Logical relations: safety of system F

Amin Timany

October 9, 2023*

1 Language

1.1 Syntax

$$\begin{array}{rll} \textit{variables}(\mathsf{Var}) & & x, y, z, \ldots \\ \textit{expressions}(\mathsf{E}) & e ::= & x \mid tt \mid (e, e) \mid \mathit{fst}\ e \mid \mathit{snd}\ e \mid \lambda x.\ e \mid e\ e \mid \Lambda\ e \mid e\ \_ \\ \textit{values}(\mathsf{Val}) & v ::= & tt \mid (v, v) \mid \lambda x.\ e \mid \Lambda\ e \end{array}$$

The set of values is a subset of the set of expressions: $\mathsf{Val} \subset \mathsf{E}$.

1.2 Types and typing

$$\begin{array}{rll} \textit{type variables}(\mathsf{TVar}) & & \alpha, \delta, \zeta, \ldots \\ \textit{types}(\mathsf{Typ}) & \tau ::= & \alpha \mid () \mid \tau \times \tau \mid \tau \to \tau \mid \forall \alpha.\ \tau \end{array}$$

The typing context that we consider is a pair of contexts consisting of Δ, the context for type variables, and Γ, the context for term variables. The "compound" typing context (typing context for short) that we use in defining the typing rules is a pair of context written as $\Delta; \Gamma$.

$$\begin{array}{rll} \textit{typing context}\ (\mathsf{TCtx}) & \Gamma ::= & \cdot \mid x : \tau, \Gamma \\ \textit{context of typing variables}\ (\mathsf{TCtx}) & \Delta ::= & \cdot \mid \alpha, \Delta \end{array}$$

Typing rules for our system F are:

$\boxed{\Delta; \Gamma \vdash e : \tau}$

T-Unit
$$\Delta; \Gamma \vdash tt : ()$$

T-Var
$$\frac{x : \tau \in \Gamma}{\Delta; \Gamma \vdash x : \tau}$$

T-Prod
$$\frac{\Delta; \Gamma \vdash e_1 : \tau_1 \qquad \Delta; \Gamma \vdash e_2 : \tau_2}{\Delta; \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

T-Fst
$$\frac{\Delta; \Gamma \vdash e : \tau_1 \times \tau_2}{\Delta; \Gamma \vdash \mathit{fst}\ e : \tau_1}$$

T-Snd
$$\frac{\Delta; \Gamma \vdash e : \tau_1 \times \tau_2}{\Delta; \Gamma \vdash \mathit{snd}\ e : \tau_2}$$

T-Lam
$$\frac{\Delta; x : \tau_1, \Gamma \vdash e : \tau_2}{\Delta; \Gamma \vdash \lambda x.\ e : \tau_1 \to \tau_2}$$

T-App
$$\frac{\Delta; \Gamma \vdash e : \tau_1 \to \tau_2 \qquad \Delta; \Gamma \vdash e' : \tau_1}{\Delta; \Gamma \vdash e\ e' : \tau_2}$$

*These notes were originally released on August 17, 2016. The present version includes significant improvements.

$$\text{T-TLAM}\ \frac{\alpha, \Delta; \Gamma \vdash e : \tau \qquad \alpha \text{ does not appear freely in } \Gamma}{\Delta; \Gamma \vdash \Lambda\ e : \forall \alpha.\ \tau} \qquad \text{T-TAPP}\ \frac{\Delta; \Gamma \vdash e : \forall \alpha.\ \tau}{\Delta; \Gamma \vdash e\ \_ : \tau[\tau'/\alpha]}$$

1.3 Operational semantics (CBV)

We describe the small-step call-by-value (CBV) operational semantics for system F. In the first step we give the head reduction relation ($\rightsquigarrow_h$). In the second step we extend this to non-head reductions using evaluation context (ECtx).

Head reduction:

$$\mathit{fst}\ (v_1, v_2) \rightsquigarrow_h v_1 \qquad \mathit{snd}\ (v_1, v_2) \rightsquigarrow_h v_2 \qquad (\lambda x.\ e)\ v \rightsquigarrow_h e[v/x] \qquad (\Lambda\ e)\ \_ \rightsquigarrow_h e$$

Note that here v's are values *and not any expression*. $e[v/x]$ is the expression e where all instances of x are replaced with v. *Remember that all substitutions are capture avoiding*.

Non-head reduction: If the redex (what is being reduced) is not in the head position (see above) then evaluation contexts determine where in the term a reduction can happen.

$$\frac{e \rightsquigarrow_h e'}{\mathbf{K}[e] \rightsquigarrow \mathbf{K}[e']}$$

where $\mathbf{K}$ is an evaluation context, i.e., $\mathbf{K} \in \mathsf{ECtx}$ and $\mathbf{K}[e]$ is the expression where the single whole in the context $\mathbf{K}$ (see below) is substituted with e.

Evaluation Contexts:

$$\textit{evaluation contexts}\ (\mathsf{ECtx}) \quad \mathbf{K} ::= [] \mid \mathit{fst}\ \mathbf{K} \mid \mathit{snd}\ \mathbf{K} \mid (\mathbf{K}, e) \mid (v, \mathbf{K}) \mid \mathbf{K}\ e \mid v\ \mathbf{K} \mid \mathbf{K}\ \_$$

Remark: In the sequel, we will use the word "context" to refer to both typing contexts and evaluation contexts whenever the distinction is clear from the discussion at hand.

Example: The following is the only possible reduction for the expression

$$\mathit{fst}\ ((\lambda x.\ ((\lambda y.\ tt)\ x, (\lambda y.\ x)\ tt))\ tt)$$

showing that it reduces to tt

$$\mathit{fst}\ ((\lambda x.\ ((\lambda y.\ tt)\ x, (\lambda y.\ x)\ tt))\ tt) \rightsquigarrow \mathit{fst}\ ((\lambda y.\ tt)\ tt, (\lambda y.\ tt)\ tt) \rightsquigarrow \mathit{fst}\ (tt, (\lambda y.\ tt)\ tt)$$

$$\rightsquigarrow \mathit{fst}\ (tt, tt) \rightsquigarrow tt$$

We use $\rightsquigarrow^*$ to denote reflexive and transitive closure of $\rightsquigarrow$.

2 Type safety

We say a language is type safe or has the type safety property if:

$$\forall e, \tau.\ \cdot; \cdot \vdash e : \tau \implies \mathbf{Safe}(e)$$

where

$$\mathbf{Safe}(e) \triangleq \forall e'.\ e \rightsquigarrow^* e' \implies e' \in \mathsf{Val} \vee \exists e''.\ e \rightsquigarrow e''$$

For system F, type safety can be proven using the well-known progress and preservation technique. In these notes we instead use the logical relations technique which also allows us to derive some "theorems for free".

3 Logical relations (First Attempt)

It seems only natural, given the statement of type safety, *i.e.* $e : \tau$ implies safety, to try to prove this statement by induction on the typing derivation. As we will see below, applying induction directly on the statement of type safety does not allow us to prove it.

What we do in this section is attempting to apply induction to type safety and each time this fails we change (stengthen) the statement of the theorem until we can successfully apply induction. In each step, we show that the statement that we consider implies type safety.

Attempt 1: If we simply apply induction on the the typing derivation to prove type safety as stated above, we immediately run into a problem in the case T-APP. In this case we have to prove $\mathbf{Safe}(e\ e')$ and the only thing that we know is that $\mathbf{Safe}(e)$ and $\mathbf{Safe}(e')$ both hold. Particularly, notice the case where $e = \lambda x.\ f$ for some expression f. Since e is already a value, its safety says absolutely nothing about f. Note that in this case, we can have $e' \rightsquigarrow^* v'$ which means that $e\ e' \rightsquigarrow^* (\lambda x.\ f)\ v' \rightsquigarrow f[v'/x]$. In this case, we do not have enough information to show that

$$f[v'/x] \in \mathsf{Val} \vee \exists g.\ f[v'/x] \rightsquigarrow g$$

This is the key observation motivating logical relations. It is not sufficient to know that expressions are safe. We need in addition to know that the resulting value has certain properties depending on its type. For instance, if e is an expression of function type $\tau_1 \to \tau_2$, then any value v obtained from e must be safe when applied to a value of type τ_1 and moreover, the resulting value must satisfy the appropriate properties of the type τ_2. The main idea of logical relations is simply just that. We define relations (in this case a *unary* relation, *i.e.*, a predicate) by induction on the types of the programming language. This predicate, in the case of function types (whose values are lambdas) specifically states that terms are in the predicate if they take terms in the predicate for the domain to terms in the predicate for the codomain.

Note: A similar issue is not observed in the case of T-TAPP. In that case, $(\Lambda\ e)\ \_ \rightsquigarrow e$ and the safety of e is simply the induction hypothesis.

4 Parameterized Safety

Above, we argued that logical relations for safety of our system would essentially state that an expression in the relation must be safe, and in addition, the resulting value must satisfy certain properties. In order to formalize this idea we introduce the following parameterized safety predicate (parameterized over a predicate on values) which we will later use in the definition of logical relations.

$$\mathbf{Safe}_P(e) \triangleq \forall e'.\ e \rightsquigarrow^* e' \implies (e' \in \mathsf{Val} \wedge P(e')) \vee \exists e''.\ e \rightsquigarrow e''$$

Note how the predicate $\mathbf{Safe}_P(e)$, regardless of the predicate P, implies $\mathbf{Safe}(e)$ as defined earlier. The parameterized safety predicate satisfies the following interesting properties:

Safe-Mono. *Let e be an expression and P and Q be two predicates on values, we have*

$$(\forall v.\ P(v) \implies Q(v)) \implies \boldsymbol{Safe}_P(e) \implies \boldsymbol{Safe}_Q(e)$$

Safe-Val. *Let v be a value and P be a predicate on values, we have*

$$P(v) \implies \boldsymbol{Safe}_P(v)$$

Safe-Bind. *Let $\mathbf{K}$ be an evaluation context, e be an expression, and P be a predicate on values. The following holds:*

$$\boldsymbol{Safe}_Q(e) \wedge (\forall v.\ Q(v) \implies \boldsymbol{Safe}_P(\mathbf{K}[v])) \implies \boldsymbol{Safe}_P(\mathbf{K}[e])$$

The property Safe-Bind states that to show that $\mathbf{K}[e]$ is safe, it suffices to show that $\mathbf{K}[v]$ is safe where v is the value obtained from executing e. This crucially relies on the fact that any execution of $\mathbf{K}[e]$ must necessarily first execute e, and only when it has evaluated to a value v continue with execution of $\mathbf{K}[v]$.

Safe-Step. *Let e and e' be two expressions and P be a predicate on values. We have the following:*

$$e \rightsquigarrow e' \wedge \boldsymbol{Safe}_P(e') \implies \boldsymbol{Safe}_P(e)$$

Note the order of things in Safe-Step above. Whenever $e \rightsquigarrow e'$, $\mathbf{Safe}_P(e')$ implies $\mathbf{Safe}_P(e)$. This means that for instance if we have to show $\mathbf{Safe}_P((\lambda x.\ e)\ v)$ it suffices to show $\mathbf{Safe}_P(e[v/x])$. The proof of this theorem essentially relies on the fact that the operational semantics of our programming language is deterministic. That is, when $e \rightsquigarrow e'$ we have that e' is the only possible expression that e evaluates to in one step.

5 Logical Relations

We define the predicates associated with types in two stages. We first define a predicate $P^{\mathbf{E}}$ on expressions given a predicate P on values which essentially the parameterized safety predicate we defined above — this is essentially a new, more concise, and more consistent notation for the parameterized safety relation.

$$P^{\mathbf{E}}(e) \triangleq \mathbf{Safe}_P(e)$$

This relation is essentially the safety predicate which in addition states then the resulting value must in addition also satisfy P. We then define a set of predicates on values defined recursively on types. Notice that since these predicates are defined by recursion on types and types can have free variables, we need to consider the interpretation of free type variables. Hence, the predicates the we define on values, and therefore also the predicates on all expressions, must be decorated with a mapping ξ from type variables to predicates. Intuitively, ξ maps each type variable to its associated predicate on values.

Predicates on values $\left(\llbracket \Delta \vdash \tau \rrbracket_\xi\right)$: Let us have a firs naïve attempt at defining these logical predicates.

$$\begin{aligned}
\llbracket \Delta \vdash \alpha \rrbracket_\xi &\triangleq \xi(\alpha) \\
\llbracket \Delta \vdash () \rrbracket_\xi\,(v) &\triangleq v = tt \\
\llbracket \Delta \vdash \tau_1 \times \tau_2 \rrbracket_\xi\,(v) &\triangleq \exists v_1, v_2.\ v = (v_1, v_2) \wedge \llbracket \Delta \vdash \tau_1 \rrbracket_\xi\,(v_1) \wedge \llbracket \Delta \vdash \tau_1 \rrbracket_\xi\,(v_2) \\
\llbracket \Delta \vdash \tau_1 \to \tau_2 \rrbracket_\xi\,(v) &\triangleq \exists x, e.\ v = \lambda x.\ e \wedge \forall v'.\ \llbracket \Delta \vdash \tau_1 \rrbracket_\xi\,(v') \implies \llbracket \Delta \vdash \tau_2 \rrbracket^{\mathbf{E}}_\xi\,(e[v/x]) \\
\llbracket \Delta \vdash \forall \alpha.\ \tau \rrbracket_\xi\,(x) &\triangleq \exists e.\ v = \Lambda\ e \wedge \forall \tau'.\ \llbracket \Delta \vdash \tau[\tau'/\alpha] \rrbracket^{\mathbf{E}}_\xi\,(e) \quad \textit{(incorrect)}
\end{aligned}$$

where $[\alpha \mapsto P]\xi$ is extending the mapping ξ by additionally associating α to P.

The problem with the case $\llbracket \Delta \vdash \forall \alpha.\ \tau \rrbracket_\xi$ is that in this case the type $\tau[\tau'/\alpha]$ is not *smaller* than the type $\forall \alpha.\ \tau$ which makes the inductive definition above invalid. We can for example take τ' to be $\forall \alpha.\ \tau$ itself. The unrestricted choice of τ', *i.e.*, the universal quantification, is the essence of the problem. Yet, it is necessary as the typing rule T-TAPP allows us to instantiate a polymorphic type with any arbitrary type τ'.

There are two ways that we can address this issue. The first approach is to restrict the range of τ' in the rule T-TAPP. We can do this by introducing an infinite hierarchy of so-called *universes*:

$$\mathsf{Tp}_0, \mathsf{Tp}_1, \ldots$$

We would then restrict that Tp_0 has no types that include polymorphic types (types of the form $\forall \alpha.\ \tau$). Furthermore, for any type $\forall \alpha.\ \tau$ that belongs to Tp_{n+1}, only a τ' in Tp_i for $i \leq n$ can be used to substitute α in the typing rules. This allows us define the logical relations of system F by recursion on the hierarchy of universes as well as the structure of types. This would allow us to prove type safety (and when the logical predicates are adjusted appropriately) to prove safety for system F — though it would not allow us to derive "theorems for free"!. This way of treating polymorphic types is usually referred to as *predicative* polymorphism, as opposed to the *impredicative* polymorphism of system F. This is the approach taken in the underlying languages of proof assistants such as Coq and Agda.

Here, we adjust our logical relations to work with the impredicative polymorphism system F. This is based on an insight by J. C. Reynolds, who alongside J. Y. Girard (independently) discovered system F. This insight is that the semantics (predicate) of polymorphic types should not quantifying over all of the types of the language but rather they must quantifying over all possible semantic types, *i.e.*, all possible predicates over values. Note that this also includes all the predicates that are associated to the types of our language. The new logical relations for system F is defined as follows:

$$\begin{aligned}
[\![\Delta \vdash \alpha]\!]_\xi &\triangleq \xi(\alpha) \\
[\![\Delta \vdash ()]\!]_\xi\,(v) &\triangleq v = tt \\
[\![\Delta \vdash \tau_1 \times \tau_2]\!]_\xi\,(v) &\triangleq \exists v_1, v_2.\ v = (v_1, v_2) \wedge [\![\Delta \vdash \tau_1]\!]_\xi\,(v_1) \wedge [\![\Delta \vdash \tau_1]\!]_\xi\,(v_2) \\
[\![\Delta \vdash \tau_1 \to \tau_2]\!]_\xi\,(v) &\triangleq \exists x, e.\ v = \lambda x.\ e \wedge \forall v'.\ [\![\Delta \vdash \tau_1]\!]_\xi\,(v') \implies [\![\Delta \vdash \tau_2]\!]^{\mathbf{E}}_\xi\,(e[v'/x]) \\
[\![\Delta \vdash \forall \alpha.\ \tau]\!]_\xi\,(v) &\triangleq \exists e.\ v = \Lambda\ e \wedge \forall P \in 2^{\mathsf{Val}}.\ [\![\alpha, \Delta \vdash \tau]\!]^{\mathbf{E}}_{[\alpha \mapsto P]\xi}\,(e)
\end{aligned}$$

Note: Note that we are implicitly assuming that the domain of the partial mapping ξ in $[\![\Delta \vdash \tau]\!]_\xi$ is always Δ.

Before we proceed, we state two important properties of our logical relations.

The following lemma states that as far as the logical relation is concerned substitution of a type τ' in a type τ is equivalent to associating its corresponding logical relation to the corresponding free type variable in the mapping that interprets free type variables.

LogRel-Subst. *The following holds for any ξ, Δ, v, τ, τ', α:*

$$[\![\Delta \vdash \tau[\tau'/\alpha]]\!]_\xi\,(v) \iff [\![\alpha, \Delta \vdash \tau]\!]_{[\alpha \mapsto [\![\Delta \vdash \tau']\!]_\xi]\xi}\,(v)$$

Proof. Straightforward induction on structure of type τ. □

The following lemma states that, perhaps obviously, the relation for a type τ where α does not appear freely does not depend on the predicate associated with α in the mapping that interprets free type variables.

LogRel-Weaken. *The following holds for any P ξ, Δ, v, τ, and any α the **does not** appear freely in τ:*

$$[\![\Delta \vdash \tau]\!]_\xi\,(v) \iff [\![\alpha, \Delta \vdash \tau]\!]_{[\alpha \mapsto P]\xi}\,(v)$$

Proof. Straightforward induction on structure of type τ. □

The logical relations above are defined over closed terms, *i.e.*, programs where no free (term) variables appear. We extend the defined relations to open terms by substituting for the free variables values in the logical relations for the appropriate type.[1] For this purpose we define when a sequence of values vs is in the corresponding predicates for the types in a context Γ, written as $[\![\Delta \vdash \Gamma]\!]_\xi\,(vs)$.

The relation on sequences of values A sequence of terms $vs = v_1, \ldots, v_n$ is said to be in the predicates for a context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ under the context of type variables Δ (with interpretation ξ), written as $[\![\Delta \vdash \Gamma]\!]_\xi\,(vs)$ if

$$\begin{aligned}
[\![\Delta \vdash \cdot]\!]_\xi\,(vs) &\triangleq |vs| = 0 \\
[\![\Delta \vdash x : \tau, \Gamma]\!]_\xi\,(vs) &\triangleq \exists v, vs'.\ vs = v, vs' \wedge [\![\Delta \vdash \tau]\!]_\xi\,(v) \wedge [\![\Delta \vdash \Gamma]\!]_\xi\,(vs')
\end{aligned}$$

[1] It is an interesting exercise to work out what goes wrong if we do not extend the relation to open terms. That is, if we only have a relation on closed terms. After all, the type safety theorem regards only closed terms. *Hint: check the induction hypothesis we obtain for the case of* T-LAM.

The following lemma simply extends lemma LogRel-Weaken to the relation on sequences of values.

LogRel-Seq-Weaken. *The following holds for any P ξ, Δ, vs, Γ, and any α the* ***does not*** *appear freely in Γ:*

$$[\![\Delta \vdash \Gamma]\!]_{\xi}\,(vs) \iff [\![\alpha, \Delta \vdash \Gamma]\!]_{[\alpha \mapsto P]\xi}\,(vs)$$

Proof. A simple consequence of LogRel-Weaken which follows by a simple induction on the length of Γ □

Attempt 2: In this final attempt we prove the following theorem, known as the fundamental theorem (or fundamental lemma) of logical relations.

Fundamental Theorem (of logical relations)**.** *For any e, Δ, Γ and τ such that $\Delta; \Gamma \vdash e : \tau$ we have:*

$$\forall \xi, vs.\ [\![\Delta \vdash \Gamma]\!]_{\xi}\,(vs) \implies [\![\Delta \vdash \tau]\!]^{\mathbf{E}}_{\xi}\,(e[vs/xs])$$

where xs is the sequence of variables of Γ and $e[vs/xs]$ is a shorthand for $e[v_1, \ldots, v_n/x_1, \ldots, x_n]$ which is the term e where x_i's are substituted with v_i's simultaneously.

Proof. By induction on the derivation of $\Delta; \Gamma \vdash e : \tau$. Here we prove the cases APP, TAPP, LAM and TLAM. We leave the rest of the cases as an easy exercise.

Case T-APP**:** Let us consider the case of T-APP which was problematic in the first attempt. We need to show that

$$\forall \xi, vs.\ [\![\Delta \vdash \Gamma]\!]_{\xi}\,(vs) \implies [\![\Delta \vdash \tau_2]\!]^{\mathbf{E}}_{\xi}\,(e[vs/xs]\ e'[vs/xs]) \tag{1}$$

holds knowing that both induction hypotheses

$$\forall \xi, vs.\ [\![\Delta \vdash \Gamma]\!]_{\xi}\,(vs) \implies [\![\Delta \vdash \tau_1 \to \tau_2]\!]^{\mathbf{E}}_{\xi}\,(e[vs/xs]) \tag{2}$$

and

$$\forall \xi, vs.\ [\![\Delta \vdash \Gamma]\!]_{\xi}\,(vs) \implies [\![\Delta \vdash \tau_1]\!]^{\mathbf{E}}_{\xi}\,(e'[vs/xs]) \tag{3}$$

hold. For the rest of this case, we assume some arbitrary but fixed ξ and vs. We need to show that

$$\mathbf{Safe}_{[\![\Delta \vdash \tau_2]\!]_{\xi}}(e[vs/xs]\ e'[vs/xs]) \tag{4}$$

Since $[]\ e'[vs/xs]$ is an evaluation context, we can apply Safe-Bind after which we need to show that

$$\mathbf{Safe}_{[\![\Delta \vdash \tau_2]\!]_{\xi}}(v\ e'[vs/xs]) \tag{5}$$

for a value v for which we have $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_{\xi}\,(v)$ (due to (2) above). The latter implies that $v = \lambda x.\ f$ for some expression f for which we have

$$\forall w.\ [\![\Delta \vdash \tau_1]\!]_{\xi}\,(w) \implies [\![\Delta \vdash \tau_2]\!]^{\mathbf{E}}_{\xi}\,(f[w/x]) \tag{6}$$

Now, to show that (5) holds, we can apply the Safe-Bind again, this time with the evaluation context $v\ []$. This leaves us to show

$$\mathbf{Safe}_{[\![\Delta \vdash \tau_2]\!]_\xi}((\lambda x.\ f)\ w) \tag{7}$$

for a value w for which we have

$$[\![\Delta \vdash \tau_1]\!]_\xi\ (w) \tag{8}$$

(due to (3) above). To show (7) above, by Safe-Step, it suffices to show

$$\mathbf{Safe}_{[\![\Delta \vdash \tau_2]\!]_\xi}(f[w/x]) \tag{9}$$

Which just follows by (6) as we have (8).

Case T-LAM**:** We need to show that

$$\forall \xi, vs.\ [\![\Delta \vdash \Gamma]\!]_\xi\ (vs) \implies [\![\Delta \vdash \tau_1 \to \tau_2]\!]^{\mathbf{E}}_\xi\ (\lambda x.\ e[vs/xs]) \tag{10}$$

holds knowing that the following induction hypothesis holds:

$$\forall \xi, vs.\ [\![\Delta \vdash x : \tau_1, \Gamma]\!]_\xi\ (vs) \implies [\![\Delta \vdash \tau_2]\!]^{\mathbf{E}}_\xi\ (e[vs/xs]) \tag{11}$$

That is, we need to show

$$\mathbf{Safe}_{[\![\Delta \vdash \tau_1 \to \tau_2]\!]_\xi}(\lambda x.\ e[vs/xs]) \tag{12}$$

For a sequence of values vs for which we have

$$[\![\Delta \vdash \Gamma]\!]_\xi\ (vs) \tag{13}$$

Since the term $\lambda x.\ e[vs/xs]$ is already a value, by Safe-Val, we only need to show

$$[\![\Delta \vdash \tau_1 \to \tau_2]\!]_\xi\ (\lambda x.\ e[vs/xs]) \tag{14}$$

After unfolding the logical relations, and some minor simplifications, we need to show that

$$\forall v'.\ [\![\Delta \vdash \tau_1]\!]_\xi\ (v') \implies [\![\Delta \vdash \tau_2]\!]^{\mathbf{E}}_\xi\ (e[vs/xs][v'/x]) \tag{15}$$

We proceed by introducing the universal quantifier and the implication in (15) above. Therefore, it remains to show

$$[\![\Delta \vdash \tau_2]\!]^{\mathbf{E}}_\xi\ (e[vs/xs][v/x]) \tag{16}$$

for some value v' for which we have

$$[\![\Delta \vdash \tau_1]\!]_\xi\ (v') \tag{17}$$

Now, putting (17) and (13) together we obtain

$$[\![\Delta \vdash x : \tau_1, \Gamma]\!]_\xi\ (v, vs) \tag{18}$$

Subsequently, by applying (11) to (18) we obtain

$$[\![\Delta \vdash \tau_2]\!]^{\mathbf{E}}_\xi\ (e[v', vs/x, xs]) \tag{19}$$

Which is the same as (16) above we had to prove.

Case T-TApp**:** We need to show that

$$\forall \xi, vs.\ [\![\Delta \vdash \Gamma]\!]_\xi\,(vs) \implies [\![\Delta \vdash \tau[\tau'/\alpha]]\!]^{\mathbf{E}}_\xi\,(e[vs/xs]\ \_) \tag{20}$$

holds knowing that the following induction hypothesis holds:

$$\forall \xi, vs.\ [\![\Delta \vdash \Gamma]\!]_\xi\,(vs) \implies [\![\Delta \vdash \forall \alpha.\ \tau]\!]^{\mathbf{E}}_\xi\,(e[vs/xs]) \tag{21}$$

That is, we need to show

$$\mathbf{Safe}_{[\![\Delta \vdash \tau[\tau'/\alpha]]\!]_\xi}(e[vs/xs]\ \_) \tag{22}$$

For a sequence of values vs for which we have

$$[\![\Delta \vdash \Gamma]\!]_\xi\,(vs) \tag{23}$$

Since $\mathbf{K}\ \_$ is an evaluation context we apply Safe-Bind with $\mathbf{K}$ being this evaluation context. Hence, we have to show

$$\mathbf{Safe}_{[\![\Delta \vdash \tau[\tau'/\alpha]]\!]_\xi}(v\ \_) \tag{24}$$

for some value v for which we know $v = \Lambda\ f$ such that

$$\forall P \in 2^{\mathsf{Val}}.\ [\![\alpha, \Delta \vdash \tau]\!]^{\mathbf{E}}_{[\alpha \mapsto P]\xi}\,(f) \tag{25}$$

Now, we apply Safe-Step to (24) which leaves us to show

$$\mathbf{Safe}_{[\![\Delta \vdash \tau[\tau'/\alpha]]\!]_\xi}(f) \tag{26}$$

This, however, due to the equivalence LogRel-Subst and Safe-Mono, is a simple consequence of (25) by taking P to be $[\![\Delta \vdash \tau']\!]_\xi$.

Case T-TLam**:** We need to show that

$$\forall \xi, vs.\ [\![\Delta \vdash \Gamma]\!]_\xi\,(vs) \implies [\![\Delta \vdash \forall \alpha.\ \tau]\!]^{\mathbf{E}}_\xi\,(\Lambda\ e[vs/xs]) \tag{27}$$

holds knowing that the following induction hypothesis holds:

$$\forall \xi, vs.\ [\![\alpha, \Delta \vdash \Gamma]\!]_\xi\,(vs) \implies [\![\alpha, \Delta \vdash \tau]\!]^{\mathbf{E}}_\xi\,(e[vs/xs]) \tag{28}$$

Furthermore, we know that α does not appear freely in Γ. Since $\Lambda\ e[vs/xs]$ is already a value, by Safe-Val, we only need to show

$$[\![\Delta \vdash \forall \alpha.\ \tau]\!]_\xi\,(\Lambda\ e[vs/xs]) \tag{29}$$

For some a sequence of values vs for which we have

$$[\![\Delta \vdash \Gamma]\!]_\xi\,(vs) \tag{30}$$

After unfolding the logical relations, and some minor simplifications, we need to show that

$$[\![\alpha, \Delta \vdash \tau]\!]^{\mathbf{E}}_{[\alpha \mapsto P]\xi}\,(e[vs/xs]) \tag{31}$$

for an arbitrary predicate $P \in 2^{\mathsf{Val}}$ on values. Now, by (28), we only need to show

$$[\![\alpha, \Delta \vdash \Gamma]\!]_{[\alpha \mapsto P]\xi}\,(vs) \tag{32}$$

which by LogRel-Seq-Weaken is exactly equivalent to (30) above. □

Type Safety. *For any expression e and type τ for which $\cdot;\cdot \vdash e : \tau$ we have*

$$\textbf{\textit{Safe}}(e)$$

Proof. By the Fundamental Theorem, since $\cdot;\cdot \vdash e : \tau$ we have

$$\forall \xi, vs.\ [\![\cdot \vdash \cdot]\!]_\xi\,(vs) \implies [\![\cdot \vdash \tau]\!]^{\mathbf{E}}_\xi\,(e[vs/xs])$$

We simply take ξ to by the empty map and vs to be the empty sequence of values. Consequently, we obtain

$$[\![\cdot \vdash \tau]\!]^{\mathbf{E}}_\emptyset\,(e)$$

or, written differently

$$\mathbf{Safe}_{[\![\cdot \vdash \tau]\!]_\emptyset}(e)$$

which immediately implies $\mathbf{Safe}(e)$ as required. □

6 Parametericity and theorems for free

The Fundamental Theorem that we proved above has a number of interesting consequences other than Type Safety. In particular, it says that the types that we quantify over in our language when we write polymorphic programs are not just the types that are syntactically representable in the type system of the language but are rather any conceivable semantic type, *i.e.*, any predicate over values. This ingenious idea of Reynolds has deep and very interesting consequences. Here, we only see a couple of very simple consequences of this idea which we can obtain even with our rather weak logical relations which merely establishes safety of system F.

One in particular can use this theorem to derive so-called *free theorems* about polymorphic programs. This is due to the fact that they follow rather easily from the Fundamental Theorem.

Free Theorem 1 (the polymorphic type of the identity function)**.** *Any term of type $\forall\alpha.\ \alpha \to \alpha$ is the identity function. That is,*

$$\forall e.\ \cdot;\cdot \vdash e : \forall\alpha.\ \alpha \to \alpha \implies \textbf{\textit{Safe}}_{Eq_v}((e\ \_)\ v)$$

where

$$Eq_v(w) \triangleq w = v$$

Proof. Let us assume $\cdot;\cdot \vdash e : \forall\alpha.\ \alpha \to \alpha$ for some arbitrary expression e. We need to show

$$\mathbf{Safe}_{Eq_v}((e\ \_)\ v) \tag{33}$$

By the Fundamental Theorem we know that

$$\forall \xi.\ [\![\cdot \vdash \forall\alpha.\ \alpha \to \alpha]\!]^{\mathbf{E}}_\xi\,(e) \tag{34}$$

By taking the empty mapping for ξ and unfolding the logical relation 34 implies

$$\mathbf{Safe}_{[\![\cdot \vdash \forall\alpha.\ \alpha \to \alpha]\!]_\emptyset}(e) \tag{35}$$

By using (35) above and Safe-Bind (since $([]\ \_)\ v$ is an evaluation context) it suffices to show

$$\mathbf{Safe}_{Eq_v}((w\ \_)\ v) \tag{36}$$

for some value $w = \Lambda\ f$ and some f for which we have

$$\forall P \in 2^{\mathsf{Val}}.\ [\![\alpha, \cdot \vdash \alpha \to \alpha]\!]^{\mathbf{E}}_{[\alpha \mapsto P]\emptyset}(f) \tag{37}$$

which can be simplified to

$$\forall P \in 2^{\mathsf{Val}}.\ \mathbf{Safe}_{[\![\alpha, \cdot \vdash \alpha \to \alpha]\!]_{[\alpha \mapsto pred]\emptyset}}(f) \tag{38}$$

Now, by Safe-Step we are left to show that

$$\mathbf{Safe}_{Eq_v}(f\ v) \tag{39}$$

Again, by applying Safe-Bind, this time together with (38) and taking P to be Eq_v, we only have to show

$$\mathbf{Safe}_{Eq_v}(w\ v) \tag{40}$$

for some value w and g such that $w = \lambda x.\ g$ and we have

$$[\![\alpha, \cdot \vdash \alpha \to \alpha]\!]_{[\alpha \mapsto Eq_v]\emptyset}(w) \tag{41}$$

or equivalently

$$\forall u.\ [\![\alpha, \cdot \vdash \alpha]\!]_{[\alpha \mapsto Eq_v]\emptyset}(u) \implies [\![\alpha, \cdot \vdash \alpha]\!]^{\mathbf{E}}_{[\alpha \mapsto Eq_v]\emptyset}(g[u/x]) \tag{42}$$

which we can simplify further

$$\forall u.\ Eq_v(u) \implies Eq^{\mathbf{E}}_v(g[u/x]) \tag{43}$$

and still further

$$\forall u.\ Eq_v(u) \implies \mathbf{Safe}_{Eq_v}(g[u/x]) \tag{44}$$

Now, by Safe-Step and the fact $w = \lambda x.\ g$, the following suffices to show (45):

$$\mathbf{Safe}_{Eq_v}(g[u/x]) \tag{45}$$

which is a simple consequence of (44). □

Free Theorem 2 (the empty type)**.** *Any term of type* $\forall\alpha.\ \alpha$ *must necessarily diverge. That is,*

$$\forall e, \cdot; \cdot \vdash e : \forall\alpha.\ \alpha \implies \boldsymbol{Safe}_{\emptyset}(e\ \_)$$

Proof. Let us assume $\cdot; \cdot \vdash e : \forall\alpha.\ \alpha$ for some arbitrary expression e. We need to show

$$\mathbf{Safe}_{\emptyset}(e\ \_) \tag{46}$$

By the Fundamental Theorem we know that

$$\forall \xi.\ [\![\cdot \vdash \forall \alpha.\ \alpha]\!]^{\mathbf{E}}_{\xi}(e) \tag{47}$$

By taking the empty mapping for ξ and unfolding the logical relation 47 implies

$$\mathbf{Safe}_{[\![\cdot \vdash \forall \alpha.\ \alpha]\!]_{\emptyset}}(e) \tag{48}$$

By using (48) above and Safe-Bind (since $[]\ \_$ is an evaluation context) it suffices to show

$$\mathbf{Safe}_{\emptyset}(w\ \_) \tag{49}$$

for some value w and f such that $w = \Lambda\ f$ and we have

$$\forall P \in 2^{\mathsf{Val}}.\ [\![\alpha, \cdot \vdash \alpha]\!]^{\mathbf{E}}_{[\alpha \mapsto P]\emptyset}(f) \tag{50}$$

which can be simplified to

$$\forall P \in 2^{\mathsf{Val}}.\ \mathbf{Safe}_{[\![\alpha, \cdot \vdash \alpha]\!]_{[\alpha \mapsto P]\emptyset}}(f) \tag{51}$$

and still further simplified to

$$\forall P \in 2^{\mathsf{Val}}.\ \mathbf{Safe}_{P}(f) \tag{52}$$

Now, by Safe-Step we are left to show that

$$\mathbf{Safe}_{\emptyset}(f) \tag{53}$$

which is a simple consequence of (52) when we take P to be $\emptyset$. □

7 Normalization of system F

We can define the normalization property as follows:

$$\mathbf{Normalizes}(e) \triangleq \exists v.\ e \rightsquigarrow^* v$$

Corresponding to normalization, we can define parameterized normalization predicate as follows:

$$\mathbf{Normalizes}_P(e) \triangleq \exists v.\ e \rightsquigarrow^* v \wedge P(v)$$

It is not too difficult to show counterparts of Safe-Mono, Safe-Val, Safe-Bind, and Safe-Step for the parameterized normalization predicate. Hence, we can set up logical relations analogous to those for type safety above where we replace $\mathbf{Safe}_P$ by $\mathbf{Normalizes}_P$. In fact, the proof of the fundamental theorem for these logical relations will also be entirely analogous to our proof of Fundamental Theorem above. As a result, similarly to the Type Safety theorem above, we obtain a normalization theorem which would state that any closed well-typed program normalizes to a value.