A Proof of Strong Normalization for the Theory of Constructions
Using a Kripke-Like Interpretation

Thierry Coquand
INRIA

Jean H. Gallier
University of Pennsylvania, jean@cis.upenn.edu
A Proof of Strong Normalization for the Theory of Constructions Using a Kripke-Like Interpretation

Abstract
We give a proof that all terms that type-check in the theory of contructions are strongly normalizing (under β-reduction). The main novelty of this proof is that it uses a "Kripke-like" interpretation of the types and kinds, and that it does not use infinite contexts. We explore some consequences of strong normalization, consistency and decidability of typechecking. We also show that our proof yields another proof of strong normalization for LF (under β-reduction), using the reducibility method.

Comments
A Proof Of Strong Normalization
For The Theory Of Constructions
Using A Kripke-Like Interpretation

MS-CIS-90-44
LOGIC & COMPUTATION 21

Thierry Coquand
Jean Gallier

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

July 1990
A Proof of Strong Normalization For the Theory of Constructions Using a Kripke-Like Interpretation

Thierry Coquand         Jean Gallier

INRIA
Domaine de Voluceau, Rocquencourt
B.P. 105
78153 Le Chesnay Cedex, France

Department of Computer and Information Science
University of Pennsylvania
200 South 33rd St.
Philadelphia, PA 19104, USA

Abstract. We give a proof that all terms that type-check in the theory of contructions are strongly normalizing (under $\beta$-reduction). The main novelty of this proof is that it uses a "Kripke-like" interpretation of the types and kinds, and that it does not use infinite contexts. We explore some consequences of strong normalization, consistency and decidability of type-checking. We also show that our proof yields another proof of strong normalization for $LF$ (under $\beta$-reduction), using the reducibility method.

1 Introduction

We give a proof that all terms that type-check in the theory of constructions are strongly normalizing (under $\beta$-reduction). The main novelty of this proof is that it uses a "Kripke-like" interpretation of the types and kinds, and that it does not use infinite contexts. The idea used for avoiding infinite contexts comes from Coquand’s thesis [Coq85]. Our proof yields as a corollary another proof of strong normalization (under $\beta$-reduction) of well-formed terms of $LF$. In fact, it is easy to see that this proof does not use the candidates of reducibility at all. We are unaware of similar proofs (using reducibility “à la Tait”) for $LF$.

$^1$The results in this paper were first presented at the First Annual Workshop on Logical Frameworks, Esprit Basic Research Action, Antibes, May 7-11, 1990.

$^2$Partially supported by ONR Grant N00014-88-K-0593.
Our experience with proofs of strong normalization is that besides their intrinsic difficulty, their clarity and ease of understanding are greatly affected by the choice of notation, and the order in which the concepts are introduced. For example, it is logical to define “the values” before defining $[\Gamma \triangleright A] \rho \Delta$, the interpretation of types, since this latter definition requires the former (the term “values” is used by Coquand [Coq87], but in Girard’s terminology and ours, these are the sets $C_{A,\Delta}$ of “candidates of reducibility”). However, we believe that it more intuitive and easier for understanding such proofs, if $[\Gamma \triangleright A] \rho \Delta$ is defined before the families of candidates (In Coquand [Coq87], $[\Gamma \triangleright A] \rho \Delta$ is called the interpretation of a term, and it is denoted as $Eval \rho M$). It is possible to do so by first giving a rough and intuitive idea of what families of candidates are. Another difficulty is to package in a convenient way the various ingredients making up a candidate assignment (the substitution component, the candidate assignment component, etc). This is one place where the idea of viewing a context $\Delta$ as a world, as in Kripke semantics of intuitionistic logic, seems helpful.

A key remark for our presentation is the following: proofs of normalization that follow the reducibility method are intuitionistic. Hence, it should be possible to carry them in any intuitionistic model, hence in any Kripke model. There is furthermore one natural Kripke “term model” where we take for the Kripke worlds, the valid contexts of the type system. It should be noted that this paper represents “ongoing research”, and that this is a preliminary version of a paper in which we intend to explore more thoroughly the nature of Kripke models for the theory of constructions.

Among the sources of inspiration for this research, Moggi and Mitchell’s work on Kripke models for the simply-typed $\lambda$-calculus [MM87] should be mentioned. We also became recently aware of work by Aarne Ranta [Ran90], in which the notion of “contexts as (finite approximation of) worlds” is used. One of the motivations for this work is to give an intuitionistic treatment of the notion of “possible worlds”. Ranta’s notion of Kripke structure is more general than ours, in that he does consider any interpretation between contexts, and not only projection (here seen dually as inclusion). It may be interesting to see if one can formulate a normalization proof in this framework.

One should be careful in referring to “the” theory of constructions, since different versions of this theory have been formulated and these versions are not all equivalent. Thus, in order to avoid ambiguities, we formulate in the next section a version of the theory of constructions equivalent (but not identical in syntax) to the version presented by Coquand and Huet [CH88]. We refer to this version of the theory of constructions as $CC$.

## 2 Syntax of the Theory of Constructions $CC$

We find it pedagogically convenient to first describe a theory of constructions whose syntax has three levels ($kinds$, $type families$, and $terms$). The special kind $\star$ is the logical kind of propositions. In other words, types (propositions) are exactly those type families whose kind
is \(*\). In the simple theory of type, Church used the notation \(o\) for \(*\). Another notation used in place of \(*\) is \(Prop\) (or even \(Type\)). Furthermore, some authors use \(Type\) instead of \textit{kind}, but we find this practice somewhat confusing, since in the Curry-Howard formula-as-type analogy, propositions correspond to types.

We begin by defining \textit{raw terms}.

**Definition 2.1** We use the nonterminal \(K\) to range over kinds, \(A\) to range over type families, and \(M\) to range over terms. We also use two kinds of variables, ranging over kinds and type families. \textit{Raw terms} are defined by the following grammar.

\[
\begin{align*}
K \rightarrow & \, * \mid (\Pi t: K)K \mid (\Pi x: A)K \\
A \rightarrow & \, t \mid (\forall t: K)A \mid (\forall x: A)A \mid (AA) \mid (AM) \mid (\Lt t: K. A) \mid (\lx: A. A) \\
M \rightarrow & \, x \mid (MM) \mid (MA) \mid (\Lt t: K. M) \mid (\lx: A. M).
\end{align*}
\]

A \textit{context} is an ordered sequence of pairs \(\Delta = \{(x_1, A_1), \ldots, (x_m, A_m)\}\), where \(x_i\) is a variable and \(A_i\) is a kind or a type family, and for any two \(x_i, A_i\) and \(x_j, A_j\) in \(\Delta\), \(i \neq j\) implies that \(x_i \neq x_j\). A context \(\Delta\) is usually written as \(x_1: A_1, \ldots, x_m: A_m\). There are four categories of \textit{judgments}:

**Definition 2.2** \textit{Judgments} are expressions of the form:

\[
\begin{align*}
\Delta \vdash (\Delta \text{ is a valid context}), \\
\Delta \vdash K: \text{kind} (K \text{ is a valid kind in context } \Delta), \\
\Delta \vdash A: K (A \text{ kind-checks with kind } K \text{ in context } \Delta), \\
\Delta \vdash M: A (M \text{ type-checks with type } A \text{ in context } \Delta).
\end{align*}
\]

We define \(\beta\)-reduction and \(\beta\)-conversion in the usual manner on raw terms. This means that redexes will be of the form \((\Lt t: K. A)B, (\lx: A. B)M, (\Lt t: K. M)B, \text{ and } (\lx: A. M)N\). We emphasize that we do \textit{not} consider \(\eta\)-conversion in this paper. There appears to be some difficulties with the Church-Rosser theorem if \(\beta\eta\)-conversion is defined on raw terms, and it seems that one needs to define judgments of the form \(\Delta \vdash M \leftrightarrow_{CC} M'\) (equality judgments), which is quite cumbersome.

3 \textbf{Typing Rules for \(CC\)}

We could list the typing rules assuming the above syntax, but it is possible to state them more concisely if certain conventions are adopted.
• Firstly, we will not distinguish between type variables and term variables.

• Secondly, we will use the symbol $\kappa$ to denote either $kind$ or $\ast$.

• Thirdly, we will denote both judgments $\Delta \vdash K: kind$ and $\Delta \vdash \sigma: \ast$ as $\Delta \vdash A: \kappa$, and similarly, we will denote both judgments $\Delta \vdash \sigma: K$ and $\Delta \vdash M: \sigma$ as $\Delta \vdash M: A$.

• Finally, we identify $\forall$ and $\Pi$, and $\Lambda$ and $\lambda$.

Note that now, there is only one kind of raw terms given by the following grammar:

$$M \rightarrow x \mid \ast \mid (\Pi x: M)M \mid (MM) \mid (\lambda x: M. M).$$

With the above conventions, we only have one rule for each kind of rule.

**Definition 3.1** In the rules below, $\kappa, \kappa_1, \kappa_2 \in \{\ast, kind\}$.

**Context Formation:**

$$\begin{align*}
\emptyset \vdash & \\
\Delta \vdash & \\
\Delta \vdash \ast: kind & \\
\Delta \vdash A: \kappa & \\
\Delta, x: A \vdash & \\
\end{align*} \quad \text{empty context}$$

**Axiomatic Judgments:**

$$\begin{align*}
\Delta \vdash & \\
\Delta \vdash x: A & \quad x : A \in \Delta
\end{align*}$$

**Product Formation and Quantification:**

$$\begin{align*}
\Delta \vdash A_1: \kappa_1 & \\
\Delta, x: A_1 \vdash A_2: \kappa_2 & \\
\Delta \vdash (\Pi x: A_1)A_2: \kappa_2 & \\
\end{align*}$$

**Abstraction:**

$$\begin{align*}
\Delta \vdash A_1: \kappa_1 & \\
\Delta, x: A_1 \vdash A_2: \kappa_2 & \\
\Delta, x: A_1 \vdash M: A_2 & \\
\Delta \vdash (\lambda x: A_1, M): (\Pi x: A_1)A_2 & \\
\end{align*}$$

**Application:**

$$\begin{align*}
\Delta \vdash M: (\Pi x: A_1)A_2 & \\
\Delta \vdash N: A_1 & \\
\Delta \vdash MN: A_2[N/x] & \\
\end{align*}$$

**Kind and Type Conversion:**

$$\begin{align*}
\Delta \vdash M: A_1 & \\
\Delta \vdash A_2: \kappa & \\
\Delta \vdash M: A_2 & \\
\end{align*} \quad A_1 \leftrightarrow_{CC} A_2$$
It turns out that the above typing rules can be simplified, because some of the premises are redundant. Of course, this has to be justified carefully, but this has been verified by Coquand and Huet [CH88], and others. For the reader’s convenience, we recall some of the main basic properties of CC.

4 Some Basic Properties of CC

We shall use the notation $\Delta \triangleright E$ as an abbreviation for all forms of judgments. Given contexts $\Gamma$ and $\Delta$, the notation $\Gamma \leq \Delta$ means that $\Gamma$ is an initial subsequence of $\Delta$.

First, we note that under $\beta$-conversion alone, the Church-Rosser theorem holds even for raw terms.

**Theorem 4.1** (*Martin Löf*)
The Church-Rosser property holds for raw terms of CC (even the economical version).

**Proof.** Such a proof using the so called “Tait/Martin Löf’s method” was given by Martin Löf [ML72]. □

It should be noted that theorem 4.1 is quite handy. It appears that if $\beta$-conversion is defined on raw terms, which is definitely more convenient than using equality judgments, many important properties of CC make use of the Church-Rosser property.

The propositions listed below consist of the translation in English and in our terminology of properties 1-7 in Chapter 1 of Coquand’s thesis [Coq85]. In some cases, these proofs require some amplification. First, we need the following definitions, which are translations in our terminology of Coquand’s definitions.

**Definition 4.2** $K$ is a kind iff $K$ is of the form $*$ or $\left(\Pi x_1: A_1 \right) \ldots \left(\Pi x_m: A_m \right) *$, and $\Delta \triangleright K: kind$ for some context $\Delta$;

$A$ is a type family iff $\Delta \triangleright A: K$ for some context $\Delta$ and some kind $K$;

$A$ is a type iff $\Delta \triangleright A: *$ for some context $\Delta$;

$M$ is a proof (or proof term) iff $\Delta \triangleright M: A$ where $A$ is not a kind.

When we want to stress that a context $\Delta$ is well-formed, that is, when $\Delta \triangleright$ is derivable, we say that $\Delta$ is a valid context, and similarly for kinds, type families, types, and proofs.

**Lemma 4.3** If $\Delta \triangleright E$, then $\Delta' \triangleright$ for every $\Delta' \leq \Delta$, and more generally, every derivation of $\Delta \triangleright E$ contains a derivation of $\Delta' \triangleright$ as a subderivation.
Lemma 4.4 If $\Delta \triangleright A: K$ and $\Delta \triangleright A: K'$ where both $K$ and $K'$ are kinds, then $K \leftrightarrow_{CC} K'$. Similarly, if $\Delta \triangleright M: A$ and $\Delta \triangleright M: A'$, where both $A$ and $A'$ are not kinds, then $A \leftrightarrow_{CC} A'$.

The proof of the above lemma actually seems to require the Church-Rosser property and the following proposition.

Proposition 4.5 Assume that $\Delta \triangleright (\Pi x: A)K: \text{kind}$ and $\Delta \triangleright (\Pi x: A')K': \text{kind}$. If $(\Pi x: A)K \leftrightarrow_{CC} (\Pi x: A')K'$, then $A \leftrightarrow_{CC} A'$ and $K \leftrightarrow_{CC} K'$.

Proposition 4.6 Both $\Delta \triangleright M: A$ and $\Delta \triangleright M: \text{kind}$ are not derivable at the same time.

Definition 4.7 Given any two contexts $\Delta$, $\Delta'$, we say that $\Delta \subseteq \Delta'$ iff for every $x$, if $x \in \text{dom}(\Delta)$ then $x \in \text{dom}(\Delta')$ and $\Delta(x) = \Delta'(x)$.

Lemma 4.8 Assume that $\Delta \triangleright$, $\Delta' \triangleright$, and $\Delta \subseteq \Delta'$. If $\Delta \triangleright E$, then $\Delta' \triangleright E$. In particular, if $\Delta \leq \Delta'$, $\Delta' \triangleright$, and $\Delta \triangleright E$, then $\Delta' \triangleright E$.

Lemma 4.9 If $\Delta \triangleright M: A$ and $\Delta, x: A, \Delta' \triangleright E$, then $\Delta, \Delta'[M/x] \triangleright E[M/x]$.

Lemma 4.10 If $\Delta \triangleright M: A$ and $A$ is not a kind, then $\Delta \triangleright A: \ast$. If $\Delta \triangleright M: A$ and $A$ is a kind, then $\Delta \triangleright A: \text{kind}$.

Lemma 4.11 If $\Delta, x: A, \Delta' \triangleright E$, $A \leftrightarrow_{CC} A'$, and either $\Delta \triangleright A': \ast$ or $\Delta \triangleright A': \text{kind}$, then $\Delta, x: A', \Delta' \triangleright E$.

Lemma 4.12 If $\Delta \triangleright M: A$ and $M \leftrightarrow_{CC} N$, then $\Delta \triangleright N: A$.

Lemma 4.13 The judgments $\Delta \triangleright M: A$ where $A$ is a type and $\Delta \triangleright M: K$ where $K$ is a kind cannot hold simultaneously.

The proof of the above lemma seems to require the Church-Rosser property.

In view of proposition 4.6, kinds are disjoint from type families and proofs. In view of lemma 4.13, proofs and type families are disjoint.
5 Strong Normalization in CC

The proof of strong normalization for well-typed terms of CC is obtained by generalizing the proof given by Girard for the system $F_\omega$ [Gir72], as presented in Gallier [Gal90]. However, there are some significant technical complications. In $F_\omega$, we have an ascending hierarchy, kinds, type families, and terms, where kinds do not depend on type families or terms, and type families do not depend on terms. However, in CC, kinds, type families, and terms, are defined in a single big simultaneous inductive construction. The main difficulty is to ensure that the interpretations $[\Gamma \vdash A]_\rho \Delta$ are nondegenerate (i.e., nonempty sets).

The first step is to define the concept of a candidate assignment, which packages together a substitution and a valuation assigning candidates to variables.

**Definition 5.1** A substitution is a function $\varphi: V \rightarrow \text{Terms}$ such that $\varphi(x) \neq x$ for only finitely many $x$, and for every $\varphi(x)$, there is some context $\Delta$ and either some type $A$ such that $\Delta \vdash \varphi(x): A$ (and $\Delta \vdash A: \ast$), or some kind $K$ such that $\Delta \vdash K: \text{kind}$. The domain $D(\varphi)$ of $\varphi$ is the set $D(\varphi) = \{x \mid \varphi(x) \neq x\}$.

Every substitution $\varphi$ has a unique homomorphic extension $\hat{\varphi}: \text{Terms} \rightarrow \text{Terms}$. Given a term $M$ (term, type family, or kind), the result of applying $\varphi$ to $M$ is $\hat{\varphi}(M)$, and it is denoted as $\varphi(M)$ or $M[\varphi]$.

Some form of Kripke structure is lurking around. Contexts are going to play the role of worlds. Consequently, most concepts will be defined “in world $\Delta$”. The notion of inclusion of worlds is the relation $\subseteq$ defined in definition 4.7. Substitutions will also play the role of valuations assigning values to variables. This motivates the following definition.

**Definition 5.2** Given two valid contexts $\Gamma, \Delta$, where $\Gamma$ is used to type/kind check, and $\Delta$ acts as a world, given a substitution $\varphi$, we say that $\Gamma[\varphi]$ type-checks in $\Delta$ iff $\Delta \vdash x[\varphi]: \Gamma(x)[\varphi]$ for every $x \in \text{dom}(\Gamma)$.

At first glance, one may be concerned that this condition is circular. However, this is not so. Indeed, if $\Gamma = x_1: A_1, \ldots, x_m: A_m$ is a valid context, it can be easily shown that $FV(A_i) \subseteq \{x_1, \ldots, x_{i-1}\}$ for all $i, 1 \leq i \leq n$, and that $\Gamma[\varphi]$ type-checks in $\Delta$ means that $\Delta \vdash x_i[\varphi]: A_i[x_1[\varphi]/x_1, \ldots, x_{i-1}[\varphi]/x_{i-1}]$ for all $i, 1 \leq i \leq n$, which is possible.

We now assume that for every world $\Delta$ and type family $A$ that kind-checks in $\Delta$, we have a set $C_{A, \Delta}$ of nonempty sets called candidates to be defined soon. All we need to know is that, when $A$ is a type, every $C \in C_{A, \Delta}$ is a set of terms $\Delta' \vdash M$ such that $\Delta' \vdash M: A$ for some $\Delta' \supseteq \Delta$, and when $A$ kind-checks with kind $(\Pi x: B)K$, every element of $C_{A, \Delta}$ is a certain function. We also have a set $C_{A, \Delta}$ consisting of nonempty sets of types $\Delta' \vdash A$ such that $\Delta' \vdash A: \ast$ for some $\Delta' \supseteq \Delta$, and a set $C_{\text{kind}, \Delta}$ consisting of nonempty sets of kinds $\Delta' \vdash K$ such that $\Delta' \vdash K: \text{kind}$ for some $\Delta' \supseteq \Delta$. 


Definition 5.3 A candidate assignment is any function \( \rho \) from \( V \cup \{ \ast, \text{kind} \} \) to \( \text{Terms} \cup \langle \text{Terms} \times \ast \rangle \), such that the following properties hold:

1. If we define the function \( \rho_\ast : V \to \text{Terms} \) such that,

\[
\rho_\ast(x) = \begin{cases} 
A & \text{if } \rho(x) = \langle A, C \rangle, \\
A & \text{if } \rho(x) = A,
\end{cases}
\]

then \( \rho_\ast \) is a substitution (which means that \( \rho_\ast(x) \neq x \) only for finitely many \( x \in V \)), and,

2. If \( \rho(x) = \langle A, C \rangle \), then \( A \) is a type-family that kind-checks in some context \( \Delta \) and \( C \in \mathcal{C}_{A,\Delta} \), else if \( \rho(x) = A \) then \( A \) is a term (proof) that type-checks in some context \( \Delta \);

3. \( \rho(\ast) = \langle \ast, C \rangle \), \( C \in \mathcal{C}_{\ast,\Delta} \), \( \rho(\text{kind}) = \langle \text{kind}, C \rangle \), and \( C \in \mathcal{C}_{\text{kind},\Delta} \).

The function \( \rho \) also defines another function \( \rho_c \) such that \( x \mapsto C, \ast \mapsto C \), and \( \text{kind} \mapsto C \). By abuse of notation, both \( \rho_\ast \) and \( \rho_c \) are often denoted as \( \rho \), when the context makes it clear which is referred to.

Definition 5.4 A candidate assignment \( \rho \) satisfies \( \Gamma \) at \( \Delta \) iff

1. \( \Gamma[\rho_\ast] \) type-checks in \( \Delta \).

2. Whenever \( \rho(x) = \langle A, C \rangle \) or \( \rho(x) = A \), then \( A \) kind/type-checks in \( \Delta \), and \( C \in \mathcal{C}_{A,\Delta} \).

It is easy to verify that if \( \Delta \subseteq \Delta' \) and \( \rho \) satisfies \( \Gamma \) at \( \Delta \), then \( \rho \) satisfies \( \Gamma \) at \( \Delta' \). We can now define \( [\Gamma \vdash A] \rho_{\Delta} \), where \( \Gamma \) is a context, either \( A \) is a type family that kind-checks in \( \Gamma \), or \( A \) is a kind valid in \( \Gamma \), or \( A = \text{kind} \), \( \rho \) is a candidate assignment, and \( \Delta \) is a context viewed as a world. The definition is by induction on the complexity of \( \Gamma \vdash A \) (if \( \Gamma = x_1 : A_1, \ldots, x_m : A_m \), then the complexity of \( \Gamma \vdash A \) is the sum of the sizes of \( A_1, \ldots, A_m, A \)). It only makes sense when \( \rho \) satisfies \( \Gamma \) at \( \Delta \).

Definition 5.5 In the clauses below, \( K \) stands for a kind, \( \sigma \) for a type, \( A, B \) for type families, \( D \) for a kind or a type, \( M \) for a type family or a term (proof), and \( N \) for a term (proof).

\[
\begin{align*}
[\Gamma \vdash \text{kind}] \rho_{\Delta} &= \rho_c(\text{kind}), \\
[\Gamma \vdash \ast] \rho_{\Delta} &= \rho_c(\ast), \\
[\Gamma \vdash x] \rho_{\Delta} &= \rho_c(x), \\
[\Gamma \vdash AB] \rho_{\Delta} &= [\Gamma \vdash A] \rho_{\Delta}(\Delta \triangleright B[\rho_\ast]), \quad [\Gamma \vdash B] \rho_{\Delta}, \\
[\Gamma \vdash AN] \rho_{\Delta} &= [\Gamma \vdash A] \rho_{\Delta}(\Delta \triangleright N[\rho_\ast]),
\end{align*}
\]
\[ [\Gamma \vdash (\Pi x: K)D]_{\rho} \Delta = \{ \Delta' \vdash M \mid \Delta' \vdash M : ((\Pi x: K)D)[\rho]_s, \Delta' \supseteq \Delta, \text{ and} \]
\[ \forall \Delta'' \supseteq \Delta', \forall \Delta'' \vdash A \in [\Gamma \vdash K]_{\rho} \Delta'', \forall C \in C_{A, \Delta''}, \]
\[ \Delta'' \vdash (MA) \in [\Gamma, x : K \vdash D]_{\rho}[x := (A, C)]_{\Delta''}, \]
\[ [\Gamma \vdash (\Pi x: \sigma)D]_{\rho} \Delta = \{ \Delta' \vdash M \mid \Delta' \vdash M : ((\Pi x: \sigma)D)[\rho]_s, \Delta' \supseteq \Delta, \text{ and} \]
\[ \forall \Delta'' \supseteq \Delta', \forall \Delta'' \vdash N \in [\Gamma \vdash \sigma]_{\rho} \Delta'', \]
\[ \Delta'' \vdash (MN) \in [\Gamma, x : \sigma \vdash D]_{\rho}[x := N]_{\Delta''}, \]
\[ [\Gamma \vdash \lambda x: K. B]_{\rho} \Delta = \lambda(\Delta' \vdash A) \lambda C. [\Gamma, x : K \vdash B]_{\rho}[x := (A, C)]_{\Delta'}, \]
\[ \text{a function with domain} \]
\[ \{ (\Delta' \vdash A, C) \mid \Delta' \vdash A : K[\rho]_s, \Delta' \supseteq \Delta, C \in C_{A, \Delta'} \}, \]
\[ [\Gamma \vdash \lambda x: \sigma. B]_{\rho} \Delta = \lambda(\Delta' \vdash N). [\Gamma, x : \sigma \vdash B]_{\rho}[x := N]_{\Delta'}, \]
\[ \text{a function with domain} \]
\[ \{ \Delta' \vdash N \mid \Delta' \vdash N : \sigma[\rho]_s, \Delta' \supseteq \Delta \}. \]

We emphasize again the fact that in \([\Gamma \vdash x]_{\rho} \Delta\), we have \(\Gamma \vdash x : K\) for some kind \(K\), i.e., \(x\) is a type variable.

**Definition 5.6** Given any judgment \(\Gamma \vdash M : A\) (where \(A\) can even be *kind*), given any candidate assignment \(\rho\), and any context \(\Delta\) viewed as a world, we write \(\Delta \vDash \Gamma[\rho]\) iff

(1a) \(\rho\) satisfies \(\Gamma\) at \(\Delta\), and

(2a) \(\Delta \vdash x[\rho] \in [\Gamma \vdash \Gamma(x)]_{\rho} \Delta\) for every \(x \in \text{dom}(\Gamma)\).

We will also write \(\Delta \vDash (M : A)[\rho]\) iff

(1b) \(\rho\) satisfies \(\Gamma\) at \(\Delta\), and

(2b) \(\Delta \vdash M[\rho] \in [\Gamma \vdash A]_{\rho} \Delta\).

Then, the main theorem reads as follows: Whenever \(\Gamma \vdash M : A\) and \(\Delta \vDash \Gamma[\rho]\), then \(\Delta \vDash (M : A)[\rho]\). This looks like a Kripke-style type soundness result.

Actually, it is not obvious that the inductive definition of \([\Gamma \vdash A]_{\rho} \Delta\) defines nonempty sets and total functions, and this depends on some properties of the sets \(C_{A, \Delta}\). One of the crucial facts is that for every valid context \(\Delta\) and type or kind \(A\), there is some term or type family \(\Delta' \vdash M\) with \(\Delta' \supseteq \Delta\) such that \(\Delta' \vdash M : A\). Indeed \(\Delta' = \Delta, x : A\) where \(x \notin \text{dom}(\Delta)\) does the job, since \(\Delta, x : A \vdash x : A\) is derivable.

We can now define the sets \(C_{A, \Delta}\). For this this, we need a complexity measure for types and kinds.
Definition 5.7 Let \( A \) be any valid type, and \( K \) any valid kind. We define \( c(A) \) and \( c(K) \) inductively as follows:

\[
c(A) = 0, \quad c(K) = \begin{cases} 1 & \text{if } K = *, \\ \max(c(B), c(D)) + 1 & \text{if } K = (\Pi x : B)D. \end{cases}
\]

It is easily verified that if \( K \overset{*}{\leftrightarrow}_{CC} K' \), then \( c(K) = c(K') \). The main property of this complexity measure is that it is invariant under substitution.

Lemma 5.8 For every type family or term \( M \), for every kind \( K \), \( c(K[M/x]) = c(K) \).

Proof. We proceed by induction on the structure of \( K \). If \( K = * \), the lemma holds since \( *[M/x] = * \). If \( K = (\Pi x : B)D \), there are two cases. If \( B \) is also a kind, by the induction hypothesis, \( c(B[M/x]) = c(B) \), \( c(D[M/x]) = c(D) \), and the lemma holds since \( K[M/x] = (\Pi x : B[M/x])D[M/x] \). If \( A \) is a type, then \( A[M/x] \) is also a type, and since \( c(A[M/x]) = 0 \) and by the induction hypothesis \( c(D[M/x]) = c(D) \), the lemma holds. \( \square \)

We also let \( c(\text{kind}) = 0 \). The sets \( \mathcal{C}_{A,\Delta} \) are defined by induction on \( c(K) \), where \( \Delta \triangleright A : K \). Since \( c(K) \) only depends on the equivalence class of \( K \) modulo \( \beta \)-conversion, this definition is proper. The definition of the sets \( \mathcal{C}_{A,\Delta} \) given next is a bit more general than really required for proving strong normalization. The reason for giving it in this form is that it can be used to extend our proof to other properties besides strong normalization. This definition also contains all the closure conditions that will come up during the proof of the main result.

Definition 5.9 The family \( \mathcal{C} \) of sets \( \mathcal{C}_{A,\Delta} \) where \( A \) is a kind or a type family valid in the context \( \Delta \), is defined by the properties listed below. It is called the family of saturated sets.

1. \( \mathcal{C}_{\text{kind},\Delta} \) is the set of sets \( \mathcal{C} \), such that, each \( \mathcal{C} \) is a nonempty set of strongly normalizing kinds \( \Delta' \triangleright K \), with \( \Delta' \supseteq \Delta \), and the following properties hold:
   
   (a) \( \Delta' \triangleright * \in \mathcal{C} \) for all \( \Delta' \supseteq \Delta \).
   
   (b) For every kind \( \Delta' \triangleright (\Pi x : K)D \), with \( \Delta' \supseteq \Delta \) and \( K \) a kind, if \( \Delta' \triangleright K \in \mathcal{C} \) and \( \Delta' \triangleright D \in \mathcal{C} \), then \( \Delta' \triangleright (\Pi x : K)D \in \mathcal{C} \).
   
   (c) For every kind \( \Delta' \triangleright (\Pi x : \sigma)D \), with \( \Delta' \supseteq \Delta \) and \( \sigma \) a type, for every \( C' \in \mathcal{C}_{*,\Delta} \), if \( \Delta' \triangleright \sigma \in C' \) and \( \Delta' \triangleright D \in \mathcal{C} \), then \( \Delta' \triangleright (\Pi x : \sigma)D \in \mathcal{C} \).
   
   (d) Whenever \( \Delta' \triangleright K \in \mathcal{C} \) and \( \Delta' \subseteq \Delta'' \), then \( \Delta'' \triangleright K \in \mathcal{C} \).

2. \( \mathcal{C}_{*,\Delta} \) is the set of sets \( \mathcal{C} \), such that, each \( \mathcal{C} \) is a nonempty set of strongly normalizing types \( \Delta' \triangleright A \), with \( \Delta' \supseteq \Delta \), and the following properties hold:
(S0) For every type $\Delta' \triangleright (\pi x: K)A$, with $\Delta' \supseteq \Delta$ and $K$ a kind, for every $C' \in C_{\text{kind}, \Delta}$, if $\Delta' \triangleright K \in C'$ and $\Delta' \triangleright A \in C$, then $\Delta' \triangleright (\pi x: K)A \in C$, and

For every type $\Delta' \triangleright (\pi x: \sigma)A$, with $\Delta' \supseteq \Delta$ and $\sigma$ a type, if $\Delta' \triangleright \sigma \in C$ and $\Delta' \triangleright A \in C$, then $\Delta' \triangleright (\pi x: \sigma)A \in C$.

(S1) For every variable $x$, if $\Delta' \triangleright xN_1 \ldots N_m : \star$ for some $\Delta' \supseteq \Delta$ and $N_1, \ldots, N_m$ are SN, then $\Delta' \triangleright xN_1 \ldots N_m \in C$.

(S2) Whenever $\Delta' \triangleright M[N/x]N_1 \ldots N_m : \star$ and $\Delta' \triangleright N : B$ is SN for some $\Delta' \supseteq \Delta$, if $\Delta' \triangleright M[N/x]N_1 \ldots N_m \in C$, then $\Delta' \triangleright (\lambda x: B. M)NN_1 \ldots N_m \in C$.

(S3) Whenever $\Delta' \triangleright A \in C$ and $\Delta' \subseteq \Delta''$, then $\Delta'' \triangleright A \in C$.

3. When $A$ is a type family such that $\Delta \triangleright A: (\pi x: B)D$ (and $\Delta \triangleright (\pi x: B)D: \text{kind}$), $C_{A, \Delta}$ is the set of functions with the following properties:

(a) If $B$ is a kind, then

- $f \in C_{A, \Delta}$ is a function with domain

  \[
  \{(\Delta' \triangleright M, C) \mid \Delta' \triangleright M : B, \ \Delta' \supseteq \Delta, \ C \in C_{M, \Delta'}\}
  \]

  such that $f(\Delta' \triangleright M, C) \in C_{AM, \Delta'}$, and

- $f(\Delta' \triangleright M_1, C) = f(\Delta' \triangleright M_2, C)$ whenever $M_1 \leftrightarrow_{CC} M_2$.

(b) If $B$ is a type, then

- $f \in C_{A, \Delta}$ is a function with domain $\{\Delta' \triangleright N \mid \Delta' \triangleright N : B, \ \Delta' \supseteq \Delta\}$ such that $f(\Delta' \triangleright N) \in C_{AN, \Delta'}$, and

- $f(\Delta' \triangleright N_1) = f(\Delta' \triangleright N_2)$ whenever $N_1 \leftrightarrow_{CC} N_2$. 

11
Note that this definition is proper, because we can prove that the sets $C_{M,\Delta'}, C_{AM,\Delta'}$, and $C_{AN,\Delta}$, needed in (4) are well defined, where $\Delta \vdash A : (\Pi x : B)D$, $\Delta' \vdash M : B$, and $\Delta' \vdash N : B$ with $\Delta' \supseteq \Delta$. This is correct, since $\Delta' \vdash AM : D[M/x]$, $\Delta' \vdash AN : D[N/x]$, $c(B) < c((\Pi x : B)D)$, and by lemma 5.8, $c(D[M/x]) = c(D) < c((\Pi x : B)D)$, and $c(D[N/x]) = c(D) < c((\Pi x : B)D)$.

One can also easily prove that if $A \xrightarrow{\ast} CC A'$, then $C_{A,\Delta} = C_{A',\Delta}$.

Given a type family $A$ such that $\Delta \vdash A : K$, we can prove by induction on $c(K)$ that each $C_{A,\Delta}$ is nonempty.

**Lemma 5.10** Whenever $A$ kind-checks in $\Delta$, $C_{A,\Delta}$ is nonempty.

**Proof.** We define an element $can_{A,\Delta}$ of $C_{A,\Delta}$ where $\Delta \vdash A : K$ such that $A \xrightarrow{\ast} CC A'$ implies that $can_{A,\Delta} = can_{A',\Delta}$, by induction on $c(K)$. We call $can_{A,\Delta}$ the canonical member of $C_{A,\Delta}$.

When $A = kind$, note that the set $can_{kind,\Delta}$ of strongly normalizing kinds of the form $\Delta' \vdash K$ for some $\Delta' \supseteq \Delta$ is nonempty, since $\Delta' \vdash \star : kind$ for every $\Delta'$, and it is obvious that (b), (c), and (d), are also satisfied.

When $A = \star$, note that the set $can_{*,\Delta}$ of strongly normalizing types of the form $\Delta' \vdash N$ for some $\Delta' \supseteq \Delta$ is nonempty, since $\Delta, x : \star \vdash x : \star$ for $x \notin dom(\Delta)$. Properties (S0), (S1), (S2), and (S3), are also easily verified.

When $A$ is a type, note that the set $can_{A,\Delta}$ of strongly normalizing terms of the form $\Delta' \vdash N$ such that $\Delta' \supseteq \Delta$ is nonempty, since $\Delta, x : A \vdash x : A$ for $x \notin dom(\Delta)$. Properties (S1), (S2), and (S3), are also easily verified. That $A \xrightarrow{\ast} CC A'$ implies $can_{A,\Delta} = can_{A',\Delta}$ follows from the fact that $\Delta' \vdash N : A$ and $A \xrightarrow{\ast} CC A'$ implies that $\Delta' \vdash N : A'$.

When $\Delta \vdash A : (\Pi x : B)D$, we define the function $can_{A,\Delta}$ as follows. By the induction hypothesis, for every $M$ such that $\Delta' \vdash M : B$ for some $\Delta' \supseteq \Delta$, $can_{M,\Delta'}$ is defined. We define $can_{A,\Delta}$ such that $can_{A,\Delta}(\Delta' \vdash M, C) = can_{AM,\Delta'}$, and $can_{A,\Delta}(\Delta' \vdash M) = can_{AM,\Delta'}$ if $B$ is a type. If $A \xrightarrow{\ast} CC A'$, then $AM \xrightarrow{\ast} CC A'M$, and this implies $can_{AM,\Delta'} = can_{A'M,\Delta'}$ by the induction hypothesis.

**Remark:** It will be observed later that for proving strong normalization, we can simply define $C_{kind,\Delta}$ and $C_{*,\Delta}$ as the singleton families $C_{kind,\Delta} = \{can_{kind,\Delta}\}$ and $C_{*,\Delta} = \{can_{*,\Delta}\}$.

In order to show that the closure properties of the family $C$ insure that the sets $[\Gamma \vdash A]_{\rho,\Delta}$ are also in $C$, we need the following technical lemma.

**Lemma 5.11** If $C$ is the family of saturated sets, for any two $\rho$ and $\rho'$ satisfying $\Gamma$ at $\Delta$, if $x[\rho] \xrightarrow{\ast} CC x[\rho']$ for every $x \in dom(\Gamma)$, then $[\Gamma \vdash A]_{\rho,\Delta} = [\Gamma \vdash A]_{\rho',\Delta}$. 

12
Proof. A fairly simple induction on the size of $A$. □

Now, we can prove that $C$ contains the sets $[\Gamma \triangleright A]_{\rho \Delta}$.

**Lemma 5.12** If $C$ is the family of saturated sets, whenever $\rho$ satisfies $\Gamma$ at $\Delta$, then $[\Gamma \triangleright A]_{\rho \Delta} \in C_{A[\rho],\Delta}$.

**Proof.** One proceeds by induction on the size of $A$, also adding to the induction hypothesis the fact proved in lemma 5.11 that for any two $\rho$ and $\rho'$ satisfying $\Gamma$ at $\Delta$, if $x[\rho] \xrightarrow{\ast} CC x[\rho']$ for every $x \in dom(\Gamma)$, then $[\Gamma \triangleright A]_{\rho \Delta} = [\Gamma \triangleright A]_{\rho' \Delta}$. □

Given two valid contexts $\Gamma = x_1: A_1, \ldots, x_m: A_m$ and $\Gamma' = x_1': A_1', \ldots, x_m': A_m'$, we say that $\Gamma \xrightarrow{\ast} CC \Gamma'$ iff $A_i \xrightarrow{\ast} CC A_i'$ for all $i, 1 \leq i \leq m$.

**Lemma 5.13** If $C$ is the family of saturated sets, whenever $\rho$ satisfies $\Gamma$ and $\Gamma'$ at $\Delta$ and $\Gamma \xrightarrow{\ast} CC \Gamma'$, then $[\Gamma \triangleright A]_{\rho \Delta} = [\Gamma' \triangleright A]_{\rho \Delta}$.

**Proof.** A fairly simple induction on the size of $A$. □

We also have the following technical property known as “substitution property”. This is perhaps the lemma whose proof is the most technical.

**Lemma 5.14** If $C$ is the family of saturated sets, and $\rho$ satisfies $\Gamma$ at $\Delta$, if $\Gamma, x: K \triangleright A: B$ for some $B$, and $\Gamma \triangleright D: K$ where $K$ is a kind, then

$$[\Gamma \triangleright A[D/x]]_{\rho \Delta} = [\Gamma, x: K \triangleright A]_{\rho[x: = D[\rho], [\Gamma \triangleright D]_{\rho \Delta}]}_{\Delta},$$

and if $\Gamma, x: \sigma \triangleright A: B$ for some $B$, and $\Gamma \triangleright M: \sigma$ where $\sigma$ is a type, then

$$[\Gamma \triangleright A[M/x]]_{\rho \Delta} = [\Gamma, x: \sigma \triangleright A]_{\rho[x: = M[\rho]]_{\Delta}}.$$

**Proof.** In order to prove this lemma, it is necessary to prove the following stronger property:

Assuming that $\rho$ satisfies $\Gamma, \Gamma'$ at $\Delta$, if $\Gamma, x: K, \Gamma' \triangleright A: B$ for some $B$, and $\Gamma \triangleright D: K$ where $K$ is a kind, then

$$[\Gamma, \Gamma'[D/x] \triangleright A[D/x]]_{\rho \Delta} = [\Gamma, x: K, \Gamma' \triangleright A]_{\rho[x: = D[\rho], [\Gamma \triangleright D]_{\rho \Delta}]}_{\Delta},$$

and if $\Gamma, x: \sigma, \Gamma' \triangleright A: B$ for some $B$, and $\Gamma \triangleright M: \sigma$ where $\sigma$ is a type, then

$$[\Gamma, \Gamma'[M/x] \triangleright A[M/x]]_{\rho \Delta} = [\Gamma, x: \sigma, \Gamma' \triangleright A]_{\rho[x: = M[\rho]]_{\Delta}}.$$

The proof of this property is by induction on the size of $A$, and it uses lemma 5.11 and lemma 5.13. □

Using the previous lemma, we can show the following important lemma.
Lemma 5.15 If $C$ is the family of saturated sets, whenever $\rho$ satisfies $\Gamma$ at $\Delta$ and $A \xrightarrow{\rho} A'$, then $[\Gamma \vdash A] \rho \Delta = [\Gamma \vdash A'] \rho \Delta$.

Proof. The proof is by induction on the sum of the sizes of $A$ and $A'$, and it uses lemma 5.11, lemma 5.13, and lemma 5.14. □

Finally, we can prove the main theorem. Recall from definition 5.6 that $\Delta \vdash \Gamma[\rho]$ means

1. $\rho$ satisfies $\Gamma$ at $\Delta$, and
2. $\Delta \triangleright x[\rho] \in [\Gamma \triangleright \Gamma(x)] \rho \Delta$ for every $x \in \text{dom}(\Gamma)$.

It is easy to verify that if $\Delta \subseteq \Delta'$ and $\Delta \vdash \Gamma[\rho]$, then $\Delta' \vdash \Gamma[\rho]$.

Theorem 5.16 If $C$ is the family of saturated sets, whenever $\Gamma \vdash M: A$ and $\Delta \vdash \Gamma[\rho]$, then $\Delta \triangleright M[\rho] \in [\Gamma \triangleright A] \rho \Delta$.

Proof. The proof is by induction on a deduction proving that $A$ type/kind-checks in $\Gamma$. Lemma 5.15 is crucial in taking care of the case where the last inference is the type or kind equality rule. □

As mentioned earlier, if we define $\Delta \vdash (M: A)[\rho]$ iff

1. $\rho$ satisfies $\Gamma$ at $\Delta$, and
2. $\Delta \triangleright M[\rho] \in [\Gamma \triangleright A] \rho \Delta$,

then, the main theorem reads as follows:

Whenever $\Gamma \vdash M: A$ and $\Delta \vdash \Gamma[\rho]$, then $\Delta \vdash (M: A)[\rho]$, and this looks like a Kripke-style type soundness result.

By letting $[\rho]$ be the identity substitution and $\rho_c$ assign the canonical element $\text{can}_{\Gamma(x),\Gamma}$ to each $x \in \text{dom}(\Gamma)$, $\text{can}_{\ast,\Gamma}$ to $\ast$, and $\text{can}_{\text{kind},\Gamma}$ to $\text{kind}$, we obtain the fact that all valid terms of the theory of construction are SN.

Theorem 5.17 Whenever $\Gamma \vdash M: A$, the term $M$ is SN. This applies to kinds, types, and terms (proofs).

An interesting consequence of theorem 5.17 is an elementary proof of the consistency of $CC$. There are other elementary methods for showing that $CC$ is consistent, for example, the "proof-irrelevance semantics", which consists in interpreting types as Zermelo-Fraenkel sets, and $\ast$ as the set $\{0,1\}$ (for details, see Coquand [Coq90]). What is more interesting, is that theorem 5.17 can be used to show in an elementary fashion that certain contexts are consistent, as shown in Coquand [Coq90].
**Definition 5.18** We say that a context $\Delta$ is *consistent* iff there is some valid type $\sigma$ (with $\Delta \vdash \sigma$) such that $\Delta \vdash M: \sigma$ is not provable for any (proof) term $M$. We also say that a type $\sigma$ is *inhabited* in the context $\Delta$ iff there is some (proof) term $M$ such that $\Delta \vdash M: \sigma$ is derivable.

Saying that $CC$ is consistent means that the empty context is consistent, which is equivalent to the fact that some valid closed type is not inhabited. An elegant combinatorial proof of the consistency of $CC$ using theorem 5.17 is given below.

**Lemma 5.19** The theory $CC$ is consistent. Furthermore, the valid type $(\Pi x: \ast)x$ is not inhabited.

**Proof.** First, observe that the judgment $x: \ast \vdash x: \ast$ is derivable, and so $\vdash (\Pi x: \ast)x: \ast$ is derivable. We make use of the following crucial fact: If $M$ is a valid proof in some context $\Delta$ and $M$ is a normal form w.r.t. $\beta$-reduction, then $M$ is of the shape

$$\lambda x_1: A_1. \ldots \lambda x_m: A_m. y N_1 \ldots N_n,$$

where $y$ is a variable possibly among $x_1, \ldots, x_m$, and $N_1, \ldots, N_n$ are normal forms ($m, n \geq 0$), but not necessarily of the same shape as $M$, since some $N_i$'s could be products.

The above fact is easily shown by induction on the size of $M$. The case where $M = M_1 M_2$ is the only interesting one. Because $M$ is normal, $M_1$ cannot be an abstraction. However, it must be a proof, and by the induction hypothesis, it must be either a variable or an application of the form $x N_1 \ldots N_n$.

Now, assume that there is a valid closed proof $M$ such that $\vdash M: (\Pi x: \ast)x$ is derivable. By theorem 5.17 and by lemma 4.12, we can assume that $M$ is in normal form. But then, it is easily seen that it must be the case that we have $M = \lambda x: \ast. y N_1 \ldots N_n$ and that we have a derivation $x: \ast \vdash y N_1 \ldots N_n: x$. However, it is a simple property of $CC$ that for every judgment $\Delta \vdash E$, $FV(E) \subseteq dom(\Delta)$. This implies that $y = x$. However, $x$ is now both a proof and a type, which is impossible by lemma 4.13. \[\Box\]

In Coquand [Coq90], it is shown using theorem 5.17 that a nontrivial context $Inf$ is consistent. The proof is elementary, except for the use of theorem 5.17. As we shall see below, there cannot be any elementary direct proof of the consistency of the context $Inf$ (say in first-order Peano arithmetic, or even in classical higher-order arithmetic). Letting $Void = (\Pi x: \ast)x$ (the "absurd" type),

$$Inf = A: \ast, f: A \rightarrow A, R: A \rightarrow A \rightarrow \ast,$$

$$h_1: (\forall x: A)(Rxx \rightarrow Void),$$

$$h_2: (\forall x, y, z: A)(Rxy \rightarrow Ryz \rightarrow Rxz),$$

$$h_3: (\forall x: A)Rx(fx).$$
The context Inf can be viewed as a kind of axiom of infinity. In turn, it can be shown that the consistency of this context implies the consistency of classical higher-order arithmetic. The proof is elementary, except for the use of theorem 5.17. Thus, by Gödel’s second incompleteness theorem, we obtain that strong normalization in CC (theorem 5.17) is not provable in classical higher-order arithmetic.

Theorem 5.17 and the Church-Rosser property also imply the decidability of type-checking in CC. In fact, a stronger result holds. The main lines of a proof of the above result were given by the first author in a communication to the “Types forum”. This proof is quite similar to a proof by Martin Lof [ML72].

Lemma 5.20 Given any context \( \Delta = x_1: A_1, \ldots, x_n: A_n \) and any expression \( M \), it is decidable whether \( \Delta \vdash \), and if so, whether \( \Delta \vdash M : \text{kind} \) or \( \Delta \vdash M : A \) for some \( A \) (which is given by the algorithm).

Proof sketch. There are two kinds of problems: testing whether \( \Delta \vdash \) or \( \Delta \vdash \ast : \text{kind} \), and testing whether \( M \) kind/type-checks in the context \( \Delta \). We associate a complexity measure to these two problems as follows. Let \( c((x_1: A_1, \ldots, x_n: A_n)) = 1 + \) the sum of the sizes of each \( A_i \) (and the same value for \( c((x_1: A_1, \ldots, x_n: A_n), \ast) \)), and \( c((x_1: A_1, \ldots, x_n: A_n), M) = \) the size of \( M + \) the sum of the sizes of each \( A_i \). We proceed by induction on complexity measures. There are several cases.

1. The problem is \( \Delta \vdash \) or \( \Delta \vdash \ast : \text{kind} \) and \( \Delta = \emptyset \). The answer is yes.

2. The problem is \( x_1: A_1, \ldots, x_n: A_n \vdash \) or \( x_1: A_1, \ldots, x_n: A_n \vdash \ast : \text{kind} \) and \( n > 0 \). Check whether \( A_n \) is well formed in \( x_1: A_1, \ldots, x_{n-1}: A_{n-1} \). If the algorithm returns \( B \), check that either the normal form of \( B \) is \( \ast \), or that \( B \) is a kind.

3. \( M \) is a variable \( x \). Check whether \( A_n \) is well formed in \( x_1: A_1, \ldots, x_{n-1}: A_{n-1} \). If the algorithm returns \( B \), check that the normal form of \( B \) is \( \ast \) or that \( B \) is a kind, and whether \( x \) is one of the \( x_i \).

4. \( M \) is of the form \((\Pi x: A)B \). Check whether \( A \) is well formed in \( x_1: A_1, \ldots, x_n: A_n \) and whether \( B \) is well formed in \( x_1: A_1, \ldots, x_n: A_n, x: A \).

5. \( M \) is of the form \( \lambda x: A. N \). Check whether \( A \) is well formed in \( x_1: A_1, \ldots, x_n: A_n \) and whether \( N \) is well formed in \( x_1: A_1, \ldots, x_n: A_n, x: A \). If the answer to the second problem is yes and the algorithm returns \( P \), then \( x_1: A_1, \ldots, x_n: A_n \vdash M : (\Pi x: A)P \).

6. \( M \) is of the form \( M_1M_2 \). This case requires the fact that every term has a unique normal form. First, we check whether both \( M_1 \) and \( M_2 \) are well-formed in \( x_1: A_1, \ldots, x_n: A_n \). If so, we check whether the normal form of the type/kind of \( M_1 \) is of the form \((\Pi x: A)P \) and the normal form of the type/kind of \( M_2 \) is \( P \).
A closer look at definition 5.5, especially the definitions of \( [\Gamma \vdash (\Pi x: K)D] \Delta \) and \( [\Gamma \vdash (\Pi x: \sigma)D] \rho \Delta \), suggests the definition of certain dependent products. Let \( \Delta \) be a context and \( (\Pi x: K)D \) be a kind or a type such that \( \Delta \vdash (\Pi x: K)D: \kappa, \kappa \in \{*, \text{kind}\} \), with \( K \) a kind.

**Definition 5.21** Let \( \mathcal{A} = (\mathcal{A}_{\Delta'})_{\Delta' \supseteq \Delta} \) be any \( \Delta' \)-indexed family of candidates such that \( \mathcal{A}_{\Delta'} \in \mathcal{C}_{K, \Delta'} \), and let \( F \) be any function with domain \( \{ (\Delta' \vdash A, C) \mid \Delta' \vdash A: K, \Delta' \supseteq \Delta, \text{and} \ C \in \mathcal{C}_{\mathcal{A}, \Delta'} \} \), and such that \( F(\Delta' \vdash A, C) \in \mathcal{C}_{D[\Delta'/\Delta'], \Delta'} \). The dependent product \( \prod(\mathcal{A}, F; (\Pi x: K)D) \) is defined as follows:

\[
\prod(\mathcal{A}, F; (\Pi x: K)D) = \{ \Delta' \vdash M \mid \Delta' \vdash M: (\Pi x: K)D, \Delta' \supseteq \Delta, \text{and} \\
\forall \Delta'' \supseteq \Delta', \forall \Delta'' \vdash A \in \mathcal{A}_{\Delta''}, \forall C \in \mathcal{C}_{\mathcal{A}, \Delta''}, \\
\Delta'' \vdash (MA) \in F(\Delta'' \vdash A, C) \}. 
\]

Let \( \Delta \) be a context and \( (\Pi x: \sigma)D \) be a kind or a type such that \( \Delta \vdash (\Pi x: \sigma)D: \kappa, \kappa \in \{*, \text{kind}\} \), with \( \sigma \) a type.

**Definition 5.22** Let \( \mathcal{A} = (\mathcal{A}_{\Delta'})_{\Delta' \supseteq \Delta} \) be any \( \Delta' \)-indexed family of candidates such that \( \mathcal{A}_{\Delta'} \in \mathcal{C}_{\sigma, \Delta'} \), and let \( F \) be any function with domain \( \{ \Delta' \vdash N \mid \Delta' \vdash N: \sigma, \Delta' \supseteq \Delta \} \), and such that \( F(\Delta' \vdash N) \in \mathcal{C}_{D[N/\Delta'], \Delta'} \). The dependent product \( \prod(\mathcal{A}, F; (\Pi x: \sigma)D) \) is defined as follows:

\[
\prod(\mathcal{A}, F; (\Pi x: \sigma)D) = \{ \Delta' \vdash M \mid \Delta' \vdash M: (\Pi x: \sigma)D, \Delta' \supseteq \Delta, \text{and} \\
\forall \Delta'' \supseteq \Delta', \forall \Delta'' \vdash N \in \mathcal{A}_{\Delta''}, \\
\Delta'' \vdash (MN) \in F(\Delta'' \vdash N) \}. 
\]

Then, we can express \( [\Gamma \vdash (\Pi x: K)D] \rho \Delta \) and \( [\Gamma \vdash (\Pi x: \sigma)D] \rho \Delta \) as dependent products:

\[
[\Gamma \vdash (\Pi x: K)D] \rho \Delta = \prod([\Gamma \vdash K] \rho \Delta', \Delta' \supseteq \Delta, F; ((\Pi x: K)D)[\rho]), 
\]

where \( F \) is the function such that

\[
\langle \Delta' \vdash A, C \rangle \mapsto [\Gamma, x: K \vdash D] \rho [x := \langle A, C \rangle] \Delta', 
\]

with \( \Delta' \vdash A: K[\rho] \) and \( C \in \mathcal{C}_{\mathcal{A}, \Delta'} \), and

\[
[\Gamma \vdash (\Pi x: \sigma)D] \rho \Delta = \prod([\Gamma \vdash \sigma] \rho \Delta', \Delta' \supseteq \Delta, F; ((\Pi x: \sigma)D)[\rho]), 
\]

where \( F \) is the function such that

\[
\Delta' \vdash N \mapsto [\Gamma, x: \sigma \vdash D] \rho [x := N] \Delta', 
\]

with \( \Delta' \vdash N: \sigma[\rho] \).

The definition of \( \prod(\mathcal{A}, F; (\Pi x: \sigma)D) \) is inspired by the definition of the dependent product \( \prod(A, F) \) given by Coquand and Huet on page 107 of their paper [CH88]. The difference
is that Coquand and Huet give a definition of $\Pi(A, F)$ for untyped $\lambda$-terms. They have no
definition analogous to our dependent product $\Pi(A, F; (\Pi x: K) D)$ where $K$ is a kind. Also,
Coquand and Huet's main theorem on page 109 of their paper [CH88], can be considered as
a version of our theorem 5.16 for "stripped terms" (that is, valid terms of $CC$ from which
type information has been erased). However, theorem 5.16 is a stronger result, since it yields
theorem 5.17 as a corollary, whereas Coquand and Huet's theorem only shows that the type
erasure $\text{Erase}(M)$ of any valid term of $CC$ is SN. As far as we know, there does not seem
to be any way to infer from the fact that $\text{Erase}(M)$ is SN that $M$ itself must be SN. This
is in contrast with the situation in $\lambda^\forall$ (and system $F_\omega$).

We now examine the special case of $LF$, and note that strong normalization holds as a
corollary, but does not make any use of families of candidates. Only the canonical $\text{can}_{A, \Delta}$
are needed.

6 Strong Normalization in $LF$

Since $LF$ can be viewed as a fragment of $CC$ obtained by disallowing products and abstrac-
tions over type variables, it follows immediately from theorem 5.17 that all valid terms of
$LF$ are strongly normalizing (under $\beta$-reduction). However, it turns out that the powerful
artillery of the $\mathcal{C}_{A, \Delta}$ is unnecessary to prove this result. In $LF$, we can only have products
of the form $(\Pi x: \sigma) D$, and abstractions of the form $\lambda x: \sigma. B$, when $\sigma$ is a type (but not a kind).
Thus, we have a simpler definition of $[\Gamma \triangleright A] \rho \Delta$. Again, $A$ is either a type family or a kind
valid in $\Gamma$, and the definition only makes sense when $\rho$ satisfies $\Gamma$ at $\Delta$.

**Definition 6.1** In the clauses below, $K$ stands for a kind, $\sigma$ for a type, $A$, $B$ for type
families, $D$ for a kind or a type, $M$ for a type family or a term (proof), and $N$ for a term
(proof).

$$
\begin{align*}
[\Gamma \triangleright \text{kind}] \rho \Delta & = \rho_c(\text{kind}), \\
[\Gamma \triangleright *] \rho \Delta & = \rho_c(*), \\
[\Gamma \triangleright x] \rho \Delta & = \rho_c(x), \\
[\Gamma \triangleright AB] \rho \Delta & = [\Gamma \triangleright A] \rho \Delta (\Delta \triangleright B[\rho], [\Gamma \triangleright B] \rho \Delta), \\
[\Gamma \triangleright AN] \rho \Delta & = [\Gamma \triangleright A] \rho \Delta (\Delta \triangleright N[\rho]), \\
[\Gamma \triangleright (\Pi x: \sigma) D] \rho \Delta & = \{\Delta' \triangleright M \mid \Delta' \triangleright M: ((\Pi x: \sigma) D)[\rho], \Delta' \supseteq \Delta, \text{ and} \\
& \forall \Delta'' \supseteq \Delta', \forall \Delta'' \triangleright N \in [\Gamma \triangleright \sigma] \rho \Delta'', \\
& \Delta'' \triangleright (MN) \in [\Gamma, x: \sigma \triangleright D] \rho[x = N] \Delta'' \}, \\
[\Gamma \triangleright \lambda x: \sigma. B] \rho \Delta & = \lambda(\Delta' \triangleright N). [\Gamma, x: \sigma \triangleright B] \rho[x = N] \Delta', \\
& \text{a function with domain} \\
& \{\Delta' \triangleright N \mid \Delta' \triangleright N: \sigma[\rho], \Delta' \supseteq \Delta \}.
\end{align*}
$$
Remarkably, the candidates, that is, the sets \( C \in \mathcal{C}_{A,\Delta} \), do not appear anywhere in these definitions. The only place where they play a role is in \( \Gamma \triangleright x \rho \Delta \) and \( \Gamma \triangleright \star \rho \Delta \). However, this role is very passive. In fact, all we need to establish strong normalization is to assign the canonical sets and functions \( can_{A,\Delta} \). More precisely, \( \rho_c(\text{kind}) \) is the set \( can_{\ast,\Delta} \) of SN kinds, \( \rho_c(\ast) \) is the set \( can_{\ast,\Delta} \) of SN types, and \( \rho_c(x) = can_{\Gamma[x][\rho_\Delta]} \). Only the substitution component \( \rho_\ast \) of \( \rho \) needs to be arbitrary for the proof to go through, the other component \( \rho_c \) remaining constant (and determined by the canonical elements). Thus, the proof of strong normalization for \( LF \) uses little more than is needed for the proof of strong normalization in the simply-typed \( \lambda \)-calculus, namely the existence of the canonical sets and functions, which itself depends on the existence of the measure \( c(K) \), where \( K \) a kind. This is not surprising in view of another proof by Harper, Honsell, and Plotkin [HHP89], in which a mapping from \( LF \) into the simply-typed \( \lambda \)-calculus is used. It should be noted that their proof applies to \( \beta \) and \( \eta \) reduction, but we do not know presently how to extend our approach to \( \eta \)-reduction.

7 Other Proofs

This section lists other proofs of normalization or strong normalization that we are aware of, in chronological order. We apologize if we are unaware of other proofs not mentioned here. To us, the history of this proof seems sufficiently interesting to be told, especially in a preliminary report, even if it is incomplete. It has been reported that some of these proofs contain errors. We are indeed aware of some errors, and we will briefly mention what they are. We apologize for any (unintentional) omissions or misinterpretations.

1. Coquand, January 1985 [Coq85]. This is Thierry Coquand’s thesis. A proof of normalization is given, as well as some indications on how to extend it to strong normalization. There is a problem with the definition of the sets \( \mathcal{C}_{A,\Delta} \) when \( A \) is a type family of kind \( (\Pi x: B)D \). The members of \( \mathcal{C}_{A,\Delta} \) are indeed functions, but only of one argument, the candidate argument. A similar problem arises in the definition of \( \Gamma \triangleright \lambda x: K. B \rho \Delta \), where the argument \( \Delta' \triangleright A \) is omitted. As a consequence, \( \Gamma \triangleright A \rho \Delta \) is not always well-defined.

2. Jutting, December 1986 [vBJ86]. This is a note attempting to correct Coquand’s proof of normalization given in his thesis. The introduction mentions discussions with Coquand, leading to this note. As we see it, the definition of \( \Gamma \triangleright A \rho \Delta \) is indeed repaired correctly. However, \( \Delta \) is dropped from the \( \mathcal{C}_{A,\Delta} \), which becomes a family of sets of closed terms. To insure that each \( C \in \mathcal{C}_A \) is nonempty (\( A \) a closed type family or closed kind), Jutting adds a countably infinite set of constants. Unfortunately, this causes a problem. Indeed, the language has now been enriched, new types can be formed, and some new closed types may not be inhabited.

3. Coquand, 1987 [Coq87]. This is a note in which Coquand fixes the problem with the addition of new constants, and gives a proof of strong normalization for the first
time. The proof uses infinite contexts, and basically Henkin’s technique for adding new witnesses, so that all closed types are inhabited.

4. Pottinger, February 1987 [Pot87]. This paper refers to Coquand 1987, and gives a proof of strong normalization apparently inspired by Coquand’s proof. Infinite contexts are also used, as well as an idea due to Seldin. Although we need to examine it more closely, the proof seems correct, but rather difficult to follow.

5. Seldin, November 1987 [Sel87]. This is a report, “Mathesis: the Mathematical Foundations of Ulysses”, in which a proof of strong normalization for a variant of the theory of constructions is given. We have not yet had the time to examine this proof carefully, but it appears that it also uses infinite contexts. It appears to be more along the line of Martin Lof’s proof of normalization for $F_\omega$, defined as a Prawitz-style natural deduction system.

6. Zhaohui Luo, 1989 [Luo90]. There is apparently a proof of strong normalization for an extension of $CC$ with universes, given in Luo’s thesis. We do not have this document yet.

7. Geuvers and Nederhof, June 1989 [GN89]. The authors present what they call a modular proof of strong normalization, by reducing strong normalization in $CC$ to strong normalization in Girard’s $F_\omega$. This is accomplished by defining a mapping from $CC$ to $F_\omega$, such that reduction of terms is preserved. Strong normalization for the terms of $F_\omega$ is itself reduced to strong normalization for the erased (raw) terms of $F_\omega$, which is proved directly.


Acknowledgment: We wish to thank Val Breazu-Tannen and Sunil Shende for many helpful comments.

References


