A THEOREM PROVER FOR ELEMENTARY SET THEORY

F. M. Brown
Department of Artificial Intelligence
University of Edinburgh
Scotland

Abstract

We describe a theorem prover for elementary set theory which is based on truth value preserving transformations, and then give an example of the protocol produced by this system when trying to prove the theorem of set theory known as Cantor's Theorem.

1. Introduction

This is a report of some of our research carried out mainly during the summer and fall of 1974, it describes an implementation of a theorem prover based on truth value preserving transformations which has been applied to proving theorems in the domain of elementary set theory.

In section 2 we describe the basic deductive system, and in section 3 we present some protocols produced by the. theorem prover while trying to prove some theorems of set theory. Finally, in section 4 we draw a few conclusions.

2. Description of the Theorem Prover

Our theorem prover consists of an interpreter for mathematical expressions and many items of mathematical knowledge. This interpreter is a fairly complex mechanism, but it may be viewed as applying items of mathematical knowledge of the form $\phi \leftrightarrow \psi$ or $\phi = \psi$ to the theorem being proven, in the following manner: The interpreter evaluates the theorem recursively in a call-by-need manner. That is, if $(f a_1 \ldots a_n)$ is a sub-expression being evaluated, then the interpreter tries to apply its items of knowledge to that sub-expression before evaluating the arguma$_1 \ldots$a$_n$. For each sub-expression that the interpreter evaluates, in turn it tries to match the ϕ expression of an item to that sub-expression. If, however, during the application process an argument a does not match the corresponding argument of the ϕ expression, then a$_i$ is evaluated, and the system then tries to match the result of that evaluation. If ever the interpreter finds a sub-expression $\phi\theta$ which is an instance of ϕ of some item, then it replaces that expression by the corresponding instance $\psi\theta$ of ψ. At this point all memory of the sub-expression $\phi\theta$ is immediately lost and the interpreter now evaluates $\psi\theta$. If no items can be applied to a sub-expression then the sub-expression is not evaluated again but is simply returned.

For example if $\{x\} = \{y\} \leftrightarrow x = y$ and $\{x\} = \{x\} \leftrightarrow \blacksquare$ are the only items and if they are listed to be used in that order then evaluating the theorem $\{A\} = \{A\} \vee A \neq A$ will cause the sub-expression $\{A\} = \{A\}$ to be replaced by $A = A$ resulting in $A = A \vee A \neq A$. All memory of the sub-expression $\{A\} = \{A\}$ is immediately lost upon its replacement by $A = A$ and thus the interpreter does not attempt to apply the second item to $\{A\} = \{A\}$.

Sometimes it will be the case that our interpreter will need to use items which are only valid in certain sub-domains of a given domain. For example, if we wish to use an item $\phi x \leftrightarrow \psi x$ (or $\phi x = \psi x$) where x is restricted to the sub-domain Πx, then we represent it by a conditional item:

$\Pi x \rightarrow (\phi x \leftrightarrow \psi x)$ (or $\Pi x \rightarrow \phi x = \psi x$)

The interpreter handles conditional items in the same way in which it handles non-conditional items until it has found a $\phi\theta$ which matches the sub-expression being evaluated. At this point on a conditional item, the interpreter tries to match each element in the conjunction Πx with some expression which it believes to be true. If such matches are found with substitution $\theta\sigma$ then $\psi\theta\sigma$ is returned. Otherwise the interpreter tries to apply another item as previously described.

The use of conditional items provides a general method of restricting the free variables of an item to a particular sub-domain. Its only disadvantage is that the amount of extra matching it forces the interpreter to perform. In order to minimize the amount of matching on the most common sub-domains we allow those sub-domains to be indicated by a particular style of variables.

For example, the automatic theorem prover described in this paper which is based on the set theory system described in Quine's [1969] book Set Theory and its Logic involves two domains: The larger domain, is the domain of abstracts, where an abstract of any propositional function fx with one free variable x is simply: $\{x: fx\}$, and the smaller domain is the domain of sets where a set is nothing more than an abstract a which exists; that is where $\exists x\ x = \alpha$ is a theorem of Quine's system.

Roman letters are used to indicate the sub-domain of sets whereas Greek letters are used to represent both sets and other abstracts.

An informed reader will recall that tables of re-write rules (rather simple items represented in ones mathematical language), and other items were used in many of Bledsoe's theorem provers [1971, 1972], and that Boyer and Moore [1973, 19743 used a symbolic LISP interpreter to order the application of various recursive definitions, re-write rules, and induction rules. More recently Aubin [1976] has also used a symbolic interpreter for much the same purpose. Our interpreter for mathematical expressions has also been used in other mathematical domains (Brown [1976]).

This theorem prover includes both logical and set-theoretic knowledge. We first describe the items of logical knowledge, and then the items of set theoretic knowledge.

2.1 Logical Knowledge

Our theorem prover has knowledge about twelve logical symbols which are listed below with their English translations:

$\wedge$	and	$\supset$	implies
$\vee$	or	$\leftrightarrow$	iff
$\sim$	not	$\exists$	there exists
$\blacksquare$	true	$\forall$	for all
$\square$	false	$=$	equal
$\rightarrow$	implies (This symbol is called a sequent arrow)		
and	and (This symbol is used to form an implicit conjunction of sequents)		

The sequent arrow may be defined as follows:
$\phi 1 \ldots \phi n \rightarrow \psi 1 \ldots \psi n =_{df} (\phi 1 \wedge \ldots \wedge \phi n) \supset (\psi 1 \vee \ldots \vee \psi n)$
where ϕi and ψj are sentences. Thus a sequent may be thought of as being a database of statements $\phi 1, \ldots, \phi n$ called assertions which occur before the sequent arrow, and statements $\psi 1, \ldots, \psi n$ called goals which occur after the sequent arrow.

The items of logical knowledge, which are all schemata because they involve ellipses (i.e. dots representing arbitrary expressions), are listed below:

Assertion schemata:

$\blacksquare \rightarrow$: $(\ldots \blacksquare \ldots \rightarrow \ldots) \leftrightarrow (\ldots \ldots \rightarrow \ldots)$

$\square \rightarrow$: $(\ldots \square \ldots \rightarrow \ldots) \leftrightarrow \blacksquare$

$\sim \rightarrow$: $(\ldots \sim X \ldots \rightarrow \ldots) \leftrightarrow (\ldots \ldots \rightarrow X \ldots)$

$\wedge \rightarrow$: $(\ldots X \wedge Y \ldots \rightarrow \ldots) \leftrightarrow (\ldots X, Y \ldots \rightarrow \ldots)$

$\vee \rightarrow$: $(\ldots X \vee Y \ldots \rightarrow \ldots) \leftrightarrow (\ldots X \ldots \rightarrow \ldots)$ and $(\ldots Y \ldots \rightarrow \ldots)$

$\supset \rightarrow$: $(\ldots X \supset Y \ldots \rightarrow \ldots) \leftrightarrow (\ldots \ldots \rightarrow X \ldots)$ and $(\ldots Y \ldots \rightarrow \ldots)$

$\leftrightarrow \rightarrow$: $(\ldots X \leftrightarrow Y \ldots \rightarrow \ldots) \leftrightarrow (\ldots X, Y, \rightarrow \ldots)$ and $(\ldots \rightarrow \ldots X, Y, \ldots)$

$\exists \rightarrow$: $(\ldots \exists x\ \phi x \ldots \rightarrow \ldots) \leftrightarrow (\ldots \phi f^*_1 \ldots {}^*_n \ldots \rightarrow \ldots)$
where f is a new skolem function and $*_1 \ldots *_n$ are all the unification variables* which occur in ϕx.

$= \rightarrow$: $(\Pi a \ldots \{ t=(f^*_1 \ldots {}^*_n) \} \ldots \Gamma a \rightarrow \phi a \ldots \psi a) \leftrightarrow (\Pi t \ldots \Gamma t \rightarrow \phi t \ldots \psi t)$
where f is a skolem function not occurring in t. This is our version of the Law of Leibnitz, for example:
$(A = B \cup C \rightarrow C \subseteq A) \leftrightarrow (\rightarrow C \subseteq B \cup C)$

Goal schemata:

$\rightarrow \blacksquare$: $(\ldots \rightarrow \ldots \blacksquare \ldots) \leftrightarrow \blacksquare$

$\rightarrow \square$: $(\ldots \rightarrow \ldots \square \ldots) \leftrightarrow (\ldots \rightarrow \ldots \ldots)$

$\rightarrow \sim$: $(\ldots \rightarrow \ldots \sim X \ldots) \leftrightarrow (\ldots X \rightarrow \ldots \ldots)$

$\rightarrow \wedge$: $(\ldots \rightarrow \ldots X \wedge Y \ldots) \leftrightarrow (\ldots \rightarrow \ldots X \ldots)$ and $(\ldots \rightarrow \ldots Y \ldots)$

$\rightarrow \vee$: $(\ldots \rightarrow \ldots X \vee Y \ldots) \leftrightarrow (\ldots \rightarrow \ldots X, Y \ldots)$

$\rightarrow \supset$: $(\ldots \rightarrow \ldots X \supset Y \ldots) \leftrightarrow (\ldots X \rightarrow \ldots Y \ldots)$

$\rightarrow \leftrightarrow$: $(\ldots \rightarrow \ldots X \leftrightarrow Y \ldots) \leftrightarrow (\ldots X \rightarrow \ldots Y \ldots)$ and $(\ldots Y \rightarrow \ldots X \ldots)$

$\rightarrow \forall$: $(\ldots \rightarrow \ldots \forall x\ \phi x \ldots) \leftrightarrow (\ldots \rightarrow \ldots \phi (f^*_1 \ldots {}^*_n) \ldots)$
where f is a new skolem function and $*_1 \ldots *_n$ are all the unification variables* which occur in ϕx.

Other logical schemata:

atom: $(\ldots X \ldots \rightarrow \ldots X \ldots) \leftrightarrow \blacksquare$

and : $(\ldots$ and $\blacksquare$ and $\ldots) \leftrightarrow (\ldots$ and $\ldots)$

Logical schemata used only at certain times:

Unify: $[(\ldots X_1 \ldots \rightarrow \ldots Y_1 \ldots)$ and $\ldots$ and $\ldots (\ldots X_n \ldots \rightarrow \ldots Y_n \ldots)] \leftrightarrow$
$[(\ldots X_i \ldots \rightarrow \ldots Y_i \ldots)$ and $\ldots$ and $\ldots (\ldots X_n \ldots \rightarrow \ldots Y_n \ldots)]$

where $1 < i \leq n$ and θ is any one of the substitutions which satisfy both the forcing restriction and the instantiation restriction. These two restrictions are described below.

*Unification variables are those variables produced by the $\forall \rightarrow$ and $\rightarrow \exists$ items and which may later be instantiated by the Unify item (i.e. the Unify Schemata).

The forcing restriction is the requirement that the substitution θ makes tautologous the greatest number of sequents starting with the first sequent and progressing towards the nth sequent. That is, the substitution θ makes the first i-1 sequents tautologous, and there is no substitution which makes the first i sequents tautologous. The rational behind this restriction is similar to Bledsoe's [1971, 1972] ideas on forcing, in that we force each sequent to make its contribution to θ and then throw it away. In the case that there actually is some substitution which will make all the sequents tautologous, without further unification variables being created (by the $\forall \rightarrow$ and $\rightarrow \exists$ items which will be described later in this section) this restriction leads to a complete proof procedure, in that θ will be one such substitution. As a minor point, if θ makes all the sequents tautologous, then this item is defined to return $\blacksquare$. An example instance of this item illustrating the forcing restriction is:
$[(P^* \rightarrow Pa, Pb)$ and $(Q^* \rightarrow Qb)$ and $R^* \rightarrow Ra)] \leftrightarrow (Rb \rightarrow Ra)$
We call any strategy which chooses the ordering of the sequents to which the unify item is applied, a forcing strategy, and will later discuss our forcing strategy.

The instantiation restriction is the requirement that no unification variable be instantiated to a term which already occurs in the list associated with the quantifier from which that unification variable was produced by an application of either the $\forall \rightarrow$ item or the $\rightarrow \exists$ item. The rational behind this restriction is that if a term t is already in the list associated with a quantifier such as $\forall$ in $(\ldots \forall x \phi x \ldots \rightarrow \ldots)$ then ϕt has already been produced in that sequent, and hence producing another ϕt by instantiating * to t could only be redundant.

The list of terms associated with a quantified formulae such as $(\forall x \phi x)$ essentially represent the instances $\phi t_1 \ldots \phi t_n$ of it that have been produced. This list is stored by appending it onto the end of a list containing the bound variable x as in: $(\forall (x\ t_1 \ldots\ t_n)\ \phi x)$. An example of the use of this restriction is the following instance of the Unify schemata:
$(Qt \rightarrow (\exists (x\ t\ *)(Px)), (Q^*)) \leftrightarrow (Qt \rightarrow (\exists (x\ t\ *)(Px)), Q^*)$

$\forall \rightarrow$: $(\ldots \{ \forall x \phi x \ / \ \forall (x \ldots) \phi x \} \ldots \rightarrow \ldots) \leftrightarrow (\ldots \rightarrow \ldots \{ \exists (x^*) \phi x \ / \ \exists (x \ldots *) \phi x \}, \phi^* \ldots)$
where * is a new unification variable and no more than one Unification variable occurs in $(x \ldots)$.

$\rightarrow \exists$: $(\ldots \rightarrow \ldots \{ \exists x \phi x \ / \ \exists (x \ldots) \phi x \} \ldots) \leftrightarrow (\ldots \rightarrow \ldots \{ \exists (x^*) \phi x \ / \ \exists (x^* \ldots) \phi x \}, \phi^* \ldots)$
where * is a new unification variable and no more than one Unification variable occurs in $(x \ldots)$.

In the last two items we have seen formulae of the form $\forall (x \ldots) \phi x$ and $\exists (X \ldots) \phi x$ which are not usually thought of as being well formed sentences of logic. Such formulas should be interpreted as respectively $\forall x \phi x$ and $\exists x \phi x$ which are well formed sentences of logic. The ... list is used merely to store certain pragmatic information used by the deductive system. This information is simply the list of instantiations of the Unification variables that were produced from this quantifier by applica-

tions of the ∀ → or → ∃ item. This pragmatic information is kept in order to be able to check the instantiation restriction, when using the Unify item (i.e. the Unify schemata), and in order to check the restriction on the ∀ → and → ∃ items that no more than one unification variable may already occur in the (x...) list. This last requirement is called the variable restriction. The rational behind this restriction is that there is no way in which three instances ϕ^*_1, ϕ^*_2, ϕ^*_3 of a formulae ϕx could interact so as to bind at least one of those unification variables, that could not be obtained by initially using only two instances and and then creating a third instance only after one of the first two unification variables is bound. An example of the use of this restriction is the fact that the → ∃ item could not be applied to the sequent: $(\rightarrow (\exists (x^*_1\ ^*_2)\phi)$

The logical items are not all used at the same time. In particular the ∀ →, → ∃ and unify items are used in a special way. Initially, the interpreter evaluates each sequent trying to apply the items in the following order:

(1) Non splitting assertion items:
¬→, ∧→, ∨→, ∃→, =→
(2) Non splitting goal items:
→¬, →∧, →∨, → ⊃, →∀
(A) (3) Non logical items
(4) The atom and "and" items
(5) Splitting assertion items:
⊃→, ↔ →
(6) Splitting goal items:
→↔, → ↔
(B) The Unify item
(C) The ∀ → and →∃ items once.

(A) The group (A) of items are applied first and are used in what is the normal evaluation process in a call-by-need manner. The sub-ordering from (1) to (6) reflect the fact that we try to delay splitting as long as possible, and within that restriction, try to work on assertions in a sequent, before working on goals. A reason for preferring to work on assertions first is that applications of the = → item can often simplify the problem a great deal. A simple reason for delaying splitting and applying the other items first is so that we don't have to apply those items twice after the split to each sequent separately. However, we find that the time gained by not having to apply items a second time, seems to be balanced by the time lost necessary to effect the delaying operation. A more substantial reason for delaying the splitting items is to handle certain subtle interactions between the →∀, ∃→, →∃, and ∀→ items necessitated by the incompleteness caused by the forcing restriction. These interactions will be described later in section 3 when we discuss our proof of Cantor's theorem.

(B) After the items of group (A) have been applied as many times as possible, the interpreter then tries to apply the unify item to the particular reordering of the conjunction of sequents, that is determined by the forcing strategy. The forcing strategy we have used is this: The conjunction of sequents are re-ordered such that those sequents which contain formulas beginning with a quantifier such that the quantifier

(1) satisfies the variable restriction
and
(2) has the shortest associated list of instantiations of any sequent in the conjunction.

are at the end, and hence will be unified last. The rational behind our forcing strategy is that each quantifier should get its fair chance to contribute instances towards proving the theorem. Thus, for example, if we forced out a sequent (by unifying it first) containing a quantifier which had not contributed any instances yet, and if that quantifier was actually needed in order to prove the theorems, then it is very probable that the substitutions made in unifying that sequent are irrelevant and in fact detrimental to solving the other sequents in the conjunction of sequents. An example of the effect of our forcing strategy is to re-order: $[(Qa\rightarrow\exists(x\ ^*_1)Q^*_1)$ and $(Qb\rightarrow\exists(x\ ^*_1\ s)Q^*_1)]$ as: $[(Qb\rightarrow\exists(x\ ^*_1\ c)Q^*_1)$ and $(Qa\rightarrow\exists(x\ ^*_1)Q^*_1)]$

(C) Next, if the application of the unification item does not result in ■, then the interpreter picks a sequent from the conjunction of sequents and tries to apply the ∀ → and ∃ → items once to each formula in the sequent which begins with a quantifier such that, the quantifier has the fewest number of terms in its associated list of any quantifier which satisfies the variable restriction in that sequent. We call this strategy for creating unification variables the creation strategy. The process of applying items then repeats itself starting at step (A). Note that a quantifier followed by only a variable, not a list, counts as having zero terms. Thus, for example, the creation strategy implies that only the ∀ → item would be applied to the sequent:

((∀x Px) → (∃(xa) Qx), (∃(y*) Qy))

The rational behind the creation strategy is that for each sequent it simply implements a breath first method of creating unification variables from the quantifiers in that sequent. This strategy is initially, therefore, the method that one unification variable will be created for each quantifier, before a second unification variable is ever created in that sequent. It differs from such a method in that if the theorem cannot be proven with a single unification variable for each quantifier, then it does not fail but continues to create more instances as called for by the creation strategy. This allows us, for example, to prove theorems which need multiple variables from their quantifiers such as Example 2 given in section 3.

Finally, if the application of the unification item of step (B) resulted in ■ then the processes terminates because the theorem has been proven.

An informed reader will recall that these propositional rules are used in Wang's algorithm in the LISP 1.5 manual [1965], and that Wang [1960] used the other rules restricted to the decidable case of where only skolem constants, but not skolem functions were necessary. The general idea of unification is due to Prawitz [1960] who used rules similar to all the rules given here except for = → and unify. His unification rule leads to a complete logic, ours does not. Robinson [1965] clarified Prawitz's unification algorithm by re-defining it, as we have done, in terms of skolem functions,

rather than in terms of ordering restrictions on Prawitz variables and skolem constants. More recently Bibel and Schreiber [1974] have implemented a complete sequent logic.

2.2 Set theoretic knowledge

Our theorem prover has knowledge about one primitive set theoretic symbol: ε which is interpreted as: is an element of, and knowledge about a large number of defined symbols. Some of these defined symbols along with their definitions and English translations are listed in table 1. In that table, the name of each expression usually indicates where that expression may be found in Quine's [1969] book: Set Theory and its Logic. For example, the definition of the symbol $\cap$ whose name is Q2P5 is expression number 5 in chapter 2 of that book. Note that a definition essentially defines the expression on the left side of the outer most $\leftrightarrow$ or = sign in terms of the expression on the right side.

With the sole exception of the axiom of extentionality, there are four kinds of items of set theoretic knowledge used by our theorem prover. They are: definitions, reduction lemmas, existence axioms, and existence lemmas. As previously mentioned some of the definitions that were used are listed in table 1. Some reduction lemmas that were used are listed in table 2. Some existence axioms that were used are given in table 3, and some existence lemmas that were used are given in table 4.

Definitions and Reduction lemmas are of the forms $\phi \leftrightarrow \psi$, $\phi = \psi$, $\Pi\rightarrow(\phi \leftrightarrow \psi)$, or $\Pi\rightarrow(\phi = \psi)$. Existence axioms and existence lemmas are of the form: $\phi\varepsilon V$ where V is the universal abstract.

The definitions and reduction lemmas are used only as items by the interpreter to evaluate sub-expressions, as described at the beginning of section 2.

For any given sub-expression that is being evaluated the interpreter tries to apply the definitions and reduction lemmas in the following order:

first: Q2P1
then: All reduction lemmas in reverse order from that listed in Table 2.
and finally: all other definitions.

For example, since the definition Q9P1 is tried only after the reduction lemma Q9P3, <xy> = <uv> evaluates to x = u $\wedge$ y = v using Q9P3 and not to ({x}={u} $\wedge$ {x y}={uv}) $\vee$ ({x}={uv} $\wedge$ {xy}={u}) using Q7P9 via definition Q9P1.

The philosophy behind restricting definitions and reduction lemmas to being used only as items is twofold: First, since the transformations made by the interpreter are truth value preserving assuming our axioms of set theory (and in the case of conditional items also assuming the particular sequent in which the sub-expression being evaluated occurs) the resulting sequent is a theorem of our set theory iff the original sequent was a theorem of our set theory. Second, at least in the case of definitions (because after all they are definitions and hence can be eliminated), we know that in a certain sense the complexity of the resulting expression is less. This fact, combined with the manner in which the sequent calculus itself reduces the complexity of its sequents, means that the resulting expressions are becoming less complex.

Furthermore, in the case of reduction lemmas such as $\phi \leftrightarrow \psi$, if the ϕ expression contains at least one symbol which was defined later than any symbol in ψ, then it is quite probable, but not at all certain, that the above termination property will hold. For example, all the non-conditional reduction lemmas in table 2 contain a later defined symbol in their ϕ expression, and thus the use of such items probably will reduce complexity. Thus our philosophy for using reduction lemmas as items is basically the same as for our use of definitions as items.

It should be noted that several people have considered the problem of proving that various sets of reduction lemmas preserve termination when used in this manner, notably Plotkin [1972], Lankford [1975] and Siekmann [1973].

It should not be thought that the termination property must hold in all cases for it to be useful to use reduction lemmas in this manner, for it may be the case that it fails only for pathelogical expressions which are of little interest in mathematics. An example of this is the fact that Q2P1 fails to reduce complexity when y is replaced by the following name in ZF of the null set: {x:xεx}.

The existence axioms and the existence lemmas are used to increase the range of applicability of the logical schema = → the contextual definition Q2P1 and of the reduction lemmas. Namely, whenever an expression of the form R ε V where R is some abstract is either assumed or proven, then for every combination of variables and abstracts X such that X ε V is now known, new versions of Q2P1 and the reduction lemmas are asserted.

Table 1: Definitions

Name	Definition	English translation
Q2P1:	$\forall y\ y\varepsilon\{x:\Gamma x\}\leftrightarrow\Gamma y$	the abstract of all x such that Γ
Q2P2:	$\alpha\subseteq\beta\leftrightarrow\forall x\ x\varepsilon\alpha\rightarrow x\varepsilon\beta$	is contained in
Q2P5:	$\alpha\cap\beta=\{x:\ x\varepsilon\alpha\wedge x\varepsilon\beta\}$	intersection
Q2P7:	$\alpha=\beta\leftrightarrow\forall x\ x\varepsilon\alpha\leftrightarrow x\varepsilon\beta$	equals
Q2P9:	$V = \{z:\ \blacksquare\}$	universe
Q5P5:	$\{x:\Gamma x\}\varepsilon\beta\leftrightarrow\exists y\ y=\{x:\Gamma x\}_{\wedge}y\varepsilon\beta$	the set {x:Γx} is in β
Q7P1a:	$\{\alpha\} = \{z:\ z=\alpha\}$	unit set
Q7P1B:	$\{\alpha\beta\} = \{z:\ z=\alpha\wedge z=\beta\}$	pair set
Q9P1:	$\langle\alpha\beta\rangle = \{\{\alpha\}\{\alpha\beta\}\}$	ordered pair
Q9P4:	$\{\langle xy\rangle:\Gamma xy\}=\{u:\exists x\exists y\ u=\langle xy\rangle_{\wedge}\Gamma xy\}$	abstract of ordered pairs
Q7P6:	$^{\circ}\alpha=\{\langle xy\rangle:\langle xy\rangle\varepsilon\alpha\}$	relational part
Q9P11:	$\alpha\times\beta=\{\langle xy\rangle:x\varepsilon\alpha_{\wedge}y\varepsilon\beta\}$	cartesian product
Q9P14:	$\alpha"\beta=\{x:\exists y\langle xy\rangle\varepsilon\alpha_{\wedge}y\varepsilon\beta\}$	image
Q1OP1:	$\text{Func}\ \alpha\leftrightarrow(\forall x\forall y\forall z\langle xz\rangle\varepsilon\alpha_{\wedge}\langle yz\rangle\varepsilon\alpha\rightarrow x=y)\wedge\alpha=^{\circ}\alpha$	is a function
Q1OP11:	$\alpha'\beta=\ ?y\langle y\beta\rangle\varepsilon\alpha$	apply
D1:	$P\alpha=\{u:\ u\subseteq\alpha\}$	powerset
Q11P1:	$\alpha\leq\beta\leftrightarrow\exists f\ \text{Func}f\wedge\ \alpha\subseteq f"\beta$	The cardinality of α is less than or equal to the cardinality of β
Q2OP3:	$\alpha < \beta\leftrightarrow\sim\beta\leq\alpha$	The cardinality of α is less than the cardinality of β

Table 2: Reduction Lemmas:

Q7P7: $\forall x \forall y\ \{x\} = \{y\} \leftrightarrow x=y$
Q7P8A: $\forall x\ \forall y\ \forall z\ \{xy\} = \{z\} \leftrightarrow x=z \wedge y=z$
Q7P8B: $\forall x\ \forall y\ \forall z\ \{z\} = \{xy\} \leftrightarrow z=x \wedge z=y$
Q7P9: $\forall x\ \forall y\ \forall u\ \forall v \{xy\}=\{uv\} \leftrightarrow (x=u \wedge y=v) \vee (x=v \wedge y=u)$
Q9P3: $\forall x\ \forall y\ \langle xy\rangle = \langle uv\rangle \leftrightarrow x=u \wedge y=v$
Q9P5: $\forall x\ \forall y\ \langle xy\rangle \in \{\langle uv\rangle : \phi uv\} \leftrightarrow \phi xy$
CRL1: Func $f \rightarrow (\langle wy\rangle \in f \leftrightarrow w=f'y)$

Table 3: Existence axioms and axiom of extentionality

Q7P10A: $\emptyset \in V$
Q7P10B: $\forall x\ \forall y\ \{xy\} \in V$
Q4P1: $\forall x\ \forall y\ \forall z\ (x=y \wedge x \in z. \supset y \in z)$

Table 4: Existence Lemmas:

Q7P12: $\{\alpha\} \in V$
Q7P13: $\{\alpha\beta\} \in V$
E2: $\langle\alpha\beta\rangle \in V$

A simpler method of implementing the addition of these new reduction rules, is to agree that the initial universally quantified variables of a definition or a reduction lemma are allowed to match, not just variables, but also any abstract R such that R ε V is known.

There is one further way in which the existence axioms are used, and this has to do with the fact that they are axioms. For this reason, if the interpreter gets "stuck" while trying to solve a sequent (that is, if no items can be applied to that sequent) in proving a theorem, it will in privitive notation, i.e. with the defined symbols eliminated, add each axiom as an additional hypothesis in that sequent.

Again an informed reader will recall Bledsoe's set theory theorem prover [1971].

3. Example

We give below a protocol produced by our theorem prover while proving Quine's version of Cantor's Theorem. We have listed on the right the name of the item which was applied to each sequent in the protocol . Any name which is starred: '*' is the name of a lemma. A list of names indicates the use of a reduction lemma corresponding to the item named by the first name in the list, and created by the items named by the remaining names in the list.

The protocol is presented as a tree of sequents, grown top down, starting with the sequent containing only the theorem the system is trying to prove. On each line of the proof the item name following the colon indicates that that item was applied to the sequent on that line, producing the sequent on the line immediately below that one. In the case of a logical split item ($\rightarrow\wedge$, $\rightarrow\leftrightarrow$, $\forall\rightarrow$, $\supset\rightarrow$, $\leftrightarrow\rightarrow$) two sequents will be produced, and they are indicated by drawn arrows.

The proof is segmented by the scope of attempted applications of the Unify items. The boundaries of each segment are indicated by stating where each segment starts. That is by stating:

$:\theta_k$ starts here

where k refers to segment k. The system attempts to apply the Unify item once to a sequence of all endsequents of each segment. Thus, every sequent in the segment marked as having the Unify item applied to it are jointly the conjunction of sequents to which the Unify schema is actually applied.

Example 1

Q28P17 Cantor's Theorem: The cardinality of any abstract is less than the cardinality of its power abstract, provided that $\alpha \cap \{y:\sim y \in w'y\}$ is a set.

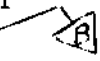
In order to present this protocol the following abbreviations are made:

① = df $(\forall w\ \alpha \cap \{y:\sim y \in w'y\} \in V)$
② = df $(\forall x\ x \in P\alpha \rightarrow x \in f''\alpha)$
③ = df $(\forall x\ x \in (a^*_2) \leftrightarrow x \in \alpha \cap \{y:\sim y \in {}^*_2'y\})$
④ = df $(\forall x\ x \in (a^*_2) \leftrightarrow x \in f'(b^*_2))$

$\rightarrow (\forall w.\alpha \cap \{y:\sim y \in w'y\} \in V) \supset \alpha < P\alpha$:→⊃
$(\forall w.\alpha \cap \{y:\sim y \in w'y\} \in V) \rightarrow \alpha < P\alpha$:Q20P3
① $\rightarrow \sim P\alpha \leq \alpha$:→∼
① , $P\alpha \leq \alpha \rightarrow$:Q11P1
① , $\exists x$ Func $x \wedge P\alpha \subseteq x''\alpha \rightarrow$:∃ →
① , Func $f \wedge P x \in f''\alpha \rightarrow$:∧ →
① , Func f, $P\alpha \subseteq f''\alpha \rightarrow$:Q2P2

:01 starts here

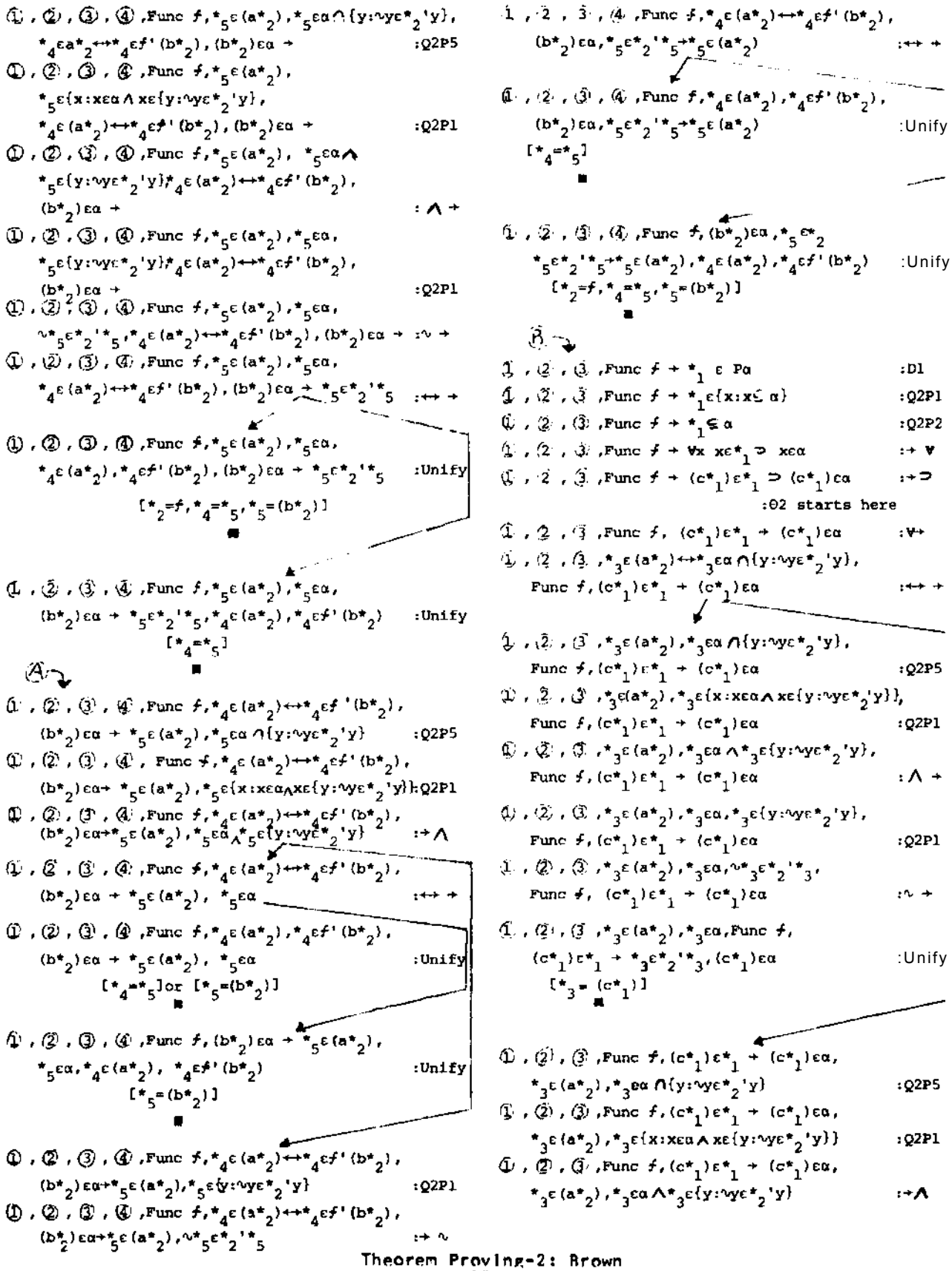
① , Func f, $(\forall x\ x \in P\alpha \supset x \in f''\alpha) \rightarrow$:∀ →
① , Func f, ${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha$, ② → :∀ →
① , ② , Func f, $\alpha \cap \{y:\sim y \in {}^*_2'y\} \in V$,
${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha \rightarrow$:Q5P5
① , ② , Func f, $\exists x\ x=\alpha \cap \{y:\sim y \in {}^*_2'y\} \wedge x \in V$,
${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha \rightarrow$:∃ →
① , ② , Func f, $(a^*_2)=\alpha \cap \{y:\sim y \in {}^*_2'y\} \wedge (a^*_2) \in V$,
${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha \rightarrow$:∧ →
① , ② , Func f, $(a^*_2)=\alpha \cap \{y:\sim y \in {}^*_2'y\}$,
$(a^*_2) \in V$, ${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha \rightarrow$:Q2P9
① , ② , Func f, $(a^*_2)=\alpha \cap \{y:\sim y \in {}^*_2'y\}$,
$(a^*_2) \in \{y:\blacksquare\}$, ${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha \rightarrow$:Q2P1
① , ② , Func f, $(a^*_2)=\alpha \cap \{y:\sim y \in {}^*_2'y\}$,
$\blacksquare$, ${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha \rightarrow$:Q2P7
① , ② , Func f, $(\forall x\ x \in (a^*_2) \leftrightarrow x \in \alpha \cap \{y:\sim y \in {}^*_2'y\})$,
$\blacksquare$, ${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha \rightarrow$:■ →
① , ② , Func f, ③ , ${}^*_1 \in P\alpha \supset {}^*_1 \in f''\alpha \rightarrow$:⊃ →

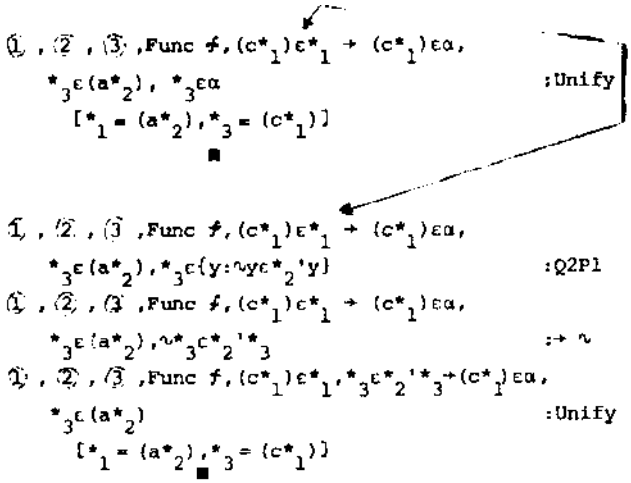


① , ② , ③ , Func f, ${}^*_1 \in f''\alpha \rightarrow$:Q9P14
① , ② , ③ , Func f, ${}^*_1 \in \{x:\exists y \langle xy\rangle \in f \wedge y \in \alpha\} \rightarrow$:Q2P1
① , ② , ③ , Func f, $\exists y\ \langle {}^*_1 y\rangle \in f \wedge y \in \alpha \rightarrow$:∃ →
① , ② , ③ , Func f, $\langle {}^*_1 (b^*_1)\rangle \in f \wedge (b^*_1) \in \alpha \rightarrow$:∧ →
① , ② , ③ , Func f, $\langle {}^*_1 (b^*_1)\rangle \in f$, $(b^*_1) \in \alpha \rightarrow$:CRL1*
① , ② , ③ , Func f, ${}^*_1 = f'(b^*_1)$, $(b^*_1) \in \alpha \rightarrow$:Q2P7
① , ② , ③ , Func f, $(\forall x\ x \in {}^*_1 \leftrightarrow x \in f'(b^*_1))$, :propogation from θ2
$(b^*_1) \in \alpha \rightarrow$

:03 starts here

① , ② , ③ , Func f, $(\forall x\ x \in (a^*_2) \leftrightarrow x \in f'(b^*_2))$,
$(b^*_2) \in \alpha \rightarrow$:∀ →
① , ② , ③ , Func f, ④ , ${}^*_4 \in (a^*_2) \leftrightarrow {}^*_4 \in f'(b^*_2)$,
$(b^*_2) \in \alpha \rightarrow$:∀ →
① , ② , ③ , ④ , Func f, ${}^*_5 \in (a^*_2) \leftrightarrow {}^*_5 \in \alpha \cap \{y:\sim y \in {}^*_2'y\}$, ${}^*_4 \in (a^*_2) \leftrightarrow {}^*_4 \in f'(b^*_2)$,
$(b^*_2) \in \alpha \rightarrow$:↔ →


①, ②, ③, ④, Func f, *5ε(a*2), *5εα∩{y:∿yε*2'y},
*4εa*2↔*4εf'(b*2), (b*2)εα → :Q2P5
①, ②, ③, ④, Func f, *5ε(a*2),
*5ε{x:xεα∧xε{y:∿yε*2'y},
*4ε(a*2)↔*4εf'(b*2), (b*2)εα → :Q2P1
①, ②, ③, ④, Func f, *5ε(a*2), *5εα∧
*5ε{y:∿yε*2'y}, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα → :∧ →
①, ②, ③, ④, Func f, *5ε(a*2), *5εα,
*5ε{y:∿yε*2'y}, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα → :Q2P1
①, ②, ③, ④, Func f, *5ε(a*2), *5εα,
∿*5ε*2'*5, *4ε(a*2)↔*4εf'(b*2), (b*2)εα → :∿ →
①, ②, ③, ④, Func f, *5ε(a*2), *5εα,
*4ε(a*2)↔*4εf'(b*2), (b*2)εα → *5ε*2'*5 :↔ →
①, ②, ③, ④, Func f, *5ε(a*2), *5εα,
*4ε(a*2), *4εf'(b*2), (b*2)εα → *5ε*2'*5 :Unify
[*2=f, *4=*5, *5=(b*2)]
■
①, ②, ③, ④, Func f, *5ε(a*2), *5εα,
(b*2)εα → *5ε*2'*5, *4ε(a*2), *4εf'(b*2) :Unify
[*4=*5]
■
A
①, ②, ③, ④, Func f, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα → *5ε(a*2), *5εα∩{y:∿yε*2'y} :Q2P5
①, ②, ③, ④, Func f, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα → *5ε(a*2), *5ε{x:xεα∧xε{y:∿yε*2'y}} :Q2P1
①, ②, ③, ④, Func f, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα → *5ε(a*2), *5εα∧*5ε{y:∿yε*2'y} :→∧
①, ②, ③, ④, Func f, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα → *5ε(a*2), *5εα :↔ →
①, ②, ③, ④, Func f, *4ε(a*2), *4εf'(b*2),
(b*2)εα → *5ε(a*2), *5εα :Unify
[*4=*5]or [*5=(b*2)]
■
①, ②, ③, ④, Func f, (b*2)εα → *5ε(a*2),
*5εα, *4ε(a*2), *4εf'(b*2) :Unify
[*5=(b*2)]
■
①, ②, ③, ④, Func f, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα → *5ε(a*2), *5ε{y:∿yε*2'y} :Q2P1
①, ②, ③, ④, Func f, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα → *5ε(a*2), ∿*5ε*2'*5 :→ ∿
i, 2, 3, 4, Func f, *4ε(a*2)↔*4εf'(b*2),
(b*2)εα, *5ε*2'*5 → *5ε(a*2) :↔ →
①, ②, ③, ④, Func f, *4ε(a*2), *4εf'(b*2),
(b*2)εα, *5ε*2'*5 → *5ε(a*2) :Unify
[*4=*5]
■
①, ②, ③, ④, Func f, (b*2)εα, *5ε*2
*5ε*2'*5 → *5ε(a*2), *4ε(a*2), *4εf'(b*2) :Unify
[*2=f, *4=*5, *5=(b*2)]
■
B
1, 2, 3, Func f → *1 ε Pα :D1
1, 2, 3, Func f → *1ε{x:x⊆ α} :Q2P1
1, 2, 3, Func f → *1⊆ α :Q2P2
1, 2, 3, Func f → ∀x xε*1 ⊃ xεα :→ ∀
1, 2, 3, Func f → (c*1)ε*1 ⊃ (c*1)εα :→⊃
:Θ2 starts here
1, 2, 3, Func f, (c*1)ε*1 → (c*1)εα :∀→
1, 2, 3, *3ε(a*2)↔*3εα∩{y:∿yε*2'y},
Func f, (c*1)ε*1 → (c*1)εα :↔ →
1, 2, 3, *3ε(a*2), *3εα∩{y:∿yε*2'y},
Func f, (c*1)ε*1 → (c*1)εα :Q2P5
1, 2, 3, *3ε(a*2), *3ε{x:xεα∧xε{y:∿yε*2'y}},
Func f, (c*1)ε*1 → (c*1)εα :Q2P1
1, 2, 3, *3ε(a*2), *3εα∧*3ε{y:∿yε*2'y},
Func f, (c*1)ε*1 → (c*1)εα :∧ →
1, 2, 3, *3ε(a*2), *3εα, *3ε{y:∿yε*2'y},
Func f, (c*1)ε*1 → (c*1)εα :Q2P1
1, 2, 3, *3ε(a*2), *3εα, ∿*3ε*2'*3,
Func f, (c*1)ε*1 → (c*1)εα :∿ →
1, 2, 3, *3ε(a*2), *3εα, Func f,
(c*1)ε*1 → *3ε*2'*3, (c*1)εα :Unify
[*3= (c*1)]
■
1, 2, 3, Func f, (c*1)ε*1 → (c*1)εα,
*3ε(a*2), *3εα∩{y:∿yε*2'y} :Q2P5
1, 2, 3, Func f, (c*1)ε*1 → (c*1)εα,
*3ε(a*2), *3ε{x:xεα∧xε{y:∿yε*2'y}} :Q2P1
1, 2, 3, Func f, (c*1)ε*1 → (c*1)εα,
*3ε(a*2), *3εα∧*3ε{y:∿yε*2'y} :→∧



Time = 1,494 milliseconds or approximately: 1½ seconds on a DEC10 computer with a KA10 CPU

Note that two distinct instantiations of formulae ③ have been used:

Variable	Formulae	Instantiation	Branch of Proof
$*_1$	②	a	$\theta_1(\theta 2)$
$*_2$	①	f	$\theta_1(\theta_3)$
$*_3$	③	c	θ_2
$*_4$	④	b	θ_3
$*_5$	③	b	θ_3

A theorem prover which:
(1) immediately skolemized it's formulae and
(2) allowed no more than one instance of each variable
would not be able to prove this theorem except by a very lucky accident.

The basic problem of this proof is at the start of θ1: and is to decide which of the following two formulas to work on first:

(1) $\alpha \cap \{y:\sim y\varepsilon *_2 'y\} \varepsilon V$

(2) $*_1 \varepsilon P\alpha \supset *_1 \varepsilon f$ " α

If one begins by splitting on (2) then on each branch of the search space formulae (1) will reduce to

$$\exists x\ x=\alpha \cap \{y:\sim y\varepsilon *_2 'y\} \wedge x\varepsilon V$$

which produces on each branch a distinct* skolem function $(a*_2)$, $(a'*_2)$, and this will not lead to a proof. Thus one must begin working on the first formulae. However, this first formulae in a few steps reduced to ③ . If x in ③ is immediately replaced by a single Unification variable, then again no proof will be obtained, for we have seen that from our proof that two variables $*_3$ and $*_5$ bound to distinct skolem functions c and b were needed.

Thus what one must do is to start on formulae (1), then switch to formulae (2) and split, and then come back on each branch and instantiate (1).

*It is not obvious that two quantifiers on different branches of the search space could be replaced by the same gensymed skolem function, in a system with unification variables because instantiating them can cause these skolem functions to appear in the same sequent.

4. Conclusion

We have described an automatic theorem prover for the domain of elementary set theory, and have presented some protocols of its behaviour when proving some theorems. We wish to stress that the program is fast and compact. For example,we have seen that it took less than 2 seconds to obtain a proof of Cantor's Theorem. It should be noted that our theorem prover is implemented in compiled UCI LISP (Bobrow *1*19733) and was executed on a DEC10 with a KA10 CPU.

Acknowledgements

I thank Bernard Meltzer, Robert Kowalski and Raymond Aubin for many enlightening discussions. This research was supported by a scholarship awarded to the author by the University of Edinburgh, and is currently supported by a grant BRG 94431 from the Science Research Council.

References

Aubin, R. "Mechanizing Structural Induction",PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1976.

Bibel, W. and Schreiber, J. "Proof Search in a Gentzen Like System of First Order Logic", International Computing Symposium, 1974, North Holland Publishing Company.

Bledsoe, W.W. "Splitting and Reduction Heuristics in Automatic Theorem Proving", Artificial Intelligence, Vol. 2, No. 1., Spring 1971.

Bledsoe, W.W., Boyer, R.S., Henneman, W.H. "Computer Proofs of Limit Theorems", Artificial Intelligence, Vol. 3, 1972.

Bobrow, R.J. et al. UCI LISP Manual, Technical Report 21, Dept. of Computer Science, University of Cal., Irvine, 1973.

Boyer, R.S., Moore, J S. "Proving Theorems about LISP functions", IJCAI3 Proceedings 1973.

Brown, F.M. "Doing Arithmetic without Diagrams", DAI Research Report No. 16A. Also to appear in Journal of Artificial Intelligence, 1976.

Lankford, D.S. "Canonical Algebraic Simplification", Maths Department, Memo AIP-25, The University of Texas at Austin, May 1975.

McCarthy, J., et al. LISP 1.5 Programmers Manual, MIT Press, Cambridge, Mass., 1965.

Moore, J S. "Computational Logic: Structure Sharing and Proof of Program Properties, Part II", DCL Memo No. 68, 1974.

Plotkin, G. "Building in Equational Theories" Machine Intelligence 7, 1972.

Prawitz, D. "An Improved Proof Procedure", Theoria, Vol.26, pp 102-139, 1960.

Quine, W.V.O. Set Theory and its Logic, revised edition 1969, Oxford University Press,Library of Congress Catalogue Card No.68-14271.

Robinson,J.A. "A Machine Oriented Logic based on the Resolution Principle",J.ACM,Vol.12,p.23-41, 1965.

Siekmann,J. "A Modification of the Unification Algorithm in Automatic Theorem Proving",Masters thesis,Univ.of Essex,1972.

Wang,H. "Towards Mechanical Mathematics",IBM Jnl. of Research and Development,Vol.4,pp 2-22,1960.