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

October 7, 2002

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 $\mathit{Type}\colon \Sigma \to \mathcal{T}$ 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 $\mathit{Erase}\colon \mathcal{P}\Lambda \to \Lambda$ is defined recursively as follows:

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

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 $\mathit{dom}(\varphi)$. If $\mathit{dom}(\varphi) = \{x_1, \ldots, x_n\}$ 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:

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

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 $M_1N_1 \equiv_\alpha M_2N_2$.

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

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:

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

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:

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

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 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 $\Delta \rhd M \colon \sigma$, 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:

$$\Delta \rhd c \colon Type(c), \qquad c \in \Sigma \qquad (constants)$$

$$\Delta \rhd x \colon \Delta(x), \qquad x \in dom(\Delta) \qquad (variables)$$

Inference Rules:

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

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

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

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

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

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

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 under*

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

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

This example suggests that α-equivalence should be extended to typing judgments as well. Indeed, $\rhd \Lambda t_1.\, \lambda x_1\colon t_1.\, x_1\colon \forall t_1.\, (t_1 \to t_1)$ and $\rhd \Lambda t_2.\, \lambda x_2\colon t_2.\, x_2\colon \forall t_3.\, (t_3 \to t_3)$ 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 $\Delta \rhd M\colon \sigma$ and $\Delta' \rhd M'\colon \sigma'$ are *α-equivalent* iff $\Delta \equiv_\alpha \Delta'$, $M \equiv_\alpha M'$, and $\sigma \equiv_\alpha \sigma'$.

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

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

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

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

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 $\Delta \rhd M\colon \sigma$ 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 $\vdash \Delta' \rhd M'\colon \sigma'$ (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. $\square$

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:

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

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

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

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

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

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

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

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

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

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 $\vdash \Delta \rhd M{:}\,\sigma$ and $M \longrightarrow_{\lambda^\forall} N$, then N also type-checks with some proof $\vdash \Delta \rhd N{:}\,\sigma$.

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

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 $\xrightarrow{*}_{\lambda}$ 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 $\xrightarrow{*}$ 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 $a_1 \xrightarrow{*} a_3$ and $a_2 \xrightarrow{*} a_3$. We say that $\longrightarrow$ is *confluent* iff for all $a, a_1, a_2 \in A$, if $a \xrightarrow{*} a_1$ and $a \xrightarrow{*} a_2$, then there is some $a_3 \in A$ such that $a_1 \xrightarrow{*} a_3$ and $a_2 \xrightarrow{*} a_3$. Let $\longleftrightarrow \;=\; \longrightarrow \cup \longrightarrow^{-1}$. We say that $\longrightarrow$ is *Church-Rosser* iff for all $a_1, a_2 \in A$, if $a_1 \overset{*}{\longleftrightarrow} a_2$, then there is some $a_3 \in A$ such that $a_1 \xrightarrow{*} a_3$ and $a_2 \xrightarrow{*} a_3$.

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 $Erase(M)$. $\square$

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 $M_1 \overset{*}{\longrightarrow}_\lambda M_2$ and $N_1 \overset{*}{\longrightarrow}_\lambda N_2$, then $M_1[N_1/x] \overset{*}{\longrightarrow}_\lambda M_2[N_2/x]$.

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. $\square$

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 $\eta\colon \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, 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 $\eta_1\colon \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$ and $\eta_2\colon \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, 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 $\eta\colon \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, 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 $\mu\colon \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, $[\![\sigma]\!]\mu \in \mathcal{C}$. 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 $\eta\colon \mathcal{B} \cup \mathcal{V} \to \mathcal{C}$, 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.

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

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.

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

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.

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

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.

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

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 $\Delta \rhd \Lambda t.\, M{:}\,\forall t.\,\sigma$. 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 $\vdash \Delta \rhd M{:}\,\sigma$, 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 $M_1 \in [C \rightarrow D]$, $M_2 \in \Lambda$ 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 $M_2 \in [C \rightarrow D]$, i.e., $[C \rightarrow D]$ is closed under ρ. It is obvious that every intersection $\bigcap_{C \in \mathcal{C}} A_C$ is closed under ρ. $\square$

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. $\square$

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 $uN_1 \ldots N_m$ 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 $(\lambda x.\, M)NN_1 \ldots N_m$, then there is an infinite reduction sequence from $M[N/x]N_1 \ldots N_m$.[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 $(\lambda x.\, M)NN_1 \ldots N_m$. There are three different possible patterns:

(1) every term in this sequence is of the form $(\lambda x.\, M')N'N_1' \ldots N_m'$, where $M \xrightarrow{*}_\lambda M'$, $N \xrightarrow{*}_\lambda N'$, and $N_i \xrightarrow{*}_\lambda N_i'$ for $i = 1, \ldots, m$, or

(2) there is a step $(\lambda x.\, M')N'N_1' \ldots N_m' \longrightarrow_\beta M'[N'/x]N_1' \ldots N_m'$ in this sequence, for some 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 $i = 1, \ldots, m$, or

(3) $M \xrightarrow{*}_\lambda M_1'x$, and there is a step

$$(\lambda x.\, (M_1'x))N'N_1' \ldots N_m' \longrightarrow_\eta M_1'N'N_1' \ldots N_m',$$

for some N', $N_1', \ldots, N_m'$, such that $N \xrightarrow{*}_\lambda N'$, and $N_i \xrightarrow{*}_\lambda N_i'$, for $i = 1, \ldots, m$.

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, $N_1, \ldots, N_m$. 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 $M \xrightarrow{*}_\lambda M'_1x$, 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, $M'_1[N/x]N'N'_1 \ldots N'_m = M'_1N'N'_1 \ldots N'_m$. Since there is an infinite reduction from $M'_1N'N'_1 \ldots N'_m$, 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. $\square$

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.