ON GIRARD'S "CANDIDATS DE REDUCTIBILITÉ"

Jean H. Gallier

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

Chapter in *Logic and Computer Science*,
P. Odifreddi, editor, Academic Press, 1989

November 13, 2012

This research was partially supported by ONR under Grant No. N00014-88-K-0593.

ON GIRARD'S "CANDIDATS DE REDUCTIBILITÉ"

Jean H. Gallier

Abstract: We attempt to elucidate the conditions required on Girard's candidates of reducibility (in French, "candidats de reductibilité") in order to establish certain properties of various typed lambda calculi, such as strong normalization and Church-Rosser property. We present two generalizations of the candidates of reducibility, an untyped version in the line of Tait and Mitchell, and a typed version which is an adaptation of Girard's original method. As an application of this general result, we give two proofs of strong normalization for the second-order polymorphic lambda calculus under $\beta\eta$-reduction (and thus under β-reduction). We present two sets of conditions for the typed version of the candidates. The first set consists of conditions similar to those used by Stenlund (basically the typed version of Tait's conditions), and the second set consists of Girard's original conditions. We also compare these conditions, and prove that Girard's conditions are stronger than Tait's conditions. We give a new proof of the Church-Rosser theorem for both β-reduction and $\beta\eta$-reduction, using the modified version of Girard's method. We also compare various proofs that have appeared in the literature (see section 11). We conclude by sketching the extension of the above results to Girard's higher-order polymorphic calculus F_ω, and in appendix 1, to F_ω with product types.

1 Introduction

In this article, we attempt to elucidate the conditions required on Girard's candidates of reducibility (in French, "candidats de reductibilité") in order to establish certain properties of various typed lambda calculi, such as strong normalization and Church-Rosser property. We present two generalizations of the candidates of reducibility, an untyped version in the line of Tait and Mitchell [37, 24], and a typed version which is an adaptation of Girard's original method [9, 10]. As an application of this general result, we give two proofs of strong normalization for the second-order polymorphic lambda calculus under $\beta\eta$-reduction (and thus under β-reduction). We present two sets of conditions for the typed version of the candidates: a set of conditions similar to those used by Stenlund [35], (basically the typed version of Tait's conditions, Tait 1973 [37]), and Girard's original conditions (Girard [10], [11]). We also compare these conditions, and prove that Girard's conditions are stronger than Tait's conditions. We give a new proof of the Church-Rosser theorem for both β-reduction and $\beta\eta$-reduction, using the modified version of Girard's method. We also compare various proofs that have appeared in the literature (see section 11). We conclude by sketching the extension of the above results to Girard's higher-order polymorphic calculus F_ω, and in appendix 1, to F_ω with product types.

It is worth noting that the generalized method of candidates plays an important role in Breazu-Tannen and Gallier [4], where conservation results conjectured in Breazu-Tannen [3] are proved about the combination of algebraic rewriting with $\beta\eta$-reduction in polymorphic λ-calculi.

Familiarity with the polymorphic typed lambda calculus is not assumed for reading this article. This explains why we have included some rather lengthy introductory sections. An expert should probably proceed directly to section 6. On the other hand, a certain familiarity with the simply-typed lambda calculus will help. Good references on the lambda calculus include Barendregt [1], Hindley and Seldin [15], Stenlund [35], Girard [11], and Huet [16, 18]. An extensive discussion of the role and importance of type theory and an exposition of related results are given in Scedrov [32]. Another excellent introduction to type systems and their relevance to programming language theory appears in Mitchell [25].

2 Syntax of the Second-Order Polymorphic Lambda Calculus

Our presentation of the Girard/Reynolds second-order lambda calculus [9, 30, 11] is heavily inspired by Breazu-Tannen and Coquand [2]. Let $\mathcal{V}$ be a countably infinite set of *type variables*, $\mathcal{X}$ a countably infinite set of *term variables* (for short, variables), and $\mathcal{B}$ a set of *base types*.

Definition 2.1 The set $\mathcal{T}$ of *second-order polymorphic type expressions* (for short, *types*) is defined inductively as follows:

$t \in \mathcal{T}$, whenever $t \in \mathcal{V}$,
$\sigma \in \mathcal{T}$, whenever $\sigma \in \mathcal{B}$,
$(\sigma \to \tau) \in \mathcal{T}$, whenever $\sigma, \tau \in \mathcal{T}$, and
$\forall t.\, \sigma \in \mathcal{T}$, whenever $t \in \mathcal{V}$ and $\sigma \in \mathcal{T}$.

In omitting parentheses, we follow the usual convention that $\to$ associates to the right, that is, $\sigma_1 \to \sigma_2 \to \ldots \sigma_{n-1} \to \sigma_n$ abbreviates $(\sigma_1 \to (\sigma_2 \to \ldots (\sigma_{n-1} \to \sigma_n)\ldots))$. The subset of $\mathcal{T}$ consisting of the type expressions built up inductively from $\mathcal{B}$ using only the type constructor $\to$ is called the set of *simple types*. Obviously, simple types cannot contain type variables or quantifiers.

Next, we define polymorphic raw terms. Let Σ be a set of constant symbols and $Type\colon \Sigma \to \mathcal{T}$ a function assigning a closed polymorphic type (i.e., a type expression containing no free type variable) to every symbol in Σ.

Definition 2.2 The set $\mathcal{P}\Lambda$ of *polymorphic lambda raw Σ-terms* (for short, polymorphic 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 $M, N \in \mathcal{P}\Lambda$,
$(\lambda x\colon \sigma.\, M) \in \mathcal{P}\Lambda$, 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 $M \in \mathcal{P}\Lambda$,
$(\Lambda t.\, M) \in \mathcal{P}\Lambda$, whenever $t \in \mathcal{V}$ 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, $M_1 M_2 \ldots M_{n-1} M_n$ is an abbreviation for $((\ldots(M_1 M_2)\ldots M_{n-1})M_n)$. The subset of $\mathcal{P}\Lambda$ consisting of all terms built up using only the first four clauses of definition 2.2 and only simple types is called the set of *simply typed raw terms*.

Every polymorphic raw term corresponds to an untyped lambda term obtained by erasing the types. This technique will be useful in proving strong normalization for the second-order polymorphic lambda calculus. Thus, we define untyped lambda terms and the Erase function as follows.

Let Σ be a set of constant symbols.

Definition 2.3 The set Λ of *untyped lambda Σ-terms* (for short, lambda terms) is defined inductively as follows:

$c \in \Lambda$, whenever $c \in \Sigma$,
$x \in \Lambda$, whenever $x \in \mathcal{X}$,
$(MN) \in \Lambda$, whenever $M, N \in \Lambda$,
$(\lambda x.\, M) \in \Lambda$, whenever $x \in \mathcal{X}$ and $M \in \Lambda$.

The function $Erase\colon \mathcal{P}\Lambda \to \Lambda$ is defined recursively as follows:

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

Not all polymorphic raw terms are acceptable, only those that type-check. In order to type-check a raw term, one needs to make assumptions about the types of the (term) variables free in M. This can be done by introducing type assignments. Then, type-checking a raw term is done using a proof system working on certain expressions called type 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.

3 Substitution and α-equivalence

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

Definition 3.1 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 $dom(\varphi) = \{x_1, \ldots, x_n\}$ and $\varphi(x_i) = u_i$ for every i, $1 \leq i \leq n$, the substitution φ is also denoted by $[u_1/x_1, \ldots, u_n/x_n]$.

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 3.2 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:

$$\begin{aligned}
\widehat{\varphi}(x) &= \varphi(x), \qquad x \in \mathcal{X},\\
\widehat{\varphi}(t) &= \varphi(t), \qquad t \in \mathcal{V},\\
\widehat{\varphi}(f) &= f, \qquad f \in \Sigma,\\
\widehat{\varphi}(\sigma) &= \sigma, \qquad \sigma \in \mathcal{B},\\
\widehat{\varphi}(\sigma \to \tau) &= \widehat{\varphi}(\sigma) \to \widehat{\varphi}(\tau), \qquad \sigma, \tau \in \mathcal{T},\\
\widehat{\varphi}(\forall t.\, \sigma) &= \forall t.\, \widehat{\varphi_{-t}}(\sigma), \qquad \sigma \in \mathcal{T}, t \in \mathcal{V},\\
\widehat{\varphi}(PQ) &= \widehat{\varphi}(P)\widehat{\varphi}(Q), \qquad P, Q \in \mathcal{P}\Lambda,\\
\widehat{\varphi}(M\sigma) &= \widehat{\varphi}(M)\widehat{\varphi}(\sigma), \qquad M \in \mathcal{P}\Lambda, \sigma \in \mathcal{T},\\
\widehat{\varphi}(\lambda x\colon \sigma.\, M) &= \lambda x\colon \widehat{\varphi}(\sigma).\, \widehat{\varphi_{-x}}(M), \qquad M \in \mathcal{P}\Lambda, \sigma \in \mathcal{T}, x \in \mathcal{X},\\
\widehat{\varphi}(\Lambda t.\, M) &= \Lambda t.\, \widehat{\varphi_{-t}}(M), \qquad M \in \mathcal{P}\Lambda, t \in \mathcal{V}.
\end{aligned}$$

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 $\sigma[\sigma_1/t_1, \ldots, \sigma_n/t_n]$).

A substitution of untyped lambda terms is defined as a function $\varphi\colon \mathcal{X} \to \Lambda$ with finite domain.

Definition 3.3 The extension $\widehat{\varphi}\colon \Lambda \to \Lambda$ of a substitution $\varphi\colon \mathcal{X} \to \Lambda$ is defined recursively as follows:

$$\begin{aligned}
\widehat{\varphi}(x) &= \varphi(x), \qquad x \in \mathcal{X},\\
\widehat{\varphi}(f) &= f, \qquad f \in \Sigma,\\
\widehat{\varphi}(PQ) &= \widehat{\varphi}(P)\widehat{\varphi}(Q), \qquad P, Q \in \Lambda,\\
\widehat{\varphi}(\lambda x.\, M) &= \lambda x.\, \widehat{\varphi_{-x}}(M), \qquad M \in \Lambda, x \in \mathcal{X}.
\end{aligned}$$

The notational conventions used for substitutions on polymorphic raw terms are also used for substitutions on untyped terms. In particular, if M is any untyped lambda term, we also denote $\widehat{\varphi}(M)$ as $\varphi(M)$.

We now have to face the painful task of dealing with α-conversion and variable capture in substitutions. The motivation for α-conversion is that we want terms that only differ by the names of their bound variables to have the same behavior (and meaning).

Example 3.4 For example, we would like to consider the terms $M_1 = \Lambda t_1.\, \lambda x_1{:}t_1.\, x_1$ and $M_2 = \Lambda t_2.\, \lambda x_2{:}t_2.\, x_2$ to be equivalent. They both represent the "polymorphic identity function". This can be handled by defining an equivalence relation $\equiv_\alpha$ which relates terms that differ only by renaming of their bound variables.

Definition 3.5 The relation $\longrightarrow_\alpha$ of *immediate α-reduction* is defined by the following proof system:

Axioms:

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

Inference Rules:

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

$$\frac{\sigma \longrightarrow_\alpha \tau}{\forall t.\, \sigma \longrightarrow_\alpha \forall t.\, \tau}$$

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

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

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

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

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 $\longrightarrow_\alpha$ ($\equiv_\alpha = (\longrightarrow_\alpha \cup \longrightarrow_\alpha^{-1})^*$).[1]

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

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

Lemma 3.6 The following properties hold:

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

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 $\Lambda t.\, M_1 \equiv_\alpha \Lambda t.\, M_2$.

Proof. Straightforward by induction. □

Lemma 3.6 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:

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

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 $[\sigma]$.

In view of the above, α-equivalence should also be extended to substitutions as well. By this, we mean that we should expect that if $M_1 \equiv_\alpha M_2$ and $N_1 \equiv_\alpha N_2$, then $M_1[N_1/x] \equiv_\alpha M_2[N_2/x]$. However, this may not be true due to the problem of variable capture. As an illustration, let $M_1 = \lambda y{:}\sigma.\, x$, $M_2 = \lambda w{:}\sigma.\, x$, and $N_1 = N_2 = w$. We have $M_1 \equiv_\alpha M_2$ and of course $N_1 \equiv_\alpha N_2$, but $M_1[N_1/x] = (\lambda y{:}\sigma.\, x)[w/x] = \lambda y{:}\sigma.\, w$ and $M_2[N_2/x] = (\lambda w{:}\sigma.\, x)[w/x] = \lambda w{:}\sigma.\, w$. However, $\lambda y{:}\sigma.\, w$ and $\lambda w{:}\sigma.\, w$ are **not** α-equivalent. What went wrong is that when w was substituted for x in $M_2 = \lambda w{:}\sigma.\, x$, it became bound in the result $\lambda w{:}\sigma.\, w$. We say that w was *captured*. In order to fix this problem, we need to only allow substitutions that do not cause variable capture. This can be achieved in several ways. One solution is to redefine substitution so that bound variables involved in variable capture are renamed. Essentially, α-conversion is incorporated into substitution. We find this solution rather unclean, and instead, we will define when a term is safe for a substitution, and use α-conversion to get around variable capture.

Given a substitution $\varphi{:}\,\mathcal{X} \cup \mathcal{V} \to \mathcal{P}\Lambda \cup \mathcal{T}$, 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 3.7 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:

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

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 3.8 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 3.9 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 3.10 (i) If $(\lambda x\colon \sigma_1.\, M_1)N_1 \equiv_\alpha (\lambda y\colon \sigma_2.\, M_2)N_2$, M_1 is safe for $[N_1/x]$, and M_2 is safe for $[N_2/y]$, then $M_1[N_1/x] \equiv_\alpha M_2[N_2/y]$. (ii) If $(\Lambda t.\, M_1)\tau_1 \equiv_\alpha (\Lambda v.\, M_2)\tau_2$, 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 type-checking raw terms.

4 Type Assignments and Type-Checking

First, we need the notion of a type assignment.

Definition 4.1 A *type assignment* is a partial function $\Delta\colon \mathcal{X} \to \mathcal{T}$ with a finite domain denoted as $dom(\Delta)$. Thus, a type assignment Δ is a finite set of pairs $\{x_1\colon \sigma_1, \ldots, x_n\colon \sigma_n\}$ where the variables are pairwise distinct. Given a type assignment Δ and a pair $\langle x, \sigma\rangle$ where $x \in \mathcal{X}$ and $\sigma \in \mathcal{T}$, provided that $x \notin dom(\Delta)$, we write $\Delta, x\colon \sigma$ for $\Delta \cup \{\langle x, \sigma\rangle\}$.

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

Definition 4.2 A *typing judgment of type* σ is an expression of the form $\Delta \triangleright M\colon \sigma$, where Δ is a type assignment, M is a polymorphic raw term, and σ is a type.

The proof system for deriving typing judgments is the following:

Axioms:

$$\Delta \triangleright c\colon Type(c), \qquad c \in \Sigma \qquad (\textit{constants})$$

$$\Delta \triangleright x\colon \Delta(x), \qquad x \in dom(\Delta) \qquad (\textit{variables})$$

Inference Rules:

$$\frac{\Delta \triangleright M\colon \sigma \to \tau \qquad \Delta \triangleright N\colon \sigma}{\Delta \triangleright MN\colon \tau} \qquad (\textit{application})$$

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

$$\frac{\Delta \triangleright M\colon \forall t.\ \sigma}{\Delta \triangleright M\tau\colon \sigma[\tau/t]} \qquad (\textit{type application})$$

where σ is safe for the substitution $[\tau/t]$

$$\frac{\Delta \triangleright M\colon \sigma}{\Delta \triangleright (\Lambda t.\ M)\colon \forall t.\ \sigma} \qquad (\textit{type abstraction})$$

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

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 under*

Δ iff there exists some type σ such that $\Delta \triangleright M\colon\sigma$ is derivable. Finally, 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.

Note that the terms M_1 and M_2 of example 3.4 both type-check, since it is easily shown that $\vdash \triangleright \Lambda t_1.\, \lambda x_1\colon t_1.\, x_1\colon \forall t_1.\, (t_1 \to t_1)$ and $\vdash \triangleright \Lambda t_2.\, \lambda x_2\colon t_2.\, x_2\colon \forall t_2.\, (t_2 \to t_2)$.

This example suggests that α-equivalence should be extended to typing judgments as well. Indeed, $\triangleright \Lambda t_1.\, \lambda x_1\colon t_1.\, x_1\colon \forall t_1.\, (t_1 \to t_1)$ and $\triangleright \Lambda t_2.\, \lambda x_2\colon t_2.\, x_2\colon \forall t_3.\, (t_3 \to t_3)$ should be considered equivalent. We extend $\equiv_\alpha$ to typing judgments as follows.

Definition 4.3 First, we define α-equivalence of type assignments. Given two type assignments $\Delta = \{x_1\colon\sigma_1, \ldots, x_n\colon\sigma_n\}$ and $\Delta' = \{x_1\colon\sigma'_1, \ldots, x_n\colon\sigma'_n\}$, we write $\Delta \equiv_\alpha \Delta'$ iff $\sigma_i \equiv_\alpha \sigma'_i$ for all i, $1 \le i \le n$. Two type 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 $\sigma \equiv_\alpha \sigma'$.

Following Hindley and Seldin, we also add the following inference rules to the proof system of definition 4.2.

$$\frac{\Delta \triangleright M\colon\sigma \qquad \Delta \equiv_\alpha \Delta'}{\Delta' \triangleright M\colon\sigma} \quad \text{where } \Delta \triangleright M\colon\sigma \text{ is an axiom} \qquad (\equiv'_\alpha)$$

$$\frac{\Delta \triangleright M\colon\sigma \qquad \sigma \equiv_\alpha \sigma'}{\Delta \triangleright M\colon\sigma'} \quad \text{where } \Delta \triangleright M\colon\sigma \text{ is an axiom} \qquad (\equiv''_\alpha)$$

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

It is obvious that α-equivalence of type-judgments is an equivalence relation. The following lemma shows that it is legitimate to work with equivalence classes of terms and types modulo α-equivalence.

Lemma 4.4 If two type judgments $\Delta \triangleright M\colon\sigma$ and $\Delta' \triangleright M'\colon\sigma'$ are α-equivalent and there is a proof $\vdash \Delta \triangleright M\colon\sigma$, then there is a proof $\vdash \Delta' \triangleright M'\colon\sigma'$ (in the extended system of definition 4.3).

Proof. A tedious induction on the depth of proof trees with many cases corresponding to the definition of α-equivalence. □

In view of lemma 3.6, lemma 3.9, and lemma 4.4, it is legitimate to identify terms and types that are α-equivalent, and we will do so in the future. Effectively, we will be working with α-equivalence classes. The same kind of treatment applies to the untyped lambda calculus in an obvious fashion.

The following lemma shows that the notion of substitution given by definition 3.2 is type preserving when applied to terms that type-check.

Given a type assignment $\Delta = \{x_1{:}\,\sigma_1, \ldots, x_n{:}\,\sigma_n\}$ and a substitution $\theta{:}\,\mathcal{V} \to \mathcal{T}$, let $\theta(\Delta) = \{x_1{:}\,\theta(\sigma_1), \ldots, x_n{:}\,\theta(\sigma_n)\}$.

Lemma 4.5 (i) For any term M and any substitution $\varphi{:}\,\mathcal{X} \to \mathcal{P}\Lambda$ s.t. $FV(M) \subseteq dom(\varphi)$, if M type-checks with proof $\vdash \Gamma \triangleright M{:}\,\tau$, and there is some Δ such that for every $x \in FV(M)$, $\varphi(x)$ type-checks with some proof $\vdash \Delta \triangleright \varphi(x){:}\,\Gamma(x)$ and M is safe for φ, then $\varphi(M)$ type-checks with some proof $\vdash \Delta \triangleright \varphi(M){:}\,\tau$. (ii) For any term $M \in \mathcal{P}\Lambda$ and any substitution $\theta{:}\,\mathcal{V} \to \mathcal{T}$, if M type-checks with proof $\vdash \Delta \triangleright M{:}\,\sigma$ and Δ, M, and σ are safe for θ, then, $\theta(M)$ type-checks with some proof $\vdash \theta(\Delta) \triangleright \theta(M){:}\,\theta(\sigma)$.

Proof. Both proofs are tedious but not difficult. They are left to the courageous readers. □

Definition 4.6 Given any contexts Γ, Δ, a *type-preserving substitution* is a function $\varphi : dom(\Gamma) \to \mathcal{P}\Lambda$ such that, for every $x \in dom(\Gamma)$, $\vdash \Delta \triangleright \varphi(x) : \Gamma(x)$. Such a substitution is denoted as $\varphi : \Gamma \to \Delta$.

Another useful and tedious lemma shows that substitution of raw terms is preserved under erasing.

Lemma 4.7 (i) For all raw $M, N \in \mathcal{P}\Lambda$, $Erase(M[N/x]) = Erase(M)[Erase(N)/x]$. (ii) For every raw term $M \in \mathcal{P}\Lambda$ and type $\tau \in \mathcal{T}$, $Erase(M[\tau/t]) = Erase(M)$.

Proof. The proof proceeds by cases and it is tedious but not difficult. □

We are finally ready to define the notion of reduction.

5 Reduction and Conversion

It is convenient to define reduction on raw terms, and verify that it is type-preserving when applied to a term that type-checks. Actually, we will define reduction on α-equivalence classes and use lemma 3.6 and corollary 3.10 to ensure that this definition makes sense. Thus, in what follows, terms and types are identified with their $\equiv_\alpha$-equivalence class. 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 $[(\Lambda t.\ M)\tau]$.

Definition 5.1 The relation $\longrightarrow_{\lambda^\forall}$ 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:

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

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

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

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

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

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

$$\frac{M \longrightarrow_r N}{M\sigma \longrightarrow_r N\sigma} \qquad \text{for all } \sigma \in \mathcal{T} \qquad (\textit{type congruence})$$

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

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

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

The following lemma shows that reduction is type-preserving.

Lemma 5.2 Given any two raw terms $M, N \in \mathcal{P}\Lambda$, if M type-checks with proof $\vdash \Delta \triangleright M{:}\sigma$ and $M \longrightarrow_{\lambda^\forall} N$, then N also type-checks with some proof $\vdash \Delta \triangleright N{:}\sigma$.

Proof. Again, the proof proceeds by cases and it is tedious but not difficult. □

It is evident that lemma 5.2 also holds for $\overset{*}{\longrightarrow}_{\lambda^\forall}$.

Reduction and conversion can also be defined for the untyped lambda calculus. The reduction relation $\overset{*}{\longrightarrow}_\lambda$ is defined by only retaining the β and η reduction axioms of definition 5.1, and the inference rules not involving types. The notion of conversion $\overset{*}{\longleftrightarrow}_\lambda$ is defined from $\longrightarrow_\lambda$ in the usual way. It is easy to see that an analog of lemma 3.9 holds

for untyped λ-terms. When we need to distinguish between a λ-reduction step and a $\lambda^\forall$-reduction step, we add the name of the calculus as a subscript. For example, $\longrightarrow_{\beta,\lambda}$ is a β-conversion step in the untyped lambda calculus λ, whereas $\longrightarrow_{\beta,\lambda^\forall}$ is a β-conversion step in the polymorphic lambda calculus $\lambda^\forall$.

We have the following lemma showing how a reduction step $\longrightarrow_{\lambda^\forall}$ is mapped to a reduction step $\stackrel{*}{\longrightarrow}_\lambda$ by the Erase function.

Lemma 5.3 Let $M, N \in \mathcal{P}\Lambda$ be two raw terms. If $M \longrightarrow_{\beta,\lambda^\forall} N$ or $M \longrightarrow_{\eta,\lambda^\forall} N$, then $Erase(M) \longrightarrow_{\beta,\lambda} Erase(N)$ or $Erase(M) \longrightarrow_{\eta,\lambda^\forall} Erase(N)$ respectively. If $M \longrightarrow_{\tau\beta,\lambda^\forall} N$ or $M \longrightarrow_{\tau\eta,\lambda^\forall} N$, then $Erase(M) = Erase(N)$.

Proof. Another tedious but not difficult proof using lemma 4.7. □

Definition 5.4 Let $\longrightarrow \subseteq A \times A$ be a binary relation on a set A, and $\stackrel{*}{\longrightarrow}$ be the reflexive and transitive closure of $\longrightarrow$. An element $a \in A$ is *strongly normalizing* (with respect to $\longrightarrow$, for short SN) iff there are no infinite sequences $\langle a_0, a_1, \ldots, a_n, \ldots\rangle$ such that $a_0 = a$ and $a_n \longrightarrow a_{n+1}$ for all $n \geq 0$. We say that $\longrightarrow$ is *Noetherian* iff every $a \in A$ is strongly normalizing (with respect to $\longrightarrow$). We say that $\longrightarrow$ is *locally confluent* iff for all $a, a_1, a_2 \in A$, if $a \longrightarrow a_1$ and $a \longrightarrow a_2$, then there is some $a_3 \in A$ such that $a_1 \stackrel{*}{\longrightarrow} a_3$ and $a_2 \stackrel{*}{\longrightarrow} a_3$. We say that $\longrightarrow$ is *confluent* iff for all $a, a_1, a_2 \in A$, if $a \stackrel{*}{\longrightarrow} a_1$ and $a \stackrel{*}{\longrightarrow} a_2$, then there is some $a_3 \in A$ such that $a_1 \stackrel{*}{\longrightarrow} a_3$ and $a_2 \stackrel{*}{\longrightarrow} a_3$. Let $\longleftrightarrow = \longrightarrow \cup \longrightarrow^{-1}$. We say that $\longrightarrow$ is *Church-Rosser* iff for all $a_1, a_2 \in A$, if $a_1 \stackrel{*}{\longleftrightarrow} a_2$, then there is some $a_3 \in A$ such that $a_1 \stackrel{*}{\longrightarrow} a_3$ and $a_2 \stackrel{*}{\longrightarrow} a_3$.

It is well known (Huet [17]) that a Noetherian relation is confluent iff it is locally confluent and that a relation is confluent iff it is Church-Rosser. We say that a lambda calculus X ($X \in \{\lambda, \lambda^\forall\}$) is Noetherian, locally confluent, or confluent iff the relation $\longrightarrow_X$ associated with X has the corresponding property. We say that it is *canonical* iff it is Noetherian and confluent.

Lemma 5.3 will be used to show that a polymorphic raw term is strongly normalizing iff its type erasure $Erase(M)$ is strongly normalizing.

Lemma 5.5 Let $M, N \in \mathcal{P}\Lambda$ be two raw terms. If $M \longrightarrow_{\tau\beta,\lambda^\forall} N$ or $M \longrightarrow_{\tau\eta,\lambda^\forall} N$, then N has one less type abstraction than M.

Proof. Immediate by the definitions. □

We now have the important "erasing trick" lemma.

Lemma 5.6 Let $M \in \mathcal{P}\Lambda$ be any raw term. If there is an infinite sequence of $\longrightarrow_{\lambda^\forall}$ reductions from M, then there is an infinite sequence of $\longrightarrow_\lambda$ reductions from $Erase(M)$.

Proof. First, observe that any infinite sequence of $\longrightarrow_{\lambda^\forall}$ reductions from M must contain a subsequence consisting of infinitely many $\longrightarrow_{\beta,\lambda^\forall}$ or $\longrightarrow_{\eta,\lambda^\forall}$ reductions, since otherwise some term in this sequence would be the head of an infinite sequence of $\longrightarrow_{\tau\beta,\lambda^\forall}$ or $\longrightarrow_{\tau\eta,\lambda^\forall}$ reductions, contradicting lemma 5.5. But then, by lemma 5.3, the infinite sequence of $\longrightarrow_{\lambda^\forall}$ reductions from M maps by $Erase$ to an infinite sequence of $\longrightarrow_\lambda$ reductions from $Erase(M)$. □

Thus, lemma 5.6 implies that a polymorphic raw term is strongly normalizing iff its type erasure $Erase(M)$ is strongly normalizing. In the next section, it will be shown that if M is a raw term that type-checks, then $Erase(M)$ is strongly normalizing, thus showing that M itself is strongly normalizing.[2] The following lemma will also be needed.

Lemma 5.7 Let $M_1, M_2, N_1, N_2 \in \Lambda$ be untyped lambda terms. If $M_1 \stackrel{*}{\longrightarrow}_\lambda M_2$ and $N_1 \stackrel{*}{\longrightarrow}_\lambda N_2$, then $M_1[N_1/x] \stackrel{*}{\longrightarrow}_\lambda M_2[N_2/x]$.

Proof. We use two inductions, one on the structure of M_1, and the other on the length of reduction sequences. The details are quite tedious. □

6 An Untyped Version of The Candidates of Reducibility

Originally, the "candidats de reductibilité" were defined by Girard as sets of typed (polymorphic) lambda terms with certain closure properties (Girard 1970 [9], Girard 1972 [10]). Soon after Girard, Tait observed that Girard's brilliant device could be simplified if the candidates are defined as certain sets of untyped lambda terms, and if a certain "erasing trick" is used (Tait 1973 [37]).[3] Roughly thirteen years after Tait, Mitchell independently noticed that an untyped version of the candidates is more flexible to work with, and he gave his own version generalizing Tait's version (Mitchell 1986 [24]).

Before we proceed with the technical details, we will attempt to reveal some of the intuitions underlying the proof. We believe that this is best accomplished by first restricting our attention to the simply typed lambda calculus. The problem is to show that every simply typed term that type-checks is strongly normalizing.

---

[2] It is interesting to note that Tait [37] used an erasing trick in his proof, but the definition of his erasing function is different from the one given here.

[3] Interestingly, the erasing trick was known to Girard himself, since it appears in his thesis, Girard 1972 [10]. However, he did not make use of it in his proof of strong normalization.

A natural way of attacking the problem is to attempt a proof by induction on the size of terms. The base case of a (typed) variable or a (typed) constant works out nicely, since no reduction applies. The case of a lambda abstraction $M = \lambda x{:}\,\sigma.\, N$ also works fine, because if any reduction applies to M, then it must in fact apply to N, and N has strictly smaller size than M, so the induction hypothesis applies. Difficulties arise in case of an application of the form $(\lambda x{:}\,\sigma.\, M)N$. The induction hypothesis applies to both $\lambda x{:}\,\sigma.\, M$ and N, but there is another possibility of reduction, namely $(\lambda x{:}\,\sigma.M)N \longrightarrow_\beta M[N/x]$, and unfortunately, $M[N/x]$ is not necessarily strictly smaller than $(\lambda x{:}\,\sigma.\, M)N$. Tait's clever solution for overcoming this problem (Tait [36]) is essentially to strengthen the induction hypothesis. He does so by defining by induction on types the class of "computable" (or "reducible") terms. Let us denote the set of simply typed terms of type σ (that type-check) as ST_σ, and the subset of all SN terms in ST_σ as SN_σ. For every simple type σ, the set $[\![\sigma]\!]$ is a subset of ST_σ defined as follows:

(1) $[\![\sigma]\!] = SN_\sigma$, for every base type σ;

(2) $[\![(\sigma \to \tau)]\!] = \{M \in ST_{(\sigma\to\tau)} \mid \forall N \in [\![\sigma]\!],\ MN \in [\![\tau]\!]\}$.

One can then prove by induction on types that

$$[\![\sigma]\!] \subseteq SN_\sigma, \tag{a}$$

that is, all terms in $[\![\sigma]\!]$ are SN. In order to finish the proof, it is sufficient to show that

$$ST_\sigma \subseteq [\![\sigma]\!], \tag{b}$$

since (a) and (b) together prove that $ST_\sigma = SN_\sigma$.

The way to prove (b) is to proceed by induction on the size of terms. Note that the problematic case of an application MN (where $M \in ST_{\sigma\to\tau}$) is now easy: by the induction hypothesis, $M \in [\![(\sigma \to \tau)]\!]$ and $N \in [\![\sigma]\!]$, and by the definition of $[\![(\sigma \to \tau)]\!]$, we have $MN \in [\![\tau]\!]$. This time, the difficult case is to prove that for every term of the form $\lambda x{:}\,\sigma.\, M$, where $M \in ST_\tau$, $\lambda x{:}\,\sigma.\, M \in [\![(\sigma \to \tau)]\!]$. We need to show that for every $N \in [\![\sigma]\!]$, $(\lambda x{:}\,\sigma.\, M)N \in [\![\tau]\!]$. Since $(\lambda x{:}\,\sigma.\, M)N \longrightarrow_\beta M[N/x]$, if we could show that $M[N/x] \in [\![\tau]\!]$ and that whenever $M[N/x] \in [\![\tau]\!]$, then $(\lambda x{:}\,\sigma.\, M)N \in [\![\tau]\!]$, we would be able to conclude.

This can be done, but it is necessary to strengthen the induction hypothesis. Roughly, the idea is show that for every term $M \in ST_\sigma$, for every substitution φ assigning to each variable of type τ in M some term in $[\![\tau]\!]$, then $\varphi(M) \in [\![\sigma]\!]$. Then, we have our result by choosing φ to be the identity substitution.

Now, it turns out that the sets of the form $[\![\sigma]\!]$ have certain closure properties (properties (S1), (S2) of definition 6.9) that make the various induction steps go through. Girard's

achievement was to generalize Tait's method (sketched above) to the polymorphic lambda calculus. This generalization requires a major leap forward, because in trying to extend the definition of the sets $[\![\sigma]\!]$ to the polymorphic types, one faces two (related) problems:

(1) What to assign to the type variables;

(2) What to assign to a type of the form $\forall t.\, \sigma$.

Girard's solution is the invention of the candidates of reducibility. Girard defines a family $\mathcal{C}$ of sets of typed terms satisfying certain closure conditions akin to conditions (S1), (S2) mentioned above. One of these conditions is that each set in $\mathcal{C}$ is a set of strongly normalizing terms. The other conditions amount to inductive conditions. The sets in $\mathcal{C}$ are called *candidates of reducibility*. Then, the solution is to assign arbitrary candidates to the type variables. More specifically, the sets $[\![\sigma]\!]$ are parameterized by an assignment η of sets from $\mathcal{C}$ to the type variable (actually, things are a bit more complicated, because in Girard, the candidates are typed). Thus, these sets are of the form $[\![\sigma]\!]\eta$, where η is an assignment of candidates to the typed variables. The set assigned to a type of the form $\forall t.\, \sigma$ is

$$[\![\forall t.\, \sigma]\!]\eta = \bigcap_{C \in \mathcal{C}} [\![\sigma]\!]\eta[t := C],$$

where $\eta[t := C]$ is like the assignment η, except that t is now assigned the candidate C (again, in Girard's setting, things are more complicated due to the types, and what we are describing holds for the untyped version of the candidates).

Now comes Girard's trick: Because the sets in the family $\mathcal{C}$ (the candidates) satisfy some well chosen conditions, for every assignment η, each set $[\![\sigma]\!]\eta$ also belongs to $\mathcal{C}$! This is remarkable, because $[\![\forall t.\, \sigma]\!]\eta$ is defined in terms of all the candidates in $\mathcal{C}$, and consequently it is defined in terms of itself. This is a splendid instance of impredicativity. Another important fact is that for every type σ, the set of (polymorphic) strongly normalizing terms of type σ that type-check is a candidate of reducibility (i.e., belongs to $\mathcal{C}$).

The main lines of Girard's proof of strong normalization are as follows.

(1) Define the family $\mathcal{C}$ of candidates of reducibility so that they consist of sets of strongly normalizing terms;

(2) Define the sets $[\![\sigma]\!]\eta$;

(3) Prove "Girard's trick", that is, prove that $[\![\sigma]\!]\eta \in \mathcal{C}$ for every type σ and assignment η;

(4) Prove that the set of (polymorphic) strongly normalizing terms of type σ is a candidate of reducibility (for every type σ);

(5) Prove that for every (polymorphic) term M of type σ that type-checks, $M \in [\![\sigma]\!]\eta$, for every assignment η.

By choosing the assignment η so that it assigns to the type variables the sets of terms that are strongly normalizing, we obtain the desired result: every term that type-checks is strongly normalizing.

It should be noted that in order to prove (5), one needs a substantially strengthened induction hypothesis (see lemma 6.8).

We will now prove strong normalization using an untyped version of the candidates of reducibility, following Mitchell and Tait. We go one step further and define a kind of abstract version of the candidates of reducibility. This way, it is easier to pinpoint the ingredients that are crucial to proofs using this concept. We first describe what we referred to as "Girard's trick".

Definition 6.1 Given two sets S and T of (untyped) lambda terms, we let $[S \to T]$ be the set of (untyped) lambda terms defined as follows:

$$[S \to T] = \{M \in \Lambda \mid \forall N \in S,\ MN \in T\}.$$

We refer to the operation $\to$ on sets of lambda terms defined above as the *function space constructor*.

Definition 6.2 Let $\mathcal{C}$ be a nonempty family of sets of (untyped) lambda terms having the following properties:

(1) Every $C \in \mathcal{C}$ is nonempty.

(2) $\mathcal{C}$ is closed under the function space constructor.

(3) Given any $\mathcal{C}$-indexed family $(A_C)_{C\in\mathcal{C}}$ of sets in $\mathcal{C}$, then $\bigcap_{C\in\mathcal{C}} A_C \in \mathcal{C}$.

A family satisfying the above conditions is called a *$\mathcal{T}$-closed family*.[4]

We shall prove shortly that such families exist.

Let $\mathcal{C}$ be a $\mathcal{T}$-closed family. Given any assignment $\eta\colon \mathcal{B}\cup\mathcal{V} \to \mathcal{C}$ of sets in $\mathcal{C}$ to the type variables and the base types, we can associate certain sets of lambda terms to the types inductively as explained below. In the following definition, given any set $C \in \mathcal{C}$ and any type variable t, $\eta[t := C]$ denotes the assignment such that, for all $v \in \mathcal{V}$,

$$\eta[t := C](v) = \begin{cases} C, & \text{if } v = t; \\ \eta(v), & \text{if } v \neq t. \end{cases}$$

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

Definition 6.3 Given any assignment $\eta: \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, for every type σ, the set $[\![\sigma]\!]\eta$ is defined as follows:

$[\![t]\!]\eta = \eta(t)$, whenever $t \in \mathcal{B} \cup \mathcal{V}$,

$[\![(\sigma \to \tau)]\!]\eta = [[\![\sigma]\!]\eta \to [\![\tau]\!]\eta]$ (where $\to$ is the function space contructor defined earlier);

$[\![\forall t.\, \sigma]\!]\eta = \bigcap_{C \in \mathcal{C}} [\![\sigma]\!]\eta[t := C]$.

The following technical lemma will be useful later.

Lemma 6.4 Given any two assignments $\eta_1: \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$ and $\eta_2: \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, for every type σ, if η_1 and η_2 agree on $\mathcal{FV}(\sigma)$ and $\mathcal{B}$, then $[\![\sigma]\!]\eta_1 = [\![\sigma]\!]\eta_2$.

Proof. Easy induction on the structure of types. □

The next result constitutes the essence of "Girard's trick".

Lemma 6.5 (Girard) If $\mathcal{C}$ is a $\mathcal{T}$-closed family, for every assignment $\eta: \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, for every type σ, then $[\![\sigma]\!]\eta \in \mathcal{C}$.

Proof. The lemma is proved by induction on the structure of types. The case of a type variable or a base type t is obvious since by the definition $[\![t]\!]\eta = \eta(t)$.

For a type $(\sigma \to \tau)$, by the induction hypothesis we have $[\![\sigma]\!]\eta \in \mathcal{C}$ and $[\![\tau]\!]\eta \in \mathcal{C}$, and by condition (2) of $\mathcal{T}$-closed families, we also have $[[\![\sigma]\!]\eta \to [\![\tau]\!]\eta] \in \mathcal{C}$.

For a type $\forall t.\, \sigma$, by the induction hypothesis, for every assignment $\mu: \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, $[\![\sigma]\!]\mu \in \mathcal{C}$. Thus, for every $C \in \mathcal{C}$ we also have $[\![\sigma]\!]\eta[t := C] \in \mathcal{C}$, since $\eta[t := C]$ is an assignment. By condition (3) of $\mathcal{T}$-closed families, we have $\bigcap_{C \in \mathcal{C}} [\![\sigma]\!]\eta[t := C] \in \mathcal{C}$. □

The following technical lemma will be needed later.

Lemma 6.6 Given any two types σ, τ, for every assignment $\eta: \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, if σ is safe for $[\tau/t]$ then $[\![\sigma[\tau/t]]\!]\eta = [\![\sigma]\!]\eta[t := [\![\tau]\!]\eta]$.

Proof. Straightforward induction on the structure of σ. □

In order to use lemma 6.5 in proving properties of polymorphic lambda calculi, we need to define $\mathcal{T}$-closed families satisfying some additional properties.

Definition 6.7 We say that a family $\mathcal{C}$ of sets of untyped lambda terms is a *family of candidates of reducibility* iff it is $\mathcal{T}$-closed and satisfies the conditions listed below.[5] For every set $C \in \mathcal{C}$:

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

R1. For every variable $x \in \mathcal{X}$, $x \in C$.
For every constant $f \in \Sigma$, $f \in C$.

R2. For all $M, N \in \bigcup \mathcal{C}$, if $M[N/x] \in C$, then $(\lambda x.\, M)N \in C$.

The conditions of definition 6.7 are sufficient to prove the following crucial lemma.

Lemma 6.8 (Girard, Tait, Mitchell) Let $\mathcal{C}$ be a family of candidates of reducibility. For every proof $\vdash \Delta \triangleright M\colon \sigma$ of some polymorphic raw term $M \in \mathcal{P}\Lambda$ that type-checks, for every assignment $\eta\colon \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, for every substitution $\varphi\colon FV(M) \to \Lambda$, if $\varphi(x) \in [\![\Delta(x)]\!]\eta$ for every $x \in FV(M)$, then $\varphi(Erase(M)) \in [\![\sigma]\!]\eta$.

Proof. It proceeds by induction on the depth of the proof tree for $\Delta \triangleright M\colon \sigma$. The base case where $x \in \mathcal{X}$ is trivial since it is assumed that $\varphi(x) \in [\![\Delta(x)]\!]\eta$, and in $\vdash \Delta \triangleright x\colon \sigma$, we have $\sigma = \Delta(x)$. The case of a constant $f \in \Sigma$ is equally obvious since $\varphi(f) = f$, by lemma 6.5, $[\![Type(f)]\!]\eta \in \mathcal{C}$, and by (R1), we have $f \in C$ for every $C \in \mathcal{C}$. There are four cases for the inference rules.

Case 1.

$$\frac{\Delta \triangleright M\colon \sigma \to \tau \qquad \Delta \triangleright N\colon \sigma}{\Delta \triangleright MN\colon \tau} \quad (\textit{application})$$

By the induction hypothesis, we have $\varphi(Erase(M)) \in [\![\sigma \to \tau]\!]\eta$ and $\varphi(Erase(N)) \in [\![\sigma]\!]\eta$. By the definition of $[\![\sigma \to \tau]\!]\eta$, we have $\varphi(Erase(M))\varphi(Erase(N)) \in [\![\tau]\!]\eta$. But $\varphi(Erase(M))\varphi(Erase(N)) = \varphi(Erase(MN))$, and so $\varphi(Erase(MN)) \in [\![\tau]\!]\eta$.

Case 2.

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

Let $k+1$ be the depth of the proof tree $\vdash \Delta \triangleright \lambda x\colon \sigma.\, M\colon \sigma \to \tau$. We have $FV(\lambda x\colon \sigma.\, M) = FV(M) - \{x\}$. Let η be any assignment, let φ be any substitution with domain $FV(M) - \{x\}$, let N be any term in $[\![\sigma]\!]\eta = [\![(\Delta, x\colon \sigma)(x)]\!]\eta$, and assume that $\varphi(y) \in [\![\Delta(y)]\!]\eta$ for every $y \in FV(M) - \{x\}$. Now, we can always choose a representative in the $\equiv_\alpha$-class of $[\lambda x\colon \sigma.\, M]$ so that $\lambda x\colon \sigma.\, M$ is safe for φ and $FV(N)$ is disjoint from $BV(M)$. Then, we form the substitution $\varphi[x := N]$ with domain $FV(M)$, and observe that M is safe for $\varphi[x := N]$ and that $\varphi[x := N](y) \in [\![(\Delta, x\colon \sigma)(y)]\!]\eta$ for every $y \in FV(M)$. Since the induction hypothesis holds for every proof of depth $\leq k$ and for every assignment η satisfying the conditions of the lemma, we have $\varphi[x := N](Erase(M)) \in [\![\tau]\!]\eta$. By lemma 6.5, $[\![\sigma]\!]\eta \in \mathcal{C}$, and since $x \in [\![\sigma]\!]\eta$ by (R1), by choosing $N = x$, we have $\varphi(Erase(M)) \in [\![\tau]\!]\eta$. But since $\lambda x\colon \sigma.\, M$ is safe for φ, we have $x \notin FV(\varphi(y))$ for every $y \in dom(\varphi)$, and therefore $\varphi[x :=$

$N](Erase(M)) = \varphi(Erase(M))[N/x]$.[6] Thus, $\varphi(Erase(M))[N/x] \in [\![\tau]\!]\eta$. Since $N \in [\![\sigma]\!]\eta$ and $\varphi(Erase(M)) \in [\![\tau]\!]\eta$, by lemma 6.5, we have $N \in \bigcup \mathcal{C}$ and $\varphi(Erase(M)) \in \bigcup \mathcal{C}$. Thus, we can apply (R2) to $\varphi(Erase(M))[N/x] \in [\![\tau]\!]\eta$, and we have $(\lambda x.\varphi(Erase(M)))N \in [\![\tau]\!]\eta$. It is easily verified that $\lambda x.\varphi(Erase(M)) = \varphi(Erase(\lambda x{:}\,\sigma.M))$ (using the fact that $\lambda x{:}\,\sigma.M$ is safe for φ). Since $\varphi(Erase(\lambda x{:}\,\sigma.M))N \in [\![\tau]\!]\eta$ holds for every $N \in [\![\sigma]\!]\eta$, by the definition of $[\![\sigma \to \tau]\!]\eta$, we have $\varphi(Erase(\lambda x{:}\,\sigma.\, M)) \in [\![\sigma \to \tau]\!]\eta$.[7]

Case 3.

$$\frac{\Delta \triangleright M{:}\,\forall t.\,\sigma}{\Delta \triangleright M\tau{:}\,\sigma[\tau/t]} \qquad (\textit{type application})$$

By the induction hypothesis, $\varphi(Erase(M)) \in [\![\forall t.\,\sigma]\!]\eta$. Since $[\![\forall t.\,\sigma]\!]\eta = \bigcap_{C\in\mathcal{C}} [\![\sigma]\!]\eta[t := C]$, we have $\varphi(Erase(M)) \in [\![\sigma]\!]\eta[t := C]$ for every $C \in \mathcal{C}$. Since by lemma 6.5, $[\![\tau]\!]\eta \in \mathcal{C}$, by setting $C = [\![\tau]\!]\eta$, we have $\varphi(Erase(M)) \in [\![\sigma]\!]\eta[t := [\![\tau]\!]\eta]$. However, by lemma 6.6, $[\![\sigma[\tau/t]]\!]\eta = [\![\sigma]\!]\eta[t := [\![\tau]\!]\eta]$, and so $\varphi(Erase(M\tau)) = \varphi(Erase(M)) \in [\![\sigma[\tau/t]]\!]\eta$.

Case 4.

$$\frac{\Delta \triangleright M{:}\,\sigma}{\Delta \triangleright \Lambda t.\, M{:}\,\forall t.\,\sigma} \qquad (\textit{type abstraction})$$

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

Let $k+1$ be the depth of the proof tree for $\Delta \triangleright \Lambda t.\, M{:}\,\forall t.\,\sigma$. Since $t \notin \mathcal{FV}(\Delta(x))$ for every $x \in dom(\Delta) \cap FV(M)$, by lemma 6.4, we have $[\![\Delta(x)]\!]\eta = [\![\Delta(x)]\!]\eta[t := C]$ for every $C \in \mathcal{C}$. Since the induction hypothesis holds for every proof tree of depth $\leq k$, for every η, and for every φ satisfying the conditions of the lemma, it holds for every $C \in \mathcal{C}$ when applied to the proof tree $\vdash \Delta \triangleright M{:}\,\sigma$, to every $\eta[t := C]$, and to every φ such that $\varphi(x) \in [\![\Delta(x)]\!]\eta$ for every $x \in FV(M)$.[8] Thus, $\varphi(Erase(M)) \in [\![\sigma]\!]\eta[t := C]$ for every $C \in \mathcal{C}$, that is, $\varphi(Erase(\Lambda t.\, M)) = \varphi(Erase(M)) \in [\![\forall t.\,\sigma]\!]\eta$, since $[\![\forall t.\,\sigma]\!]\eta = \bigcap_{C\in\mathcal{C}} [\![\sigma]\!]\eta[t := C]$. □

Remark: It should be observed that lemma 6.8 still holds if condition (R2) of definition 6.7 is changed to:

R2′. For all $M, N \in \Lambda$, if $M[N/x] \in C$, then $(\lambda x.\, M)N \in C$.

---

[6] This subtle point seems to have been overlooked in all proofs that we have read, including Girard's original proof(s). The problem is that $\varphi[x := N](Erase(M)) = \varphi(Erase(M))[N/x]$ may be **false** if x appears in $\varphi(y)$ for some $y \in dom(\varphi)$!

[7] Observe that this step of the proof is possible because we can apply the induction hypothesis to **every** substitution of the form $\varphi[x := N]$ where N is any term in $[\![\sigma]\!]\eta$. This is why we need the universal quantification on the substitution φ in the statement of the lemma. Without it, the proof would not go through.

[8] Observe that this step of the proof is possible because we can apply the induction hypothesis to **every** assignment of the form $\eta[t := C]$ where C is any set in $\mathcal{C}$. This is why we need the universal quantification on the assignment η in the statement of the lemma. Without it, the proof would not go through.

The difference between (R2) and (R2′) is that in (R2) M and N belong to $\bigcup \mathcal{C}$, whereas in (R2′), M and N are arbitrary terms. Our motivation for using (R2) is that one can take advantage of the fact that $M, N \in \bigcup \mathcal{C}$ in establishing (R2).

By choosing φ to be the identity, lemma 6.8 implies that $Erase(M) \in [\![\sigma]\!]\eta$ for every term that type-checks. Thus, in order to prove properties of terms of the form $Erase(M)$ where M type-checks, one needs to know how to generate families of candidates of reducibility satisfying some given properties. For this, it appears that it is necessary to strengthen conditions (R1) and (R2). Girard provided sufficient conditions (Girard [9], Girard [10]). Here, we give some slightly more general conditions adapted from Mitchell ([24]) and Tait ([37]).

Definition 6.9 Let S be a nonempty set of (untyped) lambda terms. We say that S is *closed* iff whenever $Mx \in S$ where $x \in \mathcal{X}$, then $M \in S$. A subset $C \subseteq S$ is *saturated* iff the following conditions hold:[9]

S1. For every variable $x \in \mathcal{X}$, for all $n \geq 0$ and all $N_1, \ldots, N_n \in S$, $xN_1 \ldots N_n \in C$.
For every constant $f \in \Sigma$, for all $n \geq 0$ and all $N_1, \ldots, N_n \in S$, $fN_1 \ldots N_n \in C$.

S2. For all $M, N \in S$, for all $n \geq 0$ and all $N_1, \ldots, N_n \in S$, if $M[N/x]N_1 \ldots N_n \in C$, then $(\lambda x.\, M)NN_1 \ldots N_n \in C$.

The following result shows the significance of saturated subsets of a closed set of lambda terms.

Lemma 6.10 (Girard, Tait, Mitchell) Let S be a nonempty closed set of (untyped) lambda terms, let $\mathcal{C}$ be the family of all saturated subsets of S, and assume that $S \in \mathcal{C}$ (i.e. S is a saturated subset of itself). Then $\mathcal{C}$ is a family of candidates of reducibility.

Proof. Since $S \in \mathcal{C}$, $\mathcal{C}$ is nonempty. Clearly, by condition (S1) of saturated sets in the case $n = 0$, each saturated set is nonempty. Let C and D be any two saturated subsets of S. Recall that $[C \to D] = \{M \mid \forall N \in C,\ MN \in D\}$. We need to show that $[C \to D]$ is a saturated subset of S.

For every $M \in [C \to D]$, by (S1), since there is some variable $x \in C$, $Mx \in D$, and since S is closed, we have $M \in S$. Thus $[C \to D]$ is a subset of S.

Since D is saturated, by (S1) for every variable $x \in \mathcal{X}$ and for all $m \geq 0$ and all $N_1, \ldots, N_m, N \in S$, we have $xN_1 \ldots N_m N \in D$. Since this holds for every $N \in C$, we have

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

$xN_1 \ldots N_n \in [C \to D]$.[10] The second case of (S1) for a constant in Σ is similar. Thus, (S1) holds for $[C \to D]$.

For every $N \in S$, all $m \geq 0$, all $N_1, \ldots, N_m \in S$, assume that $M[N/x]N_1 \ldots N_m \in [C \to D]$. This means that for every $P \in C$, $M[N/x]N_1 \ldots N_m P \in D$. Since D is saturated, by (S2), $(\lambda x.\, M)NN_1 \ldots N_m P \in D$, and since this is true for every $P \in C$, we have $(\lambda x.\, M)NN_1 \ldots N_m \in [C \to D]$.[11] This shows that (S2) holds for $[C \to D]$.

Finally, it is clear that properties (S1) and (S2) of saturated subsets of S are closed under arbitrary intersections, and so for any $\mathcal{C}$-indexed family $(A_C)_{C \in \mathcal{C}}$ of saturated subsets of S, $\bigcap_{C \in \mathcal{C}} A_C$ is also a saturated subset of S. □

Remark. Besides implying (R1) and (R2) respectively, conditions (S1) and (S2) ensure that $[C \to D]$ also satisfies (S1) and (S2) if C and D do. It should be noted that if we are interested in the version of the candidates in which condition (R2′) is used, then lemma 6.10 holds if the clause $M, N, N_1, \ldots, N_n \in S$ in condition (S2) of definition 6.9 is changed to $N \in S$, $M, N_1, \ldots, N_n \in \Lambda$, obtaining the condition (S2′). Conditions (S1) and (S2′) are essentially Mitchell's conditions [24].

It is interesting to note that the reason why lemma 6.10 holds is that (S1) and (S2) have certain "right-invariant" properties.

Definition 6.11 Define a predicate Φ on Λ to be *right-invariant* iff for every $M, N \in \Lambda$, if $\Phi(M)$ then $\Phi(MN)$. Let $S_\Phi = \{M \in \Lambda \mid \Phi(M)\}$. A binary relation ρ on Λ is *right-invariant* iff for every $M_1, M_2, N \in \Lambda$, if $\rho(M_1, M_2)$ then $\rho(M_1 N, M_2 N)$. A set $C \subseteq \Lambda$ is *closed under* ρ iff for every $M, N \in \Lambda$, if $M \in C$ and $\rho(M, N)$, then $N \in C$.

Lemma 6.12 If $S_\Phi \subseteq C$ for every set in a family $\mathcal{C}$ and Φ is right-invariant, then S_Φ is also a subset of every set of the form $[C \to D]$ with $C, D \in \mathcal{C}$, and a subset of every intersection $\bigcap_{C \in \mathcal{C}} A_C$.

Proof. Let M be any term in Λ. We have to show that if $\Phi(M)$ holds then $M \in [C \to D]$. Since Φ is right-invariant, for every $N \in C$ we have $\Phi(MN)$. Since $S_\Phi \subseteq D$, we have $MN \in D$. This shows that $M \in [C \to D]$. Obviously, S_Φ is also a subset of every intersection $\bigcap_{C \in \mathcal{C}} A_C$. □

Lemma 6.13 If ρ is right-invariant and every set in a family $\mathcal{C}$ is closed under ρ, then every set of the form $[C \to D]$ with $C, D \in \mathcal{C}$ is closed under ρ and every intersection $\bigcap_{C \in \mathcal{C}} A_C$ is closed under ρ.

---

[10] Note how we have used the fact that (S1) holds for **all** $n \geq 0$, and applied (S1) with $n = m + 1$ for any arbitrary m. The proof would not go through if (S1) was assumed only for $n = 0$.

[11] Again, note how we have used the fact that (S2) holds for **all** $n \geq 0$.

Proof. If $M_1 \in [C \to D]$, $M_2 \in \Lambda$ and $\rho(M_1, M_2)$, by right-invariance $\rho(M_1N, M_2N)$ for every $N \in C$. Since $M_1N \in D$ and D is closed under ρ, we have $M_2N \in D$. But this shows that $M_2 \in [C \to D]$, i.e., $[C \to D]$ is closed under ρ. It is obvious that every intersection $\bigcap_{C \in \mathcal{C}} A_C$ is closed under ρ. □

Lemma 6.12 can be applied to Φ defined such that for every $M \in \Lambda$, $\Phi(M)$ iff $\exists u \in \mathcal{X} \cup \Sigma,\ \exists N_1, \ldots, N_m \in \Lambda,\ M = uN_1 \ldots N_m$. In this case Φ corresponds to (S1). Lemma 6.13 can be applied to ρ defined such that $\rho(M_1, M_2)$ iff $\exists M, N \in \Lambda,\ \exists N_1, \ldots, N_m \in \Lambda,\ M_1 = M[N/x]N_1 \ldots N_m$, and $M_2 = (\lambda x.\ M)NN_1 \ldots N_m$. In this case ρ corresponds to (S2).

Thus, it seems that the way to obtain the conditions (Si) from the conditions (Ri) is to make the (Ri) right-invariant. Indeed, this is the way to handle other type constructors, such as products and existential types. We have the following lemma generalizing lemma 6.10.

Lemma 6.14 Let S be a nonempty closed set of (untyped) lambda terms, let $\mathcal{F}_\Phi$ be a family of right-invariant predicates on Λ, and $\mathcal{F}_\rho$ a family of right-invariant binary relations on Λ. Let $\mathcal{C}$ be the family of all subsets C of S such that:

(1) $S_\Phi \subseteq C$, for every $\Phi \in \mathcal{F}_\Phi$;

(2) C is closed under ρ, for every $\rho \in \mathcal{F}_\rho$.

Then $\mathcal{C}$ is closed under the function space constructor and under intersections of the form $\bigcap_{C \in \mathcal{C}} A_C$.

Proof. Similar to lemma 6.10, using lemma 6.12 and lemma 6.13. □

After this short digression, we state the fundamental result about the method of candidates.

Theorem 6.15 (Girard, Tait, Mitchell) Let S be a nonempty closed set of (untyped) lambda terms, let $\mathcal{C}$ be the family of all saturated subsets of S, and assume that $S \in \mathcal{C}$ (i.e. S is a saturated subset of itself). For every polymorphic raw term $M \in \mathcal{P}\Lambda$, if M type-checks, then $Erase(M) \in S$.

Proof. By lemma 6.10, $\mathcal{C}$ is a family of candidates of reducibility. We now apply lemma 6.8 to any assignment (for example, the constant assignment with value S) and to the identity substitution, which is legitimate since by (S1), every variable belongs to every saturated set.[12]

[12] Actually, given any term M, we may need to perform some α-renaming on M to get an M' such that M' is safe for the identity substitution. Lemma 6.8 then yields the fact that $Erase(M') \in S$. But S is closed under $\equiv_\alpha$, and so $Erase(M) \in S$.

Thus, in order to apply theorem 6.15, one needs to have useful examples of closed sets S that are saturated. This is the purpose of the next lemma.

Lemma 6.16 (Girard, Tait, Mitchell) (i) The set SN_β of (untyped) lambda terms that are strongly normalizing under β-reduction is a closed saturated set. (ii) The set $SN_{\beta\eta}$ of (untyped) lambda terms that are strongly normalizing under $\beta\eta$-reduction is a closed saturated set. (iii) The set of lambda terms M such that confluence holds under β-reduction from M and all of its subterms is a closed saturated set. (iv) The set of lambda terms M such that confluence holds under $\beta\eta$-reduction from M and all of its subterms is a closed saturated set.

Proof. (i)-(ii) Verifying closure is easy: if Mx is SN, then M must be SN, since otherwise an infinite reduction sequence from M would yield an infinite reduction from Mx. Verifying (S1) is also straightforward, since the existence of an infinite reduction sequence from $uN_1 \dots N_m$ implies that there is some infinite reduction sequence from some N_i, contradicting the assumption that each N_i is SN. Verifying (S2) is a little harder. We prove that if N is SN and there is an infinite reduction sequence from $(\lambda x.\, M)NN_1 \dots N_m$, then there is an infinite reduction sequence from $M[N/x]N_1 \dots N_m$.[13] The proof is slightly more complicated in the case of $\beta\eta$-reduction than it is in the case of β-reduction alone, because of possible *head η-reductions*.

Consider any infinite reduction sequence from $(\lambda x.\, M)NN_1 \dots N_m$. There are three different possible patterns:

(1) every term in this sequence is of the form $(\lambda x.\, M')N'N'_1 \dots N'_m$, where $M \xrightarrow{*}_\lambda M'$, $N \xrightarrow{*}_\lambda N'$, and $N_i \xrightarrow{*}_\lambda N'_i$ for $i = 1, \dots, m$, or

(2) there is a step $(\lambda x.\, M')N'N'_1 \dots N'_m \longrightarrow_\beta M'[N'/x]N'_1 \dots N'_m$ in this sequence, for some M', N', $N'_1, \dots, N'_m$, such that $M \xrightarrow{*}_\lambda M'$, $N \xrightarrow{*}_\lambda N'$, and $N_i \xrightarrow{*}_\lambda N'_i$ for $i = 1, \dots, m$, or

(3) $M \xrightarrow{*}_\lambda M'_1 x$, and there is a step

$$(\lambda x.\, (M'_1 x))N'N'_1 \dots N'_m \longrightarrow_\eta M'_1 N'N'_1 \dots N'_m,$$

for some N', $N'_1, \dots, N'_m$, such that $N \xrightarrow{*}_\lambda N'$, and $N_i \xrightarrow{*}_\lambda N'_i$, for $i = 1, \dots, m$.

In case (1), it is clear that the given infinite reduction sequence defines uniquely some independent finite or infinite reduction sequences originating from each of M, N, $N_1, \dots, N_m$. Since N is assumed to be SN, one of the sequences originating from M,

[13] This proof is inspired by Tait [37].

$N_1, \ldots, N_m$ must be infinite, and by lemma 5.7, we obtain an infinite reduction sequence from $M[N/x]N_1 \ldots N_m$.

In case (2), there are some terms M', N', $N'_1, \ldots, N'_m$, such that $M \xrightarrow{*}_\lambda M'$, $N \xrightarrow{*}_\lambda N'$, and $N_i \xrightarrow{*}_\lambda N'_i$ for $i = 1, \ldots, m$, and $M'[N'/x]N'_1 \ldots N'_m$ is the head of some infinite reduction sequence π. Using lemma 5.7, we can form the reduction sequence

$$(\lambda x.\, M)NN_1 \ldots N_m \longrightarrow_\beta M[N/x]N_1 \ldots N_m \xrightarrow{*}_\lambda M'[N'/x]N'_1 \ldots N'_m,$$

which can be extended to an infinite reduction sequence using π.

In case (3), since $M \xrightarrow{*}_\lambda M'_1 x$, using lemma 5.7, we have a reduction

$$(\lambda x.\, M)NN_1 \ldots N_m \longrightarrow_\beta M[N/x]N_1 \ldots N_m \xrightarrow{*}_\lambda M'_1[N/x]N'N'_1 \ldots N'_m.$$

However, because in (3) we have an η-reduction step, $x \notin FV(M'_1)$, and so $M'_1[N/x] = M'_1$. Thus, $M'_1[N/x]N'N'_1 \ldots N'_m = M'_1 N' N'_1 \ldots N'_m$. Since there is an infinite reduction from $M'_1 N' N'_1 \ldots N'_m$, we have an infinite reduction from $M[N/x]N_1 \ldots N_m$.

(iii)-(iv) The proof will be given in section 7 for the typed case. □

We can apply theorem 6.15 to the set $SN_{\beta\eta}$, which, by lemma 6.16, is closed and saturated, and we obtain the following corollary to theorem 6.15.

Lemma 6.17 For every polymorphic raw term M, if M type-checks then $Erase(M)$ is strongly normalizable under $\beta\eta$-reduction.

Using lemma 5.6, we have:

Lemma 6.18 Every polymorphic raw term M that type-checks is strongly normalizable under $\beta\eta$-reduction.

Applying theorem 6.15 to the set of terms M such that confluence holds (under β-reduction or $\beta\eta$-reduction) from M and all of its subterms, which, by lemma 6.16, is closed and saturated, we have:

Lemma 6.19 (Mitchell) The reduction relation $\longrightarrow_\lambda$ ($\beta\eta$-reduction) is confluent on terms of the form $Erase(M)$, where M type-checks. The result also holds for β-reduction.

Unfortunately, we have not been unable to show that lemma 6.19 implies that $\longrightarrow_{\lambda^\forall}$ is confluent on polymorphic terms that type-check. However, this result can be established using a typed version of the candidates of reducibility.

7 A Typed Version of The Candidates of Reducibility

We now describe a typed version of the "candidats de reductibilité", as originally defined by Girard (Girard 1970 [9], Girard 1972 [10]). We will present two sets of conditions for the typed candidates. The first set consists of conditions similar to those used by Stenlund [35], (basically the typed version of Tait's conditions, Tait 1973 [37]), and the second set consists of Girard's original conditions (Girard [10], [11]). We also compare these conditions, and prove that Girard's conditions are stronger than Tait's conditions.

Rather than using explicitly typed polymorphic terms as in Girard [10], we work with provable typing judgments.

Definition 7.1 For every type σ, let $\mathcal{PT}_\sigma$ be the set of all provable typing judgments of type σ (the provable typing judgments of the form $\Delta \triangleright M\colon\sigma$ for arbitrary Δ and M).

In order to reduce the amount of notation, if S is a set of provable typing judgments of type σ, rather than writing $\Delta \triangleright M\colon\sigma \in S$, we will write $\Delta \triangleright M \in S$.

Given any two types $\sigma, \tau \in \mathcal{T}$ and any two sets $S \subseteq \mathcal{PT}_\sigma$ and $T \subseteq \mathcal{PT}_\tau$, we let $[S \to T]$ be the subset of $\mathcal{PT}_{\sigma\to\tau}$ defined as follows:

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

We also refer to the operation $\to$ on sets of provable typing judgments defined above as the *function space constructor*.

Definition 7.2 Let $\mathcal{C} = (\mathcal{C}_\sigma)_{\sigma\in\mathcal{T}}$ be a $\mathcal{T}$-indexed family where each $\mathcal{C}_\sigma$ is a nonempty set of subsets of $\mathcal{PT}_\sigma$, and the following properties hold:

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

(2) For every $\sigma, \tau \in \mathcal{T}$, for every $C \in \mathcal{C}_\sigma$ and every $D \in \mathcal{C}_\tau$, we have $[C \to D] \in \mathcal{C}_{\sigma\to\tau}$.

(3) For every $\forall t.\, \sigma,\ \tau \in \mathcal{T}$, for every family $(A_{\tau,C})_{\tau\in\mathcal{T}, C\in\mathcal{C}_\tau}$, where each set $A_{\tau,C}$ is in $\mathcal{C}_{\sigma[\tau/t]}$, we have

$$\{\Delta \triangleright M \in \mathcal{PT}_{\forall t.\,\sigma} \mid \forall \tau \in \mathcal{T},\ \Delta \triangleright M\tau \in \bigcap_{C\in\mathcal{C}_\tau} A_{\tau,C}\} \in \mathcal{C}_{\forall t.\,\sigma}.$$

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

Definition 7.3 Let $\mathcal{C}$ be a $\mathcal{T}$-closed family. A pair $\langle\theta,\eta\rangle$ where $\theta\colon\mathcal{V}\to\mathcal{T}$ is a type substitution and $\eta\colon\mathcal{B}\cup\mathcal{V}\to\bigcup\mathcal{C}$ is a *candidate assignment* iff $\eta(t)\in\mathcal{C}_{\theta(t)}$ for every $t\in\mathcal{V}$ and $\eta(\sigma)\in\mathcal{C}_\sigma$ for every $\sigma\in\mathcal{B}$.

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

Definition 7.4 Given any candidate assignment $\langle\theta,\eta\rangle$, for every type σ, the set $[\![\sigma]\!]\theta\eta$ is a subset of $\mathcal{PT}_{\theta(\sigma)}$ defined as follows:

$$\begin{aligned}
[\![t]\!]\theta\eta &= \eta(t), \text{ whenever } t\in\mathcal{B}\cup\mathcal{V},\\
[\![(\sigma\to\tau)]\!]\theta\eta &= [[\![\sigma]\!]\theta\eta\to[\![\tau]\!]\theta\eta],\\
[\![\forall t.\,\sigma]\!]\theta\eta &= \{\Delta\rhd M\in\mathcal{PT}_{\theta(\forall t.\,\sigma)}\mid\forall\tau\in\mathcal{T},\\
&\qquad\Delta\rhd M\tau\in\bigcap_{C\in\mathcal{C}_\tau}[\![\sigma]\!]\theta[t:=\tau]\eta[t:=C]\}.
\end{aligned}$$

Strictly speaking, $[\![\sigma]\!]\theta\eta$ is defined for the $\equiv_\alpha$-class $[\sigma]$ of σ. Thus, it can be assumed that σ is safe for θ. For a type $\forall t.\,\sigma$, this implies that $t\notin\mathcal{FV}(\theta(v))$ for every $v\in dom(\theta)$, and consequently that $\theta[t:=\tau](\sigma)=\theta(\sigma)[\tau/t]$. This shows that the sets of terms involved in the intersection are indeed of the right type $\theta(\sigma)[\tau/t]$.

The following technical lemma will be useful later.

Lemma 7.5 Given any two candidate assignments $\langle\theta_1,\eta_1\rangle$ and $\langle\theta_2,\eta_2\rangle$, for every type σ, if θ_1, θ_2 agree on $\mathcal{FV}(\sigma)$, and η_1, η_2 agree on $\mathcal{FV}(\sigma)$ and $\mathcal{B}$, then $[\![\sigma]\!]\theta_1\eta_1=[\![\sigma]\!]\theta_2\eta_2$.

Proof. Easy induction on the structure of types. □

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

Lemma 7.6 (Girard) If $\mathcal{C}$ is a $\mathcal{T}$-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 proof is similar to the proof of lemma 6.5. The only case worth mentioning is the case of a universal type. Given a type $\forall t.\,\sigma$ and a candidate assignment $\langle\theta,\eta\rangle$, using α-renaming, it can be assumed that $\forall t.\,\sigma$ is safe for θ. By the induction hypothesis, $[\![\sigma]\!]\theta\eta\in\mathcal{C}_{\theta(\sigma)}$. Thus, for every $\tau\in\mathcal{T}$ and for every $C\in\mathcal{C}_{\theta[t:=\tau](\sigma)}$, we also have $[\![\sigma]\!]\theta[t:=\tau]\eta[t:=C]\in\mathcal{C}_{\theta[t:=\tau](\sigma)}$. However, $\theta[t:=\tau](\sigma)=\theta(\sigma)[\tau/t]$ as we observed earlier since $\forall t.\,\sigma$ is safe for θ. Thus, $[\![\sigma]\!]\theta[t:=\tau]\eta[t:=C]\in\mathcal{C}_{\theta(\sigma)[\tau/t]}$, and we conclude by condition (3) of $\mathcal{T}$-closed families. □

The following technical lemma will be needed later.

Lemma 7.7 Given any two types σ, τ, 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. □

In order to use lemma 7.6 in proving properties of polymorphic lambda calculi, we need to define $\mathcal{T}$-closed families satisfying some additional properties.

Definition 7.8 We say that a $\mathcal{T}$-indexed family $\mathcal{C}$ is a *family of sets of candidates of reducibility* iff it is $\mathcal{T}$-closed and satisfies the conditions listed below:[14]

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

R1. For every $\sigma \in \mathcal{T}$, for every set $C \in \mathcal{C}_\sigma$, $\Delta \rhd x \in C$, for every $x{:}\,\sigma \in \Delta$,
For every $\sigma = Type(f)$, for every set $C \in \mathcal{C}_\sigma$, $\Delta \rhd f \in C$, for every $f \in \Sigma$.

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

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

(ii) For all $\sigma, \tau \in \mathcal{T}$, for every $C \in \mathcal{C}_{\sigma[\tau/t]}$, if

$$\begin{aligned}
&\Delta \rhd M \in \bigcup \mathcal{C}_\sigma \text{ and}\\
&\Delta \rhd M[\tau/t] \in C, \text{ then}\\
&\Delta \rhd (\Lambda t.\, M)\tau \in C.
\end{aligned}$$

As in the untyped case, (R0), (R1), and (R2), are all we need to prove the following fundamental result.

Lemma 7.9 (Girard) Let $\mathcal{C} = (\mathcal{C}_\sigma)_{\sigma \in \mathcal{T}}$ be a $\mathcal{T}$-indexed family of sets of candidates of reducibility. For every $\Gamma \rhd M \in \mathcal{PT}_\sigma$, for every candidate assignment $\langle \theta, \eta \rangle$, for every substitution $\varphi{:}\,\Gamma \to \Delta$, if $\theta(\Delta) \rhd \varphi(x) \in [\![\Gamma(x)]\!]\theta\eta$ for $x \in FV(M)$, then $\theta(\Delta) \rhd \varphi(\theta(M)) \in [\![\sigma]\!]\theta\eta$.

Proof. It is similar to the proof of lemma 6.8 and proceeds by induction on the depth of the proof tree for $\Gamma \rhd M{:}\,\sigma$. The only cases worth considering are type abstraction and type

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

application, the other two being essentially unchanged, except that (R0) is needed in the case of λ-abstraction.

Case 3.

$$\frac{\Gamma \triangleright M\colon \forall t.\, \sigma}{\Gamma \triangleright M\tau\colon \sigma[\tau/t]} \qquad (\textit{type application})$$

First, by suitable α-renaming, it can be assumed that $M\tau$ is safe for φ and θ, and that $\forall t.\, \sigma$ is safe for θ. By the induction hypothesis, $\theta(\Delta) \triangleright \varphi(\theta(M)) \in [\![\forall t.\, \sigma]\!]\theta\eta$. By the definition of $[\![\forall t.\, \sigma]\!]\theta\eta$, we have

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

for every $\delta \in \mathcal{T}$ and $C \in \mathcal{C}_\delta$. Since $\forall t.\, \sigma$ is safe for θ, as observed before, we have $\theta(\sigma)[\delta/t] = \theta[t := \delta](\sigma)$. Since by lemma 7.6, $[\![\tau]\!]\theta\eta \in \mathcal{C}_{\theta(\tau)}$, by setting $\delta = \theta(\tau)$ and $C = [\![\tau]\!]\theta\eta$, we have

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

Since $\forall t.\sigma$ is safe for θ and φ and θ have disjoint domains, we have $\varphi(\theta(M))\theta(\tau) = \varphi(\theta(M\tau))$ and $\theta(\sigma)[\theta(\tau)/t] = \theta(\sigma[\tau/t])$, and so

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

However, by lemma 7.7, $[\![\sigma[\tau/t]]\!]\theta\eta = [\![\sigma]\!]\theta[t := \theta(\tau)]\eta[t := [\![\tau]\!]\theta\eta]$, and so

$$\theta(\Delta) \triangleright \varphi(\theta(M\tau)) \in [\![\sigma[\tau/t]]\!]\theta\eta.$$

Case 4.

$$\frac{\Gamma \triangleright M\colon \sigma}{\Gamma \triangleright \Lambda t.\, M\colon \forall t.\, \sigma} \qquad (\textit{type abstraction})$$

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

Let $k+1$ be the depth of the proof tree for $\Gamma \triangleright \Lambda t.\, M\colon \forall t.\, \sigma$. Since $t \notin \mathcal{FV}(\Gamma(x))$ for every $x \in dom(\Gamma) \cap FV(M)$, by lemma 7.5, we have $[\![\Gamma(x)]\!]\theta\eta = [\![\Gamma(x)]\!]\theta[t := \tau]\eta[t := C]$, for every $\tau \in \mathcal{T}$ and every $C \in \mathcal{C}_\tau$. By the induction hypothesis,

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

for every $\tau \in \mathcal{T}$ and every $C \in \mathcal{C}_\tau$. By suitable α-renaming, it can be assumed that $\Lambda t.\, M$ is safe for φ and θ, that $\forall t.\, \sigma$ is safe for θ, and that $t \notin \mathcal{FV}(\Delta(x))$ for every $x \in dom(\Delta)$. Then, $\theta[t := \tau](\Delta) = \theta(\Delta)$, and as observed before, $\theta[t := \tau](M) = \theta(M)[\tau/t]$, $\theta[t := \tau](\sigma) = \theta(\sigma)[\tau/t]$, and since φ and $\theta[t := \tau]$ have disjoint domains, we have

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

for every $\tau \in \mathcal{T}$ and every $C \in \mathcal{C}_\tau$. In particular, since $t \in \mathcal{T}$, by choosing $\tau = t$, we have $\theta(\Delta) \triangleright \varphi(\theta(M)) \in [\![\sigma]\!]\theta\eta[t := C]$. Since by lemma 7.6, $[\![\sigma]\!]\theta\eta[t := C] \in \mathcal{C}_{\theta(\sigma)}$, we have $\varphi(\theta(M)) \in \bigcup \mathcal{C}_{\theta(\sigma)}$. Thus, by (R2)(ii), we have

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

that is

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

for every $\tau \in \mathcal{T}$ and every $C \in \mathcal{C}_\tau$. By definition 7.4, this means that

$$\theta(\Delta) \triangleright \varphi(\theta(\Lambda t.\, M)) \in [\![\forall t.\, \sigma]\!]\theta\eta.$$

□

Remark: As in the remark just after lemma 6.8, it should be observed that lemma 7.9 still holds if condition (R2) of definition 7.8 is relaxed so that in (R2)(i), $\bigcup \mathcal{C}_\tau$ is replaced by $\mathcal{PT}_\tau$, $\bigcup \mathcal{C}_\sigma$ by $\mathcal{PT}_\sigma$, and in (R2)(ii), $\bigcup \mathcal{C}_\sigma$ is replaced by $\mathcal{PT}_\sigma$. The relaxed conditions will be called (R2′)(i) and and (R2′)(ii).

In order to prove that families of sets of candidates of reducibility exist, one needs conditions stronger that (R1) and (R2). First, we give conditions adapted from Tait and Mitchell.

8 Families of Sets of Saturated Sets

The definition of the saturated sets given in the untyped case (definition 6.9) is adapted as follows.

Definition 8.1 Let $S = (S_\sigma)_{\sigma \in \mathcal{T}}$ be a $\mathcal{T}$-indexed family such that each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$. We say that S is *closed* iff for all $\sigma, \tau \in \mathcal{T}$, for every $x \in \mathcal{X}$, if $\Delta \triangleright M \in \mathcal{PT}_{\sigma\to\tau}$ and $\Delta, x{:}\,\sigma \triangleright Mx \in S_\tau$, then $\Delta \triangleright M \in S_{\sigma\to\tau}$, and for every $t \in \mathcal{V}$, if $\Delta \triangleright M \in \mathcal{PT}_{\forall t.\,\sigma}$ and $\Delta \triangleright Mt \in S_\sigma$, then $\Delta \triangleright M \in S_{\forall t.\,\sigma}$. The family $Sat(S) = (Sat(S)_\sigma)_{\sigma\in\mathcal{T}}$ of sets of *saturated subsets of* S is defined such that for every $\sigma \in \mathcal{T}$, $Sat(S)_\sigma$ consists of those subsets $C \subseteq S_\sigma$ such that the following conditions hold:[15]

S0. Whenever $\Delta \triangleright M \in C$ and $\Delta \subseteq \Delta'$, then $\Delta' \triangleright M \in C$.

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

S1. For every type $\sigma \in \mathcal{T}$, for every $C \in Sat(S)_\sigma$, for all $n \geq 0$, for all $N_1, \ldots, N_n \in \mathcal{T} \cup \mathcal{P}\Lambda$, for every $u \in \mathcal{X} \cup \Sigma$, if

$$\begin{aligned}
&\Delta \triangleright uN_1 \ldots N_n \in \mathcal{P}\mathcal{T}_\sigma, \text{ and} \\
&\Delta \triangleright N_i \in S_{\xi_i} \text{ for some } \xi_i \text{ whenever } N_i \text{ is a term } (1 \leq i \leq n), \text{ then} \\
&\Delta \triangleright uN_1 \ldots N_n \in C.
\end{aligned}$$

S2. (i) For all $\sigma, \tau \in \mathcal{T}$, for every $C \in Sat(S)_\tau$, for all $n \geq 0$, for all $N_1, \ldots, N_n \in \mathcal{T} \cup \mathcal{P}\Lambda$, for all Δ, Δ', if

$$\begin{aligned}
&\Delta \triangleright M \in S_\xi \text{ for some } \xi, \\
&\Delta' \triangleright M[N/x]N_1 \ldots N_n \in C, \\
&\Delta' \triangleright N \in S_\sigma, \text{ and} \\
&\Delta' \triangleright N_i \in S_{\xi_i} \text{ for some } \xi_i \text{ whenever } N_i \text{ is a term } (1 \leq i \leq n), \text{ then} \\
&\Delta' \triangleright (\lambda x{:}\sigma.\, M)NN_1 \ldots N_n \in C.
\end{aligned}$$

(ii) For all $\sigma, \tau \in \mathcal{T}$, for every $C \in Sat(S)_\sigma$, for all $n \geq 0$, for all $N_1, \ldots, N_n \in \mathcal{T} \cup \mathcal{P}\Lambda$, if

$$\begin{aligned}
&\Delta \triangleright M[\tau/t]N_1 \ldots N_n \in C, \\
&\Delta \triangleright M \in S_\xi \text{ for some } \xi, \text{ and} \\
&\Delta \triangleright N_i \in S_{\xi_i} \text{ for some } \xi_i \text{ whenever } N_i \text{ is a term } (1 \leq i \leq n), \quad \text{then} \\
&\Delta \triangleright (\Lambda t.\, M)\tau N_1 \ldots N_n \in C.
\end{aligned}$$

We have the following typed version of lemma 6.10.

Lemma 8.2 (Girard, Tait, Mitchell) Let $S = (S_\sigma)_{\sigma \in \mathcal{T}}$ be a closed family where each S_σ is a nonempty subset of $\mathcal{P}\mathcal{T}_\sigma$, let $\mathcal{C}$ be the $\mathcal{T}$-indexed family of sets of saturated subsets of S, and assume that $S_\sigma \in \mathcal{C}_\sigma$ for every $\sigma \in \mathcal{T}$ (i.e. S_σ is a saturated subset of itself). Then $\mathcal{C}$ is a family of sets of candidates of reducibility.

Proof. It is similar to the proof of lemma 6.10. One point worth mentioning is the necessity of allowing $N_1 \ldots N_n$ to be terms *or types*, in order to prove closure condition (3) of definition 7.2. □

We now consider the conditions used by Girard in [10] and [11], and their relationship to Tait and Mitchell's conditions.

9 Families of Sets of Girard Sets

Girard's conditions basically assert that complete induction holds for certain simple terms w.r.t. $\longrightarrow_{\lambda^\forall}$.

Definition 9.1 A *simple term* is a term that is not an abstraction. Thus, a term M is simple iff it is either a variable x, a constant $f \in \Sigma$, an application MN, or a type application $M\tau$.

The idea behind this definition is that for a simple term M, for every term N, if $MN \longrightarrow_{\lambda^\forall} Q$, then either $M \longrightarrow_{\lambda^\forall} M'$ and $Q = M'N$, or $N \longrightarrow_{\lambda^\forall} N'$ and $Q = MN'$.

Definition 9.2 Let $S = (S_\sigma)_{\sigma\in\mathcal{T}}$ be a $\mathcal{T}$-indexed family such that each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$. A subset C of S_σ is a *Girard set* of type σ iff the following conditions hold:[16]

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 $\longrightarrow_{\lambda^\forall}$;

CR2. If $\Delta \triangleright M \in C$ and $M \longrightarrow_{\lambda^\forall} N$, 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 $M \longrightarrow_{\lambda^\forall} N$, then $\Delta \triangleright M \in C$.

Note that (CR3) implies that all simple irreducible terms are in C. We shall prove a lemma analogous to lemma 8.2 for families of sets of Girard subsets, but first, we establish a precise connection between Girard sets and saturated sets. We prove that conditions (CR1), (CR2) and (CR3) imply conditions (S1) and (S2).

Lemma 9.3 Let $S = (S_\sigma)_{\sigma\in\mathcal{T}}$ be a family where each S_σ is a Girard subset of $\mathcal{PT}_\sigma$. Every Girard subset of S is a saturated set, i.e., satisfies conditions (S1), (S2).

Proof. We first make the following observation. For every term M, it is clear that there are only finitely many terms N such that $M \longrightarrow_{\lambda^\forall} N$. Thus, for every SN term M, by König's lemma, the tree of reduction sequences from M is finite. Thus, for every SN term M, the depth of the reduction tree from M is well defined, and we denote it as $\delta(M)$.

We now prove (S1). For every $u \in \mathcal{X} \cup \Sigma$, assume that

$$\Delta \triangleright uN_1 \dots N_n \in \mathcal{PT}_\sigma, \text{ and}$$
$$\Delta \triangleright N_i \in S_{\xi_i} \text{ for some } \xi_i \text{ whenever } N_i \text{ is a term } (1 \leq i \leq n).$$

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

We prove by complete induction on $\delta = \delta(N_1) + \ldots + \delta(N_n)$ that $\Delta \rhd uN_1 \ldots N_n \in C$. Since $uN_1 \ldots N_n$ is a simple term, we can do this by using (CR3).

The base case $\delta = 0$ holds, since then $uN_1 \ldots N_n$ is simple and irreducible. For the induction step, note that $uN_1 \ldots N_n \longrightarrow_{\lambda^\forall} Q$ implies that $Q = uN_1 \ldots N_i' \ldots N_n$, where $N_i \longrightarrow_{\lambda^\forall} N_i'$. Also, if $N_i \longrightarrow_{\lambda^\forall} N_i'$, by (CR2) we have $\Delta \rhd N_i' \in S_{\xi_i}$, and since $\delta(N_i') < \delta(N_i)$, the induction hypothesis implies that $\Delta \rhd uN_1 \ldots N_i' \ldots N_n \in C$. Using (CR3), we conclude that $\Delta \rhd uN_1 \ldots N_n \in C$.

We also prove (S2). Assume that for some Δ, Δ',

$$\begin{aligned}
&\Delta \rhd M \in S_\xi \text{ for some } \xi,\\
&\Delta' \rhd M[N/x]N_1 \ldots N_n \in C,\\
&\Delta' \rhd N \in S_\sigma, \text{ and}\\
&\Delta' \rhd N_i \in S_{\xi_i} \text{ for some } \xi_i \text{ whenever } N_i \text{ is a term } (1 \leq i \leq n).
\end{aligned}$$

We want to prove that $\Delta' \rhd (\lambda x{:}\sigma.\ M)NN_1 \ldots N_n \in C$. Since $(\lambda x{:}\sigma.\ M)NN_1 \ldots N_n$ is a simple term, we can do this by using (CR3). Since the terms $M, N, N_1, \ldots, N_n$ are SN, we prove by complete induction on $\delta = \delta(M) + \delta(N) + \delta(N_1) + \ldots + \delta(N_n)$ that if $\Delta' \rhd M[N/x]N_1 \ldots N_n \in C$ then $\Delta' \rhd (\lambda x{:}\sigma.\ M)NN_1 \ldots N_n \in C$.

The base case $\delta = 0$ holds by (CR3), since then the only possible reduction is $(\lambda x{:}\sigma.\ M)NN_1 \ldots N_n \longrightarrow_{\lambda^\forall} Q$ where $Q = M[N/x]N_1 \ldots N_n$, but $\Delta' \rhd Q \in C$ by hypothesis. Otherwise, we just prove that $\Delta' \rhd Q \in C$ whenever $(\lambda x{:}\sigma.\ M)NN_1 \ldots N_n \longrightarrow_{\lambda^\forall} Q$. If $Q = M[N/x]N_1 \ldots N_n$, then we know that $\Delta' \rhd M[N/x]N_1 \ldots N_n \in C$, by the hypothesis. Otherwise, either $Q = (\lambda x{:}\sigma.\ M')NN_1 \ldots N_n$ where $M \longrightarrow_{\lambda^\forall} M'$, or $Q = (\lambda x{:}\sigma.\ M)N'N_1 \ldots N_n$ where $N \longrightarrow_{\lambda^\forall} N'$, or $Q = (\lambda x{:}\sigma.\ M)NN_1 \ldots N_i' \ldots N_n$ where $N_i \longrightarrow_{\lambda^\forall} N_i'$, or $Q = M'NN_1 \ldots N_n$ where $M = M'x$ and where $x \notin FV(M')$.

In the last case, note that $Q = M'NN_1 \ldots N_n = (M'x)[N/x]N_1 \ldots N_n$ since $x \notin FV(M')$, and since $M = M'x$, we have $\Delta' \rhd M[N/x]N_1 \ldots N_n \in C$, by the hypothesis.

In the other cases, by (CR2), we have $\Delta \rhd M' \in S_\xi$, $\Delta' \rhd N' \in S_\sigma$, and $\Delta' \rhd N_i' \in S_{\xi_i}$. Using (CR2) (and a simple induction on the number of reduction steps when $N \longrightarrow_{\lambda^\forall} N'$), since $\Delta' \rhd M[N/x]N_1 \ldots N_n \in C$, we also have $\Delta' \rhd M'[N/x]N_1 \ldots N_n \in C$ when $M \longrightarrow_{\lambda^\forall} M'$, $\Delta' \rhd M[N'/x]N_1 \ldots N_n \in C$ when $N \longrightarrow_{\lambda^\forall} N'$, and $\Delta' \rhd M[N/x]N_1 \ldots N_i' \ldots N_n \in C$ when $N_i \longrightarrow_{\lambda^\forall} N_i'$ (Note that this step of the proof seems to have been overlooked in other published proofs, as pointed out to us by Pierre Louis Curien and Roberto Di Cosmo).

Since $\delta(M') < \delta(M)$, $\delta(N') < \delta(N)$, and $\delta(N_i') < \delta(N_i)$, the induction hypothesis implies that $\Delta' \rhd (\lambda x{:}\sigma.\ M')NN_1 \ldots N_n \in C$, $\Delta' \rhd (\lambda x{:}\sigma.\ M)N'N_1 \ldots N_n \in C$, and $\Delta' \rhd$

$(\lambda x{:}\sigma.\, M)NN_1 \ldots N_i' \ldots N_n \in C$. Using (CR3), it follows that $\Delta' \rhd (\lambda x{:}\sigma.\, M)NN_1 \ldots N_n \in C$.

The case where

$$\Delta \rhd M[\tau/t]N_1 \ldots N_n \in C,$$
$$\Delta \rhd M \in S_\xi \text{ for some } \xi, \text{ and}$$
$$\Delta \rhd N_i \in S_{\xi_i} \text{ for some } \xi_i \text{ whenever } N_i \text{ is a term } (1 \le i \le n)$$

is handled similarly. We show that if $\Delta \rhd M[\tau/t]N_1 \ldots N_n \in C$, then $\Delta \rhd Q \in C$ whenever $\Delta \rhd (\Lambda t.\, M)\tau N_1 \ldots N_n \longrightarrow_{\lambda^\forall} Q$. □

We now prove the analogous to lemma 8.2 for Girard sets.

Lemma 9.4 Let $S = (S_\sigma)_{\sigma\in\mathcal{T}}$ be a closed family where each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$, let $\mathcal{C}$ be the $\mathcal{T}$-indexed family of sets of Girard subsets of S, and assume that $S_\sigma \in \mathcal{C}_\sigma$ for every $\sigma \in \mathcal{T}$ (i.e. S_σ is a Girard subset of itself). Then $\mathcal{C}$ is a family of candidates of reducibility.

Proof. We need to check that $\mathcal{C}$ satisfies the conditions of definition 7.8. By lemma 9.3, the sets in each $\mathcal{C}_\sigma$ satisfy conditions (S1) and (S2), which obviously imply (R1) and (R2). It remains to show that $\mathcal{C}$ is $\mathcal{T}$-closed. We need to prove properties (1), (2), (3), of definition 7.2. This is done by induction on types.

Property (1) is an immediate consequence of (S1). We will verify condition (2), leaving (3) as an exercise.

Let C and D be any two Girard sets. We need to show that $[C \to D]$ is a Girard set. Assume that the type of the terms in $[C \to D]$ is $\sigma \to \tau$.

For every $\Delta \rhd M \in [C \to D]$, since $\Delta, x{:}\sigma \rhd x \in C$ by (S1), by the definition of $[C \to D]$, we have $\Delta, x{:}\sigma \rhd Mx \in D$, and since S is closed, we have $\Delta \rhd M \in S_{\sigma\to\tau}$. Thus, $[C \to D]$ is a subset of $S_{\sigma\to\tau}$. Since x is obviously SN, $\Delta, x{:}\sigma \rhd Mx \in D$, and (CR1) holds for D because it is a Girard set, it follows that M is SN. Thus, $[C \to D]$ satisfies (CR1).

Assume that $M \longrightarrow_{\lambda^\forall} M'$, where $\Delta \rhd M \in [C \to D]$. For every Δ' such that $\Delta \subseteq \Delta'$ and $\Delta' \rhd N \in C$, we have $\Delta' \rhd MN \in D$ and $MN \longrightarrow_{\lambda^\forall} M'N$. By (CR2) applied to D, we have $\Delta' \rhd M'N \in D$. Thus, by the definition of $[C \to D]$, we have $\Delta \rhd M' \in [C \to D]$.

It remains to verify (CR3). Let $\Delta \rhd M$ be any simple term of type σ, and assume that $\Delta \rhd Q \in [C \to D]$ whenever $M \longrightarrow_{\lambda^\forall} Q$. We want to prove that $\Delta \rhd M \in [C \to D]$. By the definition of $[C \to D]$, this will be the case if we can show that for every Δ' such that $\Delta \subseteq \Delta'$ and $\Delta' \rhd N \in C$, then $\Delta' \rhd MN \in D$.

We prove by complete induction on $\delta(N)$ that $\Delta' \triangleright MN \in D$. Since MN is simple, we can do this by using (CR3). Because M is simple, observe that $MN \longrightarrow_{\lambda^\forall} U$ implies that either

(a) $U = M'N$ and $M \longrightarrow_{\lambda^\forall} M'$, or

(b) $U = MN'$ and $N \longrightarrow_{\lambda^\forall} N'$.

In case (a), since we have assumed that $\Delta \triangleright Q \in [C \to D]$ whenever $M \longrightarrow_{\lambda^\forall} Q$, we have $\Delta \triangleright M' \in [C \to D]$, and thus $\Delta' \triangleright M'N \in D$ since $\Delta' \triangleright N \in C$.

In case (b), since $\Delta' \triangleright N \in C$ and $N \longrightarrow_{\lambda^\forall} N'$, by (CR2) applied to C we have $\Delta' \triangleright N' \in C$, we also have $\delta(N') < \delta(N)$, and by the induction hypothesis, this yields $\Delta' \triangleright MN' \in D$. We can conclude that $\Delta' \triangleright MN \in D$ by application of (CR3) to D. Hence, we have shown that $[C \to D]$ also satisfies (CR3), and consequently it is a Girard set. □

Having shown that both closed families of saturated sets and closed families of Girard sets yield families of sets of candidates of reducibility, we can prove the typed version of Girard's fundamental theorem.

10 Girard's Fundamental Theorem

The fundamental theorem holds for both saturated and Girard sets.

Theorem 10.1 (Girard) Let $S = (S_\sigma)_{\sigma \in \mathcal{T}}$ be a closed family where each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$. Let $\mathcal{C}$ be either the $\mathcal{T}$-indexed family of sets of saturated subsets of S, or the family of Girard subsets of S, and assume that $S_\sigma \in \mathcal{C}_\sigma$ for every $\sigma \in \mathcal{T}$. For every $\Delta \triangleright M \in \mathcal{PT}_\sigma$, we have $\Delta \triangleright M \in S_\sigma$.

Proof. By lemma 8.2 or lemma 9.4, $\mathcal{C}$ is a family of sets of candidates of reducibility. We now apply lemma 7.9 to any assignment (for example, the assignment with value $\eta(t) = S_t$), the identity type substitution, and the identity term substitution, which is legitimate since by (S1), every variable belongs to every saturated set.[17]

Remark: As in the untyped case, if we are interested in the version of the candidates using conditions (R2′)(i) and (R2′)(ii), then lemma 8.2 holds if conditions (S2)(i) and (S2)(ii) of definition 8.1 are relaxed in the obvious way (for example, S_ξ is replaced by $\mathcal{PT}_\xi$).

The next lemma is a typed version of lemma 6.16 and gives interesting examples of closed families of Girard sets and saturated sets.

[17] 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.

Lemma 10.2 (Girard, Tait, Mitchell) (i) The $\mathcal{T}$-indexed family SN_β such that for every $\sigma \in \mathcal{T}$, $SN_{\beta,\sigma}$ is the set of provable typing judgments $\Delta \triangleright M\colon\sigma$ such that M is strongly normalizing under β-reduction, is a closed family of Girard and saturated sets. (ii) The $\mathcal{T}$-indexed family $SN_{\beta\eta}$ such that for every $\sigma \in \mathcal{T}$, $SN_{\beta\eta,\sigma}$ is the set of provable typing judgments $\Delta \triangleright M\colon\sigma$ such that M is strongly normalizing under $\beta\eta$-reduction, is a closed family of Girard and saturated sets. (iii) The $\mathcal{T}$-indexed family consisting for every $\sigma \in \mathcal{T}$ of the set of provable typing judgments $\Delta \triangleright M\colon\sigma$ such that confluence under β-reduction holds from M and all of its subterms, is a closed family of Girard and saturated sets. (iv) The $\mathcal{T}$-indexed family consisting for every $\sigma \in \mathcal{T}$ of the set of provable typing judgments $\Delta \triangleright M\colon\sigma$ such that confluence under $\beta\eta$-reduction holds from M and all of its subterms, is a closed family of Girard and saturated sets.

Proof. (i)-(ii) The verification that SN_β and $SN_{\beta\eta}$ are closed families of Girard sets is obvious. The verification that they are closed families of saturated sets is essentially identical to the proof of lemma 6.16. Verifying (S2)(ii) in the case of type abstraction is similar to the other case (S2)(i) but a bit simpler, since types cannot be $\beta\eta$-reduced.

(iii)-(iv) The verification that these sets are Girard sets is similar to the verification that they are saturated sets and is omitted. The verification that they are saturated sets is done in appendix 2. □

One should note that because the conditions for being a Girard set are stronger than the conditions for being a saturated set, it is trivial to show that $SN_{\beta\eta}$ is a closed family of Girard sets, whereas, showing that it is a closed family of saturated sets requires more work (namely, part of lemma 6.16). Applying theorem 10.1 to the set $SN_{\beta\eta}$, which, by lemma 9.4 is a closed family of Girard sets (or by lemma 8.2, a closed family of saturated sets), we obtain the following corollary to theorem 10.1.

Lemma 10.3 Every term M that type-checks is strongly normalizing under $\beta\eta$-reduction.

Interestingly, using parts (iii)-(iv) of lemma 10.2, we obtain a new proof of the fact that $\longrightarrow_{\lambda^\forall}$ is confluent on terms that type-check. Girard (Girard [10]) proved this result (for β-reduction) using an adaptation of Tait and Matin Löf's proof of confluence for the untyped lambda calculus.

Lemma 10.4 Confluence holds under $\beta\eta$-reduction for terms M that type-check. Confluence also holds under β-reduction for terms that type-check.

It is interesting to note that Lemma 10.4 **fails** for raw terms. The following example shows what goes wrong.

Consider the term $M = \lambda x{:}\,\sigma.(\lambda y{:}\,\tau.y)x$ where $\sigma \neq \tau$. Clearly, M does not type-check, and we have two reductions

$$\lambda x{:}\,\sigma.\ (\lambda y{:}\,\tau.\ y)x \longrightarrow_\beta \lambda x{:}\,\sigma.\ x \quad \text{and} \quad \lambda x{:}\,\sigma.\ (\lambda y{:}\,\tau.\ y)x \longrightarrow_\eta \lambda y{:}\,\tau.\ y,$$

and there is no way to achieve confluence since $\sigma \neq \tau$. Thus, $\longrightarrow_{\lambda^\forall}$ is not confluent on all raw terms, only those that type-check.

However, the polymorphic lambda calculus $\lambda^\forall$ is Church-Rosser and strongly normalizing on terms that type-check. What is interesting about this new proof of the Church-Rosser property is that it makes an essential use of the type structure. This observation was made by Statman in the context of logical relations [34] (for the simply typed lambda calculus).

11 A Comparison of Proofs

The purpose of this section is to compare various proofs that have appeared in the literature. These proofs are considered in chronological order.

1. Girard's proof(s) (1970, 1972).

The method of candidates of reducibility was invented by Girard in order to settle entirely proof theoretically a famous open problem in proof theory known as Takeuti's conjecture. Girard's "tour de force", settling positively Takeuti's conjecture for higher-order intuitionistic logic by purely proof theoretic means, is first accomplished in Girard [9]. Takeuti's conjecture is the generalization of Gentzen's cut elimination theorem to (classical) higher-order logic (for details on Takeuti's conjecture, the reader should consult Girard [12]). Girard's proof of Takeuti's conjecture (in [9]) consists in exploiting the "formulae as types" analogy, first observed by Curry and Howard. Roughly speaking, a *proof* (in a Prawitz-style deduction system) is coded as a certain kind of *lambda term*, and the *formula* occurring as the conclusion of the proof is considered to be the *type* of the term. What is remarkable about this correspondence proof – lambda term, formula – type, is that the process of normalizing a proof (eliminating certain redundancies having to do with an introduction rule followed by an elimination rule for the same logical connective) corresponds to β-reduction applied to the term representing the proof. Thus, if one succeeds in defining a typed lambda calculus in which terms represent proofs in a natural deduction system, and β-conversion corresponds to proof normalization, if in addition one is able to prove strong normalization for this typed lambda calculus, then one has shown strong normalization for proofs in the natural deduction system.

Girard's achievement was to define a typed lambda calculus, system F, which corresponds to second-order propositional intuitionistic logic, and to prove strong normalization

for system F. In order to prove strong normalization, Girard invented the method of candidates of reducibility. Girard also used the method of candidates of reducibility to prove Takeuti's conjecture for higher-order intuitionistic logic.

System F is actually more general than the calculus that we have presented, since it includes product types and existentially quantified types. In [9], candidates of reducibility are sets of typed terms satisfying certain conditions technically rather different from conditions (R1) and (R2) given in definition 7.8. We will not list these conditions here, but instead present the conditions given in Girard's thesis [10] and his class notes [11] later in this section.

Reading [9] is quite challenging, because a lot of extremely original material is presented in a short space, and also because the notations used are not the most illuminating. Nevertheless, the method of the candidates emerges very clearly and with great power.

In his thesis [10], Girard extends system F to a typed lambda calculus named F_ω. The system F_ω encodes proofs in higher-order intuitionistic logic, whereas system F only encodes the second-order fragment of this logic. The system F_ω also includes product types, existential types, and even disjunctive types. The method of candidates of reducibility is extended to F_ω, and strong normalization is shown, as well as the Church-Rosser theorem (by the method of Tait and Martin Löf). Much more is done in the thesis, but we are primarily focusing on the method of candidates of reducibility. A simpler (and more readable) version of this proof for system F is given in Girard [11].

Both in [10] and [11], Girard uses a typed version of the candidates satisfying some interesting conditions. Girard defines a *simple term* as a term that is not an abstraction. Thus, a term M is simple iff it is either a variable x, an application MN, or a type application $M\tau$. A candidate of reducibility of type σ is a set C of terms of type σ such that:

CR1. If $M \in C$, then M is SN;

CR2. If $M \in C$ and $M \longrightarrow_{\lambda^\forall} N$, then $N \in C$;

CR3. If M is a simple term and if $N \in C$ for every N such that $M \longrightarrow_{\lambda^\forall} N$, then $M \in C$.

Note that (CR3) implies that all variables of type σ are in C (what we call (R1) in definition 7.8). Girard defines what he calls *reducibility with parameters*, as we do in definition 7.4, except that he uses a notation that we find a little confusing. Given a type σ, if $\mathcal{FV}(\sigma) = \{t_1, \ldots, t_n\}$ is the set of free type variables in σ, instead of our type substitution $\theta\colon \mathcal{V} \to \mathcal{T}$ he uses a sequence $U = \langle U_1, \ldots, U_n\rangle$ of types, and instead of our assignment $\eta\colon \mathcal{V} \to \bigcup \mathcal{C}$, he uses a sequence $C = \langle C_1, \ldots, C_n\rangle$ of candidates, each C_i being of type

U_i. Then, what we denote as $[\![\sigma]\!]\theta\eta$ is denoted by Girard as $RED(\sigma[C_1/t_1, \ldots, C_n/t_n])$. $RED(\sigma[C_1/t_1, \ldots, C_n/t_n])$ is a certain set of terms of type $\theta(\sigma)$, in Girard's notation of type $\sigma[U_1/t_1, \ldots, U_n/t_n]$.

One of the problems that we have with this notation is that it is difficult to distinguish between actual substitution, as in $\sigma[U_1/t_1, \ldots, U_n/t_n]$, and assigning candidates to the type variables, as in $RED(\sigma[C_1/t_1, \ldots, C_n/t_n])$. The notation $RED(\sigma[C_1/t_1, \ldots, C_n/t_n])$ is also slightly ambiguous since it does not refer to the type substitution $[U_1/t_1, \ldots, U_n/t_n]$, which is nevertheless indispensable to know where to pick the C_i's from. We prefer the notation $[\![\sigma]\!]\theta\eta$.

It is interesting to note that the intriguing condition (CR2) is needed to show that (CR3) holds for sets of the form $[C \to D]$, and to show that Girard's conditions are stronger than conditions (S1) and (S2). Also, Girard does not need (S2) (from definition 8.1) in his proof of lemma 7.9, because he uses (CR1), (CR2), (CR3) and the fact that if a term M is SN, then there is an upper bound on the length of reduction sequences from M, as discussed in section 9. In effect, Girard uses (CR1), (CR2), (CR3) as a substitute for what we call (R2) in definition 7.8, in his proof of lemma 7.9. As we showed in section 9, Girard's conditions are stronger than conditions (S1) and (S2). We also remark that formulating lemma 7.9 in Girard's notation is rather cumbersome.

2. Stenlund's version (1972).

In [35], Stenlund presents a version of the proof of strong normalization for second-order intuitionistic logic using the method of candidates of reducibility (the theory of species). His proof is basically a typed version of Tait's proof discussed next. Stenlund eliminates condition (i) on page 247 of Tait [37] because it is redundant, and uses essentially our (S1) and (S2) of definition 7.8. The sets $[\![\sigma]\!]\theta\eta$ are defined basically as we do in definition 7.4. Stenlund's notation is easier to follow that Tait (and Girard), but lemma 7.7 is also not mentioned. The fact that in conditions (S1) and (S2), the expressions $N_1, \ldots, N_n$ must be allowed to be types as well as terms seems to have been overlooked. Nevertheless, this proof is fairly readable.

3. Tait's version (1973).

In [37], Tait proves a realizability result analogous to our lemma 6.8 for second-order intuitionistic logic (what he calls the theory of species) using the method of candidates of reducibility. As a consequence, Tait obtains a version of the proof of strong normalization for second-order intuitionistic logic (this is slightly more general than strong normalization for second-order propositional intuitionistic logic, which corresponds to system F). Tait takes advantage of the erasing trick, and he defines the candidates as sets of untyped

lambda terms. Actually, Tait's erasing function is not quite the one we use, because Tait uses conditions slightly different from our (S1), (S2) of definition 6.7. Tait assume that the untyped lambda calculus has a special constant K. Let SN denote the set of untyped lambda terms that are strongly normalizable. Then, a candidate of reducibility C is defined as a subset of SN satisfying the following properties:

(i) If $M \in C$ and $M \longrightarrow_\lambda N$, then $N \in C$;

(ii) For all $n \geq 0$ and all $M, N_1, \ldots, N_n \in \Lambda$, for every $N \in SN$, if $M[N/x]N_1 \ldots N_n \in C$, then $(\lambda x.\, M)NN_1 \ldots N_n \in C$.

(iii) For all $n \geq 0$ and all $N_1, \ldots, N_n \in SN$, $KN_1 \ldots N_n \in C$.

Condition (i) is Girard's (CR2), (ii) is basically our (S2), and (iii) is basically our (S1). Tait then proves Girard's trick (lemma 6.5) and lemma 6.8. He concludes by using the erasing trick that if M type-checks, then it is SN. As we see it, condition (i) is never used anywhere and appears to be redundant.

There are other technical differences with Girard's proof. First, Tait expands the language of types by adding a base type $\overline{C}$ for every candidate of reducibility C in $\mathcal{C}$. Then, Tait defines $[\![\sigma]\!]$ for all *closed* types over this extended language. There is no need for an explicit assignment η, since the new base types correspond to candidates in $\mathcal{C}$. The definition of $[\![\forall t.\, \sigma]\!]$ is worth noting:

$$[\![\forall t.\, \sigma]\!] = \{M \in \Lambda \mid \forall C \in \mathcal{C},\ MK \in [\![\sigma[\overline{C}/t]]\!]\},$$

where K is the special constant added to the untyped lambda calculus. Tait uses the following erasing function:

$Erase'(c) = c$, whenever $c \in \Sigma$,
$Erase'(x) = x$, whenever $x \in \mathcal{X}$,
$Erase'(MN) = Erase'(M)Erase'(N)$,
$Erase'(\lambda x{:}\,\sigma.\, M) = \lambda x.\, Erase'(M)$,
$Erase'(M\sigma) = Erase'(M)K$,
$Erase'(\Lambda t.\, M) = \lambda t.\, Erase'(M)$.

The difference between this erase function and ours is that we have $Erase(\Lambda t.\, M) = Erase(M)$, that is, the type abstraction is deleted, and we have $Erase(M\sigma) = Erase(M)$, whereas Tait uses the special constant K. Finally, in order to formulate and prove lemma 6.8, even though Tait was able to get away from using an explicit type assignment η, he is now forced to consider such assignments in the form of substitutions, which, in our opinion, is rather confusing. Given a raw term M that type-checks with proof $\Delta \triangleright M{:}\,\sigma$, if

$\mathcal{FV}(M) = \{t_1, \ldots, t_n\}$, Tait considers substitutions $[T_1/t_1, \ldots, T_n/t_n]$ of closed types for the free type variables in M, and forms $M_0 = M[T_1/t_1, \ldots, T_n/t_n]$ and $\sigma_0 = \sigma[T_1/t_1, \ldots, T_n/t_n]$. In effect, the substitution $[T_1/t_1, \ldots, T_n/t_n]$ plays the role of our assignment η.[18] Tait then proceeds basically as we do.

It should be noted that Tait does not actually justify the validity of the use of his erasing trick, and our lemma 6.6 is hidden in (2) on page 248 (however, Girard does prove explicitly a lemma analogous to our lemma 7.7). We also believe that case (3) in the proof of 4.2 on page 250 is erroneous, but it can be fixed easily (along the line of our proof).

4. Fortune, Leivant, O'Donnell's version (1983).

This version of the proof [8] can be considered as a typed version of Tait's proof for system F. Although it is not stated explicitly that the candidates are typed, this is the case since the conditions (G1), (G2), (G2'), (G3) on page 174 of [8] involve types and type abstraction. These conditions are basically our conditions (S1), (S2) of definition 8.1. As in Tait [37], the language of types is expanded by adding a base type $\overline{C}$ for every candidate of reducibility C in $\mathcal{C}$. The sets $[\![\sigma]\!]$ are only defined for *closed* types over this extended language (As far as we can see, definition 6.3.2 has no provision for assigning anything to the type variables). The definition of $[\![\forall t.\, \sigma]\!]$ is almost as in Tait:

$$[\![\forall t.\, \sigma]\!] = \{M \in \mathcal{P}\Lambda \mid \forall C \in \mathcal{C},\ M\overline{C} \in [\![\sigma[\overline{C}/t]]\!]\}.$$

However, we believe that there is a subtle problem with this clause of definition 6.3.2 (page 174) which invalidates the subsequent results. The problem is that in clause (S4) of definition 6.3.2, the sets $\{M \in \mathcal{P}\Lambda \mid \forall C \in \mathcal{C},\ M\overline{C} \in [\![\sigma[\overline{C}/t]]\!]\}$ are always **empty**, since the constants $\overline{C}$ do **not** belong to the original language, but yet the *grounds* (definition 6.3.1, page 173-174) are defined over the original language. Tait's argument does not suffer from this problem because $[\![\forall t.\, \sigma]\!]$ is a set of (untyped) terms over the original language ($\{M \in \Lambda \mid \forall C \in \mathcal{C},\ MK \in [\![\sigma[\overline{C}/t]]\!]\}$). Unfortunately, in Fortune, Leivant, O'Donnell, since $[\![\forall t.\, \sigma]\!] = \emptyset$, the subsequent lemmas are invalidated. Another minor problem arises from definition 6.4.1. Given a term M that type-checks, a *type instance* of M is a term M' obtained by substituting in M *base types* for the free type variables. This substitution essentially plays the role of our η. An *instance* of M is a substitution instance $\varphi(M')$ of M', where φ is a substitution of terms for the free variables (similar to our φ). But then, according to these definitions, it is not true that every term M is an instance of itself, because M' cannot contain any type variables. In particular, if M contains free type variables, since M' does not, $\varphi(M')$ will never be equal to M for any term substitution φ.

[18] It would be sufficient to consider substitutions of constants corresponding to the candidates.

Thus, as is, theorem 6.4.2 only holds for terms with no free type variables (but it is also false, due to the problem with definition 6.3.2). It seems difficult to fix the problem with definition 6.3.2 other than by using Girard's original definition of $[\![\forall t.\, \sigma]\!]$. Indeed, "easy" attempts to fix definition 6.3.2 seem to spoil the proof of lemma 6.3.1.

The next three versions are actually sketches of proofs, and they do not go through the various induction steps (and there are quite a few!).

5. Mitchell's version (1986).

In [24], Mitchell sketches a proof of strong normalization for system F, in which he introduces the erasing function *Erase* of definition 2.3, and an untyped version of the candidates of reducibility. Mitchell also introduces conditions (S1), (S2') (essentially our definition 6.9) and makes the clever observation that the range of applicability of the method can be broadened by relativizing the definition of saturated sets to a closed set S which is not necessarily SN (the set of lambda terms that are strongly normalizing). The definition of $[\![\sigma]\!]\eta$ given in definition 6.3 comes from lemma 4 of [24], and definition 6.9 is basically a reformulation of Mitchell's conditions. Mitchell states theorem 6.15, but does not state explicitly lemma 6.8, nor lemma 6.6.[19] Although lacking proofs, this paper is quite readable.

6. Huet's version (1987).

In [18], Huet sketches a proof of strong normalization for system F, using an untyped version of the candidates and the erasing trick, but using a special constant in condition (S1), as in Tait [37]. This proof was found by Coquand and Huet independently of Mitchell.[20] In our opinion, the role of the assignment η should be made more explicit in the definition of $[\![\sigma]\!]\eta$, and lemma 6.8 should be stated. Lemma 6.6 is not mentioned.

7. Scedrov's versions (1987, 1988).

In Scedrov [31], a very elegant and simple proof of normalization for $\lambda^{\forall}$ is presented. This proof is obtained by noticing two interesting facts. The first fact is that if one is simply interested in normalization (as opposed to strong normalization), then one can drop condition (S2) in the definition of the saturated sets, and instead require closure under β-conversion. Then, one can prove a version of lemma 6.8 stating that for every term M that type-checks, $Erase(M)$ is normalizable. The second fact already noted by Girard in his thesis ([10]), is that if $Erase(M) \xrightarrow{*}_{\lambda} Q$, then there is some term $P \in \mathcal{P}\Lambda$ such that $Erase(P) = Q$ and $M \xrightarrow{*}_{\lambda^{\forall}} P$. These two facts together yield normalization for all terms that type-check.

---

[19] However, Val Breazu-Tannen is in possession of some notes by Mitchell in which such a lemma is stated.

[20] Private communication from Thierry Coquand.

In [32], Scedrov gives an informal exposition of the proof of strong normalization for $\lambda^\forall$. His version is basically an expanded version of Mitchell's sketch, using the erasing trick. In our opinion, the role of the assignment η should be made more explicit in the definition of $[\![\sigma]\!]\eta$, and lemma 3.1 from [32] (our lemma 6.8) should be stated more clearly.

8. Related Proofs.

Other proofs of normalization or strong normalization for various natural deduction systems based on Girard's method have been published. Since they are not specifically about polymorphic lambda calculi, we will simply list them without further comments. Prawitz [29] proves strong normalization for classical first-order logic (natural deduction), and intuitionistic second-order logic (natural deduction). It interesting to observe that the notion of strong validity introduced in section 3.2 of Appendix A, and in section B.2 of Appendix B of [29], is essentially equivalent to the definition of the sets $[\![\sigma]\!]\eta$. Martin-Löf proves normalization results for various proof systems, including the Theory of Intuitionistic Iterated Inductive Definitions, Second-Order Intuitionistic Logic, and Intuitionistic Simple Type Theory [21, 22, 23]. The result in [23] is significantly strengthened in Girard [10]. We should also mention that Leivant [20] has given an interesting semantic generalization of Girard's technique. This allows him to prove various properties of terms in $\lambda^\forall$, including normalization, strong normalization, and solvability. Finally, in the case of the simply-typed lambda calculus, a radically different proof of strong normalization has been given by de Vrijer [40].

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 $\sigma \in \mathcal{TC}$,
$(\lambda t\colon K.\, \sigma) \in \mathcal{T}$, 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 $\Pi_K(\lambda t\colon K.\sigma)$ will also be denoted as $\forall t\colon K.\, \sigma$. 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 $\Theta\colon \Sigma \to \mathcal{T}$ 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 $M, N \in \mathcal{P}\Lambda$,
$(\lambda x\colon \sigma.\, M) \in \mathcal{P}\Lambda$, 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 $M \in \mathcal{P}\Lambda$,
$(\Lambda t\colon K.\, M) \in \mathcal{P}\Lambda$, 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, $M_1 M_2 \ldots M_{n-1} M_n$ is an abbreviation for $((\ldots(M_1 M_2)\ldots M_{n-1})M_n)$.

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 $dom(\varphi) = \{x_1, \ldots, x_n\}$ and $\varphi(x_i) = u_i$ for every i, $1 \leq i \leq n$, the substitution φ is also denoted by $[u_1/x_1, \ldots, u_n/x_n]$.

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:

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

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 $M_1 = \Lambda t_1\colon \star.\, \lambda x_1\colon t_1.\, x_1$ and $M_2 = \Lambda t_2\colon \star.\lambda x_2\colon t_2.x_2$ 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.

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

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:

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

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

$$\begin{aligned}
safe(\varphi, x) &= \textbf{true}, && x \in \mathcal{X},\\
safe(\varphi, t) &= \textbf{true}, && t \in \mathcal{V},\\
safe(\varphi, f) &= \textbf{true}, && f \in \Sigma,\\
safe(\varphi, \sigma) &= \textbf{true}, && \sigma \in \mathcal{TC},\\
safe(\varphi, \lambda t{:}\, K.\, \sigma) &= safe(\varphi_{-t}, \sigma) \textbf{ and } t \notin \mathcal{FV}(\varphi), && \sigma \in \mathcal{T}, K \in \mathcal{K}, t \in \mathcal{V},
\end{aligned}$$

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

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 $\Delta\colon \mathcal{V} \cup \mathcal{X} \to \mathcal{K} \cup \mathcal{T}$ 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 $t_i\colon K_i$ or $x_j\colon \sigma_j$, 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 $\Delta, t\colon K$ for $\Delta \cup \{\langle t, K\rangle\}$, and $\Delta, x\colon \sigma$ 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*
$\vdash \Delta \triangleright \sigma\colon K$	σ *kind-checks* with kind K
$\vdash \Delta \triangleright M\colon \sigma$	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 $\Delta \triangleright \sigma\colon K$, called *kinding judgments*, is the following:

Axiom:

$$\emptyset \triangleright$$

Inference Rules:

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

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

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

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

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

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:

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

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

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

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

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

We define $\longrightarrow_{\lambda\to} = (\longrightarrow_\beta \cup \longrightarrow_\eta)$, and *reduction* as the reflexive and transitive closure $\xrightarrow{*}_{\lambda\to}$ of $\longrightarrow_{\lambda\to}$. We also define *immediate conversion* $\longleftrightarrow_{\lambda\to}$ such that $\longleftrightarrow_{\lambda\to} = \longrightarrow_{\lambda\to} \cup \longrightarrow_{\lambda\to}^{-1}$, and *conversion* as the reflexive and transitive closure $\xleftrightarrow{*}_{\lambda\to}$ 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 $\Delta \triangleright M{:}\, \sigma$, called *typing judgments*, is the following:

Axioms: For every context Δ that kind-checks,

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

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

Inference Rules:

$$\frac{\Delta \rhd M\colon \sigma \Rightarrow \tau \qquad \Delta \rhd N\colon \sigma}{\Delta \rhd MN\colon \tau} \qquad (\textit{application})$$

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

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

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

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

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

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

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

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

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

Clearly, if $\Delta \rhd M\colon \sigma$ is provable, then $\Delta \rhd \sigma\colon \star$ and $\Delta \rhd$ are also provable. It is not difficult to show that if two typing judgments $\Delta \rhd M\colon \sigma$ and $\Delta' \rhd M'\colon \sigma'$ are α-equivalent and there is a proof $\vdash \Delta \rhd M\colon \sigma$, then there is a proof $\vdash \Delta' \rhd M'\colon \sigma'$. 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 $\forall u{:}\star.\, (u \Rightarrow u) = \Pi_\star(\lambda u{:}\star.\, (u \Rightarrow u))$.

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

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,

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

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

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

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

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

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 $\longleftrightarrow_{F_\omega} = \longrightarrow_{F_\omega} \cup \longrightarrow_{F_\omega}^{-1}$, 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

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

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

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

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 $\Delta \triangleright \sigma{:}K$, where $\Delta \triangleright \sigma{:}K$ 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 $\Delta \triangleright \sigma{:}K$ is a set of provable typing judgments of the form $\Delta' \triangleright M{:}\sigma$ where $\Delta \subseteq \Delta'$, and satisfying certain properties as in definition 7.8. If this approach is followed, it is also necessary to define $[\![\Delta \triangleright \sigma{:}K]\!]\theta\eta$, 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:

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

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 $\sigma \xleftrightarrow{*}_{\lambda\to} \sigma'$, then $[\![\sigma]\!]\theta\eta = [\![\sigma']\!]\theta\eta$.

Proof. It is sufficient to prove that if $\sigma \xrightarrow{*}_{\lambda\to} \sigma'$, then $[\![\sigma]\!]\theta\eta = [\![\sigma']\!]\theta\eta$. The proof proceeds by induction on the proof that $\sigma \xrightarrow{*}_{\lambda\to} \sigma'$. 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 $\lambda t\colon K.\, \sigma$ is safe for θ. In this case, we use lemma 16.7 and the fact that $(\lambda t\colon K.\, \theta(\sigma))\tau \longrightarrow_{\lambda\to} \theta(\sigma)[\tau/t]$, and that because $\lambda t\colon K.\, \sigma$ 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]

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

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

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

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

(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

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

Lemma 7.9 generalizes to F_ω as follows.

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

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 $\varphi\colon \Gamma \to \Delta$, 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 $\Gamma \triangleright M\colon \sigma$. Type conversion is handled using lemma 16.7. We only sketch the verification for two of the other cases.

Case 1.

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

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 $\Gamma, t\colon K \triangleright M\colon \sigma t$ and to any candidate assignment $\langle \theta[t := \tau], \eta[t := C] \rangle$ where $C \in \mathcal{C}_\tau$ and $\tau\colon K \in \mathcal{T}|_\kappa$ (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 $\theta(\Delta), \kappa \triangleright \varphi(\theta(M)) \in \bigcup \mathcal{C}_{\theta(\sigma t)}$. Then, by (R2)(ii),

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

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

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

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

Case 2.

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

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

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

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

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

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.

---

[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.

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 $can_\sigma = S_\sigma \in \mathcal{C}_\sigma$ 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 $x \in \mathcal{X}$,
$Erase(MN) = Erase(M)Erase(N)$,
$Erase(\lambda x{:}\sigma.\ M) = \lambda x.\ Erase(M)$,

[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.

$Erase(M\sigma) = Erase(M)$,
$Erase(\lambda t{:}K.\, M) = Erase(M)$.

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.

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 $[\![\Delta \triangleright \sigma{:}K]\!]\theta\eta$ and $[\![\Delta \triangleright K{:}Kind]\!]\theta\eta$. 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.

17 Appendix 1: Product Types in F_ω

In this section, we extend the system F_ω by allowing product types. An even more general system with disjunctive and existential types was investigated by Girard [10].

In the formula–as–type analogy, product types correspond to conjunctions, and the typing rules correspond to the introduction and elimination rules for conjunction. We will be dealing with product types with surjective pairing.

The definition of the kinds given in definition 12.1 remains unchanged. However, it is assumed that every set $\mathcal{TC}$ of type constructors contains the special symbols $\times$, $\Rightarrow$ and Π_K for every $K \in \mathcal{K}$. The type constructor $\times$ is the *product type* constructor. The type constructors are assigned kinds by a kind signature.

Definition 17.1 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 $\Xi(\Rightarrow) = \star \to (\star \to \star)$, $\Xi(\times) = \star \to (\star \to \star)$, and $\Xi(\Pi_K) = (K \to \star) \to \star$.

The definition of raw types remains unchanged, but since $\times$ belongs to every set $\mathcal{TC}$ of type constructors, more raw types are allowed. The definition is repeated for the reader's convenience.

Definition 17.2 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 $\times$ belong to $\mathcal{TC}$, by the last clause, $((\times\sigma)\tau)$ is a raw type for all $\sigma, \tau \in \mathcal{T}$, For simplicity of notation, $((\times\sigma)\tau)$ is denoted as $(\sigma \times \tau)$. 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.

Next, we define the polymorphic raw terms. There is no change in the definition of a type signature.

Definition 17.3 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 $M, N \in \mathcal{P}\Lambda$,
$\langle M, N\rangle \in \mathcal{P}\Lambda$, whenever $M, N \in \mathcal{P}\Lambda$,
$\pi_1(M), \pi_2(M) \in \mathcal{P}\Lambda$, whenever $M \in \mathcal{P}\Lambda$,
$(\lambda x{:}\, \sigma.\, M) \in \mathcal{P}\Lambda$, 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 notions of substitution and α-equivalence are extended in the obvious way. In order to deal with product types, it is necessary to add the following kind-checking rule:

$$\frac{\Delta \rhd \sigma{:}\,\star \qquad \Delta \rhd \tau{:}\,\star}{\Delta \rhd \sigma \times \tau{:}\,\star} \qquad (\times)$$

The definition of the relation $\longrightarrow_{\lambda\to}$ does not have to be changed, since the congruence rule takes care of $\Rightarrow$, $\times$, and Π_K.

It is easy to see that corollary 6.18 and corollary 6.19 hold for the new class of types. Thus, every ($\equiv_\alpha$-equivalence class of) type σ that kind-checks has a unique $\beta\eta$-normal form.

The following inference rules need to be added to the proof system used for type-checking terms.

$$\frac{\Delta \triangleright M\colon \sigma \qquad \Delta \triangleright N\colon \tau}{\Delta \triangleright \langle M, N\rangle\colon \sigma \times \tau} \qquad (\textit{product})$$

$$\frac{\Delta \triangleright M\colon \sigma \times \tau}{\Delta \triangleright \pi_1(M)\colon \sigma} \qquad \frac{\Delta \triangleright M\colon \sigma \times \tau}{\Delta \triangleright \pi_2(M)\colon \tau} \qquad (\textit{projection})$$

The notion of reduction in F_ω is defined by adding the following axioms and rules to definition 15.1.

Axioms:

$$\pi_1(\langle M, N\rangle) \longrightarrow_\pi M, \qquad (\pi)$$

$$\pi_2(\langle M, N\rangle) \longrightarrow_\pi N, \qquad (\pi)$$

$$\langle \pi_1(M), \pi_2(M)\rangle \longrightarrow_{\langle\rangle} M, \qquad (\langle\rangle)$$

Inference Rules: For each kind of reduction $\longrightarrow_r$ where $r \in \{\beta, \eta, \pi, \langle\rangle, \tau\beta, \tau\eta, \lambda^{\to}\}$,

$$\frac{M \longrightarrow_r N}{\langle M, Q\rangle \longrightarrow_r \langle N, Q\rangle} \qquad \frac{M \longrightarrow_r N}{\langle P, M\rangle \longrightarrow_r \langle P, N\rangle} \qquad \text{for all } P, Q \in \mathcal{P}\Lambda$$

$$\frac{M \longrightarrow_r N}{\pi_1(M) \longrightarrow_r \pi_1(N)} \qquad \frac{M \longrightarrow_r N}{\pi_2(M) \longrightarrow_r \pi_2(N)} \qquad \text{for all } P, Q \in \mathcal{P}\Lambda$$

We now generalize the method of candidates to F_ω with product types (with surjective pairing). As the proof given in section 16, the proof presented next 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 families of closed Girard sets. It should be noted that in the case of the simply-typed lambda calculus, a very similar method (but simpler, since only simple types need to be handled) has been used to give proofs of strong normalization, by Lambek and Scott [19], and de Vrijer [38,39].

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 $S \times T$ be the subset of $\mathcal{PT}_{\sigma\times\tau}$ defined as before:

$$S \times T = \{\Delta \triangleright M \in \mathcal{PT}_{\sigma\times\tau} |\ \Delta \triangleright \pi_1(M) \in S \quad \text{and} \quad \Delta \triangleright \pi_2(M) \in T\}.$$

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

Definition 17.4 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 $K_1 \to K_2$,

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

(5) 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 $C \times D \in \mathcal{C}_{\sigma \times \tau}$.

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

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

Definition 17.5 Given any candidate assignment $\langle \theta, \eta \rangle$, for every type $\sigma \in \mathcal{T}|_\kappa$, we define $[\![\sigma]\!]\theta\eta$ as follows:

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

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.

Lemma 16.5, 16.6, and 16.7 are unchanged. It is also easy to prove the following version of "Girard's trick" for F_ω with product types.

Lemma 17.6 (Girard) If $\mathcal{C}$ is a $\mathcal{T}|_\kappa$-closed family, for every candidate assignment $\langle\theta, \eta\rangle$, for every type σ, then $[\![\sigma]\!]\theta\eta \in \mathcal{C}_{\theta(\sigma)}$. □

One more condition needs to be added to the conditions of definition 16.9

Definition 17.7 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.[26]

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

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

R3. For all $\sigma{:}\star, \tau{:}\star \in \mathcal{T}|_\kappa$, for every $C \in \mathcal{C}_\sigma$ and $D \in \mathcal{C}_\tau$, if $\Delta \triangleright M \in C$ and $\Delta \triangleright N \in D$, then $\Delta \triangleright \langle M, N\rangle \in C \times D$.

We now have a version of lemma 16.10 for F_ω with product types.

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

Lemma 17.8 (Girard) Let $\mathcal{C} = (\mathcal{C}_\sigma)_{\sigma \in \mathcal{T}|_\kappa}$ be a $\mathcal{T}|_\kappa$-indexed family of sets of candidates of reducibility. For every $\Gamma \triangleright M \in \mathcal{PT}_\sigma$, every candidate assignment $\langle \theta, \eta \rangle$, every substitution $\varphi\colon \Gamma \to \Delta$, 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. We only sketch the verification for the new cases.

Case 1.

$$\frac{\Gamma \triangleright M\colon \sigma \qquad \Gamma \triangleright N\colon \tau}{\Gamma \triangleright \langle M, N\rangle\colon \sigma \times \tau} \qquad (\textit{product})$$

By the induction hypothesis,

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

and

$$\theta(\Delta), \kappa \triangleright \varphi(\theta(N)) \in [\![\tau]\!]\theta\eta.$$

By (R3) and the definition of $[\![\sigma \times \tau]\!]\theta\eta$, we have

$$\theta(\Delta), \kappa \triangleright \langle \varphi(\theta(M)), \varphi(\theta(N))\rangle \in [\![\sigma \times \tau]\!]\theta\eta.$$

However, this is equivalent to

$$\theta(\Delta), \kappa \triangleright \varphi(\theta(\langle M, N\rangle)) \in [\![\sigma \times \tau]\!]\theta\eta.$$

Case 2.

$$\frac{\Gamma \triangleright M\colon \sigma \times \tau}{\Gamma \triangleright \pi_1(M)\colon \sigma} \qquad \frac{\Gamma \triangleright M\colon \sigma \times \tau}{\Gamma \triangleright \pi_2(M)\colon \tau} \qquad (\textit{projection})$$

By the induction hypothesis,

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

Since $[\![\sigma \times \tau]\!]\theta\eta = [\![\sigma]\!]\theta\eta \times [\![\tau]\!]\theta\eta$, this implies that $\theta(\Delta), \kappa \triangleright \varphi(\theta(\pi_1(M))) \in [\![\sigma]\!]\theta\eta$ and $\theta(\Delta), \kappa \triangleright \varphi(\theta(\pi_2(M))) \in [\![\tau]\!]\theta\eta$. □

Remarkably, because Girard's conditions take the reduction relation $\longrightarrow_{F_\omega}$ into account, there is no need to add extra conditions besides (CR0), (CR1), (CR2), and (CR3). We simply need to modify the definition of a simple term, so that lemma 16.13 holds for F_ω with product types. For this, it is enough to preclude a pair $\langle M, N\rangle$ from being simple. Thus, a term M is *simple* iff it is either a variable x, a constant $f \in \Sigma$, an application MN, a projection $\pi_1(M)$ or $\pi_2(M)$, or a type application $M\tau$.

Definition 17.9 Let $S = (S_\sigma)_{\sigma:\star\in\mathcal{T}|_\kappa}$ be a family such that each S_σ is a nonempty subset of $\mathcal{PT}_\sigma$.[27] 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:[28]

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 in the above definition, $\longrightarrow_{F_\omega}$ is the reduction relation for F_ω with product types, and consequently, covers the case of reductions when M is of the form $\pi_1(\langle P, Q\rangle)$ or $\pi_2(\langle P, Q\rangle)$.

We need to add one more clause to definition 16.12 (defining a closed family): for all $\sigma:\star, \tau:\star \in \mathcal{T}|_\kappa$, if $\Delta \triangleright M \in \mathcal{PT}_{\sigma\times\tau}$, $\Delta \triangleright \pi_1(M) \in S_\sigma$, and $\Delta \triangleright \pi_2(M) \in S_\tau$, then $\Delta \triangleright M \in S_{\sigma\times\tau}$.

We have the following generalization of lemma 16.13.

Lemma 17.10 (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 17.4. 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. Property (R3) is shown as follows. Assume that $\Delta \triangleright M \in C$ and $\Delta \triangleright N \in D$. We show by induction on $\delta(M) + \delta(N)$ that $\Delta \triangleright Q \in C$ whenever $\pi_1(\langle M, N\rangle) \longrightarrow_{F_\omega} Q$, and that $\Delta \triangleright Q \in D$ whenever $\pi_2(\langle M, N\rangle) \longrightarrow_{F_\omega} Q$. Then, by (CR3), we have $\Delta \triangleright \pi_1(\langle M, N\rangle) \in C$ and $\Delta \triangleright \pi_2(\langle M, N\rangle) \in D$, which, by the definition of $C \times D$, shows that $\Delta \triangleright \langle M, N\rangle \in C \times D$.

We prove that $\Delta \triangleright Q \in C$ whenever $\pi_1(\langle M, N\rangle) \longrightarrow_{F_\omega} Q$, the other case being similar. The point is that either $Q = M$, or $Q = \pi_1(\langle M', N\rangle)$ where $M \longrightarrow_{F_\omega} M'$, or $Q = \pi_1(\langle M, N'\rangle)$ where $N \longrightarrow_{F_\omega} N'$. Note that the case where $M = \pi_1(U)$ and $N = \pi_2(U)$ must be considered, since in this case $\langle \pi_1(U), \pi_2(U)\rangle \longrightarrow_{F_\omega} U$. But then,

$$\pi_1(\langle M, N\rangle) = \pi_1(\langle \pi_1(U), \pi_2(U)\rangle) \longrightarrow_{F_\omega} \pi_1(U) = M.$$

[27] 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 $\star$.

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

In the first case, the hypothesis yields $\Delta \triangleright M \in C$. In the other two cases, by (CR2) we have $\Delta \triangleright M' \in C$ and $\Delta \triangleright N' \in D$, and since $\delta(M') < \delta(M)$ and $\delta(N') < \delta(N)$, we use the induction hypothesis and (CR3). The base case where M and N are irreducible follows from (CR3).

It is also necessary to verify conditions (1), (2), (3), (4), (5) of definition 17.4. This is done by induction on kinds, and for the kind $\star$ by induction on types. The only new case is case (5). Given that $C \subseteq S_\sigma$ and $D \subseteq S_\tau$, the fact that $C \times D \subseteq S_{\sigma\times\tau}$ follows from the new closure condition (on $\times$). We need to prove that (CR1), (CR2), and (CR3) hold for $C \times D$, given that they hold for C and D. Let $\Delta \triangleright M\colon \sigma \times \tau$ be a term in $C \times D$, where $C \in \mathcal{C}_\sigma$ and $D \in \mathcal{C}_\tau$. By the definition of $C \times D$, $\Delta \triangleright \pi_1(M) \in C$ (and $\Delta \triangleright \pi_2(M) \in D$). Since all terms in C are SN, $\pi_1(M)$ is SN. But then, M itself is necessary SN since any infinite reduction from M yields an infinite reduction from $\pi_1(M)$.

Assume that $M \longrightarrow_{F_\omega} M'$. Then, $\pi_1(M) \longrightarrow_{F_\omega} \pi_1(M')$ and $\pi_2(M) \longrightarrow_{F_\omega} \pi_2(M')$. Since $\pi_1(M) \in C$ and $\pi_2(M) \in D$, by (CR2) applied to C and D, we have $\pi_1(M') \in C$ and $\pi_2(M') \in D$. By the definition of $C \times D$, we have $\Delta \triangleright M' \in C \times D$, and (CR2) holds.

Now, assume that M is simple, and that whenever $M \longrightarrow_{F_\omega} Q$, then $\Delta \triangleright Q \in C \times D$. We want to prove that $\Delta \triangleright M \in C \times D$. Note that $\pi_1(M)$ and $\pi_2(M)$ are also simple, and that because M is simple,

(∗) $\pi_1(M) \longrightarrow_{F_\omega} R$ implies that $R = \pi_1(Q)$ where $M \longrightarrow_{F_\omega} Q$, and similarly for $\pi_2(M)$.

Since we assumed that $\Delta \triangleright Q \in C \times D$, we have $\Delta \triangleright \pi_1(Q) \in C$ and $\Delta \triangleright \pi_2(Q) \in D$. By (CR3) applied to C and D and (∗), since $\pi_1(M)$ and $\pi_2(M)$ are simple, we have $\Delta \triangleright \pi_1(M) \in C$ and $\Delta \triangleright \pi_2(M) \in D$. By the definition of $C \times D$, we have $\Delta \triangleright M \in C \times D$, and (CR3) holds. □

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

Theorem 17.11 (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 17.10, and assume that $S_\sigma \in \mathcal{C}_\sigma$ for every $\sigma\colon\star \in \mathcal{T}|_\kappa$. For every $\Delta\triangleright M \in \mathcal{PT}_\sigma$, we have $\Delta\triangleright M \in S_\sigma$.

Proof. By lemma 17.10, $\mathcal{C}$ is a family of sets of candidates of reducibility. We now apply lemma 17.8 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.[29] □

[29] 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.

The next lemma shows that F_ω with product types is strongly normalizing and confluent under both β-reduction and $\beta\eta$-reduction.

Lemma 17.12 (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.

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, with a few more cases involving pairs and projections. □

18 Appendix 2

This appendix contains the details that were omitted in the proof of lemma 10.2.

Proof of lemma 10.2. We need to prove that the family of sets in (iii) and (iv) is a closed family of saturated sets.

Closure is obvious since if confluence holds from Mx and all of its subterms, then it holds from M.

Verifying (S1) is easy and uses the fact that if confluence holds from $N_1, \ldots, N_n$ and all of their subterms, then confluence holds from each $uN_1 \ldots N_k$ and all of its subterms, where $u \in \mathcal{X} \cup \Sigma$ and $1 \leq k \leq n$. This is because reductions must apply withing the N_i's. Since confluence from u is trivial, confluence from each subterm of $uN_1 \ldots N_n$ follows from the assumption on $N_1, \ldots, N_n$.

Proving (S2) is very tedious. Assume that confluence holds from $M[N/x]N_1 \ldots N_n$ and all of its subterms (and that confluence holds from N, which is implied by (S2)). Thus, confluence holds from $M, N, N_1, \ldots, N_n$. We need to show that confluence holds from $(\lambda x{:}\sigma.\, M)NN_1 \ldots N_n$ and all of its subterms.

First, we show confluence from every subterm of $\lambda x{:}\sigma.\, M$. Since confluence holds from every subterm of M, the only nontrivial case is the case where $M \xrightarrow{*}_{\lambda^\forall} (M_1' x)$, for some

term M_1' such that $x \notin FV(M_1')$. In this case, we can have reductions of the form

$$\lambda x{:}\sigma.\, M \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, (M_1' x) \longrightarrow_\eta M_1' \xrightarrow{*}_{\lambda^\forall} P_1,$$

where $M \xrightarrow{*}_{\lambda^\forall} M_1' x$. There are four cases to consider.

Case 1.

$$\lambda x{:}\sigma.\, M \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, (M_1' x) \longrightarrow_\eta M_1' \xrightarrow{*}_{\lambda^\forall} P_1$$

and

$$\lambda x{:}\sigma.\, M \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, (M_1'' x) \longrightarrow_\eta M_1'' \xrightarrow{*}_{\lambda^\forall} P_2,$$

where $M \xrightarrow{*}_{\lambda^\forall} M_1' x$ and $M \xrightarrow{*}_{\lambda^\forall} M_1'' x$. Since confluence holds from M, there are reductions $P_1 x \xrightarrow{*}_{\lambda^\forall} Q$ and $P_2 x \xrightarrow{*}_{\lambda^\forall} Q$ for some Q.

If both reductions are of the form $P_1 x \xrightarrow{*}_{\lambda^\forall} P_3 x$ and $P_2 x \xrightarrow{*}_{\lambda^\forall} P_3 x$, for some P_3 such that $Q = P_3 x$, $P_1 \xrightarrow{*}_{\lambda^\forall} P_3$, and $P_2 \xrightarrow{*}_{\lambda^\forall} P_3$, then confluence holds.

If $P_2 \xrightarrow{*}_{\lambda^\forall} P_3$ and $P_1 x \xrightarrow{*}_{\lambda^\forall} P_3 x$ is of the form

$$P_1 x \xrightarrow{*}_{\lambda^\forall} (\lambda y{:}\sigma.\, Q_1) x \longrightarrow_\beta Q_1[x/y] \xrightarrow{*}_{\lambda^\forall} P_3 x,$$

since $x \notin FV(M_1'')$ due to the η-step, and since $\lambda y{:}\sigma.\, Q_1 \equiv_\alpha \lambda x{:}\sigma.\, Q_1[x/y]$, we have $x \notin FV(P_3)$ and

$$P_1 \xrightarrow{*}_{\lambda^\forall} \lambda y{:}\sigma.\, Q_1 \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, (P_3 x) \longrightarrow_\eta P_3.$$

Confluence holds, since $P_2 \xrightarrow{*}_{\lambda^\forall} P_3$.

The subcase where $P_1 \xrightarrow{*}_{\lambda^\forall} P_3$ and

$$P_2 x \xrightarrow{*}_{\lambda^\forall} (\lambda y{:}\sigma.\, Q_2) x \longrightarrow_\beta Q_2[x/y] \xrightarrow{*}_{\lambda^\forall} P_3 x,$$

is symmetric.

If both

$$P_1 x \xrightarrow{*}_{\lambda^\forall} (\lambda y{:}\sigma.\, Q_1) x \longrightarrow_\beta Q_1[x/y] \xrightarrow{*}_{\lambda^\forall} Q,$$

and

$$P_2 x \xrightarrow{*}_{\lambda^\forall} (\lambda y{:}\sigma.\, Q_2) x \longrightarrow_\beta Q_2[x/y] \xrightarrow{*}_{\lambda^\forall} Q,$$

since $\lambda y{:}\sigma.\, Q_1 \equiv_\alpha \lambda x{:}\sigma.\, Q_1[x/y]$ and $\lambda y{:}\sigma.\, Q_2 \equiv_\alpha \lambda x{:}\sigma.\, Q_2[x/y]$, we have

$$P_1 \xrightarrow{*}_{\lambda^\forall} \lambda y{:}\sigma.\, Q_1 \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, Q$$

and

$$P_2 \xrightarrow{*}_{\lambda^\forall} \lambda y{:}\sigma.\, Q_2 \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, Q.$$

Case 2.

$$\lambda x{:}\sigma.\, M \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, (M_1' x) \longrightarrow_\eta M_1' \xrightarrow{*}_{\lambda^\forall} P_1$$

and

$$\lambda x{:}\sigma.\, M \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, M_1'',$$

where $M \xrightarrow{*}_{\lambda^\forall} M_1' x$ and $M \xrightarrow{*}_{\lambda^\forall} M_1''$. This is quite similar to case 1. If $P_1 x \xrightarrow{*}_{\lambda^\forall} P_3 x$ and $M_1'' \xrightarrow{*}_{\lambda^\forall} P_3 x$, since $x \notin FV(M_1')$ due to the η-step, we have $x \notin FV(P_3)$, $P_1 \xrightarrow{*}_{\lambda^\forall} P_3$, and

$$\lambda x{:}\sigma.\, M_1'' \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, (P_3 x) \longrightarrow_\eta P_3.$$

Confluence holds since $P_1 \xrightarrow{*}_{\lambda^\forall} P_3$.

If

$$P_1 x \xrightarrow{*}_{\lambda^\forall} (\lambda y{:}\sigma.\, Q_1)x \longrightarrow_\beta Q_1[x/y] \xrightarrow{*}_{\lambda^\forall} Q,$$

and

$$M_1'' \xrightarrow{*}_{\lambda^\forall} Q,$$

since $\lambda x{:}\sigma.\, Q_1[x/y] \equiv_\alpha \lambda y{:}\sigma.\, Q_1$, we have

$$P_1 \xrightarrow{*}_{\lambda^\forall} \lambda y{:}\sigma.\, Q_1 \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, Q$$

and

$$\lambda x{:}\sigma.\, M_1'' \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, Q.$$

Case 3. Symmetric to case 2.

Case 4.

$$\lambda x{:}\sigma.\, M \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, M_1'$$

and

$$\lambda x{:}\sigma.\, M \xrightarrow{*}_{\lambda^\forall} \lambda x{:}\sigma.\, M_1'',$$

where $M \xrightarrow{*}_{\lambda^\forall} M_1'$ and $M \xrightarrow{*}_{\lambda^\forall} M_1''$. Since confluence holds from M, we conclude immediately.

We now prove that confluence holds from every term $(\lambda x{:}\sigma.\, M)NN_1 \ldots N_i$, where $1 \leq i \leq n$, and from $(\lambda x{:}\sigma.\, M)N$. Without loss of generality, we can assume that $i = n$. The proof of lemma 6.16 showed that every reduction sequence from $(\lambda x{:}\sigma.\, M)NN_1 \ldots N_n$ is of the form

$$(\lambda x{:}\sigma.\, M)NN_1 \ldots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M')N'N_1' \ldots N_n',$$

or

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M')N'N'_1 \dots N'_n \longrightarrow_\beta M'[N'/x]N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} Q,$$

or

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, (M'_1 x))N'N'_1 \dots N'_n \longrightarrow_\eta M'_1 N'N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} Q,$$

where in (1)-(2) $M \xrightarrow{*}_{\lambda^\forall} M'$, in (3) $M \xrightarrow{*}_{\lambda^\forall} M'_1 x$, and in all cases $N \xrightarrow{*}_{\lambda^\forall} N'$, and $N_i \xrightarrow{*}_{\lambda^\forall} N'_i$, for every i, $1 \leq i \leq n$. We have seven main cases.

Case 1.

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M')N'N'_1 \dots N'_n$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M'')N''N''_1 \dots N''_n.$$

Since confluence holds from $M, N, N_1 \dots N_n$ (for N, this is implied by (S2)), we have reduction sequences $M' \xrightarrow{*}_{\lambda^\forall} M'''$, $M'' \xrightarrow{*}_{\lambda^\forall} M'''$, $N' \xrightarrow{*}_{\lambda^\forall} N'''$, $N'' \xrightarrow{*}_{\lambda^\forall} N'''$, and $N'_i \xrightarrow{*}_{\lambda^\forall} N'''_i$, $N''_i \xrightarrow{*}_{\lambda^\forall} N'''_i$, for every i, $1 \leq i \leq n$, and thus reductions

$$(\lambda x{:}\sigma.\, M')N'N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M''')N'''N'''_1 \dots N'''_n$$

and

$$(\lambda x{:}\sigma.\, M'')N''N''_1 \dots N''_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M''')N'''N'''_1 \dots N'''_n.$$

Case 2.

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M')N'N'_1 \dots N'_n$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M'')N''N''_1 \dots N''_n \longrightarrow_\beta M''[N''/x]N''_1 \dots N''_n \xrightarrow{*}_{\lambda^\forall} P_2.$$

This time, we also have reduction sequences

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \longrightarrow_\beta M[N/x]N_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} M'[N'/x]N'_1 \dots N'_n$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \longrightarrow_\beta M[N/x]N_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} M''[N''/x]N''_1 \dots N''_n \xrightarrow{*}_{\lambda^\forall} P_2.$$

Using the confluence from $M[N/x]N_1 \dots N_n$, we have $M'[N'/x]N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} P_3$ and $P_2 \xrightarrow{*}_{\lambda^\forall} P_3$ for some P_3. Thus, we have reductions

$$(\lambda x{:}\sigma.\, M')N'N'_1 \dots N'_n \longrightarrow_\beta M'[N'/x]N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} P_3$$

and

$$P_2 \xrightarrow{*}_{\lambda^\forall} P_3.$$

Case 3.

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M')N'N'_1 \dots N'_n \longrightarrow_\beta M'[N'/x]N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} P_1$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M'')N''N''_1 \dots N''_n.$$

Symmetric to case 2.

Case 4.

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M')N'N'_1 \dots N'_n \longrightarrow_\beta M'[N'/x]N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} P_1$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M'')N''N''_1 \dots N''_n \longrightarrow_\beta M''[N''/x]N''_1 \dots N''_n \xrightarrow{*}_{\lambda^\forall} P_2.$$

As in case 2, we have reductions

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \longrightarrow_\beta M[N/x]N_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} M'[N'/x]N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} P_1$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \longrightarrow_\beta M[N/x]N_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} M''[N''/x]N''_1 \dots N''_n \xrightarrow{*}_{\lambda^\forall} P_2,$$

and we conclude using the confluence from $M[N/x]N_1 \dots N_n$.

Case 5.

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, (M'_1 x))N'N'_1 \dots N'_n \longrightarrow_\eta M'_1 N'N'_1 \dots N'_n \xrightarrow{*}_{\lambda^\forall} P_1,$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, (M''_1 x))N''N''_1 \dots N''_n \longrightarrow_\eta M''_1 N''N''_1 \dots N''_n \xrightarrow{*}_{\lambda^\forall} P_2,$$

where $M \xrightarrow{*}_{\lambda^\forall} M_1' x$, $M \xrightarrow{*}_{\lambda^\forall} M_1'' x$. Because of the η-steps, $x \notin FV(M_1')$ and $x \notin FV(M_1'')$, and thus $M_1'[N'/x] = M_1'$ and $M_1''[N''/x] = M_1''$, and we have reductions

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \longrightarrow_\beta M[N/x]N_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} M_1' N' N_1' \dots N_n' \xrightarrow{*}_{\lambda^\forall} P_1,$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \longrightarrow_\beta M[N/x]N_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} M_1'' N'' N_1'' \dots N_n'' \xrightarrow{*}_{\lambda^\forall} P_2,$$

and we conclude using the confluence from $M[N/x]N_1 \dots N_n$.

Case 6.

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, (M_1' x))N' N_1' \dots N_n' \longrightarrow_\eta M_1' N' N_1' \dots N_n' \xrightarrow{*}_{\lambda^\forall} P_1,$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} (\lambda x{:}\sigma.\, M_1'' N'')N_1'' \dots N_n'' \longrightarrow_\beta M_1''[N''/x]N_1'' \dots N_n'' \xrightarrow{*}_{\lambda^\forall} P_2,$$

where $M \xrightarrow{*}_{\lambda^\forall} M_1' x$, $M \xrightarrow{*}_{\lambda^\forall} M_1''$. Since we have an η-reduction step, $x \notin FV(M_1')$, which implies that $M_1'[N'/x] = M_1'$, and we have

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \longrightarrow_\beta M[N/x]N_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} M_1' N' N_1' \dots N_n' \xrightarrow{*}_{\lambda^\forall} P_1,$$

and

$$(\lambda x{:}\sigma.\, M)NN_1 \dots N_n \xrightarrow{*}_\beta M[N/x]N_1 \dots N_n \xrightarrow{*}_{\lambda^\forall} M_1''[N''/x]N_1'' \dots N_n'' \xrightarrow{*}_{\lambda^\forall} P_2.$$

We conclude using the confluence from $M[N/x]N_1 \dots N_n$.

Case 7. Symmetric to case 6.

Proving that confluence holds from $(\Lambda t.\, M)\tau N_1 \dots N_n$ and all its subterms assuming that confluence holds from $M[\tau/t]N_1 \dots N_n$ and all its subterms is similar to the previous proof, except that there are no reductions from τ. □

19 References

[1] Barendregt, H. *The Lambda Calculus. Its Syntax and Semantics*, Revised Edition, North-Holland, Amsterdam, 1984.

[2] Breazu-Tannen, V., and Coquand, Th., Extensional models for polymorphism, *Theor. Comput. Sci.* 59(1,2), 1988, 85-114.

[3] Breazu-Tannen, V., Combining Algebra and Higher-Order Types, In *3rd Annual Symposium on Logic In Computer Science*, IEEE Computer Society, Edinburgh, Scotland, July 1988, 82-90.

[4] Breazu-Tannen, V., and Gallier, J.H., Polymorphic Rewriting Conserves Algebraic Strong Normalization and Confluence, *16th International Colloquium on Automata, Languages and Programming*, ed. by G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, LNCS No. 372, Springer Verlag, July 1989, 137-150.

[5] Coquand, Th., Une Théorie des Constructions, Thèse de 3^{eme} Cycle, Université de Paris VII (1985).

[6] Coquand, Th., and Huet, G., The Calculus of Constructions, *Information and Computation* 76(2/3), 1988, 95-120.

[7] Coquand, Th. Mathematical investigation of a calculus of constructions, In: *Logic and Computer Science*, P. Odifreddi, editor, 1990.

[8] Fortune, S., Leivant, D., and O'Donnell, M., The expressiveness of simple and second-order type structures, *J.ACM* 30(1), 1983, 151-185.

[9] Girard, J.Y., Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types, *Proc. 2nd Scand. Log. Symp., 1970*, ed. J.E. Fenstad, North-Holland, Amsterdam, 1971, 63-92.

[10] Girard, J.Y., Interprétation functionelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse d'Etat, Université de Paris VII (1972).

[11] Girard, J.Y., Lafont, Y., and Taylor, P., *Proofs and Types*, Cambridge University Press, 1989.

[12] Girard, J.Y., *Proof Theory and Logical Complexity*, Studies in proof theory, Bibliopolis, Napoli, 1987.

[13] Harper, R., Honsell, F., and Plotkin, G., A Framework for Defining Logics, In *2nd Annual Symposium on Logic In Computer Science*, IEEE Computer Society, Ithaca, New York, June 1987, 194-204.

[14] Harper, R., Honsell, F., and Plotkin, G., A Framework for Defining Logics, draft submitted for publication, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1989.

[15] Hindley, J.R., and Seldin, J.P., *Introduction to Combinators and Lambda Calculus*, Cambridge University Press, 1986.

[16] Huet, G., *Résolution d'Equations dans les Langages d'Ordre* $1, 2, \ldots, \omega$, Thèse d'Etat, Université de Paris VII (1976).

[17] Huet, G., "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems," *J.ACM* 27:4, 1980, 797-821.

[18] Huet, G., *Deduction and Computation*, Lecture Notes, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1987.

[19] Lambek, J., and Scott, P.J., *Introduction to Higher Order Categorical Logic*, Cambridge University Press, 1986.

[20] Leivant, D., Typing and Computational Properties of Lambda Expressions, *Theoretical Computer Science* 44, 1986, 51-68.

[21] Martin-Löf, P., Hauptsatz for the Intuitionistic Theory of Iterated Inductive Definitions, *Proc. 2nd Scand. Log. Symp., 1970*, ed. J.E. Fenstad, North-Holland, Amsterdam, 1971, 179-216.

[22] Martin-Löf, P., Hauptsatz for the Theory of Species, *Proc. 2nd Scand. Log. Symp., 1970*, ed. J.E. Fenstad, North-Holland, Amsterdam, 1971, 217-233.

[23] Martin-Löf, P., Hauptsatz for Intuitionistic Simple Type Theory, In *Logic, Methodology and Philosophy of Science IV*, ed. by P. Suppes, L. Henkin, A. Joja, and Gr.C. Moisil, North-Holland, Amsterdam, 1973, 279-290.

[24] Mitchell, J.C., A type-inference approach to reduction properties and semantics of polymorphic expressions, In *Proc. 1986 ACM Symposium on Lisp and Functional Programming*, ACM, New York, August 1986, 308-319.

[25] Mitchell, J.C., Type Systems for Programming Languages, to appear in *Handbook of Theoretical Computer Science*, ed. by J. van Leeuwen, A. Meyer, and D. Perrin, North-Holland, 1990.

[26] Paulin-Morhing, C., Extraction de programmes dans le calcul des constructions, Thèse d'Etat, Université de Paris VII, 1989.

[27] Pfenning, F., Partial Polymorphic Type Inference and Higher-Order Unification, In *1988 ACM Conference on Lisp and Functional Programming*, ACM, Snowbird, Utah, July 25-27, 1988, 153-163.

[28] Pfenning, F., Elf, A language for logic definition and verified metaprogramming, In *4th Annual Symposium on Logic In Computer Science*, IEEE Computer Society, Asilomar, California, June 1989, 313-322.

[29] Prawitz, D., Ideas and Results in Proof Theory, *Proc. 2nd Scand. Log. Symp., 1970*, ed. J.E. Fenstad, North-Holland, Amsterdam, 1971, 235-307.

[30] Reynolds, J., Towards a theory of type structure. In *Proc. sur la Programmation*, LCNS No. 19, Springer Verlag, NY, 1974, 408-425.

[31] Scedrov, A, Normalization revisited, In: *Categories in Logic and Computer Science, Proceedings of a Summer Research Conference, Boulder, Colorado, June 1987*, ed. by J.W. Gray and A. Scedrov, Contemporary Mathematics, Vol. 92, Amer. Math. Soc., Providence, RI, 1989, 357-369.

[32] Scedrov, A., A guide to polymorphic types, In: *Logic and Computer Science*, P. Odifreddi, editor, 1990.

[33] Seldin, J.P., Mathesis: The Mathematical Foundations of Ulysses, Technical Report RADC-TR-87-223, Odyssey Research Associates, 1987.

[34] Statman, R., Logical Relations and the Typed λ-Calculus, *Information and Control* 65, 3-2, 1985, 85-97.

[35] Stenlund, S., *Combinators, Lambda Terms, and Proof Theory*, D. Reidel, Dordrecht, Holland, 1972.

[36] Tait, W.W., Intensional interpretation of functionals of finite type I, *J. Symbolic Logic* 32, 1967, 198-212.

[37] Tait, W.W., A realizability interpretation of the theory of species, In *Logic Colloquium*, ed. R. Parikh, LNCS No. 453, Springer Verlag, 1975, 240-251.

[38] de Vrijer, R.C., Surjective Pairing and Strong Normalization: Two Themes in Lambda Calculus, Doctoral thesis, Amsterdam, the Netherlands, 1987.

[39] de Vrijer, R.C., Strong Normalization in $N\text{-}HA^{\omega}_{p}$, Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen, Series A, Volume 90, No. 4, December 18, 1987, 473-478.

[40] de Vrijer, R.C., Exactly estimating functionals and strong normalization, Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen, Series A, Volume 90, No. 4, December 18, 1987, 479-493.