On Coquand's "An Analysis of Girard's Paradox"

Aaron Stump

Computer Systems Laboratory
E-mail: `stump@cs.stanford.edu`

Abstract

In his paper "An Analysis of Girard's Paradox" [3], Coquand presents a result of Girard that minimal higher-order logic extended with quantification over types is inconsistent. Using Girard's idea, he shows that some other extensions of minimal higher-order logic, several extensions of the calculus of constructions, and an early calculus of Martin-Löf with type:type are also inconsistent. He also presents several consistent extensions of minimal higher-order logic and the calculus of constructions. In this paper, I survey relevant background material and present two of Coquand's proofs of inconsistency.

1 Background

In this section, I present some material that is relevant for understanding Coquand's results. Readers familiar with this material may wish to skip to section 2 and refer to this section later as necessary.

1.1 Styles of Axiomatization

There are two styles commonly used for axiomatizing various logics; the style of natural deduction and the style of Hilbert. Systems of natural deduction are characterized by having few axioms or few axiom schemata and many inference rules (see, e.g., [14]). Systems in the style of Hilbert have many axioms or many axiom schemata and few inference rules (see, e.g., [10]).

Figures 1 and 2 give axiomatizations in these two styles of the positive, implicational fragment of propositional logic, which is the logic with just propositional parameters and implication (denoted with "$\rightarrow$"). The system in figure 1 is in the style of Hilbert. Note that the rules are schematic in sentences X, Y, and Z. The system in figure 2 is in the style of natural deduction, where sequents of the form $\Gamma \vdash A$ are used to indicate that A is derivable from the assumptions in Γ, which is a finite (possibly empty) multiset of sentences [1]. Note that these rules are also schematic. Also, Γ, A is an abbreviation for $\Gamma \cup \{A\}$ (where $\cup$ is union for multisets). For both systems, a countably infinite set of propositional parameters is assumed.

$$\frac{}{X \to (Y \to X)}\ \text{(K)}$$

$$\frac{}{(X \to (Y \to Z)) \to ((X \to Y) \to (X \to Z))}\ \text{(S)}$$

$$\frac{X \quad X \to Y}{Y}\ \text{(modus ponens)}$$

Figure 1: An axiomatization in the style of Hilbert

$$\frac{}{\Gamma, A \vdash A}\ \text{(assume)}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to\text{-intro} \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \to\text{-elim}$$

Figure 2: An axiomatization in the style of natural deduction

1.2 Typed Lambda Calculus and Typed Combinatory Logic

Lambda calculus and combinatory logic were developed by Church and Curry, respectively, for applications in the foundations of mathematics [6, 2]. These formalisms are equivalent [6]; see [12] for a proof and a presentation of the translations between them (see also section 4.5.7 of [13]). Figures 3 and 4 present the type systems of the simply typed versions of these systems. Note that in those figures, the intended interpretation of "$A \to B$" is the class of (perhaps only certain) functions from A to B. "$M : \sigma$" indicates that M has type σ. By convention, $\to$ associates to the right and application, which in both systems is denoted by juxtaposition, associates to the left.

The presentation in figure 4 of the simply typed lambda calculus is based closely on the one given in Chapter 4 of [13], but handles variable introduction slightly differently. In the lambda calculus, a countably infinite set of variables is assumed. A simply typed lambda calculus must, of course, have at least one basic type in order to form any terms, but the set of basic types (V say) will be different for different simply typed lambda calculi. Note

[1] This use of multisets of assumptions instead of sets of assumptions, which I learned from Iliano Cervesato, makes the presentation simpler by helping to hide structural rules.

$$\frac{}{K_{X,Y} : X \to (Y \to X)}\ \text{(K)}$$

$$\frac{}{S_{X,Y,Z} : (X \to (Y \to Z)) \to ((X \to Y) \to (X \to Z))}\ \text{(S)}$$

$$\frac{M : X \qquad N : X \to Y}{MN : Y}\ \text{(application)}$$

Figure 3: Simply typed combinatory logic

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ \text{(var)}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda\, x{:}\sigma.\, M : \sigma \to \tau}\ (\to\text{-intro}) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\ (\to\text{-elim})$$

Figure 4: Simply typed lambda calculus

that here, Γ is a finite set, called a context. It is assumed that a variable x does not occur with different types in the same context. Also, in writing $\Gamma, x : \sigma$, it is assumed that x does not occur in Γ. All the types in a context are assumed to be well-formed; i.e., they must be of the form of an element of the free algebra generated from V by $\to$. See Chapter 4 of [13] for a fully detailed presentation.

The presentation in figure 3 of typed combinatory logic follows [12]. Note that we have a special term $K_{X,Y}$ for every two types X and Y, and similarly for $S_{X,Y,Z}$. As in the case of the simply typed lambda calculus, the set of basic types V is left to be specified for particular typed combinatory logics.

1.3 The Curry-Howard Isomorphism

In the simplest case, the so-called Curry-Howard isomorphism (see [11]) is the correspondence between the types in these calculi and the derivable sentences in the logical systems presented in section 1.1. This is what is known as the propositions-as-types (or formulae-as-types) correspondence. The derivable sentences of the logics correspond to the inhabited types of the type theories (i.e., those types σ for which it is derivable that some M has type σ). Note that the rules for the logical systems look exactly like the rules for the typed calculi with everything except the types erased. The derivations in each of the logical systems correspond to the terms of the appropriate typed calculus. For example, $(S_{A,C\to A,A} K_{A,C\to A}) K_{A,C}$ has type $A \to A$ in the combinatory logic, and so corresponds to a derivation of $A \to A$ in the logic in the style of Hilbert. $\lambda\, x{:}A.\, \lambda\, y{:}B.\, x$ has type $A \to (B \to A)$ in the lambda calculus, and so corresponds to a derivation of $A \to (B \to A)$ in the logic in the style of natural deduction.

Howard works through this correspondence for the rest of first-order predicate calculus [11]. For example, a proof of $A \wedge B$ can be seen as corresponding to the pair consisting of a proof for A and a proof for B. $A \times B$, the type of pairs, is thus taken to correspond to the sentence $A \wedge B$. In a system of natural deduction, then, the rules for $\wedge$-introduction and $\wedge$-elimination correspond to the rules for pairing and projections, respectively.

1.4 Polymorphic Lambda Calculus

All but one[2] of the systems Coquand shows inconsistent are based on a polymorphic lambda calculus. In a polymorphic lambda calculus, abstractions over types may be formed, and such abstractions may be applied to types[3]. To avoid confusion with the abstractor over terms, λ, I will denote the abstractor over types by Λ. For example, the term

$$\Lambda\,\tau{:}\mathit{Type}.\,\lambda\,f{:}\tau \to \tau.\,\lambda\,g{:}\tau \to \tau.\,\lambda\,x{:}\tau.\,f\ \ (g\ x)$$

could define a generic simple composition of functions, and applying that term to the type *int*, say, would give the specific simple composition of functions on integers. The type of a term $\Lambda\,\tau{:}\mathit{Type}.\,M$ may be written $\Pi\,\tau{:}\mathit{Type}.\,\sigma$, where M has type σ under the assumption that τ is a type.

In a predicative polymorphic lambda calculus, abstractions may be formed over certain types only. In particular, abstraction over all types is not allowed. Let U_1 be the collection of types of the simply typed lambda calculus. Then for $i > 1$, let U_i be the collection of the types of abstractions over types in U_j, with $j < i$. The only abstractions over types that may be formed are the ones in $\bigcup U_i$. These U_i are called universes. In this calculus, an abstraction over a type is of the form $\Lambda\,\tau{:}U_i.\,M$, and its type is of the form $\Pi\,\tau{:}U_i.\,\sigma$, where M has type σ under the assumption that τ is a type in universe U_i. The type $\Pi\,\tau{:}U_i.\,\sigma$ belongs to universe U_{i+1}. Note that it is not possible to apply an abstraction over a type to that abstraction's own type, because of the restrictions on the universes. Thus a kind of circularity is prevented.

In an impredicative polymorphic lambda calculus, that circularity is allowed[4]. Abstractions may be formed over all types, and so there is just a single universe $Type$ of types, which may be left implicit in terms and types. In this calculus, it is possible to apply an abstraction over a type to that abstraction's own type. The usual example involves the polymorphic identity function I, $\Lambda\,\tau.\,\lambda\,x{:}\tau.\,x$. The type (call it α) of this function is $\Pi\,\tau.\,\tau \to \tau$. If we apply I to its own type, we get a term $I\ \alpha$ which is an identity function over terms of type α. We may thus apply $I\ \alpha$ to I itself. The resulting term, $(I\ \alpha)\ I$, has type α (and with the appropriate rules for conversion, would be convertible to I).

The additional rules needed to extend the simply typed lambda calculus to the impredicative polymorphic lambda calculus are given in figure 5. A countably infinite set of variables, which may be used as type variables or term variables, is assumed. The context Γ is a finite set declaring term variables and type variables. I omit the rules for contexts; see Chapter 9

[2]I.e., the calculus of constructions extended with strong small sums.

[3]The following discussion is based on Chapter 9 of [13].

[4]For a discussion of impredicativity in set theory, see chapter III, section 10 of [8].

of [13] for a detailed presentation. A side condition on Π-introduction is that the type variable t may not appear free in the declarations in Γ. $[\tau/t]\sigma$ denotes the result of substituting τ everywhere for t in σ.

$$\frac{\Gamma, t : Type \vdash M : \sigma}{\Gamma \vdash \Lambda t.\, M : \Pi t.\, \sigma} \ (\Pi\text{-intro}) \qquad \frac{\Gamma \vdash M : \Pi t.\, \sigma \qquad \Gamma \vdash \tau : Type}{\Gamma \vdash MN : [\tau/t]\sigma} \ (\Pi\text{-elim})$$

Figure 5: Rules specific to impredicative polymorphic lambda calculus

1.5 The Burali-Forti Paradox

The technique Coquand uses in [3] to show systems inconsistent relies on an early paradox in set theory called the Burali-Forti paradox[5]. The central idea in this paradox, as in many other set-theoretic paradoxes, is to form a set of all of a certain kind of sets, and then show that the existence of such a set entails a contradiction. Probably the best known such paradox is Russell's, where the existence of the set of all sets that aren't members of themselves is shown to entail a contradiction.

The Burali-Forti paradox relies on the concept of a well-ordered (or well-founded) set. An ordering $<$ on a set A is a binary relation on A that is transitive, irreflexive, and orders every two distinct elements of A (i.e., for every distinct $x, y \in A$, we have $x < y$ or $y < x$). A well-ordering $<$ on A is an ordering with the additional property that every non-empty subset of A has a least element according to $<$ (i.e., for every $S \subseteq A$, there is a $b \in S$ such that for every $x \in S$ with $x \neq b$, $b < x$). A set A under a binary relation $<$ is said to be well-ordered (resp., ordered) if $<$ is a well-ordering (resp., an ordering).

Say that an ordered set $(A, <_A)$ can be embedded in an ordered set $(B, <_B)$ if there is a function (f, say) from A to B that preserves A's ordering (i.e., for every $x, y \in A$, $x <_A y$ implies $f(x) <_B f(y)$). Call two ordered sets similar if one can be embedded in the other by a bijection (see p. 134 of [7]). It can be shown that similarity is an equivalence relation, so we can consider the equivalence classes determined by it. In particular, the ordinals are defined as the equivalence classes of the well-ordered sets; a well-ordered set $(A, <_A)$ is said to have ordinal α if $(A, <_A) \in \alpha$.

The crucial observation leading to the paradox is that the ordinals can themselves be well-ordered. Say that $(A, <_A)$ can be strictly embedded in $(B, <_B)$ if $(A, <_A)$ can be embedded in $(B, <_B)$ (by a function f, say) in such a way that there is an element $b \in B$ such that for every $a \in A$, $f(a) <_B b$. Intuitively, $(A, <_A)$ can be strictly embedded in $(B, <_B)$ if it can be embedded into some proper initial segment of the chain determined by B and $<_B$. Now let $[(A, <_A)]$ and $[(B, <_B)]$ be ordinals and write $[(A, <_A)] \prec [(B, <_B)]$ iff $(A, <_A)$ can be strictly embedded in $(B, <_B)$. It can be shown that $\prec$ is an ordering on the ordinals. It can further be shown that the set of all ordinals less than (according to $\prec$) some ordinal α is well-ordered by $\prec$ and has ordinal α. Finally, it can be shown that no well-ordered set can be strictly embedded in itself (see theorems 13 and 14 of chapter III, section 10 in [7]).

[5]The material in this section is drawn from [8] and [7].

Suppose there is a set W of all ordinals. Using the results stated in the previous paragraph, we can show that $(W, \prec)$ is well-ordered. Furthermore, if X is a well-ordered set with ordinal α, it can be strictly embedded in $(W, \prec)$, for the following reason. X is similar to the set (S, say) of ordinals less than α. But S is a proper initial segment of the chain determined by W and $\prec$. S cannot be similar to W, because W contains at least α in addition to S (since W contains all ordinals), and by the result stated in the previous paragraph, a well-ordered set such as $S \cup \alpha$ cannot be strictly embedded in itself; hence, $S \cup \alpha$ cannot be similar to S. By this result, it should be possible to embed $(W, \prec)$ strictly in itself. But since $(W, \prec)$ is well-ordered, we know that is impossible. So we have a contradiction.

2 Inconsistency of System U

I turn now from background material to Coquand's paper. In this section, I explain how Coquand shows that minimal higher-order logic extended with quantification over types, which he states is essentially the same as a system U analyzed by Girard[6], is inconsistent, in the sense that every sentence is derivable.

2.1 Minimal Higher-Order Logic

System U is an extension of minimal higher-order logic, which is based on the originial higher-order logic of Church [1]. Coquand's Hilbert-style presentation of minimal higher-order logic in [3] is problematic; for example, the rule for modus ponens is missing. In [5], Coquand gives an equivalent system for minimal higher-order logic in the style of natural deduction which does not suffer from the defects of the earlier presentation, although regrettably, it uses Π where the earlier system uses $\forall$, and vice versa. I will use the later system in this paper.

Figure 6 gives the rules for provability in minimal higher-order logic. This logic, like Church's original system, is an extension of the simply typed lambda calculus. There is exactly one base type, o, which will be the type of propositions. There is a constant term, $\Rightarrow$, for implication, which is of type $o \to o \to o$. For each type τ, there is also a constant term $\forall_\tau$, of type $(\tau \to o) \to o$, for universal quantification over terms of type τ. $\forall\, x{:}\tau.\, \phi$ will abbreviate $\forall_\tau(\lambda\, x{:}\tau.\, \phi)$. Note that there is no distinct basic type of individuals in this system, since o is the only basic type.

$\Gamma \vdash \phi$ indicates that the formula ϕ can be derived from the formulas in Γ, which is a finite multiset of assumed formulas. Two side conditions on the rules given in the figure are that in the rule for $\forall$-introduction, x must not appear free in any formula in Γ (to avoid problems with capturing variables in ϕ that occur in Γ); and in the rule for $\forall$-elimination, t must be a term of type τ.

In [5], Coquand states results, proved in [14], that $\wedge$, $\vee$, $\neg$, and $\exists$ are definable in minimal higher-order logic. For example, $\perp$ can be defined as $\forall\, \phi{:}o.\, \phi$, which is a sentence saying that every sentence is true. Then $\neg\phi$ can be defined as $\phi \Rightarrow \perp$. $\phi \wedge \psi$ is defined as

[6]I was unable to track down even a French copy of Girard's thesis, where he presents system U. The system in section 4 of [9] appears to be very similar to the system Coquand presents.

$$\frac{}{\Gamma, A \vdash A} \text{ (assume)}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow\text{-intro} \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow\text{-elim}$$

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \forall\, x{:}\tau.\, \phi} \forall\text{-intro} \qquad \frac{\Gamma \vdash \forall\, x{:}\tau.\, \phi}{\Gamma \vdash [t/x]\phi} \forall\text{-elim}$$

Figure 6: Provability in minimal higher order logic

$\lambda\, \phi{:}o.\, \lambda\, \psi{:}o.\, \forall\, \delta{:}o.\, (\phi \Rightarrow (\psi \Rightarrow \delta)) \Rightarrow \delta$. This definition can be motivated by observing that $\neg(\phi \Rightarrow (\neg\psi))$ would be a semantically correct definition of $\phi \wedge \psi$ for propositional logic, and given our definition of $\neg$, that formula is equivalent to $(\phi \Rightarrow (\psi \Rightarrow \perp)) \Rightarrow \perp$.

2.2 Polymorphic Higher-Order Logic and System U

Polymorphic higher-order logic, as presented in [5], has the same rules for provability as minimal higher-order logic. It is based, however, on impredicative polymorphic lambda calculus, instead of the simply typed lambda calculus. Note that polymorphic higher-order logic has exactly the same quantifiers $\forall_\tau$ that minimal higher-order logic has. In particular, it does not add a new quantifier $\forall_\tau$ for any τ a polymorphic type (i.e., a type containing Π).

System U is polymorphic higher-order logic extended with new quantifiers for polymorphic types. For example, $\exists\, B.\, \exists\, x{:}B.\, \top$ is a well-formed term in system U of type o saying that there exists an inhabited type B. The first quantifier in this term is not allowed in polymorphic higher-order logic, but is allowed in system U. The previous rules for provability in minimal higher-order logic are understood here to include the new quantifiers.

2.3 The Burali-Forti Paradox in System U

Coquand shows how to work the Burali-Forti paradox in system U to show that U is inconsistent. To display the proof in its full detail would require formalizing the arguments given in section 1.5 (and the supporting arguments from [7]), which is a task beyond the scope of this paper. Coquand's own presentation omits almost all the details of the proof. I will show just how to formalize the fundamental concepts needed for the Burali-Forti paradox in system U.

A set A may be represented in U as a type A. A binary relation R on A may be represented as a binary predicate $R : A \to A \to o$; so $R\ x\ y$ represents $R(x, y)$. For clarity, I will make the simplifying assumption that R is defined over all of A. It is easy to represent the property of being a well-ordering. We first define predicates for transitivity and irreflexivity, and the predicate saying that any two elements are comparable:

$$\text{irreflexive} \;=\; \Lambda\, A.\, \lambda\, R{:}A \to A \to o.\, \forall\, x{:}A.\, \neg(R\ x\ x)$$

$$\begin{aligned}
\text{transitive} &= \Lambda\, A.\, \lambda\, R{:}A \to A \to o.\, \forall\, x{:}A.\, \forall\, y{:}A.\, \forall\, z{:}A.\, (R\ x\ y) \Rightarrow (R\ y\ z) \Rightarrow (R\ x\ z) \\
\text{compares} &= \Lambda\, A.\, \lambda\, R{:}A \to A \to o.\, \forall\, x{:}A.\, \forall\, y{:}A.\, (R\ x\ y) \vee (R\ y\ x)
\end{aligned}$$

So (reflexive A R), which is of type o (and hence is a proposition), is the formalization of the statement that R is a reflexive binary relation on A. We can then define the predicates for ordering and well-ordering as follows:

$$\begin{aligned}
\text{ordered} &= \Lambda\, A.\, \lambda\, R{:}A \to A \to o. \\
&\qquad (\text{irreflexive } A\ R) \wedge (\text{transitive } A\ R) \wedge (\text{compares } A\ R) \\
\text{wellordered} &= \Lambda\, A.\, \lambda\, R{:}A \to A \to o.\, \lambda\, E{:}A \to A \to o. \\
&\qquad (\text{ordered } A\ R) \wedge \\
&\qquad (\forall\, P{:}A \to o.\, \exists\, e{:}A. \\
&\qquad\quad (P\ e) \wedge \exists\, b{:}A.\, \forall\, x{:}A.\, (P\ b) \Rightarrow (P\ x) \Rightarrow ((R\ b\ x) \vee (E\ b\ x)))
\end{aligned}$$

The definition of wellordered requires some explanation. First, we represent a subset by its characteristic function $P : A \to o$, where $(P\ x)$ is true iff x is in the subset. Second, in order to express the requirement that every non-empty subset have a least element b, we need a way of saying that every element in the subset that is distinct from b is greater than b. In order to formalize the fact that only elements distinct from b must be greater than b, wellordered requires a relation E on A that serves as an equivalence relation.

Finally, we can define a predicate saying that (A, R) can be strictly embedded in (B, S):

$$\begin{aligned}
\text{strictembed} &= \Lambda\, A.\, \lambda\, R{:}A \to A \to o.\, \Lambda\, B.\, \lambda\, S{:}B \to B \to o.\, \exists\, f{:}A \to B. \\
&\qquad (\forall\, x{:}A.\, \forall\, y{:}A.\, (R\ x\ y) \Rightarrow (S\ (f\ x)\ (f\ y))) \wedge \\
&\qquad (\exists\, b{:}B.\, \forall\, x{:}A.\, (S\ (f\ x)\ b))
\end{aligned}$$

A predicate embed for embeddings that are not strict can be defined simply by omitting the second conjunct in the definition of strictembed.

Coquand shows that a sufficient condition for being able to work the Burali-Forti paradox is to have a type W and a term i of type $\Pi\, A.\, (A \to A \to o) \to W$ that meets the following requirement. Say that two terms of the same type are intentionally equal if the predicates that hold for one are the same as those that hold for the other. Intentional equality can be defined as follows:

$$\begin{aligned}
\text{iequal} &= \Lambda\, A.\, \lambda\, x{:}A.\, \lambda\, y{:}A. \\
&\qquad \forall\, P{:}A \to o.\, (P\ x) \Rightarrow (P\ y)
\end{aligned}$$

The requirement is that if $(i\ A\ R)$ and $(i\ B\ S)$ are intentionally equal, then (embed A R B S) and (embed B S A R) must both be provable. If both (A, R) and (B, S) can be embedded in each other, and they are both ordered, then it follows easily that they must be similar. So we can formalize the statement that (A, R) and (B, S) are similar as (iequal W $(i\ A\ R)$ $(i\ B\ S)$). W and i will serve to formalize the set of all well-ordered sets. Coquand uses the following for W and i:

$$\begin{aligned}
W &= ((\Pi\, A.\, A \to A \to o) \to o) \to o \\
i &= \Lambda\, A.\, \lambda\, R{:}A \to A \to o. \\
&\qquad \lambda\, P{:}(\Pi\, A.\, A \to A \to o) \to o.\, (P\ A\ R)
\end{aligned}$$

These choices of W and i require some explanation. The choice of W is based on a technique of representing a pair as a lambda term. (a, b) is represented as $\Lambda\,\tau.\,\lambda\,P{:}A \to B \to \tau.\,(P\ a\ b)$. The pair is like a server that will run servlets on arguments a and b; the projection functions, for example, apply the pair to ("call the server with servlets") $\lambda\,x{:}A.\,\lambda\,y{:}B.\,x$ and $\lambda\,x{:}A.\,\lambda\,y{:}B.\,y$, respectively. $(i\ A\ R)$, which is of type W, is a lambda term that takes a predicate P over (the representation of) pairs (A, R) and applies P to A and R. So i packages up A and R into a single term. To sketch the proof that this W and i meet Coquand's requirement that (iequal $W\ (i\ A\ R)\ (i\ B\ S)$) implies that (A, R) can be embedded in (B, S) is not hard. One can construct a predicate which says that a given (packaged up) (D, T) can be embedded in (A, R). Though (A, R) cannot be strictly embedded in itself, it can always be embedded in itself, at least trivially. So the predicate is true of $(i\ A\ R)$. It is then true by hypothesis of $(i\ B\ S)$, so (B, S) can be embedded in (A, R).

All the constructions given so far can be carried out in polymorphic higher-order logic. The quantifiers over types of system U are needed to define a well-ordering on W. I will not work through the full construction here, but just demonstrate one part making essential use of type quantifiers. A predicate wellordered_W over W saying that an element of W is the packaged up version of a well-ordered set (B, S) may be defined as follows:

$$\begin{aligned} \text{wellordered}_W \quad = \quad & \lambda\,x{:}W.\,\exists\,B.\,\exists\,S{:}B \to B \to o. \\ & \qquad (\text{iequal}\ W\ x\ (i\ B\ S)) \wedge (\text{wellordered}\ B\ S) \end{aligned}$$

It is worth remarking that in [5], Coquand proves, using a different technique, that polymorphic higher-order logic is inconsistent. That result subsumes the result that system U is inconsistent, since polymorphic higher-order logic is contained in U. Coquand notes that he was unable to run the Burali-Forti paradox in polymorphic higher-order logic. A final note is that Coquand argues in [3] that a system that is just like system U except that it is based on predicative polymorphic lambda calculus instead of impredicative polymorphic lambda calculus, is consistent.

3 Inconsistency of the Calculus of Constructions with Type Quantifiers

The calculus of constructions [4] is based on the Curry-Howard isomorphism extended to a full predicate logic. The Curry-Howard isomorphism can be extended to handle universal quantification in the following way. A universally quantified sentence $\forall\,x{:}A.\,B(x)$ in a predicate logic corresponds to the new type $\Pi\,x{:}A.\,B(x)$, which is called a dependent type. If a function is of dependent type, then not only its result but the type of its result may depend on its input (cf. the rule for Π-elimination for polymorphic lambda calculus in figure 5, where the type of the result of applying a type abstraction to a type τ depends on τ). If B(x) does not contain x free, then $\Pi\,x{:}A.\,B(x)$ is abbreviated $A \to B$. Howard proves that there is a one-to-one correspondence between the derivations of a first-order logic and the terms of a lambda calculus extended with Π [11].

Coquand considers an extension of the calculus of constructions, which we might call the polymorphic calculus of constructions, which has abstractions over types, as in system U.

He states without proof that every derivable sentence in system U can be translated into a provably inhabited type of this extended calculus of constructions; that claim is the natural extension of Howard's result to the polymorphic calculus of constructions. Hence by the inconsistency of U, the type system of the calculus of constructions is inconsistent, if viewed as a logic.

4 Concluding Remarks

I want to thank John Mitchell, Vaughan Pratt, and Grigori Mints for helpful conversations during the course of writing this paper, and the committee for the MTC qualifying examination for their work organizing the exam.

During my preparation for the MTC qual, I encoded a number of formal systems described in this paper in the Edinburgh Logical Framework, a formal metatheory descended, like the calculus of constructions, from de Bruijn's Automath project. Encodings of the following systems and a number of simple proofs in them may be found from http://verify.stanford.edu/~stump: propositional logic in the style of natural deduction; polymorphic lambda calculi as given in [13] (predicative, impredicative, and type:type); typed combinatory logic; modal logics (K, T, S4, and S5); and Milner's pi-calculus.

References

[1] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[2] A. Church. *The Calculi of Lambda-Conversion.* Princeton University Press, 1941.

[3] T. Coquand. An analysis of Girard's paradox. In *1st Symposium on Logic in Computer Science*, pages 227–236, 1986.

[4] T. Coquand. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[5] T. Coquand. A new paradox in type theory. In *9th International Conference on Logic, Methodology, and Philosophy of Science*, pages 555–570, 1994.

[6] H. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland Publishing Company, 1958.

[7] A. Fraenkel. *Abstract Set Theory.* North-Holland Publishing Company, 1976. Revised by A. Levy.

[8] A. Fraenkel, Y. Bar-Hillel, and A. Levy. *Foundations of Set Theory.* Elsevier Science Publishers B.V., 1973.

[9] J.Y. Girard. Une extension de l'interpretation de Gödel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. In *2nd Scandinavian Logic Symposium*, pages 63–92, 1971.

[10] D. Hilbert and W. Ackermann. *Principles of Mathematical Logic*. Chelsea Publishing Company, 1950. Translated from 2nd German edition of 1938.

[11] W. Howard. *The formulae-as-types notion of construction*, pages 479–490. In Seldin and Hindley [15], 1980.

[12] J. Lambek. *From λ-calculus to Cartesian closed categories*, pages 375–402. In Seldin and Hindley [15], 1980.

[13] J. Mitchell. *Foundations for Programming Languages*. The MIT Press, 1996.

[14] D. Prawitz. *Natural Deduction: a Proof-Theoretical Study*. Almquist & Wiksells, 1965.

[15] J. Seldin and J. Hindley, editors. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.