Gradual Polymorphic Effects
Complete Definition and Soundness Proof

University of Chile
July 2015

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
1 Introduction 2
2 Source Language 2
  2.1 Syntax ............................................. 2
  2.2 Static Semantics ................................. 2
3 Internal Language 3
  3.1 Syntax ............................................. 3
  3.2 Static Semantics ................................. 5
  3.3 Dynamic Semantics ............................. 5
4 Source to Internal Language Translation 6
5 Auxiliary Functions and Definitions 7
6 Type Soundness 8
  6.1 Soundness of Internal Language .............. 8
    6.1.1 Progress .................................... 8
    6.1.2 Preservation ................................ 9
  6.2 Translation Preserves Typing ............... 10
  6.3 Auxiliary Lemmas and Propositions .......... 11
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) [5] and a Theory of Gradual Effect Checking (hereafter, TGE) [1] to support gradual effects and effect polymorphism. Like in TGE, the system is a generic effect system, following Marino and Millstein [3].

Section 2 describes the source language, including its syntax and static semantics. As is usual in accounts of gradually-typed languages [1, 2, 6], 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

\[
\begin{align*}
\phi & \in \text{Priv}, \quad \xi \in \text{CPriv} = \text{Priv} \cup \{\xi\} \\
\Phi & \in \text{PrivSet} = \mathcal{P}(\text{Priv}), \quad \Xi \in \text{CPrivSet} = \mathcal{P}(\text{CPriv}) \\
v & ::= \text{unit} | \{\lambda x : T . e\}^{T;\Xi;\xi} \quad \text{Values} \\
e & ::= x | v | e e | e :: \Xi \quad \text{Terms} \\
T & ::= \text{Unit} | (x : T) \xrightarrow{\Xi;\xi} T \quad \text{Types}
\end{align*}
\]

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 \text{Priv} for a given effect domain. Subeffecting is a partial order on effect privileges, denoted \phi_1 <: \phi_2. A consistent privilege, in \text{CPriv}, can additionally be the unknown privilege \xi. A consistent privilege set \Xi is an element of the power set of \text{CPriv}, i.e. a set of privileges that can include \xi.

A value can either be \text{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 \(\Xi\), and the relative effect variables \(\pi\). 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 \text{Unit} or a function type \((x : T) \xrightarrow{\Xi;\xi} T\). Although functions have only one argument, the relative effect variables \(\pi\) can include variables defined in the surrounding lexical context.

For instance, in a context \(\Gamma\) 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:

\[
(\lambda g : \text{Unit} \rightarrow \text{Unit} ...) \xrightarrow{\Xi;\xi} \text{Unit} : \{\text{@output}\}; \{f, g\}
\]

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 \(\Xi_1\) and relative effect variables \(\pi_1\), 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 it 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). check verifies that the application is allowed with the given permissions \(\Xi\). A subtlety is that if the invoked function is effect-polymorphic, its latent effects are not only \(\Xi_1\), but also include the latent effects of the relative effect variables of the functions in \(\pi\) that are not already present in the polymorphic context \(\pi\).

These additional latent effects are computed by the auxiliary function \text{latent} : (T) defined in [4]. The function needs access to both the type environment \(\Gamma\) and the polymorphic context \(\pi\) to lookup the types of the relative effect variables. An extra
subtlety is that the type of each \( f \) in \( \overline{y} \setminus \overline{x} \) 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 \( e_2 \).

Rule \([\text{AppP}]\) 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 \([\text{AppP}]\) and \([\text{App}]\) is very subtle: the typing rule \([\text{AppP}]\) 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 \([\text{AppP}]\) the application is being polymorphic on \( f \), meaning that the application is allowed to produce any effect that \( f \) may produce.

\section{Subtyping and Consistent Subtyping}

The typing rules rely on the definitions of subtyping and consistent subtyping presented in Figure 3. The judgement of the consistent subtyping rules has the form \( \Gamma \vdash T' \preceq \Sigma; \Gamma \vdash T \). Rules \([\text{CSRefl}]\) and \([\text{CSTrans}]\) represent the reflexivity and transitivity rules respectively. \( \Gamma \) is used to calculate the privilege sets of the relative effect variables of function types. \([\text{CSFun}]\) represents the rule for consistent subtyping between function types. Let us remember that the latent privilege set of a function typed \( T_1 \vdash T_2 \) consist of two components: the privilege set \( \Xi \), and the latent effects of its relative effect variables \( \overline{x} \). For this, rule \([\text{CSFun}]\) uses the relation \((\Xi', \overline{x}') = (\Xi, \overline{x})\) to compare the effect of two function types. The privilege set \( \Xi' \) must be consistently contained in \( \Xi \) and each relative effect variable \( x' \in \overline{x}' \) is either contained in the relative effect variables \( \overline{x} \), or its type \( \Gamma(x') = (y: T_a) \vdash T_b \) conforms to \((\Xi, \overline{y}) \preceq (\Xi, \overline{x})\) recursively.

Rules \([\text{SRefl}]\), \([\text{STrans}]\), \([\text{SFun}]\) represent the subtyping rules which are identical to the consistent subtyping rules but using subtyping and subcontained operators.

The auxiliary metafunction \([x/x']T\) replaces the relative effect variable \( x' \) with \( x \) in type \( T \).

\section{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).

\subsection{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 new application operator 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 \([\text{E-Cast-Fn}]\) in 41). Because once again we need to be able to distinguish effect-polymorphic applications, the new primitive operator \( \bullet \), is tagged with a variable \( x \) to represent a primitive application of a polymorphic variable \( x \).
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 $h$.

$$x \in \textsf{Var}, T \in \textsf{Type}, v \in \textsf{Value}, e \in \textsf{Term}$$

$$v ::= \text{unit} | (\lambda x: T . e)^{T: \Xi \vdash \Psi}$$

$$e ::= x | v | e | e \cdot x e | \text{Error} | \texttt{has } \Phi e | \texttt{restrict } \Xi e$$

$$T ::= \text{Unit} | (x: T)^{\Xi \vdash \Psi}$$

$f ::= \square e | v \square$

$g ::= f | h | \texttt{has } \Phi \square | \texttt{restrict } \Phi \square$

$h ::= \square e | v \cdot x \square$

$$\cdot \triangleq \cdot \square \text{ where } \forall x \in \textsf{Var}, \bot \neq x$$

Figure 3. Subtyping and Consistent subtyping rules

Figure 4. Syntax of the internal language
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. IA [App] and IA [AppP] represent the rules for application and polymorphic application. Both rules replace check with strict-check, consistent subtyping $\subseteq$, with subtyping $<: $, and the consistent containment $\subseteq$ with containment $\subseteq$. Rule IA [AppP] new applies to the polymorphic application operator $\circ$ because polymorphic variables $f$ may be casted during translation and therefore translated into new expressions.

The primitive applications counterparts of rules IA [App] and IA [AppP] rules are rules IA [Aprm] and IA [AprmP] respectively. The IA [Aprm] represent any effectful operation that is encountered must have the proper privileges to run. The rule IA [E prm] describes how a primitive variable $e$ has the correct privilege set.

The IA [E prm] describe has a primitive variable $e$ has the correct privilege set. The IA [E prm] rule describes how a primitive variable $e$ has the correct privilege set. The IA [E prm] rule describes how a primitive variable $e$ has the correct privilege set.

The IA [E prm] describes how a primitive variable $e$ has the correct privilege set. The IA [E prm] rule describes how a primitive variable $e$ has the correct privilege set. The IA [E prm] rule describes how a primitive variable $e$ has the correct privilege set.

3.3 Dynamic Semantics

GPESIL’s dynamic semantics are presented in Figure 6. The evaluation judgement has the form $\Gamma \vdash e \rightarrow e'$, meaning that $e$ reduces to $e'$ under the current privilege set $\Gamma$. The dynamic operations that are inserted either restrict the current privilege set (restrict) or check the current privilege set for a given 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 IFrame, IError and IFrameprim 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 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. The rule EAprm is the rule for primitive applications respectively.

The EHasT rule reduces the expression $e$ only if the checked privilege set $\Phi'$ 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 \text{EHasF} reduces to an \text{Error} which is propagated using \text{EError}. The \text{ERst} reduces a restricted expression \( e \) using the maximal privilege set \( \Phi'' \) that is subcontained in the current privilege set \( \Phi \). The maximal set it is computed using the function \text{max} as shown in Figure 8 (a direct adaptation of the definition of TGE to account for subeffecting). The \text{ERstV} removes \text{restrict} on values.

### 4. Source to Internal Language Translation

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T \var{e} \rightarrow e' : T )</td>
<td>( T \var{e} \rightarrow e' : T )</td>
</tr>
<tr>
<td>( \text{EFrame} \quad \Phi \vdash e \rightarrow e' )</td>
<td>( \text{EFrameprim} \quad \Phi \vdash e \rightarrow e' )</td>
</tr>
<tr>
<td>( \text{EHasT} \quad \Phi' \subseteq \Phi \quad \Phi \vdash e \rightarrow e' )</td>
<td>( \text{EHasV} \quad \Phi' \vdash \text{has} \Phi' v \rightarrow v )</td>
</tr>
<tr>
<td>( \text{ERst} \quad \Phi'' = \max({\Phi' \in \gamma(\Xi) \mid \Phi' \subseteq \Phi}) \quad \Phi'' \vdash \text{e} \rightarrow e' )</td>
<td>( \text{EFrame} \quad \Phi \vdash e \rightarrow e' )</td>
</tr>
<tr>
<td>( \text{EApp} \quad \Phi \vdash \text{check}(\Phi) )</td>
<td>( \text{EErr} \quad \Phi \vdash \text{Error} \rightarrow \text{Error} )</td>
</tr>
<tr>
<td>( \text{EApp} \quad \Phi \vdash (\lambda x : T_1 \cdot e)T_2;\Xi \cdot \pi v \rightarrow [v/x]e )</td>
<td>( \text{EApp} \quad \Phi \vdash (\lambda x : T_1 \cdot e)T_2;\Xi \cdot \pi \cdot y \rightarrow [v/x]e )</td>
</tr>
</tbody>
</table>

\[ \frac{\Xi; \Gamma \vdash e \Rightarrow e' : T}{\text{TVar}} \quad \frac{\Gamma(x) = T}{\text{TVar}} \quad \frac{\Xi; \Gamma \vdash \text{unit} \Rightarrow \text{unit} : \text{unit}}{\text{TUnit}} \]

\[ \frac{\Xi; \Gamma \vdash e \Rightarrow e' : T'}{\text{TFn}} \quad \frac{\Xi; \Gamma \vdash (\lambda x : T_1 \cdot e)T_2;\Xi \cdot \pi \Rightarrow (\lambda x : T_1 \cdot e')T_2;\Xi \cdot \pi : (x : T_1) \frac{E_{\text{GPE}} - \Xi}{} T_2}{\text{Eq}} \]

\[ \frac{\text{adjust}(\Xi); \Gamma \vdash e_1 \Rightarrow e_1' : (y : T_1) \frac{E_{\text{GPE}} - \Xi}{} T_3}{\text{TApp}} \quad \frac{\Gamma(f) = (y : T_1) \frac{E_{\text{GPE}} - \Xi}{} T_3}{\text{TAppP}} \quad \frac{\Xi; \Gamma \vdash f \Rightarrow \text{insert-has}(\Phi, e_2') : T_3}{\text{TEff}} \]

\[ \frac{\text{adjust}(\Xi); \Gamma \vdash e_2 \Rightarrow e_2' : T_2}{\text{TApp}} \quad \frac{\Xi = \Xi_1 \cup \text{lat}(\Gamma, y : T_2; \pi, \Xi)}{\text{TApp}} \quad \frac{\Xi' = \Xi_1 \cup \text{lat}(\Gamma, y : T_2; \pi, \Xi)}{\text{TAppP}} \quad \frac{\Xi_1 \subseteq \Xi}{\text{TApp}} \quad \frac{\text{check}(\Xi) \quad \Phi = \Delta(\Xi)}{\text{TAppP}} \]

The dynamic semantics of GPES are defined by augmenting its type system to generate GPESIL expressions. The types-directed elaboration judgement has the form \( \Xi, \Gamma; \pi \vdash e : e' : T \) 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 \text{TApp} describes the non-polymorphic function application. There are two main differences compared to \text{App}. First, a runtime check may be introduced using \text{insert-has}, to determine whether the statically-missing privileges in \( \Xi \) to perform the application are available at runtime. This privilege set \( \Phi \) is obtained using the metafunction \( \Delta \) defined in [1] and presented in Figure 8 which computes the minimal set of additional privileges needed to safely pass the \text{check} verification. The metafunction \text{insert-has} inserts a dynamic check for privileges only if the privilege set \( \Phi \) 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 $[\text{TAppP}]$ 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 $f$.

5. Auxiliary Functions and Definitions

\[
\begin{align*}
\text{lat}_\Pi(x)(T) & = \bigcup_{f \in \pi} \text{lat}_{\Pi}((\Pi, y : T_2)(f)) \\
\text{lat}_\Pi((y : T_1) \xrightarrow{\Sigma} T_2) & = \Xi \cup \Xi_p \\

\langle \langle T_2 \Leftarrow T_1 \rangle \rangle & = \begin{cases} 
\text{true} & \text{if } T_1 \ll : T_2 \\
(\lambda f : T_1 . (T_2 \Leftarrow T_1') \langle \langle f \rangle \rangle \text{lat}_{\Pi} x) & \text{if } T_1 \not\ll : T_2, \text{ and } e \neq x \\
(T_2 \Leftarrow T_1') \langle \langle x \rangle \rangle & \text{if } T_1 \not\ll : T_2, \text{ and } e = x
\end{cases}
\end{align*}
\]

Where $T_1' = (x_2 : T_{12}) \xrightarrow{\Sigma_1} T_{12}$, if $T_1 = (x_1 : T_{11}) \xrightarrow{\Sigma_1} T_{12}$, and $T_2 = (x_2 : T_{21}) \xrightarrow{\Sigma_2} T_{22}$

\[
\langle (x_2 : T_{21}) \xrightarrow{\Sigma_2} T_{22} \Leftarrow (x_2 : T_{11}) \xrightarrow{\Sigma_1} T_{12} \rangle \text{true } f = (\lambda x_2 : T_{21} . \langle (x_2 : T_{22}) \xrightarrow{\Sigma_2} T_{22} \Leftarrow (x_2 : T_{11}) \xrightarrow{\Sigma_1} T_{12} \rangle \text{true } \text{restrict } (\Xi_2 \cup \text{lat}(T_{11}, x_2))) \text{has } \Xi_1 \cup \text{lat}(T_{21}, x_2) | \Xi_2 | f \star ((\langle T_11 \Leftarrow T_{21} \rangle \xrightarrow{\Sigma_1} x_2 \not\in \Xi_2) \langle (x_2 : T_{22}) \xrightarrow{\Sigma_2} T_{22} \rangle \text{true } x_2) \text{false } f = (\lambda x_2 : T_{21} . \langle (x_2 : T_{22}) \xrightarrow{\Sigma_2} T_{22} \Leftarrow (x_2 : T_{11}) \xrightarrow{\Sigma_1} T_{12} \rangle \text{false } \text{restrict } (\Xi_2 \cup \text{lat}(T_{11}, x_2))) \text{has } \Xi_1 \cup \text{lat}(T_{21}, x_2) | \Xi_2 | f \star ((\langle T_11 \Leftarrow T_{21} \rangle \xrightarrow{\Sigma_1} x_2 \not\in \Xi_2) \langle (x_2 : T_{22}) \xrightarrow{\Sigma_2} T_{22} \rangle \text{false } x_2)
\]

Where $\Gamma' = (\Gamma, x_2 : T_{21})$

\[
\Delta(\Xi) = \left( \bigcup \text{min}((\Phi \in \gamma(\Xi) | \text{check}(\Phi)) \right) \bigcup |\Xi| \quad \text{strict-check}(\Xi) \iff \text{check}(\Phi) \text{ for all } \Phi \in \gamma(\Xi).
\]

\[
\text{min}(\gamma) = \{ \Phi \in \gamma | \forall \Phi' \in \gamma. \Phi' \not\subseteq \Phi\} \quad \text{max}(\gamma) = \{ \Phi \in \gamma | \forall \Phi' \in \gamma, \Phi' \subseteq \Phi\}
\]

\[
\Xi_1 \leq \Xi_2 \iff |\Xi_1| \subseteq |\Xi_2|
\]

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

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

The cast compilation metafunction $\langle \langle \rangle \rangle$ inserts a cast only if static subtyping does not hold. The first novelty with respect to TGE is the boolean variable $c$, which indicates the cast is for non-polymorphic applications ($c = \text{true}$) or polymorphic applications ($c = \text{false}$). When the cast is for non-polymorphic application, the cast must include the $\text{has}$ operator and must perform a primitive application inside the eta-abstraction (do not check for application privileges). On the other hand if the cast is for polymorphic applications the cast must not include the $\text{has}$ check and must perform a primitive polymorphic application inside the eta-abstraction (do not check for application privileges and do not check that the privilege set of the method is contained in the current privilege set).

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 \texttt{has} and \texttt{restrict} operations. Therefore, the casted expression $e$ is applied to a new lambda. In case the expression $e$ is a variable, no primitive application must be inserted.

The general \texttt{restrict/has} scheme is the same as in TGE, except for two crucial differences to regain the flexibility of effect polymorphism. First, the \texttt{has} check is conditioned to the check flag $c$ as previously mentioned. Second, the inserted \texttt{restrict} and \texttt{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 \texttt{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 \texttt{has}), but the permitted effects are bounded by the declared latent effects of that function (so, a richer \texttt{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 \texttt{true}, because the application is not polymorphic, while in rule [TAppP], the outer check flag is \texttt{false}.

The \texttt{insert-has?} metafunction only inserts a \texttt{has} if $\Phi$ is not empty. $\Delta$ calculates the minimal static privilege set necessary to safely pass the \texttt{check} function. mins and max metafunctions calculates the minimal and maximal privilege set over a set of privilege set. \texttt{strict-check}(Ξ) is defined as safely pass the \texttt{check} function for all concrete privilege set $\Phi$ contained in $\gamma(Ξ)$. Finally the $\leq$ operator is defined as consistent subcontainment between two concrete privilege sets.

## 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 GPES to GPESIL 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 $Ξ; \emptyset; \emptyset \vdash e : T$. Then either $e$ is a value $v$, an Error, or $\Phi \vdash e \rightarrow e'$ for all privilege sets $\Phi \in \gamma(Ξ)$.

**Proof.** By structural induction over derivations of $Ξ; \emptyset; \pi \vdash e : T$.

**Case [(IUnit) and [IFn]].** Both \texttt{unit} and $(\lambda x : T_1 . e)^T_2 ; \Xi_1 ; \Pi_1$ are values.

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

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

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

- A value, in which case \texttt{ERstV} can be applied to \texttt{restrict} $Ξ'$ $e$.
- An error, in which case \texttt{ERst} can be applied with $g = \texttt{restrict} Ξ' □$.
- $\forall \Phi' \in \gamma(Ξ'), \Phi' \vdash e \rightarrow e'$, in particular for the $\Phi''$ in the premise of \texttt{ERst}, thus \texttt{ERst} can be applied. This $\Phi''$ exists because $Ξ' \leq Ξ$ and the polymorphic context is empty. Thus, $\exists \Phi' \in \gamma(Ξ')$ such that $\Phi' \subseteq Φ$.

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

- a value, in which case \texttt{EHasV} applies.
- An error in which case rule \texttt{EHasF} applies with $g = \texttt{has} Φ □$.
- $\forall \Phi' \in \gamma(Φ \cup Ξ), \Phi' \vdash e \rightarrow e'$. We also know that for any $\Phi \in \gamma(Ξ)$, either
  - $Φ' \not\subseteq Φ$. In this case, rule \texttt{EHasF} applies.
  - $Φ' \subseteq Φ$. In this case, since $Φ' \subseteq Φ$ and $Φ \in \gamma(Ξ)$, then also $Φ \in \gamma(Φ' \cup Ξ)$. Thus by hypothesis, $Φ \vdash e \rightarrow e'$ and thus we can apply rule \texttt{EFrame}.

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

- An Error, in which case \texttt{EHasT} applies with $g = □ e$.
- $\forall Φ' \in \gamma(\text{adjust}(Ξ)), \Phi' \vdash e_1 \rightarrow e'_1$. By Theorem 16 since $Φ \in \gamma(Ξ)$, $\text{adjust}(Φ) \in \gamma(\text{adjust}(Ξ))$ and thus $\text{adjust}(Φ) \vdash e_1 \rightarrow e'_1$ and rule \texttt{EFrame} can be applied.
A value. By Lemma [15], then $e_1 = (\lambda x : T_1 \cdot e)_{T_2 ; \Xi_1 ; \pi}$.

At the same time, also by induction hypothesis, $e_2$ is either:

- An Error, in which case [EError] applies with $q = v \square$.
- $\forall \Phi' \in \gamma(\text{adjust}(\Xi)), \Phi' \vdash e_2 \to e'_2$. 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($\Xi$). By definition of strict-check, then $\forall \Phi \in \gamma(\Xi).\check{\Phi}$, and thus for any $\Phi \in \gamma(\Xi)$ rule [EApp] can also be applied.

Case ([IAprm]). By induction hypothesis, $e_1$ is either:

- An Error, in which case [EError] applies with $h = \square e$.
- $\forall \Phi' \in \gamma(\Xi), \Phi' \vdash e_1 \to e'_1$. Since $\Phi \in \gamma(\Xi)$ and thus $\Phi \vdash e_1 \to e'_1$ and rule [EFrameprim] can be applied.
- A value. By Lemma [15], then $e_1 = (\lambda x : T_1 \cdot e)_{T_2 ; \Xi_1 ; \pi}$.

At the same time, also by induction hypothesis, $e_2$ is either:

- An Error, in which case [EError] applies with $h = v \square$.
- $\forall \Phi' \in \gamma(\Xi), \Phi' \vdash e_2 \to e'_2$. In which case by analogous arguments to the same case for $e_1$, rule [EFrameprim] can be applied.
- A value. In this case [EApp] can be applied.

\begin{proof}

6.1.2 Preservation

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

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

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

Case ([IApp] and [EFrame] with $f = \square t$). Thanks to Theorem [16] we can use the induction hypothesis to establish that $\text{adjust}(\Xi); \Gamma; \pi \vdash e'_1 : T'_1 \quad \Xi'_1 \triangleleft \Xi'_2$ and $T'_1 \rightarrow \Xi'_2 \rightarrow T'_3$. By definition of subtyping, $T_1 <: T_i'$ and therefore $T_2 <: T'_i$. By definition of latent effect and subtyping $|\Xi'_1 \cup \text{lat}(\Gamma; \pi, \Xi)| \subseteq |\Xi \cup \text{lat}(\Gamma; \pi)|$ and therefore $|\Xi'_1 \cup \text{lat}(\Gamma; \pi, \Xi)| \subseteq |\Xi|$. Thus we can reuse rule [IApp] to establish that $\Xi; \Gamma; \pi \vdash e_1 \to e'_2 : T_2'$ and we know that $T_2' <: T_3$.

Case ([IApp] and [EFrame] with $f = v \square$). By Theorem [16] we can use the induction hypothesis to establish that $\text{adjust}(\Xi); \Gamma; \pi \vdash e'_2 : T_2'$ and $T_2' <: T_2$.

Since $T_2 <: T_1$, then also $T_2' <: T_1$ and we can reuse rule [IApp] to establish that $\Xi; \Gamma; \pi \vdash e_1 e'_2 : T_3$.

Case ([IApp] and [EApp]). In this case $e_1 = (\lambda y : T_1 \cdot e)_{T_3 ; \Xi_1 ; \pi}$ and $\Xi_1; \Gamma, y : T_1; \pi \vdash e : T_3$.

Thus by Theorem [18] $\Xi_1; \Gamma; \pi \vdash [\varepsilon y / e] : T_3$, with $T_3' <: T_3$. Then by Proposition [14] $\Xi_1; \Gamma; \pi \vdash e_2 y : T_3'$, $T_3' <: T_3$.

Case ([IHas] and [EHasT]). $e = \textbf{has } \Phi e'$. Therefore, application of [EHasT] takes the form $\overline{\Phi \subseteq \Phi'} \Phi' \vdash e' \to e'' \overline{\Phi'} \vdash \textbf{has } \Phi e''$ with $\Phi' \in \gamma(\Xi)$.

Since $\Phi \subseteq \Phi'$, then also $\Phi' \in \gamma(\Phi \cup \Xi)$ and then by induction hypothesis $\Phi \cup \Xi; \Gamma; \pi \vdash e'' : T', T' <: T$. We can then use rule [IHas] to establish that $\Xi; \Gamma; \pi \vdash \textbf{has } \Phi e'' : T'$ too.

Case ([IHas] and [EHasV]). $e = \textbf{has } \Phi v$. $\Xi; \Gamma; \pi \vdash e : T$ and $\Phi \vdash \textbf{has } \Phi v \vdash v$. By induction hypothesis $(\Phi \cup \Xi); \Gamma; \pi \vdash v : T$. Using Lemma [17] we can conclude that $\Xi; \Gamma; \pi \vdash v : T$.

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 the induction hypothesis to establish that $\Xi_1; \Gamma; \pi \vdash e' : T', T' <: T$. Then we reuse rule [IRst] to establish that $\Xi_1; \Gamma; \pi \vdash \textbf{restrict } \Xi_1 e' : T$.

Case ([IRst] and [ERstV]). By induction hypothesis and using Lemma [17] using the same argument of [IHas] and [EHasV].

Case ([IAprm] and [EFrameprim] with $h = \square e$). Same argument of case [IApp] and [EFrame] but using [IAprm] instead of [IApp].
Case ([IAprm] and [EFrameprim] with \( h = v \cdot e \)). By Theorem \([16]\) we can use the induction hypothesis to establish that \( \text{adjunct}(\Xi; \Gamma; \bar{x} \vdash e_1'; \bar{T}_2' \text{ and } \bar{T}_2' < : T_2) \).

Since \( \bar{T}_2' < : T_1 \), then also \( \bar{T}_2' < : T_1 \) and we can reuse rule [IAprm] to establish that \( \Xi; \Gamma; \bar{x} \vdash v \cdot e_2' : T_3 \).

Case ([IAprm] and [EAPrime]). In this case \( e_1 = (\lambda y : T_1 . e) \bar{T}_3; \Xi \) and \( e_2 = v \).

1. By assumption

(a) \( \Xi; \Gamma; \bar{x} \vdash (\lambda y : T_1 . e) \bar{T}_3; \Xi \times v : T_3 \)

(b) \( \Xi; \Gamma; y : T_1 \vdash e : T_3 \)

(c) \( \Xi; \Gamma; \bar{x} \vdash v : T_2 \) and \( T_2 < : T_1 \).

(d) \( \Xi \uplus \text{lat}(\Gamma, \bar{y}, \bar{x}) | \subseteq | \Xi \)

2. We need to prove that \( \Xi; \Gamma; \bar{x} \vdash [v/y] e : T_3' \) with \( T_3' < : T_3 \).

3. We proceed by cases for \( [v/y] e \).

(a) \( y \notin \bar{x} \)

i. By Theorem \([18]\) \( \Xi; \Gamma; y \vdash [v/y] e : T_3' \), with \( T_3' < : T_3 \)

ii. By (1.d) and Proposition \([14]\) \( \Xi; \Gamma; \bar{x} \vdash [v/y] e : T_3', T_3' < : T_3 \).

(b) \( y \in \bar{x} \)

i. By Theorem \([9]\) \( \Xi \uplus \text{lat}(\Gamma, y) | \subseteq | \Xi \)

ii. By definition of latent function \( \text{lat}(\Gamma, y, \bar{x}) = \text{lat}(\Gamma, y : T_2), \bar{y}, \bar{x}) \)

iii. As \( y \notin \bar{x} \) and \( y \in \bar{x} \), by definition of latent function \( \text{lat}(\Gamma, y : T_2), \bar{y}, \bar{x}) \subseteq \text{lat}(\Gamma, y) \)

iv. By (3.b.ii), (1.d) and Proposition \([14]\) \( \Xi; \Gamma; \bar{x} \vdash [v/y] e : T_3', T_3' < : T_3 \).

\( \square \)

6.2 Translation Preserves Typing

Theorem 3 (Translation preserves typing). If \( \Xi; \Gamma; \bar{x} \vdash e \Rightarrow e' : T \) in the source language then \( \Xi; \Gamma; \bar{x} \vdash e' : T \) 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) \( \Xi; \Gamma; \bar{x} \vdash e_1 e_2 \Rightarrow \text{insert-has?}(\Phi, e_1'' e_2') \)

2. By induction on \( \text{Ia} \)

(a) \( \text{adjust}(\Xi) ; \Gamma; \bar{x} \vdash e_1' : (y : T_1) \xrightarrow{\Xi} T_3 \)

(b) \( \text{adjust}(\Xi) ; \Gamma; \bar{x} \vdash e_2' : T_2 \)

3. We also know that \( T_2 \subseteq : T_1 \) and \( \Xi_1' \subseteq : \Xi \), then \( (y : T_1) \xrightarrow{\Xi} T_3 \subseteq (y : T_2) \xrightarrow{\Xi} T_3 \).

4. Since \( e_1' \notin \bar{x} \), then \( \text{adjust}(\Xi) ; \Gamma; \bar{x} \vdash (\lambda y : T_2) \xrightarrow{\Xi} T_3 \Rightarrow (y : T_2) \xrightarrow{\Xi} T_3 \xrightarrow{\Xi_1'} (y : T_2') \xrightarrow{\Xi} T_3 \text{ and } (y : T_2') \xrightarrow{\Xi} T_3 < : (y : T_2) \xrightarrow{\Xi} T_3 \text{ by Ia, 3 and proposition } [21] \)

5. Since \( \text{check}(\Xi) \), by lemma \([20]\) we know that \textbf{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 \textbf{strict-check}(\Xi) \) because \( \emptyset \cup \Xi = \Xi \). Then we can apply rule [IApp] to establish that \( \Xi; \Gamma; \bar{x} \vdash (\lambda y : T_2) \xrightarrow{\Xi} T_3 \Rightarrow (y : T_1) \xrightarrow{\Xi_1} T_3 \xrightarrow{\Xi_1'} e_1'; (y : T_2') \xrightarrow{\Xi} T_3 \text{ and } \)

(b) \( \Phi \neq \emptyset \)

i. \( \text{adjust}(\Delta(\Xi) \cup \Xi) ; \Gamma; \bar{x} \vdash (\lambda y : T_2) \xrightarrow{\Xi} T_3 \Rightarrow (y : T_1) \xrightarrow{\Xi_1} T_3 \xrightarrow{\Xi_1'} e_1' ; (y : T_2') \xrightarrow{\Xi} T_3 \text{ by 4, privilege monotonicity and subsumption proposition } [14] \)

ii. \( \text{adjust}(\Delta(\Xi) \cup \Xi) ; \Gamma; \bar{x} \vdash e_2' ; T_2 \text{ by 2b, privilege monotonicity and subsumption proposition } [14] \)

iii. \( \Delta(\Xi) \cup \Xi ; \Gamma; \bar{x} \vdash (\lambda y : T_2) \xrightarrow{\Xi} T_3 \Rightarrow (y : T_1) \xrightarrow{\Xi_1} T_3 \xrightarrow{\Xi_1'} e_2' ; T_3 \text{ by i, ii, 5 and [IApp] } \)
iv. $\Xi; \Gamma; x \vdash \text{has } \Delta(\Xi) \left( \langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} e_1} e_2 \right) : T_3$ by [IHas]

Case (TApp). 1. By assumption
(a) $\Xi; \Gamma; x \vdash f_2 \Rightarrow \text{insert-has?}(\Phi, e_1 e_2')$
(b) $f \in \pi$
2. adjust($\Xi$); $\Gamma; x \vdash e_2': T_2$. by induction on 1a.
3. We also know that $T_3 \ll T_1$.
4. Since check($\Xi$), by 2b we know that strict-check($\Delta(\Xi) \cup \Xi$)
5. We proceed by cases for $\langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f}$

Case ($(y: T_1) \xrightarrow{e_1} T_3 \ll (y: T_2) \xrightarrow{f} T_3$), Then
(a) $\langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f} = f$
(b) Finally we proceed on the cases for insert-has?.
   i. $\Phi = \emptyset$. In this case, we also know that strict-check($\Xi$) because $\emptyset \cup \Xi = \Xi$. We can apply rule [IAppP], to establish that $\Xi; \Gamma; x \vdash f_2; T_3$.
   ii. $\Phi \neq \emptyset$
      A. $\Gamma(f) = (y: T_1) \xrightarrow{e_1} T_3$
      B. adjust($\Delta(\Xi) \cup \Xi$); $\Gamma; x \vdash e_2': T_2$ by 2b, privilege monotonicity and subsumption proposition 14
      C. $\Delta(\Xi) \cup \Xi; \Gamma; x \vdash f_2': T_3$ by A, B, 1b, 4 and [IAppP].
      D. $\Xi; \Gamma; x \vdash \text{has } \Delta(\Xi) \left( \langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f} e_2 \right) : T_3$ by [IHas]

Case ($(y: T_1) \xrightarrow{e_1} T_3 \ll (y: T_2) \xrightarrow{f} T_3$), Then
(a) $\langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f} = f$
(b) adjust($\Xi$); $\Gamma; x \vdash \langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f} : (y: T_2) \xrightarrow{f} T_3$ from proposition 22
(c) For 1b: $\Xi \cup \text{lat}(\Gamma, \{f\}, x) \subseteq \Xi$
(d) Finally we proceed on the cases for insert-has?.
   i. $\Phi = \emptyset$. In this case, we also know that strict-check($\Xi$) because $\emptyset \cup \Xi = \Xi$. Then we can apply [IApp] to establish that $\Xi; \Gamma; x \vdash \langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f} e_2; T_3$.
   ii. $\Phi \neq \emptyset$
      A. adjust($\Delta(\Xi) \cup \Xi$); $\Gamma; x \vdash \langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f} ; (y: T_2) \xrightarrow{f} T_3$ by 4, privilege monotonicity and subsumption proposition 14
      B. adjust($\Delta(\Xi) \cup \Xi$); $\Gamma; x \vdash e_2': T_2$ by 2b, privilege monotonicity and subsumption proposition 14
      C. $\Delta(\Xi) \cup \Xi; \Gamma; x \vdash \langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f} e_2': T_3$ by A, B, 4 and (c).
      D. $\Xi; \Gamma; x \vdash \text{has } \Delta(\Xi) \left( \langle \langle y: T_2 \rangle \xrightarrow{y} T_3 \right. \left. \leadsto (y: T_1) \xrightarrow{e_1} T_3 \rangle \xrightarrow{\text{false} f} e_2 \right)' : T_3$ by [IHas]

6.3 Auxiliary Lemmas and Propositions

Property 1 (Privilege Monotonicity).
• If $\Phi_1 \subseteq \Phi_2$ then check($\Phi_1$) $\Rightarrow$ check($\Phi_2$);
• If $\Phi_1 \subseteq \Phi_2$ then adjust($\Phi_1$) $\subseteq$ adjust($\Phi_2$).
Definition 1 (Consistent Adjust).
Let \( \tilde{\text{adjust}} : \text{CPrivSet} \to \text{CPrivSet} \) be defined as follows:
\[
\tilde{\text{adjust}}(\Xi) = \alpha(\{\text{adjust}(\Phi) | \Phi \in \gamma(\Xi)\}).
\]

Lemma 4. \( \forall \Phi \in \gamma(\Xi), |\Xi| \subseteq \Phi. \)

Proof. By definition of \(|\cdot|\),
\[
|\Xi| = \bigcap_{\Phi \in \gamma(\Xi)} \Phi
\]
and then the lemma follows by definition of intersection.

Proposition 5. \( |\Xi| = \Xi \setminus \{\xi\} \)

Proof. By cases on the definition of \(\gamma\).

Case \(\xi \notin \Xi\). Then \( |\Xi| = \bigcap \{\Xi\} = \Xi = \Xi \setminus \{\xi\} \).
Case \(\xi \in \Xi\). Then \( |\Xi| = \bigcap \{\Xi \setminus \{\xi\} \cup \Phi | \Phi \in \mathcal{P}(\text{PrivSet})\} = \Xi \setminus \{\xi\} \)

Lemma 6. \( |\Xi| \in \gamma(\Xi) \)

Proof. By cases on the definition of \(\gamma\):

Case \(\xi \notin \Xi\). Since \(\gamma\) produces a singleton with \(\Xi\), intersection over the singleton retrieves \(\Xi\).
Case \(\xi \in \Xi\). Since \(\emptyset \in \mathcal{P}(\text{CPrivSet})\), \(\Xi \setminus \{\xi\} \in \gamma(\Xi)\), which also is the intersection of every possible set in \(\gamma(\Xi)\).

Lemma 7. \( \Xi_1 \subseteq \Xi_2 \Rightarrow \Xi_1 \leq \Xi_2 \)

Proof. By Proposition 5 and definition of \(\subseteq\), \(\Xi_1 \subseteq \Xi_2\), which is the definition of \(\leq\).

Lemma 8. \( \Xi_1 \leq \Xi_2 \text{ and } \text{strict-check}(\Xi_1) \Rightarrow \text{strict-check}(\Xi_2) \)

Proof. Since \(\text{strict-check}(\Xi_1)\), then \(\forall \Phi \in \gamma(\Xi_1), \text{check}(\Phi)\). In particular, by Lemma 6 \(\text{check}(|\Xi_1|)\). By Privilege Monotonicity Property 1 for \(\text{check}\), therefore, \(\text{check}(|\Xi_2|)\). Then by Property 1 for \(\text{check}\) and by Lemma 2 \(\text{check}(\Phi) \forall \Phi \in \Xi_2 \) and thus \(\text{strict-check}(\Xi_2)\).

Lemma 9. If \(\text{strict-check}(\Xi_1)\) and \(\Xi_1 \subseteq \Xi_2\) then \(\text{strict-check}(\Xi_2)\).

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

Lemma 10. \( |\alpha(\Upsilon)| = \bigcap \Upsilon, \text{ for } \Upsilon \neq \emptyset \)

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

Case \(\Upsilon = \{\Phi\} \) branch. then \(\Phi = \alpha(\Upsilon)\), and since \(\text{dom}(\alpha) = \mathcal{P}(\text{PrivSet})\), \(\xi \notin \Phi\). Therefore \(\gamma(\Phi) = \Upsilon\), and therefore by definition of \(|\cdot|\), \(|\alpha(\Upsilon)| = \bigcap \Upsilon\).
Case (otherwise branch). Then \(\alpha(\Upsilon) = (\bigcap \Upsilon) \cup \{\xi\} \). Thus \(|\alpha(\Upsilon)| = \bigcap \{((\bigcap \Upsilon) \cup \Phi) | \Phi \in \mathcal{P}(\text{PrivSet})\}\) and thus \(|\alpha(\Upsilon)| = \bigcap \Upsilon\).

Lemma 11. If \(\bigcap(\Upsilon_1) \in \Upsilon_1 \) and \(\bigcap(\Upsilon_1) \subseteq \bigcap(\Upsilon_2)\), then \(\bigcap \{\text{adjust}(\Phi) | \forall \Phi \in \Upsilon_1\} \subseteq \bigcap \{\text{adjust}(\Phi) | \forall \Phi \in \Upsilon_2\}\).
Proof. Suppose \( \bigcap(Y_1) \in Y_1 \) and \( \bigcap(Y_1) \subseteq \bigcap(Y_2) \). Now suppose \( \phi \in \{ \text{adjust}(\Phi) | \forall \Phi \in Y_1 \} \). Then since \( \bigcap(Y_1) \in Y_1 \), in particular \( \phi \in \text{adjust}(\bigcap(Y_1)) \) too.

Now let \( \Phi \in Y_2 \). Since \( \bigcap(Y_1) \subseteq \bigcap(Y_2) \), it follows that \( \bigcap(Y_1) \subseteq \Phi \). So by monotonicity, \( \phi \in \text{adjust}(\Phi) \).

Thus, since \( \Phi \) is arbitrary, \( \phi \in \text{adjust}(\Phi) \) for all \( \Phi \in Y_2 \) and thus \( \phi \in \{ \text{adjust}(\Phi) | \forall \Phi \in Y_2 \} \).

\[ \square \]

**Lemma 12.** If \( \Xi_1 \leq \Xi_2 \) then \( \text{adjust}(\Xi_1) \leq \text{adjust}(\Xi_2) \)

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

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

\[ \square \]

**Lemma 13.** If \( \Xi_1 ; \Gamma ; \pi \vdash e : T \) and \( \Xi_1 \cup \text{lat}(\Gamma, \pi, \pi) \leq \Xi_2 \), then \( \Xi_2 ; \Gamma ; \pi \vdash e : T \).

Proof. By structural induction over the typing derivations for \( \Xi_1 ; \Gamma ; \pi \vdash e : T \).

**Case (Rules [IFn], [IUnit], [IVar], [IERror]).** All of these rules do not enforce a restriction between the \( \Xi_2 \) in the conclusions and any \( \Xi \) (if existent) in the premises, so the same rule can be directly re-used to infer \( \Xi_2 ; \Gamma ; \pi \vdash e : T \).

**Case (Rule [IApp]).** By Lemma 12 since \( \Xi_1 \cup \text{lat}(\Gamma, \pi, \pi) \leq \Xi_2 \), \( \text{adjust}(\Xi_1 \cup \text{lat}(\Gamma, \pi, \pi)) \leq \text{adjust}(\Xi_2) \). By property \( \text{adjust}(\Xi_1) \cup \text{lat}(\Xi_2) \leq \text{adjust}(\Xi_1 \cup \text{lat}(\Gamma, \pi, \pi)) \), then \( \text{adjust}(\Xi_1) \cup \text{lat}(\Gamma, \pi, \pi) \leq \Xi_2 \).

Thus by induction hypothesis, we can infer both that \( \text{adjust}(\Xi_2) ; \Gamma ; \pi \vdash e_1 : T_1 \xrightarrow{\xi} T_3 \) and that \( \text{adjust}(\Xi_2) ; \Gamma ; \pi \vdash e_2 : T_2 \).

By Lemma 8 we also know that \( \text{strict-check}(\Xi_2) \).

By hypothesis we also know that \( T_2 < : T_1 \) and \( |\Xi' \cup \text{lat}(\Gamma, \pi, \pi)| \subseteq |\Xi_1| \), and then we can use rule [IAppm] to establish that \( \Xi_2 ; \Gamma ; \pi \vdash e_1 e_2 : T_3 \).

**Case (Rule [IAApp]).** By Lemma 12 since \( \Xi_1 \cup \text{lat}(\Gamma, \pi, \pi) \leq \Xi_2 \), \( \text{adjust}(\Xi_1 \cup \text{lat}(\Gamma, \pi, \pi)) \leq \text{adjust}(\Xi_2) \). By property \( \text{adjust}(\Xi_1) \cup \text{lat}(\Xi_2) \leq \text{adjust}(\Xi_1 \cup \text{lat}(\Gamma, \pi, \pi)) \), then \( \text{adjust}(\Xi_1) \cup \text{lat}(\Gamma, \pi, \pi) \leq \Xi_2 \).

Thus by induction hypothesis, we can infer both that \( \text{adjust}(\Xi_2) ; \Gamma ; \pi \vdash f : T_1 \xrightarrow{\xi} T_3 \) and that \( \text{adjust}(\Xi_2) ; \Gamma ; \pi \vdash e_2 : T_2 \).

By Lemma 8 we also know that \( \text{strict-check}(\Xi_2) \).

By hypothesis we also know that \( T_2 < : T_1 \) and then we can use rule [IAppP] to establish that \( \Xi_2 ; \Gamma ; \pi \vdash f e_2 : T_3 \).

**Case (Rule [IAprm]).** Similar argument to rule [IApp] without the need of lemma 8.

**Case (Rule [IAprmP]).** Similar argument to rule [IAppP] without the need of lemma 8.

**Case ([IHas]).** Since by hypothesis, \( |\Xi_1 \cup \text{lat}(\Gamma, \pi, \pi)| \subseteq |\Xi_2| \), in particular we know that \( \Phi \cup |\Xi_1 \cup \text{lat}(\Gamma, \pi, \pi)| \subseteq \Phi \cup \Xi_2 \).

We know that \( |\Phi \cup |\Xi_1 \cup \text{lat}(\Gamma, \pi, \pi)| \subseteq \Phi \cup \Xi_1 \) and thus \( \Phi \cup \Xi_1 \cup \text{lat}(\Gamma, \pi, \pi) \subseteq \Phi \cup \Xi_2 \).

By induction hypothesis, \( \Phi \cup \Xi_1 \cup \text{lat}(\Gamma, \pi, \pi) \leq \Xi_2 \).

**Case (Rule [IRst]).** (\( \Xi_1 \vdash \text{restrict } \Xi' : e : T \))

By hypothesis we know that \( \Xi' \leq \Xi_1 \cup \text{lat}(\Gamma, \pi, \emptyset) \). By definition of lat function, the relation \( \Xi_1 \cup \text{lat}(\Gamma, \pi, \emptyset) \leq \Xi_2 \) is equivalent to \( \Xi_1 \cup \text{lat}(\Gamma, \pi, \emptyset) \leq \Xi_2 \cup \text{lat}(\Gamma, \pi, \emptyset) \). Thus by transitivity of \( \subseteq \), \( \Xi' \leq \Xi_2 \cup \text{lat}(\Gamma, \pi, \emptyset) \). Therefore, we can use rule [IRst] with the premises of the hypothesis to establish that \( \Xi_2 ; \Gamma ; \pi \vdash \text{restrict } \Xi' : e : T \).

\[ \square \]

**Proposition 14** (Subsumption). If \( \Xi_1 ; \Gamma ; \pi \vdash e : T \) and \( \Xi_1 \cup \text{lat}(\Gamma, \pi, \pi) \subseteq \Xi_2 \), then \( \Xi_2 ; \Gamma ; \pi \vdash e : T \).

Proof. By Lemma 7 \( \Xi_1 \leq \Xi_2 \). Thus, by Strong Subsumption Lemma 13 \( \Xi_2 ; \Gamma ; \pi \vdash e : T \).

\[ \square \]

**Lemma 15** (Canonical Values). 1. If \( \Xi ; \Gamma ; \pi \downarrow \text{Unit} \), then \( \pi = \text{unit} \).

2. If \( \Xi ; \Gamma ; \pi \downarrow : T_1 \xrightarrow{T_1} T_2 \), then \( \pi = (\lambda x : T_1 \cdot e)_{T_2 \Xi_1} \).

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 conclusions.

\[ \square \]
Theorem 16. \( \Phi \in \gamma(\Xi) \Rightarrow \text{adjust}(\Phi) \in \gamma(\text{adjust}(\Xi)) \).

Proof. Let \( \Phi \in \gamma(\Xi) \). Then \( \text{adjust}(\Phi) \in \{\text{adjust}(\Phi') \mid \Phi' \in \gamma(\Xi)\} \).

By Proposition \( [\text{adjust}(\Phi') \mid \Phi' \in \gamma(\Xi)] \subseteq \gamma(\alpha(\{\text{adjust}(\Phi') \mid \Phi' \in \gamma(\Xi)\})) \), which by Definition \( [\] is equivalent to \( \gamma(\text{adjust}(\Xi)) \). \( \square \)

Lemma 17.

1. \( \Xi; \Gamma; \overline{x} \vdash v: T \Rightarrow \Xi'; \Gamma; \overline{x} \vdash v: T \)
2. \( \Xi; \Gamma; \overline{x} \vdash x: T \Rightarrow \Xi'; \Gamma; \overline{x} \vdash x: T \)

Proof. 1. We proceed by cases on \( v \).

Case (\text{unit}). Then we can use rule [\text{IUnit}] for any other \( \Xi' \).

Case (\( (\lambda x: T_1 . e) T_2 ; \Xi : \overline{x} \)). 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 \( \Xi' \) by reusing the original premise.

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

\( \square \)

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

Proof. By structural induction over the typing derivation for \( e_2 \).

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

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

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

Case ([IApp]).

* (\( \lambda x: T . e \)) \( T_2 ; \Xi : \overline{x} \). Then substitution does not affect the body and thus we reuse the original type derivation.
* (\( \lambda y: T . e \)) \( T_2 ; \Xi : \overline{x} \). 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 \( [\] we can infer that \( \Xi'; \Gamma; \overline{x} \vdash \overline{T_2} \), in particular for \( \Xi' = \text{adjust}(\Xi) \). Thus we can use our induction hypotheses in both subexpressions of \( e_3 = e_1' e_2' \).

Therefore, while \( \text{adjust}(\Xi); \Gamma; \overline{x} \vdash e_1': (y: T_1') \xrightarrow{\overline{\Xi}} T_3' \) and \( \text{adjust}(\Xi); \Gamma; \overline{x} \vdash e_2': T_2' \) with \( T_2' <: T_1' \) and \( |\Xi' \cup \text{lat}(\Gamma, \overline{y}, \overline{x})| \subseteq |\Xi| \) also \( \text{adjust}(\Xi); \Gamma; \overline{x} \vdash [\overline{x} / x] e_1': T''_1 \xrightarrow{\overline{\Xi}} T_3'' \) and \( \text{adjust}(\Xi); \Gamma; \overline{x} \vdash [\overline{x} / x] e_2': T''_2 \) with \( T''_1 \xrightarrow{\overline{\Xi}} T''_2 \)

We therefore know that \( T''_2 <: T''_1 \), \( |\Xi' \cup \text{lat}(\Gamma, \overline{y}, \overline{x})| \subseteq |\Xi| \) and we can use rule [IApp] to infer back that \( \Xi; \Gamma; \overline{x} \vdash [\overline{x} / x] e_1' e_2' \) and \( T''_3 <: T_3 \).

Case ([IApp]). By Lemma \( [\] we can infer that \( \Xi'; \Gamma; \overline{x} \vdash \overline{T_2} \), in particular for \( \Xi' = \text{adjust}(\Xi) \). Thus we can use our induction hypotheses in both subexpressions of \( e_3 = f e_2 \). Also, given that \( x \notin \overline{x} \), and \( f \in \overline{x} \) then \( f \neq x \).

Therefore, while \( \Gamma(f) = (y: T_1') \xrightarrow{\overline{\Xi}} T_3' \) and \( \text{adjust}(\Xi); \Gamma; \overline{x} \vdash e_2': T_2' \) with \( T_2' <: T_1' \) also \( \text{adjust}(\Xi); \Gamma; \overline{x} \vdash [\overline{x} / x] f = f: T_1' \xrightarrow{\overline{\Xi}} T_3' \) and \( \text{adjust}(\Xi); \Gamma; \overline{x} \vdash [\overline{x} / x] e_2': T''_2 \) with \( T''_2 <: T_2' \).
We therefore know that $T''_2 < : T'_1$ and we can use rule [IAppP] to infer back that $\Xi; \Gamma; x \vdash [v/x] e \phi_2 : T''_3$, and by transitivity of subtyping, $T''_5 < : T_3$.

Case (IApPm) and (IApPmP). Same argument of rules [IApp] and [IAppP] respectively.

\[\square\]

**Theorem 19** (Preservation of types under substitution for polymorphic abstractions). If $\Xi; \Gamma; x : T_1; \pi \vdash e_3 : T_3$ and $\Xi; \Gamma; x \vdash v : T_2$ with $x \in \pi$ and $T_2 < : T_1$, then $\Xi \cup \text{latent}_{\Gamma; \theta}(T_2); \Gamma; \pi \setminus \{x\} \vdash [v/x] e_3 : T' \text{ and } T'' < : T_3$.

**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). Similar to case (IVar) of Theorem 18

Case (IFn).

\[
\begin{align*}
&\cdot (\lambda x : T \cdot e)_{T_2 \cdot \Xi} \equiv \Xi. \\
&\cdot (\lambda y : T \cdot e)_{T_2 \cdot \Xi} \equiv \Xi.
\end{align*}
\]

Then substitution does not affect the body and thus we reuse the original type derivation.

\[
\begin{align*}
&\cdot (\lambda y : T \cdot e)_{T_2 \cdot \Xi} \equiv \Xi.
\end{align*}
\]

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 $\Xi' ; \Gamma; \pi \setminus \{x\} \vdash v : T_2$, in particular for $\Xi' = \text{adjust}(\Xi)$. Thus we can use our induction hypotheses in both subexpressions of $e_3 = e_1' e_2'$.

Therefore, while $\text{adjust}(\Xi) ; \Gamma; \pi \vdash e_1' : (y ; T'_1) \overset{\Xi'}{\rightarrow} T'_3$ and $\text{adjust}(\Xi) ; \Gamma; \pi \vdash e_2' ; T'_2$ with $T'_2 < : T'_1$, $\Gamma' = \Gamma ; y ; T'_2$ and $| \Xi \cup \text{lat}(\Gamma ; \gamma ; \pi)| \subseteq | \Xi \cup \text{adjust}(\Xi) \cup \text{latent}_{\Gamma; \theta}(T_2); \Gamma; \pi \setminus \{x\} \vdash [v/x] e_3' : T'_1 \overset{\Xi'}{\rightarrow} T'_3$ and $\text{adjust}(\Xi) \cup \text{latent}_{\Gamma; \theta}(T_2); \Gamma; \pi \setminus \{x\} \vdash [v/x] e_3' : T'_2$ and $T''_2 < : T'_2$.

We also know that the adjustment function is monotonically increasing, therefore $\text{adjust}(\Xi) \cup \text{latent}_{\Gamma; \theta}(T_2) \subseteq \text{adjust}(\Xi \cup \text{latent}_{\Gamma; \theta}(T_2))$.

By proposition 14 $\text{adjust}(\Xi \cup \text{latent}_{\Gamma; \theta}(T_2)) ; \Gamma; \pi \setminus \{x\} \vdash [v/x] e_3' : T'_1 \overset{\Xi'}{\rightarrow} T'_3$ and $\text{adjust}(\Xi) \cup \text{latent}_{\Gamma; \theta}(T_2)) ; \Gamma; \pi \setminus \{x\} \vdash [v/x] e_3' : T'_2$.

We know that $T''_2 < : T'_1$, $| \Xi' \cup \text{lat}(\Gamma ; y ; T'_2 ; x ; T_2), \overline{\gamma}; \pi(x)| \subseteq | \Xi' \cup \text{lat}(\Gamma ; x ; T_2), \overline{\gamma}; \pi(x)|$ and $| \Xi' \cup \text{lat}(\Gamma ; y ; \pi)| \subseteq | \Xi|$, replacing $x$ for $v$ and adding the latent effects of $v$ in both sides, $| \Xi' \cup \text{lat}(\Gamma ; y ; T'_2 ; x ; T_2), \overline{\gamma}; \pi(x)| \subseteq | \Xi' \cup \text{lat}(\Gamma ; T'_1 ; x ; T_2), \overline{\gamma}; \pi(x)|$.

Therefore $| \Xi' \cup \text{lat}(\Gamma ; y ; T'_2 ; x ; T_2), \overline{\gamma}; \pi(x)| \subseteq | \Xi \cup \text{latent}_{\Gamma; \theta}(T_2)|$.

Then, we can use rule [IApp] to infer back that $\Xi \cup \text{latent}_{\Gamma; \theta}(T_2); \Gamma; \pi \setminus \{x\} \vdash [v/x] x \vdash [v/x] e_3' : T'_3$, and by transitivity of subtyping, $T''_3 < : T_3$.

Case (IAppP) with $f = x$). By Lemma 17 we can infer that $\Xi \cup \text{adjust}(\Xi) \cup \text{latent}_{\Gamma; \theta}(T_2) \subseteq \text{adjust}(\Xi \cup \text{latent}_{\Gamma; \theta}(T_2))$.

Thus we can use our induction hypothesis to both subexpressions of $e_3 = x e_2'$.

Therefore, while $\Gamma(x) = (y ; T'_1) \overset{\Xi'}{\rightarrow} T'_3$ and $\text{adjust}(\Xi) ; \Gamma; x \vdash e_2' ; T'_3$ with $T'_3 < : T'_1$ also $\text{adjust}(\Xi \cup \text{latent}_{\Gamma; \theta}(T_2)) ; \Gamma; \pi \setminus \{x\} \vdash [v/x] x = v : T'_1 \overset{\Xi'}{\rightarrow} T'_3$ and $\text{adjust}(\Xi) ; \Gamma; \pi \setminus \{x\} \vdash [v/x] e_2' : T'_2$ with $T'_2 \overset{\Xi'}{\rightarrow} T'_3 < : T'_1$.

We also know that the adjustment function is monotonically increasing, therefore $\text{adjust}(\Xi) \cup \text{latent}_{\Gamma; \theta}(T_2) \subseteq \text{adjust}(\Xi \cup \text{latent}_{\Gamma; \theta}(T_2))$.

By proposition 14 $\text{adjust}(\Xi \cup \text{latent}_{\Gamma; \theta}(T_2)) ; \Gamma; \pi \setminus \{x\} \vdash [v/x] e_2' : T'_2$.

Also, by definition $\text{latent}_{\Gamma; \theta}(T_2) = \Xi'' \cup \text{lat}(\Gamma , \overline{\gamma}, \pi(x))$. As $T''_2 < : T'_1$ then $\Xi'' \cup \text{lat}(\Gamma , y ; T'_2), \overline{\gamma}, \pi(x) \subseteq \Xi'' \cup \text{lat}(\Gamma , y ; T'_2), \overline{\gamma}, \pi(x)$.

Therefore $| \Xi'' \cup \text{lat}(\Gamma , y ; T'_2), \overline{\gamma}, \pi(x)| \subseteq | \Xi \cup \text{latent}_{\Gamma; \theta}(T_2)|$.

We therefore know that $T''_2 < : T'_1$ and we can use rule [IApp] to infer back that $\Xi \cup \text{latent}_{\Gamma; \theta}(T_2); \Gamma; \pi \setminus \{x\} \vdash [v/x] x \vdash [v/x] e_2' : T'_3$, and by transitivity of subtyping, $T''_3 < : T_3$.

While LPE, TGE and Marino and Millstein’s work are framed as generic effect frameworks, it turns out that LPE is less expressive than the other two. The system defined in LPE cannot represent the capability of the adjustment function to remove effect privileges for the evaluation of some sub-expressions. This is mainly because the mechanism that LPE uses to check effects are effect abstractions, but effect abstractions cannot express the requirement that an expression has “the inferred effects minus a certain effect”. Because this work is based on LPE, we need to restrict the adjustment function to be monotonically increasing. We leave the study of gradual polymorphic effects in a more general setting that does not impose monotonicity on adjustment for future work.
Lemma 20. \( \text{check}(\Xi) \Rightarrow \text{strict-check}(\Delta(\Xi) \cup \Xi) \)

i.e. If \( \text{check}(\Phi) \) for some \( \Phi \in \gamma(\Xi) \), then \( \text{check}(\Phi) \) for every \( \Phi \in \gamma(\Delta(\Xi) \cup \Xi) \).

Proof. Suppose \( \text{check}(\Phi) \) for some \( \Phi \in \gamma(\Xi) \)

Then \( \Upsilon = \{ \Phi \in \gamma(\Xi) \mid \text{check}(\Phi) \} \neq \emptyset \) so \( \Phi = \bigcup \text{mins}(\Upsilon) \) exists.

Furthermore, by monotonicity [3], \( \text{check}(\Phi) \).

Note that \( \Phi \subseteq \Phi \mid \Xi \mid \Xi = \Delta(\Xi) \cup \Xi \), so if \( \Phi_2 \in \gamma(\Delta(\Xi) \cup \Xi) \) then \( \Phi \subseteq \Phi_2 \) and by monotonicity [3], \( \text{check}(\Phi_2) \).

\( \square \)

Proposition 21. If \( \Xi; \Gamma; \pi \vdash e : T_1, e \notin \pi \) and \( T_1 \subseteq T_2 \) in the internal language, then \( \Xi; \Gamma; \pi \vdash \langle T_2 \Leftarrow T_1 \rangle^\pi e : T_2' \) and \( T_1' \subseteq T_2' \).

Proof. By Case analysis

Case (1) \( T_1 \subset T_2 \).

1. By assumption \( \Xi; \Gamma; \pi \vdash e : T_1 \).

2. \( \langle T_2 \Leftarrow T_1 \rangle^\pi e \equiv e \) by definition of metafunction.

3. \( \Xi; \Gamma; \pi \vdash \langle T_2 \Leftarrow T_1 \rangle^\pi e : T_1 \) by 1 and 2.

Case \( (x_1 : T_{11}) \frac{\Xi_1}{\pi_1} T_{12} \nvdash (x_2 : T_{21}) \frac{\Xi_2}{\pi_2} T_{22} \) and \( e \neq x \). Where \( T_1 = (x_1 : T_{11}) \frac{\Xi_1}{\pi_1}, T_{12} = (x_2 : T_{21}) \frac{\Xi_2}{\pi_2}, T_{22} \) and \( \Gamma_1 = (\Gamma, x_1 : T_{21}, x_2 : T_{11}, f : T_1) \).

1. \( \langle T_2 \Leftarrow T_1 \rangle^\pi e \equiv (\lambda f : T_1 \Rightarrow (T_2 \Leftarrow T_1)^\pi e : T_{2'} \cdots \Theta \circ f \cdots \circ e \).

2. \( \vdash \Gamma, f : T_1 \Rightarrow (T_2 \Leftarrow T_1)^\pi e : T_{2'}, \) where \( T_{2'} \subset T_2 \) by proposition 22.

3. \( \Xi; \Gamma; \pi \vdash (\lambda f : T_1 \Rightarrow (T_2 \Leftarrow T_1)^\pi e : T_{2'}, \) and \( T_{2'} \subset T_2 \) by [Ifun].

4. \( \Xi; \Gamma; \pi \vdash (\lambda f : T_1 \Rightarrow (T_2 \Leftarrow T_1)^\pi e : T_{2'}, \) and \( T_{2'} \subset T_2 \) by [IApp].

Case \( (x_1 : T_{11}) \frac{\Xi_1}{\pi_1} T_{12} \nvdash (x_2 : T_{21}) \frac{\Xi_2}{\pi_2} T_{22} \) and \( e = x \). Where \( T_1 = (x_1 : T_{11}) \frac{\Xi_1}{\pi_1}, T_{12} = (x_2 : T_{21}) \frac{\Xi_2}{\pi_2}, T_{22} \) and \( \Gamma_1 = (\Gamma, x_1 : T_{21}, x_2 : T_{11}) \).

1. \( \langle T_2 \Leftarrow T_1 \rangle^\pi e \equiv \langle T_2 \Leftarrow T_1 \rangle^\pi e \) by definition of metafunction.

2. \( \Xi; \Gamma; \pi \vdash \langle T_2 \Leftarrow T_1 \rangle^\pi e : T_{2'}, \) where \( T_{2'} \subset T_2 \) by proposition 22.

3. \( \Xi; \Gamma; \pi \vdash \langle T_2 \Leftarrow T_1 \rangle^\pi e : T_{2'}, \) by 1 and 2.

\( \square \)

Proposition 22. If \( \Xi; \Gamma; \pi \vdash f : (x_2 : T_{21}) \frac{\Xi_2}{\pi_2} T_{22}, x_1 \in \Gamma, x_2 \in \Gamma, T_{21} \subseteq T_{11}, \) and \( T_{12} \subseteq T_{22}, \) then

\( \Xi; \Gamma; \pi \vdash \langle x_2 : T_{21} \rangle \frac{\Xi_2}{\pi_2} T_{22} \Leftarrow (x_2 : T_{11}) \frac{\Xi_1}{\pi_1} T_{12}, f : (x_2 : T_{21}) \frac{\Xi_2}{\pi_2} T_{22}' \)

(depending on the cast function, \( T_{22}' = T_{22} \) or \( T_{22}' = T_{12} \)).

Proof. Let \( \Xi'_1 = \Xi_1 \cup \text{lat}(\Gamma_1, \pi_2, \emptyset) \) and \( \Xi'_2 = \Xi_2 \cup \text{lat}(\Gamma_1, \pi_2, \emptyset) \). Let \( \Gamma' = \Gamma_1 = \Gamma, x_2 : T_{21} \).
• We proceed by cases over \( c \).

**Case** \( c = \text{true} \), \( |\Xi_1\setminus\Xi_2| \neq \emptyset \).

**Prop. 2**

\[
T_{11}' \leq T_{11}
\]

**IAPRM 1 & 2**

\[
T_{11}' < T_{11}
\]

**IAPRMP**

\[
\Xi_1' \cup \Xi_2'; \Gamma'; \Psi_3 \vdash f\colon (x_2 : T_{11}) \xrightarrow{\Xi_1'} T_{12}
\]

**IVAR**

\[
\Gamma'(f) = (x_2 : T_{11}) \xrightarrow{\Xi_1} T_{12}
\]

**HAs**

\[
\Xi_2'; \Gamma'; \Psi_3 \vdash \text{has}(\Xi_1' \setminus \Xi_2) f\star((\{T_{11} \leftarrow T_{21}\})^2 \mathbb{E}_{\Psi_3} x_2); T_{12}
\]

**IFN**

\[
\Xi; \Gamma; \Pi \vdash (\lambda x_2 : T_{21} . \{T_{22} \leftarrow T_{12}\}) \xrightarrow{\Gamma'\text{reduce } \Xi_2} \text{has}(\Xi_1' \setminus \Xi_2) f\star((\{T_{11} \leftarrow T_{21}\})^2 \mathbb{E}_{\Psi_3} x_2)); T_{22}'
\]

**Case** \( c = \text{false} \). Trivial by using the same argument for \( c = \text{true} \), \( |\Xi_1\setminus\Xi_2| \neq \emptyset \).

**Prop. 2**

\[
T_{11}' \leq T_{11}
\]

**IAPRMP**

\[
\Xi_1' \cup \Xi_2'; \Gamma'; \Psi_3 \vdash f\colon (x_2 : T_{11}) \xrightarrow{\Xi_1'} T_{12}
\]

**IVAR**

\[
\Gamma'(f) = (x_2 : T_{11}) \xrightarrow{\Xi_1} T_{12}
\]

**HAs**

\[
\Xi_2'; \Gamma'; \Psi_3 \vdash \text{has}(\Xi_1' \setminus \Xi_2) f\star((\{T_{11} \leftarrow T_{21}\})^2 \mathbb{E}_{\Psi_3} x_2); T_{12}
\]

**IFN**

\[
\Xi; \Gamma; \Pi \vdash (\lambda x_2 : T_{21} . \{T_{22} \leftarrow T_{12}\}) \xrightarrow{\Gamma'\text{reduce } \Xi_2} \text{has}(\Xi_1' \setminus \Xi_2) f\star((\{T_{11} \leftarrow T_{21}\})^2 \mathbb{E}_{\Psi_3} x_2)); T_{22}'
\]

**References**


