Gradual Polymorphic Effects

Full Definitions and Proofs

Matías Toro [*]
PLEIAD Lab
Computer Science Department (DCC)
University of Chile
mtoro@dcc.uchile.cl

Éric Tanter [†]
PLEIAD Lab
Computer Science Department (DCC)
University of Chile
etanter@dcc.uchile.cl

Contents

[*] Funded by CONICYT-PCHA/Magíster Nacional/2013-22131048
[†] Partially funded by Fondecyt project 1150017

1. Introduction

What follows is a formalization of a gradual polymorphic effect system, which works as a privilege checking system. This system combines the work of Lightweight Polymorphic Effects (hereafter, LPE) [4] and a Theory of Gradual Effect Checking (hereafter, TGE) [1] to support gradual effects and effect polymorphism.

Section 2 describes the source language, including its syntax and static semantics. As is usual in accounts of gradually-typed languages [1, 2, 5], the dynamic semantics is given indirectly through a translation to an internal language. The internal language itself is presented in Section 3, and the translation from source programs to programs in the internal language is formalized in Section 4. Section 5 gathers auxiliary definitions. Finally, the proof of type soundness is presented in Section 6.

2. Source Language

We now the core language with integrated support for gradual effect checking and effect polymorphism. The language is inspired by TGE and LPE, is call Gradual Polymorphic Effect System (GPES).

2.1 Syntax

$$\phi \in \textbf{Priv}, \quad \xi \in \textbf{CPriv} = \textbf{Priv} \cup \{¿\}$$
$$\Phi \in \textbf{PrivSet} = \mathcal{P}(\textbf{Priv}), \quad \Xi \in \textbf{CPrivSet} = \mathcal{P}(\textbf{CPriv})$$
$$\begin{array}{rcll} v & ::= & \texttt{unit} \mid (\lambda x\colon T\,.\,e)^{T;\Xi;\overline{x}} & \text{Values} \\ e & ::= & x \mid v \mid e\;e \mid e :: \Xi & \text{Terms} \\ T & ::= & \texttt{Unit} \mid (x\colon T) \xrightarrow[\overline{x}]{\Xi} T & \text{Types} \end{array}$$

Figure 1. Syntax of the source language

Figure 1 presents the syntax of GPES. As in TGE, the language is parameterized on some finite set of privileges **Priv** for a given effect domain. Subeffecting is a partial order on effect privileges, denoted $\phi_1 <: \phi_2$. A consistent privilege, in **CPriv**, can additionally be the unknown privilege ¿. A consistent privilege set Ξ is an element of the power set of **CPriv**, *i.e.* a set of privileges that can include ¿.

A value can either be **`unit`** or a function. The main difference with TGE is that functions are fully annotated, including the type of the argument T_1, the return type T_2, the latent (consistent) privilege set Ξ, and the relative effect variables $\overline{x}$. A term e can be a variable x, a value v, an application $e\;e$, or an effect ascription $e :: \Xi$. A type is either `Unit` or a function type $(x\colon T) \xrightarrow[\overline{x}]{\Xi} T$. Although functions have only one argument, the relative effect variables $\overline{x}$ can include variables defined in the surrounding lexical context.

For instance, in a context Γ where f is defined, a function that takes a function g as argument, performs some output, and applies both f and g, can be defined as follows:

$$(\lambda g\colon \texttt{Unit} \xrightarrow{\top} \texttt{Unit}\;.\;...)^{\texttt{Unit};\{\texttt{@output}\};\{f,g\}}$$

2.2 Static Semantics

The typing rules are presented in Figure 2.

Rule [Var] is self explanatory. Rule [Fn] typechecks the body of the function using the annotated privilege set Ξ_1 and relative effect variables $\overline{x_1}$, and verifies that the type of the body T' is a consistent subtype of the annotated return type T_2.

To type an effect ascription (rule [Eff]), the ascribed privilege set is used to typecheck the inner expression. This rule is the same as in TGE save for the polymorphic context and the fact that is uses consistent subcontainment to check that the ascribed privilege set is valid in the current context.

Rule [App] is an adaptation of the corresponding TGE typing rule to support relative effects. The sub-expressions e_1 and e_2 are typed using adjusted privilege sets (according to each domain). $\widetilde{\textbf{check}}$ verifies that the application is allowed with the given permissions Ξ. A subtlety is that if the invoked function is effect-polymorphic, its latent effects are not only Ξ_1, but also include the latent effects of the relative effect variables of the functions in $\overline{y}$ that are not already present in the polymorphic context $\overline{x}$.

These additional latent effects are computed by the auxiliary function $\textit{latent}_{\Gamma;\overline{x}}(T)$ defined in [3]. The function needs access to both the type environment Γ and the polymorphic context $\overline{x}$ to lookup the types of the relative effect variables. An extra subtlety is that the type of each f in $\overline{y}\backslash\overline{x}$ is obtained in an environment in which the argument y has type T_2, not T_1. This is to account for effect polymorphism: the actual latent effects of the argument come from e_2.

$$\boxed{\Xi; \Gamma; \overline{x} \vdash e \colon T} \qquad \text{Var}\,\frac{\Gamma(x) = T}{\Xi; \Gamma; \overline{x} \vdash x \colon T} \qquad \text{Fn}\,\frac{\Xi_1; \Gamma, x\colon T_1; \overline{x_1} \vdash e \colon T' \qquad T' \lesssim\colon T_2}{\Xi; \Gamma; \overline{x} \vdash (\lambda x\colon T_1 \,.\, e)^{T_2; \Xi_1; \overline{x_1}} \colon (x\colon T_1) \xrightarrow[\overline{x_1}]{\Xi_1} T_2}$$

$$\text{App}\,\frac{\begin{array}{c} \widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_1 \colon (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \\ \widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_2 \colon T_2 \\ \Xi_1{}' = \Xi_1 \cup (\cup_{f \in (\overline{y} \setminus \overline{x})} \mathit{latent}_{\Gamma;\overline{x}}((\Gamma, y\colon T_2)(f))) \\ \Xi_1{}' \mathrel{\underset{\sim}{\sqsubset}}\colon \Xi \qquad T_2 \lesssim\colon T_1 \qquad \widetilde{\mathbf{check}}(\Xi) \end{array}}{\Xi; \Gamma; \overline{x} \vdash e_1\; e_2 \colon T_3}$$

$$\text{AppP}\,\frac{\begin{array}{c} \Gamma(f) = (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \qquad \widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_2 \colon T_2 \\ f \in \overline{x} \qquad T_2 \lesssim\colon T_1 \qquad \widetilde{\mathbf{check}}(\Xi) \end{array}}{\Xi; \Gamma; \overline{x} \vdash f\; e_2 \colon T_3}$$

$$\text{Eff}\,\frac{\Xi_1; \Gamma; \overline{x} \vdash e \colon T \qquad \Xi_1 \mathrel{\underset{\sim}{\sqsubset}}\colon \Xi}{\Xi; \Gamma; \overline{x} \vdash (e :: \Xi_1) \colon T}$$

Figure 2. Type rules of the source language

Rule [AppP] is a new rule for the application of functions that are the parameter of an enclosing effect-polymorphic function (*i.e.* $f \in \overline{x}$). The difference between [AppP] and [App] is very subtle: the typing rule [AppP] does not need to check if the latent effects of the function being applied are consistently subcontained in the set of privileges of the enclosing application. The reason is that in [AppP] the application is being polymorphic on f, meaning that the application is allowed to produce any effect that f may produce.

The typing rules rely on the definitions of subtyping and consistent subtyping presented in Figure 3.

3. Internal Language

GPES leaves many aspects of dynamic privilege checking implicit. This section introduces an internal language, GPESIL, that makes these details explicit. GPES's semantics are then defined by type-directed translation to GPESIL (Section 4).

3.1 Syntax

GPESIL is structured much like GPES but elaborates several concepts as shown in Figure 4.

Following TGE, the internal language includes a new term `Error` to denote runtime effect check failures. The **`has`** operation checks for the availability of particular privilege sets at runtime, and the **`restrict`** operation restricts the privileges available while evaluating its subexpression.

In addition, in order to support effect polymorphism and the cast compilation approach described later, the internal language introduces a number of application operators. First is a polymorphic application operator $\circ$, which is used when translating polymorphic applications $f\ e_2$ in the source language, to $e_f \circ e_2$ (when casts are inserted), in order to not "forget" that the application is effect-polymorphic. Second, new application operators are introduced to denote *primitive applications* that are introduced internally as part of the eta-expansion performed during translation. These applications should not interfere with effect checking (in TGE, where casts are not compiled away but interpreted at runtime, the dynamic semantics use a direct substitution to avoid checking wrapper applications; see Rule [E-Cast-Fn] in [1]). Because once again we need to be able to distinguish effect-polymorphic applications, two new primitive operators are introduced: plain primitive application $\bullet_\Gamma$ and polymorphic primitive application $\bullet$. Note that the Γ in $\bullet_\Gamma$ is only used statically as explained in Section 4. At runtime both primitive applications have the same meaning and the Γ can be erased.

Finally, GPESIL adds the corresponding frames to represent evaluation contexts in the small-step semantics. One for applications and polymorphic applications f. Another frame for errors g. And last, a frame for the primitive operations h.

3.2 Static Semantics

The type system of the internal language is presented in Figure 5. GPESIL mostly extends the source language with a few critical differences.

In the internal language, effectful operations must have enough privileges to be performed. [IApp] and [IAppP] represent the rules for application and polymorphic application. Both rules replace $\widetilde{\mathbf{check}}$ with ***strict-check***, consistent subtyping $\lesssim\colon$ with subtyping $<\colon$, and the consistent containment $\mathrel{\underset{\sim}{\sqsubset}}\colon$ with containment $\subseteq$. Rule [IAppP] new applies to the new polymorphic application operator $\circ$ because polymorphic variables f may be casted during translation and therefore translated into new expressions.

$$\boxed{\Gamma \vdash T' \lesssim: T} \qquad \textrm{CSRefl}\,\frac{}{\Gamma \vdash T \lesssim: T} \qquad \textrm{CSTrans}\,\frac{\Gamma \vdash T_1 \lesssim: T_2 \qquad \Gamma \vdash T_2 \lesssim: T_3}{\Gamma \vdash T_1 \lesssim: T_3}$$

$$\textrm{CSFun}\,\frac{\Gamma \vdash T_1 \lesssim: {T_1}' \qquad \Gamma, x\colon T_1 \vdash (\Xi', [x/x']\overline{x'}) \precsim (\Xi, \overline{x}) \qquad \Gamma, x\colon T_1 \vdash [x/x']{T_2}' \lesssim: T_2}{\Gamma \vdash (x'\colon {T_1}') \xrightarrow[\overline{x}]{\Xi'} {T_2}' \lesssim: (x\colon T_1) \xrightarrow[\overline{x'}]{\Xi} T_2}$$

$$\boxed{\Gamma \vdash (\Xi', \overline{x'}) \precsim (\Xi, \overline{x})} \qquad \textrm{CCnf}\,\frac{\Xi' \underset{\sim}{\sqsubseteq}: \Xi \qquad \forall x' \in \overline{x'}. \Gamma \vdash x' \precsim (\Xi, \overline{x})}{\Gamma \vdash (\Xi', \overline{x'}) \precsim (\Xi, \overline{x})}$$

$$\boxed{\Gamma \vdash x \precsim (\Xi, \overline{x})} \qquad \textrm{CCnfVar}\,\frac{x \in \overline{x}}{\Gamma \vdash x \precsim (\Xi, \overline{x})}$$

$$\textrm{CCnfRel}\,\frac{x \notin \overline{x} \qquad \Gamma(x) = (y\colon T_a) \xrightarrow[\overline{y}]{\Xi_y} T_b \qquad y \notin \overline{x} \qquad \Gamma, y\colon T_a \vdash (\Xi_y, \overline{y}) \precsim (\Xi, \overline{x})}{\Gamma \vdash x \precsim (\Xi, \overline{x})}$$

$$\boxed{\Gamma \vdash T' <: T} \qquad \textrm{SRefl}\,\frac{}{\Gamma \vdash T <: T} \qquad \textrm{STrans}\,\frac{\Gamma \vdash T_1 <: T_2 \qquad \Gamma \vdash T_2 <: T_3}{\Gamma \vdash T_1 <: T_3}$$

$$\textrm{SFun}\,\frac{\Gamma \vdash T_1 <: {T_1}' \qquad \Gamma, x\colon T_1 \vdash (\Xi', [x/x']\overline{x'}) \preceq (\Xi, \overline{x}) \qquad \Gamma, x\colon T_1 \vdash [x/x']{T_2}' <: T_2}{\Gamma \vdash (x'\colon {T_1}') \xrightarrow[\overline{x}]{\Xi'} {T_2}' <: (x\colon T_1) \xrightarrow[\overline{x'}]{\Xi} T_2}$$

$$\boxed{\Gamma \vdash (\Xi', \overline{x'}) \preceq (\Xi, \overline{x})} \qquad \textrm{Cnf}\,\frac{\Xi' \subseteq: \Xi \qquad \forall x' \in \overline{x'}. \Gamma \vdash x' \preceq (\Xi, \overline{x})}{\Gamma \vdash (\Xi', \overline{x'}) \preceq (\Xi, \overline{x})}$$

$$\boxed{\Gamma \vdash x \preceq (\Xi, \overline{x})} \qquad \textrm{CnfVar}\,\frac{x \in \overline{x}}{\Gamma \vdash x \preceq (\Xi, \overline{x})}$$

$$\textrm{CnfRel}\,\frac{x \notin \overline{x} \qquad \Gamma(x) = (y\colon T_a) \xrightarrow[\overline{y}]{\Xi_y} T_b \qquad y \notin \overline{x} \qquad \Gamma, y\colon T_a \vdash (\Xi_y, \overline{y}) \preceq (\Xi, \overline{x})}{\Gamma \vdash x \preceq (\Xi, \overline{x})}$$

$$\boxed{[x/x']T} \qquad \frac{T = (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_y} T_2 \qquad y \notin \{x, x'\}}{[x/x']T = (y\colon [x/x']T_1) \xrightarrow[{[x/x']\overline{y}}]{\Xi_y} [x/x']T_2}$$

$$\boxed{[x/x']\overline{x}} \qquad [x/x']\overline{x} = \overline{y} \text{ where } y_i = \begin{cases} x & \text{if } x_i = x' \\ x_i & \text{otherwise} \end{cases}$$

Figure 3. Subtyping and Consistent subtyping rules

v	$::=$	$\texttt{unit} \mid (\lambda x\colon T \,.\, e)^{T;\Xi;\overline{x}}$	Values
e	$::=$	$x \mid v \mid e\ e \mid e \circ e \mid e \bullet_\Gamma e \mid e \bullet e \mid \texttt{Error} \mid \textbf{has}\ \Phi\ e \mid \textbf{restrict}\ \Xi\ e$	Terms
T	$::=$	$\texttt{Unit} \mid (x\colon T) \xrightarrow[\overline{x}]{\Xi} T$	Types
f	$::=$	$\Box\ e \mid v\ \Box \mid \Box \circ e \mid v \circ \Box$	Frames
g	$::=$	$f \mid h \mid \textbf{has}\ \Phi\ \Box \mid \textbf{restrict}\ \Phi\ \Box$	Error Frames
h	$::=$	$\Box \bullet_\Gamma e \mid v \bullet_\Gamma \Box \mid \Box \bullet e \mid v \bullet \Box$	Primitives Frames

Figure 4. Syntax of the internal language

The primitive applications counterparts of rules [IApp] and [IAppP] rules are rules [IAprm] and [IAprmP] respectively. The major difference is that the primitive rules do not perform a ***strict-check*** given that they are internal artefacts introduced by the translation, and therefore should be "transparent" for static effect checking. To calculate the latent effects of e_1, [IAprm] uses Γ' instead of Γ to use the correct type of y during cast insertion as will be explained later in Section 5.

The **`restrict`** operator constrains its subexpression to be typable with a privilege set that is statically contained in the union of its current privilege set and the latent effects of the relative variables $\overline{x}$. For example the body of a **`map`** function that only produces the effects of its argument Ξ_1, can restrict its body to some privilege set smaller than Ξ_1, otherwise no restrictions could be inserted.

$$\boxed{\Xi; \Gamma; \overline{x} \vdash e : T} \qquad \text{IVar}\frac{\Gamma(x) = T}{\Xi; \Gamma; \overline{x} \vdash x : T} \qquad \text{IUnit}\frac{}{\Xi; \Gamma; \overline{x} \vdash \texttt{unit} : \texttt{Unit}}$$

$$\text{IFn}\frac{\Xi_1; \Gamma, x : T_1; \overline{x_1} \vdash e : T' \qquad T' <: T_2}{\Xi; \Gamma; \overline{x} \vdash (\lambda x : T_1 \, . \, e)^{T_2; \Xi_1; \overline{x_1}} : (x : T_1) \xrightarrow[\overline{x_1}]{\Xi_1} T_2}$$

$$\text{IApp}\frac{\begin{array}{c}\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_1 : (y : T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \\ \widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_2 : T_2 \\ T_2 <: T_1 \qquad |\Xi_1 \cup \mathit{lat}(\Gamma, y : T_2, \overline{y}, \overline{x})| \subseteq: |\Xi| \\ \textbf{\textit{strict-check}}(\Xi)\end{array}}{\Xi; \Gamma; \overline{x} \vdash e_1 \; e_2 : T_3} \qquad \text{IAppP}\frac{\begin{array}{c}\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_1 : (y : T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \\ \widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_2 : T_2 \\ T_2 <: T_1 \qquad \textbf{\textit{strict-check}}(\Xi)\end{array}}{\Xi; \Gamma; \overline{x} \vdash e_1 \circ e_2 : T_3}$$

$$\text{IAprm}\frac{\begin{array}{c}\Xi; \Gamma; \overline{x} \vdash e_1 : (y : T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \\ \Xi; \Gamma; \overline{x} \vdash e_2 : T_2 \\ T_2 <: T_1 \qquad |\Xi_1 \cup \mathit{lat}(\Gamma', \overline{y}, \overline{x})| \subseteq: |\Xi|\end{array}}{\Xi; \Gamma; \overline{x} \vdash e_1 \bullet_{\Gamma'} e_2 : T_3} \qquad \text{IAprmP}\frac{\begin{array}{c}\Gamma(f) = (y : T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \\ \Xi; \Gamma; \overline{x} \vdash e_2 : T_2 \qquad T_2 <: T_1\end{array}}{\Xi; \Gamma; \overline{x} \vdash f \bullet e_2 : T_3} \qquad \text{IHas}\frac{(\Phi \cup \Xi); \Gamma; \overline{x} \vdash e : T}{\Xi; \Gamma; \overline{x} \vdash \textbf{has} \; \Phi \; e : T}$$

$$\text{IRst}\frac{\Xi_1; \Gamma; \overline{x} \vdash e : T \qquad \Xi_1 \leq \Xi \cup (\cup_{f \in \overline{x}} \mathit{latent}_{\Gamma; \overline{x}}(\Gamma(f)))}{\Xi; \Gamma; \overline{x} \vdash \textbf{restrict} \; \Xi_1 \; e : T} \qquad \text{IError}\frac{}{\Xi; \Gamma; \overline{x} \vdash \texttt{Error} : T}$$

Figure 5. Type rules of the internal language

3.3 Dynamic Semantics

GPESIL's dynamic semantics are presented in Figure 6. The evaluation judgement has the form $\Phi \vdash e \rightarrow e'$, meaning that e reduces to e' under the current privilege set Φ. The dynamic operations that are inserted either restrict the current privilege set (**`restrict`**) or check the current privilege set for a gievn effect privilege (**`has`**). These operations are inserted whenever the unknown effect is used in a typing derivation, to enforce the corresponding dynamic checks. If an effect check fails, a runtime effect error is raised.

The [EFrame], [EError] and [EFrameprim] are rules for reducing context frames f, g, and h respectively. The [EApp] and [EAppP] describes how an application of a lambda with a value reduces to the body by replacing the variable x with the value v. Both rules are guarded by a $\mathbf{check}$. Just like [1], if this check fails, then the program is stuck; if programs never get stuck, then any effectful operation that is encountered must have the proper privileges to run. Rules [EApprim] and [EApprimP] are the rules for primitive applications and primitive polymorphic applications respectively. Both rules are identical save for the operation symbol.

The [EHasT] rule reduces the expression e only if the checked privilege set Φ' is contained in the current privilege set. The [EHasV] rule describes how a **`has`** operation applied to a value reduces to the same value (values do not produce effects). In case the checked privilege set is not contained in the current privilege set, rule [EHasF] reduces to an `Error` which is propagated using [EError]. The [ERst] reduces a restricted expression e using the maximal privilege set Φ'' that is subcontained in the current privilege set Φ. The maximal set it is computed using the function *max* as shown in Figure 8 (a direct adaptation of the definition of TGE to account for subeffecting). The [ERstV] removes **`restrict`** on values.

$$\boxed{\Phi \vdash e \to e'}$$

$$\text{EFrame}\;\frac{\mathbf{adjust}(\Phi) \vdash e \to e'}{\Phi \vdash f[e] \to f[e']} \qquad \text{EError}\;\frac{}{\Phi \vdash g[\texttt{Error}] \to \texttt{Error}}$$

$$\text{EApp}\;\frac{\mathbf{check}(\Phi)}{\Phi \vdash (\lambda x \colon T_1 \;.\; e)^{T_2;\Xi_1;\overline{x_1}}\; v \to [v/x]e} \qquad \text{EAppP}\;\frac{\mathbf{check}(\Phi)}{\Phi \vdash (\lambda x \colon T_1 \;.\; e)^{T_2;\Xi_1;\overline{x_1}} \circ v \to [v/x]e}$$

$$\text{EHasT}\;\frac{\Phi' \subseteq \Phi \qquad \Phi \vdash e \to e'}{\Phi \vdash \texttt{has}\;\; \Phi'\; e \to \texttt{has}\;\; \Phi'\; e'} \qquad \text{EHasV}\;\frac{}{\Phi \vdash \texttt{has}\;\; \Phi'\; v \to v} \qquad \text{EHasF}\;\frac{\Phi' \not\subseteq \Phi}{\Phi \vdash \texttt{has}\;\; \Phi\; e \to \texttt{Error}}$$

$$\text{ERst}\;\frac{\Phi'' = \mathit{max}(\{\Phi' \in \gamma(\Xi) \mid \Phi' \subseteq\colon \Phi\} \qquad \Phi'' \vdash e \to e'}{\Phi \vdash \texttt{restrict}\;\; \Xi\; e \to \texttt{restrict}\;\; \Xi\; e'} \qquad \text{ERstV}\;\frac{}{\Phi \vdash \texttt{restrict}\;\; \Xi\; v \to v}$$

$$\text{EFrameprim}\;\frac{\Phi \vdash e \to e'}{\Phi \vdash h[e] \to h[e']} \qquad \text{EAppprim}\;\frac{}{\Phi \vdash (\lambda x \colon T_1 \;.\; e)^{T_2;\Xi_1;\overline{x_1}} \bullet_\Gamma v \to [v/x]e}$$

$$\text{EAppprimP}\;\frac{}{\Phi \vdash (\lambda x \colon T_1 \;.\; e)^{T_2;\Xi_1;\overline{x_1}} \bullet v \to [v/x]e}$$

Figure 6. Evaluation rules of the internal language

$$\boxed{\Xi; \Gamma; \overline{x} \vdash e \Rightarrow e' \colon T}$$

$$\text{TVar}\;\frac{\Gamma(x) = T}{\Xi; \Gamma; \overline{x} \vdash x \Rightarrow x \colon T} \qquad \text{TUnit}\;\frac{}{\Xi; \Gamma; \overline{x} \vdash \texttt{unit} \Rightarrow \texttt{unit}\colon \texttt{Unit}}$$

$$\text{TFn}\;\frac{\Xi_1; \Gamma, x \colon T_1; \overline{x_1} \vdash e \Rightarrow e' \colon T' \qquad T' \lesssim\colon T_2}{\Xi; \Gamma; \overline{x} \vdash (\lambda x \colon T_1 \;.\; e)^{T_2;\Xi_1;\overline{x_1}} \Rightarrow (\lambda x \colon T_1 \;.\; e')^{T_2;\Xi_1;\overline{x_1}} \colon (x \colon T_1) \xrightarrow[\overline{x_1}]{\Xi_1} T_2}$$

$$\text{TApp}\;\frac{\begin{array}{c} \widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_1 \Rightarrow {e_1}' \colon (y \colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \\ \widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_2 \Rightarrow {e_2}' \colon T_2 \\ {\Xi_1}' = \Xi_1 \cup \mathit{lat}(\Gamma, y \colon T_2, \overline{y}, \overline{x}) \qquad {\Xi_1}' \mathrel{\underset{\sim}{\sqsubset}}\colon \Xi \qquad T_2 \lesssim\colon T_1 \\ {e_1}'' = \langle\!\langle (y \colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y \colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \rangle\!\rangle_{\Gamma}^{\mathit{true}} {e_1}' \\ e_1 \notin \overline{x} \qquad \widetilde{\mathbf{check}}(\Xi) \qquad \Phi = \Delta(\Xi) \end{array}}{\Xi; \Gamma; \overline{x} \vdash e_1\; e_2 \Rightarrow \textit{insert-has?}(\Phi, {e_1}''\; {e_2}') \colon T_3}$$

$$\text{TAppP}\;\frac{\begin{array}{c} \Gamma(f) = (y \colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \qquad \widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_2 \Rightarrow {e_2}' \colon T_2 \\ \Gamma_f = \Gamma, f \colon (y \colon T_1) \xrightarrow[f]{\bot} T_3 \\ e_f = \langle\!\langle (y \colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y \colon T_1) \xrightarrow[f]{\bot} T_3 \rangle\!\rangle_{\Gamma_f}^{\mathit{false}} f \\ f \in \overline{x} \qquad T_2 \lesssim\colon T_1 \qquad \widetilde{\mathbf{check}}(\Xi) \qquad \Phi = \Delta(\Xi) \end{array}}{\Xi; \Gamma; \overline{x} \vdash f\; e_2 \Rightarrow \textit{insert-has?}(\Phi, e_f \circ {e_2}') \colon T_3}$$

$$\text{TEff}\;\frac{\Xi_1; \Gamma; \overline{x} \vdash e \Rightarrow e' \colon T \qquad \Xi_1 \mathrel{\underset{\sim}{\sqsubset}}\colon \Xi \qquad \Phi = (|\Xi_1| \backslash |\Xi|)}{\Xi; \Gamma; \overline{x} \vdash (e :: \Xi_1) \Rightarrow \textit{insert-has?}(\Phi, \texttt{restrict}\;\; \Xi_1\; e') \colon T}$$

Figure 7. Transformation rules to the internal language

4. Source to Internal Language Translation

The dynamic semantics of GPES are defined by augmenting its type system to generate GPESIL expressions. The type-directed elaboration judgement has the form $\Xi; \Gamma; \overline{x} \vdash e \Rightarrow e' \colon T$ where e is translated into e'. The translation uses static type and effect information from the source program to determine where runtime checks must be inserted.

Most of this translation is straightforward. Rule [TApp] describes the non-polymorphic function application. There are two main differences compared to [App]. First, a runtime check may be introduced using *insert-has?*, to determine whether the statically-missing privileges in Ξ to perform the application are available at runtime. This privilege set Φ is obtained using the metafunction Δ defined in [1] and presented in Figure 8, which computes the minimal set of additional privileges needed to safely pass the $\widetilde{\textbf{check}}$ verification. The metafunction *insert-has?* inserts a dynamic check for privileges only if the privilege set Φ is not empty. Second, a higher-order cast may be introduced to ensure that ${e_1}'$ has the proper type to accept ${e_2}'$ as argument. A subtlety here is that the relative effects of ${e_1}'$ must be taken into consideration when inserting the cast. The cast is compiled at translation time as seen in Figure 8 and discussed further in Section 5 below.

Rule [TAppP] is the transformation rule for applications of functions that are the parameter of an enclosing effect-polymorphic function. The compiled cast metafunction is inserted with a flag indicating to not insert dynamic checks for the effects of f. Notice how [TAppP] inserts a cast by altering Γ changing the effect information of f to be pure and polymorphic on itself (recursive functions). This way, when the cast is inserted, **`restrict`** , **`has`** and the primitive applications will consider f to be pure. As previously noted, the casted expression e_f may loose the information about being a polymorphic function application in the internal language, hence the application $f\ e_2$ is transformed into an explicitly polymorphic application $\circ$.

5. Auxiliary Functions and Definitions

The auxiliary functions and definitions are presented in Figure 8. The *latent* metafunction calculates the latent effects of a function type. It is the union of the concrete effect Ξ and the latent effects of its relative effects $\overline{y}$ (analysing the relative effects types defined in Γ).

The cast compilation metafunction $\langle\!\langle \cdot \rangle\!\rangle^c_\Gamma$ inserts a cast only if static subtyping does not hold. The first novelty with respect to TGE is the boolean variable c, which indicates whether the cast must include the **`has`** check or not. The second novelty is that casts are transformed away during translation, in contrast to TGE where casts are new forms dealt with in the runtime semantics. For this, if the casted expression e is not a variable, it must be first reduced to a value and then perform the **`has`** and **`restrict`** operations. Therefore, the casted expression e is applied to a new lambda. This new lambda is "artificial" and does not need have permissions to perform an application, therefore the application is actually a primitive application. In case the expression e is a variable, no primitive application must be inserted. Notice that each case of the cast compilation metafunction changes the variabe context Γ so that it includes all free variables needed to compute the latents effects. Also in case of a cast from/to a polymorphic function, Γ is modified so it considers the effects of its argument as the check performed in [App] and [Tapp].

Cast themselves are defined by compilation to function wrappers. The function have two versions: one for non-polymorphic applications ($c = \mathit{true}$), and one for polymorphic applications ($c = \mathit{false}$). The polymorphic version does not insert a **`has`** operation and the primitive application is instead a primitive polymorphic application, i.e. it must not check for f privileges given that we are beign polymorphic on that variable.

The general **`restrict`**/**`has`** scheme is the same as in TGE, except for two crucial differences to regain the flexibility of effect polymorphism. First, the **`has`** check is conditioned to the check flag c using *has?*. For the argument cast, c is true only if the target type of the cast is not polymorphic in its argument x_2, *i.e.* $x_2 \notin \overline{x_2}$. Second, the inserted **`restrict`** and **`has`** must include the latent effects of the relative effect variables of both types, because they represent the maximal privilege set that x_2 and x_1 *may* produce. This adaptation of **`restrict`**/**`has`** corresponds to the flexibility of effect polymorphism: applying a function on which the expression is polymorphic is considered to not produce any effect (so, no **`has`**), but the permitted effects are bounded by the declared latent effects of that function (so, a richer **`restrict`**). Finally, the cast on the return type always inserts a dynamic check (there is no polymorphism on return values). In the translation rule [TApp], the higher-order cast starts with the check flag set to true, because the application is not polymorphic, while in rule [TAppP], the outer check flag is false. Notice that when the flag c is false, the primitive application is instead a primitive polymorphic application, i.e. it must not check for f privilege set given that f is part of the relative efffect variables.

$$\boxed{latent_{\Gamma;\overline{x}}(T)} \qquad \frac{\Xi_p = \cup_{f \in (\overline{y} \setminus \overline{x})} latent_{\Gamma;\overline{x}}((\Gamma, y\colon T_1)(f)))}{latent_{\Gamma;\overline{x}}((y\colon T_1) \xrightarrow[\overline{y}]{\Xi} T_2) = \Xi \cup \Xi_p}$$

$$\langle\!\langle T_2 \Leftarrow T_1 \rangle\!\rangle_{\Gamma}^{c}\ e = \begin{cases} e & \text{if } T_1 <: T_2 \\ (\lambda f\colon T_1 \,.\, \langle T_2 \Leftarrow T_1 \rangle_{\Gamma_l{}'}^{c}\ f)^{T_2{}';\bot;\emptyset} \bullet_{\Gamma} e & \text{if } T_1 \not<: T_2, \text{and } e \neq x \\ \langle T_2 \Leftarrow T_1 \rangle_{\Gamma_l}^{c}\ x & \text{if } T_1 \not<: T_2, \text{and } e = x \end{cases}$$

Where $T_2{}' <: T_2, \Gamma_l = (\Gamma, x_1\colon T_{21}, x_2\colon T_{11}), \Gamma_l{}' = (\Gamma_l, f\colon T_1)$, if $T_1 = (x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}$, and $T_2 = (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} T_{22}$

$$lat(\Gamma, \overline{x_1}, \overline{x}) = (\cup_{f \in (\overline{x_1} \setminus \overline{x})} latent_{\Gamma;\overline{x}}(\Gamma(f)))$$

$$\langle (x_2 : T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} T_{22} \Leftarrow (x_1 : T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12} \rangle_{\Gamma}^{true}\ f =$$

$$(\lambda x\colon T_{21} \,.\, \langle\!\langle T_{22} \Leftarrow T_{12} \rangle\!\rangle_{\Gamma}^{true}\ \mathtt{restrict}\ (\Xi_2 \cup lat(\Gamma, \overline{x_2}, \emptyset))\,\mathtt{has}\ |\Xi_1 \cup\ lat(\Gamma, \overline{x_1}, \overline{x_2})| \setminus |\Xi_2|\ f \bullet_{\Gamma} (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle_{\Gamma}^{x_2 \notin \overline{x_2}}\ x))^{T_{22}{}';\Xi_2;\overline{x_2}}$$

Where $T_{22}{}' <: T_{22}$

$$\langle (x_2 : T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} T_{22} \Leftarrow (x_1 : T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12} \rangle_{\Gamma}^{false}\ f =$$

$$(\lambda x\colon T_{21} \,.\, \langle\!\langle T_{22} \Leftarrow T_{12} \rangle\!\rangle_{\Gamma}^{true}\ \mathtt{restrict}\ (\Xi_2 \cup lat(\Gamma, \overline{x_2}, \emptyset))\, f \bullet (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle_{\Gamma}^{x_2 \notin \overline{x_2}}\ x))^{T_{22}{}';\Xi_2;\overline{x_2}}$$

Where $T_{22}{}' <: T_{22}$

$$insert\text{-}has?(\Phi, e) = \begin{cases} e & \text{if } \Phi = \emptyset \\ \mathtt{has}\ \Phi\ e & \text{otherwise} \end{cases}$$

$$\Delta(\Xi) = \left(\bigcup mins(\{\Phi \in \gamma(\Xi) \mid \mathbf{check}(\Phi)\})\right) \setminus |\Xi|$$

$$mins(\Upsilon) = \left\{\Phi \in \Upsilon \mid \forall \Phi' \in \Upsilon.\ \Phi' \not\subset: \Phi\right\}$$

$$max(\Upsilon) = \{\Phi \in \Upsilon \mid \forall \Phi' \in \Upsilon, \Phi' \subseteq: \Phi\}$$

$$\textbf{\textit{strict-check}}(\Xi) \iff \mathbf{check}(\Phi) \text{ for } all\ \Phi \in \gamma(\Xi).$$

$$\Xi_1 \leq \Xi_2 \iff |\Xi_1| \subseteq: |\Xi_2|$$

Figure 8. Auxiliary functions and definitions used in the gradual polymorphic effect system

6. Type Soundness

This section establishes type soundness of GPES. First we prove soundness of GPESIL (Section 6.1) through progress (Section 6.1.1) and preservation (Section 6.1.2). Then we prove that the translation from GFT to GFTIL preserves typing (Section 6.2), thereby establishing type soundness for GPES. Auxiliary lemmas and propositions used in the proofs of the main theorems are proven in Section 6.3.

6.1 Soundness of Internal Language

6.1.1 Progress

Theorem 1. *(Progress).*
Suppose $\Xi; \emptyset; \emptyset \vdash e\colon T$. Then either e is a value v, an `Error`*, or $\Phi \vdash e \to e'$ for all privilege sets $\Phi \in \gamma(\Xi)$.*

Proof. By structural induction over derivations of $\Xi; \emptyset; \overline{x} \vdash e\colon T$.

Case ([IUnit] and [IFn]). *Both* `unit` *and $(\lambda x\colon T_1 \,.\, e)^{T_2;\Xi_1;\overline{x_1}}$ are values.*

Case ([IVar]). *This case cannot happen by hypothesis.*

Case ([IError]). `Error` *is an* `Error`.

Case ([IRst]). *By induction Hyphothesis, e is either*

- *A value, in which case* [ERstV] *can be applied to* **`restrict`** Ξ' *e.*
- *An error, in which case* [EError] *can be applied with $g =$* **`restrict`** Ξ' $\square$.
- $\forall \Phi' \in \gamma(\Xi'), \Phi' \vdash e \to e'$*, in particular for the Φ'' in the premise of* [ERst]*, thus it can be applied. This Φ'' exists because since $\Xi' \leq \Xi$ and there are no relative effect variables. Thus there $\exists \Phi' \in \gamma(\Xi')$ such that $\Phi' \subseteq\colon \Phi$.*

Case ([IHas]). *. By induction Hypothesis, e is either*

- *a value, in which case* [EHasV] *applies.*
- *An error in which case rule* [EError] *applies with $g =$* **`has`** Φ $\square$.
- $\forall \Phi' \in \gamma(\Phi \cup \Xi), \Phi' \vdash e \to e'$*. We also know that for any $\Phi \in \gamma(\Xi)$, either*
  - $\Phi' \not\subseteq\colon \Phi$*. In this case, rule* [EHasF] *applies.*
  - $\Phi' \subseteq\colon \Phi$*. In this case, since $\Phi' \subseteq\colon \Phi$ and $\Phi \in \gamma(\Xi)$, then also $\Phi \in \gamma(\Phi' \cup \Xi)$. Thus by hypothesis, $\Phi \vdash e \to e'$ and thus we can apply rule* [EHasT].

Case ([IAprmP]). *This case cannot happen by hypothesis.*

Case ([IApp]). *By induction Hypothesis, e_1 is either*

- *An* `Error`*, in which case* [EError] *applies with $g = \square\ e$.*
- $\forall \Phi' \in \gamma(\widetilde{\mathbf{adjust}}(\Xi)), \Phi' \vdash e_1 \to e_1{}'$*. By Theorem 16, since $\Phi \in \gamma(\Xi)$, $\widetilde{\mathbf{adjust}}(\Phi) \in \gamma(\widetilde{\mathbf{adjust}}(\Xi))$ and thus $\widetilde{\mathbf{adjust}}(\Phi) \vdash e_1 \to e_1{}'$ and rule* [EFrame] *can be applied.*
- *A value. By Lemma 15 then $e_1 = (\lambda x\colon T_1 \,.\, e)^{T_2;\Xi_1;\overline{x}}$*
  At the same time, also by induction hyphotesis, e_2 is either:
  - *An* `Error`*, in which case* [EError] *applies with $g = v\ \square$.*
  - $\forall \Phi' \in \gamma(\widetilde{\mathbf{adjust}}(\Xi)), \Phi' \vdash e_2 \to e_2{}'$*. In which case by analogous arguments to the same case for e_1, rule* [EFrame] *can be applied.*
  - *A value. By typing premises, also **strict-check**(Ξ). By definition of **strict-check**, then $\forall \Phi \in \gamma(\Xi).\mathbf{check}(\Phi)$, and thus for any $\Phi \in \gamma(\Xi)$ rule* [EApp] *can also be applied.*

Case ([IAppP]). *By induction Hypothesis, e_1 is either*

- *An* `Error`*, in which case* [EError] *applies with $g = \square\ e$.*
- $\forall \Phi' \in \gamma(\widetilde{\mathbf{adjust}}(\Xi)), \Phi' \vdash e_1 \to e_1{}'$*. By Theorem 16, since $\Phi \in \gamma(\Xi)$, $\widetilde{\mathbf{adjust}}(\Phi) \in \gamma(\widetilde{\mathbf{adjust}}(\Xi))$ and thus $\widetilde{\mathbf{adjust}}(\Phi) \vdash e_1 \to e_1{}'$ and rule* [EFrame] *can be applied.*
- *A value. By Lemma 15 then $e_1 = (\lambda x\colon T_1 \,.\, e)^{T_2;\Xi_1;\overline{x}}$*
  At the same time, also by induction hyphotesis, e_2 is either:

- *An* Error, *in which case* [EError] *applies with* $g = v \circ \Box$.
- $\forall \Phi' \in \gamma(\widetilde{\mathbf{adjust}}(\Xi)), \Phi' \vdash e_2 \to {e_2}'$. *In which case by analogous arguments to the same case for* e_1, *rule* [EFrame] *can be applied.*
- *A value. By typing premises, also* ***strict-check***(Ξ). *By definition of* ***strict-check***, *then* $\forall \Phi \in \gamma(\Xi).\mathbf{check}(\Phi)$, *and thus for any* $\Phi \in \gamma(\Xi)$ *rule* [EAppP] *can also be applied.*

Case ([IApprm]). *By induction Hypothesis,* e_1 *is either*

- *An* Error, *in which case* [EError] *applies with* $g = \Box\ e$.
- $\forall \Phi' \in \gamma(\Xi), \Phi' \vdash e_1 \to {e_1}'$. *Since* $\Phi \in \gamma(\Xi)$ *and thus* $\Phi \vdash e_1 \to {e_1}'$ *and rule* [EFrameprim] *can be applied.*
- *A value. By 15 then* $e_1 = (\lambda x\colon T_1\ .\ e)^{T_2;\Xi_1;\overline{x}}$
  At the same time, also by induction hyphotesis, e_2 *is either:*
  - *An* Error, *in which case* [EError] *applies with* $g = v\ \Box$.
  - $\forall \Phi' \in \gamma(\Xi), \Phi' \vdash e_2 \to {e_2}'$. *In which case by analogous arguments to the same case for* e_1, *rule* [EFrameprim] *can be applied.*
  - *A value. In this case* [EAppprim] *can be applied.*

$\Box$

6.1.2 Preservation

Theorem 2 (Preservation). *If* $\Xi; \Gamma; \overline{x} \vdash e\colon T$, *and* $\Phi \vdash e \to e'$ *for* $\Phi \in \gamma(\Xi)$, *then* $\Xi; \Gamma; \overline{x} \vdash e'\colon T'$ *and* $T' <: T$

Proof. By structural induction over the typing derivation and the applicable evaluation rules.

Case ([IFn], [IUnit], [IVar], [IAppP], [IAprmP] and [IError]). *There rules are triviel since there is no rule in the operational semantics that takes these expressions as premises to step.*

Case ([IApp] and [EFrame] with $f = \Box\ t$). *Thanks to Theorem 16, we can use the induction hypothesis to infer that* $\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash {e_1}'\colon {T_1}' \xrightarrow[\overline{y'}]{{\Xi_1}'} {T_3}'$ *and* ${T_1}' \xrightarrow[\overline{y'}]{{\Xi_1}'} {T_3}' <: T_1 \xrightarrow[\overline{y}]{\Xi'} {T_3}'$. *By definition of subtyping,* $T_1 <: {T_1}'$ *and therefore* $T_2 <: {T_1}'$. *By definition of latent effect and subtyping* $|{\Xi_1}' \cup lat(\Gamma', \overline{y'}, \overline{x})| \subseteq: |\Xi_1 \cup lat(\Gamma', \overline{y}, \overline{x})|$ *and therefore* $|{\Xi_1}' \cup lat(\Gamma', \overline{y'}, \overline{x})| \subseteq: |\Xi|$. *Thus we can reuse rule* [IApp] *to infer that* $\Xi; \Gamma; \overline{x} \vdash {e_1}'\ e_2\colon {T_3}'$ *and we know that* ${T_3}' <: T_3$.

Case ([IApp] and [EFrame] with $f = v\ \Box$). *By Theorem 16 we can use the induction hypothesis to infer that* $\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash {e_2}'\colon {T_2}'$ *and* ${T_2}' <: T_2$.
Since $T_2 <: T_1$, *then also* ${T_2}' <: T_1$ *and we can reuse rule* [IApp] *to infer that* $\Xi; \Gamma; \overline{x} \vdash e_1\ {e_2}'\colon T_3$.

Case ([IApp] and [EApp]). *In this case* $e_1 = (\lambda y\colon T_1\ .\ e)^{T_3;\Xi_1;\overline{y}}$ *and* $\Xi_1; \Gamma, y\colon T_1; \overline{y} \vdash e\colon T_3$.
Thus by Theorem 18, $\Xi_1; \Gamma; \overline{y} \vdash [{}^{e_2}/{}_{y}]\, e\colon T_3$, *with* ${T_3}' <: T_3$. *Then by Proposition 14,* $\Xi; \Gamma; \overline{x} \vdash [{}^{e_2}/{}_{y}]\, e\colon {T_3}'$, ${T_3}' <: T_3$.

Case ([IHas] and [EHasT]). $e = \mathtt{has}\ \Phi\, e'$. *Therefore, application of* [EHasT] *takes the form* $\dfrac{\Phi \subseteq: \Phi' \qquad \Phi' \vdash e' \to e''}{\Phi' \vdash \mathtt{has}\ \Phi\, e' \to \mathtt{has}\ \Phi\, e''}$ *with* $\Phi' \in \gamma(\Xi)$.
Since $\Phi \subseteq: \Phi'$, *then also* $\Phi' \in \gamma(\Phi \cup \Xi)$ *and then by induction hypothesis* $\Phi \cup \Xi; \Gamma; \overline{x} \vdash e''\colon T'$, $T' <: T$. *We can then use rule* [IHas] *to infer that* $\Xi; \Gamma; \overline{x} \vdash \mathtt{has}\ \Phi\, e''\colon T'$ *too.*

Case ([IHas] and [EHasV]). *By induction hypothesis and Lemma 17, in particular* Ξ *instead of* $\Phi \cup \Xi$

Case ([IHas] and [EHasF]). *Trivial by using rule* [IError]

Case ([IRst] and [ERst]). *Since by rule* [ERst] $\Phi'' \in \gamma(\Xi_1)$, *we can use induction hypothesis to infer that* $\Xi_1; \Gamma; \overline{x} \vdash e'\colon T'$, $T' <: T$. *Then we reuse rule* [IRst] *to infer that* $\Xi; \Gamma; \overline{x} \vdash \mathtt{restrict}\ \Xi_1\, e'\colon T$

Case ([IRst] and [ERstV]). *By induction hypothesis and using Lemma 17, in particular* Ξ *instead of* Ξ_1 *(analogous to* [IHas] *and* [EHasV]*).*

Case ([IAprm] and [EFrameprim] with $h = \Box \bullet_{\Gamma'} e$). *We can use induction hypothesis to infer that* $\Xi; \Gamma; \overline{x} \vdash e_1{}' \colon (y\colon T_1{}') \xrightarrow[\overline{y'}]{\Xi_1{}'} T_3{}'$ *and* $(y\colon T_1{}') \xrightarrow[\overline{y'}]{\Xi_1{}'} T_3{}' <\colon (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3$. *By definition of subtyping,* $T_1 <\colon T_1{}'$ *and therefore* $T_2 <\colon T_1{}'$. *By definition of latent effect and subtyping* $|\Xi_1{}' \cup \mathit{lat}(\Gamma', \overline{y'}, \overline{x})| \subseteq\colon |\Xi_1 \cup \mathit{lat}(\Gamma', \overline{y}, \overline{x})|$ *and therefore* $|\Xi_1{}' \cup \mathit{lat}(\Gamma', \overline{y'}, \overline{x})| \subseteq\colon |\Xi|$. *Thus we can reuse rule* [IAprm] *to infer that* $\Xi; \Gamma; \overline{x} \vdash e_1{}'\ e_2 \colon T_3{}'$ *and* $T_3{}' <\colon T_3$.

Case ([IAprm] and [EFrameprim] with $h = v \bullet_{\Gamma'} \Box$). *By Theorem 16 we can use the induction hypothesis to infer that* $\mathbf{adjust}(\Xi); \Gamma; \overline{x} \vdash e_2{}' \colon T_2{}'$ *and* $T_2{}' <\colon T_2$.
Since $T_2 <\colon T_1$, *then also* $T_2{}' <\colon T_1$ *and we can reuse rule* [IAprm] *to infer that* $\Xi; \Gamma; \overline{x} \vdash e_1\ e_2{}' \colon T_3$.

Case ([IAprm] and [EApprim]). *In this case* $e_1 = (\lambda y \colon T_1 \,.\, e)^{T_3; \Xi_1; \overline{y}}$ *and* $\Xi_1; \Gamma, y \colon T_1; \overline{y} \vdash e \colon T_3$.
Thus by Theorem 18, $\Xi_1; \Gamma; \overline{y} \vdash [e_2/y]\, e \colon T_3$, *with* $T_3{}' <\colon T_3$. *Then by Proposition 14,* $\Xi; \Gamma; \overline{x} \vdash [e_2/y]\, e \colon T_3{}'$, $T_3{}' <\colon T_3$.

□

6.2 Translation Preserves Typing

Theorem 3 (Translation preserves typing). *If* $\Xi; \Gamma; \overline{x} \vdash e \Rightarrow e' \colon T$ *in the source language then* $\Xi; \Gamma; \overline{x} \vdash e' \colon T$ *in the internal language.*

Proof. By Case analysis

Case ([TUnit] and [TVar]). *Using the rule premises we can trivially apply rules* [IUnit] *and* [IVar], *respectively.*

Case ([TApp]). *1. By assumption*

(a) $\Xi; \Gamma; \overline{x} \vdash e_1\ e_2 \Rightarrow$ *insert-has?*$(\Phi, e_1{}''\ e_2{}')$

2. *By induction on 1a*
   (a) $\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_1{}' \colon (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3$
   (b) $\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_2{}' \colon T_2$
3. *We also know that* $T_2 \lesssim\colon T_1$ *and* $\Xi_1{}' \sqsubseteq\colon \Xi$, *then* $(y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \lesssim (y\colon T_2) \xrightarrow{\Xi} T_3$.
4. *Since* $e_1{}' \notin \overline{x}$, *then* $\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash \langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \rangle\!\rangle_{\Gamma}^{\mathit{false}} e_1{}' \colon (y\colon T_2{}') \xrightarrow[\overline{z}]{\Xi'} T_3$ *and* $(y\colon T_2{}') \xrightarrow[\overline{z}]{\Xi'} T_3 <\colon (y\colon T_2) \xrightarrow{\Xi} T_3$ *by 1a, 3 and proposition 20.*
5. *Since* $\widetilde{\mathbf{check}}(\Xi)$, *by lemma 19, we know that* ***strict-check***$(\Delta(\Xi) \cup \Xi)$
6. *Finally we proceed on the cases for insert-has?.*
   (a) $\Phi = \emptyset$. *In this case, we also know that* ***strict-check***(Ξ) *because* $\emptyset \cup \Xi = \Xi$. *Then we can apply rule* [IApp] *to infer that* $\Xi; \Gamma; \overline{x} \vdash (\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \rangle\!\rangle_{\Gamma}^{\mathit{false}} e_1{}')\ e_2 \colon T_3$
   (b) $\Phi \neq \emptyset$
      i. $\widetilde{\mathbf{adjust}}(\Delta(\Xi) \cup \Xi); \Gamma; \overline{x} \vdash \langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \rangle\!\rangle_{\Gamma}^{\mathit{false}} e_1{}' \colon (y\colon T_2{}') \xrightarrow[\overline{z}]{\Xi'} T_3$ *by 4, privilege monoticity and subsumption proposition 14*
      ii. $\widetilde{\mathbf{adjust}}(\Delta(\Xi) \cup \Xi); \Gamma; \overline{x} \vdash e_2{}' \colon T_2$ *by 2b, privilege monoticity and subsumption proposition 14*
      iii. $\Delta(\Xi) \cup \Xi; \Gamma; \overline{x} \vdash \left(\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \rangle\!\rangle_{\Gamma}^{\mathit{false}} e_1{}'\right)\ e_2{}' \colon T_3$ *by i, ii, 5 and* [IApp]
      iv. $\Xi; \Gamma; \overline{x} \vdash \mathtt{has}\ \ \Delta(\Xi)\Big(\left(\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3 \rangle\!\rangle_{\Gamma}^{\mathit{false}} e_1{}'\right)\ e_2\Big) \colon T_3$ *by* [IHas]

Case ([TAppP]). *1. By assumption*

(a) $\Xi; \Gamma; \overline{x} \vdash f\ e_2 \Rightarrow$ *insert-has?*$(\Phi, e_f \circ e_2{}')$

2. $\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash e_2{}' \colon T_2$, *by induction on 1a.*
3. *We also know that* $T_2 \lesssim\colon T_1$.
4. *Since* $\widetilde{\mathbf{check}}(\Xi)$, *by 19, we know that* ***strict-check***$(\Delta(\Xi) \cup \Xi)$

5. *We proceed by cases for* $\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle\!\rangle^{false}_{\Gamma} f$

   Case $((y\colon T_1) \xrightarrow[f]{\bot} T_3 <: (y\colon T_2) \xrightarrow{\Xi} T_3)$**.** *Then*

   (a) $\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle\!\rangle^{false}_{\Gamma} f = f$

   (b) *Finally we proceed on the cases for insert-has?.*

   i. $\Phi = \emptyset$*. In this case, we also know that* ***strict-check***(Ξ) *because* $\emptyset \cup \Xi = \Xi$*. We can apply rule* [IAppP]*, to infer that* $\Xi; \Gamma; \overline{x} \vdash f \circ e_2 \colon T_3$.

   ii. $\Phi \neq \emptyset$

   A. $\Gamma(f) = (y\colon T_1) \xrightarrow[\overline{y}]{\Xi_1} T_3$

   B. $\widetilde{\mathbf{adjust}}(\Delta(\Xi) \cup \Xi); \Gamma; \overline{x} \vdash {e_2}' \colon T_2$ *by 2b, privilege monoticity and subsumption proposition* **??**

   C. $\Delta(\Xi) \cup \Xi; \Gamma; \overline{x} \vdash f\ {e_2}' \colon T_3$ *by A, B, 4 and* [IAppP].

   D. $\Xi; \Gamma; \overline{x} \vdash \texttt{has}\ \Delta(\Xi)\Big(\big(\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle\!\rangle^{false}_{\Gamma} f\big) \circ e_2 \Big) \colon T_3$ *by* [IHas]

   Case $((y\colon T_1) \xrightarrow[f]{\bot} T_3 \not<: (y\colon T_2) \xrightarrow{\Xi} T_3)$**.** *Then*

   (a) $\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle\!\rangle^{false}_{\Gamma} f = \langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle^{false}_{\Gamma_l} f$

   (b) $\widetilde{\mathbf{adjust}}(\Xi); \Gamma; \overline{x} \vdash \langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle^{false}_{\Gamma_l} f \colon (y\colon T_2) \xrightarrow{\Xi} T_3$ *from proposition 21.*

   (c) *Finally we proceed on the cases for insert-has?.*

   i. $\Phi = \emptyset$*. In this case, we also know that* ***strict-check***(Ξ) *because* $\emptyset \cup \Xi = \Xi$*. Then we can apply* [IAppP] *to infer that* $\Xi; \Gamma; \overline{x} \vdash (\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle\!\rangle^{false}_{\Gamma} f) \circ e_2 \colon T_3$.

   ii. $\Phi \neq \emptyset$

   A. $\widetilde{\mathbf{adjust}}(\Delta(\Xi) \cup \Xi); \Gamma; \overline{x} \vdash \langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle^{false}_{\Gamma_l} f \colon (y\colon T_2) \xrightarrow{\Xi} T_3$ *by 4, privilege monoticity and subsumption proposition* **??**

   B. $\widetilde{\mathbf{adjust}}(\Delta(\Xi) \cup \Xi); \Gamma; \overline{x} \vdash {e_2}' \colon T_2$ *by 2b, privilege monoticity and subsumption proposition* **??**

   C. $\Delta(\Xi) \cup \Xi; \Gamma; \overline{x} \vdash \big(\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle\!\rangle^{false}_{\Gamma} f\big) \circ {e_2}' \colon T_3$ *by A, B, 4 and* [IAppP].

   D. $\Xi; \Gamma; \overline{x} \vdash \texttt{has}\ \Delta(\Xi)\Big(\big(\langle\!\langle (y\colon T_2) \xrightarrow{\Xi} T_3 \Leftarrow (y\colon T_1) \xrightarrow[f]{\bot} T_3 \rangle\!\rangle^{false}_{\Gamma} f\big) \circ {e_2}' \Big) \colon T_3$ *by* [IHas]

□

6.3 Auxiliary Lemmas and Propositions

All lemmas and propositions that are identical or based on a lemma or proposition of TGE [1] are presented indicating its original number in TGE accompanied by a star "*".

Property 1 (Privilege Monotonicity)**.** *(Property 1*)*

- *If* $\Phi_1 \subseteq: \Phi_2$ *then* $\mathbf{check}(\Phi_1) \Longrightarrow \mathbf{check}(\Phi_2)$*;*
- *If* $\Phi_1 \subseteq: \Phi_2$ *then* $\mathbf{adjust}(\Phi_1) \subseteq: \mathbf{adjust}(\Phi_2)$.

Definition 1 (Consistent Adjust)**.** *(Definition 6*)*
Let $\widetilde{\mathbf{adjust}} : \mathbf{CPrivSet} \to \mathbf{CPrivSet}$ *be defined as follows:*

$$\widetilde{\mathbf{adjust}}(\Xi) = \alpha\left(\{\mathbf{adjust}(\Phi)\,|\Phi \in \gamma(\Xi)\}\right).$$

Lemma 4 (Lemma 12*)**.** $\forall \Phi \in \gamma(\Xi)$, $|\Xi| \subseteq: \Phi$.

Proof. By definition of $|\cdot|$,

$$|\Xi| = \bigcap_{\Phi \in \gamma(\Xi)} \Phi$$

and then the lemma follows by definition of intersection. □

Proposition 5 (Proposition 13*). $|\Xi| = \Xi \backslash \{¿\}$

Proof. By cases on the definition of γ.

Case ($¿ \notin \Xi$). *Then* $|\Xi| = \bigcap\{\Xi\} = \Xi = \Xi\backslash\{¿\}$.

Case ($¿ \in \Xi$). *Then* $|\Xi| = \bigcap\{(\Xi\backslash\{¿\}) \cup \Phi | \Phi \in \mathcal{P}\left(\mathbf{PrivSet}\right)\} = \Xi\backslash\{¿\}$

$\square$

Lemma 6 (Lemma 14*). $|\Xi| \in \gamma(\Xi)$.

Proof. By cases on the definition of γ:

Case ($¿ \notin \Xi$). *Since γ produces a singleton with Ξ, intersection over the singleton retrieves Ξ.*

Case ($¿ \in \Xi$). *Since* $\emptyset \in \mathcal{P}\left(\mathbf{CPrivSet}\right)$, $\Xi \setminus \{¿\} \in \gamma(\Xi)$*, which also is the intersection of every possible set in* $\gamma(\Xi)$.

$\square$

Lemma 7 (Lemma 15*). $\Xi_1 \subseteq: \Xi_2 \Rightarrow \Xi_1 \leq \Xi_2$.

Proof. By Proposition 5 and definition of $\subseteq$, $\Xi_1 \subseteq: \Xi_2$, which is the definition of $\leq$. $\square$

Lemma 8 (Lemma 16*). $\Xi_1 \leq \Xi_2$ *and* ***strict-check***$(\Xi_1) \Rightarrow$ ***strict-check***(Ξ_2)

Proof. Since ***strict-check***(Ξ_1), then $\forall \Phi \in \gamma(\Xi_1)$, **check**$(\Phi)$. In particular, by Lemma 6, **check**$(|\Xi_1|)$. By Privilege Monotonicity Property 1 for **check**, therefore, **check**$(|\Xi_2|)$. Then by Property 1 for **check** and by lemma 4, **check**$(\Phi)\, \forall \Phi \in \Xi_2$ and thus ***strict-check***(Ξ_2). $\square$

Lemma 9 (Lemma 17*). *If* ***strict-check***(Ξ_1) *and* $\Xi_1 \subseteq: \Xi_2$ *then* ***strict-check***(Ξ_2).

Proof. By lemma 7, $\Xi_1 \leq \Xi_2$. Therefore, the lemma follows from Lemma 8. $\square$

Lemma 10 (Lemma 18*). $|\alpha(\Upsilon)| = \bigcap \Upsilon$*, for* $\Upsilon \neq \emptyset$.

Proof. By cases on the definition of $\alpha(\Upsilon)$.

Case ($\Upsilon = \{\Phi\}$ branch). *then* $\Phi = \alpha(\Upsilon)$*, and since* $dom\,(\alpha) = \mathcal{P}\left(\mathbf{PrivSet}\right)$, $¿ \notin \Phi$. *Therefore* $\gamma(\Phi) = \Upsilon$*, and therefore by definition of* $|\cdot|$, $|\alpha(\Upsilon)| = \bigcap \Upsilon$.

Case (otherwise branch). *Then* $\alpha(\Upsilon) = (\bigcap \Upsilon) \cup \{¿\}$. *Thus* $|\alpha(\Upsilon)| = \bigcap\{(\bigcap \Upsilon) \cup \Phi | \Phi \in \mathcal{P}\left(\mathbf{PrivSet}\right)\}$ *and thus* $|\alpha(\Upsilon)| = \bigcap \Upsilon$.

$\square$

Lemma 11 (Lemma 19*). *If* $\bigcap(\Upsilon_1) \in \Upsilon_1$ *and* $\bigcap(\Upsilon_1) \subseteq: \bigcap(\Upsilon_2)$*, then* $\bigcap\{\mathbf{adjust}(\Phi)\,|\forall \Phi \in \Upsilon_1\} \subseteq: \bigcap\{\mathbf{adjust}(\Phi)\,|\forall \Phi \in \Upsilon_2\}$.

Proof. Suppose $\bigcap(\Upsilon_1) \in \Upsilon_1$ and $\bigcap(\Upsilon_1) \subseteq: \bigcap(\Upsilon_2)$. Now suppose $\phi \in \bigcap\{\mathbf{adjust}(\Phi)\,|\forall \Phi \in \Upsilon_1\}$. Then since $\bigcap(\Upsilon_1) \in \Upsilon_1$, in particular $\phi \in \mathbf{adjust}(\bigcap(\Upsilon_1))$ too.

Now let $\Phi \in \Upsilon_2$. Since $\bigcap(\Upsilon_1) \subseteq: \bigcap(\Upsilon_2)$, it follows that $\bigcap(\Upsilon_1) \subseteq: \Phi$. So by monotonicity, $\phi \in \mathbf{adjust}(\Phi)$.

Thus, since Φ is arbitrary, $\phi \in \mathbf{adjust}(\Phi)$ for all $\Phi \in \Upsilon_2$ and thus $\phi \in \bigcap\{\mathbf{adjust}(\Phi)\,|\forall \Phi \in \Upsilon_2\}$. $\square$

Lemma 12 (Lemma 20*). *If* $\Xi_1 \leq \Xi_2$ *then* $\widetilde{\mathbf{adjust}}(\Xi_1) \leq \widetilde{\mathbf{adjust}}(\Xi_2)$

Proof. By definition of $\leq$ and $|\cdot|$, $\bigcap(\gamma(\Xi_1)) \subseteq: \bigcap(\gamma(\Xi_2))$. Also, by Lemma 6, $\bigcap(\gamma(\Xi_1)) \in \gamma(\Xi_1)$. Thus, by Lemma 11, $\bigcap\{\mathbf{adjust}(\Phi)\,|\forall \Phi \in \gamma(\Xi_1)\} \subseteq: \bigcap\{\mathbf{adjust}(\Phi)\,|\forall \Phi \in \gamma(\Xi_2)\}$.

Given that by definition of γ, for any Ξ $\gamma(\Xi) \neq \emptyset$, we can infer by Lemma 10 that $|\alpha(\{\mathbf{adjust}(\Phi)\,|\forall \Phi \in \gamma(\Xi_1)\})| \subseteq: |\alpha(\{\mathbf{adjust}(\Phi|\forall \Phi \in \gamma(\Xi_2))\})|$. By definition of $\widetilde{\mathbf{adjust}}$, this is equivalent to $|\widetilde{\mathbf{adjust}}(\Xi_1)| \subseteq: |\widetilde{\mathbf{adjust}}(\Xi_2)|$, which at the same time is the definition of $\widetilde{\mathbf{adjust}}(\Xi_1) \leq \widetilde{\mathbf{adjust}}(\Xi_2)$. $\square$

Lemma 13 (Lemma 21*). *If* $\Xi_1; \Gamma; \overline{x} \vdash e\colon T$ *and* $\Xi_1 \leq \Xi_2$*, then* $\Xi_2; \Gamma; \overline{x} \vdash e\colon T$.

Proof. By structural induction over the typing derivations for $\Xi_1; \Gamma; \overline{x} \vdash e\colon T$.

Case (Rules [IFn], [IUnit], [IVar], [IError]). *All of these rules do not enforce a restriction between the* Ξ_2 *in the conclusions and any* Ξ *(if existent) in the premises, so the same rule can be directly re-used to infer* $\Xi_2; \Gamma; \overline{x} \vdash e\colon T$.

Case (Rule [IApp]). *By lemma 12, since* $\Xi_1 \leq \Xi_2$, $\widetilde{\mathbf{adjust}}(\Xi_1) \leq \widetilde{\mathbf{adjust}}(\Xi_2)$.
Thus by induction hypothesis, we can infer both that $\widetilde{\mathbf{adjust}}(\Xi_2); \Gamma; \overline{x} \vdash e_1\colon T_1 \xrightarrow[\overline{y}]{\Xi'} T_3$ *and that* $\widetilde{\mathbf{adjust}}(\Xi_2); \Gamma; \overline{x} \vdash e_2\colon T_2$.
By Lemma 8, we also know that ***strict-check***(Ξ_2).
By hypothesis we also know that $T_2 <: T_1$ *and* $|\Xi' \cup \mathit{lat}(\Gamma, \overline{y}, \overline{x})| \subseteq: |\Xi_1|$, *and then we can use rule* [IApprm] *to infer that* $\Xi_2; \Gamma; \overline{x} \vdash e_1\ e_2\colon T_3$.

Case (Rule [IAppP]). *By lemma 12, since* $\Xi_1 \leq \Xi_2$, $\widetilde{\mathbf{adjust}}(\Xi_1) \leq \widetilde{\mathbf{adjust}}(\Xi_2)$.
Thus by induction hypothesis, we can infer both that $\widetilde{\mathbf{adjust}}(\Xi_2); \Gamma; \overline{x} \vdash e_1\colon T_1 \xrightarrow[\overline{y}]{\Xi'} T_3$ *and that* $\widetilde{\mathbf{adjust}}(\Xi_2); \Gamma; \overline{x} \vdash e_2\colon T_2$.
By Lemma 8, we also know that ***strict-check***(Ξ_2).
By hypothesis we also know that $T_2 <: T_1$ *and then we can use rule* [IAppP] *to infer that* $\Xi_2; \Gamma; \overline{x} \vdash e_1 \circ e_2\colon T_3$.

Case (Rule [IAprm]). *By lemma 12, since* $\Xi_1 \leq \Xi_2$, $\widetilde{\mathbf{adjust}}(\Xi_1) \leq \widetilde{\mathbf{adjust}}(\Xi_2)$.
Thus by induction hypothesis, we can infer both that $\widetilde{\mathbf{adjust}}(\Xi_2); \Gamma; \overline{x} \vdash e_1\colon T_1 \xrightarrow[\overline{y}]{\Xi'} T_3$ *and that* $\widetilde{\mathbf{adjust}}(\Xi_2); \Gamma; \overline{x} \vdash e_2\colon T_2$.
By hypothesis we also know that $T_2 <: T_1$ *and* $|\Xi' \cup \mathit{lat}(\Gamma', \overline{y}, \overline{x})| \subseteq: |\Xi_1|$, *and then we can use rule* [IAprm] *to infer that* $\Xi_2; \Gamma; \overline{x} \vdash e_1 \bullet_{\Gamma'} e_2\colon T_3$.

Case (Rule [IAprmP]). *By lemma 12, since* $\Xi_1 \leq \Xi_2$, $\widetilde{\mathbf{adjust}}(\Xi_1) \leq \widetilde{\mathbf{adjust}}(\Xi_2)$.
Thus by induction hypothesis, we can infer that $\widetilde{\mathbf{adjust}}(\Xi_2); \Gamma; \overline{x} \vdash e_2\colon T_2$.
By hypothesis we also know that $T_2 <: T_1$, *and then we can use rule* [IAprmP] *to infer that* $\Xi_2; \Gamma; \overline{x} \vdash f \bullet e_2\colon T_3$.

Case ([IHas]). *Since by hypothesis,* $|\Xi_1| \subseteq: |\Xi_2|$, *in particular we know that* $\Phi \cup |\Xi_1| \subseteq: \Phi \cup \Xi_2$. *We know that* $|\Phi \cup \Xi| = \Phi \cup |\Xi|$, *then* $|\Phi \cup \Xi_1| \subseteq: |\Phi \cup \Xi_2|$ *and thus* $\Phi \cup \Xi_1 \leq \Phi \cup \Xi_2$.
By induction hypothesis, $\Phi \cup \Xi_2; \Gamma; \overline{x} \vdash e\colon T$. *Then we can use rule* [IHas] *to infer that* $\Xi_2; \Gamma; \overline{x} \vdash \texttt{has}\ \Phi\ e\colon T$.

Case (Rule [IRst]). *(*$\Xi_1; \Gamma; \overline{x} \vdash \texttt{restrict}\ \Xi'\ e\colon T$*)*
By hypothesis we know that $\Xi' \leq \Xi_1$ *and thus by transitivity of* $\subseteq$, $\Xi' \leq \Xi_2$. *Therefore, we can use rule* [IRst] *with the premises of the hypothesis to infer that* $\Xi_2; \Gamma; \overline{x} \vdash \texttt{restrict}\ \Xi'\ e\colon T$.

□

Proposition 14 (Subsumption). *(Lemma 22*) If* $\Xi_1; \Gamma; \overline{x} \vdash e\colon T$ *and* $\Xi_1 \subseteq: \Xi_2$, *then* $\Xi_2; \Gamma; \overline{x} \vdash e\colon T$.

Proof. By Lemma 7, $\Xi_1 \leq \Xi_2$. Thus, by String Subsumption Lemma 13, $\Xi_2; \Gamma; \overline{x} \vdash e\colon T$. □

Lemma 15 (Canonical Values). *(Lemma 25*)*

1. *If* $\Xi; \Gamma; \overline{x} \vdash v\colon \texttt{Unit}$, *then* $v = \texttt{unit}$
2. *If* $\Xi; \Gamma; \overline{x} \vdash v\colon T_1 \xrightarrow[\overline{x_1}]{\Xi_1} T_2$, *then* $v = (\lambda x\colon T_1\ .\ e)^{T_2; \Xi_1; \overline{x}}$

Proof. The only rules for typing values in our type system are [IUnit], [IFn] and [IFnprm], respectively. They associate the type premises with the expressions in the conclussions. □

Theorem 16 (Theorem 26*). $\Phi \in \gamma(\Xi) \Rightarrow \mathbf{adjust}(\Phi) \in \gamma(\widetilde{\mathbf{adjust}}(\Xi))$.

Proof. Let $\Phi \in \gamma(\Xi)$. Then $\mathbf{adjust}(\Phi) \in \{\mathbf{adjust}(\Phi') \mid \Phi' \in \gamma(\Xi)\}$.
By Proposition 1, $\{\mathbf{adjust}(\Phi') \mid \Phi' \in \gamma(\Xi)\} \subseteq: \gamma(\alpha(\{\mathbf{adjust}(\Phi') \mid \Phi' \in \gamma(\Xi)\}))$, which by Definition 1 is equivalent to $\gamma(\widetilde{\mathbf{adjust}}(\Xi))$. □

Lemma 17 (Lemma 28*).

1. $\Xi; \Gamma; \overline{x} \vdash v\colon T \Rightarrow \Xi'; \Gamma \vdash v\colon T$
2. $\Xi; \Gamma; \overline{x} \vdash x\colon T \Rightarrow \Xi'; \Gamma \vdash x\colon T$

Proof. 1. We proceed by cases on v.

Case (unit). *Then we can use rule* [IUnit] *for any other* Ξ'.

Case ($(\lambda x\colon T_1 \,.\, e)^{T_2;\Xi_1;\overline{y}}$). *There is only one typing rule for functions. We can reuse the same* [IFn] *To type the function to the same type in a context* Ξ' *by reusing the original premise.*

2. There is only one rule for typing variable identifiers, [IVar]. Since the lemma preserves the environment Γ, we can use rule [IVar] to type the identifier in any Ξ' context.

□

Theorem 18 (Preservation of types under substitution). *(Theorem 29*) If* $\Xi;\Gamma,x\colon T_1;\overline{x} \vdash e_3\colon T_3$ *and* $\Xi;\Gamma;\overline{x} \vdash v\colon T_2$ *with* $T_2 <: T_1$, *then* $\Xi;\Gamma;\overline{x} \vdash [e_2/x]\, e_3\colon T'$ *and* $T' <: T_3$.

Proof. By structural induction over the typing derivation for e_2.

Case ([IUnit] and [IError]). *Trivial since substitution does not change the expression.*

Case ([IVar]). *By definition of substitution, the interesting cases are:*

- $e_3 = y \neq x$ ($[v/x]\, y = y$). *Then by assumption we know that* $\Gamma(y) = T_3$ *and thus we can infer that* $\Xi;\Gamma;\overline{x} \vdash y\colon T_3$.
- $e_3 = x$ ($[v/x]\, x = e_2$). *Then by the theorem hypothesis we know that* $\Xi;\Gamma;\overline{x} \vdash v\colon T_2$. *We also know that* $\Xi;\Gamma,x\colon T_1;\overline{x} \vdash x\colon T_3$, *which means that* $T_3 = T_1$ *and thus* $T' = T_2 <: T_1 = T_3$.

Case ([IFn]).

- $(\lambda x\colon T \,.\, e)^{T_2;\Xi_1;\overline{y}}$. *Then substitution does not affect the body and thus we reuse the original type derivation.*
- $(\lambda y\colon T \,.\, e)^{T_2;\Xi_1;\overline{y}}$ *Then by induction hypothesis, substitution of the body preserves typing and thus rule* [IFn] *can be used to reconstruct the type for the modified expression.*

Case ([IHas] and [IRst]). *Analogous to the case for* [IFn], *since substitution for these expression is defined just as recursive calls to substitution for the premises in the typing rules.*

Case ([IApp]). *By Lemma 17, we can infer that* $\Xi';\Gamma;\overline{x} \vdash v\colon T_2$, *in particular for* $\Xi' = \widetilde{\mathbf{adjust}}(\Xi)$. *Thus we can use our induction hypotheses to in both subexpressions of* $e_3 = e'_1\ e'_2$.

Therefore, while $\widetilde{\mathbf{adjust}}(\Xi)\,;\Gamma;\overline{x} \vdash e'_1\colon (y\colon T'_1) \xrightarrow[\overline{y'}]{\Xi'} T'_3$ *and* $\widetilde{\mathbf{adjust}}(\Xi)\,;\Gamma;\overline{x} \vdash e'_2\colon T'_2$ *with* $T'_2 <: T'_1$ *and* $|\Xi' \cup \mathit{lat}(\Gamma,\overline{y'},\overline{x})| \subseteq: |\Xi|$ *also* $\widetilde{\mathbf{adjust}}(\Xi)\,;\Gamma;\overline{x} \vdash [v/x]\, e'_1\colon T''_1 \xrightarrow[\overline{y''}]{\Xi''} T''_3$ *and* $\widetilde{\mathbf{adjust}}(\Xi)\,;\Gamma;\overline{x} \vdash [v/x]\, e'_2\colon T''_2$ *with* $T''_1 \xrightarrow[\overline{y''}]{\Xi''} T''_3 <: T'_1 \xrightarrow[\overline{y'}]{\Xi'} T'_3$ *and* $T''_2 <: T'_2$.

We therefore know that $T''_2 <: T''_1$, $|\Xi'' \cup \mathit{lat}(\Gamma,\overline{y''},\overline{x})| \subseteq: |\Xi|$ *and we can use rule* [IApp] *to infer back that* $\Xi;\Gamma;\overline{x} \vdash [e_2/x]\, e'_1\ \ [e_2/x]\, e'_2\colon T''_3$, *and by transitivity of subtyping,* $T''_3 <: T_3$.

Case ([IAppP]). *By Lemma 17, we can infer that* $\Xi';\Gamma;\overline{x} \vdash v\colon T_2$, *in particular for* $\Xi' = \widetilde{\mathbf{adjust}}(\Xi)$. *Thus we can use our induction hypotheses to in both subexpressions of* $e_3 = e'_1 \circ e'_2$.

Therefore, while $\widetilde{\mathbf{adjust}}(\Xi)\,;\Gamma;\overline{x} \vdash e'_1\colon (y\colon T'_1) \xrightarrow[\overline{y'}]{\Xi'} T'_3$ *and* $\widetilde{\mathbf{adjust}}(\Xi)\,;\Gamma;\overline{x} \vdash e'_2\colon T'_2$ *with* $T'_2 <: T'_1$ *also* $\widetilde{\mathbf{adjust}}(\Xi)\,;\Gamma;\overline{x} \vdash [v/x]\, e'_1\colon T''_1 \xrightarrow[\overline{y''}]{\Xi''} T''_3$ *and* $\widetilde{\mathbf{adjust}}(\Xi)\,;\Gamma;\overline{x} \vdash [v/x]\, e'_2\colon T''_2$ *with* $T''_1 \xrightarrow[\overline{y''}]{\Xi''} T''_3 <: T'_1 \xrightarrow[\overline{y'}]{\Xi'} T'_3$ *and* $T''_2 <: T'_2$.

We therefore know that $T''_2 <: T''_1$ *and we can use rule* [IAppP] *to infer back that* $\Xi;\Gamma;\overline{x} \vdash [e_2/x]\, e'_1 \circ [e_2/x]\, e'_2\colon T''_3$, *and by transitivity of subtyping,* $T''_3 <: T_3$.

□

Lemma 19 (lemma 33*). $\widetilde{\mathbf{check}}(\Xi) \Rightarrow$ ***strict-check***$(\Delta(\Xi) \cup \Xi)$
i.e. If $\mathbf{check}(\Phi)$ *for some* $\Phi \in \gamma(\Xi)$, *then* $\mathbf{check}(\Phi)$ *for every* $\Phi \in \gamma(\Delta(\Xi) \cup \Xi)$.

Proof. Suppose $\mathbf{check}(\Phi)$ for some $\Phi \in \gamma(\Xi)$
Then $\Upsilon = \{\Phi \in \gamma(\Xi) \mid\ \mathbf{check}(\Phi)\} \neq \emptyset$ so $\Phi = \bigcup \mathit{mins}(\Upsilon)$ exists.
Furthermore, by M & M monoticity, $\mathbf{check}(\Phi)$.
Note that $\Phi \subseteq: \Phi \backslash |\Xi| \cup \Xi = \Delta(\Xi) \cup \Xi$, so if $\Phi_2 \in \gamma(\Delta(\Xi) \cup \Xi)$ then $\Phi \subseteq: \Phi_2$ and by M & M monoticity, $\mathbf{check}(\Phi_2)$. □

Proposition 20. *If $\Xi; \Gamma; \overline{x} \vdash e\colon T_1$, $e \notin \overline{x}$ and $T_1 \lesssim\colon T_2$ in the internal language, then $\Xi; \Gamma; \overline{x} \vdash \langle\!\langle T_2 \Leftarrow T_1 \rangle\!\rangle^{c}_{\Gamma} e\colon {T_2}'$ and ${T_2}' <\colon T_2$.*

Proof. By Case analysis

Case $(T_1 <\colon T_2)$**.** *1. By assumption $\Xi; \Gamma; \overline{x} \vdash e\colon T_1$*
2. $\langle\!\langle T_2 \Leftarrow T_1 \rangle\!\rangle^{c}_{\Gamma} e = e$ by definition of metafunction.
3. $\Xi; \Gamma; \overline{x} \vdash \langle\!\langle T_2 \Leftarrow T_1 \rangle\!\rangle^{c}_{\Gamma} e\colon T_1$ by 1 and 2.

Case $((x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12} \not<\colon (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} T_{22}$ and $e \neq x)$**.** *Where $T_1 = (x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}, T_2 = (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} T_{22}$ and $\Gamma_l = (\Gamma, x_1\colon T_{21}, x_2\colon T_{11}, f\colon T_1)$*

1. $\langle\!\langle T_2 \Leftarrow T_1 \rangle\!\rangle^{c}_{\Gamma} e = (\lambda f\colon T_1 \,.\, \langle T_2 \Leftarrow T_1 \rangle^{c}_{\Gamma_l}\ f)^{{T_2}';\perp;\emptyset} \bullet_{\Gamma} e$
2. $\Xi; \Gamma, f\colon T_1; \overline{x} \vdash \langle T_2 \Leftarrow T_1 \rangle^{c}_{\Gamma_l}\ f\colon {T_2}'$, where ${T_2}' <\colon T_2$ by proposition 21.
3. $\Xi; \Gamma; \overline{x} \vdash (\lambda f\colon T_1 \,.\, \langle T_2 \Leftarrow T_1 \rangle^{c}_{\Gamma_l}\ f)^{{T_2}';\perp;\emptyset}\colon T_1 \xrightarrow{\perp} {T_2}'$ by [IFun]
4. $\Xi; \Gamma; \overline{x} \vdash (\lambda f\colon T_1 \,.\, \langle T_2 \Leftarrow T_1 \rangle^{c}_{\Gamma_l}\ f)^{{T_2}';\perp;\emptyset} \bullet_{\Gamma} e\colon {T_2}'$, and ${T_2}' <\colon T_2$ by [IAprm]

Case $((x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12} \not<\colon (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} T_{22}$ and $e = x)$**.** *Where $T_1 = (x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}, T_2 = (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} T_{22}$ and $\Gamma_l = (\Gamma, x_1\colon T_{21}, x_2\colon T_{11})$*

1. $\langle\!\langle T_2 \Leftarrow T_1 \rangle\!\rangle^{c}_{\Gamma} e = \langle T_2 \Leftarrow T_1 \rangle^{c}_{\Gamma_l}$ by definition of metafunction.
2. $\Xi; \Gamma; \overline{x} \vdash \langle T_2 \Leftarrow T_1 \rangle^{c}_{\Gamma_l}\colon {T_2}'$ where ${T_2}' <\colon T_2$ by proposition 21.
3. $\Xi; \Gamma; \overline{x} \vdash \langle\!\langle T_2 \Leftarrow T_1 \rangle\!\rangle^{c}_{\Gamma} e\colon {T_2}'$ by 1 and 2.

□

Proposition 21. *If $\Xi; \Gamma; \overline{x} \vdash f\colon (x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}$, $x_1 \in \Gamma_l$, $x_2 \in \Gamma_l$, then $\Xi; \Gamma; \overline{x} \vdash \langle (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} T_{22} \Leftarrow (x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12} \rangle^{true}_{\Gamma_l} f\colon (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} {T_{22}}'$, (depending on the cast function, ${T_{22}}' = T_{22}$ or ${T_{22}}' = T_{12}$)*

Proof. Let $\Xi_1^l = \Xi_1 \cup \mathit{lat}(\Gamma_l, \overline{x_1}, \overline{x_2})$ and $\Xi_2^l = \Xi_2 \cup \mathit{lat}(\Gamma_l, \overline{x_2}, \emptyset)$. Let $\Gamma' = \Gamma, x\colon T_2$.

Case (c = true, $|\Xi_1^l| \backslash |\Xi_2^l| \neq \emptyset$)**.**

$$\textsc{IFn}\ \dfrac{\textsc{Prop.2}\ \dfrac{\textsc{IRst}\ \dfrac{\textsc{IHas}\ \dfrac{\textsc{IAprm 1 \& 2}\ \dfrac{\dfrac{\Gamma'(f) = (x_1 : T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}}{|\Xi_1^l| \cup \Xi_2^l; \Gamma'; \overline{x_2} \vdash f\colon (x_1 : T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}}\,\textsc{IVar} \qquad \dfrac{{T_{11}}' \lesssim\colon T_{11}}{|\Xi_1^l| \cup \Xi_2^l; \Gamma'; \overline{x_2} \vdash (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle^{x_2 \notin \overline{x_2}}_{\Gamma}\ x)\colon {T_{11}}'}\,\textsc{Prop.2} \quad {T_{11}}' <\colon T_{11} \quad |\Xi_1 \cup \mathit{lat}(\Gamma_l, \overline{x_1}, \overline{x_2})| \subseteq\colon |\Xi_1^l| \cup |\Xi_2^l|}{|\Xi_1^l| \cup \Xi_2^l; \Gamma'; \overline{x_2} \vdash f \bullet_{\Gamma_l} (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle^{x_2 \notin \overline{x_2}}_{\Gamma}\ x)\colon T_{12}}}{\Xi_2^l; \Gamma'; \overline{x_2} \vdash \textit{insert-has?}(|\Xi_1^l| \backslash |\Xi_2|, f \bullet_{\Gamma_l} (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle^{x_2 \notin \overline{x_2}}_{\Gamma}\ x))\colon T_{12}}}{\Xi_2; \Gamma'; \overline{x_2} \vdash \mathtt{restrict}\ (\Xi_2^l)\textit{insert-has?}(|\Xi_1^l| \backslash |\Xi_2|, f \bullet_{\Gamma_l} (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle^{x_2 \notin \overline{x_2}}_{\Gamma}\ x))\colon T_{12}}}{\Xi_2; \Gamma'; \overline{x_2} \vdash \langle\!\langle T_{22} \Leftarrow T_{12} \rangle\!\rangle^{true}_{\Gamma} \mathtt{restrict}\ (\Xi_2^l)\textit{insert-has?}(|\Xi_1^l| \backslash |\Xi_2|, f \bullet_{\Gamma_l} (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle^{x_2 \notin \overline{x_2}}_{\Gamma}\ x))\colon {T_{22}}'}}{\Xi; \Gamma; \overline{x} \vdash (\lambda x\colon T_{21} \,.\, \langle\!\langle T_{22} \Leftarrow T_{12} \rangle\!\rangle^{true}_{\Gamma} \mathtt{restrict}\ (\Xi_2^l)\textit{insert-has?}(|\Xi_1^l| \backslash |\Xi_2|, f \bullet_{\Gamma_l} (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle^{x_2 \notin \overline{x_2}}_{\Gamma}\ x)))^{{T_{22}}';\Xi_2;\overline{x_2}}\colon (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} {T_{22}}'}$$

Case (c=true, $|\Xi_1^l| \backslash |\Xi_2^l| = \emptyset$)**.** *Trivial by using the same argument for c=true, $|\Xi_1^l| \backslash |\Xi_2^l| \neq \emptyset$.*

Case (c = false). *Let* $\Gamma' = \Gamma, f\colon (x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}$ *and* $\Gamma'' = \Gamma, x\colon T_2$.

$$
\textsc{IFn}\;\dfrac{\textsc{IRst}\;\dfrac{\textsc{IAprm}\;\dfrac{\textsc{IVar}\;\dfrac{\Gamma'(f) = (x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}}{|\Xi_1^l| \cup \Xi_2^l; \Gamma'; \overline{x_2} \vdash f\colon (x_1\colon T_{11}) \xrightarrow[\overline{x_1}]{\Xi_1} T_{12}} \qquad \textsc{Prop.2}\;\dfrac{{T_{11}}' \lesssim\colon T_{11}}{|\Xi_1^l| \cup \Xi_2^l; \Gamma'; \overline{x_2} \vdash (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle_{\Gamma}^{x_2 \notin \overline{x_2}}\, x)\colon {T_{11}}'} \qquad {T_{11}}' <\colon T_{11}}{\Xi_2^l; \Gamma'; \overline{x_2} \vdash f \bullet (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle_{\Gamma}^{x_2 \notin \overline{x_2}}\, x)\colon T_{12}}}{\Xi_2; \Gamma'; \overline{x_2} \vdash \langle\!\langle T_{22} \Leftarrow T_{12} \rangle\!\rangle_{\Gamma}^{true} \mathtt{restrict}\; \left(\Xi_2^l\right) f \bullet (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle_{\Gamma}^{x_2 \notin \overline{x_2}}\, x)\colon {T_{22}}'}}{\Xi; \Gamma; \overline{x} \vdash (\lambda x\colon T_{21}\,.\, \langle\!\langle T_{22} \Leftarrow T_{12} \rangle\!\rangle_{\Gamma}^{true} \mathtt{restrict}\; \left(\Xi_2^l\right) f \bullet (\langle\!\langle T_{11} \Leftarrow T_{21} \rangle\!\rangle_{\Gamma}^{x_2 \notin \overline{x_2}}\, x))^{{T_{22}}'; \Xi_2; \overline{x_2}}\colon (x_2\colon T_{21}) \xrightarrow[\overline{x_2}]{\Xi_2} {T_{22}}'}
$$

$\square$

References

[1] F. Bañados, R. Garcia, and É. Tanter. A theory of gradual effect systems. In *Proceedings of the 19th ACM SIGPLAN Conference on Functional Programming (ICFP 2014)*, pages 283–295, Gothenburg, Sweden, Sept. 2014. ACM Press.

[2] R. Garcia, É. Tanter, R. Wolff, and J. Aldrich. Foundations of typestate-oriented programming. *ACM Transactions on Programming Languages and Systems*, 36(4):12:1–12:44, Oct. 2014.

[3] L. Rytz. *A Practical Effect System for Scala*. PhD thesis, École Polytechnique Fédérale de Lausanne, Sept. 2013.

[4] L. Rytz, M. Odersky, and P. Haller. Lightweight polymorphic effects. In J. Noble, editor, *Proceedings of the 26th European Conference on Object-oriented Programming (ECOOP 2012)*, volume 7313 of *Lecture Notes in Computer Science*, pages 258–282, Beijing, China, June 2012. Springer-Verlag.

[5] J. Siek and W. Taha. Gradual typing for functional languages. In *Proceedings of the Scheme and Functional Programming Workshop*, pages 81–92, Sept. 2006.