\ \{\lambda\rho\,.\ \uparrow^{\mathbb{U}}_{\mathbb{R}}([])\} \\
\mathcal{C}_1[\![@\underline{\{}\textsc{n} = e\underline{\}}]\!] & = & \{\lambda\rho\,.\ \uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda r\,.\,(r \underline{\in} \mathbb{R}\ \underline{?}\ (S\rho \underline{\in} \mathbb{W}\ \underline{?}\ \Omega\ \underline{:}\ \uparrow^{\mathbb{U}}_{\mathbb{R}}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(r)[\textsc{n} \mapsto v]))\ \underline{:}\ \Omega)) \mid S \in \mathcal{C}_1[\![e]\!]\} \\
\mathcal{C}_1[\![\#\textsc{n}]\!] & = & \{\lambda\rho\,.\ \uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda r\,.\,(r \underline{\in} \mathbb{R}\ \underline{?}\ \text{if } \textsc{n} \in \mathbf{dom}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(r)) \text{ then } \downarrow^{\mathbb{U}}_{\mathbb{R}}(r)(\textsc{n}) \text{ else } \Omega\ \underline{:}\ \Omega))\} \\
\mathcal{C}_1[\![\mathbf{if}\ e_s\ \mathbf{then}\ e_t\ \mathbf{else}\ e_e]\!] & = & \{\lambda\rho\,.\,(S_s\rho \underline{\in} \mathbb{A}\ \underline{?}\ \text{if } \downarrow^{\mathbb{U}}_{\mathbb{A}}(S_s\rho) \subseteq \mathbb{Z} \text{ then } S_t\rho \text{ else } \Omega\ \underline{:}\ \Omega) \mid S_s \in \mathcal{C}_1[\![e_s]\!] \wedge S_t \in \mathcal{C}_1[\![e_t]\!]\} \\
 & \cup & \{\lambda\rho\,.\,(S_s\rho \underline{\in} \mathbb{A}\ \underline{?}\ \text{if } \downarrow^{\mathbb{U}}_{\mathbb{A}}(S_s\rho) \subseteq \mathbb{Z} \text{ then } S_e\rho \text{ else } \Omega\ \underline{:}\ \Omega) \mid S_s \in \mathcal{C}_1[\![e_s]\!] \wedge S_e \in \mathcal{C}_1[\![e_e]\!]\}
\end{array}
$$

Figure 3. collecting semantics for $e \in \mathbb{E}$ with **if**-statements being abstracted

follows:

$$\rho \quad \in \quad \mathbb{X} \to \mathbb{U}_\bot$$

The denotational semantics $\mathcal{S}[\![e]\!] \in (\mathbb{X} \to \mathbb{U}_\bot) \to \mathbb{U}_\bot$ of an expression $e \in \mathbb{E}$ is defined in Fig 2. Here, the conditional statement is used in two ways: Firstly, $(v \underline{\in} \mathbb{D}\ \underline{?}\ \ldots \downarrow^{\mathbb{U}}_{\mathbb{D}}(v) \ldots\ \underline{:}\ \Omega)$ is used to test if $v \in \mathbb{U}$ is in fact element of $\mathbb{D}$ and then to extract this element $\downarrow^{\mathbb{U}}_{\mathbb{D}}(v) \in \mathbb{D}$ while returning the type error Ω otherwise. Secondly, if a value v is merely required not to be a type error, the statement $(v \underline{\in} \mathbb{W}\ \underline{?}\ \Omega\ \underline{:}\ \ldots)$ is used which returns Ω if v is a type error. Note that both conditionals evaluate to $\bot_{\mathbb{U}}$ if $v = \bot_{\mathbb{U}}$ and, hence, the definition of $e_1\ e_2$ returns $\bot_{\mathbb{U}}$ if e_2 is undefined which gives the language a call-by-value semantics.

The **let**-rule computes a $\preceq$-least fixpoint of a function ψ, written $\mathrm{lfp}^{\preceq}_{\bot_{\mathbb{U}}} \psi$ where $\bot_{\mathbb{U}}$ is the start value. The order $\preceq$ is a partial order on values $v \in \mathbb{U}_\bot$, defined as follows: $v_1 \preceq v_2$ iff $v_1 = \bot_{\mathbb{U}}$ or, for $i = 1, 2$, if $f_i = \downarrow^{\mathbb{U}}_{\mathbb{F}}(v_i) \neq \bot_{\mathbb{U}}$ then $f_1(v) \preceq f_2(v)$ for all $v \in \mathbb{U}$ or if $r_i = \downarrow^{\mathbb{U}}_{\mathbb{R}}(v_i) \neq \bot_{\mathbb{U}}$ then $\mathbf{dom}(r_1) \subseteq \mathbf{dom}(r_2)$.

The record update $(@\underline{\{}\textsc{n} = e\underline{\}}\ r)$ overwrites or adds the field $\textsc{n}$ in r. Selecting a field $(\#\textsc{n}\ r)$ is a type error if $\textsc{n}$ is not present in r.

We now discuss how to lift this semantics to sets.

B.1 Collecting Semantics and its Abstraction

In program analysis, every abstract value (here: a type) represents many concrete values. In order to relate a type with a set of values, we lift the semantics $\mathcal{S}[\![e]\!] : (\mathbb{X} \to \mathbb{U}_\bot) \to \mathbb{U}_\bot$ of an expression $e \in \mathbb{E}$ to sets of functions, giving the collecting semantics $\mathcal{C}_1[\![e]\!] : \mathscr{P}((\mathbb{X} \to \mathbb{U}_\bot) \to \mathbb{U}_\bot)$. Specifically, we convert each rule $\mathcal{S}[\![e]\!]\ \rho = \ldots \mathcal{S}[\![e']\!]\ \ldots$ of the standard semantics in Fig. 2 to $\mathcal{C}_1[\![e]\!] = \{\lambda\rho\,.\,\ldots S \ldots \mid S \in \mathcal{C}_1[\![e']\!]\}$ as hinted at in Fig. 3. The only exception is the **if**-statement that is abstracted to two function sets, one that always returns e_t and one that always returns e_e. In other word, we abstract conditionals to a non-deterministic choice. This abstraction is required in order to claim that our abstraction is backward-complete. Due to this abstraction, we also use the more general term "value semantics" rather than collecting semantics. This concludes the discussion of the concrete and value semantics.

The complete collecting semantics is presented in Fig. 3.

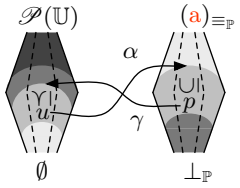

Figure 4. antitone abstraction

B.2 Abstraction to Types

We sketch the abstraction of the value semantics presented in Sect. B.1 to sets of monotypes. Figure 5 presents $\alpha_{\mathbb{M}1}$ which ab-

$$
\begin{array}{rcl}
\alpha_{\mathbb{M}1} & : & \mathbb{U}_\perp \to \mathscr{P}(\mathbb{M}) \\
\alpha_{\mathbb{M}1}(\bot_{\mathbb{U}}) & = & \mathbb{M} \\
\alpha_{\mathbb{M}1}(\uparrow^{\mathbb{U}}_{\mathbb{A}}(c)) & = & \{\texttt{Int}\} \qquad \text{if } c \in \mathbb{Z} \\
\alpha_{\mathbb{M}1}(\uparrow^{\mathbb{U}}_{\mathbb{F}}(f)) & = & \{t_1 \to t_2 \mid v_1 \in \mathbb{U} \wedge v_2 = f(v_1) \\
 & & \qquad \wedge t_i \in \alpha_{\mathbb{M}1}(v_i)\} \\
\alpha_{\mathbb{M}1}(\uparrow^{\mathbb{U}}_{\mathbb{R}}([n_i \mapsto v_i]_{i\in\mathcal{I}})) & = & \underline{\{n_i : t_i \mid t_i \in \alpha_{\mathbb{M}1}(v_i), i \in \mathcal{I}\}} \\
\alpha_{\mathbb{M}1}(\Omega) & = & \emptyset
\end{array}
$$

Figure 5. abstracting single values to types

stracts a single value to a set of types. It follows mainly the definition in [1] but also abstracts records $[n_i \mapsto v_i]_{i\in\mathcal{I}}$ for any index set $\mathcal{I} \subseteq \mathbb{L}$. Note that we assume that the record type is a set rather than a sequence. A particularity of type systems is that $\alpha_{\mathbb{M}1}$ is antitone: an undefined value $\bot_{\mathbb{U}}$ is $\preceq$-smallest and maps to the $\subseteq$-largest set of types $\mathbb{M}$ whereas Ω maps to an empty set. An antitone abstraction is a Galois connection $\langle \alpha, \gamma\rangle$ with $p \subseteq \alpha(u)$ iff $u \preceq \gamma(p)$ as illustrated in Fig. 4 and an abstract transformer $f^\sharp$ is sound with respect to the concrete transformer f if $f^\sharp \circ \alpha \subseteq \alpha \circ f$.

Given how to abstract a single value, a single environment is abstracted by $\alpha^{\mathcal{X}}_{\mathbb{M}1} : (\mathbb{X} \to \mathbb{U}_\perp) \to \mathscr{P}(\mathbb{X} \to \mathbb{M})$, defined as:

$$
\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho) := \bigcup \left\{ \overline{\rho}_{\mathbb{M}} \subseteq \mathbb{X} \to \mathbb{M} \;\middle|\; \begin{array}{l} \forall \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}, x \in \mathbb{X}. \\ \rho_{\mathbb{M}}(x) \in \alpha_{\mathbb{M}1}(\rho(x)) \end{array} \right\}
$$

In order to derive inference rules using the formula $f^\sharp = \alpha \circ f \circ \gamma$, we define the concretization $\gamma^{\mathcal{X}}_{\mathbb{M}} : \mathscr{P}(\mathbb{X} \to \mathbb{M}) \to \mathscr{P}(\mathbb{X} \to \mathbb{U}_\perp)$ to value environments as $\gamma^{\mathcal{X}}_{\mathbb{M}}(\overline{\rho}_{\mathbb{M}}) := \{\rho \in \mathbb{X} \to \mathbb{U}_\perp \mid \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\}$. Observe that $\gamma^{\mathcal{X}}_{\mathbb{M}}$ maps to sets of concrete environments which is necessary since even a single type environment corresponds to several programs. We thus define a symmetric $\alpha^{\mathcal{X}}_{\mathbb{M}} : \mathscr{P}(\mathbb{X} \to \mathbb{U}_\perp) \to \mathscr{P}(\mathbb{X} \to \mathbb{M})$ as $\alpha^{\mathcal{X}}_{\mathbb{M}}(\bar{\rho}) := \bigcap_{\rho\in\bar{\rho}} \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)$. These maps form a Galois connection between program value environments (with $\bar{\preceq}$ is $\preceq$ lifted point-wise) and sets of monotype environments, written as follows:

$$
\langle \mathscr{P}(\mathbb{X} \to \mathbb{U}_\perp), \bar{\preceq}\rangle \underset{\alpha^{\mathcal{X}}_{\mathbb{M}}}{\overset{\gamma^{\mathcal{X}}_{\mathbb{M}}}{\leftrightarrows}} \langle \mathscr{P}(\mathbb{X} \to \mathbb{M}), \subseteq\rangle
$$

Moreover, since records are interpreted as sets, rather than sequences, no two monotype environments represent the same program values, thus $\alpha^{\mathcal{X}}_{\mathbb{M}} \circ \gamma^{\mathcal{X}}_{\mathbb{M}}$ is the identity, $\gamma^{\mathcal{X}}_{\mathbb{M}}$ an injection "$\hookleftarrow$" and $\alpha^{\mathcal{X}}_{\mathbb{M}}$ a surjection "$\twoheadrightarrow$". A Galois connection with this property is called a Galois insertion [2]. It ensures that the optimal type of a program also has a unique representation in $\mathscr{P}(\mathbb{X} \to \mathbb{M})$.

Note that computing $f^\sharp = \alpha \circ f \circ \gamma$ requires f to accept sets of environments but $S \in \mathcal{C}_1[\![e]\!]$ expects one environment and returns a value. Lifting S to operate on sets forces us to join all resulting values, thereby losing the information which input environment resulted in which output value. Hence, we use a trick that allows us to keep the link between the input environment and the output value by returning the input environment in which a special symbol $\kappa \notin \mathbb{X}$ is bound to the output. Thus, the concrete semantics we abstract is $\lambda\bar{\rho}.\{\rho[\kappa \mapsto S\rho] \mid S \in \mathcal{C}_1[\![e]\!] \wedge \rho \in \bar{\rho}\}$. We lift this semantics to types by computing $f^\sharp = \alpha \circ f \circ \gamma$ as $\mathcal{T}[\![e]\!] = \alpha^{\mathcal{X}}_{\mathbb{M}} \circ \lambda\bar{\rho}.\{\rho[\kappa \mapsto S\rho] \mid S \in \mathcal{C}_1[\![e]\!] \wedge \rho \in \bar{\rho}\} \circ \gamma^{\mathcal{X}}_{\mathbb{M}}$. After inlining $\alpha^{\mathcal{X}}_{\mathbb{M}}$ and $\gamma^{\mathcal{X}}_{\mathbb{M}}$ we get the following equation $\mathcal{T}[\![e]\!]\ \overline{\rho}_{\mathbb{M}} := \bigcap_{\rho\in\mathbb{X}\to\mathbb{U}_\perp} \{\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho[\kappa \mapsto S\rho]) \mid S \in \mathcal{C}_1[\![e]\!] \wedge \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\}$. Before we compute this equation for each language construct $e \in \mathbb{E}$, we ensure that the output type of each language construct does not depend on the value of its input types.

C. Preservation of Types

We establish that $\alpha(c_1) = \alpha(c_2) \Rightarrow \alpha(f(c_1)) = \alpha(f(c_2))$ holds in our context. We thereby derive at Lemma 2 in [7] which is presented here as Corollary 1. We assert this first for abstracting values to sets of monotypes:

LEMMA 1. *Let $\rho_1, \rho_2 \in \mathbb{X} \to \mathbb{U}_\perp$ with $\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_1) = \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_2)$. Then $\alpha_{\mathbb{M}1}(S\rho_1) = \alpha_{\mathbb{M}1}(S\rho_2)$ for all $S \in \mathcal{C}_1[\![e]\!]$ and $e \in \mathbb{E}$.*

Proof. By structural induction over $e \in \mathbb{E}$. We use "i.h." (induction hypothesis) for $\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_1) = \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_2)$.

x: $S = \lambda\rho.\,\rho(x)$. Thus, $\alpha_{\mathbb{M}1}(S\rho_1) = \alpha_{\mathbb{M}1}(\rho_1(x)) = \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_1)(x) \overset{\text{i.h.}}{=} \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_2)(x) = \alpha_{\mathbb{M}1}(\rho_2(x)) = \alpha_{\mathbb{M}1}(S\rho_2)$.

$\lambda x\,.\,e$: Pick $S \in \mathcal{C}_1[\![e]\!]$. Let $u_i = S(\rho_i[x \mapsto v])$ for some $v \in \mathbb{U}_\perp$. Thus $\alpha_{\mathbb{M}1}(\uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda v\,.\,S(\rho_i[x \mapsto v]))) = \{t_v \to t_u \mid t_v \in \alpha_{\mathbb{M}1}(v) \wedge t_u \in \alpha_{\mathbb{M}1}(u_i)\}$ for $i = 1, 2$. By induction hypothesis, it follows that $\alpha_{\mathbb{M}1}(u_1) = \alpha_{\mathbb{M}1}(u_2)$.

$e^1\,e^2$: Pick $S^j \in \mathcal{C}_1[\![e^j]\!]$. The bodies of $\mathcal{C}_1[\![e^1\,e^2]\!]$ have type $t_i = \alpha_{\mathbb{M}1}((S^1\rho_i \underline{\in} \mathbb{F} \mathbin{\underline{?}} (S^2\rho_i \underline{\in} \mathbb{W} \mathbin{\underline{?}} \Omega \mathbin{\underline{:}} \downarrow^{\mathbb{U}}_{\mathbb{F}}(S^1\rho_i)S^2\rho_i) \mathbin{\underline{:}} \Omega))$ for $i = 1, 2$. Assume $S^2\rho_i \neq \Omega$ and $S^1\rho_i \underline{\in} \mathbb{F}$ since otherwise the claim holds trivially by i.h. Thus, $t_i = \alpha_{\mathbb{M}1}(\downarrow^{\mathbb{U}}_{\mathbb{F}}(S^1\rho_i)S^2\rho_i) = \{t_r \mid t_a \to t_r \in \alpha_{\mathbb{M}1}(S^1\rho_i) \wedge t_a \in \alpha_{\mathbb{M}1}(S^2\rho_i)\}$. By induction hypothesis $\alpha_{\mathbb{M}1}(S^j\rho_1) = \alpha_{\mathbb{M}1}(S^j\rho_2)$ for $j = 1, 2$ and thus $t_1 = t_2$.

let $x = e$ in e': Pick $S \in \mathcal{C}_1[\![e]\!]$. Let $v'_i = S\rho_i[x \mapsto v])$ for all $v \in \mathbb{U}_\perp$, then $\alpha_{\mathbb{M}1}(v'_1) = \alpha_{\mathbb{M}1}(v'_2)$ by i.h., and, by extension, it follows that for $\hat{v}_i = \mathrm{lfp}^{\preceq}_{\bot_{\mathbb{U}}} \lambda v\,.\,\mathcal{S}[\![e]\!]\ (\rho_i[x \mapsto v])$ it holds that $\alpha_{\mathbb{M}1}(\hat{v}_1) = \alpha_{\mathbb{M}1}(\hat{v}_2)$. Now let $t_i = \alpha_{\mathbb{M}1}((v \underline{\in} \mathbb{W} \mathbin{\underline{?}} \Omega \mathbin{\underline{:}} S'(\rho[x \mapsto v])))$. Pick $S' \in \mathcal{C}_1[\![e']\!]$. Then $t_1 = t_2$ follows.

if e_s then e_t else e_e: Pick $S_s \in \mathcal{C}_1[\![e_s]\!]$, $S_t \in \mathcal{C}_1[\![e_t]\!]$, and $S_e \in \mathcal{C}_1[\![e_e]\!]$. For either branch, the result is Ω unless $S_s\rho \underline{\in} \mathbb{A}$ and $\downarrow^{\mathbb{U}}_{\mathbb{A}}(S_s\rho) \subseteq \mathbb{Z}$ holds. By i.h. these conditions hold in $S_s\rho_1$ iff they hold in $S_s\rho_2$. Thus, if they do not hold, the $\alpha_{\mathbb{M}1}(S_s\rho_1) = \alpha_{\mathbb{M}1}(S_s\rho_2) = \emptyset$ and the claim follows. If the conditions hold, the returned values are $S_t\rho_i$ and $S_e\rho_i$, $i = 1, 2$. By i.h. $\alpha_{\mathbb{M}1}(S_t\rho_1) = \alpha_{\mathbb{M}1}(S_t\rho_2)$ and $\alpha_{\mathbb{M}1}(S_e\rho_1) = \alpha_{\mathbb{M}1}(S_e\rho_2)$ and, hence, the claim follows. ■

Adjust Lemma 1 so that the results of the transfer function is stored in the environment, that is, show $\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_1[\kappa \mapsto S\rho_1]) = \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_2[\kappa \mapsto S\rho_2])$ instead of $\alpha_{\mathbb{M}1}(S\rho_1) = \alpha_{\mathbb{M}1}(S\rho_2)$ for $S \in \mathcal{C}_1[\![e]\!]$. Lifting this result to sets yields:

COROLLARY 1 (Lemma 1 of [7]). *For all $\bar{\rho}_1, \bar{\rho}_2 \subseteq \mathbb{X} \to \mathbb{U}_\perp$ with $\alpha^{\mathcal{X}}_{\mathbb{M}}(\bar{\rho}_1) = \alpha^{\mathcal{X}}_{\mathbb{M}}(\bar{\rho}_2)$ and all expressions $e \in \mathbb{E}$ it follows that $\alpha^{\mathcal{X}}_{\mathbb{M}}(\{\rho_1[\kappa \mapsto S\rho_1] \mid S \in \mathcal{C}_1[\![e]\!] \wedge \rho_1 \in \bar{\rho}_1\}) = \alpha^{\mathcal{X}}_{\mathbb{M}}(\{\rho_2[\kappa \mapsto S\rho_2] \mid S \in \mathcal{C}_1[\![e]\!] \wedge \rho_2 \in \bar{\rho}_2\})$.*

By applying $\overline{\mathbf{lca}}$ on each side of the equations we obtain Lemma 2:

LEMMA 2. *For all $\bar{\rho}_1, \bar{\rho}_2 \subseteq \mathbb{X} \to \mathbb{U}_\perp$ with $\overline{\mathbf{lca}}(\alpha^{\mathcal{X}}_{\mathbb{M}}(\bar{\rho}_1)) = \overline{\mathbf{lca}}(\alpha^{\mathcal{X}}_{\mathbb{M}}(\bar{\rho}_2))$ and all expressions $e \in \mathbb{E}$, it follows that $\overline{\mathbf{lca}}(\alpha^{\mathcal{X}}_{\mathbb{M}}(\{\rho_1[\kappa \mapsto S\rho_1] \mid S \in \mathcal{C}_1[\![e]\!] \wedge \rho_1 \in \bar{\rho}_1\})) = \overline{\mathbf{lca}}(\alpha^{\mathcal{X}}_{\mathbb{M}}(\{\rho_2[\kappa \mapsto S\rho_2] \mid S \in \mathcal{C}_1[\![e]\!] \wedge \rho_2 \in \bar{\rho}_2\}))$.*

Proof. For the sake of a contradiction, let $t_i = \overline{\mathbf{lca}}(\alpha^{\mathcal{X}}_{\mathbb{M}}(\{\rho_i[\kappa \mapsto S\rho_i] \mid S \in \mathcal{C}_1[\![e]\!] \wedge \rho_i \in \bar{\rho}_i\}))\kappa$ and $t_1 \neq t_2$. Then there exists $S \in \mathcal{C}_1[\![e]\!]$ and $\rho^i_{\mathbb{M}} \in \mathscr{P}(\mathbb{X} \cup \{\kappa\} \to \mathbb{M})$ such that $\rho^i_{\mathbb{M}} \in \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho_i[\kappa \mapsto S\rho_i])$ and $\rho^1_{\mathbb{M}}(\kappa) \neq \rho^2_{\mathbb{M}}(\kappa)$. However, this is a contradiction to Lem. 1. ■

Lemma 2 allows us to derive backward complete abstract transfer functions as $f^\sharp_{\mathbb{P}} = \alpha \circ f \circ \gamma$ for each concrete f.

$$
\begin{array}{lll}
\mathcal{T}[\![\cdot]\!] & : & \mathbb{E} \to \mathscr{P}(\mathbb{X} \to \mathbb{M}) \to \mathscr{P}(\mathbb{X} \cup \{\kappa\} \to \mathbb{M}) \\
\mathcal{T}[\![x]\!]\ \overline{\rho}_{\mathbb{M}} & = & \bigcap \{\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho[\kappa \mapsto S\rho]) \mid S \in \mathcal{C}_1[\![x]\!] \wedge \rho \in \gamma^{\mathcal{X}}_{\mathbb{M}}(\overline{\rho}_{\mathbb{M}})\} \\
 & \overset{1}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho[\kappa \mapsto \rho(x)]) \mid \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \\
 & \overset{2}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \left\{ \bigcup \left\{ \overline{\rho}'_{\mathbb{M}} \subseteq \mathbb{X} \cup \{\kappa\} \to \mathbb{M} \middle| \begin{array}{l} \forall \rho_{\mathbb{M}} \in \overline{\rho}'_{\mathbb{M}}, y \in \mathbb{X} \cup \{\kappa\} \,.\, \rho_{\mathbb{M}}(y) \in \alpha_{\mathbb{M}1}(\rho[\kappa \mapsto \rho(x)](y)) \wedge \\ \overline{\rho}_{\mathbb{M}} \subseteq \exists_\kappa(\overline{\rho}'_{\mathbb{M}}) \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho) \end{array} \right\} \right\} \\
\text{if } x \notin \mathbb{X}^\lambda & \overset{3}{=} & \exists_{\{y'_i \mid i \in [0,n] \wedge i \neq k\}} (expand_{y_0 \ldots y_n, y'_0 \ldots y'_n}(\overline{\rho}_{\mathbb{M}})) \text{ where } y'_i \text{ fresh and} \\
 & & \qquad \wedge x = y_k \wedge \kappa = y'_k \wedge \{y_0, \ldots y_n\} = \{y \in \mathbf{dom}(\overline{\rho}_{\mathbb{M}}) \mid m \in \mathit{args} \wedge x \in m \wedge \min_{\preccurlyeq_{\mathbb{X}}}(m) \preccurlyeq_{\mathbb{X}} y\} \\
\text{if } x \notin \mathbb{X}^\lambda & \overset{4}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \wedge t \in \mathbb{M}\} \cap \{\rho_{\mathbb{M}}[y_0 \mapsto t_0, \ldots y_n \mapsto t_n, \kappa \mapsto \rho_{\mathbb{M}}(x)] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \wedge t_i \in \mathbb{M}\} \\
 & & \text{where } \mathbf{dom}(\overline{\rho}_{\mathbb{M}}) = \langle z_1, \ldots z_m, y_0, \ldots y_n \rangle \wedge \exists k \in [0,n] \,.\, x = y_k \wedge \{y_0, \ldots y_n\} \text{ as above} \\
\text{if } x \notin \mathbb{X}^\lambda & \overset{5}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \wedge t \in \mathbb{M}\} \cap \{\rho_{\mathbb{M}}[x \mapsto t_0, y_1 \mapsto t_1, \ldots y_n \mapsto t_n, \kappa \mapsto \rho_{\mathbb{M}}(x)] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \wedge t_i \in \mathbb{M}\} \\
\text{if } x \in \mathbb{X}^\lambda & \overset{6}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto \rho_{\mathbb{M}}(x)] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \qquad \text{note: solution approximate unless } \alpha^{\mathcal{X}}_{\mathbb{M}} \text{ form [6] is used} \\
\mathcal{T}[\![\lambda x \,.\, e]\!]\ \overline{\rho}_{\mathbb{M}} & \overset{7}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}(S\rho) \wedge S \in \mathcal{C}_1[\![\lambda x \,.\, e]\!] \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \\
 & \overset{8}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}(\uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda v \,.\, \exists_x(S(\rho[x \mapsto v])))) \wedge S \in \mathcal{C}_1[\![e]\!] \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}) \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \\
 & \overset{9}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t_1 \to t_2] \mid v_2 = \exists_x(S(\rho[x \mapsto v_1])) \wedge S \in \mathcal{C}_1[\![e]\!] \wedge v_1 \in \mathbb{U} \wedge t_i \in \alpha_{\mathbb{M}1}(v_i) \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \\
 & \overset{10}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho'_{\mathbb{M}}[\kappa \mapsto t_1 \to t_2] \mid t_2 = \rho'_{\mathbb{M}}(\kappa) \wedge \rho'_{\mathbb{M}} \in \exists_x(\mathcal{T}[\![e]\!]\{\rho_{\mathbb{M}}[x \mapsto t_1]\}) \wedge t_1 \in \mathbb{M} \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \\
 & \overset{11}{=} & \{\exists_x(\rho'_{\mathbb{M}}[\kappa \mapsto t_1 \to t_2]) \mid t_1 \in \rho'_{\mathbb{M}}(x) \wedge t_2 \in \rho'_{\mathbb{M}}(\kappa) \wedge \rho'_{\mathbb{M}} \in \overline{\rho}'_{\mathbb{M}}\} \text{ with } \overline{\rho}'_{\mathbb{M}} = \mathcal{T}[\![e]\!]\{\rho_{\mathbb{M}}[x \mapsto t_1] \mid t_1 \in \mathbb{M} \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \\
\mathcal{T}[\![e_1\ e_2]\!]\ \overline{\rho}_{\mathbb{M}} & = & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}(S\rho) \wedge S \in \mathcal{C}_1[\![e_1\ e_2]\!] \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \\
 & \overset{12}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid \downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1) \in \mathbb{F} \wedge v_2 \neq \Omega \wedge t \in \alpha_{\mathbb{M}1}(\downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1) v_2) \wedge \mathbf{env}\} \text{ where} \\
 & & \qquad \mathbf{env} \equiv v_i = S_i\rho \wedge S_i \in \mathcal{S}[\![e_i]\!]\ \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho) \\
 & \overset{13}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid \exists t_a, t_r \in \mathbb{M} \,.\, \downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1) \in \mathbb{F} \wedge v_2 \neq \Omega \wedge \uparrow^{\mathbb{U}}_{\mathbb{F}}(\downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1)) \in \gamma_{\mathbb{M}1}(t_a \to t_r) \wedge \\
 & & \qquad \forall v \in \gamma_{\mathbb{M}1}(t_a) \,.\, \downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1) v \in \gamma_{\mathbb{M}1}(t_r) \setminus \perp_{\mathbb{P}} \wedge t \in \alpha_{\mathbb{M}1}(\downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1) v_2) \wedge \mathbf{env}\} \\
 & \overset{14}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid \exists t_r \in \mathbb{M} \,.\, v_1 \in \gamma_{\mathbb{M}1}(t_a \to t_r) \wedge v_2 \in \gamma_{\mathbb{M}1}(t_a) \wedge \downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1) v_2 \in \gamma_{\mathbb{M}1}(t_r) \wedge t \in \alpha_{\mathbb{M}1}(\downarrow^{\mathbb{U}}_{\mathbb{F}}(v_1) v_2) \wedge \\
 & \overset{15}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t_r] \mid \exists t_r \in \mathbb{M} \,.\, v_1 \in \gamma_{\mathbb{M}1}(t_a \to t_r) \wedge v_2 \in \gamma_{\mathbb{M}1}(t_a) \wedge \mathbf{env}\} \qquad \mathbf{env}\} \\
 & \overset{16}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t_r] \mid \exists t_r \in \mathbb{M} \,.\, t_1 = t_a \to t_r \wedge t_2 = t_a \wedge t_i \in \rho^i_{\mathbb{M}}(\kappa) \wedge \rho^i_{\mathbb{M}} \in \mathcal{T}[\![e_i]\!]\{\rho_{\mathbb{M}}\} \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \\
 & \overset{17}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t_r] \mid t_a \to t_r = \rho_{\mathbb{M}}(\kappa) \wedge \rho_{\mathbb{M}} \in (\mathcal{T}[\![e_1]\!]\overline{\rho}_{\mathbb{M}}) \cap \{\rho_{\mathbb{M}}[\kappa \mapsto t_a \to t_r] \mid t_r \in \mathbb{M} \wedge t_a \in \rho_{\mathbb{M}}(\kappa) \wedge \rho_{\mathbb{M}} \in \mathcal{T}[\![e_2]\!]\overline{\rho}_{\mathbb{M}}\}\} \\
\mathcal{T}[\![\mathbf{let}\ x = e\ \mathbf{in}\ e']\!]\ \overline{\rho}_{\mathbb{M}} & = & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} (\bigcup \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}(S\rho) \wedge S \in \mathcal{C}_1[\![\mathbf{let}\ x = e\ \mathbf{in}\ e']\!] \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\}) \\
 & \overset{18}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t'] \mid v \neq \Omega \wedge t' \in \alpha_{\mathbb{M}1}(S'\rho[x \mapsto v]) \wedge S' \in \mathcal{S}[\![e']\!] \wedge \\
 & & \qquad v \in \{\mathrm{lfp}^{\preccurlyeq}_{\perp_{\mathbb{U}}} \lambda v \,.\, S(\rho[x \mapsto v]) \mid S \in \mathcal{C}_1[\![e]\!]\} \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \\
 & \overset{19}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t'] \mid t' \in \rho'_{\mathbb{M}}(\kappa) \wedge \rho'_{\mathbb{M}} \in \exists_x(\mathcal{T}[\![e']\!]\{\rho_{\mathbb{M}}[x \mapsto t]\}) \wedge t \in \mathrm{gfp}^{\subseteq}_{\overline{\rho}_{\mathbb{M}}} \lambda T \,.\, \{\rho'_{\mathbb{M}}(\kappa) \mid \rho'_{\mathbb{M}} \in \mathcal{T}[\![e]\!]\{\rho_{\mathbb{M}}[x \mapsto t] \mid t \in T\}\} \wedge \\
 & \overset{20}{=} & \bigcup \{\exists_x(\mathcal{T}[\![e']\!]\{\rho_{\mathbb{M}}\} \mid \rho_{\mathbb{M}} \in \mathrm{gfp}^{\subseteq}_{\{\rho_{\mathbb{M}}[x \mapsto t] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}, t \in \mathbb{M}\}} \lambda \overline{\rho}_{\mathbb{M}} \,.\, \{\exists_\kappa(\rho_{\mathbb{M}}[x \mapsto \rho_{\mathbb{M}}(\kappa)]) \mid \rho_{\mathbb{M}} \in \mathcal{T}[\![e]\!]\overline{\rho}_{\mathbb{M}}\}\} \qquad \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \\
 & \overset{21}{=} & \exists_x(\mathcal{T}[\![e']\!]\overline{\rho}'_{\mathbb{M}}) \text{ where } \overline{\rho}'_{\mathbb{M}} = \mathrm{gfp}^{\subseteq}_{\{\rho_{\mathbb{M}}[x \mapsto t] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}, t \in \mathbb{M}\}} \lambda \overline{\rho}_{\mathbb{M}} \,.\, \{\rho_{\mathbb{M}}[x \mapsto \rho_{\mathbb{M}}(\kappa)] \mid \rho_{\mathbb{M}} \in \mathcal{T}[\![e]\!]\overline{\rho}_{\mathbb{M}}\} \\
\mathcal{T}[\![c]\!]\ \overline{\rho}_{\mathbb{M}} & \overset{22}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}(c) \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} = \{\rho_{\mathbb{M}}[\kappa \to \mathtt{Int}] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \\
\mathcal{T}[\![\underline{\{\}}]\!]\ \overline{\rho}_{\mathbb{M}} & \overset{23}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}(\underline{\{\}}) \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} = \{\rho_{\mathbb{M}}[\kappa \to \underline{\{\}}] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \\
\mathcal{T}[\![\underline{@}\underline{\{}\mathrm{N} = e\underline{\}}]\!]\ \overline{\rho}_{\mathbb{M}} & \overset{24}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}(\uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda r \,.\, (r \underline{\in} \mathbb{R}\ \underline{?}\ (v \underline{\in} \mathbb{W}\ \underline{?}\ \Omega\ \underline{:}\ \uparrow^{\mathbb{U}}_{\mathbb{R}}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(r)[\mathrm{N} \mapsto v])))\ \underline{:}\ \Omega))) \wedge \mathbf{env}\} \\
 & & \qquad \text{where } \mathbf{env} \equiv v = S\rho \wedge S \in \mathcal{C}_1[\![e]\!] \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho) \\
 & \overset{25}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t_r \to t_u] \mid t_r \in \alpha_{\mathbb{M}1}(r) \wedge r \in \mathbb{R} \wedge t_u \in \alpha_{\mathbb{M}1}(r[\mathrm{N} \mapsto v]) \wedge v \neq \Omega \wedge \mathbf{env}\} \\
 & \overset{26}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t_r \to t_u] \mid t_r = \underline{\{}\mathrm{N}_1 : t_1; \ldots \mathrm{N}_n : t_n\underline{\}} \wedge t_i \in \mathbb{M} \wedge t_u = t_r[\mathrm{N} \mapsto \rho_{\mathbb{M}}(\kappa)] \wedge \rho_{\mathbb{M}} \in \mathcal{T}[\![e]\!]\ \overline{\rho}_{\mathbb{M}}\} \\
\mathcal{T}[\![\#\mathrm{N}]\!]\ \overline{\rho}_{\mathbb{M}} & \overset{27}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}(\uparrow^{\mathbb{U}}_{\mathbb{F}}(\lambda r \,.\, (r \underline{\in} \mathbb{R}\ \underline{?}\ \text{if } \mathrm{N} \in \mathbf{dom}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(r)) \text{ then } \downarrow^{\mathbb{U}}_{\mathbb{R}}(r)(\mathrm{N}) \text{ else } \Omega\ \underline{:}\ \Omega))) \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \\
 & \overset{28}{=} & \{\rho_{\mathbb{M}}[\kappa \mapsto t_r \to t_i] \mid t_r = \underline{\{}\mathrm{N}_1 : t_1; \ldots \mathrm{N}_n : t_n\underline{\}} \wedge t_i \in \mathbb{M} \wedge \exists i \,.\, \mathrm{N}_i = \mathrm{N} \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \\
\mathcal{T}[\![\mathbf{if}\ e_s\ \mathbf{then}\ e_t\ \mathbf{else}\ e_e]\!]\ \overline{\rho}_{\mathbb{M}} & \overset{29}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid t \in \alpha_{\mathbb{M}1}((S_s\rho \underline{\in} \mathbb{A}\ \underline{?}\ \text{if } \downarrow^{\mathbb{U}}_{\mathbb{A}}(S_s\rho) \subseteq \mathbb{Z} \text{ then } S_t\rho \text{ else } \Omega\ \underline{:}\ \Omega)) \wedge \\
 & & \qquad S_s \in \mathcal{C}_1[\![e_s]\!] \wedge S_t \in \mathcal{C}_1[\![e_t]\!] \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \cap \ldots \text{ (analogous for } e_e) \\
 & \overset{30}{=} & \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\rho_{\mathbb{M}}[\kappa \mapsto t] \mid \mathtt{Int} = \rho'_{\mathbb{M}}(\kappa) \wedge \rho'_{\mathbb{M}} \in \mathcal{T}[\![e_s]\!] \wedge t \in \rho_{\mathbb{M}}(\kappa) \wedge \rho_{\mathbb{M}} \in \mathcal{T}[\![e_t]\!]\overline{\rho}_{\mathbb{M}} \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\} \cap \ldots \\
 & \overset{31}{=} & \{\rho_{\mathbb{M}} \mid \mathtt{Int} = \rho'_{\mathbb{M}}(\kappa) \wedge \rho'_{\mathbb{M}} \in \mathcal{T}[\![e_s]\!]\overline{\rho}_{\mathbb{M}} \wedge \rho_{\mathbb{M}} \in \mathcal{T}[\![e_t]\!]\overline{\rho}_{\mathbb{M}}\} \cap \{\rho_{\mathbb{M}} \mid \mathtt{Int} = \rho'_{\mathbb{M}}(\kappa) \wedge \rho'_{\mathbb{M}} \in \mathcal{T}[\![e_s]\!]\overline{\rho}_{\mathbb{M}} \wedge \rho_{\mathbb{M}} \in \mathcal{T}[\![e_e]\!]\overline{\rho}_{\mathbb{M}}\}
\end{array}
$$

Figure 6. deriving the type inference algorithm by abstracting the collecting semantics

$$\mathcal{S}[\![\textbf{when } \text{N} \textbf{ in } x \textbf{ then } e_1 \textbf{ else } e_2]\!]\ \rho = (\rho(x) \underline{\in} \mathbb{R} \ \underline{?}\ \text{if } \text{N} \in \textbf{dom}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(\rho(x))) \text{ then } \mathcal{S}[\![e_t]\!]\ \rho \text{ else } \mathcal{S}[\![e_e]\!]\ \rho \ \underline{:}\ \Omega)$$

$$\begin{aligned}\mathcal{C}_1[\![\cdot]\!] &= \{\lambda\rho\,.\,(\rho(x) \underline{\in} \mathbb{R} \ \underline{?}\ \text{if } \text{N} \in \textbf{dom}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(\rho(x))) \text{ then } S_t\rho \text{ else } S_t\rho[x \mapsto \uparrow^{\mathbb{U}}_{\mathbb{R}}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(\rho(x))[\text{N} \mapsto \bot_{\mathbb{U}}])] \ \underline{:}\ \Omega) \mid S_t \in \mathcal{C}_1[\![e_1]\!]\} \\ &\cup \{\lambda\rho\,.\,(\rho(x) \underline{\in} \mathbb{R} \ \underline{?}\ \text{if } \text{N} \in \textbf{dom}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(\rho(x))) \text{ then } S_e\rho[x \mapsto \uparrow^{\mathbb{U}}_{\mathbb{R}}(\exists_{\text{N}}(\downarrow^{\mathbb{U}}_{\mathbb{R}}(\rho(x))))] \text{ else } S_e\rho \ \underline{:}\ \Omega) \mid S_e \in \mathcal{C}_1[\![e_2]\!]\}\end{aligned}$$

$$\begin{aligned}\mathcal{T}[\![\cdot]\!]\overline{\rho}_{\mathbb{M}} &= \mathcal{T}[\![e_1]\!]\{\rho_{\mathbb{M}}[x \mapsto t'_r] \mid t'_r = t_r[\text{N} \mapsto \text{if } \text{N} \in \textbf{dom}(t_r) \text{ then } t_r(\text{N}) \text{ else } t] \wedge t \in \mathbb{M} \wedge t_r = \underline{\{} \ldots \underline{\}} \wedge t_r \in \rho_{\mathbb{M}}(x) \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\} \\ &\cup \mathcal{T}[\![e_2]\!]\{\rho_{\mathbb{M}}[x \mapsto t'_r] \mid t'_r = \exists_{\text{N}}(t_r) \wedge t_r = \underline{\{} \ldots \underline{\}} \wedge t_r \in \rho_{\mathbb{M}}(x) \wedge \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\}\end{aligned}$$

Figure 7. Deriving monotype semantics for the extensions.

D. Deriving Monotype Semantics

We prove Lemma 1 in [7] by computing the monotypes semantics $\mathcal{T}[\![e]\!]$ from the value semantics $\mathcal{C}_1[\![e]\!]$ for each of the constructs $e \in \mathbb{E}$. In other words, we apply the equation $\mathcal{T}[\![e]\!]\ \overline{\rho}_{\mathbb{M}} := \bigcap_{\rho \in \mathbb{X} \to \mathbb{U}_\perp} \{\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho[\kappa \mapsto S\rho]) \mid S \in \mathcal{C}_1[\![e]\!] \wedge \overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)\}$ to each language construct $e \in \mathbb{E}$. The derivation is shown in Fig. 6.

- $\mathcal{T}[\![x]\!]$: The collecting semantics for variable lookup is inlined. Equation (1) inlines the definition of the concretization function, leading to a set comprehension over all sets of monotype environments $\overline{\rho}_{\mathbb{M}} \subseteq \alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho)$ that satisfy the concrete environment ρ. Equation (2) expands the definition of $\alpha^{\mathcal{X}}_{\mathbb{M}1}(\rho[\kappa \mapsto \rho(x)])$, thus returning all monotype environments $\rho'_{\mathbb{M}}$ that are extended with a binding to κ. The types of κ are those of $\rho(x)$ but they are not equal: the first condition $\rho_{\mathbb{M}}(y) \in \alpha_{\mathbb{M}1}(\rho[\kappa \mapsto \rho(x)](y))$ requires that the returned environments agree for each variable (but does not require that the relations between the variables are maintained); the second condition requires that the returned environments have the same relations on all variables except for the new κ variable. In order to illustrate how the types of x and the result κ differ, suppose that $\rho(x)$ is a λ-abstraction (i.e. x itself is a function) that can be applied to values of different types. Then the set of monotypes of κ will eventually be restricted by the monotypes of the value that it is applied to. Since applying κ to some value does not affect the argument value of x, the type of x does not change either. This behavior is reflected by computing an instance of the type of x for κ so that restricting the type of κ does not change the type of x. However, the denotation of x may be a closure that refers to other values in the environment, namely those variables $y \prec_{\mathbb{X}} x$ that are in scope at the definition point of x. Thus, the type of x must be instantiated without losing the relation to the types that might be in the closure of x. In a related work [6] we stipulate that a universal expansion operation can be used to express this instantiation. Equation (3) applies expansion to the variables $y_0, \ldots y_n$ where $x = y_k$ is expanded to κ. The variables $y_i \neq x$ are those that are in scope of the function that x is defined in: if x itself is **let**-bound then $x = y_0$ and y_i are the arguments of x and other functions defined locally in x; if $k > 0$ then x is λ-bound and y_0 is the function that owns the argument x in the sense of Def. 1. (Observe that Eqs. (3) to (5) are tagged with $x \notin \mathbb{X}^\lambda$ meaning they only apply for **let**-bound variables. This is explained below.) The definition of expand and the removal of excess variables y'_i leads to Eq. (4) which is simplified for the case that x is **let**-bound (Eq. (5)). A variation of Eq.(4) can be formulated for λ-bound variables, namely by specializing Eq.(4) for $k > 1$. However, the resulting rule would be incorrect: Consider computing the type of the use of x in $f = \lambda x\,.\,x$ in the (polytype) environment $[f \mapsto \text{a}{\to}\text{b}, x \mapsto \text{a}]$. Computing an instance of x would expand f and x to, say, $f \mapsto \text{c}{\to}\text{d}$ and $x \mapsto \text{c}$. Since there is no more relation between c and the argument type a, the type a is never restricted even when c is unified later. The problem is the direction of value flow: A concrete argument flows into the function where it is applied to a value and may yield a type error. In contrast, in the absence of an actual argument, the requirements from the application site has to be propagated back to the formal argument. However, the computation of the abstract type computation mirrors the computation of the concrete value and hence propagates from the formal argument to the usage site and therefore cannot be used to infer requirements. One solution is to change the direction of information flow by computing the type requirements at all uses of a function argument. This would result in one requirement for each usage. The complete forward semantics requires that the formal argument is instantiated for each usage. While instantiation using expansion is agnostic to the direction, the result always contains more monotypes so that instantiating each requirement leads to requirements that are larger than what is required and, thus, unsound. A sound solution must underapproximate the set of type for which it is safe to call a function. The best type that is smaller than all the requirements it the intersection of all requirements. The intersection of all requirements is unfortunately only approximate. (By not computing the intersection but retaining each requirement separately, a type system with intersection types [4, 8] can be derived.) Computing the intersection of all requirements means forcing all uses of a variable to be equal. This solution can be expressed in a forward computation on monotype environments by stating that κ has the same monotype as x in each environment $\rho_{\mathbb{M}}$. This is the solution adapted in all standard type inferences [3] and expressed by the computation in Eq. (5).

  This leaves us with a typing rule for λ-bound variables that is not backward complete. However, by using the abstraction $\alpha^{\mathcal{X}}_{\mathbb{M}}$ defined elsewhere [6] that explicitly restricts the types of λ-bound variables, a derivation is possible where the shown solution is exact. Thus, with this more abstract $\alpha^{\mathcal{X}}_{\mathbb{M}}$, the rule $\mathcal{T}[\![x]\!] = \{\rho_{\mathbb{M}}[\kappa \mapsto \rho_{\mathbb{M}}(x)] \mid \rho_{\mathbb{M}} \in \overline{\rho}_{\mathbb{M}}\}$ is backward complete.
- $\mathcal{T}[\![\lambda x\,.\,e]\!]$: Equation (7) shows the semantics of the abstraction which is inlined in Eq. (8). The semantics of the function value is computed in Eq. (9) by inlining the definition of $\alpha_{\mathbb{M}1}$ which is translated to types by Eq. (10). The inference of the body is separated in Eq. (11).
- $\mathcal{T}[\![e_1\ e_2]\!]$: The semantics of the function application is inlined in Eq. (8) where an abbreviation **env** is defined that computes the value of each expression. Eq. (9) inlines the definition of $\alpha_{\mathbb{M}1}$. The restriction that v_2 is not a type error and that the result of the function evaluation may not be bottom drop out as they correspond to an empty set of typings, leading to Eq. (14). Substituting t_r for t (since $\alpha_{\mathbb{M}1} \circ \gamma_{\mathbb{M}1}$ is the identity) gives in Eq. (15). Eq. (17) translates this equation to one using a single intersection between environments.
- $\mathcal{T}[\![\textbf{let } x = e \textbf{ in } e']\!]$: After inlining the collecting semantics of the **let**-construct in Eq. (18), lift the value semantics to types, thereby turning the least-fixpoint computation into a greatest-fixpoint computation due to the antitone abstraction $\alpha^{\mathcal{X}}_{\mathbb{M}1}$ (Eq. (19)). Equation (20) lifts the fixpoint computation over the single type to one over environments. Eq. (21) com-

putes a set of environments instead of evaluating the body with singleton environments. The κ symbol is not removed anymore since it is overwritten when evaluating e'.

- $\mathcal{T}[\![c]\!]$: In Eq. (22), inlining the abstraction of a constant $c \in \mathbb{Z}$ binds the type constant `Int` to κ.
- $\mathcal{T}[\![\{\}]\!]$: Equation (23) inlines the abstraction of the empty record. The result is that κ is bound to the monotype $\{\}$ that contains no field.
- $\mathcal{T}[\![@\{\text{N} = e\}]\!]$: The semantic of the record update function is inlined in Eq. (24) which also introduces the abbreviation $\mathbf{env}$ that is used in the following rules. Abstracting the λ-expression yields a type $t_r \to t_u$ where t_r must be a record (Eq. (25)). Equation (26) lifts the remaining value to types where t_u is defined to be the type t_r where the field N is updated to the type of v. The restriction that v is not a type error drops out as it reduces to assert that the set of types of v is non-empty.
- $\mathcal{T}[\![\#\text{N}]\!]$: The semantics of the record selector is inlined in Eq. (27). The query that checks if the field N is in the type of the input r is translated into $\exists i \,.\, \text{N}N_i = \text{N}$ so that the returned set of types is empty if the field is not in the record t_r.
- $\mathcal{T}[\![\mathbf{if}\ e_s\ \mathbf{then}\ e_t\ \mathbf{else}\ e_e]\!]$: Due to the abstracted collecting semantics, Eq. (29) is an intersection of two sets of monotypes, one for each branch. Only the $\mathbf{then}$-branch is shown for brevity. Each branch contains the restriction that e_s must abstract to the type `Int` and that the return type is that of the corresponding branch. Equation (30) translates the value semantics to types. Equation (31) simplifies the type expressions; here, the result of both branches are shown again.

The last lines of the monotype semantics are transcribed in Fig. 6 [7].

E. Semantics of Extensions

Figure 7 shows the concrete, collecting and monotype semantics of the record operation $\mathbf{when}\ \text{N}\ \mathbf{in}\ x\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2$.

References

[1] P. Cousot. Types as Abstract Interpretations. In *Principles of Programming Languages*, pages 316–331, Paris, France, Jan. 1997. ACM. invited paper.

[2] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Principles of Programming Languages*, pages 269–282, San Antonio, Texas, USA, Jan. 1979. ACM.

[3] L. Damas and R. Milner. Principal Type-Schemes for Functional Programs. In *Principles of Programming Languages*, pages 207–212, Albuquerque, New Mexico, USA, 1982. ACM.

[4] T. Jim. What are principal typings and what are they good for? In *Principles of Programming Languages*, pages 42–53, St. Petersburg Beach, Florida, USA, 1996. ACM.

[5] R. Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.

[6] A. Simon. Deriving a Complete Type Inference for Hindley-Milner and Vector Sizes using Expansion. *Science of Computer Programming*, 2014.

[7] A. Simon. Optimal Inference of Fields in Row-Polymorphic Records. In *Programming Language Design and Implementation*, SIGPLAN, Edinburg, UK, June 2014. ACM.

[8] J. B. Wells. The Essence of Principal Typings. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Colloquium on Automata, Languages and Programming*, volume 2380 of *LNCS*, pages 913–925, Malaga, Spain, July 2002. Springer.