Supporting Unrestricted Recursive Types

Zhaozhong Ni

Microsoft Research
One Microsoft Way, Redmond, WA 98052, U.S.A.
zhaozhong.ni@microsoft.com

Abstract. Recursive types capture important invariants in programs and proofs. It is well-known that the naïve treatment of unrestricted recursive types causes inconsistency. Typically, recursive types are admitted by putting various restrictions on their formations. This puts limitation and complexity on the usages of recursive types. In this paper we present a general approach to support unrestricted recursive types. Instead of putting restrictions on formation (introduction), we design the elimination rules to control the (dangerous) unfolding (elimination) of recursive types, while maintaining the intuition behind the naïve understanding. Thus the usage of recursive types in our solution is simple and intuitive. Using System F as an example, we demonstrate how to safely admit recursive types and prove strong normalization for the new system. Our method is applicable to similar type systems and logics.

1 Introduction

Recursive types (e.g., $\mu\alpha.\,\tau$) are self-referential formulas that can be used to capture complex invariants found in programs and proofs, such as objects, symmetry threading, weak updates, etc.

Intuitively, the most straight-forward treatment of recursive types should be achieved via the following two rules (assuming Δ and Γ are the type and term environments and $\tau[\tau'/\alpha]$ stands for substituting all α by τ' in τ).

$$\frac{\Delta;\Gamma \vdash e:\tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma \vdash \mathsf{fold}\ e:\mu\alpha.\,\tau}\ (\textsc{fold}) \qquad \frac{\Delta;\Gamma \vdash e:\mu\alpha.\,\tau}{\Delta;\Gamma \vdash \mathsf{unfold}\ e:\tau[\mu\alpha.\,\tau/\alpha]}\ (\textsc{unfold-naive})$$

Unfortunately, it is a well-known fact that such kind of naïve treatment of recursive types will result in inconsistency of systems with the presence of $\to$ constructor—one can easily construct a term and a typing derivation for every type through the following derivation. (We omit the type environment and term and write $\Delta;\Gamma \vdash e:\tau$ as $\Gamma \vdash \tau$.)

$$\dfrac{\dfrac{\dfrac{\Gamma, \mu\alpha.\,\alpha\to\tau \vdash \mu\alpha.\,\alpha\to\tau}{\Gamma, \mu\alpha.\,\alpha\to\tau \vdash (\mu\alpha.\,\alpha\to\tau)\to\tau} \quad \Gamma, \mu\alpha.\,\alpha\to\tau \vdash \mu\alpha.\,\alpha\to\tau}{\Gamma, \mu\alpha.\,\alpha\to\tau \vdash \tau} \quad \dfrac{\text{(same as left)}}{\Gamma \vdash (\mu\alpha.\,\alpha\to\tau)\to\tau}}{\Gamma \vdash \tau} \quad \dfrac{}{\Gamma \vdash \mu\alpha.\,\alpha\to\tau}$$

Apparently, breaking some of the inference steps in the above derivation might allow one to avoid breaking the consistency of the system.

Previously, to support recursive types and ensure the consistency, it is typical to put various restrictions on the formation of recursive types—only "good" recursive types get introduced in the system. Inductive definitions, for example, requires syntactic checks on the "positions" (occurrences) of recursive variables in recursive formulae, and reject those in the "negative" positions [1, 2]. Nakano [3] requires a "properness" check in the formation of recursive types that are supported by an approximation-based semantic model. In the modal model of types work, Appel et al. [4] also require a "contractiveness" check, which is first used in MacQueen [5], to construct their approximation-based semantic model.

What type can and can not appear in these systems are no longer just determined by the BNF of types. Instead, meta-logical checks on the syntax of recursive types or semantics of their typing need to be done. This causes several problems for the usage of these solutions. The first problem is that some of these restrictions on recursive types being overly conservative, thus prohibiting the specification and reasoning of some "good" ones. The second problem is that sometimes the "goodness" of a recursive type might depends on its usage, thus can not be fully determined upon formation time. The third problem is that, in practice, especially in program verification, the recursive types formulae in program specifications are typically indeed "good" (reasoning about them will not causes inconsistency). This because the meta logic used for informal reasoning of the programs are presumably consistent. Requiring additional meta-level check of the "goodness" of these program specifications is unnecessary.

It will be ideal if one can specify recursive types freely based on the BNF of types and do "good" reasoning with them. Recently, Hawblitzel et al. [6] support such kind of unrestricted recursive types in their GTAL language by utilizing a special "modal" operator $\circ$ and prohibiting reduction inside the fold constructor. The recursive type there is used to construct the invariant for supporting weak updates of a TAL. However, a recursive type $\mu\alpha.\,\tau$ in GTAL unfolds to $\circ(\tau[\mu\alpha.\,\tau/\alpha])$, which is different from the naïve expectation of $\tau[\mu\alpha.\,\tau/\alpha]$. Also, Church-Rosser property does not hold for GTAL.

In this paper we present a general approach to support unrestricted recursive types. Instead of putting restrictions on formation (introduction rules), we design the elimination rules to control the (dangerous) unfolding (elimination rules) of recursive types, while maintaining the intuition behind the naïve understanding. Thus the usage of recursive types in our solution is simple and intuitive. Using System F [7] as an example, we demonstrate how to safely admit unrestricted recursive types and show the complete proof of the strong normalization property of F_μ, our new system. Our method is applicable to other type systems and logics.

The paper is organized as follows. We first recap the baseline System F in Sect. 2. We then define our new F_μ system with unrestricted recursive types in Sect. 3. In Sect. 4 and Sect. 5 we explain in details how to prove the strong normalization property of F_μ in two steps. We further discuss the usage of F_μ in Sect. 6. Finally, we compare with related work and conclude in Sect. 7.

$$
\begin{array}{rl}
(\mathit{Types}) & \tau ::= \alpha \mid \tau \to \tau' \mid \Pi\alpha.\,\tau \\
(\mathit{Terms}) & e ::= x \mid \lambda x{:}\tau.\,e \mid e\ e' \mid \Lambda\alpha.\,e \mid e\,[\tau] \\
(\mathit{Values}) & v ::= \lambda x{:}\tau.\,e \mid \Lambda\alpha.\,e \\
(\mathit{Type\ Env.}) & \Delta ::= \cdot \mid \Delta, \alpha \\
(\mathit{Term\ Env.}) & \Gamma ::= \cdot \mid \Gamma, x{:}\tau
\end{array}
$$

Fig. 1. Syntax of System F

$$
(\lambda x{:}\tau.\,e)\ e' \hookrightarrow e[e'/x] \quad \text{(R-APP)} \qquad (\Lambda\alpha.\,e)\,[\tau] \hookrightarrow e[\tau/\alpha] \quad \text{(R-TAPP)}
$$

$$
\frac{e \hookrightarrow e'}{\lambda x{:}\tau.\,e \hookrightarrow \lambda x{:}\tau.\,e'}\ \text{(R-LAM)} \qquad \frac{e \hookrightarrow e'}{e\ e'' \hookrightarrow e'\ e''}\ \text{(R-APP1)}
$$

$$
\frac{e \hookrightarrow e'}{v\ e \hookrightarrow v\ e'}\ \text{(R-APP2)} \qquad \frac{e \hookrightarrow e'}{\Lambda\alpha.\,e \hookrightarrow \Lambda\alpha.\,e'}\ \text{(R-TLAM)} \qquad \frac{e \hookrightarrow e'}{e\,[\tau] \hookrightarrow e'\,[\tau]}\ \text{(R-TAPP1)}
$$

Fig. 2. Reduction rules of System F

$$
\frac{\mathrm{FTV}(\tau) \subseteq \Delta}{\Delta \vdash \tau}\ \text{(TYPE)} \qquad \frac{\Delta \vdash \Gamma(x)}{\Delta;\Gamma \vdash x{:}\Gamma(x)}\ \text{(VAR)}
$$

$$
\frac{\Delta;\Gamma, x{:}\tau \vdash e{:}\tau'}{\Delta;\Gamma \vdash \lambda x{:}\tau.\,e{:}\tau \to \tau'}\ \text{(LAM)} \qquad \frac{\Delta;\Gamma \vdash e{:}\tau \to \tau' \qquad \Delta;\Gamma \vdash e'{:}\tau}{\Delta;\Gamma \vdash e\ e'{:}\tau'}\ \text{(APP)}
$$

$$
\frac{\Delta,\alpha;\Gamma \vdash e{:}\tau}{\Delta;\Gamma \vdash \Lambda\alpha.\,e{:}\Pi\alpha.\,\tau}\ \text{(TLAM)} \qquad \frac{\Delta;\Gamma \vdash e{:}\Pi\alpha.\,\tau \qquad \Delta \vdash \tau'}{\Delta;\Gamma \vdash e\,[\tau']{:}\tau[\tau'/\alpha]}\ \text{(TAPP)}
$$

Fig. 3. Static semantics of System F

2 System F

We briefly recap the definition and properties of System F [7]. We first present the syntax of System F in Fig. 1.

The reduction steps of System F is presented in Fig. 2. Substitutions $\tau[\tau'/\alpha]$, $e[\tau/\alpha]$, and $e[e'/x]$ are defined in their usually way.

We present the static semantics of System F in Fig. 3.

The following properties are well-known to hold true for System F. Interested reader can refer to Girard [7] for proofs.

Theorem 1 (System F Subject Reduction).
If $\Gamma;\Delta \vdash e{:}\tau$ *and* $e \hookrightarrow e'$ *then* $\Delta;\Gamma \vdash e'{:}\tau$.

Theorem 2 (System F Progress).
If $\cdot;\cdot \vdash e{:}\tau$ *then either* $e = v$ *or* $e \hookrightarrow e'$.

Theorem 3 (System F Strong Normalization).
If $\cdot;\cdot \vdash e{:}\tau$ *then the length of any reduction sequence starting with* e *is finite.*

$$
\begin{array}{rl}
(Types) & \tau \ ::= \alpha \mid \tau \to \tau' \mid \Pi\alpha.\,\tau \mid \mu\alpha.\,\tau \\
(Terms) & e \ ::= x \mid \lambda x{:}\tau.\,e \mid e\ e' \mid \Lambda\alpha.\,e \mid e\,[\tau] \mid \mathsf{fold}_\tau\ e \mid \mathsf{unfold}_\tau\ e\ e' \\
(Values) & v \ ::= \lambda x{:}\tau.\,e \mid \Lambda\alpha.\,e \mid \mathsf{fold}_\tau\ e \\
(Type\ Env.) & \Delta ::= \cdot \mid \Delta, \alpha \\
(Term\ Env.) & \Gamma \ ::= \cdot \mid \Gamma,\, x{:}\tau
\end{array}
$$

Fig. 4. Syntax of F_μ

$$
(\lambda x{:}\tau.\,e)\ e' \hookrightarrow e[e'/x] \quad \text{(R-APP)} \qquad (\Lambda\alpha.\,e)\,[\tau] \hookrightarrow e[\tau/\alpha] \quad \text{(R-TAPP)}
$$

$$
\frac{e \hookrightarrow e'}{\lambda x{:}\tau.\,e \hookrightarrow \lambda x{:}\tau.\,e'}\ \text{(R-LAM)} \qquad \frac{e \hookrightarrow e'}{e\ e'' \hookrightarrow e'\ e''}\ \text{(R-APP1)}
$$

$$
\frac{e \hookrightarrow e'}{v\ e \hookrightarrow v\ e'}\ \text{(R-APP2)} \qquad \frac{e \hookrightarrow e'}{\Lambda\alpha.\,e \hookrightarrow \Lambda\alpha.\,e'}\ \text{(R-TLAM)} \qquad \frac{e \hookrightarrow e'}{e\,[\tau] \hookrightarrow e'\,[\tau]}\ \text{(R-TAPP1)}
$$

$$
\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_\tau\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e') \hookrightarrow \mathsf{fold}_{\tau'}\ (e'\ e) \quad \text{(R-UNFOLD)}
$$

$$
\frac{e \ \hookrightarrow\ e'}{\mathsf{fold}_\tau\ e \ \hookrightarrow\ \mathsf{fold}_\tau\ e'}\ \text{(R-FOLD)} \qquad \frac{e \hookrightarrow e'}{\mathsf{unfold}_\tau\ e''\ e \hookrightarrow \mathsf{unfold}_\tau\ e''\ e'}\ \text{(R-UNFOLD1)}
$$

$$
\frac{e \hookrightarrow e'}{\mathsf{unfold}_\tau\ e\ v \hookrightarrow \mathsf{unfold}_\tau\ e'\ v}\ \text{(R-UNFOLD2)}
$$

Fig. 5. Reduction rules of F_μ

3 F_μ

Our approach to support unrestricted recursive types is general and applicable to many type systems and logics. In this paper, we use system F as an example, extend it with unrestricted recursive types, and name it as F_μ. In this section we give the formal definition of F_μ, with the extended constructs highlighted. We discuss subject reduction and progress properties for F_μ here and leave the discussion of the strong normalization property of F_μ to the next two sections.

We present the syntax of F_μ in Figure 4. Not surprisingly, a recursive type $\mu\alpha.\,\tau$ specifies a recursive type variable α and a type body τ. The folding constructor fold takes a term e of type τ and rolls it into $(\mathsf{fold}_\tau\ e)$. The first interesting feature of F_μ is its unfolding constructor unfold, which takes not one, but two, terms e and e', and unrolls them into term $(\mathsf{unfold}_\tau\ e\ e')$ of type τ. Any terms starting with fold are considered a value.

The reduction steps of F_μ is presented in Figure 5. Rule (R-FOLD), (R-UNFOLD1), and (R-UNFOLD2) are very simple—they allow the reduction

$$\frac{\mathrm{FTV}(\tau) \subseteq \Delta}{\Delta \vdash_\mu \tau} \text{ (TYPE)} \qquad \frac{\Delta \vdash_\mu \Gamma(x)}{\Delta; \Gamma \vdash_\mu x : \Gamma(x)} \text{ (VAR)}$$

$$\frac{\Delta; \Gamma, x:\tau \vdash_\mu e:\tau'}{\Delta; \Gamma \vdash_\mu \lambda x:\tau.\, e : \tau \to \tau'} \text{ (LAM)} \qquad \frac{\Delta; \Gamma \vdash_\mu e:\tau \to \tau' \qquad \Delta; \Gamma \vdash_\mu e':\tau}{\Delta; \Gamma \vdash_\mu e\ e' : \tau'} \text{ (APP)}$$

$$\frac{\Delta, \alpha; \Gamma \vdash_\mu e:\tau}{\Delta; \Gamma \vdash_\mu \Lambda\alpha.\, e : \Pi\alpha.\, \tau} \text{ (TLAM)} \qquad \frac{\Delta; \Gamma \vdash_\mu e:\Pi\alpha.\, \tau \qquad \Delta \vdash_\mu \tau'}{\Delta; \Gamma \vdash_\mu e\,[\tau'] : \tau[\tau'/\alpha]} \text{ (TAPP)}$$

$$\frac{\Delta; \Gamma \vdash_\mu e:\tau[\mu\alpha.\, \tau/\alpha]}{\Delta; \Gamma \vdash_\mu \mathsf{fold}_{\tau[\mu\alpha.\, \tau/\alpha]}\ e : \mu\alpha.\, \tau} \text{ (FOLD)}$$

$$\frac{\Delta; \Gamma \vdash_\mu e:\mu\alpha.\, \tau \qquad \Delta; \Gamma \vdash_\mu e':\mu\_.\, (\tau[\mu\alpha.\, \tau/\alpha] \to \tau'[\mu\alpha'.\, \tau'/\alpha'])}{\Delta; \Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\, \tau'}\ e\ e' : \mu\alpha'.\, \tau'} \text{ (UNFOLD)}$$

Fig. 6. Static semantics of F_μ

of sub-terms. Rule (R-UNFOLD) might looks strange when first being seen, but will become easy to understand when the static semantics of F_μ is discussed in later this section. Although rule (R-FOLD) looks not that special, to admit it while maintaining the strong normalization properties, as will be discussed in the next two sections, is not trivial at all. It is worth noting that Church-Rosser theorem holds for F_μ, but not if rule (R-FOLD) is excluded from the system. In contrast, GTAL [6] essentially does not have rule (R-FOLD), and thus does not have Church-Rosser property. In this paper, however, we focus on the strong normalization proof and do not discuss the Church-Rosser proof.

We present the static semantics of F_μ in Figure 6. The introduction rule for recursive types, rule (FOLD), is not surprising at all. However, the elimination rule for recursive types, rule (UNFOLD), might look unconventional. The idea of rule (UNFOLD) is to take the term e of a recursive type $\mu\alpha.\, \tau$, do the naïve unfolding and extract a term of type $\tau[\mu\alpha.\, \tau/\alpha]$, and supply it to another term that transforms terms of type $\tau[\mu\alpha.\, \tau/\alpha]$ into terms of type $\tau'[\mu\alpha'.\, \tau'/\alpha']$, as encapsulated in term e' of a "dummy" recursive type $\mu\_.\, (\tau[\mu\alpha.\, \tau/\alpha] \to \tau'[\mu\alpha'.\, \tau'/\alpha'])$. The naïve reasoning tells us that we should obtain a term of type $\tau'[\mu\alpha'.\, \tau'/\alpha']$ by now. However, to make sure that one do not "abuse" the folding/unfolding operations, that term is immediately folded back and becomes of type $\mu\alpha'.\, \tau'$, which is the type of $(\mathsf{unfold}_{\mu\alpha'.\, \tau'}\ e\ e')$. This is obvious from the reduction rule (R-UNFOLD), which extracts two sub-terms of recursive types, applies one to another, and wraps the result back into terms of recursive types. Despite rule (R-UNFOLD)'s unconventional shape, we still call it the elimination rule of recursive types, as it does consume two recursive types and only generate one.

We proved the following subject reduction and progress theorems of F_μ. The detailed proof is available in Appendix A. Instead of directly proving the strong normalization theorem of F_μ here, we prove it in the next two sections by indirections to the strong normalization property of System F.

Lemma 4 ($\mathbf{F}_\mu$ Subsitutions).

1. *If* $\Delta, \alpha \vdash_\mu \tau$ *and* $\Delta \vdash_\mu \tau'$ *then* $\Delta \vdash_\mu \tau[\tau'/\alpha]$;
2. *If* $\Delta, \alpha; \Gamma \vdash_\mu e : \tau$ *and* $\Delta \vdash_\mu \tau'$ *then* $\Delta; \Gamma[\tau/\alpha] \vdash_\mu e[\tau'/\alpha] : \tau[\tau'/\alpha]$;
3. *If* $\Delta; \Gamma, x : \tau \vdash_\mu e : \tau'$ *and* $\Delta; \Gamma \vdash_\mu e' : \tau$ *then* $\Delta; \Gamma \vdash_\mu e[e'/x] : \tau'$.

Theorem 5 ($\mathbf{F}_\mu$ Subject Reduction).
If $\Delta; \Gamma \vdash_\mu e : \tau$ *and* $e \hookrightarrow e'$ *then* $\Delta; \Gamma \vdash_\mu e' : \tau$.

Theorem 6 ($\mathbf{F}_\mu$ Progress).
If $\cdot; \cdot \vdash_\mu e : \tau$ *then either* $e = v$ *or* $e \hookrightarrow e'$.

4 Strong Normalization of $\mathbf{F}_\mu$ without Rule (R-FOLD)

In this and next sections we prove the strong normalization theorem of F_μ in two steps. For the first step, we prove in this section that F_μ, without reduction rule (R-FOLD), satisfies strong normalization. This is done by mapping F_μ types, terms, and typing and reduction derivations into System F, and utilize the strong normalization result of System F. For the second step, we prove in next section the strong normalization theorem of the complete F_μ system (including reduction rule (R-FOLD)), by mapping all F_μ reduction sequences into those without rule (R-FOLD)) and utilize the strong normalization result proved in this section.

We use $\rightharpoonup$ to denote F_μ reduction steps that do not use rule (R-FOLD) in their derivations. We use $\hookrightarrow^+$ to represent one or more consective steps of $\hookrightarrow$ reduction, i.e.,

$$e \hookrightarrow^+ e' \triangleq e \hookrightarrow e_1 \hookrightarrow e_2 \hookrightarrow \ldots \hookrightarrow e'.$$

Similarly, $\rightharpoonup^+$ is used to represent one or more consecutive steps of $\rightharpoonup$ reduction. Notice that, in this section when $\hookrightarrow$ and $\hookrightarrow^+$ are used, they only denote the reduction steps in System F, as is obvious from the context.

In Figure 7 we define the translations from F_μ types and terms to System F ones. Similar to the core idea used in GTAL [6], we map all recursive types to True type, i.e., $\Pi\alpha.\, \alpha \to \alpha$. This is because the recursive types does not make major impact to the reductions structure. All introduction term of recursive types, $(\mathsf{fold}_\tau\ e)$, is mapped to true term, i.e., $\Lambda\alpha.\, \lambda x : \alpha.\, x$, which is of type $\Pi\alpha.\, \alpha \to \alpha$. This is in co-ordinance with the fact that in this section we do not allow reductions of sub-terms inside fold. The translation of term $\mathsf{unfold}_\tau\ e\ e'$ keeps both term e and e' after the translation, so that the reductions inside them can be mapped to the System F level.

The translation of term $((\lambda x : \tau.\, \mathsf{fold}_\tau\ x)\ e)$ is rather interesting. Instead of doing the usually translation for application terms, we lift the fold constructor from the right half into the left half, and obtain $((\lambda x : \Pi\alpha.\, \alpha \to \alpha.\, x)\ \ulcorner \mathsf{fold}_\tau\ e \urcorner)$.

$$
\begin{array}{lcl}
\ulcorner \alpha \urcorner & = & \alpha \\
\ulcorner \tau \to \tau' \urcorner & = & \ulcorner \tau \urcorner \to \ulcorner \tau' \urcorner \\
\ulcorner \Pi\alpha.\, \tau \urcorner & = & \Pi\alpha.\, \ulcorner \tau \urcorner \\
\ulcorner \mu\alpha.\, \tau \urcorner & = & \Pi\alpha.\, \alpha \to \alpha \\
 & & \\
\ulcorner x \urcorner & = & x \\
\ulcorner \lambda x{:}\tau.\, e \urcorner & = & \lambda x{:}\ulcorner \tau \urcorner.\, \ulcorner e \urcorner \\
\ulcorner (\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)\ e \urcorner & = & (\lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x)\ \ulcorner \mathsf{fold}_\tau\ e \urcorner \\
\ulcorner e\ e' \urcorner & = & \ulcorner e \urcorner\ \ulcorner e' \urcorner \\
\ulcorner \Lambda\alpha.\, e \urcorner & = & \Lambda\alpha.\, \ulcorner e \urcorner \\
\ulcorner e\,[\tau] \urcorner & = & \ulcorner e \urcorner\,[\ulcorner \tau \urcorner] \\
\ulcorner \mathsf{fold}_\tau\ e \urcorner & = & \Lambda\alpha.\, \lambda x{:}\alpha.\, x \\
\ulcorner \mathsf{unfold}_\tau\ e\ e' \urcorner & = & (\lambda\_{:}\Pi\alpha.\, \alpha \to \alpha.\, \ulcorner e' \urcorner)\ \ulcorner e \urcorner \\
 & & \\
\ulcorner x_1{:}\tau_1, \ldots, x_n{:}\tau_n \urcorner & = & x_1{:}\ulcorner \tau_1 \urcorner, \ldots, x_n{:}\ulcorner \tau_n \urcorner
\end{array}
$$

Fig. 7. Translation of F_μ constructs into System F

The intuition here is that this adds one meaningless reduction step in the System F level, and will compensate for the one virtual reduction step introduced by rule (R-FOLD1) in the F_μ level, as will be discussed in the next section. It will be more obvious when we move on to the next section.

We prove the strong normalization property of F_μ without rule (R-FOLD) by showing a correspondence between reductions in it and those in System F, using the above translation. In the following we list the lemmas and theorems used for the proof, as well as some interesting cases. The detailed proof is available in Appendix A.

Lemma 7 (Translation Substituion Preservation).

1. $\ulcorner \tau[\tau'/\alpha] \urcorner = \ulcorner \tau \urcorner[\ulcorner \tau' \urcorner/\alpha]$;
2. $\ulcorner e[\tau/\alpha] \urcorner = \ulcorner e \urcorner[\ulcorner \tau \urcorner/\alpha]$;
3. $\ulcorner e[e'/x] \urcorner = \ulcorner e \urcorner[\ulcorner e' \urcorner/x]$.

Theorem 8 (Translation Type Wellformedness Preservation).
If $\Delta \vdash_\mu \tau$ *then* $\Delta \vdash \ulcorner \tau \urcorner$.

Theorem 9 (Translation Typing Preservation).
If $\Delta; \Gamma \vdash_\mu e{:}\tau$ *then* $\Delta; \ulcorner \Gamma \urcorner \vdash \ulcorner e \urcorner{:}\ulcorner \tau \urcorner$.

Proof. By induction on the derivation of $\Delta; \Gamma \vdash_\mu e{:}\tau$.

case (APP). $\dfrac{\Delta; \Gamma \vdash_\mu e{:}\tau \to \tau' \qquad \Delta; \Gamma \vdash_\mu e'{:}\tau}{\Delta; \Gamma \vdash_\mu e\ e'{:}\tau'}$.

By induction hypothesis it follows that $\Delta; \ulcorner \Gamma \urcorner \vdash \ulcorner e' \urcorner{:}\ulcorner \tau \urcorner$.

Depending on the shape of e, there are two sub-cases.

case $e = \lambda x{:}\tau.\, \mathsf{fold}_\tau\ x$.

By $\Delta; \Gamma \vdash_\mu \lambda x{:}\tau.\, \mathsf{fold}_\tau\ x : \tau \to \tau'$ it follows that τ' has the shape $\mu\alpha.\, \tau''$.

We have $$\frac{}{\Delta; \ulcorner\Gamma\urcorner, x{:}\Pi\alpha.\, \alpha \to \alpha \vdash x : \Pi\alpha.\, \alpha \to \alpha}$$
$$\frac{}{\Delta; \ulcorner\Gamma\urcorner \vdash \lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x : (\Pi\alpha.\, \alpha \to \alpha) \to \Pi\alpha.\, \alpha \to \alpha}$$

and $$\frac{}{\Delta, \alpha; \ulcorner\Gamma\urcorner \vdash \lambda x{:}\alpha.\, x : \alpha \to \alpha}$$
$$\frac{}{\Delta; \ulcorner\Gamma\urcorner \vdash \Lambda\alpha.\, \lambda x{:}\alpha.\, x : \Pi\alpha.\, \alpha \to \alpha}\ .$$

It then follows that

$$\frac{\begin{array}{l}\Delta; \ulcorner\Gamma\urcorner \vdash \lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x : (\Pi\alpha.\, \alpha \to \alpha) \to \Pi\alpha.\, \alpha \to \alpha \\ \Delta; \ulcorner\Gamma\urcorner \vdash \Lambda\alpha.\, \lambda x{:}\alpha.\, x : \Pi\alpha.\, \alpha \to \alpha\end{array}}{\Delta; \ulcorner\Gamma\urcorner \vdash (\lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x)\ \Lambda\alpha.\, \lambda x{:}\alpha.\, x : \Pi\alpha.\, \alpha \to \alpha}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\Delta; \ulcorner\Gamma\urcorner \vdash \ulcorner(\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)\ e'\urcorner : \ulcorner\tau'\urcorner$.

□

Theorem 10 (Translation Reduction Preservation).
If $e \rightharpoonup e'$ then $\ulcorner e\urcorner \hookrightarrow^+ \ulcorner e'\urcorner$.

Proof. By induction on the derivation of $e \rightharpoonup e'$.

case (R-APP). $(\lambda x{:}\tau.\, e)\ e' \rightharpoonup e[e'/x]$.

Depending on the shape of e, there are two sub-cases.

case $e = \mathsf{fold}_\tau\ x$.

By the definition of $\ulcorner\cdot\urcorner$ and Lemma 7 it follows that

$$\ulcorner(\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)\ e'\urcorner = (\lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x)\ \Lambda\alpha.\, \lambda x{:}\alpha.\, x$$
$$\hookrightarrow \Lambda\alpha.\, \lambda x{:}\alpha.\, x = (\Lambda\alpha.\, \lambda x{:}\alpha.\, x)[\ulcorner e'\urcorner/x] = \ulcorner\mathsf{fold}_\tau\ x\urcorner[\ulcorner e'\urcorner/x] = \ulcorner(\mathsf{fold}_\tau\ x)[e'/x]\urcorner.$$

case (R-UNFOLD). $\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_\tau\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e') \rightharpoonup \mathsf{fold}_{\tau'}\ (e'\ e)$.

By the definition of $\ulcorner\cdot\urcorner$ it follows that

$$\ulcorner\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_\tau\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e')\urcorner = (\lambda\_{:}\Pi\alpha.\, \alpha \to \alpha.\, \ulcorner\mathsf{fold}_{\tau\to\tau'}\ e'\urcorner)\ \ulcorner\mathsf{fold}_\tau\ e\urcorner$$
$$= (\lambda\_{:}\Pi\alpha.\, \alpha \to \alpha.\, \Lambda\alpha.\, \lambda x{:}\alpha.\, x)\ (\Lambda\alpha.\, \lambda x{:}\alpha.\, x) \hookrightarrow \Lambda\alpha.\, \lambda x{:}\alpha.\, x = \ulcorner\mathsf{fold}_{\tau'}\ (e'\ e)\urcorner.$$

□

Theorem 11 ($\mathbf{F}_\mu$ Strong Normalization ($\rightharpoonup$)).
If $\cdot; \cdot \vdash_\mu e : \tau$ then the length of any reduction sequence starting with e, without using reduction rule (R-FOLD)*, is finite.*

Proof. For any reduction sequence $e \rightharpoonup e_1 \rightharpoonup e_2 \rightharpoonup \ldots$, by Theorem 10 it follows that there exists the following reduction sequence

$$\ulcorner e \urcorner \hookrightarrow^+ \ulcorner e_1 \urcorner \hookrightarrow^+ \ulcorner e_2 \urcorner \hookrightarrow^+ \ldots.$$

By Theorem 9 it follows that $\cdot;\cdot \vdash \ulcorner e \urcorner : \ulcorner \tau \urcorner$. By Theorem 3 the length of the above reduction sequence is finite.

But the above reduction sequence is no shorter than the original sequence, thus the length of reduction sequence $e \rightharpoonup e_1 \rightharpoonup e_2 \rightharpoonup \ldots$ is also finite. ■

5 Strong Normalization of F_μ

In this section we prove that strong normalization property of F_μ still holds even when reduction rule (R-FOLD) is allowed. This is done by mapping all F_μ reduction steps ($\hookrightarrow$) into those without rule (R-FOLD) ($\rightharpoonup$), and utilize the strong normalization result proved in the previous section.

The key observation here is that any reduction steps that involve reduction of sub-terms **inside** a fold term corresponds to some reduction steps **outside** a fold term, while preserving the whole terms' typing. This is illustrated by the following example.

$$\frac{e \hookrightarrow e'}{\mathsf{fold}_\tau\ e \hookrightarrow \mathsf{fold}_\tau\ e'} \quad \Longrightarrow \quad \frac{e \hookrightarrow e'}{(\lambda x{:}\tau.\,\mathsf{fold}_\tau\ x)\ e \hookrightarrow (\lambda x{:}\tau.\,\mathsf{fold}_\tau\ x)\ e'}$$

and

$$\frac{\Delta;\Gamma \vdash_\mu e : \tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma \vdash_\mu \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e : \mu\alpha.\,\tau} \quad \Longrightarrow$$

$$\frac{\Delta;\Gamma \vdash_\mu \lambda x{:}\tau[\mu\alpha.\,\tau/\alpha].\,\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x{:}\tau[\mu\alpha.\,\tau/\alpha]{\to}\mu\alpha.\,\tau \quad \Delta;\Gamma \vdash_\mu e : \tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma \vdash_\mu (\lambda x{:}\tau[\mu\alpha.\,\tau/\alpha].\,\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x)\ e : \mu\alpha.\,\tau}$$

In particular, we do not directly map $\hookrightarrow$ reduction steps to $\rightharpoonup$ reduction steps. Instead, we first define the following new reduction rule

$$(\lambda x{:}\mu\alpha.\,\tau.\,x)(\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e) \hookrightarrow (\lambda x{:}\tau[\mu\alpha.\,\tau/\alpha].\,\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x)\,e \quad \text{(R-FOLD1)}$$

and use $\rightsquigarrow$ to represent reduction steps that might use rule (R-FOLD1) as well as those rules allowed for $\rightharpoonup$, but not rule (R-FOLD). Similar to $\hookrightarrow^+$ and $\rightharpoonup^+$, we use $\rightsquigarrow^+$ to represent one or more consecutive steps of $\rightsquigarrow$ reduction.

The proof is done by first proving the strong normalization of $\rightsquigarrow$ reduction sequences, utilizing the strong normalization results from previous sections. We then map $\hookrightarrow$ reduction steps to $\rightsquigarrow$ steps, and thus prove the strong normalization property for the complete F_μ.

5.1 Strong Normalization of $\rightsquigarrow$ Reduction Sequences

The key insight here is that, while the newly defined reduction rule (R-FOLD1) introduces "virtual" reduction steps that does not seems to map to any real reduction steps in the System F level, there "happens" to be a specially defined path for term of the shape $((\lambda x\!:\!\tau.\,\mathsf{fold}_\tau\ x)\ e)$ in the translation $\ulcorner\cdot\urcorner$, and the additional "concrete" reduction steps introduced there exactly compensates these "virtual" reduction steps.

In the following we list the lemmas and theorems used for the proof, as well as some interesting cases. The detailed proof is available in Appendix A.

Lemma 12 ((R-FOLD1) **Reduction Steps Finite).**
The length of any $\rightsquigarrow$ *reduction sequences that uses rule* (R-FOLD1) *in every step is finite.*

Proof.

Every use of rule (R-FOLD1) consumes a sub-term of the shape $(\lambda x\!:\!\mu\alpha.\,\tau.\,x)$.

When rule (R-FOLD1) is used in a derivation, rule (R-APP), (R-TAPP), and (R-UNFOLD) can not be used in the same derivation.

Other reduction rules can not generate any new sub-terms by themselves.

So every reduction step using rule (R-FOLD1) consumes one $(\lambda x\!:\!\mu\alpha.\,\tau.\,x)$.

But the number of $(\lambda x\!:\!\mu\alpha.\,\tau.\,x)$ sub-terms in any given term is finite.

Thus the length of such $\rightsquigarrow$ reduction sequences must be finite. ■

Lemma 13 ((R-FOLD1) **Translation Equivalence).**
If the derivation of $e \rightsquigarrow e'$ *uses reduction rule* (R-FOLD1) *then* $\ulcorner e\urcorner = \ulcorner e'\urcorner$.

Proof. By induction on the derivation of $e \rightsquigarrow e'$.

case (R-FOLD1).

$$(\lambda x\!:\!\mu\alpha.\,\tau.\,x)\ (\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e) \hookrightarrow (\lambda x\!:\!\tau[\mu\alpha.\,\tau/\alpha].\,\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x)\ e.$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that

$$\begin{aligned}
&\ulcorner(\lambda x\!:\!\mu\alpha.\,\tau.\,x)\ (\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e)\urcorner\\
&= (\lambda x\!:\!\ulcorner\mu\alpha.\,\tau\urcorner.\,x)\ \ulcorner\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e\urcorner\\
&= (\lambda x\!:\!\Pi\alpha.\,\alpha\to\alpha.\,x)\ \ulcorner\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e\urcorner\\
&= \ulcorner(\lambda x\!:\!\tau[\mu\alpha.\,\tau/\alpha].\,\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x)\ e\urcorner.
\end{aligned}$$ □

Theorem 14 ($\mathbf{F}_\mu$ Strong Normalization ($\rightsquigarrow$)).
If $\cdot;\cdot\vdash_\mu e:\tau$ *then the length of any reduction sequence starting with* e*, without using reduction rule* (R-FOLD)*, but might using the additional reduction rule* (R-FOLD1)*, is finite.*

$$
\begin{array}{rcl}
|x| & = & x \\
|\lambda x{:}\tau.\, e| & = & \lambda x{:}\tau.\, |e| \\
|e\ e'| & = & |e|\ |e'| \\
|\Lambda\alpha.\, e| & = & \Lambda\alpha.\, |e| \\
|e\,[\tau]| & = & |e|\,[\tau] \\
|\mathsf{fold}_\tau\ e| & = & (\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)\ |e| \\
|\mathsf{unfold}_\tau\ e\ e'| & = & (\lambda x{:}\tau.\, x)\ (\mathsf{unfold}_\tau\ |e|\ |e'|)
\end{array}
$$

Fig. 8. Coercion function $|\cdot|$

Proof.

By checking whether rule (R-FOLD1) appears in each reduction step, we can write any such reduction sequence in the following shape

$$e \rightsquigarrow_f^* e' \multimap^+ e_1 \rightsquigarrow_f^* e'_1 \multimap^+ e_2 \rightsquigarrow_f^* e'_2 \multimap^+ \ldots$$

where $\rightsquigarrow_f^*$ stands for zero or more consecutive $\rightsquigarrow$ reduction steps that uses rule (R-FOLD1). (Remember that the only difference between $\rightsquigarrow$ and $\multimap$ is the usage of rule (R-FOLD1).)

By Theorem 10 and 13 it follows that there exists the following reduction sequence

$$\ulcorner e \urcorner = \ulcorner e' \urcorner \hookrightarrow^+ \ulcorner e_1 \urcorner = \ulcorner e'_1 \urcorner \hookrightarrow^+ \ulcorner e_2 \urcorner = \ulcorner e'_2 \urcorner \hookrightarrow^+ \ldots.$$

By Theorem 9 it follows that $\cdot;\cdot \vdash \ulcorner e \urcorner : \ulcorner \tau \urcorner$.

By Theorem 11 the length of the above reduction sequence is finite.

But the length of the above sequence is no shorter than the sum of the lengths of all $\multimap^+$ in the original sequence. Thus the length of any $\multimap^+$ in the original sequences is finite.

Since every $\hookrightarrow^+$ in the above reduction sequence corresponds to a $\multimap^+$ in the original sequence, the number of $\multimap^+$ in the original sequence must be finite.

Thus the number of $\rightsquigarrow_f^*$ in the original sequence must also be finite.

But by Theorem 12 the length of every $\rightsquigarrow_f^*$ in the original sequence is finite.

Thus the length of the original reduction sequence must be finite. ■

5.2 Strong Normalization of $\hookrightarrow$ Reduction Sequences

The idea in this step is to define a coercion function $|\cdot|$, as presented in Figure 8, that transforms any F_μ terms into beta-equivalent forms that only contain fold constructor inside sub-terms in shape of $(\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)$, thus allowing the reduction of those sub-terms originally inside fold terms, even if reduction rule (R-FOLD) is missing.

In the following we list the lemmas and theorems used for the proof, as well as some interesting cases. The detailed proof is available in Appendix A.

Lemma 15 (Coercion Substituion Preservation).

1. $|e[\tau/\alpha]| \;=\; |e|[\tau/\alpha]$;
2. $|e[e'/x]| \;=\; |e|[|e'|/x]$.

Theorem 16 (Coercion Typing Preservation).
If $\Delta;\Gamma \vdash_\mu e:\tau$ *then* $\Delta;\Gamma \vdash_\mu |e|:\tau$.

Theorem 17 (Coercion Reduction Preservation).
If $e \hookrightarrow e'$ *then* $|e| \rightsquigarrow^+ |e'|$.

Proof. By induction on the derivation of $e \hookrightarrow e'$.

case (R-UNFOLD). $\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_\tau\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e') \;\hookrightarrow\; \mathsf{fold}_{\tau'}\ (e'\ e)$.

By the definition of $|\cdot|$ it follows that

$$
\begin{array}{ll}
 & |\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_\tau\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e')| \\
= & (\lambda x{:}\tau_0.\, x)\ (\mathsf{unfold}_{\tau''}\ |\mathsf{fold}_\tau\ e|\ |\mathsf{fold}_{\tau\to\tau'}\ e'|) \\
= & (\lambda x{:}\tau_0.\, x)\ (\mathsf{unfold}_{\tau''}\ ((\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)\ |e|)\ ((\lambda x{:}\tau\to\tau'.\, \mathsf{fold}_{\tau\to\tau'}\ x)\ |e'|)) \\
\rightsquigarrow & (\lambda x{:}\tau_0.\, x)\ (\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_\tau\ |e|)\ ((\lambda x{:}\tau\to\tau'.\, \mathsf{fold}_{\tau\to\tau'}\ x)\ |e'|)) \\
\rightsquigarrow & (\lambda x{:}\tau_0.\, x)\ (\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_\tau\ |e|)\ (\mathsf{fold}_{\tau\to\tau'}\ |e'|)) \\
\rightsquigarrow & (\lambda x{:}\tau_0.\, x)\ (\mathsf{fold}_{\tau'}\ (|e'|\ |e|)) \\
\rightsquigarrow & (\lambda x{:}\tau'.\, \mathsf{fold}_{\tau'}\ x)\ (|e'|\ |e|) \;=\; (\lambda x{:}\tau'.\, \mathsf{fold}_{\tau'}\ x)\ |e'\ e| \;=\; |\mathsf{fold}_{\tau'}\ (e'\ e)|.
\end{array}
$$

Notice that here in the shaded step we use the additional reduction rule (R-FOLD1) defined in the beginning of this section to lift the fold constructor, as the coercion $|\cdot|$ would do. □

Now we are able to show the strong normalization property for the entire F_μ.

Theorem 18 ($\mathbf{F}_\mu$ Strong Normalization ($\hookrightarrow$)).
If $\cdot;\cdot \vdash_\mu e:\tau$ *then the length of any reduction sequence starting with* e *is finite.*

Proof. For any reduction sequence $e \hookrightarrow e_1 \hookrightarrow e_2 \hookrightarrow \ldots$, by Theorem 17 it follows that there exists the following reduction sequence

$$|e| \;\rightsquigarrow^+\; |e_1| \;\rightsquigarrow^+\; |e_2| \;\rightsquigarrow^+\; \ldots.$$

By Theorem 16 it follows that $\cdot;\cdot \vdash |e|:\tau$. By Theorem 14 the length of the above reduction sequence is finite.

But the above reduction sequence is no shorter than the original sequence, thus the length of reduction sequence $e \hookrightarrow e_1 \hookrightarrow e_2 \hookrightarrow \ldots$ is also finite. ■

Reviewer #2 provided a "counter-example" to the strong normalization property of F_μ by giving the following definitions

$$\begin{array}{rcl} r & = & \mu\beta.\,\beta \\ s & = & \mu\alpha.\,(\alpha \to r) \\ t & = & s \to r \\ & = & (\mu\alpha.\,(\alpha \to r)) \to r \\ & = & (\alpha \to r)[\mu\alpha.\,(\alpha \to r)/\alpha] \\ u & = & t \to r \\ f & = & \lambda x\!:\!s.\,\mathsf{unfold}_r\ x\ (\mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x)) \end{array}$$

and showing the following diverging reduction steps

$$\begin{array}{rcl} f\ (\mathsf{fold}_t\ f) & \hookrightarrow & \mathsf{unfold}_r\ (\mathsf{fold}_t\ f)\ (\mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ (\mathsf{fold}_t\ f))) \\ & \hookrightarrow & \mathsf{fold}_r\ ((\lambda x'\!:\!t.\,x'\ (\mathsf{fold}_t\ f))\ f) \\ & \hookrightarrow & \mathsf{fold}_r\ (f\ (\mathsf{fold}_t\ f)) \end{array}$$

as the source term $(f\ (\mathsf{fold}_t\ f))$ appears as a sub-term in the reduction target.

We want to point out that the above reduction is impossible, thus the above is not a "counter-example" to the strong normalization property of F_μ.

The problem is that both f and $(f\ (\mathsf{fold}_t\ f))$ are not well-typed terms. We show this by first infering the complete type of f, as it is not given in the review.

$$x \text{ has type } s.$$

$$x' \text{ has type } t, \text{ i.e., } s \to r.$$

$$x'\ x \text{ has type } r.$$

$$(\lambda x'\!:\!t.\,x'\ x) \text{ has type } t \to r, \text{ i.e, } u.$$

$$\text{By inspection } \mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x) \text{ can only have type } \mu\_.\,(t \to r).$$

$\mathsf{unfold}_r\ x\ (\mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x))$ is not a well-formed term, as type s, the type of x, unfolds to type t, and the type of $\mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x)$ is $\mu\_.\,(t \to r)$, which, according to rule (UNFOLD), means that the type of $\mathsf{unfold}_r\ x\ (\mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x))$ should unfold to r. As annotated in the term, this type is r itself. However, by no means can type r unfold to itself. Instead, the only valid term here should be $\mathsf{unfold}_{\mu\_.\,r}\ x\ (\mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x))$, which is validated by the following derivation.

$$\frac{\Delta;\Gamma \vdash_\mu x\!:\!s \qquad \Delta;\Gamma \vdash_\mu \mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x)\!:\!\mu\_.\,(t \to r)}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\mu\_.\,r}\ x\ (\mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x))\!:\!\mu\_.\,r}$$

With this change, term f can only be defined now as

$$\lambda x\!:\!s.\,\mathsf{unfold}_{\mu\_.\,r}\ x\ (\mathsf{fold}_u\ (\lambda x'\!:\!t.\,x'\ x)) \qquad \text{and has type} \qquad s \to \mu\_.\,r$$

Term f no longer has the type t as reviewer #2 implied by $\mathsf{fold}_t\ f$. In fact, $\mathsf{fold}_t\ f$ is not a well-formed term. The only valid term here is $\mathsf{fold}_{s \to \mu\_.\,r}\ f$, which can only have type $\mu\_.\,(s \to \mu\_.\,r)$.

Apparently $f\ (\mathsf{fold}_{s \to \mu\_.\, r}\ f)$ is not a well-formed term, as f is of type $s \to \mu\_.\, r$, and s and $\mu\_.\, (s \to \mu\_.\, r)$ are not the same. Thus the reduction steps given by reviewer #2 can not happen, and does not form a "counter-example" to the strong normalization property of F_μ.

Reviewer #2 also estimated the "error" associated with the above "counter-example" is in the proof of Theorem 10, case (R-APP2), as v could be in the form of $\lambda x\!:\!\tau.\, \mathsf{fold}_\tau\ x$, and was overlooked by the proof. We thank reviewer #2 for pointing this out and will address it in the revision of the paper.

6 The Usage of F_μ

Now that F_μ is proved to be consistent and strongly normalizing, we discuss in this section on how the unrestricted recursive types in it might be used.

The folding rule (FOLD) defined in Figure 6 is very user friendly—no "goodness" checking is needed. Even "bad" recursive types are allowed, which comes to the point here: instead of defining and exclude "bad" recursive types and claiming that all usage of them "good", we declare all recursive types "good" and define and exclude "bad" usage of them.

The unfolding rule (UNFOLD) defined in Figure 6 (as duplicated here below) is easy for the strong normalization proof in the previous sections to proceed, but not exactly intuitive to use, as it requires a seemingly "dummy" ($\mu\_.$) header for the "actual" reasoning (e') between unfolded recursive types.

$$\frac{\Delta;\Gamma \vdash_\mu e : \mu\alpha.\,\tau \qquad \Delta;\Gamma \vdash_\mu e' : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e' : \mu\alpha'.\,\tau'}\ (\text{UNFOLD})$$

From rule (UNFOLD) we can derive the following rules (UNFOLD1) and (UNFOLD2). In fact, rule (UNFOLD2) can also be derived as special case of rule (UNFOLD1).

$$\frac{\begin{array}{l}\Delta;\Gamma \vdash_\mu e : \mu\alpha.\,\tau \qquad \Delta;\Gamma \vdash_\mu e' : \mu\alpha'.\,\tau' \\ \Delta;\Gamma \vdash_\mu e'' : \tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'] \to \tau''[\mu\alpha''.\,\tau''/\alpha'']\end{array}}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\dots}\ e'\ (\mathsf{unfold}_{\mu\_.\,\tau'[\dots/\alpha'] \to \tau''[\dots/\alpha'']}\ e\ (\mathsf{fold}_{\dots}\ e'')) : \mu\alpha''.\,\tau''}\ (\text{UNFOLD1})$$

$$\frac{\Delta;\Gamma \vdash_\mu e : \mu\alpha.\,\tau \qquad \Delta;\Gamma \vdash_\mu e' : \tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha']}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ (\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha']}\ e') : \mu\alpha'.\,\tau'}\ (\text{UNFOLD2})$$

Both of the these two derived rules, especially rule (UNFOLD2), are more intuitive and user-friendly. It is obvious from these two rules that the user can freely construct fucntion terms that converts between any unfolding form of recrusive types.

Furthermore, it is easy to prove the equivalence between rule (UNFOLD) and rule (UNFOLD1), i.e., define a variant of F_μ with following rule (UNFOLD1) instead of rule (UNFOLD), and show a two-way mapping between the derivations of F_μ and the variant. We omit the term part in the following derivations. It is trivial to prove

$$\Delta;\Gamma \vdash_\mu \tau[\mu\alpha.\,\tau/\alpha] \to (\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha']) \to \tau'[\mu\alpha'.\,\tau'/\alpha'].$$

and thus

$$\frac{\begin{array}{l}\Delta;\Gamma \vdash_\mu \mu\alpha.\,\tau \qquad \Delta;\Gamma \vdash_\mu \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha']) \\ \Delta;\Gamma \vdash_\mu \tau[\mu\alpha.\,\tau/\alpha] \to (\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha']) \to \tau'[\mu\alpha'.\,\tau'/\alpha']\end{array}}{\Delta;\Gamma \vdash_\mu \mu\alpha'.\,\tau'}$$

But one might ask the following question: since rule (UNFOLD) can only derive terms of recursive types, how can one use it in the derivation and derive non-recursive types?

If one wants to obtain $\tau[\mu\alpha.\,\tau/\alpha]$ from $\mu\alpha.\,\tau$, he should encapsulate the result and instead obtain $\mu\_.\,\tau[\mu\alpha.\,\tau/\alpha]$. If one originally wants to use $\tau[\mu\alpha.\,\tau/\alpha]$ to form a bigger derivation that leads to some non-recursive type τ', he should encapsulate the other part of the derivation in using a dummy ($\mu\_.$) header and use that together with $\mu\_.\,\tau[\mu\alpha.\,\tau/\alpha]$ to build a derivation that leads to type $\mu\_.\,\tau'$. Due to strong normalization and Church-Rosser properties it follows that

$$\cdot;\cdot \vdash_\mu e : \mu\alpha.\,\tau \qquad \Longrightarrow \qquad \cdot;\cdot \vdash_\mu e' : \tau[\mu\alpha.\,\tau/\alpha].$$

A special case of the above is

$$\cdot;\cdot \vdash_\mu e : \mu\_.\,\tau \qquad \Longrightarrow \qquad \cdot;\cdot \vdash_\mu e' : \tau.$$

which means any ground term of (potentially non-recursive) type τ hidden inside $\mu\_.\,\tau$ can be safely extracted.

7 Related Work and Conclusion

GTAL [6] is a closely-related work to this paper. The formation of unfolding rules and the first stage of strong normalization proof for F_μ was partly inspired by the GTAL and its plain-text version of termination proof. The core of GTAL and F_μ are very similar, as the modal operator $\circ$ can be roughly viewed as ($\mu\_.$). It should be possible to apply the result in this paper to GTAL to 1) remove the "modal" operator $\circ$ and let the unfolding form of $\mu\alpha.\,\tau$ be $\tau[\mu\alpha.\,\tau/\alpha]$ instead of $\circ(\tau[\mu\alpha.\,\tau/\alpha])$ and 2) allow reduction of sub-terms inside fold terms, so that it can gain the Church-Rosser property.

Other than the work mentioned in the introduction section [1–5], there are many other work such as Barthe et al. [8] that supports restricted forms of recursive types by various forms of meta-logical checking other than typing. The precise relation between those introduction-restricted systems and elimination-restricted systems such as GTAL [6] and the F_μ system in this paper is not entirely clear at this moment. Of particular interests here is the relative expressive power of these two style of systems.

The F_μ strong normalization proof presented in this paper uses the following three stages to map reduction steps between four sets of reduction rules in two languages.

$$\mathrm{F}_\mu \;\hookrightarrow\; \Longrightarrow\; \mathrm{F}_\mu \;\rightsquigarrow\; \Longrightarrow\; \mathrm{F}_\mu \;\rightharpoonup\; \Longrightarrow\; \text{System F} \;\hookrightarrow$$

This kind of reduction steps mapping technique has been widely used before, such as those for λ-cube by Barendregt [9].

The proof, although rather straightforward, is somewhat tedious. It might be possible to derive more elegant proof for strong normalization and other properties using techniques such as logical relations, such as presented by Crary and Harper [10].

Conclusion. We presented a new approach to add the support of unrestricted recursive types. Using System F as an example, we extended it to F_μ and construct detailed proof of its varitous properties, including strong normalization. Our approach is general and applicable to other similar type systems and logics, as well as hybrid verification systems such as XCAP [11, 12] or HTT [13, 14].

References

1. Coquand, T., Huet, G.: The calculus of constructions. Information and Computation **76** (1988) 95–120
2. Shao, Z., Saha, B., Trifonov, V., Papaspyrou, N.: A type system for certified binaries. In: Proc. 29th ACM Symposium on Principles of Programming Languages, ACM Press (January 2002) 217–232
3. Nakano, H.: A modality for recursion. In: Proc. 15th IEEE Symposium on Logic in Computer Science, Santa Barbara, CA, USA (June 2000) 255–266
4. Appel, A.W., Mellies, P.A., Richards, C.D., Vouillon., J.: A very modal model of a modern, major, general type system. In: Proc. 34th ACM Symposium on Principles of Programming Languages, Nice, France (January 2007) 109–122
5. MacQueen, D., Plotkin, G., Sethi, R.: An ideal model for recursive polymorphic types. In: Proc. 11th ACM Symposium on Principles of Programming Languages, Salt Lake City, UT, USA (January 1984) 165–174
6. Hawbilitzel, C., Huang, H., Wittie, L., Chen, J.: A garbage-collecting typed assembly language. In: Proc. 2007 ACM SIGPLAN International Workshop on Types in Language Design and Implementation, New York, NY, USA, ACM Press (January 2007)
7. Girard, J.Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge University Press (1989)
8. Barthe, G., Grégoire, B., Pastawski, F.: Cic[( )]: Type-based termination of recursive definitions in the calculus of inductive constructions. In: Proc. Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, Phnom Penh, Cambodia. (November 2006) 257–271
9. Barendregt, H.P.: Lambda calculi with types. In Abramsky, S., Gabbay, D., Maibaum, T., eds.: Handbook of Logic in Computer Science (volume 2), Oxford Univ. Press (1991)
10. Crary, K., Harper, R.: Syntactic logical relations for polymorphic and recursive types. Electron. Notes Theor. Comput. Sci. **172** (2007) 259–299
11. Ni, Z., Shao, Z.: Certified assembly programming with embedded code pointers. In: Proc. 33rd ACM Symposium on Principles of Programming Languages, Charleston, South Carolina (January 2006)
12. Ni, Z., Yu, D., Shao, Z.: Using xcap for systems programming: Machine context management. In: Proc. 20th International Conference on Theorem Proving in Higher Order Logics, Kaiserslautern, Germany (September 2007) 189–206
13. Nanevski, A., Morrisett, G., Birkedal, L.: Polymorphism and separation in hoare type theory. In: Proc. 11th ACM SIGPLAN International Conference on Functional Programming, Portland, OR, USA, ACM Press (September 2006) 62–73
14. Nanevski, A., Ahmed, A., Morrisett, G., Birkedal, L.: Abstract predicates and mutable adts in hoare type theory. In: Proc. 2007 European Symposium on Programming, Braga, Portugal (March 2007) 189–204

A Complete Proof of Lemmas and Theorems

Here we supply the complete proof of those lemmas and theorems with incomplete proof in the main paper.

Theorem 1 (System F Subject Reduction).
If $\Gamma;\Delta \vdash e:\tau$ *and* $e \hookrightarrow e'$ *then* $\Delta;\Gamma \vdash e':\tau$.

Theorem 2 (System F Progress).
If $\cdot;\cdot \vdash e:\tau$ *then either* $e = v$ *or* $e \hookrightarrow e'$.

Theorem 3 (System F Strong Normalization).
If $\cdot;\cdot \vdash e:\tau$ *then the length of any reduction sequence starting with* e *is finite.*

Lemma 4 ($\mathbf{F}_\mu$ Subsitutions).

1. *If* $\Delta, \alpha \vdash_\mu \tau$ *and* $\Delta \vdash_\mu \tau'$ *then* $\Delta \vdash_\mu \tau[\tau'/\alpha]$;
2. *If* $\Delta, \alpha; \Gamma \vdash_\mu e:\tau$ *and* $\Delta \vdash_\mu \tau'$ *then* $\Delta;\Gamma[\tau/\alpha] \vdash_\mu e[\tau'/\alpha]:\tau[\tau'/\alpha]$;
3. *If* $\Delta;\Gamma, x:\tau \vdash_\mu e:\tau'$ *and* $\Delta;\Gamma \vdash_\mu e':\tau$ *then* $\Delta;\Gamma \vdash_\mu e[e'/x]:\tau'$.

Proof. By induction on the derivations. Trivial. ■

Theorem 5 ($\mathbf{F}_\mu$ Subject Reduction).
If $\Delta;\Gamma \vdash_\mu e:\tau$ *and* $e \hookrightarrow e'$ *then* $\Delta;\Gamma \vdash_\mu e':\tau$.

Proof. Let $\mathcal{D}$ be the derivation of $\Delta;\Gamma \vdash_\mu e:\tau$.

The proof is by induction on the derivation of $e \hookrightarrow e'$.

case (R-APP). $(\lambda x{:}\tau.\, e)\ e' \ \hookrightarrow\ e[e'/x]$.

Derivation $\mathcal{D}$ has the shape

$$\dfrac{\dfrac{\Delta;\Gamma,\, x:\tau \vdash_\mu e:\tau'}{\Delta;\Gamma \vdash_\mu \lambda x:\tau.\, e:\tau\to\tau'} \qquad \Delta;\Gamma \vdash_\mu e':\tau}{\Delta;\Gamma \vdash_\mu (\lambda x:\tau.\, e)\ e':\tau'}.$$

By Lemma 4 it follows that $\Delta;\Gamma \vdash_\mu e[e'/x]:\tau'$.

case (R-TAPP). $(\Lambda\alpha.\, e)\,[\tau] \ \hookrightarrow\ e[\tau/\alpha]$.

Derivation $\mathcal{D}$ has the shape

$$\dfrac{\dfrac{\Delta,\alpha;\Gamma \vdash_\mu e:\tau'}{\Delta;\Gamma \vdash_\mu \Lambda\alpha.\, e:\Pi\alpha.\,\tau'} \qquad \Delta \vdash_\mu \tau}{\Delta;\Gamma \vdash_\mu (\Lambda\alpha.\, e)\,[\tau]:\tau'[\tau/\alpha]}.$$

By Lemma 4 it follows that $\Delta;\Gamma[\tau/\alpha] \vdash_\mu e[\tau/\alpha]:\tau'[\tau/\alpha]$.
Since $\alpha \notin FTV(\Gamma)$, it follows that $\Delta;\Gamma \vdash_\mu e[\tau/\alpha]:\tau'[\tau/\alpha]$.

case (R-LAM). $$\frac{e \;\hookrightarrow\; e'}{\lambda x{:}\tau.\, e \;\hookrightarrow\; \lambda x{:}\tau.\, e'}\ .$$

Derivation $\mathcal{D}$ has the shape $\dfrac{\Delta;\Gamma, x{:}\tau \vdash_\mu e:\tau'}{\Delta;\Gamma \vdash_\mu \lambda x{:}\tau.\, e:\tau \to \tau'}$.

By induction hypothesis it follows that $\Delta;\Gamma, x{:}\tau \vdash_\mu e':\tau'$, and thus

$$\frac{\Delta;\Gamma, x{:}\tau \vdash_\mu e':\tau'}{\Delta;\Gamma \vdash_\mu \lambda x{:}\tau.\, e':\tau \to \tau'}\ .$$

case (R-APP1). $$\frac{e \;\hookrightarrow\; e'}{e\ e'' \;\hookrightarrow\; e'\ e''}\ .$$

Derivation $\mathcal{D}$ has the shape $\dfrac{\Delta;\Gamma \vdash_\mu e:\tau \to \tau' \qquad \Delta;\Gamma \vdash_\mu e'':\tau}{\Delta;\Gamma \vdash_\mu e\ e'':\tau'}$.

By induction hypothesis it follows that $\Delta;\Gamma \vdash_\mu e':\tau \to \tau'$, and thus

$$\frac{\Delta;\Gamma \vdash_\mu e':\tau \to \tau' \qquad \Delta;\Gamma \vdash_\mu e'':\tau}{\Delta;\Gamma \vdash_\mu e'\ e'':\tau'}\ .$$

case (R-APP2). $$\frac{e \;\hookrightarrow\; e'}{v\ e \;\hookrightarrow\; v\ e'}\ .$$

Derivation $\mathcal{D}$ has the shape $\dfrac{\Delta;\Gamma \vdash_\mu v:\tau \to \tau' \qquad \Delta;\Gamma \vdash_\mu e:\tau}{\Delta;\Gamma \vdash_\mu v\ e:\tau'}$.

By induction hypothesis it follows that $\Delta;\Gamma \vdash_\mu e':\tau$, and thus

$$\frac{\Delta;\Gamma \vdash_\mu v:\tau \to \tau' \qquad \Delta;\Gamma \vdash_\mu e':\tau}{\Delta;\Gamma \vdash_\mu v\ e':\tau'}\ .$$

case (R-TLAM). $$\frac{e \;\hookrightarrow\; e'}{\Lambda\alpha.\, e \;\hookrightarrow\; \Lambda\alpha.\, e'}\ .$$

Derivation $\mathcal{D}$ has the shape $\dfrac{\Delta, \alpha;\Gamma \vdash_\mu e:\tau}{\Delta;\Gamma \vdash_\mu \Lambda\alpha.\, e:\Pi\alpha.\, \tau}$.

By induction hypothesis it follows that $\Delta, \alpha;\Gamma \vdash_\mu e':\tau$, and thus

$$\frac{\Delta, \alpha;\Gamma \vdash_\mu e':\tau}{\Delta;\Gamma \vdash_\mu \Lambda\alpha.\, e':\Pi\alpha.\, \tau}\ .$$

case (R-TAPP1). $$\frac{e \;\hookrightarrow\; e'}{e\,[\tau] \;\hookrightarrow\; e'\,[\tau]}\ .$$

Derivation $\mathcal{D}$ has the shape $\dfrac{\Delta;\Gamma \vdash_\mu e : \Pi\alpha.\,\tau' \qquad \Delta \vdash_\mu \tau}{\Delta;\Gamma \vdash_\mu e\,[\tau] : \tau'[\tau/\alpha]}$.

By induction hypothesis it follows that $\Delta;\Gamma \vdash_\mu e' : \Pi\alpha.\,\tau'$, and thus

$$\frac{\Delta;\Gamma \vdash_\mu e' : \Pi\alpha.\,\tau' \qquad \Delta \vdash_\mu \tau}{\Delta;\Gamma \vdash_\mu e'\,[\tau] : \tau'[\tau/\alpha]}\ .$$

case (R-UNFOLD). $\mathsf{unfold}_{\tau_0}\ (\mathsf{fold}_{\tau_1}\ e)\ (\mathsf{fold}_{\tau_1\to\tau_2}\ e') \;\hookrightarrow\; \mathsf{fold}_{\tau_2}\ (e'\ e)$.

Derivation $\mathcal{D}$ has the shape

$$\frac{\dfrac{\Delta;\Gamma \vdash_\mu e : \tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma \vdash_\mu \mathsf{fold}_{\tau_1}\ e : \mu\alpha.\,\tau} \qquad \dfrac{\Delta;\Gamma \vdash_\mu e' : \tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha']}{\Delta;\Gamma \vdash_\mu \mathsf{fold}_{\tau_1\to\tau_2}\ e' : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])}}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\tau_0}\ (\mathsf{fold}_{\tau_1}\ e)\ (\mathsf{fold}_{\tau_1\to\tau_2}\ e') : \mu\alpha'.\,\tau'}$$

where $\tau_0 = \mu\alpha'.\,\tau'$, $\tau_1 = \tau[\mu\alpha.\,\tau/\alpha]$ and $\tau_2 = \tau'[\mu\alpha'.\,\tau'/\alpha']$.

It follows that

$$\frac{\dfrac{\Delta;\Gamma \vdash_\mu e' : \tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'] \qquad \Delta;\Gamma \vdash_\mu e : \tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma \vdash_\mu e'\ e : \tau'[\mu\alpha'.\,\tau'/\alpha']}}{\Delta;\Gamma \vdash_\mu \mathsf{fold}_{\tau'[\mu\alpha'.\,\tau'/\alpha']}\ (e'\ e) : \mu\alpha'.\,\tau'}\ .$$

case (R-FOLD). $$\frac{e \;\hookrightarrow\; e'}{\mathsf{fold}_\tau\ e \;\hookrightarrow\; \mathsf{fold}_\tau\ e'}\ .$$

Derivation $\mathcal{D}$ has the shape (where $\tau = \tau'[\mu\alpha.\,\tau'/\alpha]$)

$$\frac{\Delta;\Gamma \vdash_\mu e : \tau'[\mu\alpha.\,\tau'/\alpha]}{\Delta;\Gamma \vdash_\mu \mathsf{fold}_{\tau'[\mu\alpha.\,\tau'/\alpha]}\ e : \mu\alpha.\,\tau'}\ .$$

By induction hypothesis it follows that $\Delta;\Gamma \vdash_\mu e' : \tau'[\mu\alpha.\,\tau'/\alpha]$, and thus

$$\frac{\Delta;\Gamma \vdash_\mu e' : \tau'[\mu\alpha.\,\tau'/\alpha]}{\Delta;\Gamma \vdash_\mu \mathsf{fold}_{\tau'[\mu\alpha.\,\tau'/\alpha]}\ e' : \mu\alpha.\,\tau'}\ .$$

case (R-UNFOLD1). $$\frac{e \;\hookrightarrow\; e'}{\mathsf{unfold}_{\tau_0}\ e''\ e \;\hookrightarrow\; \mathsf{unfold}_{\tau_0}\ e''\ e'}\ .$$

Derivation $\mathcal{D}$ has the shape (where $\tau_0 = \mu\alpha'.\,\tau'$)

$$\frac{\Delta;\Gamma \vdash_\mu e'' : \mu\alpha.\,\tau \qquad \Delta;\Gamma \vdash_\mu e : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e''\ e : \mu\alpha'.\,\tau'}\ .$$

By induction hypothesis it follows that

$$\Delta; \Gamma \vdash_\mu e' : \mu\_.\, (\tau[\mu\alpha.\, \tau/\alpha] \to \tau'[\mu\alpha'.\, \tau'/\alpha'])$$

and thus

$$\frac{\Delta; \Gamma \vdash_\mu e'' : \mu\alpha.\, \tau \qquad \Delta; \Gamma \vdash_\mu e' : \mu\_.\, (\tau[\mu\alpha.\, \tau/\alpha] \to \tau'[\mu\alpha'.\, \tau'/\alpha'])}{\Delta; \Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\, \tau'}\ e''\ e' : \mu\alpha'.\, \tau'}\ .$$

case (R-UNFOLD2). $$\frac{e \ \hookrightarrow\ e'}{\mathsf{unfold}_{\tau_0}\ e\ v \ \hookrightarrow\ \mathsf{unfold}_{\tau_0}\ e'\ v}\ .$$

Derivation $\mathcal{D}$ has the shape (where $\tau_0 = \mu\alpha'.\, \tau'$)

$$\frac{\Delta; \Gamma \vdash_\mu e : \mu\alpha.\, \tau \qquad \Delta; \Gamma \vdash_\mu v : \mu\_.\, (\tau[\mu\alpha.\, \tau/\alpha] \to \tau'[\mu\alpha'.\, \tau'/\alpha'])}{\Delta; \Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\, \tau'}\ e\ v : \mu\alpha'.\, \tau'}\ .$$

By induction hypothesis it follows that $\Delta; \Gamma \vdash_\mu e' : \mu\alpha.\, \tau$, and thus

$$\frac{\Delta; \Gamma \vdash_\mu e' : \mu\alpha.\, \tau \qquad \Delta; \Gamma \vdash_\mu v : \mu\_.\, (\tau[\mu\alpha.\, \tau/\alpha] \to \tau'[\mu\alpha'.\, \tau'/\alpha'])}{\Delta; \Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\, \tau'}\ e'\ v : \mu\alpha'.\, \tau'}\ . \quad \blacksquare$$

Theorem 6 ($\mathbf{F}_\mu$ Progress).
If $\cdot; \cdot \vdash_\mu e : \tau$ *then either* $e = v$ *or* $e \hookrightarrow e'$.

Proof. By induction on the derivation of $\cdot; \cdot \vdash_\mu e : \tau$.
case (VAR). Trivial by contradiction.

case (LAM). $$\frac{\cdots}{\cdot; \cdot \vdash_\mu \lambda x : \tau.\, e : \tau \to \tau'}\ .$$

$v = \lambda x : \tau.\, e$.

case (APP). $$\frac{\cdot; \cdot \vdash_\mu e : \tau \to \tau' \qquad \cdot; \cdot \vdash_\mu e' : \tau}{\cdot; \cdot \vdash_\mu e\ e' : \tau'}\ .$$

By induction hypothesis on $\cdot; \cdot \vdash_\mu e : \tau \to \tau'$ there are two possible cases.
case $e = v$. By $\cdot; \cdot \vdash_\mu v : \tau \to \tau'$ it follows that $v = \lambda x : \tau.\, e''$, and thus

$$e\ e' = v\ e' = (\lambda x : \tau.\, e'')\ e' \ \hookrightarrow\ e''[e'/x].$$

case $e \hookrightarrow e''$. It follows that $$\frac{e \ \hookrightarrow\ e''}{e\ e' \ \hookrightarrow\ e''\ e'}\ .$$

case (TLAM). $$\frac{\cdots}{\cdot; \cdot \vdash_\mu \Lambda\alpha.\, e : \Pi\alpha.\, \tau}\ .$$

$v = \Lambda\alpha.\, e$.

case (TAPP). $$\frac{\cdot;\cdot \vdash_\mu e : \Pi\alpha.\,\tau \qquad \cdot \vdash_\mu \tau'}{\cdot;\cdot \vdash_\mu e\,[\tau'] : \tau[\tau'/\alpha]}\ .$$

By induction hypothesis on $\cdot;\cdot \vdash_\mu e : \Pi\alpha.\,\tau$ there are two possible cases.

case $e = v$. By $\cdot;\cdot \vdash_\mu v : \Pi\alpha.\,\tau$ it follows that $v = \Lambda\alpha.\,e'$, and thus

$$e\,[\tau'] = v\,[\tau'] = (\Lambda\alpha.\,e')\,[\tau'] \ \hookrightarrow\ e'[\tau'/\alpha].$$

case $e \hookrightarrow e'$. It follows that $$\frac{e \ \hookrightarrow\ e'}{e\,[\tau'] \ \hookrightarrow\ e'\,[\tau']}\ .$$

case (FOLD). $$\frac{\cdots}{\cdot;\cdot \vdash_\mu \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e : \mu\alpha.\,\tau}\ .$$

$v = \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e$.

case (UNFOLD).

$$\frac{\cdot;\cdot \vdash_\mu e : \mu\alpha.\,\tau \qquad \cdot;\cdot \vdash_\mu e' : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])}{\cdot;\cdot \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e' : \mu\alpha'.\,\tau'}\ .$$

By induction hypothesis on $\cdot;\cdot \vdash_\mu e' : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])$ there are two sub-cases.

case $e' = v$.

By induction hypothesis on $\cdot;\cdot \vdash_\mu e : \mu\alpha.\,\tau$ there are again two sub-cases.

case $e = v'$.

By $\cdot;\cdot \vdash_\mu v : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])$ it follows that

$$v = \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha']}\ e''.$$

By $\cdot;\cdot \vdash_\mu v' : \mu\alpha.\,\tau$ it follows that $v' = \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e'''$, and thus

$$\begin{aligned} & \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e' \\ =\ & \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ (\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e''')\ (\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha']}\ e'') \\ \hookrightarrow\ & \mathsf{fold}_{\tau'[\mu\alpha'.\,\tau'/\alpha']}\ (e''\ e''').\end{aligned}$$

case $e \hookrightarrow e''$. It follows that $$\frac{e \ \hookrightarrow\ e''}{\mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ v \ \hookrightarrow\ \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e''\ v}\ .$$

Then it follows that $\mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e' \ \hookrightarrow\ \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e''\ e'$.

case $e' \hookrightarrow e''$. It follows that $$\frac{e \ \hookrightarrow\ e''}{\mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e' \ \hookrightarrow\ \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e''}\ .$$ ■

Lemma 7 (Translation Substituion Preservation).

1. $\ulcorner \tau[\tau'/\alpha] \urcorner = \ulcorner \tau \urcorner [\ulcorner \tau' \urcorner / \alpha]$;
2. $\ulcorner e[\tau/\alpha] \urcorner = \ulcorner e \urcorner [\ulcorner \tau \urcorner / \alpha]$;
3. $\ulcorner e[e'/x] \urcorner = \ulcorner e \urcorner [\ulcorner e' \urcorner / x]$.

Proof. By induction on the structure of τ and e. Trivial. ■

Theorem 8 (Translation Type Wellformedness Preservation).
If $\Delta \vdash_\mu \tau$ *then* $\Delta \vdash \ulcorner \tau \urcorner$.

Proof. By induction on the structure of τ. Trivial. ■

Theorem 9 (Translation Typing Preservation).
If $\Delta; \Gamma \vdash_\mu e : \tau$ *then* $\Delta; \ulcorner \Gamma \urcorner \vdash \ulcorner e \urcorner : \ulcorner \tau \urcorner$.

Proof. By induction on the derivation of $\Delta; \Gamma \vdash_\mu e : \tau$.

case (VAR). $$\frac{\Delta \vdash_\mu \Gamma(x)}{\Delta; \Gamma \vdash_\mu x : \Gamma(x)} \ .$$

By induction hypothesis it follows that $\Delta \vdash \ulcorner \Gamma(x) \urcorner$.
By the definition of $\ulcorner \cdot \urcorner$ it follows that $\Delta \vdash \ulcorner \Gamma \urcorner(x)$ and thus

$$\frac{\Delta \vdash \ulcorner \Gamma \urcorner(x)}{\Delta; \ulcorner \Gamma \urcorner \vdash x : \ulcorner \Gamma \urcorner(x)} \ .$$

By the definition of $\ulcorner \cdot \urcorner$ it follows that $\Delta; \ulcorner \Gamma \urcorner \vdash x : \ulcorner \Gamma(x) \urcorner$.

case (LAM). $$\frac{\Delta; \Gamma, x : \tau \vdash_\mu e : \tau'}{\Delta; \Gamma \vdash_\mu \lambda x : \tau.\, e : \tau \to \tau'} \ .$$

By induction hypothesis it follows that $\Delta; \ulcorner \Gamma, x : \tau \urcorner \vdash \ulcorner e \urcorner : \ulcorner \tau' \urcorner$.
By the definition of $\ulcorner \cdot \urcorner$ it follows that $\Delta; \ulcorner \Gamma \urcorner, x : \ulcorner \tau \urcorner \vdash \ulcorner e \urcorner : \ulcorner \tau' \urcorner$ and thus

$$\frac{\Delta; \ulcorner \Gamma \urcorner, x : \ulcorner \tau \urcorner \vdash \ulcorner e \urcorner : \ulcorner \tau' \urcorner}{\Delta; \ulcorner \Gamma \urcorner \vdash \lambda x : \ulcorner \tau \urcorner.\, \ulcorner e \urcorner : \ulcorner \tau \urcorner \to \ulcorner \tau' \urcorner} \ .$$

It then follows that $\Delta; \ulcorner \Gamma \urcorner \vdash \ulcorner \lambda x : \tau.\, e \urcorner : \ulcorner \tau \to \tau' \urcorner$.

case (APP). $$\frac{\Delta; \Gamma \vdash_\mu e : \tau \to \tau' \qquad \Delta; \Gamma \vdash_\mu e' : \tau}{\Delta; \Gamma \vdash_\mu e\ e' : \tau'} \ .$$

By induction hypothesis it follows that $\Delta; \ulcorner \Gamma \urcorner \vdash \ulcorner e' \urcorner : \ulcorner \tau \urcorner$.
Depending on the shape of e, there are two sub-cases.

case $e = \lambda x : \tau.\, \mathsf{fold}_\tau\ x$.

By $\Delta;\Gamma \vdash_\mu \lambda x{:}\tau.\,\mathsf{fold}_\tau\ x{:}\tau\to\tau'$ it follows that τ' has the shape $\mu\alpha.\,\tau''$.

We have $$\frac{\Delta;\ulcorner\Gamma\urcorner, x{:}\Pi\alpha.\,\alpha\to\alpha \vdash x{:}\Pi\alpha.\,\alpha\to\alpha}{\Delta;\ulcorner\Gamma\urcorner \vdash \lambda x{:}\Pi\alpha.\,\alpha\to\alpha.\,x{:}(\Pi\alpha.\,\alpha\to\alpha)\to\Pi\alpha.\,\alpha\to\alpha}$$

and $$\frac{\dfrac{\dfrac{\Delta,\alpha\vdash\alpha}{\Delta,\alpha;\ulcorner\Gamma\urcorner, x{:}\alpha\vdash x{:}\alpha}}{\Delta,\alpha;\ulcorner\Gamma\urcorner\vdash\lambda x{:}\alpha.\,x{:}\alpha\to\alpha}}{\Delta;\ulcorner\Gamma\urcorner\vdash\Lambda\alpha.\,\lambda x{:}\alpha.\,x{:}\Pi\alpha.\,\alpha\to\alpha}\ .$$

It then follows that

$$\frac{\begin{array}{l}\Delta;\ulcorner\Gamma\urcorner\vdash\lambda x{:}\Pi\alpha.\,\alpha\to\alpha.\,x{:}(\Pi\alpha.\,\alpha\to\alpha)\to\Pi\alpha.\,\alpha\to\alpha\\ \Delta;\ulcorner\Gamma\urcorner\vdash\Lambda\alpha.\,\lambda x{:}\alpha.\,x{:}\Pi\alpha.\,\alpha\to\alpha\end{array}}{\Delta;\ulcorner\Gamma\urcorner\vdash(\lambda x{:}\Pi\alpha.\,\alpha\to\alpha.\,x)\ \Lambda\alpha.\,\lambda x{:}\alpha.\,x{:}\Pi\alpha.\,\alpha\to\alpha}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner(\lambda x{:}\tau.\,\mathsf{fold}_\tau\ x)\ e'\urcorner{:}\ulcorner\tau'\urcorner$.

$\mathsf{case}\ e\neq\lambda x{:}\tau.\,\mathsf{fold}_\tau\ x$.

By induction hypothesis it follows that $\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner{:}\ulcorner\tau\to\tau'\urcorner$.
It follows that $\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner{:}\ulcorner\tau\urcorner\to\ulcorner\tau'\urcorner$ and thus

$$\frac{\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner{:}\ulcorner\tau\urcorner\to\ulcorner\tau'\urcorner\qquad\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e'\urcorner{:}\ulcorner\tau\urcorner}{\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner\ulcorner e'\urcorner{:}\ulcorner\tau'\urcorner}\ .$$

It then follows that $\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\ e'\urcorner{:}\ulcorner\tau'\urcorner$.

case (TLAM). $$\frac{\Delta,\alpha;\Gamma\vdash_\mu e{:}\tau}{\Delta;\Gamma\vdash_\mu\Lambda\alpha.\,e{:}\Pi\alpha.\,\tau}\ .$$

By induction hypothesis it follows that $\Delta,\alpha;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner{:}\ulcorner\tau\urcorner$, and thus

$$\frac{\Delta,\alpha;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner{:}\ulcorner\tau\urcorner}{\Delta;\ulcorner\Gamma\urcorner\vdash\Lambda\alpha.\ulcorner e\urcorner{:}\Pi\alpha.\ulcorner\tau\urcorner}\ .$$

It then follows that $\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner\Lambda\alpha.\,e\urcorner{:}\ulcorner\Pi\alpha.\,\tau\urcorner$.

case (TAPP). $$\frac{\Delta;\Gamma\vdash_\mu e{:}\Pi\alpha.\,\tau\qquad\Delta\vdash_\mu\tau'}{\Delta;\Gamma\vdash_\mu e\,[\tau']{:}\tau[\tau'/\alpha]}\ .$$

By induction hypothesis it follows that $\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner{:}\ulcorner\Pi\alpha.\,\tau\urcorner$.
It follows that $\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner{:}\Pi\alpha.\ulcorner\tau\urcorner$.
By induction hypothesis it follows that $\Delta\vdash\ulcorner\tau'\urcorner$, and thus

$$\frac{\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner{:}\Pi\alpha.\ulcorner\tau\urcorner\qquad\Delta\vdash\ulcorner\tau'\urcorner}{\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\urcorner[\ulcorner\tau'\urcorner]{:}\ulcorner\tau\urcorner[\ulcorner\tau'\urcorner/\alpha]}\ .$$

By Lemma 7 it follows that $\Delta;\ulcorner\Gamma\urcorner\vdash\ulcorner e\,[\tau']\urcorner{:}\ulcorner\tau[\tau'/\alpha]\urcorner$.

case (FOLD). $$\frac{\Delta;\Gamma \vdash_\mu e : \tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma \vdash_\mu \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e : \mu\alpha.\,\tau}\ .$$

We have $$\frac{\dfrac{\dfrac{\Delta,\alpha \vdash \alpha}{\Delta,\alpha;\ulcorner\Gamma\urcorner, x:\alpha \vdash x:\alpha}}{\Delta,\alpha;\ulcorner\Gamma\urcorner \vdash \lambda x:\alpha.\,x:\alpha\to\alpha}}{\Delta;\ulcorner\Gamma\urcorner \vdash \Lambda\alpha.\,\lambda x:\alpha.\,x:\Pi\alpha.\,\alpha\to\alpha}\ .$$

It follows that $\Delta;\ulcorner\Gamma\urcorner \vdash \ulcorner\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e\urcorner : \ulcorner\mu\alpha.\,\tau\urcorner$.

case (UNFOLD).

$$\frac{\Delta;\Gamma \vdash_\mu e : \mu\alpha.\,\tau \qquad \Delta;\Gamma \vdash_\mu e' : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e' : \mu\alpha'.\,\tau'}\ .$$

By induction hypothesis it follows that $\Delta;\ulcorner\Gamma\urcorner \vdash \ulcorner e\urcorner : \ulcorner\mu\alpha.\,\tau\urcorner$.
It follows that $\Delta;\ulcorner\Gamma\urcorner \vdash \ulcorner e\urcorner : \Pi\alpha.\,\alpha\to\alpha$.
By induction hypothesis it follows that

$$\Delta;\ulcorner\Gamma\urcorner \vdash \ulcorner e'\urcorner : \ulcorner\mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])\urcorner.$$

It follows that $\Delta;\ulcorner\Gamma\urcorner \vdash \ulcorner e'\urcorner : \Pi\alpha.\,\alpha\to\alpha$, and thus

$$\frac{\dfrac{\dfrac{\Delta;\ulcorner\Gamma\urcorner \vdash \ulcorner e'\urcorner : \Pi\alpha.\,\alpha\to\alpha}{\Delta;\ulcorner\Gamma\urcorner, \_ : \Pi\alpha.\,\alpha\to\alpha \vdash \ulcorner e'\urcorner : \Pi\alpha.\,\alpha\to\alpha}}{\Delta;\ulcorner\Gamma\urcorner \vdash \lambda\_ : \Pi\alpha.\,\alpha\to\alpha.\ulcorner e'\urcorner : (\Pi\alpha.\,\alpha\to\alpha)\to(\Pi\alpha.\,\alpha\to\alpha)} \qquad \Delta;\ulcorner\Gamma\urcorner \vdash \ulcorner e\urcorner : \Pi\alpha.\,\alpha\to\alpha}{\Delta;\ulcorner\Gamma\urcorner \vdash (\lambda\_ : \Pi\alpha.\,\alpha\to\alpha.\ulcorner e'\urcorner)\ \ulcorner e\urcorner : \Pi\alpha.\,\alpha\to\alpha}\ .$$

It then follows that $\Delta;\ulcorner\Gamma\urcorner \vdash \ulcorner\mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e'\urcorner : \ulcorner\mu\alpha'.\,\tau'\urcorner$. ■

Theorem 10 (Translation Reduction Preservation).
If $e \rightharpoonup e'$ then $\ulcorner e\urcorner \hookrightarrow^+ \ulcorner e'\urcorner$.

Proof. By induction on the derivation of $e \rightharpoonup e'$.

case (R-APP). $$(\lambda x:\tau.\,e)\ e' \ \rightharpoonup\ e[e'/x].$$

Depending on the shape of e, there are two sub-cases.

case $e = \mathsf{fold}_\tau\ x$.
By the definition of $\ulcorner\cdot\urcorner$ and Lemma 7 it follows that

$$\ulcorner(\lambda x:\tau.\,\mathsf{fold}_\tau\ x)\ e'\urcorner = (\lambda x:\Pi\alpha.\,\alpha\to\alpha.\,x)\ \Lambda\alpha.\,\lambda x:\alpha.\,x$$
$$\hookrightarrow \Lambda\alpha.\,\lambda x:\alpha.\,x = (\Lambda\alpha.\,\lambda x:\alpha.\,x)[\ulcorner e'\urcorner/x] = \ulcorner\mathsf{fold}_\tau\ x\urcorner[\ulcorner e'\urcorner/x] = \ulcorner(\mathsf{fold}_\tau\ x)[e'/x]\urcorner.$$

case $e \neq \mathsf{fold}_\tau\ x$.
By the definition of $\ulcorner\cdot\urcorner$ and Lemma 7 it follows that

$$\ulcorner(\lambda x{:}\tau.\, e)\ e'\urcorner = (\lambda x{:}\ulcorner\tau\urcorner.\ulcorner e\urcorner)\ \ulcorner e'\urcorner \quad \hookrightarrow\ \ulcorner e\urcorner[\ulcorner e'\urcorner/x] = \ulcorner e[e'/x]\urcorner.$$

case (R-TAPP). $$(\Lambda\alpha.\, e)\,[\tau] \ \rightharpoonup\ e[\tau/\alpha].$$

By the definition of $\ulcorner\cdot\urcorner$ and Lemma 7 it follows that

$$\ulcorner(\Lambda\alpha.\, e)\,[\tau]\urcorner = (\Lambda\alpha.\ulcorner e\urcorner)\,[\ulcorner\tau\urcorner] \quad \hookrightarrow\ \ulcorner e\urcorner[\ulcorner\tau\urcorner/\alpha] = \ulcorner e[\tau/\alpha]\urcorner.$$

case (R-LAM). $$\frac{e \ \rightharpoonup\ e'}{\lambda x{:}\tau.\, e \ \rightharpoonup\ \lambda x{:}\tau.\, e'}\ .$$

By induction hypothesis it follows that $\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner$. It then follows that

$$\frac{\ulcorner e\urcorner \ \hookrightarrow\ e_1 \ \hookrightarrow\ \ldots \ \hookrightarrow\ e_n \ \hookrightarrow\ \ulcorner e'\urcorner}{\lambda x{:}\ulcorner\tau\urcorner.\ulcorner e\urcorner \ \hookrightarrow\ \lambda x{:}\ulcorner\tau\urcorner.\, e_1 \ \hookrightarrow\ \ldots \ \hookrightarrow\ \lambda x{:}\ulcorner\tau\urcorner.\, e_n \ \hookrightarrow\ \lambda x{:}\ulcorner\tau\urcorner.\ulcorner e'\urcorner}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\ulcorner\lambda x{:}\tau.\, e\urcorner \ \hookrightarrow^+\ \ulcorner\lambda x{:}\tau.\, e'\urcorner$.

case (R-APP1). $$\frac{e \ \rightharpoonup\ e'}{e\ e'' \ \rightharpoonup\ e'\ e''}\ .$$

By induction hypothesis it follows that $\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner$. It then follows that

$$\frac{\ulcorner e\urcorner \ \hookrightarrow\ e_1 \ \hookrightarrow\ \ldots \ \hookrightarrow\ e_n \ \hookrightarrow\ \ulcorner e'\urcorner}{\ulcorner e\urcorner\ulcorner e''\urcorner \ \hookrightarrow\ e_1\ulcorner e''\urcorner \ \hookrightarrow\ \ldots \ \hookrightarrow\ e_n\ulcorner e''\urcorner \ \hookrightarrow\ \ulcorner e'\urcorner\ulcorner e''\urcorner}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\ulcorner e\ e''\urcorner \ \hookrightarrow^+\ \ulcorner e'\ e''\urcorner$.

case (R-APP2). $$\frac{e \ \rightharpoonup\ e'}{v\ e \ \rightharpoonup\ v\ e'}\ .$$

By induction hypothesis it follows that $\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner$. It then follows that

$$\frac{\ulcorner e\urcorner \ \hookrightarrow\ e_1 \ \hookrightarrow\ \ldots \ \hookrightarrow\ e_n \ \hookrightarrow\ \ulcorner e'\urcorner}{\ulcorner v\urcorner\ulcorner e\urcorner \ \hookrightarrow\ \ulcorner v\urcorner\, e_1 \ \hookrightarrow\ \ldots \ \hookrightarrow\ \ulcorner v\urcorner\, e_n \ \hookrightarrow\ \ulcorner v\urcorner\ulcorner e'\urcorner}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\ulcorner v\ e\urcorner \ \hookrightarrow^+\ \ulcorner v\ e'\urcorner$.

case (R-TLAM). $$\frac{e \ \rightharpoonup\ e'}{\Lambda\alpha.\, e \ \rightharpoonup\ \Lambda\alpha.\, e'}\ .$$

By induction hypothesis it follows that $\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner$. It then follows that

$$\frac{\ulcorner e\urcorner \ \hookrightarrow\ e_1 \ \hookrightarrow\ \ldots \ \hookrightarrow\ e_n \ \hookrightarrow\ \ulcorner e'\urcorner}{\Lambda\alpha.\ulcorner e\urcorner \ \hookrightarrow\ \Lambda\alpha.\, e_1 \ \hookrightarrow\ \ldots \ \hookrightarrow\ \Lambda\alpha.\, e_n \ \hookrightarrow\ \Lambda\alpha.\ulcorner e'\urcorner}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\ulcorner \Lambda\alpha.\, e\urcorner \hookrightarrow^+ \ulcorner \Lambda\alpha.\, e'\urcorner$.

case (R-TAPP1). $$\frac{e \rightharpoonup e'}{e\,[\tau] \rightharpoonup e'\,[\tau]}\ .$$

By induction hypothesis it follows that $\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner$.
It then follows that

$$\frac{\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner}{\ulcorner e\urcorner[\ulcorner\tau\urcorner] \hookrightarrow e_1\,[\ulcorner\tau\urcorner] \hookrightarrow \ldots \hookrightarrow e_n\,[\ulcorner\tau\urcorner] \hookrightarrow \ulcorner e'\urcorner[\ulcorner\tau\urcorner]}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\ulcorner e\,[\tau]\urcorner \hookrightarrow^+ \ulcorner e'\,[\tau]\urcorner$.

case (R-UNFOLD). $\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_{\tau}\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e') \rightharpoonup \mathsf{fold}_{\tau'}\ (e'\ e)$.

By the definition of $\ulcorner\cdot\urcorner$ it follows that

$\ulcorner\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_{\tau}\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e')\urcorner = (\lambda\_{:}\Pi\alpha.\,\alpha\to\alpha.\ulcorner\mathsf{fold}_{\tau\to\tau'}\ e'\urcorner)\ \ulcorner\mathsf{fold}_{\tau}\ e\urcorner$
$= (\lambda\_{:}\Pi\alpha.\,\alpha\to\alpha.\,\Lambda\alpha.\,\lambda x{:}\alpha.\,x)\ (\Lambda\alpha.\,\lambda x{:}\alpha.\,x) \hookrightarrow \Lambda\alpha.\,\lambda x{:}\alpha.\,x = \ulcorner\mathsf{fold}_{\tau'}\ (e'\ e)\urcorner$.

case (R-UNFOLD1). $$\frac{e \rightharpoonup e'}{\mathsf{unfold}_{\tau}\ e''\ e \rightharpoonup \mathsf{unfold}_{\tau}\ e''\ e'}\ .$$

By induction hypothesis it follows that $\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner$.
It then follows that

$$\frac{\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner}{\mathsf{unfold}_{\tau}\ulcorner e''\urcorner\ulcorner e\urcorner \hookrightarrow \mathsf{unfold}_{\tau}\ulcorner e''\urcorner\ e_1 \hookrightarrow \ldots \hookrightarrow \mathsf{unfold}_{\tau}\ulcorner e''\urcorner\ e_n \hookrightarrow \mathsf{unfold}_{\tau}\ulcorner e''\urcorner\ulcorner e'\urcorner}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\ulcorner\mathsf{unfold}_{\tau}\ e''\ e\urcorner \hookrightarrow^+ \ulcorner\mathsf{unfold}_{\tau}\ e''\ e'\urcorner$.

case (R-UNFOLD2). $$\frac{e \rightharpoonup e'}{\mathsf{unfold}_{\tau}\ e\ v \rightharpoonup \mathsf{unfold}_{\tau}\ e'\ v}\ .$$

By induction hypothesis it follows that $\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner$.
It then follows that

$$\frac{\ulcorner e\urcorner \hookrightarrow e_1 \hookrightarrow \ldots \hookrightarrow e_n \hookrightarrow \ulcorner e'\urcorner}{\mathsf{unfold}_{\tau}\ulcorner e\urcorner\ulcorner v\urcorner \hookrightarrow \mathsf{unfold}_{\tau}e_1\ulcorner v\urcorner \hookrightarrow \ldots \hookrightarrow \mathsf{unfold}_{\tau}e_n\ulcorner v\urcorner \hookrightarrow \mathsf{unfold}_{\tau}\ulcorner e'\urcorner\ulcorner v\urcorner}\ .$$

By the definition of $\ulcorner\cdot\urcorner$ it follows that $\ulcorner\mathsf{unfold}_{\tau}\ e\ v\urcorner \hookrightarrow^+ \ulcorner\mathsf{unfold}_{\tau}\ e'\ v\urcorner$. ■

Theorem 11 ($\mathbf{F}_\mu$ Strong Normalization ($\rightharpoonup$)).
If $\cdot;\cdot \vdash_\mu e:\tau$ then the length of any reduction sequence starting with e, without using reduction rule (R-FOLD)*, is finite.*

Lemma 12 ((R-FOLD1) **Reduction Steps Finite).**
The length of any $\rightsquigarrow$ reduction sequences that uses rule (R-FOLD1) *in every step is finite.*

Lemma 13 ((R-FOLD1) **Translation Equivalence).**
If the derivation of $e \rightsquigarrow e'$ *uses reduction rule* (R-FOLD1) *then* $\ulcorner e \urcorner = \ulcorner e' \urcorner$.

Proof. By induction on the derivation of $e \rightsquigarrow e'$.

case (R-APP). $(\lambda x{:}\tau.\, e)\ e' \ \rightsquigarrow\ e[e'/x]$.

Trivial by contradiction.

case (R-TAPP). $(\Lambda\alpha.\, e)\,[\tau] \ \rightsquigarrow\ e[\tau/\alpha]$.

Trivial by contradiction.

case (R-LAM). $\dfrac{e \ \rightsquigarrow\ e'}{\lambda x{:}\tau.\, e \ \rightsquigarrow\ \lambda x{:}\tau.\, e'}$.

By induction hypothesis it follows that $\ulcorner e \urcorner = \ulcorner e' \urcorner$.
It follows that $\ulcorner \lambda x{:}\tau.\, e \urcorner = \lambda x{:}\tau.\ulcorner e \urcorner = \lambda x{:}\tau.\ulcorner e \urcorner = \ulcorner \lambda x{:}\tau.\, e' \urcorner$.

case (R-APP1). $\dfrac{e \ \rightsquigarrow\ e'}{e\ e'' \ \rightsquigarrow\ e'\ e''}$.

Depending on the shape of e, there are two sub-cases.

case $e = \lambda x{:}\tau.\, \mathsf{fold}_\tau\ x$.
Trivial by contradiction since $(\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)$ is a normal value.

case $e \neq \lambda x{:}\tau.\, \mathsf{fold}_\tau\ x$.
Depending on the shape of e', there are two sub-cases.

case $e' = \lambda x{:}\tau.\, \mathsf{fold}_\tau\ x$.
The derivation of $e \rightsquigarrow e'$ must use rule (R-FOLD1).
By contradiction as e' can not be of form $(\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)$.

case $e' \neq \lambda x{:}\tau.\, \mathsf{fold}_\tau\ x$.
By induction hypothesis it follows that $\ulcorner e \urcorner = \ulcorner e' \urcorner$.
It follows that $\ulcorner e\ e'' \urcorner = \ulcorner e \urcorner\ \ulcorner e'' \urcorner = \ulcorner e' \urcorner\ \ulcorner e'' \urcorner = \ulcorner e'\ e'' \urcorner$.

case (R-APP2). $\dfrac{e \ \rightsquigarrow\ e'}{v\ e \ \rightsquigarrow\ v\ e'}$.

Depending on the shape of v, there are two sub-cases.

case $v = \lambda x{:}\tau.\, \mathsf{fold}_\tau\ x$. It follows that

$$\begin{aligned}\ulcorner (\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)\ e \urcorner &= (\lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x)\ \ulcorner \mathsf{fold}_\tau\ e \urcorner \\ &= (\lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x)\ \Lambda\alpha.\, \lambda x{:}\alpha.\, x \\ &= (\lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x)\ \ulcorner \mathsf{fold}_\tau\ e' \urcorner = \ulcorner (\lambda x{:}\tau.\, \mathsf{fold}_\tau\ x)\ e' \urcorner.\end{aligned}$$

case $v \neq \lambda x{:}\tau.\, \mathsf{fold}_\tau\ x$.
By induction hypothesis it follows that $\ulcorner e \urcorner = \ulcorner e' \urcorner$.
It follows that $\ulcorner v\ e \urcorner = \ulcorner v \urcorner\ \ulcorner e \urcorner = \ulcorner v \urcorner\ \ulcorner e' \urcorner = \ulcorner v\ e' \urcorner$.

case (R-TLAM). $\dfrac{e \rightsquigarrow e'}{\Lambda\alpha.\, e \rightsquigarrow \Lambda\alpha.\, e'}$.

By induction hypothesis it follows that $\ulcorner e \urcorner = \ulcorner e' \urcorner$.
It follows that $\ulcorner \Lambda\alpha.\, e \urcorner = \Lambda\alpha.\, \ulcorner e \urcorner = \Lambda\alpha.\, \ulcorner e' \urcorner = \ulcorner \Lambda\alpha.\, e' \urcorner$.

case (R-TAPP1). $\dfrac{e \rightsquigarrow e'}{e\,[\tau] \rightsquigarrow e'\,[\tau]}$.

By induction hypothesis it follows that $\ulcorner e \urcorner = \ulcorner e' \urcorner$.
It follows that $\ulcorner e\,[\tau] \urcorner = \ulcorner e \urcorner\,[\ulcorner \tau \urcorner] = \ulcorner e' \urcorner\,[\ulcorner \tau \urcorner] = \ulcorner e'\,[\tau] \urcorner$.

case (R-FOLD1).

$$(\lambda x{:}\mu\alpha.\, \tau.\, x)\ (\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e) \hookrightarrow (\lambda x{:}\tau[\mu\alpha.\,\tau/\alpha].\, \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x)\ e.$$

By the definition of $\ulcorner \cdot \urcorner$ it follows that

$$\begin{aligned}
&\ulcorner (\lambda x{:}\mu\alpha.\, \tau.\, x)\ (\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e) \urcorner \\
&= (\lambda x{:}\ulcorner \mu\alpha.\, \tau \urcorner.\, x)\ \ulcorner \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e \urcorner \\
&= (\lambda x{:}\Pi\alpha.\, \alpha \to \alpha.\, x)\ \ulcorner \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e \urcorner \\
&= \ulcorner (\lambda x{:}\tau[\mu\alpha.\,\tau/\alpha].\, \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x)\ e \urcorner.
\end{aligned}$$

case (R-UNFOLD). $\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_{\tau}\ e)\ (\mathsf{fold}_{\tau \to \tau'}\ e') \rightsquigarrow \mathsf{fold}_{\tau'}\ (e'\ e)$.

Trivial by contradiction.

case (R-UNFOLD1). $\dfrac{e \rightsquigarrow e'}{\mathsf{unfold}_{\tau}\ e''\ e \rightsquigarrow \mathsf{unfold}_{\tau}\ e''\ e'}$.

By induction hypothesis it follows that $\ulcorner e \urcorner = \ulcorner e' \urcorner$.
It follows that

$$\begin{aligned}
\ulcorner \mathsf{unfold}_{\tau}\ e''\ e \urcorner &= (\lambda_{:}\Pi\alpha.\, \alpha \to \alpha.\, \ulcorner e \urcorner)\ \ulcorner e'' \urcorner \\
&= (\lambda_{:}\Pi\alpha.\, \alpha \to \alpha.\, \ulcorner e' \urcorner)\ \ulcorner e'' \urcorner = \ulcorner \mathsf{unfold}_{\tau}\ e''\ e' \urcorner.
\end{aligned}$$

case (R-UNFOLD2). $\dfrac{e \rightsquigarrow e'}{\mathsf{unfold}_{\tau}\ e\ v \rightsquigarrow \mathsf{unfold}_{\tau}\ e'\ v}$.

By induction hypothesis it follows that $\ulcorner e \urcorner = \ulcorner e' \urcorner$.
It follows that

$$\begin{aligned}
\ulcorner \mathsf{unfold}_{\tau}\ e\ v \urcorner &= (\lambda_{:}\Pi\alpha.\, \alpha \to \alpha.\, \ulcorner v \urcorner)\ \ulcorner e \urcorner \\
&= (\lambda_{:}\Pi\alpha.\, \alpha \to \alpha.\, \ulcorner v \urcorner)\ \ulcorner e' \urcorner = \ulcorner \mathsf{unfold}_{\tau}\ e'\ v \urcorner.
\end{aligned}$$

■

Theorem 14 ($\mathbf{F}_\mu$ Strong Normalization ($\rightsquigarrow$)).
If $\cdot;\cdot \vdash_\mu e : \tau$ then the length of any reduction sequence starting with e, without using reduction rule (R-FOLD)*, but might using the additional reduction rule* (R-FOLD1)*, is finite.*

Lemma 15 (Coercion Substituion Preservation).

1. $|e[\tau/\alpha]| \;=\; |e|[\tau/\alpha]$;
2. $|e[e'/x]| \;=\; |e|[|e'|/x]$.

Proof. By induction on the structure of e.

case x'. $\quad |x'[\tau/\alpha]| \;=\; |x'| \;=\; x' \;=\; x'[\tau/\alpha] \;=\; |x'|[\tau/\alpha]$.

If $x = x'$ then $\quad |x'[e'/x]| \;=\; |e'| \;=\; x'[|e'|/x] \;=\; |x'|[|e'|/x]$.

Otherwise $\quad |x'[e'/x]| \;=\; |x'| \;=\; x' \;=\; x'[|e'|/x] \;=\; |x'|[|e'|/x]$.

case $\lambda x' : \tau' . e$. By induction hypothesis it follows that

$$|e[\tau/\alpha]| \;=\; |e|[\tau/\alpha] \qquad \text{and} \qquad |e[e'/x]| \;=\; |e|[|e'|/x].$$

$$\begin{aligned}|(\lambda x' : \tau' . e)[\tau/\alpha]| \;&=\; |\lambda x' : \tau' . e[\tau/\alpha]| \;=\; \lambda x' : \tau' . |e[\tau/\alpha]| \\ &=\; \lambda x' : \tau' . |e|[\tau/\alpha] \;=\; (\lambda x' : \tau' . |e|)[\tau/\alpha] \;=\; |(\lambda x' : \tau' . e)|[\tau/\alpha].\end{aligned}$$

$$\begin{aligned}|(\lambda x' : \tau' . e)[e'/x]| \;&=\; |\lambda x' : \tau' . e[e'/x]| \;=\; \lambda x' : \tau' . |e[e'/x]| \\ &=\; \lambda x' : \tau' . |e|[|e'|/x] \;=\; (\lambda x' : \tau' . |e|)[|e'|/x] \;=\; |(\lambda x' : \tau' . e)|[|e'|/x].\end{aligned}$$

case $e\ e''$. By induction hypothesis it follows that

$$\begin{aligned}|e[\tau/\alpha]| \;&=\; |e|[\tau/\alpha], & |e[e'/x]| \;&=\; |e|[|e'|/x], \\ |e''[\tau/\alpha]| \;&=\; |e''|[\tau/\alpha] \qquad \text{and} & |e''[e'/x]| \;&=\; |e''|[|e'|/x].\end{aligned}$$

$$\begin{aligned}|(e\ e'')[\tau/\alpha]| \;&=\; |e[\tau/\alpha]\ e''[\tau/\alpha]| \;=\; |e[\tau/\alpha]|\ |e''[\tau/\alpha]| \\ &=\; |e|[\tau/\alpha]\ |e''|[\tau/\alpha] \;=\; (|e|\ |e''|)[\tau/\alpha] \;=\; |e\ e''|[\tau/\alpha].\end{aligned}$$

$$\begin{aligned}|(e\ e'')[e'/x]| \;&=\; |e[e'/x]\ e''[e'/x]| \;=\; |e[e'/x]|\ |e''[e'/x]| \\ &=\; |e|[|e'|/x]\ |e''|[|e'|/x] \;=\; (|e|\ |e''|)[|e'|/x] \;=\; |e\ e''|[|e'|/x].\end{aligned}$$

case $\Lambda\alpha' . e$. By induction hypothesis it follows that

$$|e[\tau/\alpha]| \;=\; |e|[\tau/\alpha] \qquad \text{and} \qquad |e[e'/x]| \;=\; |e|[|e'|/x].$$

$$|(\Lambda\alpha'.\,e)[\tau/\alpha]| \;=\; |\Lambda\alpha'.\,e[\tau/\alpha]| \;=\; \Lambda\alpha'.\,|e[\tau/\alpha]|$$
$$=\; \Lambda\alpha'.\,|e|[\tau/\alpha] \;=\; (\Lambda\alpha'.\,|e|)[\tau/\alpha] \;=\; |(\Lambda\alpha'.\,e)|[\tau/\alpha].$$

$$|(\Lambda\alpha'.\,e)[e'/x]| \;=\; |\Lambda\alpha'.\,e[e'/x]| \;=\; \Lambda\alpha'.\,|e[e'/x]|$$
$$=\; \Lambda\alpha'.\,|e|[|e'|/x] \;=\; (\Lambda\alpha'.\,|e|)[|e'|/x] \;=\; |(\Lambda\alpha'.\,e)|[|e'|/x].$$

$\mathsf{case}\ e\,[\tau']$. By induction hypothesis it follows that

$$|e[\tau/\alpha]| \;=\; |e|[\tau/\alpha] \qquad \text{and} \qquad |e[e'/x]| \;=\; |e|[|e'|/x].$$

$$|(e\,[\tau'])[\tau/\alpha]| \;=\; |e[\tau/\alpha]\,[\tau']| \;=\; |e[\tau/\alpha]|\,[\tau']$$
$$=\; |e|[\tau/\alpha]\,[\tau'] \;=\; (|e|\,[\tau'])[\tau/\alpha] \;=\; |e\,[\tau']|[\tau/\alpha].$$

$$|(e\,[\tau'])[e'/x]| \;=\; |e[e'/x]\,[\tau']| \;=\; |e[e'/x]|\,[\tau']$$
$$=\; |e|[|e'|/x]\,[\tau'] \;=\; (|e|\,[\tau'])[|e'|/x] \;=\; |e\,[\tau']|[|e'|/x].$$

$\mathsf{case}\ \mathsf{fold}_{\tau'}\ e$. By induction hypothesis it follows that

$$|e[\tau/\alpha]| \;=\; |e|[\tau/\alpha] \qquad \text{and} \qquad |e[e'/x]| \;=\; |e|[|e'|/x].$$

$$|(\mathsf{fold}_{\tau'}\ e)[\tau/\alpha]| \;=\; |\mathsf{fold}_{\tau'}\ e[\tau/\alpha]| \;=\; (\lambda x'\!:\!\tau'.\,\mathsf{fold}_{\tau'}\ x')\ |e[\tau/\alpha]|$$
$$=(\lambda x'\!:\!\tau'.\,\mathsf{fold}_{\tau'}\ x')\ |e|[\tau/\alpha]=((\lambda x'\!:\!\tau'.\,\mathsf{fold}_{\tau'}\ x')\ |e|)[\tau/\alpha]=|((\mathsf{fold}_{\tau'}\ e)|[\tau/\alpha].$$

$$|(\mathsf{fold}_{\tau'}\ e)[e'/x]| \;=\; |\mathsf{fold}_{\tau'}\ e[e'/x]| \;=\; (\lambda x'\!:\!\tau'.\,\mathsf{fold}_{\tau'}\ x')\ |e[e'/x]|$$
$$=(\lambda x'\!:\!\tau'.\,\mathsf{fold}_{\tau'}\ x')\ |e|[|e'|/x]=((\lambda x'\!:\!\tau'.\,\mathsf{fold}_{\tau'}\ x')\ |e|)[|e'|/x]=|((\mathsf{fold}_{\tau'}\ e)|[|e'|/x].$$

$\mathsf{case}\ \mathsf{unfold}_{\tau'}\ e\ e''$. By induction hypothesis it follows that

$$|e[\tau/\alpha]| \;=\; |e|[\tau/\alpha], \qquad\qquad |e[e'/x]| \;=\; |e|[|e'|/x],$$
$$|e''[\tau/\alpha]| \;=\; |e''|[\tau/\alpha] \qquad \text{and} \qquad |e''[e'/x]| \;=\; |e''|[|e'|/x].$$

$$\begin{aligned}
&|(\mathsf{unfold}_{\tau'}\ e\ e'')[\tau/\alpha]| \;=\; |\mathsf{unfold}_{\tau'}\ e[\tau/\alpha]\ e''[\tau/\alpha]| \\
&=\; (\lambda x\!:\!\tau'.\,x)\ (\mathsf{unfold}_{\tau'}\ |e[\tau/\alpha]|\ |e''[\tau/\alpha]|) \\
&=\; (\lambda x\!:\!\tau'.\,x)[\tau/\alpha]\ (\mathsf{unfold}_{\tau'}\ |e|[\tau/\alpha]\ |e''|[\tau/\alpha]) \\
&=\; ((\lambda x\!:\!\tau'.\,x)\ (\mathsf{unfold}_{\tau'}\ |e|\ |e''|))[\tau/\alpha] \;=\; |\mathsf{unfold}_{\tau'}\ e\ e''|[\tau/\alpha].
\end{aligned}$$

$$\begin{aligned}
&|(\mathsf{unfold}_{\tau'}\ e\ e'')[e'/x]| \;=\; |\mathsf{unfold}_{\tau'}\ e[e'/x]\ e''[e'/x]| \\
&=\; (\lambda x\!:\!\tau'.\,x)\ (\mathsf{unfold}_{\tau'}\ |e[e'/x]|\ |e''[e'/x]|) \\
&=\; (\lambda x\!:\!\tau'.\,x)[|e'|/x]\ (\mathsf{unfold}_{\tau'}\ |e|[|e'|/x]\ |e''|[|e'|/x]) \\
&=\; ((\lambda x\!:\!\tau'.\,x)\ (\mathsf{unfold}_{\tau'}\ |e|\ |e''|))[|e'|/x] \;=\; |\mathsf{unfold}_{\tau'}\ e\ e''|[|e'|/x].
\end{aligned}$$

■

Theorem 16 (Coercion Typing Preservation).
If $\Delta; \Gamma \vdash_\mu e : \tau$ *then* $\Delta; \Gamma \vdash_\mu |e| : \tau$.

Proof. By induction on the derivation of $\Delta; \Gamma \vdash_\mu e : \tau$.

case (VAR). $$\frac{\Delta \vdash_\mu \Gamma(x)}{\Delta; \Gamma \vdash_\mu x : \Gamma(x)} \ .$$

By definition of $|\cdot|$ it follows that $|x| = x$ and thus

$$\frac{\Delta \vdash_\mu \Gamma(|x|)}{\Delta; \Gamma \vdash_\mu |x| : \Gamma(|x|)} \ .$$

It then follows that $\Delta; \Gamma \vdash_\mu |x| : \Gamma(x)$.

case (LAM). $$\frac{\Delta; \Gamma, x : \tau \vdash_\mu e : \tau'}{\Delta; \Gamma \vdash_\mu \lambda x : \tau.\, e : \tau \to \tau'} \ .$$

By induction hypothesis it follows that $\Delta; \Gamma, x : \tau \vdash_\mu |e| : \tau'$ and thus

$$\frac{\Delta; \Gamma, x : \tau \vdash_\mu |e| : \tau'}{\Delta; \Gamma \vdash_\mu \lambda x : \tau.\, |e| : \tau \to \tau'} \ .$$

It then follows that $\Delta; \Gamma \vdash_\mu |\lambda x : \tau.\, e| : \tau \to \tau'$.

case (APP). $$\frac{\Delta; \Gamma \vdash_\mu e : \tau \to \tau' \qquad \Delta; \Gamma \vdash_\mu e' : \tau}{\Delta; \Gamma \vdash_\mu e\ e' : \tau'} \ .$$

By induction hypothesis it follows that $\Delta; \Gamma \vdash_\mu |e| : \tau \to \tau'$.
By induction hypothesis it follows that $\Delta; \Gamma \vdash_\mu |e'| : \tau$ and thus

$$\frac{\Delta; \Gamma \vdash_\mu |e| : \tau \to \tau' \qquad \Delta; \Gamma \vdash_\mu |e'| : \tau}{\Delta; \Gamma \vdash_\mu |e|\ |e'| : \tau'} \ .$$

It then follows that $\Delta; \Gamma \vdash_\mu |e\ e'| : \tau'$.

case (TLAM). $$\frac{\Delta, \alpha; \Gamma \vdash_\mu e : \tau}{\Delta; \Gamma \vdash_\mu \Lambda\alpha.\, e : \Pi\alpha.\, \tau} \ .$$

By induction hypothesis it follows that $\Delta, \alpha; \Gamma \vdash_\mu |e| : \tau$, and thus

$$\frac{\Delta, \alpha; \Gamma \vdash_\mu |e| : \tau}{\Delta; \Gamma \vdash_\mu \Lambda\alpha.\, |e| : \Pi\alpha.\, \tau} \ .$$

It then follows that $\Delta; \Gamma \vdash_\mu |\Lambda\alpha.\, e| : \Pi\alpha.\, \tau$.

case (TAPP). $$\frac{\Delta;\Gamma \vdash_\mu e : \Pi\alpha.\,\tau \qquad \Delta \vdash_\mu \tau'}{\Delta;\Gamma \vdash_\mu e\,[\tau'] : \tau[\tau'/\alpha]}\ .$$

By induction hypothesis it follows that $\Delta;\Gamma \vdash_\mu |e| : \Pi\alpha.\,\tau$ and thus

$$\frac{\Delta;\Gamma \vdash_\mu |e| : \Pi\alpha.\,\tau \qquad \Delta \vdash_\mu \tau'}{\Delta;\Gamma \vdash_\mu |e|\,[\tau'] : \tau[\tau'/\alpha]}\ .$$

It then follows that $\Delta;\Gamma \vdash_\mu |e\,[\tau']| : \tau[\tau'/\alpha]$.

case (FOLD). $$\frac{\Delta;\Gamma \vdash_\mu e : \tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma \vdash_\mu \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e : \mu\alpha.\,\tau}\ .$$

By induction hypothesis it follows that $\Delta;\Gamma \vdash_\mu |e| : \tau[\mu\alpha.\,\tau/\alpha]$ and thus

$$\frac{\dfrac{\dfrac{\Delta;\Gamma, x:\tau[\mu\alpha.\,\tau/\alpha] \vdash_\mu x : \tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma, x:\tau[\mu\alpha.\,\tau/\alpha] \vdash_\mu \mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x : \mu\alpha.\,\tau}}{\Delta;\Gamma \vdash_\mu \lambda x:\tau[\mu\alpha.\,\tau/\alpha].\,\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x : \tau[\mu\alpha.\,\tau/\alpha] \to \mu\alpha.\,\tau} \qquad \Delta;\Gamma \vdash_\mu |e| : \tau[\mu\alpha.\,\tau/\alpha]}{\Delta;\Gamma \vdash_\mu (\lambda x:\tau[\mu\alpha.\,\tau/\alpha].\,\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ x)\ |e| : \mu\alpha.\,\tau}$$

It then follows that $\Delta;\Gamma \vdash_\mu |\mathsf{fold}_{\tau[\mu\alpha.\,\tau/\alpha]}\ e| : \mu\alpha.\,\tau$.

case (UNFOLD).

$$\frac{\Delta;\Gamma \vdash_\mu e : \mu\alpha.\,\tau \qquad \Delta;\Gamma \vdash_\mu e' : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e' : \mu\alpha'.\,\tau'}\ .$$

By induction hypothesis it follows that $\Delta;\Gamma \vdash_\mu |e| : \mu\alpha.\,\tau$.
By induction hypothesis it follows that

$$\Delta;\Gamma \vdash_\mu |e'| : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])$$

and thus

$$\frac{\Delta;\Gamma \vdash_\mu |e| : \mu\alpha.\,\tau \qquad \Delta;\Gamma \vdash_\mu |e'| : \mu\_.\,(\tau[\mu\alpha.\,\tau/\alpha] \to \tau'[\mu\alpha'.\,\tau'/\alpha'])}{\Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ |e|\ |e'| : \mu\alpha'.\,\tau'}\ .$$

It follows that

$$\frac{\dfrac{\Delta;\Gamma, x:\mu\alpha'.\,\tau' \vdash_\mu x : \mu\alpha'.\,\tau'}{\Delta;\Gamma \vdash_\mu \lambda x:\mu\alpha'.\,\tau'.\,x : (\mu\alpha'.\,\tau') \to \mu\alpha'.\,\tau'} \qquad \Delta;\Gamma \vdash_\mu \mathsf{unfold}_{\mu\alpha'.\,\tau'}\ |e|\ |e'| : \mu\alpha'.\,\tau'}{\Delta;\Gamma \vdash_\mu (\lambda x:\mu\alpha'.\,\tau'.\,x)\ (\mathsf{unfold}_{\mu\alpha'.\,\tau'}\ |e|\ |e'|) : \mu\alpha'.\,\tau'}$$

It then follows that $\Delta;\Gamma \vdash_\mu |\mathsf{unfold}_{\mu\alpha'.\,\tau'}\ e\ e'| : \mu\alpha'.\,\tau'$. ■

Theorem 17 (Coercion Reduction Preservation).
If $e \hookrightarrow e'$ *then* $|e| \rightsquigarrow^+ |e'|$.

Proof. By induction on the derivation of $e \hookrightarrow e'$.

case (R-APP). $(\lambda x{:}\tau.\, e)\ e' \ \hookrightarrow\ e[e'/x]$.

By the definition of $|\cdot|$ and Lemma 15 it follows that

$$|(\lambda x{:}\tau.\, e)\ e'| = (\lambda x{:}\tau.\, |e|)\ |e'| \quad \rightsquigarrow\ |e|[|e'|/x] = |e[e'/x]|.$$

case (R-TAPP). $(\Lambda\alpha.\, e)\,[\tau] \ \hookrightarrow\ e[\tau/\alpha]$.

By the definition of $|\cdot|$ and Lemma 15 it follows that

$$|(\Lambda\alpha.\, e)\,[\tau]| = (\Lambda\alpha.\, |e|)\,[\tau] \quad \rightsquigarrow\ |e|[\tau/\alpha] = |e[\tau/\alpha]|.$$

case (R-LAM). $$\frac{e \ \hookrightarrow\ e'}{\lambda x{:}\tau.\, e \ \hookrightarrow\ \lambda x{:}\tau.\, e'}\ .$$

By induction hypothesis it follows that $|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|$.
It then follows that

$$\frac{|e| \ \rightsquigarrow\ e_1 \ \rightsquigarrow\ \ldots \ \rightsquigarrow\ e_n \ \rightsquigarrow\ |e'|}{\lambda x{:}\tau.\, |e| \ \rightsquigarrow\ \lambda x{:}\tau.\, e_1 \ \rightsquigarrow\ \ldots \ \rightsquigarrow\ \lambda x{:}\tau.\, e_n \ \rightsquigarrow\ \lambda x{:}\tau.\, |e'|}\ .$$

By the definition of $|\cdot|$ it follows that $|\lambda x{:}\tau.\, e| \ \rightsquigarrow^+\ |\lambda x{:}\tau.\, e'|$.

case (R-APP1). $$\frac{e \ \hookrightarrow\ e'}{e\ e'' \ \hookrightarrow\ e'\ e''}\ .$$

By induction hypothesis it follows that $|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|$.
It then follows that

$$\frac{|e| \ \rightsquigarrow\ e_1 \ \rightsquigarrow\ \ldots \ \rightsquigarrow\ e_n \ \rightsquigarrow\ |e'|}{|e|\ |e''| \ \rightsquigarrow\ e_1\ |e''| \ \rightsquigarrow\ \ldots \ \rightsquigarrow\ e_n\ |e''| \ \rightsquigarrow\ |e'|\ |e''|}\ .$$

By the definition of $|\cdot|$ it follows that $|e\ e''| \ \rightsquigarrow^+\ |e'\ e''|$.

case (R-APP2). $$\frac{e \ \hookrightarrow\ e'}{v\ e \ \hookrightarrow\ v\ e'}\ .$$

By induction hypothesis it follows that $|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|$.
It then follows that

$$\frac{|e| \ \rightsquigarrow\ e_1 \ \rightsquigarrow\ \ldots \ \rightsquigarrow\ e_n \ \rightsquigarrow\ |e'|}{|v|\ |e| \ \rightsquigarrow\ |v|\ e_1 \ \rightsquigarrow\ \ldots \ \rightsquigarrow\ |v|\ e_n \ \rightsquigarrow\ |v|\ |e'|}\ .$$

By the definition of $|\cdot|$ it follows that $|v\ e| \rightsquigarrow^+ |v\ e'|$.

case (R-TLAM). $\dfrac{e \hookrightarrow e'}{\Lambda\alpha.\, e \hookrightarrow \Lambda\alpha.\, e'}$.

By induction hypothesis it follows that $|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|$.
It then follows that

$$\frac{|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|}{\Lambda\alpha.\,|e| \rightsquigarrow \Lambda\alpha.\, e_1 \rightsquigarrow \ldots \rightsquigarrow \Lambda\alpha.\, e_n \rightsquigarrow \Lambda\alpha.\,|e'|}\ .$$

By the definition of $|\cdot|$ it follows that $|\Lambda\alpha.\, e| \rightsquigarrow^+ |\Lambda\alpha.\, e'|$.

case (R-TAPP1). $\dfrac{e \hookrightarrow e'}{e\,[\tau] \hookrightarrow e'\,[\tau]}$.

By induction hypothesis it follows that $|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|$.
It then follows that

$$\frac{|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|}{|e|\,[\tau] \rightsquigarrow e_1\,[\tau] \rightsquigarrow \ldots \rightsquigarrow e_n\,[\tau] \rightsquigarrow |e'|\,[\tau]}\ .$$

By the definition of $|\cdot|$ it follows that $|e\,[\tau]| \rightsquigarrow^+ |e'\,[\tau]|$.

case (R-UNFOLD). $\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_{\tau}\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e') \hookrightarrow \mathsf{fold}_{\tau'}\ (e'\ e)$.

By the definition of $|\cdot|$ it follows that

$$\begin{aligned}
& |\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_{\tau}\ e)\ (\mathsf{fold}_{\tau\to\tau'}\ e')| \\
= \ & (\lambda x{:}\tau_0.\, x)\ (\mathsf{unfold}_{\tau''}\ |\mathsf{fold}_{\tau}\ e|\ |\mathsf{fold}_{\tau\to\tau'}\ e'|) \\
= \ & (\lambda x{:}\tau_0.\, x)\ (\mathsf{unfold}_{\tau''}\ ((\lambda x{:}\tau.\, \mathsf{fold}_{\tau}\ x)\ |e|)\ ((\lambda x{:}\tau\to\tau'.\, \mathsf{fold}_{\tau\to\tau'}\ x)\ |e'|)) \\
\rightsquigarrow \ & (\lambda x{:}\tau_0.\, x)\ (\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_{\tau}\ |e|)\ ((\lambda x{:}\tau\to\tau'.\, \mathsf{fold}_{\tau\to\tau'}\ x)\ |e'|)) \\
\rightsquigarrow \ & (\lambda x{:}\tau_0.\, x)\ (\mathsf{unfold}_{\tau''}\ (\mathsf{fold}_{\tau}\ |e|)\ (\mathsf{fold}_{\tau\to\tau'}\ |e'|)) \\
\rightsquigarrow \ & (\lambda x{:}\tau_0.\, x)\ (\mathsf{fold}_{\tau'}\ (|e'|\ |e|)) \\
\rightsquigarrow \ & (\lambda x{:}\tau'.\, \mathsf{fold}_{\tau'}\ x)\ (|e'|\ |e|) \ = \ (\lambda x{:}\tau'.\, \mathsf{fold}_{\tau'}\ x)\ |e'\ e| \ = \ |\mathsf{fold}_{\tau'}\ (e'\ e)|.
\end{aligned}$$

Notice that here in the shaded step we use the additional reduction rule (R-FOLD1) defined in the beginning of this section to lift the fold constructor, as the coercion $|\cdot|$ would do.

case (R-FOLD). $\dfrac{e \hookrightarrow e'}{\mathsf{fold}_{\tau}\ e \hookrightarrow \mathsf{fold}_{\tau}\ e'}$.

By induction hypothesis it follows that $|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|$.
It then follows that

$$\frac{|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow |e'|}{(\lambda x{:}\tau.\,\mathsf{fold}_\tau\ x)\ |e| \rightsquigarrow (\lambda x{:}\tau.\,\mathsf{fold}_\tau\ x)\ e_1 \rightsquigarrow \ldots \rightsquigarrow (\lambda x{:}\tau.\,\mathsf{fold}_\tau\ x)\ |e'|}\ .$$

By the definition of $|\cdot|$ it follows that $|\mathsf{fold}_\tau\ e| \rightsquigarrow^+ |\mathsf{fold}_\tau\ e'|$.

case (R-UNFOLD1). $$\frac{e \hookrightarrow e'}{\mathsf{unfold}_\tau\ e''\ e \hookrightarrow \mathsf{unfold}_\tau\ e''\ e'}\ .$$

By induction hypothesis it follows that $|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|$.
It then follows that

$$\frac{|e| \rightsquigarrow \ldots \rightsquigarrow |e'|}{(\lambda x{:}\tau.\,x)\ (\mathsf{unfold}_\tau\ |e''|\ |e|) \rightsquigarrow \ldots \rightsquigarrow (\lambda x{:}\tau.\,x)\ (\mathsf{unfold}_\tau\ |e''|\ |e'|)}\ .$$

By the definition of $|\cdot|$ it follows that $|\mathsf{unfold}_\tau\ e''\ e| \rightsquigarrow^+ |\mathsf{unfold}_\tau\ e''\ e'|$.

case (R-UNFOLD2). $$\frac{e \hookrightarrow e'}{\mathsf{unfold}_\tau\ e\ v \hookrightarrow \mathsf{unfold}_\tau\ e'\ v}\ .$$

By induction hypothesis it follows that $|e| \rightsquigarrow e_1 \rightsquigarrow \ldots \rightsquigarrow e_n \rightsquigarrow |e'|$.
It then follows that

$$\frac{|e| \rightsquigarrow \ldots \rightsquigarrow |e'|}{(\lambda x{:}\tau.\,x)\ (\mathsf{unfold}_\tau\ |e|\ |v|) \rightsquigarrow \ldots \rightsquigarrow (\lambda x{:}\tau.\,x)\ (\mathsf{unfold}_\tau\ |e'|\ |v|)}\ .$$

By the definition of $|\cdot|$ it follows that $|\mathsf{unfold}_\tau\ e\ v| \rightsquigarrow^+ |\mathsf{unfold}_\tau\ e'\ v|$. ■

Theorem 18 ($\mathbf{F}_\mu$ Strong Normalization ($\hookrightarrow$)).
If $\cdot;\cdot \vdash_\mu e{:}\tau$ then the length of any reduction sequence starting with e is finite.