12 Syntax of the Higher-Order Polymorphic Lambda Calculus F_ω

In this section, we extend $\lambda^\forall$ to the system F_ω, by allowing type variables to have higher-order types. The system F_ω was first defined by Girard [10]. Our presentation is inspired by Pfenning [27].

In $\lambda^\forall$, all type variables implicitly have the same base type. By allowing type variables to have higher-order types, we obtain a richer class of types and terms. In order to avoid confusions, we will say that a *type* is of a certain *kind*, rather than saying that a type is of a certain type. There is a distinguished kind that we will denote by $\star$, which, in the formula–as–type analogy of Curry and Howard, corresponds to the type of truth values. Some authors also denote this special kind as Type, which, to us, seems an unfortunate choice. Church and Andrews denote this kind as o. In the formula–as–type analogy, the types of kind $\star$ correspond to formulae, and terms have types of kind $\star$, since in this analogy, terms correspond to proofs.

Let $\mathcal{BK}$ be a set of *base kinds* containing the special kind $\star$.

Definition 12.1 The set $\mathcal{K}$ of *kinds* is defined inductively as follows:

$K \in \mathcal{K}$, whenever $K \in \mathcal{BK}$, and
$(K_1 \to K_2) \in \mathcal{K}$, whenever $K_1, K_2 \in \mathcal{K}$.

In omitting parentheses, we follow the usual convention that $\to$ associates to the right, that is, $K_1 \to K_2 \to \ldots K_{n-1} \to K_n$ abbreviates $(K_1 \to (K_2 \to \ldots (K_{n-1} \to K_n)\ldots))$. It should be noted that in Girard [10], kinds are called orders.

Let $\mathcal{V}$ be a countably infinite set of *type variables*, and let $\mathcal{TC}$ be a set of *type constructors*. It is assumed that every set of type constructors contains the special symbols $\Rightarrow$ and Π_K for every $K \in \mathcal{K}$. The type constructors are assigned kinds by a kind signature.

Definition 12.2 A *kind signature* is a function $\Xi\colon \mathcal{TC} \to \mathcal{K}$ assigning a kind to every type constructor in $\mathcal{TC}$, and such that $\Xi(\Rightarrow) = \star \to (\star \to \star)$, and $\Xi(\Pi_K) = (K \to \star) \to \star$.

The type constructor $\Rightarrow$ is the function type constructor, which, in the formula–as–type analogy, corresponds to logical implication ($\supset$), and Π_K constructs types of polymorphic functions. The idea behind the type constructor Π_K is due to Church who used it as a constant to serve as a universal quantifier (with lambda abstraction, see below). First-order function and predicate symbols can be handled by viewing a many-sorted function symbol f of rank n as a type constructor of kind $K_1 \to \ldots \to K_n \to K_{n+1}$, where $K_i \in \mathcal{BK}$ and $K_i \neq \star$ $(1 \leq i \leq n+1)$, and a many-sorted predicate symbol of rank n as a type constructor of kind $K_1 \to \ldots \to K_n \to \star$, where where $K_i \in \mathcal{BK}$ and $K_i \neq \star$ $(1 \leq i \leq n)$.

We now define raw types. Raw types do not necessarily "kind-check", and this will be taken care of by "kinding rules".

Definition 12.3 The set $\mathcal{T}$ of *raw type expressions* (for short, *raw types*) is defined inductively as follows:

$t \in \mathcal{T}$, whenever $t \in \mathcal{V}$,
$\sigma \in \mathcal{T}$, whenever $\sigma \in \mathcal{TC}$,
$(\lambda t{:}\, K.\, \sigma) \in \mathcal{T}$, whenever $t \in \mathcal{V}$, $\sigma \in \mathcal{T}$, and $K \in \mathcal{K}$, and
$(\sigma\tau) \in \mathcal{T}$, whenever $\sigma, \tau \in \mathcal{T}$.

Since $\Rightarrow$ and Π_K belong to $\mathcal{TC}$, by the last clause, $((\Rightarrow \sigma)\tau)$ and $(\Pi_K\sigma)$ are raw types for all $\sigma, \tau \in \mathcal{T}$, and all $K \in \mathcal{K}$. For simplicity of notation, $((\Rightarrow \sigma)\tau)$ is denoted as $(\sigma \Rightarrow \tau)$, and $(\Pi_K\sigma)$ as $\Pi_K\sigma$. In omitting parentheses, we follow the usual convention that application associates to the left. The subset of $\mathcal{T}$ consisting of the raw types of kind $\star$ is the set of types that can actually be the types of terms. A raw type of the form $\Pi_K(\lambda t{:}\, K.\sigma)$ will also be denoted as $\forall t{:}\, K.\, \sigma$. It should be noted that in Girard [10], types are called operators.

Next, we define the polymorphic raw terms. Let $\mathcal{X}$ be a countably infinite set of *term variables* (for short, variables), and let Σ be a set of constant symbols. The constants are assigned types by a type signature.

Definition 12.4 A *type signature* is a function $\Theta{:}\, \Sigma \to \mathcal{T}$ assigning a closed type (a type with no free type variables) to every symbol in Σ. A further restriction will be imposed later, namely that $\Theta(f)$ is of kind $\star$ for every $f \in \Sigma$.

Definition 12.5 The set $\mathcal{P}\Lambda$ of *polymorphic lambda raw Σ-terms* (for short, *raw terms*) is defined inductively as follows:

$c \in \mathcal{P}\Lambda$, whenever $c \in \Sigma$,
$x \in \mathcal{P}\Lambda$, whenever $x \in \mathcal{X}$,
$(MN) \in \mathcal{P}\Lambda$, whenever $M, N \in \mathcal{P}\Lambda$,
$(\lambda x{:}\, \sigma.\, M) \in \mathcal{P}\Lambda$, whenever $x \in \mathcal{X}$, $\sigma \in \mathcal{T}$, and $M \in \mathcal{P}\Lambda$,
$(M\sigma) \in \mathcal{P}\Lambda$, whenever $\sigma \in \mathcal{T}$ and $M \in \mathcal{P}\Lambda$,
$(\Lambda t{:}\, K.\, M) \in \mathcal{P}\Lambda$, whenever $t \in \mathcal{V}$, $K \in \mathcal{K}$, and $M \in \mathcal{P}\Lambda$.

The set of free variables in M will be denoted as $FV(M)$, and the set of free type variables in M as $\mathcal{FV}(M)$. The set of bound variables in M will be denoted as $BV(M)$, and the set of bound type variables in M as $\mathcal{BV}(M)$. The same notation is also used to denote the sets of free and bound variables in a type.

In omitting parentheses, we follow the usual convention that application associates to the left, that is, $M_1 M_2 \dots M_{n-1} M_n$ is an abbreviation for $((\dots(M_1 M_2)\dots M_{n-1})M_n)$.

Not all types are acceptable, only those that kind-check. Similarly, not all polymorphic raw terms are acceptable, only those that type-check. In order to kind-check a raw type and to type-check a raw term, one needs to make assumptions about the kinds and the types of the free variables. This can be done by introducing contexts. Then, kind-typing a raw type, or type-checking a raw term is done using a proof system working on certain expressions called judgments. However, substitution plays a crucial role in specifying the inference rules of this proof system, and so, we now focus our attention on substitutions.

13 Substitution and α-equivalence

We first define the notion of a substitution on raw types and raw terms.

Definition 13.1 A *substitution* is a function $\varphi\colon \mathcal{X} \cup \mathcal{V} \to \mathcal{P}\Lambda \cup \mathcal{T}$ such that, $\varphi(x) \neq x$ for only finitely many $x \in \mathcal{X} \cup \mathcal{V}$, $\varphi(x) \in \mathcal{P}\Lambda$ for all $x \in \mathcal{X}$, and $\varphi(t) \in \mathcal{T}$ for all $t \in \mathcal{V}$. The finite set $\{x \in \mathcal{X} \cup \mathcal{V} \mid \varphi(x) \neq x\}$ is called the *domain* of the substitution and is denoted by $dom(\varphi)$. If $dom(\varphi) = \{x_1, \dots, x_n\}$ and $\varphi(x_i) = u_i$ for every i, $1 \leq i \leq n$, the substitution φ is also denoted by $[u_1/x_1, \dots, u_n/x_n]$.

Given any substitution φ, any variable $y \in \mathcal{X} \cup \mathcal{V}$, and any term $u \in \mathcal{P}\Lambda \cup \mathcal{T}$, $\varphi[y := u]$ denotes the substitution such that, for all $z \in \mathcal{X} \cup \mathcal{V}$,

$$\varphi[y := u](z) = \begin{cases} u, & \text{if } y = z; \\ \varphi(z), & \text{if } z \neq y. \end{cases}$$

We also denote $\varphi[x := x]$ as φ_{-x}. The result of applying a substitution to a raw term or a type is defined recursively as follows.

Definition 13.2 Given any substitution $\varphi\colon \mathcal{X} \cup \mathcal{V} \to \mathcal{P}\Lambda \cup \mathcal{T}$, the function $\widehat{\varphi}\colon \mathcal{P}\Lambda \cup \mathcal{T} \to \mathcal{P}\Lambda \cup \mathcal{T}$ extending φ is defined recursively as follows:

$$\begin{aligned}
\widehat{\varphi}(x) &= \varphi(x), && x \in \mathcal{X}, \\
\widehat{\varphi}(t) &= \varphi(t), && t \in \mathcal{V}, \\
\widehat{\varphi}(f) &= f, && f \in \Sigma, \\
\widehat{\varphi}(\sigma) &= \sigma, && \sigma \in \mathcal{TC}, \\
\widehat{\varphi}(\lambda t\colon K.\, \sigma) &= \lambda t\colon K.\, \widehat{\varphi_{-t}}(\sigma), && \sigma \in \mathcal{T}, K \in \mathcal{K}, t \in \mathcal{V}, \\
\widehat{\varphi}(\sigma\tau) &= \widehat{\varphi}(\sigma)\widehat{\varphi}(\tau), && \sigma, \tau \in \mathcal{T},
\end{aligned}$$

$$\begin{aligned}
\widehat{\varphi}(PQ) &= \widehat{\varphi}(P)\widehat{\varphi}(Q), \qquad P,Q \in \mathcal{P}\Lambda,\\
\widehat{\varphi}(M\sigma) &= \widehat{\varphi}(M)\widehat{\varphi}(\sigma), \qquad M \in \mathcal{P}\Lambda, \sigma \in \mathcal{T},\\
\widehat{\varphi}(\lambda x{:}\sigma.\, M) &= \lambda x{:}\widehat{\varphi}(\sigma).\, \widehat{\varphi_{-x}}(M), \qquad M \in \mathcal{P}\Lambda, \sigma \in \mathcal{T}, x \in \mathcal{X},\\
\widehat{\varphi}(\Lambda t{:}K.\, M) &= \Lambda t{:}K.\, \widehat{\varphi_{-t}}(M), \qquad M \in \mathcal{P}\Lambda, K \in \mathcal{K}, t \in \mathcal{V}.
\end{aligned}$$

Given a polymorphic raw term M or a type σ, we also denote $\widehat{\varphi}(M)$ as $\varphi(M)$ and $\widehat{\varphi}(\sigma)$ as $\varphi(\sigma)$. Also, if $dom(\varphi) = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ and $\varphi = [M_1/x_1, \ldots, M_n/x_n]$, then $\widehat{\varphi}(M)$ is denoted as $M[M_1/x_1, \ldots, M_n/x_n]$. If $dom(\varphi) = \{t_1, \ldots, t_n\} \subseteq \mathcal{V}$ and $\varphi = [\sigma_1/t_1, \ldots, \sigma_n/t_n]$, then $\widehat{\varphi}(M)$ is denoted as $M[\sigma_1/t_1, \ldots, \sigma_n/t_n]$ (If σ is a type, then $\widehat{\varphi}(\sigma)$ is denoted as $\sigma[\sigma_1/t_1, \ldots, \sigma_n/t_n]$).

As for $\lambda^{\forall}$, we have to deal with α-conversion and variable capture in substitutions.

Example 13.3 We would like to consider the terms $M_1 = \Lambda t_1{:}\star.\, \lambda x_1{:}t_1.\, x_1$ and $M_2 = \Lambda t_2{:}\star.\lambda x_2{:}t_2.x_2$ to be equivalent. They both represent the "polymorphic identity function." This can be handled by defining an equivalence relation $\equiv_\alpha$ that relates terms that differ only by renaming of their bound variables.

Definition 13.4 The relation $\longrightarrow_\alpha$ of *immediate α-reduction* is defined by the following proof system:

Axioms:

$$\begin{aligned}
\lambda t{:}K.\, \sigma &\longrightarrow_\alpha \lambda v{:}K.\, \sigma[v/t] && \text{for all } v \in \mathcal{V} \text{ s.t. } v \notin \mathcal{FV}(\sigma) \cup \mathcal{BV}(\sigma)\\
\lambda x{:}\sigma.\, M &\longrightarrow_\alpha \lambda y{:}\sigma.\, M[y/x] && \text{for all } y \in \mathcal{X} \text{ s.t. } y \notin FV(M) \cup BV(M)\\
\Lambda t{:}K.\, M &\longrightarrow_\alpha \Lambda v{:}K.\, M[v/t] && \text{for all } v \in \mathcal{V} \text{ s.t. } v \notin \mathcal{FV}(M) \cup \mathcal{BV}(M)
\end{aligned}$$

Inference Rules:

$$\frac{\sigma \longrightarrow_\alpha \tau}{\sigma\rho \longrightarrow_\alpha \tau\rho} \qquad \frac{\sigma \longrightarrow_\alpha \tau}{\rho\sigma \longrightarrow_\alpha \rho\tau}$$

$$\frac{\sigma \longrightarrow_\alpha \tau}{(\sigma \Rightarrow \delta) \longrightarrow_\alpha (\tau \Rightarrow \delta)} \qquad \frac{\sigma \longrightarrow_\alpha \tau}{(\gamma \Rightarrow \sigma) \longrightarrow_\alpha (\gamma \Rightarrow \tau)}$$

$$\frac{\sigma \longrightarrow_\alpha \tau}{\Pi_K\sigma \longrightarrow_\alpha \Pi_K\tau}$$

$$\frac{\sigma \longrightarrow_\alpha \tau}{\lambda t{:}K.\, \sigma \longrightarrow_\alpha \lambda t{:}K.\, \tau}$$

$$\frac{M \longrightarrow_\alpha N}{MQ \longrightarrow_\alpha NQ} \qquad \frac{M \longrightarrow_\alpha N}{PM \longrightarrow_\alpha PN}$$

$$\frac{M \longrightarrow_\alpha N}{M\sigma \longrightarrow_\alpha N\sigma} \qquad \frac{\sigma \longrightarrow_\alpha \tau}{M\sigma \longrightarrow_\alpha M\tau}$$

$$\frac{M \longrightarrow_\alpha N}{\lambda x{:}\sigma.\, M \longrightarrow_\alpha \lambda x{:}\sigma.\, N} \qquad \frac{\sigma \longrightarrow_\alpha \tau}{\lambda x{:}\sigma.\, M \longrightarrow_\alpha \lambda x{:}\tau.\, M}$$

$$\frac{M \longrightarrow_\alpha N}{\Lambda t{:}K.\, M \longrightarrow_\alpha \Lambda t{:}K.\, N}$$

We define *α-reduction* as the reflexive and transitive closure $\overset{*}{\longrightarrow}_\alpha$ of $\longrightarrow_\alpha$. Finally, we define *α-conversion*, also called *α-equivalence*, as the least equivalence relation $\equiv_\alpha$ containing $\longrightarrow_\alpha$ ($\equiv_\alpha \;=\; (\longrightarrow_\alpha \cup \longrightarrow_\alpha^{-1})^*$).[21]

The following lemma shows that α-equivalence is "congruential" with respect to the term (and type) constructor operations.

Lemma 13.5 The following properties hold:

If $\sigma_1 \equiv_\alpha \tau_1$ and $\sigma_2 \equiv_\alpha \tau_2$, then $\sigma_1\sigma_2 \equiv_\alpha \tau_1\tau_2$.

If $\sigma_1 \equiv_\alpha \sigma_2$, then $\lambda t{:}K.\, \sigma_1 \equiv_\alpha \lambda t{:}K.\, \sigma_2$.

If $M_1 \equiv_\alpha M_2$ and $N_1 \equiv_\alpha N_2$, then $M_1N_1 \equiv_\alpha M_2N_2$.

If $M_1 \equiv_\alpha M_2$ and $\sigma_1 \equiv_\alpha \sigma_2$, then $M_1\sigma_1 \equiv_\alpha M_2\sigma_2$.

If $M_1 \equiv_\alpha M_2$ and $\sigma_1 \equiv_\alpha \sigma_2$, then $\lambda x{:}\sigma_1.\, M_1 \equiv_\alpha \lambda x{:}\sigma_2.\, M_2$.

If $M_1 \equiv_\alpha M_2$, then $\Lambda t{:}K.\, M_1 \equiv_\alpha \Lambda t{:}K.\, M_2$.

Proof. Straightforward by induction. □

The above lemma allows us to consider the term (and type) constructors as operating on $\equiv_\alpha$-equivalence classes. Let us denote the equivalence class of a term M modulo $\equiv_\alpha$ as $[M]$, and the equivalence class of a type σ modulo $\equiv_\alpha$ as $[\sigma]$. We extend application, type application, abstraction, and type abstraction, to equivalence classes as follows:

$$\begin{aligned}
[\sigma_1][\sigma_2] &= [\sigma_1\sigma_2],\\
[\lambda t{:}K.\, [\sigma]] &= [\lambda t{:}K.\, \sigma],\\
[M_1][M_2] &= [M_1M_2],\\
[M][\sigma] &= [M\sigma],\\
[\lambda x{:}[\sigma].\, [M]] &= [\lambda x{:}\sigma.\, M],\\
[\Lambda t{:}K.\, [M]] &= [\Lambda t{:}K.\, M].
\end{aligned}$$

[21] **Warning**: $\longrightarrow_\alpha$ is not symmetric!

From now on, we will usually identify a term or a type with its α-equivalence class and simply write M for $[M]$ and σ for $[\sigma]$.

Given a substitution $\varphi\colon \mathcal{X} \cup \mathcal{V} \to \mathcal{P}\Lambda \cup \mathcal{T}$, we let $FV(\varphi) = \bigcup_{x \in dom(\varphi)} FV(\varphi(x))$, and $\mathcal{FV}(\varphi) = \bigcup_{x \in dom(\varphi)} \mathcal{FV}(\varphi(x))$.

Definition 13.6 Given a substitution $\varphi\colon \mathcal{X} \cup \mathcal{V} \to \mathcal{P}\Lambda \cup \mathcal{T}$, given any term M or type σ, $safe(\varphi, M)$ and $safe(\varphi, \sigma)$ are defined recursively as follows:

$$
\begin{aligned}
safe(\varphi, x) &= \textbf{true}, && x \in \mathcal{X},\\
safe(\varphi, t) &= \textbf{true}, && t \in \mathcal{V},\\
safe(\varphi, f) &= \textbf{true}, && f \in \Sigma,\\
safe(\varphi, \sigma) &= \textbf{true}, && \sigma \in \mathcal{TC},\\
safe(\varphi, \lambda t\colon K.\, \sigma) &= safe(\varphi_{-t}, \sigma) \textbf{ and } t \notin \mathcal{FV}(\varphi), && \sigma \in \mathcal{T}, K \in \mathcal{K}, t \in \mathcal{V},\\
safe(\varphi, \sigma\tau) &= safe(\varphi, \sigma) \textbf{ and } safe(\varphi, \tau), && \sigma, \tau \in \mathcal{T},\\
safe(\varphi, PQ) &= safe(\varphi, P) \textbf{ and } safe(\varphi, Q), && P, Q \in \mathcal{P}\Lambda,\\
safe(\varphi, M\sigma) &= safe(\varphi, M) \textbf{ and } safe(\varphi, \sigma), && M \in \mathcal{P}\Lambda, \sigma \in \mathcal{T},\\
safe(\varphi, \lambda x\colon \sigma.\, M) &= safe(\varphi_{-x}, \sigma) \textbf{ and } safe(\varphi_{-x}, M) \textbf{ and } x \notin FV(\varphi),\\
&\qquad M \in \mathcal{P}\Lambda, \sigma \in \mathcal{T}, x \in \mathcal{X},\\
safe(\varphi, \Lambda t\colon K.\, M) &= safe(\varphi_{-t}, M) \textbf{ and } t \notin \mathcal{FV}(\varphi), && M \in \mathcal{P}\Lambda, t \in \mathcal{V}.
\end{aligned}
$$

When $safe(\varphi, M)$ holds we say that M *is safe for* φ, and when $safe(\varphi, \sigma)$ holds we say that σ *is safe for* φ.

Given any substitution φ and any term M (or type σ), it is immediately seen that there is some term M' (or type σ') such that $M \equiv_\alpha M'$ ($\sigma \equiv_\alpha \sigma'$) and M' is safe for φ (σ' is safe for φ). From now on, it is assumed that terms and types are α-renamed before a substitution is applied, so that the substitution is safe. It is natural to extend α-equivalence to substitutions as follows.

Definition 13.7 Given any two substitutions φ and φ' such $dom(\varphi) = dom(\varphi')$, we write $\varphi \equiv_\alpha \varphi$ iff $\varphi(x) \equiv_\alpha \varphi'(x)$ for every $x \in dom(\varphi)$.

We have the following lemma.

Lemma 13.8 For any two substitutions φ and φ', terms M, M', and types σ and σ', if M, M', σ, σ' are safe for φ and φ', $\varphi \equiv_\alpha \varphi'$, $M \equiv_\alpha M'$, and $\sigma \equiv_\alpha \sigma'$, then $\varphi(M) \equiv_\alpha \varphi'(M')$, and $\varphi(\sigma) \equiv_\alpha \varphi'(\sigma')$.

Proof. A very tedious induction on terms with many cases corresponding to the definition of α-equivalence. □

Corollary 13.9 (i) If $(\lambda t{:}\, K.\, \sigma_1)\tau_1 \equiv_\alpha (\lambda v{:}\, K.\, \sigma_2)\tau_2$, σ_1 is safe for $[\tau_1/t]$, and σ_2 is safe for $[\tau_2/v]$, then $\sigma_1[\tau_1/t] \equiv_\alpha \sigma_2[\tau_2/v]$. (ii) If $(\lambda x{:}\, \sigma_1.\, M_1)N_1 \equiv_\alpha (\lambda y{:}\, \sigma_2.\, M_2)N_2$, M_1 is safe for $[N_1/x]$, and M_2 is safe for $[N_2/y]$, then $M_1[N_1/x] \equiv_\alpha M_2[N_2/y]$. (iii) If $(\Lambda t{:}\, K.\, M_1)\tau_1 \equiv_\alpha (\Lambda v{:}\, K.\, M_2)\tau_2$, M_1 is safe for $[\tau_1/t]$, and M_2 is safe for $[\tau_2/v]$, then $M_1[\tau_1/t] \equiv_\alpha M_2[\tau_2/v]$.

We are now ready to present the proof system for kind-checking raw types and type-checking raw terms.

14 Contexts, Kind-Checking, and Type-Checking

First, we need the notion of a context.

Definition 14.1 A *context* is a partial function $\Delta{:}\, \mathcal{V} \cup \mathcal{X} \to \mathcal{K} \cup \mathcal{T}$ with a finite domain denoted as $dom(\Delta)$, and such that $\Delta(t) \in \mathcal{K}$ for every type variable $t \in \mathcal{V}$ and $\Delta(x) \in \mathcal{T}$ for every term variable $x \in \mathcal{X}$. Thus, a context Δ is a finite set of pairs of the form $t_i{:}\, K_i$ or $x_j{:}\, \sigma_j$, where the variables are pairwise distinct. Given a context Δ and a pair $\langle t, K\rangle$ where $t \in \mathcal{V}$ and $K \in \mathcal{K}$, or a pair $\langle x, \sigma\rangle$ where $x \in \mathcal{X}$ and $\sigma \in \mathcal{T}$, provided that $t \notin dom(\Delta)$ and $x \notin dom(\Delta)$, we write $\Delta, t{:}\, K$ for $\Delta \cup \{\langle t, K\rangle\}$, and $\Delta, x{:}\, \sigma$ for $\Delta \cup \{\langle x, \sigma\rangle\}$.

In order to determine whether a raw type kind-checks, or whether a raw term type-checks, we attempt to construct a proof of a judgment using the proof systems described below.

Definition 14.2 We define a number of judgments. A *judgment* is one of the following assertions:

Judgments

$\vdash \Delta \triangleright$	Δ *kind-checks*
$\vdash \Delta \triangleright \sigma{:}\, K$	σ *kind-checks* with kind K
$\vdash \Delta \triangleright M{:}\, \sigma$	M *type-checks* with type σ

Definition 14.3 Given any context Δ, the proof system for proving judgments of the form $\Delta \triangleright$ or judgments of the form $\Delta \triangleright \sigma{:}\, K$, called *kinding judgments*, is the following:

Axiom:

$$\emptyset \triangleright$$

Inference Rules:

$$\frac{\Delta \triangleright \qquad K \in \mathcal{K}}{\Delta, t{:}\, K \triangleright}, \qquad \text{where } t \notin dom(\Delta)$$

$$\frac{\Delta \triangleright}{\Delta \triangleright t\colon \Delta(t)}, \qquad \text{where } t \in dom(\Delta) \cap \mathcal{V} \qquad (\textit{type variables})$$

$$\frac{\Delta \triangleright}{\Delta \triangleright \sigma\colon K}, \qquad \text{where } \Xi(\sigma) = K \qquad (\textit{type constructors})$$

$$\frac{\Delta \triangleright \sigma\colon \star}{\Delta, x\colon \sigma \triangleright}, \qquad \text{where } x \notin dom(\Delta)$$

$$\frac{\Delta, t\colon K_1 \triangleright \sigma\colon K_2}{\Delta \triangleright (\lambda t\colon K_1.\ \sigma)\colon K_1 \to K_2} \qquad (\textit{abstraction})$$

where $t \notin \mathcal{FV}(\Delta(x))$ for every $x \in dom(\Delta) \cap \mathcal{X}$

$$\frac{\Delta \triangleright \sigma\colon K_1 \to K_2 \qquad \Delta \triangleright \tau\colon K_1}{\Delta \triangleright \sigma\tau\colon K_2} \qquad (\textit{application})$$

$$\frac{\Delta \triangleright \sigma\colon \star \qquad \Delta \triangleright \tau\colon \star}{\Delta \triangleright \sigma \Rightarrow \tau\colon \star} \qquad (\Rightarrow)$$

$$\frac{\Delta \triangleright \sigma\colon K \to \star}{\Delta \triangleright \Pi_K \sigma\colon \star} \qquad (\Pi)$$

A context Δ *kind-checks* iff $\Delta \triangleright$ is provable. When a judgment $\Delta \triangleright \sigma\colon K$ is provable, we say that the type σ *kind-checks* with kind K. It is not difficult to show that if $\Delta \triangleright \sigma\colon K$ is provable, then Δ kind-checks. From now on, we assume that $\Theta\colon \Sigma \to \mathcal{T}$ satisfies the following property: if $\Theta(f) = \sigma$, then $\triangleright \sigma\colon \star$ (recall that σ has no free variables).

We extend $\equiv_\alpha$ to (kinding) judgments as follows.

Definition 14.4 First, we define α-equivalence of contexts. Given two contexts $\Delta = \Gamma \cup \{x_1\colon \sigma_1, \ldots, x_n\colon \sigma_n\}$ and $\Delta' = \Gamma \cup \{x_1\colon \sigma'_1, \ldots, x_n\colon \sigma'_n\}$, where all pairs in Γ are of the form $t\colon K$, $t \in \mathcal{V}$, $K \in \mathcal{K}$, we write $\Delta \equiv_\alpha \Delta'$ iff $\sigma_i \equiv_\alpha \sigma'_i$ for all i, $1 \leq i \leq n$. Two kinding judgments $\Delta \triangleright \sigma\colon K$ and $\Delta' \triangleright \sigma'\colon K$ are *α-equivalent* iff $\Delta \equiv_\alpha \Delta'$ and $\sigma \equiv_\alpha \sigma'$.

In order to be able to manipulate $\equiv_\alpha$-equivalence classes of types, we add the following inference rules to the proof system of definition 14.3.

$$\frac{\Delta \triangleright \sigma\colon K \qquad \Delta \equiv_\alpha \Delta'}{\Delta' \triangleright \sigma\colon K} \qquad (\equiv'_\alpha)$$

$$\frac{\Delta \triangleright \sigma\colon K \qquad \sigma \equiv_\alpha \sigma'}{\Delta \triangleright \sigma'\colon K} \qquad (\equiv''_\alpha)$$

It it is not difficult to show that if two kinding judgments $\Delta \triangleright \sigma\colon K$ and $\Delta' \triangleright \sigma'\colon K$ are $\equiv_\alpha$-equivalent and there is a proof $\vdash \Delta \triangleright \sigma\colon K$, then there is a proof $\vdash \Delta' \triangleright \sigma'\colon K$. Consequently, it is legitimate to identify $\equiv_\alpha$-equivalent types and contexts, and we will do so from now on.

The types that kind-check form a simply-typed lambda calculus with set $\mathcal{BK}$ of base types. Any two types that are $\beta\eta$-convertible will be considered equivalent. Thus, we review the conversion rules for this calculus.

It is convenient to define reduction on raw types, and verify that it is kind-preserving when applied to a type that kind-checks.

Definition 14.5 The relation $\longrightarrow_{\lambda\to}$ of *immediate reduction* is defined in terms of the two relations $\longrightarrow_\beta$ and $\longrightarrow_\eta$, defined by the following proof system:

Axioms:

$$(\lambda t\colon K.\, \sigma)\tau \longrightarrow_\beta \sigma[\tau/t], \qquad \text{provided that } \sigma \text{ is safe for } [\tau/t] \tag{β}$$

$$\lambda t\colon K.\, (\sigma t) \longrightarrow_\eta \sigma, \qquad \text{provided that } t \notin \mathcal{FV}(\sigma) \tag{η}$$

Inference Rules: For each kind of reduction $\longrightarrow_r$ where $r \in \{\beta, \eta\}$,

$$\frac{\sigma \longrightarrow_r \tau}{\sigma\rho \longrightarrow_r \tau\rho} \qquad \frac{\sigma \longrightarrow_r \tau}{\rho\sigma \longrightarrow_r \rho\tau} \qquad \text{for all } \sigma, \tau \in \mathcal{T} \tag{\textit{congruence}}$$

$$\frac{\sigma \longrightarrow_r \tau}{\lambda t\colon K.\, \sigma \longrightarrow_r \lambda t\colon K.\, \tau} \qquad t \in \mathcal{V}, K \in \mathcal{K} \tag{ξ}$$

We define $\longrightarrow_{\lambda\to} = (\longrightarrow_\beta \cup \longrightarrow_\eta)$, and *reduction* as the reflexive and transitive closure $\overset{*}{\longrightarrow}_{\lambda\to}$ of $\longrightarrow_{\lambda\to}$. We also define *immediate conversion* $\longleftrightarrow_{\lambda\to}$ such that $\longleftrightarrow_{\lambda\to} = \longrightarrow_{\lambda\to} \cup \overset{-1}{\longrightarrow}_{\lambda\to}$, and *conversion* as the reflexive and transitive closure $\overset{*}{\longleftrightarrow}_{\lambda\to}$ of $\longleftrightarrow_{\lambda\to}$.

It is easily shown that reduction is kind-preserving. The relation $\longrightarrow_{\lambda\to}$ given in definition 14.5 induces a notion of reduction $\longrightarrow_{\lambda\to,\alpha}$ on $\equiv_\alpha$-equivalence classes of types defined as follows:

$$[\sigma] \longrightarrow_{\lambda\to,\alpha} [\tau] \qquad \text{iff} \qquad \sigma \longrightarrow_{\lambda\to} \tau.$$

It is immediately verified using lemma 13.5 and corollary 13.9 that $\longrightarrow_{\lambda\to,\alpha}$ is also defined by the proof system of definition 14.5 applied to $\equiv_\alpha$-equivalence classes.

Corollary 6.18 and corollary 6.19 imply that every type σ that kind-checks is strongly normalizable under $\beta\eta$-reduction, and that the Church-Rosser theorem holds under $\beta\eta$-reduction. Thus, every ($\equiv_\alpha$-equivalence class of) type σ that kind-checks has a unique $\beta\eta$-normal form. We can now define the proof system used for type-checking terms.

Definition 14.6 The proof system for proving judgments of the form $\Delta \triangleright M\colon\sigma$, called *typing judgments*, is the following:

Axioms: For every context Δ that kind-checks,

$$\Delta \triangleright c\colon\sigma, \qquad \text{where } \Theta(c) = \sigma \tag{\textit{constants}}$$

$$\Delta \triangleright x\colon\Delta(x), \qquad \text{where } x \in dom(\Delta) \cap \mathcal{X} \tag{\textit{variables}}$$

Inference Rules:

$$\frac{\Delta \triangleright M\colon\sigma \Rightarrow \tau \qquad \Delta \triangleright N\colon\sigma}{\Delta \triangleright MN\colon\tau} \tag{\textit{application}}$$

$$\frac{\Delta, x\colon\sigma \triangleright M\colon\tau}{\Delta \triangleright (\lambda x\colon\sigma.\ M)\colon\sigma \Rightarrow \tau} \tag{\textit{abstraction}}$$

$$\frac{\Delta \triangleright M\colon\Pi_K \sigma \qquad \Delta \triangleright \tau\colon K}{\Delta \triangleright M\tau\colon\sigma\tau} \tag{\textit{type application}}$$

$$\frac{\Delta, t\colon K \triangleright M\colon\sigma t}{\Delta \triangleright (\Lambda t\colon K.\ M)\colon\Pi_K \sigma} \tag{\textit{type abstraction}}$$

where in this rule, $t \notin \mathcal{FV}(\sigma)$, and $t \notin \mathcal{FV}(\Delta(x))$ for every $x \in dom(\Delta) \cap \mathcal{X}$.

$$\frac{\Delta \triangleright \tau\colon\star \qquad \Delta \triangleright M\colon\sigma \qquad \sigma \overset{*}{\longleftrightarrow}_{\lambda\to} \tau}{\Delta \triangleright M\colon\tau} \tag{\textit{type conversion}}$$

If $\Delta \triangleright M\colon\sigma$ is provable using the above proof system, we say that M *type-checks with type* σ *under* Δ and we write $\vdash \Delta \triangleright M\colon\sigma$. We say that the raw term M *type-checks* (or is *typable*) iff there is some Δ and some σ such that $\Delta \triangleright M\colon\sigma$ is derivable. It is not difficult to show that if a typing judgment $\Delta \triangleright M\colon\sigma$ is provable, then Δ kind-checks and $\Delta \triangleright \sigma\colon\star$ is provable.

In order to deal with $\equiv_\alpha$-equivalence, we define $\equiv_\alpha$-equivalent typing judgments as follows.

Definition 14.7 Two typing judgments $\Delta \triangleright M\colon\sigma$ and $\Delta' \triangleright M'\colon\sigma'$ are *α-equivalent* iff $\Delta \equiv_\alpha \Delta'$, $M \equiv_\alpha M'$, and $\sigma \equiv_\alpha \sigma'$.

We also add the following inference rules to the proof system of definition 14.6.

$$\frac{\Delta \triangleright M\colon\sigma \qquad \Delta \equiv_\alpha \Delta'}{\Delta' \triangleright M\colon\sigma} \tag{$\equiv'_\alpha$}$$

$$\frac{\Delta \rhd M : \sigma \qquad M \equiv_\alpha M'}{\Delta \rhd M' : \sigma} \qquad (\equiv''_\alpha)$$

$$\frac{\Delta \rhd M : \sigma \qquad \sigma \equiv_\alpha \sigma'}{\Delta \rhd M : \sigma'} \qquad (\equiv'''_\alpha)$$

Clearly, if $\Delta \rhd M : \sigma$ is provable, then $\Delta \rhd \sigma : \star$ and $\Delta \rhd$ are also provable. It is not difficult to show that if two typing judgments $\Delta \rhd M : \sigma$ and $\Delta' \rhd M' : \sigma'$ are α-equivalent and there is a proof $\vdash \Delta \rhd M : \sigma$, then there is a proof $\vdash \Delta' \rhd M' : \sigma'$. Thus, it is legitimate to work with equivalence classes of types, terms, and contexts, modulo $\equiv_\alpha$-equivalence. This is also true for substitutions.

Example 14.8 The following is a proof that $M = \Lambda t{:}\star.\ \lambda x{:}t.\ x$ type-checks with type $\forall u{:}\star.\ (u \Rightarrow u) = \Pi_\star(\lambda u{:}\star.\ (u \Rightarrow u))$.

$$\begin{array}{l} t{:}\star, x{:}t \rhd x{:}t \\ \quad t{:}\star \rhd (\lambda x{:}t.\ x){:}(t \Rightarrow t) \qquad (t \Rightarrow t) \overset{*}{\longleftrightarrow}_{\lambda\to} (\lambda u{:}\star.\ (u \Rightarrow u))t \\ \quad t{:}\star \rhd (\lambda x{:}t.\ x){:}(\lambda u{:}\star.\ (u \Rightarrow u))t \\ \qquad \rhd (\Lambda t{:}\star.\ \lambda x{:}t.\ x){:}\Pi_\star(\lambda u{:}\star.\ (u \Rightarrow u)). \end{array}$$

Remark. It is also possible to formulate the typing rules for F_ω by choosing $\forall$ as a primitive instead of Π. For instance, this is the choice adopted in Girard [10]. In this case, we have the following two rules that replace type application and type abstraction:

$$\frac{\Delta \rhd M : \forall t{:}K.\ \sigma \qquad \Delta \rhd \tau : K}{\Delta \rhd M\tau : \sigma[\tau/t]} \qquad (\textit{type application'})$$

$$\frac{\Delta, t{:}K \rhd M : \sigma}{\Delta \rhd (\Lambda t{:}K.\ M){:}\forall t{:}K.\ \sigma} \qquad (\textit{type abstraction'})$$

where in this rule, $t \notin \mathcal{FV}(\Delta(x))$ for every $x \in dom(\Delta) \cap \mathcal{X}$.

Let F'_ω be this new system. Recall that if Π is chosen as a primitive, then $\forall t{:}K.\ \sigma$ is an abbreviation for $\Pi_K(\lambda t{:}K.\ \sigma)$. Then, using the fact that $(\lambda t{:}K.\ \sigma)t \longrightarrow_{\lambda\to} \sigma$, that $(\lambda t{:}K.\ \sigma)\tau \longrightarrow_{\lambda\to} \sigma[\tau/t]$, and the type conversion rule, it is immediately verified by induction on the depth of proofs that every judgment $\Delta \rhd M : \sigma$ provable in F'_ω is also provable in F_ω (translating $\forall t{:}K.\ \sigma$ in F'_ω to $\Pi_K(\lambda t{:}K.\ \sigma)$ in F_ω). Conversely, using the fact that for every $t \notin \mathcal{FV}(\sigma)$, $\forall t{:}K.\ (\sigma t) = \Pi_K(\lambda t{:}K.\ (\sigma t)) \longrightarrow_\eta \Pi_K\sigma$, $\sigma\tau = (\sigma t)[\tau/t]$, and the type conversion rule, it is immediately verified by induction on the depth of proofs that every judgment $\Delta \rhd M : \sigma$ provable in F_ω is also provable in F'_ω. Thus, the two proof systems

are equivalent. In the absence of η-conversion (on types), it is an interesting exercise to show that every proof in F_ω can be converted into a proof in F'_ω (it can be shown that $\Pi_K(\lambda t{:}K.\,(\sigma t))$ and $\Pi_K\sigma$ are equivalent in F'_ω). Thus, F_ω and F'_ω are also equivalent in the absence of η-conversion.

We can define the concept of the order of a kind or of a type. This will enable us to define subsets of F_ω.

Definition 14.9 The *order* of a kind $K \in \mathcal{K}$ is defined inductively as follows:

$$\begin{aligned} ord(K) &= 0, \quad K \in \mathcal{BK} - \{\star\} \\ ord(\star) &= 1 \\ ord(K_1 \to K_2) &= max(ord(K_1)+1, ord(K_2)). \end{aligned}$$

The order of a type σ that kind-checks is the order of its kind, and in particular, given a context Δ, for every $t{:}K \in \Delta$ where t is a type variable, $ord(t) = ord(K)$. Then, given any $m > 0$, we define F_m as the subset of F_ω obtained by restricting the order of all type variables and of all type constructors to be at most m, and the order of all bound type variables to be at most $m-1$. Thus, in F_1, only type variables of base kind (other than $\star$) can be bound, and F_1 corresponds to minimal first-order logic. In F_2, we can also have bound type variables of kind $\star$ or $K_1 \to \ldots K_n \to K$, with $K_1, \ldots, K_n \in \mathcal{BK} - \{\star\}$, $K \in \mathcal{BK}$. This corresponds to a slight extension of $\lambda^\forall$. In F_3, we can also have bound variables of kind $\star \to \ldots \star \to \star$, and also $(K_1 \to K_2) \to (K_1 \to K_2)$ where $K_1, K_2 \in \mathcal{BK}$, and generally, bound variables of order ≤ 2. It is also natural to identify F_0 with the simply-typed λ-calculus.

We can now define the notion of reduction in F_ω.

15 Reduction and Conversion

As in definition 14.5, we first define reduction on raw terms, and then extend it to $\equiv_\alpha$-equivalence classes.

Definition 15.1 The relation $\longrightarrow_{F_\omega}$ of *immediate reduction* is defined in terms of the four relations $\longrightarrow_\beta$, $\longrightarrow_\eta$, $\longrightarrow_{\tau\beta}$, and $\longrightarrow_{\tau\eta}$, defined by the following proof system:

Axioms:

$$(\lambda x{:}\sigma.\,M)N \longrightarrow_\beta M[N/x], \qquad \text{provided that } M \text{ is safe for } [N/x] \qquad (\beta)$$

$$\lambda x{:}\sigma.\,(Mx) \longrightarrow_\eta M, \qquad \text{provided that } x \notin FV(M) \qquad (\eta)$$

$$(\Lambda t{:}K.\,M)\tau \longrightarrow_{\tau\beta} M[\tau/t], \qquad \text{provided that } M \text{ is safe for } [\tau/t] \qquad (\textit{type } \beta)$$

$$\Lambda t{:}K.\,(Mt) \longrightarrow_{\tau\eta} M, \qquad \text{provided that } t \notin \mathcal{FV}(M) \qquad (\textit{type } \eta)$$

Inference Rules: For each kind of reduction $\longrightarrow_r$ where $r \in \{\beta, \eta, \tau\beta, \tau\eta, \lambda^{\to}\}$, where $\longrightarrow_{\lambda^{\to}}$ is from definition 14.5,

$$\frac{M \longrightarrow_r N}{MQ \longrightarrow_r NQ} \qquad \frac{M \longrightarrow_r N}{PM \longrightarrow_r PN} \quad \text{for all } P, Q \in \mathcal{P}\Lambda \qquad (\textit{congruence})$$

$$\frac{M \longrightarrow_r N}{M\sigma \longrightarrow_r N\sigma} \qquad \frac{\sigma \longrightarrow_r \tau}{M\sigma \longrightarrow_r M\tau} \qquad \frac{\sigma \longrightarrow_r \tau}{\lambda x{:}\sigma.\,M \longrightarrow_r \lambda x{:}\tau.\,M} \quad \sigma, \tau \in \mathcal{T}$$

$$(\textit{type congruence})$$

$$\frac{M \longrightarrow_r N}{\lambda x{:}\sigma.\,M \longrightarrow_r \lambda x{:}\sigma.\,N} \quad x \in \mathcal{X}, \sigma \in \mathcal{T} \qquad (\xi)$$

$$\frac{M \longrightarrow_r N}{\Lambda t{:}K.\,M \longrightarrow_r \Lambda t{:}K.\,N} \quad t \in \mathcal{V}, K \in \mathcal{K} \qquad (\textit{type } \xi)$$

We define $\longrightarrow_{F_\omega} = \longrightarrow_\beta \cup \longrightarrow_\eta \cup \longrightarrow_{\tau\beta} \cup \longrightarrow_{\tau\eta} \cup \longrightarrow_{\lambda^{\to}}$, and *reduction* as the reflexive and transitive closure $\overset{*}{\longrightarrow}_{F_\omega}$ of $\longrightarrow_{F_\omega}$. We also define *immediate conversion* $\longleftrightarrow_{F_\omega}$ such that $\longleftrightarrow_{F_\omega} = \longrightarrow_{F_\omega} \cup \overset{-1}{\longrightarrow}_{F_\omega}$, and *conversion* as the reflexive and transitive closure $\overset{*}{\longleftrightarrow}_{F_\omega}$ of $\longleftrightarrow_{F_\omega}$.

It can be shown that reduction and conversion are type-preserving. The relation $\longrightarrow_{F_\omega}$ given in definition 15.1 induces a notion of reduction $\longrightarrow_{F_\omega,\alpha}$ on $\equiv_\alpha$-equivalence classes of terms defined as follows:

$$[M] \longrightarrow_{F_\omega,\alpha} [N] \qquad \text{iff} \qquad M \longrightarrow_{F_\omega} N.$$

It is immediately verified using lemma 13.5 and corollary 13.9 that $\longrightarrow_{F_\omega,\alpha}$ is also defined by the proof system of definition 15.1 applied to $\equiv_\alpha$-equivalence classes. Thus, in what follows, contexts, types, and terms, are identified with their $\equiv_\alpha$-equivalence classes. In particular, if we consider an equivalence class of the form $[(\lambda x{:}\sigma.\,M)N]$, we can assume that M has been α-renamed so that M is safe for the substitution $[N/x]$, and similarly for a class of the form $[(\Lambda t{:}K.\,M)\tau]$ (and for types). For simplicity of notation, we will write $\longrightarrow_{F_\omega}$ instead of $\longrightarrow_{F_\omega,\alpha}$.

It should be noted that the type congruence rules are indispensable, unless one requires that the result of performing a substitution is $\beta\eta$-normalized.

Example 15.2 It is easy to give a proof for the typing judgment

$$t{:}\star \triangleright (\Lambda u{:}(\star \to \star).\ \lambda x{:}ut.\ x){:}\,\Pi_{\star\to\star}(\lambda u{:}(\star \to \star).\ (ut \Rightarrow ut)),$$

and since the type $\lambda v{:}\star.\ v$ kind-checks (with kind $\star \to \star$), the typing judgment

$$t{:}\star \triangleright ((\Lambda u{:}(\star \to \star).\ \lambda x{:}ut.\ x)\lambda v{:}\star.\ v){:}\,((\lambda u{:}(\star \to \star).\ (ut \Rightarrow ut))\lambda v{:}\star.\ v),$$

is also provable. Note that all the types in the term $(\Lambda u{:}(\star \to \star).\ \lambda x{:}ut.\ x)\lambda v{:}\star.\ v$ are $\beta\eta$-normalized. Now, we have the reduction

$$(\Lambda u{:}(\star \to \star).\ \lambda x{:}ut.\ x)\lambda v{:}\star.\ v \longrightarrow_{F_\omega} \lambda x{:}(\lambda v{:}\star.\ v)t.\ x,$$

but $\lambda x{:}(\lambda v{:}\star.\ v)t.\ x$ is not $\beta\eta$-normalized. For that, it is necessary to perform the reduction

$$\lambda x{:}(\lambda v{:}\star.\ v)t.\ x \longrightarrow_{F_\omega} \lambda x{:}t.\ x.$$

We also have the reduction sequence

$$\begin{aligned}(\lambda u{:}(\star \to \star).\ (ut \Rightarrow ut))\lambda v{:}\star.\ v &\longrightarrow_{F_\omega} ((\lambda v{:}\star.\ v)t \Rightarrow (\lambda v{:}\star.\ v)t)\\ &\xrightarrow{*}_{F_\omega} (t \Rightarrow t).\end{aligned}$$

Note that the typing judgment $t{:}\star \triangleright (\lambda x{:}t.\ x){:}\,(t \Rightarrow t)$ is provable.

16 The Method of Candidates

We now generalize the method of candidates to F_ω. The proof that we sketch is modelled after Girard's original proof, and only differs in the notation and in the fact that we present it in a slightly more general setting, using $\mathcal{T}$-closed families, and closed families of Girard sets. As pointed out by Thierry Coquand, it is possible to prove strong normalization for F_ω using an untyped version of the candidates and the erasing trick. However, the typed version seems necessary when the system F_ω is enriched, for example with first-order rewriting, and we present the typed version. The main complication is that we now have new types formed by λ-abstraction and application. Thus, it is necessary to define candidates of reducibility by induction on the *kind* of types. As before, let $\mathcal{C}_\sigma$ denote the set of candidates of type σ. For types of kind $\star$, basically nothing changes. For a type σ of kind $K_1 \to K_2$, a candidate of type σ is any function

$$f{:}\bigcup_{\tau:K_1} \{\tau\} \times \mathcal{C}_\tau \to \bigcup_{\tau:K_1} \mathcal{C}_{\sigma\tau}$$

such that $f(\tau, C) \in \mathcal{C}_{\sigma\tau}$ for every $C \in \mathcal{C}_\tau$ and satisfying a technical condition listed in definition 16.2.

Actually, there is a problem with this definition, namely that types may contain type variables, and in order for these types to kind-check, we need to assume that the type variables have been assigned kinds. There are two ways to overcome this problem. The first solution, which is the solution adopted by Coquand in his proof of normalization for the theory of constructions [5], is to define the notion of a candidate of type $\Delta \triangleright \sigma\colon K$, where $\Delta \triangleright \sigma\colon K$ kind-checks. In this approach, we deal with families $\mathcal{C}_{\Delta\triangleright\sigma:K}$ of sets of candidates indexed by provable kinding judgments. Roughly, a candidate C of type $\Delta \triangleright \sigma\colon K$ is a set of provable typing judgments of the form $\Delta' \triangleright M\colon\sigma$ where $\Delta \subseteq \Delta'$, and satisfying certain properties as in definition 7.8. If this approach is followed, it is also necessary to define $[\![\Delta \triangleright \sigma\colon K]\!]\theta\eta$, as opposed to simply $[\![\sigma]\!]\theta\eta$. The proof can be carried out, but the notation is quite formidable.

However, in F_ω, since the fact that a type kind-checks only depends on assigning kinds to types variables, and kinds are independent of the types, there is a second simpler solution (adopted by Girard). This second solution is to relativize the definition of a family of sets of candidates to a global kind assignment $\kappa\colon \mathcal{V} \to \mathcal{K}$. This way, we can deal with types σ that kind-check under some context that agrees with κ on $\mathcal{V}$. We also assume that κ is extended to the type constructors, so that it agrees with Ξ on $\mathcal{TC}$. The above discussion leads to the following definition.

Definition 16.1 Given a kind assignment $\kappa\colon \mathcal{V} \to \mathcal{K}$, we let $\mathcal{T}|_\kappa$ be the set of all types σ that "kind-check under κ", that is, such that $\Delta \triangleright \sigma\colon K$ is provable for some kind $K \in \mathcal{K}$ and some context Δ whose restriction to $\mathcal{V}$ agrees with κ. Given $\kappa\colon \mathcal{V} \to \mathcal{K}$, for every type $\sigma \in \mathcal{T}|_\kappa$ of kind $\star$, we let $\mathcal{PT}_\sigma$ be the set of all provable typing judgments of the form $\Delta \triangleright M\colon\sigma$, where Δ is any context whose restriction to $\mathcal{V}$ agrees with κ.

We will use the abbreviation $\Delta \triangleright M \in S$ for $\Delta \triangleright M\colon\sigma \in S$ when S is a subset of $\mathcal{PT}_\sigma$. We also use the notation $\sigma\colon K \in \mathcal{T}|_\kappa$ to express the fact that σ kind-checks with kind K under κ, and $\Delta, \kappa \triangleright M\colon\sigma$ to mean that $\Delta' \triangleright M\colon\sigma$ is provable for some (finite) context Δ' such that $\Delta \subseteq \Delta'$ and the restriction of Δ' to $\mathcal{V}$ agrees with κ.

Given any two types $\sigma, \tau \in \mathcal{T}|_\kappa$ of kind $\star$ and any two sets $S \subseteq \mathcal{PT}_\sigma$ and $T \subseteq \mathcal{PT}_\tau$, we let $[S \Rightarrow T]$ be the subset of $\mathcal{PT}_{\sigma\Rightarrow\tau}$ defined as before:

$$[S \Rightarrow T] = \{\Delta \triangleright M \in \mathcal{PT}_{\sigma\Rightarrow\tau} \mid \forall \Delta' \triangleright N, \text{ if } \Delta \subseteq \Delta' \text{ and } \Delta' \triangleright N \in S, \text{ then } \Delta' \triangleright MN \in T\}.$$

Given a kind assignment $\kappa\colon \mathcal{V} \to \mathcal{K}$, a $\mathcal{T}|_\kappa$-closed family is defined as follows.

Definition 16.2 Let $\mathcal{C} = (\mathcal{C}_\sigma)_{\sigma \in \mathcal{T}|_\kappa}$ be a $\mathcal{T}|_\kappa$-indexed family where for each σ, if σ is of kind $\star$ then $\mathcal{C}_\sigma$ is a nonempty set of subsets of $\mathcal{PT}_\sigma$, else if σ is of kind $K_1 \to K_2$ then $\mathcal{C}_\sigma$ is a nonempty set of functions from $\bigcup_{\tau:K_1} \{\tau\} \times \mathcal{C}_\tau$ to $\bigcup_{\tau:K_1} \mathcal{C}_{\sigma\tau}$, and the following properties hold:

(1) For every $\sigma \in \mathcal{T}|_\kappa$ of kind $\star$, every $C \in \mathcal{C}_\sigma$ is a nonempty subset of $\mathcal{PT}_\sigma$.

(2) For every $\sigma, \tau \in \mathcal{T}|_\kappa$ of kind $\star$, for every $C \in \mathcal{C}_\sigma$ and $D \in \mathcal{C}_\tau$, we have $[C \Rightarrow D] \in \mathcal{C}_{\sigma \Rightarrow \tau}$.

(3) For every $\sigma \in \mathcal{T}|_\kappa$ of kind $K \to \star$, for every $\tau \in \mathcal{T}|_\kappa$ of kind K, for every family $(A_{\tau,C})_{\tau \in \mathcal{T}|_\kappa, C \in \mathcal{C}_\tau}$, where each set $A_{\tau,C}$ is in $\mathcal{C}_{\sigma\tau}$, we have

$$\{\Delta \triangleright M \in \mathcal{PT}_{\Pi_K \sigma} \mid \forall (\tau\colon K) \in \mathcal{T}|_\kappa,\ \Delta, \kappa \triangleright M\tau \in \bigcap_{C \in \mathcal{C}_\tau} A_{\tau,C}\} \in \mathcal{C}_{\Pi_K \sigma}.$$

(4) For every $\sigma \in \mathcal{T}|_\kappa$ of kind $K_1 \to K_2$,

$$\begin{aligned} \mathcal{C}_\sigma = \{f\colon & \bigcup_{\tau:K_1 \in \mathcal{T}|_\kappa} \{\tau\} \times \mathcal{C}_\tau \to \bigcup_{\tau:K_1 \in \mathcal{T}|_\kappa} \mathcal{C}_{\sigma\tau} \\ & \text{such that } f(\tau, C) \in \mathcal{C}_{\sigma\tau} \text{ for every } C \in \mathcal{C}_\tau, \text{ and} \\ & f(\tau_1, C) = f(\tau_2, C) \text{ whenever } \tau_1 \overset{*}{\longleftrightarrow}_{\lambda\to} \tau_2\}. \end{aligned}$$

A family satisfying the above conditions is called a $\mathcal{T}|_\kappa$-*closed family*.

Definition 16.3 Let $\mathcal{C}$ be a $\mathcal{T}|_\kappa$-closed family. A pair $\langle \theta, \eta \rangle$ where $\theta\colon \mathcal{V} \to \mathcal{T}|_\kappa$ is a substitution and $\eta\colon \mathcal{TC} \cup \mathcal{V} \to \bigcup \mathcal{C}$ is a *candidate assignment* iff for every $t \in \mathcal{V}$, $\kappa(t) = K$ implies that $\theta(t)\colon K \in \mathcal{T}|_\kappa$, $\eta(t) \in \mathcal{C}_{\theta(t)}$, and $\eta(\sigma) \in \mathcal{C}_\sigma$ for every $\sigma \in \mathcal{TC}$.

We can associate certain sets of provable typing judgments to the types inductively as explained below.

Definition 16.4 Given any candidate assignment $\langle \theta, \eta \rangle$, for every type $\sigma \in \mathcal{T}|_\kappa$, we define $[\![\sigma]\!]\theta\eta$ as follows:

$$\begin{aligned} [\![t]\!]\theta\eta &= \eta(t), \text{ whenever } t \in \mathcal{TC} \cup \mathcal{V}; \\ [\![(\sigma \Rightarrow \tau)]\!]\theta\eta &= [[\![\sigma]\!]\theta\eta \Rightarrow [\![\tau]\!]\theta\eta]; \\ [\![\Pi_K \sigma]\!]\theta\eta &= \{\Delta \triangleright M \in \mathcal{PT}_{\theta(\Pi_K \sigma)} \mid \forall (\tau\colon K) \in \mathcal{T}|_\kappa, \\ &\qquad \Delta, \kappa \triangleright M\tau \in \bigcap_{C \in \mathcal{C}_\tau} [\![\sigma]\!]\theta\eta(\tau, C)\}; \\ [\![\sigma\tau]\!]\theta\eta &= [\![\sigma]\!]\theta\eta(\theta(\tau), [\![\tau]\!]\theta\eta); \\ [\![\lambda t\colon K.\ \sigma]\!]\theta\eta &= \lambda\tau\lambda C \in \mathcal{C}_{\tau:K}.\ [\![\sigma]\!]\theta[t := \tau]\eta[t := C]. \end{aligned}$$

In the last clause of this definition, $\lambda\tau\lambda C \in \mathcal{C}_{\tau:K}.\, [\![\sigma]\!]\theta[t := \tau]\eta[t := C]$ denotes the function f such that $f(\tau, C) = [\![\sigma]\!]\theta[t := \tau]\eta[t := C]$ for every $C \in \mathcal{C}_\tau$ such that $\tau \in \mathcal{T}|_\kappa$ is of kind K.

The following technical lemmas will be useful later.

Lemma 16.5 Given any candidate assignments $\langle\theta_1, \eta_1\rangle$ and $\langle\theta_2, \eta_2\rangle$, for every $\sigma \in \mathcal{T}|_\kappa$, if θ_1, θ_2 agree on $\mathcal{FV}(\sigma)$, and η_1, η_2 agree on $\mathcal{FV}(\sigma)$ and $\mathcal{TC}$, then $[\![\sigma]\!]\theta_1\eta_1 = [\![\sigma]\!]\theta_2\eta_2$.

Proof. Easy induction on the structure of types. □

Lemma 16.6 Given any two types $\sigma, \tau \in \mathcal{T}|_\kappa$, for every candidate assignment $\langle\theta, \eta\rangle$,

$$[\![\sigma[\tau/t]]\!]\theta\eta = [\![\sigma]\!]\theta[t := \theta(\tau)]\eta[t := [\![\tau]\!]\theta\eta].$$

Proof. Straightforward induction on the structure of σ. □

The following lemma is crucial and shows that $[\![\sigma]\!]\theta\eta$ actually has a constant value on the equivalence class of σ modulo $\lambda^{\to}$-convertibility.

Lemma 16.7 For every candidate assignment $\langle\theta, \eta\rangle$, for every two types $\sigma, \sigma' \in \mathcal{T}|_\kappa$, if $\sigma \overset{*}{\longleftrightarrow}_{\lambda\to} \sigma'$, then $[\![\sigma]\!]\theta\eta = [\![\sigma']\!]\theta\eta$.

Proof. It is sufficient to prove that if $\sigma \overset{*}{\longrightarrow}_{\lambda\to} \sigma'$, then $[\![\sigma]\!]\theta\eta = [\![\sigma']\!]\theta\eta$. The proof proceeds by induction on the proof that $\sigma \overset{*}{\longrightarrow}_{\lambda\to} \sigma'$. The only nontrivial cases are β and η-conversion, and those are handled using lemma 16.6 and lemma 16.5. □

We now have a version of "Girard's trick" for F_ω.

Lemma 16.8 (Girard) If $\mathcal{C}$ is a $\mathcal{T}|_\kappa$-closed family, for every candidate assignment $\langle\theta, \eta\rangle$, for every type σ, then $[\![\sigma]\!]\theta\eta \in \mathcal{C}_{\theta(\sigma)}$.

Proof. The lemma is proved by induction on the structure of types. The only case worth mentioning is the case of a typed λ-abstraction. By α-renaming, it can be assumed that $\lambda t{:}\, K.\, \sigma$ is safe for θ. In this case, we use lemma 16.7 and the fact that $(\lambda t{:}\, K.\, \theta(\sigma))\tau \longrightarrow_{\lambda\to} \theta(\sigma)[\tau/t]$, and that because $\lambda t{:}\, K.\, \sigma$ is safe for θ, $\theta(\sigma)[\tau/t] = \theta[t := \tau](\sigma)$. □

In order to use lemma 16.8 in proving properties of polymorphic lambda calculi, we need to define $\mathcal{T}|_\kappa$-closed families satisfying some additional properties.

Definition 16.9 We say that a $\mathcal{T}|_\kappa$-indexed family $\mathcal{C}$ is a *family of sets of candidates of reducibility* iff it is $\mathcal{T}|_\kappa$-closed and satisfies the conditions listed below.[22]

[22] Again, we also have to assume that every $C \in \mathcal{C}$ is closed under α-equivalence.

R0. Whenever $\Delta \triangleright M \in C$ and $\Delta \subseteq \Delta'$, then $\Delta' \triangleright M \in C$.

R1. For every $\sigma{:}\star \in \mathcal{T}|_\kappa$, for every set $C \in \mathcal{C}_\sigma$, $\Delta \triangleright x \in C$, for every $x{:}\sigma \in \Delta$,
For every $\sigma{:}\star \in \mathcal{T}|_\kappa$ where $\sigma = \Theta(f)$, for every set $C \in \mathcal{C}_\sigma$, $\Delta \triangleright f \in C$, for every $f \in \Sigma$.

R2. (i) For all $\sigma{:}\star, \tau{:}\star \in \mathcal{T}|_\kappa$, for every $C \in \mathcal{C}_\tau$, for all Δ, Δ', if

$$\begin{aligned}&\Delta \triangleright M \in \bigcup \mathcal{C}_\tau,\\ &\Delta' \triangleright N \in \bigcup \mathcal{C}_\sigma, \text{ and}\\ &\Delta' \triangleright M[N/x] \in C, \text{ then}\\ &\Delta' \triangleright (\lambda x{:}\sigma.\, M)N \in C.\end{aligned}$$

(ii) For every $\sigma \in \mathcal{T}|_\kappa$ of kind $K \to \star$, every $\tau \in \mathcal{T}|_\kappa$ of kind K, for every $C \in \mathcal{C}_{\sigma\tau}$, for all Δ, Δ', if

$$\begin{aligned}&\Delta \triangleright M \in \bigcup \mathcal{C}_{\sigma t} \text{ and}\\ &\Delta' \triangleright M[\tau/t] \in C, \text{ then}\\ &\Delta' \triangleright (\Lambda t{:}K.\, M)\tau \in C.\end{aligned}$$

Lemma 7.9 generalizes to F_ω as follows.

Lemma 16.10 (Girard) Let $\mathcal{C} = (\mathcal{C}_\sigma)_{\sigma \in \mathcal{T}|_\kappa}$ be a family of sets of candidates of reducibility. For every $\Gamma \triangleright M \in \mathcal{PT}_\sigma$, for every candidate assignment $\langle \theta, \eta \rangle$, for every substitution $\varphi{:}\Gamma \to \Delta$, if $\theta(\Delta), \kappa \triangleright \varphi(x) \in [\![\Gamma(x)]\!]\theta\eta$ for $x \in FV(M)$, then $\theta(\Delta), \kappa \triangleright \varphi(\theta(M)) \in [\![\sigma]\!]\theta\eta$.

Proof. It is similar to the proof of lemma 7.9 and proceeds by induction on the depth of the proof tree for $\Gamma \triangleright M{:}\sigma$. Type conversion is handled using lemma 16.7. We only sketch the verification for two of the other cases.

Case 1.

$$\frac{\Gamma, t{:}K \triangleright M{:}\sigma t}{\Gamma \triangleright (\Lambda t{:}K.\, M){:}\Pi_K \sigma} \qquad (\textit{type abstraction})$$

where in this rule, $t \notin \mathcal{FV}(\sigma)$, and $t \notin \mathcal{FV}(\Gamma(x))$ for every $x \in dom(\Gamma) \cap \mathcal{X}$.

Given any $\tau \in \mathcal{T}|_\kappa$, the induction hypothesis applies to $\Gamma, t{:}K \triangleright M{:}\sigma t$ and to any candidate assignment $\langle \theta[t := \tau], \eta[t := C] \rangle$ where $C \in \mathcal{C}_\tau$ and $\tau{:}K \in \mathcal{T}|_\kappa$ (and by suitable α-renaming, $t \notin \mathcal{FV}(\Delta(x))$ for every $x \in dom(\Delta)$, and the safeness conditions for substitution hold). Thus, $\theta[t := \tau](\Delta) = \theta(\Delta)$, and due to the proviso on the inference rule, $\theta[t := \tau](\Gamma) = \theta(\Gamma)$, and $\theta[t := \tau](M) = \theta(M)[\tau/t]$. Thus, we have

$$\theta(\Delta), \kappa \triangleright \varphi(\theta(M))[\tau/t] \in [\![\sigma t]\!]\theta[t := \tau]\eta[t := C].$$

In particular, this holds for $\tau = t$, and so $\theta(\Delta), \kappa \rhd \varphi(\theta(M)) \in \bigcup \mathcal{C}_{\theta(\sigma t)}$. Then, by (R2)(ii),

$$\theta(\Delta), \kappa \rhd (\Lambda t\colon K.\ \varphi(\theta(M)))\tau \in [\![\sigma t]\!]\theta[t := \tau]\eta[t := C],$$

that is, $\theta(\Delta), \kappa \rhd \varphi(\theta(\Lambda t\colon K.\ M))\tau \in [\![\sigma t]\!]\theta[t := \tau]\eta[t := C]$.

Again, due to the the proviso on the rule, $[\![\sigma]\!]\theta[t := \tau]\eta[t := C] = [\![\sigma]\!]\theta\eta$, and since

$$[\![\sigma t]\!]\theta[t := \tau]\eta[t := C] = [\![\sigma]\!]\theta[t := \tau]\eta[t := C](\theta[t := \tau](t), [\![t]\!]\theta[t := \tau]\eta[t := C])$$

and $[\![t]\!]\theta[t := \tau]\eta[t := C] = C$, $\theta[t := \tau](t) = \tau$, we have

$$[\![\sigma t]\!]\theta[t := \tau]\eta[t := C] = [\![\sigma]\!]\theta\eta(\tau, C).$$

Thus, $\theta(\Delta), \kappa \rhd \varphi(\theta(\Lambda t\colon K.\ M))\tau \in [\![\sigma]\!]\theta\eta(\tau, C)$ for all $C \in \mathcal{C}_\tau$ such that $\tau\colon K \in \mathcal{T}|_\kappa$, which proves that

$$\theta(\Delta) \rhd \varphi(\theta(\Lambda t\colon K.\ M)) \in [\![\Pi_K \sigma]\!]\theta\eta.$$

Case 2.

$$\frac{\Gamma \rhd M\colon \Pi_K \sigma \qquad \Gamma \rhd \tau\colon K}{\Gamma \rhd M\tau\colon \sigma\tau} \qquad (\textit{type application})$$

By the induction hypothesis, $\theta(\Delta), \kappa \rhd \varphi(\theta(M)) \in [\![\Pi_K \sigma]\!]\theta\eta$, and so

$$\theta(\Delta), \kappa \rhd \varphi(\theta(M))\delta \in [\![\sigma]\!]\theta\eta(\delta, C)$$

for every $C \in \mathcal{C}_\delta$ where $\delta\colon K \in \mathcal{T}|_\kappa$. By choosing $\delta = \theta(\tau)$, $C = [\![\tau]\!]\theta\eta$, and using the fact that $\varphi(\theta(M))\theta(\tau) = \varphi(\theta(M\tau))$ and $[\![\sigma\tau]\!]\theta\eta = [\![\sigma]\!]\theta\eta(\theta(\tau), [\![\tau]\!]\theta\eta)$, we have

$$\theta(\Delta), \kappa \rhd \varphi(\theta(M\tau)) \in [\![\sigma\tau]\!]\theta\eta,$$

as desired. □

As for $\lambda^\forall$, in order to show the existence of families of candidates of reducibility, we need stronger conditions. One can define saturated sets as in section 8, or Girard sets as in section 9. We shall present the version of the Girard sets, leaving the other version as an exercise to the reader.

A simple term is defined as in definition 9.1, that is, a term M is *simple* iff it is either a variable x, a constant $f \in \Sigma$, an application MN, or a type application $M\tau$.

Definition 16.11 Let $S = (S_\sigma)_{\sigma:\star\in\mathcal{T}|_\kappa}$ be a family such that each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$.[23] For every type $\sigma:\star \in \mathcal{T}|_\kappa$, a subset C of S_σ is a *Girard set* of type σ iff the following conditions hold:[24]

CR0. Whenever $\Delta \triangleright M \in C$ and $\Delta \subseteq \Delta'$, then $\Delta' \triangleright M \in C$.

CR1. If $\Delta \triangleright M \in C$, then M is SN w.r.t. $\longrightarrow_{F_\omega}$;

CR2. If $\Delta \triangleright M \in C$ and $M \longrightarrow_{F_\omega} N$, then $\Delta \triangleright N \in C$;

CR3. For every simple term $\Delta \triangleright M \in \mathcal{PT}_\sigma$, if $\Delta \triangleright N \in C$ for every N such that $M \longrightarrow_{F_\omega} N$, then $\Delta \triangleright M \in C$.

Note that (CR3) implies that all simple irreducible terms are in C. Also, (CR1), (CR2), and (CR3) are defined w.r.t. $\longrightarrow_{F_\omega}$, which means that $\beta\eta$-reduction on types is taken into account. This is crucial for proving strong normalization.

Definition 16.12 Let $S = (S_\sigma)_{\sigma:\star\in\mathcal{T}|_\kappa}$ be a family such that each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$. We say that S is *closed* iff for all $\sigma:\star, \tau:\star \in \mathcal{T}|_\kappa$, for every $x \in \mathcal{X}$, if $\Delta \triangleright M \in \mathcal{PT}_{\sigma\Rightarrow\tau}$ and $\Delta, x:\sigma \triangleright Mx \in S_\tau$, then $\Delta \triangleright M \in S_{\sigma\Rightarrow\tau}$, and for every $t:K \in \mathcal{T}|_\kappa$ and $\sigma:K \in \mathcal{T}|_\kappa$, if $\Delta \triangleright M \in \mathcal{PT}_{\Pi_K\sigma}$ and $\Delta, t:K \triangleright Mt \in S_{\sigma t}$ then $\Delta \triangleright M \in S_{\Pi_K\sigma}$.

We have the following generalization of lemma 9.4.

Lemma 16.13 (Girard) Let $S = (S_\sigma)_{\sigma:\star\in\mathcal{T}|_\kappa}$ be a closed family where each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$, and let $\mathcal{C}$ be the $\mathcal{T}|_\kappa$-indexed family such that for each $\sigma:\star \in \mathcal{T}|_\kappa$, $\mathcal{C}_\sigma$ is the set of Girard subsets of S_σ, and for $\sigma:K_1 \to K_2 \in \mathcal{T}|_\kappa$, $\mathcal{C}_\sigma$ is defined as in clause (4) of definition 16.2. If $S_\sigma \in \mathcal{C}_\sigma$ for every $\sigma:\star \in \mathcal{T}|_\kappa$ (i.e. S_σ is a Girard subset of itself), then $\mathcal{C}$ is a family of sets of candidates of reducibility.

Proof. It is similar to the proof of lemmas 9.3 and 9.4. A subtlety arises in proving that (R2) holds. We proceed as in the proof that (S2) holds (given in lemma 9.3), that is, we show that $\Delta \triangleright Q \in C$ whenever $(\lambda x:\sigma.\ M)N \longrightarrow_{F_\omega} Q$, and that $\Delta \triangleright Q \in C$ whenever $(\Lambda t:K.\, M)\tau \longrightarrow_{F_\omega} Q$, assuming that M and N are SN. However, σ or τ can be $\beta\eta$-reduced, and we also need to prove that $\beta\eta$-reduction on types ($\longrightarrow_{\lambda\to}$) is strongly normalizing. Fortunately, this is a special case of corollary 6.18, as observed earlier.

It is also necessary to verify conditions (1), (2), (3), (4) of definition 16.2. This is done by induction on kinds, and for the kind $\star$ by induction on types. Verifying (1), (2), (3) is done as in lemma 9.4. We still need to check (4), that for a type $\sigma:K_1 \to K_2 \in \mathcal{T}|_\kappa$, $\mathcal{C}_\sigma$ is nonempty. This is done by induction on $K_1 \to K_2$. The base case holds since

[23] Note that $S = (S_\sigma)_{\sigma:\star\in\mathcal{T}|_\kappa}$ is not a $\mathcal{T}|_\kappa$-indexed family. It is indexed by the set of types of kind $\star$.

[24] We also have to assume that every Girard subset of S is closed under α-equivalence.

$can_\sigma = S_\sigma \in \mathcal{C}_\sigma$ for every $\sigma{:}\star \in \mathcal{T}|_\kappa$. For $\sigma{:}K_1 \to K_2 \in \mathcal{T}|_\kappa$, for every $\tau{:}K_1 \in \mathcal{T}|_\kappa$, since $\sigma\tau{:}K_2 \in \mathcal{T}|_\kappa$, by the induction hypothesis there is some function $can_{\sigma\tau} \in \mathcal{C}_{\sigma\tau}$, and so the function can_σ such that $can_\sigma(\tau, C) = can_{\sigma\tau}$ for every $C \in \mathcal{C}_\tau$ $(\tau{:}K_1 \in \mathcal{T}|_\kappa)$ is in $\mathcal{C}_\sigma$. □

We now have a version of Girard's fundamental theorem for F_ω.

Theorem 16.14 (Girard) Let $S = (S_\sigma)_{\sigma:\star\in\mathcal{T}|_\kappa}$ be a closed family where each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$, let $\mathcal{C}$ be the $\mathcal{T}|_\kappa$-indexed family of sets defined in lemma 16.13, and assume that $S_\sigma \in \mathcal{C}_\sigma$ for every $\sigma{:}\star \in \mathcal{T}|_\kappa$. For every $\Delta{\triangleright}M \in \mathcal{PT}_\sigma$, we have $\Delta{\triangleright}M \in S_\sigma$.

Proof. By lemma 16.13, $\mathcal{C}$ is a family of sets of candidates of reducibility. We now apply lemma 16.10 to any assignment (for example, the assignment with value $\eta(t) = can_t$), the identity type substitution, and the identity term substitution, which is legitimate since by (CR3), every variable belongs to every Girard set.[25] □

Remark: Thierry Coquand pointed out to us that because $\beta\eta$-reduction on types $(\longrightarrow_\lambda\to)$ is strongly normalizing, an untyped version of the candidates using the erasing trick works for F_ω. The function $Erase{:}\mathcal{P}\Lambda \to \Lambda$ for F_ω is defined recursively as follows:

$Erase(c) = c$, whenever $c \in \Sigma$,
$Erase(x) = x$, whenever $x \in \mathcal{X}$,
$Erase(MN) = Erase(M)Erase(N)$,
$Erase(\lambda x{:}\sigma.\, M) = \lambda x.\, Erase(M)$,
$Erase(M\sigma) = Erase(M)$,
$Erase(\lambda t{:}K.\, M) = Erase(M)$.

However, obtaining the confluence property using the erasing trick is an open problem. The next lemma is a generalization of lemma 10.2 and gives interesting examples of closed families of Girard sets.

Lemma 16.15 (i) The family SN_β such that for every $\sigma{:}\star \in \mathcal{T}|_\kappa$, $SN_{\beta,\sigma}$ is the set of typing judgments $\Delta \triangleright M{:}\sigma$ provable in F_ω such that M is strongly normalizing under β-reduction, is a closed family of Girard sets. (ii) The family $SN_{\beta\eta}$ such that for every $\sigma{:}\star \in \mathcal{T}|_\kappa$, $SN_{\beta\eta,\sigma}$ is the set of typing judgments $\Delta \triangleright M{:}\sigma$ provable in F_ω such that M is strongly normalizing under $\beta\eta$-reduction, is a closed family of Girard sets. (iii) The family consisting for every $\sigma{:}\star \in \mathcal{T}|_\kappa$ of the set of typing judgments $\Delta \triangleright M{:}\sigma$ provable in F_ω such that confluence under β-reduction holds from M and all of its subterms, is a closed family of Girard sets. (iv) The family consisting for every $\sigma{:}\star \in \mathcal{T}|_\kappa$ of the set of typing judgments $\Delta \triangleright M{:}\sigma$ provable in F_ω such that confluence under $\beta\eta$-reduction holds from M and all of its subterms, is a closed family of Girard sets.

[25] Actually, some α-renaming may have to be performed on M and σ so that they are both safe for the type and term identity substitution.

Proof. (i)-(ii) It is trivial to verify that closure and (CR0)–(CR3) hold. (iii)-(iv) The proof is similar to the one given in appendix 2. □

It should be noted that the fact that the above results are relativized to a type assignment κ is really not a restriction. Indeed, we could assume that the set of type variables is partitioned into a family of countable sets, one for each kind. Thus, we can assume that we are dealing with a single κ.

The system F_ω can be extended in several ways. For example, product types and existential types can be added. In fact, such an extension is studied in Girard's thesis [10], including disjunctive types. Strong normalization still holds. Another way of extending F_ω is to allow a richer class of kinds and types. This can be achieved by allowing term variables in types and the formation of new types by λ-abstraction over these variables. At the same time, richer kinds are allowed, namely dependent products. Such a system, the *theory of constructions*, was invented by Coquand [5, 7]. The theory of constructions is also investigated in Huet and Coquand [6]. A related system, LF, has been investigated by Harper, Honsell, and Plotkin [13, 14]. Strong normalization holds for these systems. For the theory of construction, the proof uses Girard's method of candidates of reducibility and follows the general scheme used in the proof of strong normalization for F_ω, but it is more complex because types may contain terms. The problem is that it is no longer possible to prove first that $\beta\eta$-reduction is strongly normalizing on types. Roughly, one needs to define $[\![\Delta \triangleright \sigma\colon K]\!]\theta\eta$ and $[\![\Delta \triangleright K\colon Kind]\!]\theta\eta$. For details, the interested (and perseverant) reader is referred to Coquand [5]. A proof of strong normalization for the theory of constructions is also given in Seldin [33], which also contains an extensive study of type systems including $\lambda^\forall$ and the theory of constructions. For LF, a proof of strong normalization consists in mapping LF into the simply-typed lambda calculus. For details, the reader should consult [14]. Other interesting work on the theory of constructions and F_ω appears in Paulin-Morhing [26], where it is shown how programs and their proofs can be extracted. Related work is done in Pfenning [28].

Acknowledgment. I am grateful to Jean Yves Girard for providing invaluable inspiration, advice, and encouragement. I also wish to thank Val Breazu-Tannen, Pierre Louis Curien, Bob Harper, Zhaohui Luo, and Kathleen Milsted, for looking at this paper very carefully and for their help and advice with the correctness and the presentation. Val caught some bugs and gave me advice for simplifying the notation, Bob and Kathleen read the entire manuscript and contributed a large number of improvements. Zhaohui Luo pointed out to me the necessity of condition (CR2), which I had overlooked in earlier versions. Pierre Louis Curien and Roberto Di Cosmo reported a gap in the proof of lemma 9.3. Finally, I wish to thank Thierry Coquand, Gerard Huet, Tomas Isakowitz, John Mitchell, Frank Pfenning, Andre Scedrov, Wayne Snyder, and Rick Statman, for their comments.