

CHECKING PROOFS IN THE METAMATHEMATICS OF FIRST ORDER LOGIC

Mario Aiello^{oo}

Istituto di Scienze dell'Informazione, Università di Pisa,
Corso Italia 40, 56100 Pisa Italy

and

Richard Weyhrauch

A.I. Lab., Computer Science Department, Stanford University,
Stanford, California 94305 U.S.A.

Abstract

First order theories not only can be used in proving properties of programs, but have also relevance in representation theory. The desire to represent first order theories in a computer in a feasible way requires the facility to discuss mathematical notions. Using metamathematics will eventually allow to construct systems which can formally discuss how they reason. In this paper we present two different first order axiomatizations of the metamathematics of the logic which FOL (First Order Logic proof checker) checks and show several proofs using each one. The difference between the axiomatizations is that one defines the metamathematics in a many sorted logic and the other does not. Proofs are then compared and used to discuss the adequacy of some FOL features.

Section 1 Introduction

This paper represents a first attempt at axiomatizing the metamathematics of a first order theory and at using the new proof checker FOL (First Order Logic). The logic which FOL checks is described in detail in the user manual for this program, Weyhrauch and Thomas 1974. It is based on a system of natural deduction described in Prawitz 1965, 1970.

Our motivation in axiomatizing the metamathematics of FOL was the desire to work on an example which could be used as a case study for projected features of FOL and, at the same time, had independent interest with respect to representing the proofs of significant mathematical results to a computer.

The eventual ability to clearly express the theorems of mathematics to a computer will require the facility to state and prove theorems of metamathematics. There are several clear examples:

- a. Axiom schemas. How exactly do we express that

$$P(\emptyset) \wedge \forall n. (P(n) \supset P(n+1)) \supset \forall n. P(n)$$

is an axiom schema? We need to say: "if for any first order sentence P with one free variable y we denote by $P(n)$ the formula obtained from P by substituting n for y assuming n is free for y in P , then the sentence

$$P(\emptyset) \wedge \forall n. (P(n) \supset P(n+1)) \supset \forall n. P(n)$$

^{oo}

This research was carried out while the author was visiting at the A.I. Lab., Computer Science Department, Stanford University.

is an axiom of arithmetic."

- b. Theorem schemas. The following kind of "theorem" is sometimes seen in set theory books

$$\forall x_1 \dots x_n \exists S. \exists T. \forall u. (\langle x_1, \dots, x_n \rangle \in T \equiv \exists y. (\langle x_1, \dots, x_n, y \rangle \in S)).$$

It asserts the existence of some particular projection of $n+1$ -tuples. In its usual formulation this is not a theorem of set theory at all, but a meta-theorem which states that, for each n , the above sentence is a theorem. We do not know of any machine implementation of first order logic capable of expressing the above notion in a straightforward way.

c. Subsidiary deduction rules. Below we show how to prove that if there is a proof of $\forall x y. WFF$ then there is also a proof of $\forall y x. WFF$, where WFF is any well formed formula. We chose this task because it seemed simple enough to do, and is a theorem which may actually be used. The use of metatheorems as rules of inference by means of a reflection principle will be discussed in a future memo by Richard Weyhrauch. Eventually we hope to check some more substantial metamathematical theorems.

d. Interesting mathematical theorems. We present two examples. The first is any theorem about finite groups. The notion of finite group cannot be defined in the usual first order language of group theory. Thus many "theorems" are actually metatheorems, unless you axiomatize groups in set theory. The second theorem is the "duality principle" in projective geometry.

Finally, from the viewpoint of A.I. and representation theory the ability to state and prove theorems of metamathematics can be very helpful in answering the questions of how we "reflect" on the reasoning we are doing and if a proper axiomatization of the metamathematics of an FOL language together with some sort of computationally realizable reflection principle allows us to discuss in an adequate way our reasonings.

This paper is divided into two sections. In the first one, we present the two axiom systems and the proof of the metatheorem: "for all variables x, y and well formed formulas f , $\forall x y. f$ is a theorem also $\forall y x. f$ is a theorem". In the second section we look at proofs appearing in the appendices in order to explore the features of FOL that need improving and their use in carrying out formal proofs.

In this section we present two axiomatizations of the metamathematics of first order logic. The main difference between them is that one is done in a many sorted first order logic and the other not. These axiomatizations represent an attempt at experimenting with proofs about properties of formulas and deductions. No effort has been spent on guaranteeing that the axioms are independent. It would not only have been uninteresting but also contrary to our basic philosophy. We wish to find axioms which naturally reflect the relevant notions. At the moment this axiomatization is far from being in its final form. Neither the extent of the notions involved nor the best way of expressing them is considered settled.

Strings and sequences of strings have been axiomatized and used to define metamathematical notions. For instance, well formed formulas are represented as strings of symbols which satisfy the predicate FORM defining which combinations of constants, variables predicates and functions symbols represent a wff.; deductions are then represented as sequences of wff's satisfying the predicate PROOFTREE.

2.1 The Sorts

The sorts we have defined correspond to the basic notions of the metamathematics i.e. terms, formulas, individual variables, logical symbols, function symbols etc. and the notions of the domains (strings and sequences of strings) in which the axiomatization has been defined. FOL (see Weyhrauch and Thomas 1974) allows the declaration of variables to be of a certain sort. In the formulas appearing in this paper, each variable is declared to be of some particular sort. For instance $f, f_1, f_2, \dots$ are of sort well formed formulas, $t, t_1, t_2, \dots$ of sort terms etc. For the complete set of FOL declarations see Aiello and Weyhrauch 1974.

2.2 The Domain of Representation of the metamathematics.

The basic notions of the metamathematics of first order logic have been axiomatized in terms of strings and sequences of strings. The primitive functions on them are concatenation (c for strings, cc for sequences) and selectors (car, cdr for strings and $scar, scdr$ for sequences). c and cc are infix operators.

2.2.1 Formulas and terms

Formulas and terms are represented by the string of symbols appearing in them. Terms are defined recursively as strings which either represent an individual variable or can be decomposed into $n+1$ substrings representing a function symbol of arity n , followed by n terms. The two predicates defining terms are:

```
TERMSEQ(0, LAMBDA)
Vs. (TERM(s) INDVAR(s) V Jn fn. (fn=car(s) ^
    n=arity(fn) ^ TERMSEQ(n, cdr(s))))
Vn s. (TERMSEQ(n, s) ((car(s)=LPARSYM) ^ ((len(s)gl s)=
    RPARSYM ^ Jnl. (TERM(substring(s, 2, nl)) ^
    TERMSEQ(n-1, substring(s, nl+1, len(s)-1))))))
```

where the function substring(s, m, n) returns the substring of s starting from its m -th element and ending with the n -th. $len(s)$ computes the length of s and $(n \text{ gl } s)$ selects the n -th element of s .

Well formed formulas (wffs) are represented as strings which either are elementary formulas (defined by the predicate ELF) or can be partitioned into substrings for formulas and logical connectives. Formulas are defined by:

```
Vs. (ELF(s) (s=FALSESYM V PREDPAR0(s) V Jn P. (P=
    =car(s) ^ n=arity(P) ^ TERMSEQ(n, cdr(s)))))
Vs. (FORM(s) ((ELF(s) V Jx f. (s=(x gen f) V s=
    =(x ex f)) V Jf1 f2. (s=(f1 dis f2) V s=
    =(f1 con f2) V s=(f1 impl f2)) V Jf. s=neg(f)))
```

gen is the infix operator that maps its arguments x and f into the string "(FORALLSYM c x) c f " representing the well formed formula $\forall x.f$. The operator ex is used for the existential quantifier. dis , con and $impl$ are the infix operators for the disjunction, conjunction and implication of two formulas. Finally neg is the operator which maps a formula into its negation.

We could possibly represent wffs as structured objects (lists, trees, etc.) which contain all the information about the structure of the formula and do not require any parsing. This approach amounts to axiomatizing metamathematics in terms of the abstract syntax of first order logic, instead of strings of symbols. Both of these possibilities should be explored. We have chosen the first alternative because:

1) It is the most traditional, i.e. metamathematics, as it appears in logic books, is usually stated in terms of strings.

2) Axioms in terms of abstract syntax are simply theorems of the theory expressed in terms of strings. Thus the two representations look substantially the same with respect to "high level" theorems.

3) Ill-formed formulas can be mentioned. This is of course impossible in an axiomatization in terms of the abstract syntax.

The properties of wffs relevant to our theory have been defined by the predicates FR, FRN, GEB and SBT. $FR(x, f)$ is true iff the variable x has at least one free occurrence in the wff f , while $FRN(x, n, f)$ and $GEB(x, n, f)$ are respectively true when the variable x occurs free or bound at the place n in the formula f . In addition to these predicates, some generalized selector functions are defined, which evaluate the first or the k -th free occurrence of a variable in a wff, or the number of its free occurrences. The predicate SBT is then defined. It axiomatizes the notion of substitution of a term for any free occurrence of a variable in a wff.

```
Vx t f1 f2. (SBT(x, t, f1, f2) = Vn1 n2. ((n2=
    =(numbfreeocc(x, n1, f1) * (len(t)-1)) + n1) ^
    ((¬INDVAR(n1 gl f1) ^ (n1 gl f1) =
    =(n2 gl f2)) ^ (INDVAR(n1 gl f1) ^
    ((FRN(x, n1, f1) ^ SUBT(t, f2, n2)) ^
    (¬FRN(x, n1, f1) ^ INVART(n1, f1, n2, f2))))))
```

$\forall t \ f2 \ n2. (SUBT(t, f2, n2) \ \forall x2 \ k. ((k \ g1 \ t) = x2 \rightarrow$
 $FRN(x2, n2 - (len(t) - k), f2)))$,

$\forall n \ f1 \ n1 \ f2. (INVART(n, f1, n1, f2) ((GEB(n1 \ g1 \ f2,$
 $n1, f2) \ GEB(n \ g1 \ f1, n, f1)) \wedge (FRN(n1 \ g1 \ f2, n1, f2) =$
 $FRN(n \ g1 \ f1, n, f1)) \wedge (n1 \ g1 \ f2) = (n \ g1 \ f1)))$

In the previous definition, $n1$ is any position in the string $f1$ and $n2$ is the corresponding position in $f2$. The auxiliary predicate $SUBT$ states that the variables appearing in the term t substituted for a free occurrence of the variable x are still free. $INVART$ defines which properties of $f1$ are still true for $f2$. If the term t is a variable, then SBT reduces to SBV :

$\forall x1 \ x2 \ f1 \ f2. (SBV(x1, x1, f1, f2) \equiv \forall n. ((\neg INDVAR(n \ g1 \ f1) \rightarrow (n \ g1 \ f1) = (n \ g1 \ f2) \wedge (INDVAR(n \ g1 \ f1) \rightarrow ((FRN(x1, n, f1)) \rightarrow FRN(x2, n, f2)) \wedge (\neg FRN(x1, n, f1) \rightarrow INVART(n, f1, f2))))))$,

$\forall n \ f1 \ f2. (INVART(n, f1, f2) \equiv ((GEB(n \ g1 \ f2, n, f2) \ GEB(n \ g1 \ f1, n, f1)) \wedge (FRN(n \ g1 \ f2, n, f2) \rightarrow FRN(n \ g1 \ f1, n, f1)) \wedge (n \ g1 \ f2) = (n \ g1 \ f1)))$.

The proof of the equivalence of SBT and SBV when t is a variable is very simple. It is based on the fact that $n2$ coincides with $n1$ when the term t has length 1 (see Aiello and Weyhrauch 1974). The function sbt (sbv) evaluates to the string representing the result of substituting a term (variable) for every free occurrence of a variable in a given wff. sbt and sbv are defined from the predicates SBT and SBV as follows:

$\forall x \ t \ f1 \ f2. (SBT(x, t, f1, f2) \rightarrow sbt(x, t, f1) = f2)$

$\forall x1 \ x2 \ f1 \ f2. (SBV(x1, x2, f1, f2) \rightarrow sbv(x1, x2, f1) = f2)$

The problem of finding the best way of defining functions in FOL is crucial: in the axiom system given in this paper a uniform way has not been followed. In defining the substitution we are interested in properties of the functions sbt and sbv and in drawing conclusions from the fact that a substitution has been made. It is thus useful to have a predicate which defines the relation between formulas before and after a substitution instead of inferring it from the definition of the functions (stated for example as a system of equations, as in Kleene 1952). One of the motivations of the present experiment was to explore different ways of defining functions. We do not yet have enough examples of proofs to make a clear statement about this matter.

2.2.2 Rules of inference, deductions and the notion of provability

The rules of inference with one premise, are expressed by means of a binary predicate whose arguments are two sequences of wffs (sq , pf) which satisfy $PROOFTREE$.

The predicate is true iff pf is the scdr of sq and the first element of sq is a wff obtained by applying that particular deduction rule to the first wff of pf . The rules with more antecedents are defined in a similar way.

Derivations are recursively defined as sequences of wffs which either are a single wff or are obtained from one or more derivations by applying one of the deduction rules. The recursion is implicitly stated by saying that there exist objects of sort $PROOFTREE$ which satisfy one of the predicates defining the rules of inference. These

sequences represent the linearization of a deduction-tree and are defined as follows:

$\forall sq. (PROOFTREE(sq) \rightarrow (FORM(sq) \vee \exists pf. (ORI(sq, pf) \vee$
 $ANDE(sq, pf) \vee FALSEE(sq, pf) \vee NOTI(sq, pf) \vee$
 $NOTE(sq, pf) \vee IMPLI(sq, pf) \vee \exists pf \ x \ t. (GENI(sq, pf,$
 $x, t) \vee GENE(sq, pf, x, t) \vee EXI(sq, pf, x, t) \vee \exists pf1$
 $pf2. (ANDI(sq, pf1, pf2) \vee FALSEI(sq, pf1, pf2) \vee$
 $IMPLE(sq, pf1, pf2) \vee \exists pf1 \ pf2 \ x1 \ x2. EXE(sq, pf1,$
 $pf2, x1, x2) \vee \exists pf1 \ pf2 \ pf3. ORE(sq, pf1, pf2, pf3) \vee$

A sequence of wffs is a prooftree if either it consists of a single wff or one of the following alternatives holds: there exists another prooftree and a one premise deduction rule has been applied; there exist two prooftrees and one of the two premises rules has been applied; finally, there are three prooftrees and the predicate defining the $\vee$ -elimination rule is true. Note that the root of a prooftree is not necessarily a theorem in a given theory. A predicate $DEPEND$ has been defined which is true if a given wff is a dependence for the root of a prooftree. The axioms about $DEPEND$ allow to decide all the dependencies of a prooftree.

Since some of the deduction rules (the implication introduction, for instance) eliminate dependencies, not all the leaves of a prooftree pf are dependencies for a wff f such that $f = \text{scar}(pf)$. The predicate $DEPEND$ is true only for those leaves of the prooftree which the formula f actually depends on. The axioms $DEPEND$ state which dependencies do not change by applying the deduction rules and are transferred from one prooftree to the other. The axioms $NDEPEND$ state which rules discharge dependencies in a given prooftree.

Using this notion of dependence, the provability of a formula in a theory is defined as follows:

$\forall f. (BEW(f) \equiv \exists sq. (PROOFTREE(sq) \wedge f = \text{scar}(sq) \wedge$
 $\forall f1. (DEPEND(sq, f1) \rightarrow AXIOM(f1)))$

A wff f is a theorem in a given theory if there exists a prooftree whose first element is f and whose only dependencies are axioms in that theory. We have limited our attention to theories in which axioms have no free variables. This property is defined by the axiom: $\forall x \ f. (AXIOM(f) \rightarrow FR(x, f))$.

2.3 The Main Proof in the "any Sorted Logic"

The main theorem we have proved in this axiomatization of the metamathematics states that if $\forall x \ f$. wff is provable in some theory, then $\forall y \ x$. wff is also provable. We have chosen this theorem because, even if very simple, it involves basic notions of provability, substitution and universal quantification. Its proof is found in appendix 2. The theorem depends on the first three lines of the proof. The first step is a lemma stating that $\forall x \ wff. sbt(x, x, wff) = wff$, i.e. substituting a variable x for any free occurrence of x in wff doesn't change that wff. Steps 2 and 3 give simple facts about sequences. The theorem is then proved by instantiating two other lemmas: a) if $\forall x. wff$ is a theorem, then wff is also a theorem; b) if wff is provable, then x cannot be free in the dependencies of the proof of wff and so $\forall x. wff$ is provable. This is of course true only for theories with no free variables in their axioms.

The only property of the inference rules used in this proof involves universal quantification. The restriction on the applicability of the $\forall$ -introduction rule is that the variable to be universal quantified in a wff must not appear free in any of its dependencies. This restriction is reflected in our axiomatization by the predicate APGENI. In this proof APGENI is satisfied because if wff is provable, its dependencies are axioms with no free variables.

The following is an informal proof of the above theorems. If $\forall x.wff$ is provable, then there is a proof tree pf whose first string is $\forall x.wff$. The sequence " $\forall x.wff$) cc pf" is still a proof tree. It is obtained by applying the $\forall$ -elimination rule. The application of this rule doesn't add any dependency to the proof tree. As its only dependencies are axioms, it follows from the definition of BEW that wff is a theorem. On the other hand, if wff is a theorem there exists a proof tree pf whose first element is wff. By applying the $\forall$ -introduction rule to pf we obtain the proof tree " $\forall x.wff$) cc pf". This rule is applicable since theorems have no free variables in their dependencies. It follows that $\forall x.wff$ is a theorem. If $\forall x y.wff$ is provable then $\forall x.wff$ and wff are provable using the first lemma. Finally, we can quantify first over x and then over y, obtaining $\forall y x.wff$ as a theorem.

2.4 Another Axiomatization.

A different axiomatization has been given in an earlier version of FOL where there was no facility for creating sorts. We present it here as we want to do some comparisons between proofs, and discuss some of the features of FOL. Some differences between the two axiomatizations are due to the new features available in FOL. They will be discussed in the next section. Here we only discuss the difference between the definition of formulas and terms. The list of all the axioms can be found in Aiello Weyhrauch 1974.

In this axiomatization, formulas and terms are still represented as the string of the symbols appearing in them. They are defined as strings that can be decomposed into a sequence of substrings recording the construction of that formula or term from elementary formulas and individual variables, according to the usual formation rules (see appendix 3 for the list of axioms). These sequences are defined by the predicate TERMSEQ for terms and FRR for wffs. A sequence satisfies the predicate TERMSEQ if it represents the history of the construction of its first element (the term to be defined), starting from symbols, functions and individual variables. Similarly, a string is a wff if there exists a sequence which satisfies the predicate FRR and represents the history of the construction of that wff from elementary formulas and the logical connectives.

Section 3 The Proofs

In this section we look at the proofs appearing in the appendices, in order to explore the features of FOL that need improving and their use in carrying out formal proofs.

3.1 A Look At Sorts

As already noted, the primary difference

between the two axiomatizations we presented is the introduction of a many sorted logic. In the earlier version of FOL there was no facility for creating sorts, but it soon became evident that relativization of wffs to predicates was desirable. The notion of partially ordered sorts was a natural outgrowth. The axioms in the sorted logic are simpler and more readable and, most important, proofs are considerably shorter. First of all, in the axiomatization done in the earlier version of FOL the partial order of sorts wasn't explicit and was to be derived as a theorem. In the proofs shown in the appendices these theorems appear as dependencies. At the moment FOL has no facility for using already proved statements as lemmas in making new proofs. In FOL there is also the possibility of declaring for each function symbol the sorts of its arguments and of its value. These sorts were defined in the original version by additional axioms. For example, together with the definition of the functions sbt and sbv, the second axiomatization has two extra axioms.

$$\forall x t f1. ((\text{INDVAR}(x) \wedge \text{TERM}(t) \wedge \text{FORM}(f1)) \rightarrow \text{FORM}(\text{sbt}(x, t, f1)))$$

$$\forall x1 x2 f1. ((\text{INDVAR}(x1) \wedge \text{INDVAR}(x2) \wedge \text{FORM}(f1)) \rightarrow \text{FORM}(\text{sbv}(x1, x2, f1)))$$

Proofs are shorter in a many sorted logic. For examples see Aiello and Weyhrauch 1974.

3.2 The Unify And Tautology Commands.

FOL proofs are greatly simplified by the existence of the commands TAUT and TAUTEQ. They decide if a given formula is a tautological consequence of a specified set of wffs. The difference between TAUT and TAUTEQ is that the latter uses properties of the equality and the former doesn't. These commands make proofs shorter since they allow to decide every propositional sentence in one step. As a consequence, the rules of inference most frequently used manipulate quantifiers. The form of almost all the proofs we presented is the same. First of all, the right instantiations of the relevant axioms and theorems are done. Then the propositional consequences are asserted by using TAUT and TAUTEQ. The tautology commands cannot of course manipulate the quantifiers appearing in statements. Hence, the statements produced by them have quantifiers as main symbols or it is necessary to introduce a quantifier to proceed in the proof. After the right introductions or eliminations have been done to them, the tautology commands are used again. This process is iterated until the completion of the proof.

The command UNIFY decides if a given wff can be obtained by instantiation of quantified variables or introduction of them for free occurrences of variables or terms in a second wff. The code for this command has been written by Ashok Chandra and is still in an experimental stage. In the proofs presented here, this command has been used for the simultaneous introduction of the existential quantifier. As an example, consider the following assumption:

$$1 \forall x.(P(x) \rightarrow (Q(f1 c f2) \wedge \forall t.R(t))) \quad (1) \text{ ASSUME}$$

the command

$$\text{unify } \exists x.(P(x) \rightarrow \exists f.(Q(f) \wedge R(g(t)))) , 1;$$

deduces in a single step

2 }x.(P(x)→ }f.(Q(f)∧R(g(t)))) (6) UNIFY 1

A good example of a combined use of these features is found in appendix 1.2:

```
19 FRR(xl gen f) cc SQ=(SEQUENCE((xl gen f) cc
SQ)∧(((xl gen f) cc U)≠SLAMBDA(ELF(scar((xl gen
f) cc SQ)∨(FRR(scdr((xl gen f) cc SQ)∧ }s1 s2.
(STRING(s1)∧(STRING(s2)∧((scar((s1 gen f) cc
SQ)=NEG(s1)∧find(1,s1,scdr((xl gen f) cc SQ)))∨
((scar((xl gen f) cc SQ)=(s1 dis s2)∧find(2,s1
c s2,scdr((xl gen f) cc SQ)))∨((scar((xl gen f)
cc SQ)=(s1 con s2)∧find(2,s1 c s2,scdr((xl gen
f) cc SQ)))∨((scar((xl gen f) cc SQ)=(s1 impl
s2)∧find(2,s1 c s2,scdr((xl gen f) cc SQ)))∨
((scar((xl gen f) cc SQ)=(s1 gen s2)∧(INDVAR(s1)
∧find(1,s2,scdr((xl gen f) cc SQ)))∨(scar((xl
gen f) cc SQ)=(s1 ex s2)∧(INDVAR(s1)∧find(1,s2,
scdr((xl gen f) cc SQ))))))))))
```

```
20 STRING(xl)∧(STRING(f)∧((scar((xl gen f) cc SQ)=
NEG(xl)∧find(1,xl,scdr((xl gen f) cc SQ)))∨
((scar((xl gen f) cc SQ)=(xl dis f)∧find(2,xl
c f,scdr((xl gen f) cc SQ)))∨((scar((xl gen f)
cc SQ)=(xl con f)∧find(2,xl c f, scdr((xl gen
f) cc SQ)))∨((scar((xl gen f) cc SQ)=(xl impl
f)∧find(2,xl c f,scdr((xl gen f) cc SQ)))∨((scar
((xl gen f) cc SQ)=(xl gen f)∧(INDVAR(xl)∧find
(1,f,scdr((xl gen f) cc SQ)))∨(scar((xl gen f)
cc SQ)=(xl ex f)∧(INDVAR(xl)∧find(1,f,scdr((xl
gen f) cc SQ)))))))))) (1 2 3 4 5 6 7 8 11)
```

```
21 }s1 s2.(STRING(s1)∧(STRING(s2)∧((scar((s1 gen
f) cc SQ)=NEG(s1)∧find(1,s1,scdr((xl gen f) cc
SQ)))∨((scar((xl gen f) cc SQ)=(s1 dis s2)∧find
(2,s1 c s2,scdr((xl gen f) cc SQ)))∨((scar((xl
gen f) cc SQ)=(s1 con s2)∧find(2,s1 c s2,scdr((
xl gen f) cc SQ)))∨((scar((xl gen f) cc SQ)=(s1
impl s2)∧find(2,s1 c s2,scdr((xl gen f) cc SQ)))
∨((scar((xl gen f) cc SQ)=(s1 gen s2)∧(INDVAR
(s1)∧find(1,s2,scdr((xl gen f) cc SQ)))∨(scar
((xl gen f) cc SQ)=(s1 ex s2)∧(INDVAR(s1)∧find
(1,s2,scdr((xl gen f) cc SQ)))))))))) (1 2
3 4 5 6 7 8 11)
```

Line 19 is the instantiation of an axiom.

Line 20 is generated by the command,

TAUTEQ 19:#2#2#2#2#2#1#1[sl+f : s2+xl] 1:19;

note how the use of the FOL subpart designators allows us to mention the desired subpart of 19, without having to retype it. In addition we can do the appropriate substitutions. Line 21 is just a use of UNIFY:

UNIFY 19:#2#2#2#2#2 20;

Because we can mention the conclusion, without writing it down explicitly, the amount of typing necessary is severely reduced. Without UNIFY, line 21 would have required two }-introductions and the commands would have been:

```
}I 20 xl+s1 OCC 1,2,3,4,7,8,11,12,15,16,19,20,
23,24;
```

```
}I 20 f+s2 OCC 1,5,9,13,17,18,21,22;
```

We don't enter into details of the command UNIFY. It is our intention to do it elsewhere. It should be thought of as the routine which handles quantifiers in "simple" inferences. As seen above, the saving to a user can be large.

Section 4 Conclusion

The desire to represent mathematics in a

computer in a feasible way certainly requires the facility to discuss metamathematical notions. The axiomatization presented here only treats the syntactic part of the problem. Any mention of the models involved needs the addition of set theory to the axiomatization. However, it is clear from the simple theorems we proved that any practical system needs more extensive features even to do a satisfactory job of writing down the theorems we might want.

An important point for future work is how (in a practical way) to use these theorems. Consider for instance:

$\forall x \ x2 \ f. (BEW(x1 \ gen \ f)) \supset BEW(x2 \ gen(x1 \ gen \ f))$

What we mean by reflection principle is a rule of FOL which says:

BEW(f)	in meta FOL

f	in FOL

That is, if in the axiomatization of the metamathematics of FOL, we can prove the existence of an FOL proof of f, then we can assert f in FOL. Suppose we have a proof in FOL of $\forall x \ y. wff$. Then instantiating the above theorem gives us

$BEW(x \ gen \ (y \ gen \ wff)) \supset BEW(y \ gen \ (x \ gen \ wff))$

Since we started with a proof of $\forall x \ y. wff$ in FOL and BEW represents the proof predicate for FOL, we can conclude $BEW(x \ gen \ (y \ gen \ wff))$. Using modus ponens we get $BEW(y \ gen \ (x \ gen \ wff))$, and using the above rule we can conclude $\forall y \ x. wff$ in FOL.

The exact form of such a rule requires more examples of proofs and is one of the main reasons for doing the example in the paper. It is not just a proof checking exercise, but a case study for fundamental questions of representing mathematical information in a computer. Using metamathematics also prepares the way for more comprehensive systems which can formally discuss how they reason. That is exactly what the metamathematics is good for.

Acknowledgement

Our gratitude to Ashok Chandra who contributed to the early stages of this research. In particular the axioms for strings, sequences and well formed formulas in the unsorted logic are due to him.

Appendix 1

The proof of "if f is a WFF. also $\forall x. f$ is a WFF."

1.1 FOL commands and printout in the many sorted logic

Commands

WE WFF1, x gen f;

TAUTEQ(x gen f=x gen f)∨(x gen f=x ex f);

Unify --: #2#2#1,-;

TAUT ---: #1, 1:-;

Proof

```
1 FORM(x gen f)=(ELF(x gen f)∨(}x1 f1.((x gen f)=
(x1 gen f1)∨(x gen f)=(x1 ex f1))∨(}f1 f2.((x
gen f)=(f1 dis f2)∨(x gen f)=(f1 con f2)∨(x gen
f)=(f1 impl f2)))∨}f1.(x gen f)=neg(f1)))
```

2 (x gen f)=(x gen f) $\vee$ (x gen f)=(x ex f)
3 $\exists$ xl fl.((x gen f)=(xl gen f) $\vee$ (x gen f)=(xl ex f))
4 FORM(x gen f)

1.2 Printout of the oproof in the second axiomatization

1 FORM(f) $\wedge$ INDVAR(x1)
2 $\forall$ sq s.((SEQUENCE(sq) $\wedge$ sq $\neq$ SLAMBDA) $\supset$ (STRING(s) $\supset$ (s cc sq) $\neq$ SLAMBDA)) (2)
3 $\forall$ s sq.((STRING(s) $\wedge$ SEQUENCE(sq)) $\supset$ scar(s cc sq)=s) (3)
4 $\forall$ s sq.((STRING(s) $\wedge$ SEQUENCE(sq)) $\supset$ s cdr(s cc sq)=sq) (4)
5 $\forall$ sq.((SEQUENCE(sq) $\wedge$ sq $\neq$ SLAMBDA) $\equiv$ find(1,scar(sq),sq)) (5)
6 $\forall$ f x.((FORM(f) $\wedge$ INDVAR(x)) $\supset$ STRING(x gen f)) (6)
7 $\forall$ s sq.((STRING(s) $\wedge$ SEQUENCE(sq)) $\supset$ SEQUENCE(s cc sq)) (7)
8 $\forall$ x.((INDVAR(x) $\supset$ STRING(x)) (8)
9 FORM(f) $\equiv$ (STRING(f) $\wedge$ $\exists$ sq.(FRR(sq) $\wedge$ f=scar(sq)))
10 $\exists$ sq.(FRR(sq) $\wedge$ f=scar(sq)) (1 2 3 4 5 6 7 8)
11 FRR(SQ) $\wedge$ f=scar(SQ) (11)
12 FRR(SQ) $\equiv$ (SEQUENCE(SQ) $\wedge$ (SQ $\neq$ SLAMBDA $\wedge$ (ELF(scar(SQ)) $\vee$ (FRR(s cdr(SQ)) $\wedge$ $\exists$ s1 s2.(STRING(s1) $\wedge$ (STRING(s2) $\wedge$ ((scar(SQ)=NEG(s1) $\wedge$ find(1,s1,s cdr(SQ))) $\vee$ ((scar(SQ)=(s1 dis s2) $\wedge$ find(2,s1 c s2,s cdr(SQ))) $\vee$ ((scar(SQ)=(s1 con s2) $\wedge$ find(2,s1 c s2,s cdr(SQ))) $\vee$ ((scar(SQ)=(s1 impl s2) $\wedge$ find(2,s1 c s2,s cdr(SQ))) $\vee$ ((scar(SQ)=(s1 gen s2) $\wedge$ (INDVAR(s1) $\wedge$ find(1,s2,s cdr(SQ))) $\vee$ (scar(SQ)=(s1 ex s2) $\wedge$ (INDVAR(s1) $\wedge$ find(1,s2,s cdr(SQ))))))))))))))
13 (SEQUENCE(SQ) $\wedge$ sq $\neq$ SLAMBDA) $\supset$ (STRING(x1 gen f) $\supset$ ((x1 gen f) cc SQ) $\neq$ SLAMBDA) (2)
14 (STRING(x1 gen f) $\wedge$ SEQUENCE(SQ)) $\equiv$ scar((x1 gen f) cc SQ)(x1 gen f) (3)
15 (STRING(x1 gen f) $\wedge$ SEQUENCE(SQ)) $\equiv$ s cdr((x1 gen f) cc SQ)=SQ (4)
16 (SEQUENCE(SQ) $\wedge$ sq $\neq$ SLAMBDA) $\equiv$ find(1,scar(SQ),SQ) (5)
17 (FORM(f) $\wedge$ INDVAR(x1)) $\supset$ string(x1 gen f) (6)
18 INDVAR(x1) $\supset$ STRING(x1) (8)
19 FRR(x1 gen f) cc SQ $\equiv$ (SEQUENCE((x1 gen f) cc SQ) $\wedge$ ((x1 gen f) cc U) $\neq$ SLAMBDA $\wedge$ (ELF(scar((x1 gen f) cc SQ)) $\vee$ (FRR(s cdr((x1 gen f) cc SQ)) $\wedge$ $\exists$ s1 s2.(STRING(s1) $\wedge$ (STRING(s2) $\wedge$ ((scar((s1 gen f) cc SQ)=NEG(s1) $\wedge$ find(1,s1,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(s1 dis s2) $\wedge$ find(2,s1 c s2,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(s1 con s2) $\wedge$ find(2,s1 c s2,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(s1 impl s2) $\wedge$ find(2,s1 c s2,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(s1 gen s2) $\wedge$ (INDVAR(s1) $\wedge$ find(1,s2,s cdr((x1 gen f) cc SQ))) $\vee$ (scar((x1 gen f) cc SQ)=(s1 ex s2) $\wedge$ (INDVAR(s1) $\wedge$ find(1,s2,s cdr((x1 gen f) cc SQ))))))))))))))
20 STRING(x1) $\wedge$ (STRING(f) $\wedge$ ((scar((x1 gen f) cc SQ)=NEG(x1) $\wedge$ find(1,x1,s cdr((x1 gen f) cc SQ))) $\vee$

((scar((x1 gen f) cc SQ)=(x1 dis f) $\wedge$ find(2,x1 c f,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(x1 con f) $\wedge$ find(2,x1 c f,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(x1 impl f) $\wedge$ find(2,x1 c f,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(x1 gen f) $\wedge$ (INDVAR(x1) $\wedge$ find(1,f,s cdr((x1 gen f) cc SQ))) $\vee$ (scar((x1 gen f) cc SQ)=(x1 ex f) $\wedge$ (INDVAR(x1) $\wedge$ find(1,f,s cdr((x1 gen f) cc SQ)))))))))) (1 2 3 4 5 6 7 8 11)
21 $\exists$ s1 s2.(STRING(s1) $\wedge$ (STRING(s2) $\wedge$ ((scar((s1 gen f) cc SQ)=NEG(s1) $\wedge$ find(1,s1,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(s1 dis s2) $\wedge$ find(2,s1 c s2,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(s1 con s2) $\wedge$ find(2,s1 c s2,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(s1 impl s2) $\wedge$ find(2,s1 c s2,s cdr((x1 gen f) cc SQ))) $\vee$ ((scar((x1 gen f) cc SQ)=(s1 gen s2) $\wedge$ (INDVAR(s1) $\wedge$ find(1,s2,s cdr((x1 gen f) cc SQ))) $\vee$ (scar((x1 gen f) cc SQ)=(s1 ex s2) $\wedge$ (INDVAR(s1) $\wedge$ find(1,s2,s cdr((x1 gen f) cc SQ)))))))))))))) (1 2 3 4 5 6 7 8 11)
22 (STRING(x1 gen) $\wedge$ SEQUENCE(SQ)) $\supset$ SEQUENCE((x1 gen f) cc SQ) (7)
23 FORM(x1 gen f) (STRING(x1 gen f) $\wedge$ $\exists$ sq.(FRR(sq) $\wedge$ (x1 gen f)=scar(sq)))
24 FRR(x1 gen f) cc SQ $\wedge$ (x1 gen f)=scar((x1 gen f) cc SQ) (1 2 3 4 5 6 7 8 11) TAUTEQ I:23
25 $\exists$ sq.(FRR(sq) $\wedge$ (x1 gen f)=scar(sq)) (1 2 3 4 5 6 7 8 11)
26 FORM(x1 gen f) (1 2 3 4 5 6 7 8 11)
27 FORM(x1 gen f) (1 2 3 4 5 6 7 8)
28 (FORM(f) $\wedge$ INDVAR(x1)) $\supset$ FORM(x1 gen f) (2 3 4 5 6 7 8)
29 $\forall$ x1.((FORM(f) $\wedge$ INDVAR(x1)) $\supset$ FORM(x1 gen f)) (2 3 4 5 6 7 8)

Appendix 2

The proof that universal quantifiers can be interchanged Printout of the proof in the many sorted logic

1 $\forall$ x f.sbt(x,x,f)=f (1)
2 $\forall$ f sq.scar(f cc sq)=f (2)
3 $\forall$ f sq.scdr(f cc sq)=sq (3)
4 BEW(x gen f) (4)
5 sbt(x,x,f)=f (1)
6 BEW(x gen f) $\equiv$ $\exists$ sq.(PROOFTREE(sq) $\wedge$ ((x gen f)=scar(sq) $\wedge$ $\forall$ fl.(DEPEND(sq,fl) $\supset$ AXIOM(fl))))
7 $\exists$ sq.(PROOFTREE(sq) $\wedge$ ((x gen f)=scar(sq) $\wedge$ $\forall$ fl.(DEPEND(sq,fl) $\supset$ AXIOM(fl)))) (4)
8 PROOFTREE(sq) $\wedge$ ((x gen f)=scar(sq) $\wedge$ $\forall$ fl.(DEPEND(sq,fl) $\supset$ AXIOM(fl))) (8)
9 GENE(f cc sq,sq,x,x) $\equiv$ (s cdr(f cc sq)=sq $\wedge$ (PROOFTREE(sq) $\wedge$ $\exists$ f.(scar(sq)=(x gen f) $\wedge$ scar(f cc sq)=sbt(x,x,f))))
10 scar(f cc sq)=f (2)
11 s cdr(f cc sq)=sq (3)
12 scar(sq)=(x gen f) $\wedge$ scar(f cc sq)=sbt(x,x,f) (1 2 3 4 8)
13 $\exists$ fl.(scar(sq)=(x gen f) $\wedge$ scar(f cc sq)=sbt(x,

```

x,fl)) (1 2 3 4 8)
14 GENE(f cc sq,sq,x,x) (1 2 3 4 8)
15 PROOFTREE(f cc sq)= (FORM(f cc sq)V(∃pf.(ORI(f
cc sq,pf)V(ANDE(f cc sq,pf)V(FALSEE(f cc sq,pf)
V(NOTI(f cc sq,pf)V(NOTE(f cc sq,pf)VIMPLI(f
cc sq,pf))))))V(∃pf x t.(GENI(f cc sq,pf,x,t)
V(GENE(f cc sq,pf,x,t)VEXI(f cc sq,pf,t)))V(∃
pf1 pf2.(ANDI(f cc sq,pf1,pf2)V(FALSEI(f cc sq,
pf1,pf2)VIMPLE(f cc sq,pf1,pf2)))V(∃pf1 pf2 x
t.EXE(f cc sq,pf1,pf2,x,t)V(∃pf1 pf2 pf3.ORE(f
cc sq, pf1,pf2,pf3))))))
16 GENI(f cc sq,sq,x,x)V(GENE(f cc sq,sq,x,x)V
EXI(f cc sq,sq,x,x)) (1 2 3 4 8)
17 ∃pf x t.(GENI(f cc sq,pf,x,t)V(GENE(f cc sq,pf,
x,t)VEXI(f cc sq,pf,t))) (1 2 3 4 8)
18 PROOFTREE(f cc sq) (1 2 3 4 8)
19 ∀ fl.(DEPEND(sq,fl)→AXIOM(fl)) (8)
20 DEPEND(sq,fl)→AXIOM(fl) (8)
21 PROOFTREE(f cc sq)→(PROOFTREE(sq)→((sq=scdr(f
cc sq)→(DEPEND(f cc sq,fl)→DEPEND(sq,fl)))→(
ORI(f cc sq,sq)V(ANDE(f cc sq,sq)V(FALSEE(f cc
sq,sq)V(∃f.((NOTID(f cc sq,sq,f)V(NOTED(f cc
sq,sq,f)VIMPLID(f cc sq,sq,f)))→f#fl)V∃x t.(
GENI(f cc sq,sq,x,t)V(GENE(f cc sq,sq,x,t)V
EXI(f cc sq,sq,x,t))))))))))
22 ∃x t.(GENI(f cc sq,sq,x,t)V(GENE(f cc sq,sq,
x,t)VEXI(f cc sq,sq,x,t))) (1 2 3 4 8)
23 DEPEND(f cc sq,fl)→AXIOM(fl) (1 2 3 4 8)
24 ∀fl.(DEPEND(f cc sq,fl)→AXIOM(fl)) (1 2 3 4 8)
25 f=scar(f cc sq) (2)
26 PROOFTREE(f cc sq)^(f=scar(f cc sq)→f1.(
DEPEND(f cc sq,fl)→AXIOM(fl))) (1 2 3 4 8)
27 BEW(f)→∃sq.(PROOFTREE(sq)^(f=scar(sq)→f1.(
DEPEND(sq,fl)→AXIOM(fl))))
28 ∃sq.(PROOFTREE(sq)^(f=scar(sq)→f1.(DEPEND(sq,
fl)→AXIOM(fl)))) (1 2 3 4)
29 BEW(f) (1 2 3 4)
30 BEW(x gen f)→BEW(f) (1 2 3)
31 BEW(f) (31)
32 ∃sq.(PROOFTREE(sq)^(f=scar(sq)→f1.(DEPEND(sq,
fl)→AXIOM(fl)))) (31)
33 PROOFTREE(sq)^(f=scar(sq)→f1.(DEPEND(sq,fl)→
AXIOM(fl))) (33)
34 ∀f1.(DEPEND(sq,fl)→AXIOM(fl)) (33)
35 DEPEND(sq,fl)→AXIOM(fl) (33)
36 APGENI(x,sq)≡(∀f.(DEPEND(sq,f)→¬FR(x,f))∧
PROOFTREE(sq))
37 AXIOM(fl)→¬FR(x,fl)
38 DEPEND(sq,fl)→¬FR(x,fl) (31 33)
39 ∀f1.(DEPEND(sq,fl)→¬FR(x,f1)) (31 33)
40 APGENI(x,sq) (31 33)
41 GENI((x gen f) cc sq,sq,x,x)≡(scdr(x gen f) cc
sq)=sq∧(PROOFTREE(sq)∧∃f1.(scar((x gen f) cc
sq)=(x gen f1)∧(scar(sq)=sbt(x,x,f1)∧APGENI(x,
sq))))
42 scar((x gen f) cc sq)=(x gen f) (2)
43 scdr((x gen f) cc sq)=sq (3)
44 scar((x gen f) cc sq)=(x gen f)∧(scar(sq)=sbt
(x,x,f)∧APGENI(x,sq)) (1 2 3 31 33)
45 ∃f1.(scar((x gen f) cc sq)=(x gen f1)∧(scar(sq)
=sbt(x,x,f1)∧APGENI(x,sq))) (1 2 3 31 33)
46 GENI((x gen f) cc sq,sq,x,x) (1 2 3 31 33)
47 PROOFTREE((x gen f) cc sq)→(FORM((x gen f) cc
sq)V(∃pf.(ORI((x gen f) cc sq,pf)V(ANDE((x gen
f) cc sq,pf)V(FALSEE((x gen f) cc sq,pf)V(NOTI
((x gen f) cc sq,pf)V(NOTE((x gen f) cc sq,pf)
VIMPLI((x gen f) cc sq,pf))))))V(∃pf x1 t.(GENI
((x gen f) cc sq,pf,x1,t)V(GENE((x gen f) cc
sq,pf,x1,t)VEXI((x gen f) cc sq,pf,x1,t)))V
(∃pf1 pf2.(ANDI((x gen f) cc sq,pf1,pf2)V(
FALSEI((x gen f) cc sq,pf1,pf2)VIMPLE((x gen f)
cc sq,pf1,pf2)))V(∃pf1 pf2 x1 t.EXE((x gen f)
cc sq,pf1,pf2,x1,t)V(∃pf1 pf2 pf3.ORE((x gen f)
cc sq,pf1,pf2,pf3))))))
48 GENI((x gen f)cc sq,sq,x,x)V(GENE((x gen f) cc
sq,sq,x,x)VEXI((x gen f) cc sq,sq,x,x)) (1 2
3 31 33)
49 ∃pf x1 t.(GENI((x gen f) cc sq,pf,x1,t)V(GENE
((x gen f) cc sq,pf,x1,t)VEXI((x gen f) cc sq,
pf,x1,t))) (1 2 3 31 33)
50 PROOFTREE((x gen f) cc sq) (1 2 3 31 33)
51 PROOFTREE(x gen f) cc sq)→(PROOFTREE(sq)→((sq=
scdr((x gen f) cc sq)→(DEPEND((x gen f) cc sq,
fl) (DEPEND(sq,fl))) (ORI((x gen f) cc sq,sq)V
(ANDE((x gen f) cc sq,sq)V(FALSEE((x gen f) cc
sq,sq)V(∃f.((NOTID((x gen f) cc sq,sq,f)V(NOTED
((x gen f) cc sq,sq,f)VIMPLID((x gen f) cc sq,
sq,f)))→f#fl)V∃x1 t.(GENI((x gen f) cc sq,sq,
x1,t)V(GENE((x gen f) cc sq,sq,x1,t)VEXI(x gen
f) cc sq,sq,x1,t))))))))))
52 ∃x1 t.(GENI((x gen f) cc sq,sq,x1,t)V(GENE((x
gen f) cc sq,sq,x1,t)VEXI((x gen f) cc sq,sq,
x1,t))) (1 2 3 31 33)
53 DEPEND((x gen f) cc sq,fl)→AXIOM(fl) (1 2 3
31 33)
54 ∀f1.(DEPEND((x gen f) cc sq,fl)→AXIOM(fl))
(1 2 3 31 33)
55 (x gen f)=scar((x gen f) cc sq) (2)
56 PROOFTREE((x gen f) cc sq)^(x gen f)=scar((x
gen f) cc sq)→f1.(DEPEND((x gen f) cc sq,fl)→
AXIOM(fl)) (1 2 3 31 33)
57 BEW(x gen f)→∃sq.(PROOFTREE(sq)^(x gen f)=
scar(sq)→f1.(DEPEND(sq,fl)→AXIOM(fl))))
58 ∃sq.(PROOFTREE(sq)^(x gen f)=scar(sq)→f1.(
DEPEND(sq,fl)→AXIOM(fl))) (1 2 3 31)
59 BEW(x gen f) (1 2 3 31)
60 BEW(f)→BEW(x gen f) (1 2 3)
61 BEW(x gen f)≡BEW(f) (1 2 3)
62 ∀x f.(BEW(x gen f)≡BEW(f)) (1 2 3)
63 BEW(x1 gen(x2 gen f))≡BEW(x2 gen f) (1 2 3)
64 BEW(x2 gen f)≡BEW(f) (1 2 3)
65 BEW(x1 gen f)≡BEW(f) (1 2 3)
66 BEW(x2 gen(x1 gen f))≡BEW(x1 gen f) (1 2 3)
67 BEW(x1 gen(x2 gen f))→BEW(x2 gen(x1 gen f))
(1 2 3)
68 ∀x1 x2 f.(BEW(x1 gen(x2 gen f))→BEW(x2 gen(x1
gen f))) (1 2 3)

```

Appendix 3

The axioms for formulas in the unsorted logic

AXIOM FIND:

$\forall sq. (FIND(\emptyset, LAMBDA, sq) \equiv SEQUENCE(sq)),$
 $\forall n \ s \ sq. (FIND(n, s, sq) \equiv INTEGER(n) \wedge STRING(s) \wedge$
 $SEQUENCE(sq) \wedge \exists n \ s1 \ s2. (INTEGER(n) \wedge STRING(s1) \wedge$
 $STRING(s2) \wedge (\emptyset < s \wedge s < slen(sq)) \wedge (s1 = (n \ s1 \ sq)) \wedge$
 $(s = (s1 \ c \ s2)) \wedge FIND(n-1, s2, sq))) ; ;$

AXIOM FINDTOP:

$\forall sq. (FINDTOP(\emptyset, SLAMBDA, sq) \equiv SEQUENCE(sq)), \forall n \ s$
 $sq. (FINDTOP(n, s, sq) \equiv INTEGER(n) \wedge STRING(s) \wedge$
 $SEQUENCE(sq) \wedge \exists s1 \ s2. (STRING(s1) \wedge STRING(s2) \wedge$
 $(s1 \neq LAMBDA) \wedge (s = (s1 \ c \ s2)) \wedge (s = scar(sq)) \wedge$
 $FINDTOP(n-1, s2, scar(sq)))) ; ;$

AXIOM TERM:

$\forall sq. (TERMSEQ(sq) \equiv SEQUENCE(sq) \wedge ((slen(sq) = 1 \wedge$
 $INDVAR(1 \ s1 \ sq)) \vee (slen(sq) > 1 \wedge TERMSEQ(scdr$
 $(sq)) \wedge (INDVAR(scar(sq)) \vee \exists n \ s. INTEGER(n) \wedge$
 $STRING(s) \wedge (s = car(scar(sq)) \wedge OPCONST(s) \wedge n = arity$
 $(s) \wedge FIND(n, cdr(scar(sq)), scdr(sq)))))) ; ;$
 $\forall t. (TERM(t) \equiv STRING(t) \wedge \exists sq. (TERMSEQ(sq) \wedge t = car$
 $(sq))) ; ;$

AXIOM WFF:

$\forall f. (ELF(f) \equiv STRING(f) \wedge (f = FALSESYMVPREDPARO(f) \vee$
 $\exists n \ sq. (INTEGER(n) \wedge SEQUENCE(sq) \wedge PREDPAR(car$
 $(f)) \wedge n = arity(car(f)) \wedge TERMSEQ(sq) \wedge FINDTOP(n,$
 $cdr(f), sq)))) ; ;$
 $\forall sq. (FRR(sq) \equiv SEQUENCE(sq) \wedge (sq \neq SLAMBDA) \wedge (ELF$
 $(scar(sq)) \vee (FRR(scdr(sq)) \wedge \exists s1 \ s2. (STRING(s1) \wedge$
 $STRING(s2) \wedge ((scar(sq) = neg(s1) \wedge FIND(1, x1,$
 $scdr(sq)) \vee (scar(sq) = (s1 \ dis \ s2) \wedge FIND(2, s1 \ c$
 $s2), scdr(sq)) \vee (scar(sq) = (s1 \ con \ s2) \wedge FIND(2,$
 $(s1 \ c \ s2), scdr(sq)) \vee (scar(sq) = (s1 \ impl \ s2) \wedge$
 $FIND(2, (s1 \ c \ s2), scdr(sq)) \vee (scar(sq) = (s1 \ gen$
 $s2) \wedge INDVAR(s1) \wedge FIND(1, s2, scdr(sq)) \vee (scar(sq)$
 $= (s1 \ ex \ s2) \wedge INDVAR(s1) \wedge FIND(1, s2, scdr(sq))))$
 $)))) ; ;$
 $\forall f. (FORM(f) \equiv STRING(f) \wedge \exists sq. (FRR(sq) \wedge f = scar(sq)))$

References

- Prawitz, D.,
1965 Natural deduction, A proof theoretical study
Almqvist and Wiksell, Stockholm (1965).
- Godel, K.,
1930 Die Vollständigkeit der Axiome des logischen Funktionenkalküls Monatshefte für Mathematik und Physik 37, (1930) 349-360.
- Godel, K.,
1931 Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I, Monatshefte für Mathematik und Physik 38, (1931) 173-198.
- Weyhrauch, R.W., and Thomas, A.J.,
1974 FOL: A Proof Checker for First-order Logic, Stanford Artificial Intelligence Laboratory, Memo AIM-235 (1974).
- Aiello, M., and Weyhrauch, R.W.,
1974 Checking Proofs in the metamathematics of first order logic, Stanford Artificial Intelligence Laboratory, Memo AIM-222 (1974).