TOWARDS THE AUTOMATION OF MATHEMATICAL REASONING

Frank Malloy Brown

PhD

University of Edinburgh

1977

This thesis has been composed entirely by myself and the research described herein is entirely my own.

Frank Malloy Brown

I give my permission for this thesis to contain previously published material by the author.

Professor Bernard Meltzer

Supervisor

Note

Early versions of parts of this thesis have already been published under the following titles.

Chapter	Title
III abbreviated	"A Deductive System for Elementary Arithmetic", Proc. 2nd AISB Summer Conference on Artificial Intelligence, Edinburgh, 1976.
III in full	"Doing Arithmetic without Diagrams", DAI Research Report 16 and 16A, 1976. Also: Journal of Artificial Intelligence, Vol. 8, 1977.
IV abbreviated	"The Role of Extensible Deductive Systems in Mathematical Reasoning", Proc. 2nd AISB Summer Conference on Artificial Intelligence, Edinburgh, 1976.
V abbreviated	"A Theorem Prover for Elementary Set Theory", DAI Research Report 34a,1977. Also to appear in the Proc. of the 5th International Joint Conference on Artificial Intelligence (IJCAI5), 1977.
V in full	"Towards the Automation of Set Theory and its Logic", DAI Research Report 34, 1977. Also submitted to the Journal of Artificial Intelligence.

Abstract

We first argue that the improved performance of modern theorem provers is due basically to the use of domain dependent deductive knowledge possessing certain simple properties. We exemplify this thesis by discussing various theorem provers for propositional logic and then for elementary arithmetic. In particular we describe a domain dependent theorem prover for elementary arithmetic which we have implemented.

Secondly, we then argue that theorem provers based on domain dependent knowledge, such as our arithmetic theorem prover, are not the kinds of deductive systems that are needed as a component of a mathematical reasoning system. The reason for this being that such systems are not extensible in that they cannot assimilate and use new deductive knowledge produced by a mathematical reasoning system. Finally, we describe a prototype deductive system, for the domain of elementary set theory, which is extensibe in this sense.

Contents

I. Introduction

This report is concerned with the kinds of deductive systems that are needed as a component of an automated mathematical reasoning system. It is not particularly concerned with the development of deductive mechanisms as a goal in itself, and in this distinction lies our thesis.

We begin in Chapter II by describing and comparing two automatic theorem provers for propositional logic. We will find that one of these theorem provers, namely the one called Wang's Algorithm is a far better proof procedure than is the other one. The reason for this being,that Wang's Algorithm does not prove theorems by using the given axioms and inference rules of propositional logic as does the other theorem prover, but instead proves theorems by using a special set of rules which,when applied, tend in an intuitive sense to result in simpler and simpler subgoals to prove. In other words, the superiority of Wang's Algorithm lies in its use of special domain dependent knowledge; that is in its use of specific rules with certain properties, applied to a specific mathematical domain which is in this case the propositional logic.

Next, we suggest that this is a very general phenomena, and that for just about any mathematical domain, one can in practice find a special set of rules of such a nature as to result in domain dependent theorem prover which would be a far better theorem prover than one based on using only the original and generally accepted axioms and inference rules of that domain. In Chapter III we will exemplify the truth of this last remark by describing a domain de-

pendent theorem prover for the domain of elementary arithmetic which we have implemented. This theorem prover which uses about 50 special arithmetic rules rather than just the Peano axioms can easily be seen to be quite efficient. As a side issue we will also compare this theorem prover to another domain dependent theorem prover for arithmetic, which, however, is based on the use of special rules which are not quite as well chosen as are the rules of our system. This will provide some evidence for our assertion in Chapter II, that the special rules of an efficient domain dependent theorem prover, should possess certain specific properties.

Then in Chapter IV we present our major thesis: that domain dependent deductive knowledge systems such as the one described in Chapter III are not the kind of deductive system that is needed in order to automate mathematical reasoning. Essentially we argue that deductive knowledge systems for mathematical reasoning must, in a realistic way, be capable of being extended by the acquisition of new deductive knowledge such as new rules, or new strategies for using rules.

We call a deductive system with this capability an extensible deductive knowledge system and in Chapter V we describe a prototype extensible deductive knowledge system for the domain of elementary set theory which we have implemented. Specifically this system proves theorems in the set theory described in W.V.O.Quine's book Set Theory and its Logic [1].

Note, however, that an extensible deductive system does not in itself include the acquisition procedures which create the new de-

ductive knowledge, for these acquisition procedures fall in the area of inductive reasoning. We will, however, call any system which includes both an extensible deductive system and an inductive reasoning system, a mathematical reasoning system.

Finally, in Chapter VI we draw a few conclusions.

The reader should note that this is a very modular thesis in that each chapter may be read independently of any other chapter. Thus, for example, if the busy reader only wishes to know why extensible deductive knowledge systems are needed for mathematical reasoning, he need only read Chapter IV. Or, for example, if he is only interested in how one might automate deductive reasoning in elementary set theory, he need only read Chapter V.

II The Rise of Deductive Knowledge Systems

1. Introduction

The purpose of this chapter is to explain how the kinds of automatic theorem provers for mathematics*, that were prevalent in 1974, and indeed prevalent even now, came to have been produced. In opposition to such an explanation, if one were to make only a superficial historical survey of the research in this field one would probably obtain the impression that this field consists of many fundamentally different approaches and techniques. In particular, one would find a bewildering variety of approaches to proving theorems, and an even more bewildering variety of propaganda supporting those various approaches.

However, since an examination of the actual results that have been obtained will certainly convince one that a great deal of progress in this field has,in fact, been made, though not necessarily for the reasons that have been given, the question remains as to the reasons for this progress.

Ignoring relatively minor considerations such as the improvement in computer hardware, our view is that the improved performance of automatic theorem provers during the last twenty years is directly attributable to the application of a single basic idea to larger and larger areas of mathematics. This basic idea is simply that one should build into the automatic theorem prover as much deductive knowledge, in terms of rules and how to use them, as is possible

---

* This thesis is not concerned with the use of automatic theorem provers as interpreters of logical programming languages.

about the mathematical domain under consideration. To emphasize the importance of this idea we will call such theorem provers: domain dependent deductive knowledge systems. Domain dependent deductive systems are, of course, no longer general purpose theorem provers which can accept axioms from any domain, for their power depends crucially on the embedding of special rules of that particular mathematical domain.

In the remainder of this chapter we will exemplify this basic idea by describing and comparing a domain dependent deductive system for the propositional logic, called Wang's Algorithm [10] with a general purpose theorem proving method which simply applies the inference rules of that domain in reverse to the theorem one is trying to prove, producing new subgoals to prove, and continuing in this fashion until all the subgoals are themselves axioms of that domain. A particular theorem prover for the propositional logic which is based on this general purpose theorem proving method is the Logic Theorist [25]. Before describing these two theorem provers, we first, however, describe the propositional logic itself.

2. Propositional Logic

Propositional logic is a systemitized version of a very small part of the language we speak. Its subject matter is the formation of complex sentences from elementary sentences and the methods of determining the logical truth or non truth of such sentences.

The syntax of Propositional Logic consists of the following elements:

(1) Elementary sentences which express propositions such as:

(2 plus 2 equals 4), (block1 is on the table)

(2) Propositional variables such as:

P, Q, R

and (3) the logical connectives which are written below followed by their English translations:

$\wedge$	and
$\vee$	or
$\supset$	implies
$\leftrightarrow$	iff (if and only if)
$\sim$	not
$\blacksquare$	true
$\square$	false

Using these elementary sentences, propositional variables, and logical connectives we may form complex sentences such as:

(2 plus 2 equals 4) $\wedge$ (block1 is on the table)

P $\supset$ P

P $\vee$ ((block1 is on the table) $\wedge$ Q)

Having now exemplified what are the sentences of this language, we can now give a criteria for determining which sentences are logically true, and which are not. First we note that some sentences such as:

$\blacksquare$, P $\supset$ P

are logically true, that some sentences such as:

$\square$, P $\wedge$ ($\sim$ P)

are logically false, and that some sentences such as:

(block1 is on the table)

are neither logically true nor logically false.

In order to define which sentences are logically true we will first simplify the problem by showing that if we can define logical truths for those sentences containing no logical connectives other than $\supset$ and $\sim$, then we can define logical truth for all sentences. We do this by simply defining all the other logical connectives in terms of $\supset$ and $\sim$ as follows*:

$P \vee Q$ =df $(\sim P) \supset Q$

$P \wedge Q$ =df $\sim(P \supset \sim Q)$

$P \leftrightarrow Q$ =df $(P \supset Q) \wedge (Q \supset P)$

$\Box$ =df $P \vee \sim P$

$\Diamond$ =df $\sim \Box$

Given this simplification, we can then give a criteria to determine which sentences are logically true, and which are not, namely**: the sentences which are logically true are precisely those which may be deduced from the axioms and inference rules of propositional logic. The three axioms of propositional logic are:

A1: $P \supset (Q \supset P)$

A2: $(P \supset (Q \supset R)) \supset ((P \supset Q) \supset (P \supset R))$

A3: $(\sim(\sim P)) \supset P$

The two inference rules of propositional logic are:

---

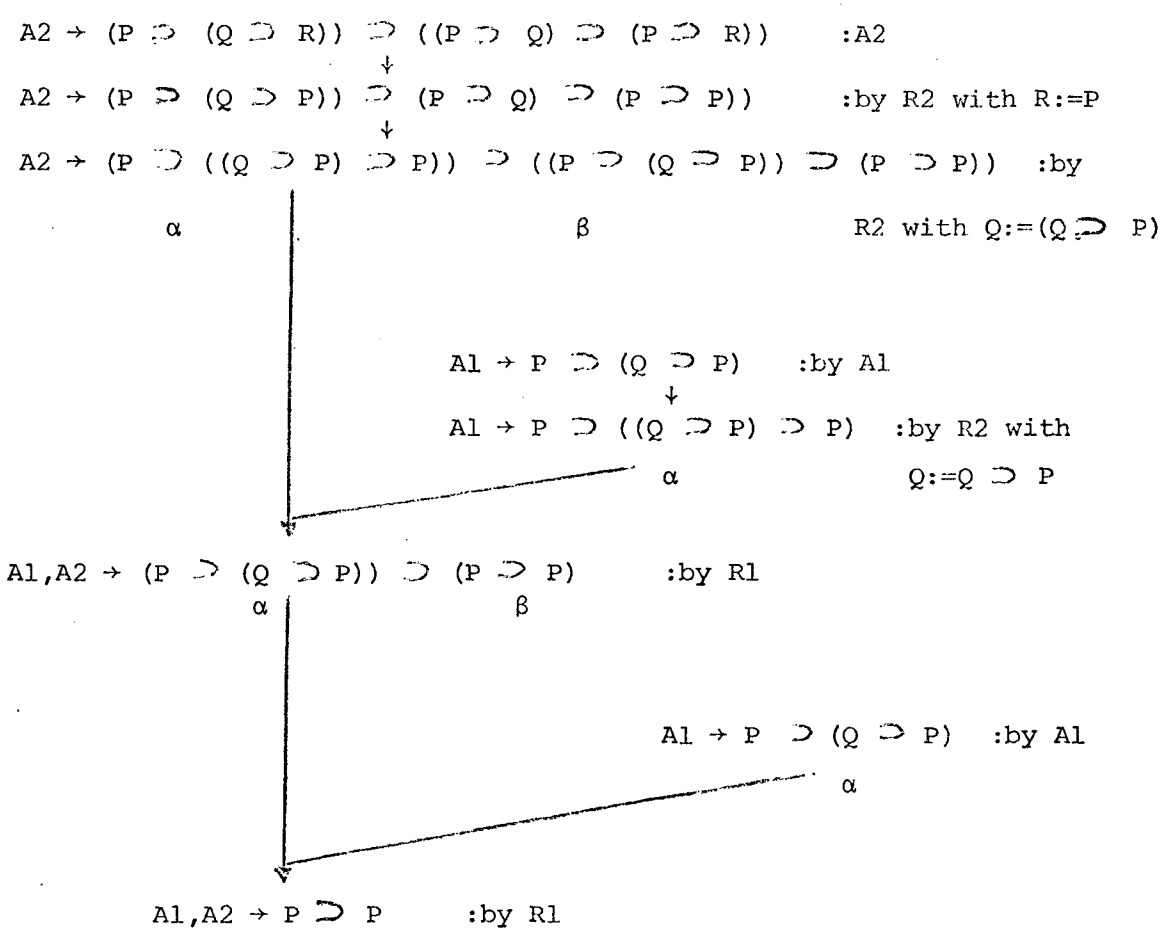
* =df is an informal symbol such that for all expressions α and for all expressions β, α =df β means that α is defined to be β.

** In view of the completeness theorem for propositional logic we know that logical truth and theoremhood are equivalent concepts.

R1: from a sentence α and a sentence of the form $(\alpha \supset \beta)$ deduce β.

R2: from a sentence α deduce the sentence obtained from α by substituting any sentence β for all occurrences in α of any particular propositional variable.

For example, using these axioms and inference rules we could determine that $P \supset P$ is logically true by the following deduction:



In this proof the → sign in $\beta_1 \ldots \beta_n \rightarrow \alpha$ is just an informal symbol which we use to indicate that the sentence α is deducible from the axioms $\beta_1 \ldots \beta_n$. Note that the sentences α at the tip of this tree

are all axioms and that the sentence at the root of the tree is the theorem we are trying to prove. Note also, that the sentence at the root of any subtree is obtained from the sentences at the nodes just above that root node.

3. The Logic Theorist

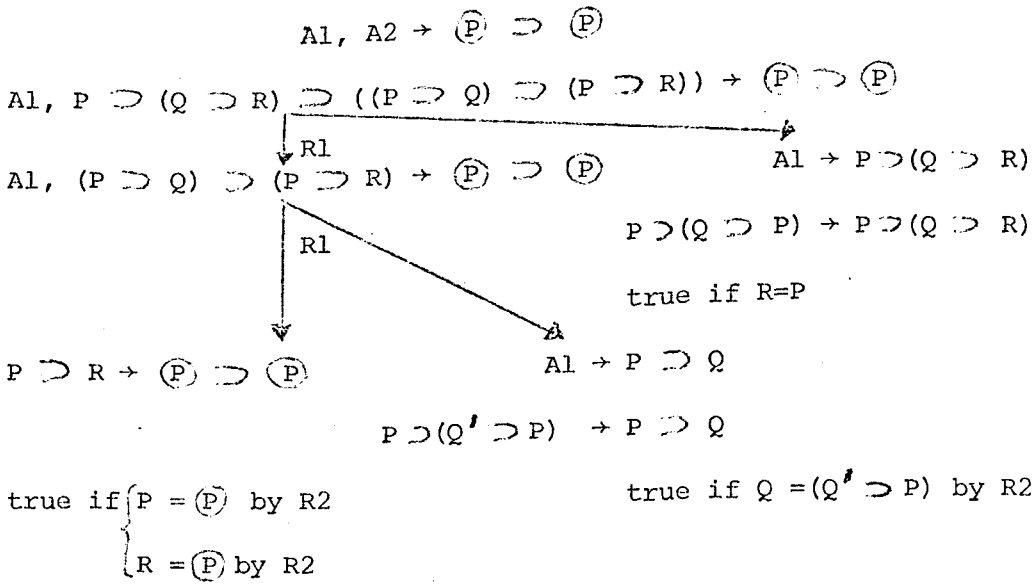
Having now defined what sentences are logically true we are left with the problem of developing a feasible method of determining whether any given sentence is logically true. One method of determining this would be to apply the inference rules in all possible ways to the axioms and to the resulting sentences obtained by applying those rules, and periodically checking to see if the theorem is one of those results. However, this method would appear to be rather infeasible as it makes no real use* of the structure of the sentence to be proven in directing the search for its proof. Probably for this reason the first automatic theorem prover for the propositional logic, the Logic Theorist [25] was designed so as to search for a proof by starting from the theorem to be proven and applying the inference rules R1, R2 backwards, producing new subgoals as theorems to be proven until the only remaining subgoals were themselves axioms.

Thus, for example, in order to prove $P \supset P$ the Logic Theorist

---

* Of course one could make use of the structure of the theorem in a weak heuristic sense by choosing to first apply inference rules to those axioms and results which are the most syntactically similar to the theorem being proven.

would produce a trace similar to the following one:



Note that since we are trying to prove P ⊃ P for all P, the variable P must not itself be instantiated by R2. For this reason we have circled this P as Ⓟ, so as to indicate that it must be treated as being a constant. Note also, that one occurrence of Q is indicated as Q'. This is due to the fact that the variables of distinct axioms are conceptually distinct.

Note also that we have only given that part of the trace of the search for this proof of P ⊃ P, that was actually relevant to it. For example, some of the other possible branches of the trace are where R1 is first applied not to the main implication sign in A2 but to the main implication sign in A1 or A3.

The basic problem of searching for a proof in this manner is that although the Logic Theorist uses the structure of the theorems in

searching for their proof, it does not really use the structure of the axioms themselves except in the weak sense of choosing to apply inference rules backwards first to those goals which are most syntactically similar to some axiom.*

4. Wang's Algorithm

The first automatic theorem prover to essentially make use of the structure of both the axioms and the theorem was one implemented by Wang in about 1960 [10]. This system which is based on applying the rules of Gentzen's [27] LK sequent calculus backwards, involves letting $\rightarrow$ which had been an informal symbol now be a formal symbol of the calculus such that:

$$\phi_1 \cdots \phi_n \rightarrow \psi_1 \cdots \psi_m$$

is interpreted to mean

$$\phi_1 \wedge \cdots \wedge \phi_n \supset \psi_1 \vee \cdots \vee \psi_m$$

Thus $\rightarrow$ is sort of an n+m ary implication symbol.

The axioms and rules of this system are not those previously given, but instead, those axioms and inference rules are deleted, and

---

* Since there are several axioms and only one theorem to be proven it is easy to see that more sentences will be syntactically similar to at least one axiom, than to that theorem. For this reason the search space obtained when searching forwards from the axioms will probably be smaller than the search space obtained when searching backwards. Thus it is probably better to search forwards as did Siklossy and Rich [26]. Note, however, that the structure of the theorem is not really used, and hence this system, also suffers the same basic defect as the Logic Theorist.

the following five new inference rules are added:

$\supset\rightarrow$: $(..P \supset Q..\rightarrow..)$ iff $(..Q..\rightarrow..)$ and $(.. .. \rightarrow P ..)$

$\rightarrow\supset$: $(..\rightarrow..P \supset Q..)$ iff $(..P \rightarrow Q ..)$

$\sim\rightarrow$: $(..\sim P ..\rightarrow..)$ iff $(.. .. \rightarrow P ..)$

$\rightarrow\sim$: $(..\rightarrow.. \sim P ..)$ iff $(.. P \rightarrow.. ..)$

Atom: $(..P.. \rightarrow..P ..)$ iff true

These five rules are applied by replacing expressions which match the left hand side of these rules by the expression obtained from the right hand side expression. Note that the $\supset\rightarrow$ rule is similar to but not the same as applying R1 backwards. We see that this system trivially makes essential use of its axioms by the mere fact that there is only one axiom, namely "true". Of course, this fact alone is not a sufficient condition for a theorem prover to be very efficient, as the efficiency also depends on the precise nature of the inference rules of the system, but it is a necessary condition. Later in section 5 we will characterize the nature of these rules which cause this system to be so efficient at proving theorems. We now give a few examples of proofs obtained by using this system:

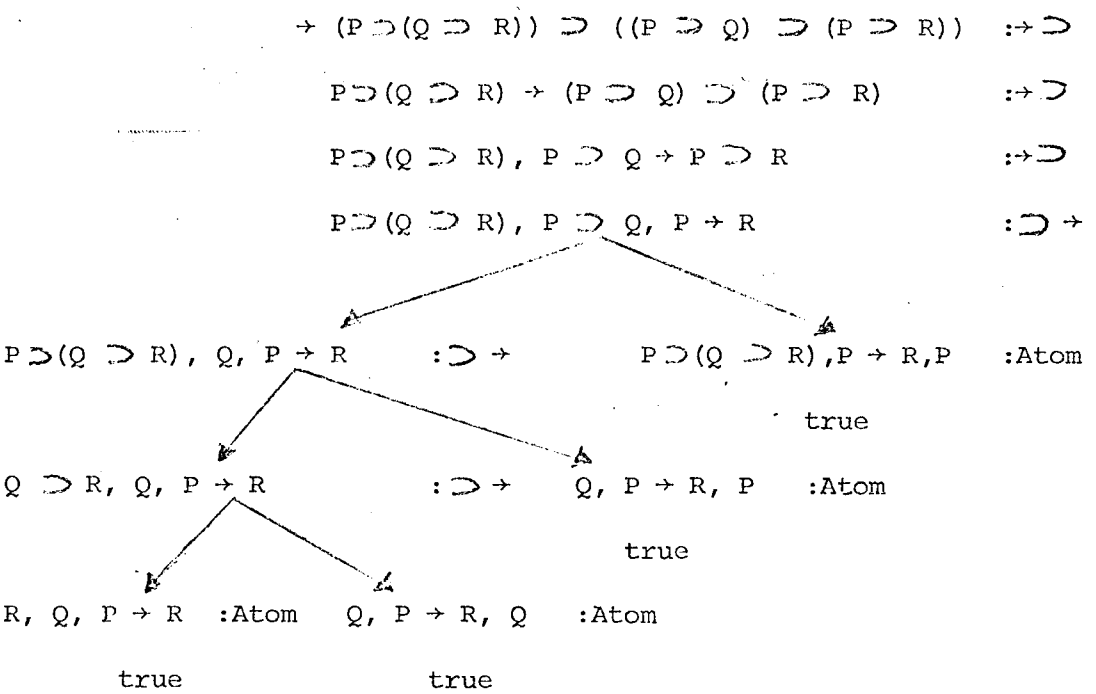
The first example is to prove that P implies P:

$\rightarrow P \supset P$ $:\rightarrow\supset$

$P \rightarrow P$:Atom

true

A more substantial example is the proof of A2:



Just a few examples should convince anyone as to the superiority of this theorem prover in comparison to the Logic Theorist.

5. Comparison

Having now described the propositional calculus and two deductive knowledge systems for it, we will now draw a few conclusions. First, we see that Wang's Algorithm is a far better method of proving theorems than is the Logic Theorist, for whereas Wang's Algorithm involves no searching and the Logic Theorist does involve searching.

Essentially we see that there are three underlying technical reasons for the superiority of Wang's systems:

(1) First it is semantically complete. That is, the conjunction of sequents resulting from the application of a rule to a sequent is logically equivalent to it.

(2) Second, it is terminal. That is the result of applying a rule decreases in an obvious way the complexity of the sequent.

(3) Third, it is deterministic, in that essentially only one rule is applicable to any formula in a sequent.

In summary then, we can say that the superiority of Wang's Algorithm is due precisely to its use of special rules for the domain of propositional logic which possess these three properties. On the other hand note that the Logic Theorist does not possess deductive knowledge with these properties but instead uses only the axiom and inference rules of propositional logic. So, in fact, we can say that the superiority of Wang's Algorithm is due to its possession of relatively sophisticated domain dependent deductive knowledge possessing the above characteristics.

6. Conclusion

We have seen that the superiority of Wang's Algorithm lies in its use of special domain dependent knowledge. We suggest that this is a very general phenomena and that given just about any mathematical domain, one can in practice find a special set of rules of such a nature as to result in a domain dependent theorem prover which would be superior to a theorem prover based on using only the axioms and inference rules of the given domain. In the next chapter we will investigate this last assertion as applied to the domain of elementary arithmetic.

In reading the next chapter, it should be borne in mind that it is not the fact that a domain dependent theorem prover is more efficient than a general purpose one based in this case on the Peano axioms of

arithmetic, that is being defended for who would seriously question this assertion? Rather it is the specific nature of the special rules (as stated in section 5) which should be used by such a domain dependent theorem prover, that is being defended.

III Doing Arithmetic without Diagrams

1. Introduction

This is a report of work completed in seventeen days during the fall of 1974. It describes an automatic theorem prover for the elementary theory of natural numbers, which represents theorems as lists and proves them by applying truth-value preserving transformations.

The motivation for this research derives from some suggestions made in 1973 as to how "Knowledgeable" theorem provers might be constructed. It will be recalled that at that time a number of researchers, Bundy [20,21] for the domain of elementary number theory and Ballantyne and Bennett [22] for the domain of topology, had advocated the construction of theorem provers along the following lines: That the hypotheses of theorems should be represented as diagrams, or rather semantic nets whose nodes are objects such as 1, 2, $\emptyset$, and whose arcs are properties such as $=$, $\leq$, $\subseteq$, ε; and that theorems should be proven by a process of forward chaining from the hypotheses in the diagram. The objective of this research was to determine whether it was better to base a theorem prover for elementary number theory on the traditional list notation and truth-value preserving transformations, or on diagrams and forward chaining.

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 $(fa_1 \ldots a_n)$ is a sub-expression being evaluated, then the interpreter tries to apply its items of knowledge to that sub-expression before evaluating the arguments $a_1 \ldots a_n$. For each sub-expression that the interpreter evaluates, in turn it tries to match the ϕ expression of each item to that sub-expression. If, however, during the application process an argument a_i 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+o=x and x+o=o+x are the only items and if they are listed to be used in that order then evaluating the theorem A+(B+o)≥A will cause the sub-expression B+o to be replaced by B resulting in A+B≥A. All memory of the sub-expression B+o is immediately lost upon its replacement by B and thus the interpreter does not attempt to apply the second item to B+o.

This theorem prover has a great deal of logical and arithmetic knowledge. We first describe the items of logical knowledge, and then the items of arithmetic knowledge.

2.1 Logical Knowledge

Out theorem prover has knowledge about twelve logical symbols

which are listed below with their English translations:

$\wedge$	and
$\vee$	or
$\sim$	not
$\blacksquare$	true
$\square$	false
$\supset$	implies
$\leftrightarrow$	iff
$\exists$	there exists
V	for all
=	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 $\phi 1$, ... ϕn called assertions which occur before the sequent arrow, and statements $\psi 1$, ... ψn called goals which occur after the sequent arrow. The implicit conjunction of different sequents may be thought of as being a group of different databases.

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

Assertion schemata:

$\square\rightarrow$: (.. $\square$ ..→..) ↔ (.. ..→..)

$\neg\rightarrow$: (.. $\neg$ ..→..) ↔ $\square$

$\sim\rightarrow$: (..$\sim$x ..→..) ↔ (.. ..→x ..)

$\wedge\rightarrow$: (..x$\wedge$y..→..) ↔ (.. x,y ..→..)

$\vee\rightarrow$: (..x$\vee$y..→..) ↔ (..x..→..) and (..y..→..)

$\supset\rightarrow$: (..x$\supset$y..→..) ↔ (.. ..→x..) and (..y..→..)

$\leftrightarrow\rightarrow$: (..x$\leftrightarrow$y..→..) ↔ (.. x,y ..→..) and (.. ..→ x,y ..) .

$\forall\rightarrow$: (..$\forall$xϕx..→..) ↔ (.. $\forall$xϕx,ϕ* ..→..)

where * is a new unification variable.

$\exists\rightarrow$: (..$\exists$xϕx..→..) ↔ (..$\phi(f^*_1 \ldots {}^*_n)$..→..)

where f is a new skolem function and $*_1 \ldots *_n$ are all the unification variables which occur in ϕx.

$=\rightarrow$: $(\Pi a .. \left\{ \begin{matrix} (f^*_1 \ldots {}^*_n)=t \\ t=(f^*_1 \ldots {}^*_n) \end{matrix} \right\} .. \Gamma a \rightarrow \phi a \; .. \psi a) \leftrightarrow (\Pi t .. \; .. \; \Gamma t \rightarrow \phi t \; .. \; \psi t)$

where a is $(f^*_1 \ldots {}^*_n)$
and f is a skolem function not occurring in t.

This is our version of the Law of Leibnitz, for example:

$(A=B+C \rightarrow C \leq A) \leftrightarrow (\rightarrow C \leq B+C)$

Goal schemata:

→$\square$: (..→.. $\square$ ..) ↔ $\square$

→$\neg$: (..→.. $\neg$ ..) ↔ (..→ .. ..)

→$\sim$: (..→..$\sim$x ..) ↔ (..x→.. ..)

→$\wedge$: (..→..x$\wedge$y..) ↔ (..→ ..x..) and (..→..y..)

→$\vee$: (..→..x$\vee$y..) ↔ (..→ ..x,y..)

→$\supset$: (..→..x$\supset$y..) ↔ (..x→..y..)

→$\leftrightarrow$: (..→..x$\leftrightarrow$y..) ↔ (..x→..y..) and (..y→..x..)

→$\forall$: (..→..$\forall$xϕx..) ↔ (..→ ..$\phi(f^*_1 \ldots {}^*_n)$..)

where f is a new skolem function and $*_1 \ldots *_n$ are all the unification variables which occur in ϕx.

$\rightarrow \exists$: $(..\rightarrow..\exists x \phi x..) \leftrightarrow (..\rightarrow..\phi *, \exists x \phi x..)$

where * is a new unification variable.

Other logical schemata:

Atom: $(..x..\rightarrow..x..) \leftrightarrow$ ▣ .

And : (..and ▣ and..) $\leftrightarrow$ (..and..)

Unify: $(..x_1..\rightarrow..y_1..)$ and..and $(..x_n..\rightarrow..y_n..)$

$\leftrightarrow$

$[(..x_i..\rightarrow..y_i..)$ and..and $(..x_n..\rightarrow..y_n..)]\theta$

where θ is any one of the substitutions which makes tautologous the greatest number of sequents starting with the first sequent and progressing towards the nth sequent. For example, an instance of this schema is:

[(P* → Pa, Pb) and (Q* → Qb) and (R* → Ra)] ↔ [Rb → Ra]

Note that the unification variable was instantiated to b, not to a, because the substitution [* =b] unifies the first two sequents, whereas [* =a] unifies only the first sequent.

Finally, we note that if all sequents can be made tautologous by some substitution then ▣ is returned.

A substitution θ is applied to a sequent as follows: Successively, for each substitution $[* = (t\ *_1 \ldots *_n)]$ every occurrence of * which occurs inside a skolem function is replaced by the segment $*_1 \ldots *_n$, and every other occurrence of * is replaced by the term $(t\ *_1 \ldots *_n)$. Thus for example if the sequent is

$$\rightarrow * = (f*)+*$$

where f is a skolem function, then applying the substitution $[* = (*_1 \doteq *_2)]$ produces the new sequent:

$$\rightarrow *_1 \doteq *_2 = (f *_1 *_2) + (*_1 \doteq *_2)$$

If the substitution is (*:= A) then the new sequent would be:

$$\rightarrow A = f + A$$

where f becomes a skolem constant.

The schemata $\forall \rightarrow$ and $\rightarrow \exists$ and applied only as last resorts. Furthermore, all other schemata are applied before the unify schema. More precisely, the logical schemata are used as follows: Initially the interpreter evaluates each sequent first trying to apply any assertion schema other than $\forall \rightarrow$, then trying to apply any goal schema other than $\rightarrow \exists$, and then trying to apply the atom schema. When no more of these schemata can be applied and if the result is not then the interpreter picks a sequent from the implicit conjunction of sequents and applies the $\rightarrow \exists$ and $\forall \rightarrow$ schemata once to each formula in that sequent which begins with $\exists$ or $\forall$ respectively. The resulting sequent is then evaluated as before using all the schemata except for $\rightarrow \exists$, $\forall \rightarrow$, and Unify. After evaluation the Unify schema is applied to the resulting sequents. Any sequents returned after applying the Unify schema, are added to the implicit conjunction of sequents and the process of picking a new sequent to work on repeats itself until

is derived.

It will turn out, however, that the theorem prover will prove all the arithmetic theorems given in this paper without needing to use the $\rightarrow \exists$, $\forall \rightarrow$, or Unify schemata. This is not the case for other mathematical domains such as set theory.

The fact that only assertion equations are ever eliminated by = → schema implies a preference, for sequents containing only equations, not to have more than one goal equation. The reason for this preference is that there is no logical item which allows two goal equations to interact to produce ▣. Thus one of the goals would probably be irrelevant.

The application of every assertion or goal procedure to a database involves a particular logical symbol other than 'and' or '→'. For this reason we may view such procedures as being instructions of a language which manipulates databases. For example, the execution of the goal instruction → ∧ on A ∧ B creates two new databases (worlds, contexts) as follows:

(1) First A ∧ B is erased from the current database.

(2) A copy of the database is then created. (Actually in our system all this structure is shared.)

(3) B is added as a new goal to the copy.

(4) A is added as a new goal to the current database.

A summary of the effect of goal and assertion instructions is given in Figure 1. Note that the interpreter has complete freedom of choice both as to what database to work on first, and as to what instruction to execute first.

An informed reader will recall that these propositional rules are used in Wang's algorithm in the LISP 1.5 manual [9], and that Wang [10] 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 [11] who used rules

Table 1: Summary of Logical Instructions

Instructions*	effect as an assertion	effect as a goal
	no-op****	succeed
¤	succeed***	no-op
∿A	add A as a goal	assert A
A ∧ B	assert A and B	split {add A as a goal; add B as a goal}
A ∨ B	split** {assert A; assert B}	add A and B as goals
A ⊃ B	split {add A as a goal; assert B}	assert A, add B as a goal
A ↔ B	split {add A and b as goals; assert A and B}	split {assert A, and add goal B; assert B, and add goal A}
$\forall x \phi x$	assert ϕ^* where * is a new unification variable.	add goal $\phi(f^*_1 \ldots ^*_n)$ where f is a new skolem function.
$\exists x \phi x$	assert $\phi(f^*_1 \ldots ^*_n)$ where f is a new skolem function	add goal ϕ^* where * is a new unification variable.
$\{(f^*_1 \ldots ^*_n) = t;\ t = (f^*_1 \ldots ^*_n)\}$	replace all occurrences of $(f^*_1 \ldots ^*_n)$ by t.	———

* With the exception of the ∀ assertion instruction and the ∃ goal instruction, and executed instruction is always erased.

** Split means that a new data base is created.

*** Succeed means that the data base itself is erased.

**** A null operation.

similar to all the rules given here except for $= \rightarrow$ and unify. His unification rule leads to a complete logic, ours does not.

2.2 Arithmetic Knowledge

Our theorem prover has knowledge about ten arithmetic symbols which are listed overleaf with their English translations:

o	zero
1	one
S	successor
P	funny predecessor (PO=O)
+	plus
$\dot{-}$	funny minus ($O\dot{-}X=X$)
•	times
<	less than
$\leqslant$	less than or equal to
|	divides

The items of arithmetic knowledge, some of which are lemmas because they do not involve ellipses, and some of which are schemata are listed below. It should be noted that certain items, as indicated, are applied only to goals, or only to assertions. Such items have been carefully chosen with these restrictions in mind, so as to minimize the introduction of unification variables.

Definitions:

A1: $S\ x = 1 + x$

A2: $P\ x = x \dot{-} 1$

A3: $x<y \leftrightarrow \sim\exists u\ y+u = x$ if the sub-expression being evaluated occurs in in a goal,

A4: $x<y \leftrightarrow \exists u\ x+(u+1) = y$ if the sub-expression being evaluated occurs in an assertion.

A5: $x \leq y \leftrightarrow \neg\exists u\ y+(u+1) = x$ if the sub-expression being evaluated occurs in a goal.

A6: $x \leq y \leftrightarrow \exists u\ x+u = y$ if the sub-expression being evaluated occurs in an assertion.

A7: $2 = S1$

A8: $3 = S2$

etc.

Items for equality:

A9: $x+y = o \leftrightarrow x = 0 \wedge y = o$

A10: $o = x+y \leftrightarrow x = o \wedge y = o$

A11: $x \cdot y = o \leftrightarrow x = o \vee y = o$

A12: $o = x \cdot y \leftrightarrow x = o \vee y = o$

A13: $x+y = 1 \leftrightarrow (x = o \wedge y = 1) \vee (x = 1 \wedge y = o)$

A14: $1 = x+y \leftrightarrow (x = o \wedge y = 1) \vee (x = 1 \wedge y = o)$

A15: $x \cdot y = 1 \leftrightarrow x = 1 \wedge y=1$

A16: $1 = x \cdot y \leftrightarrow x=1 \wedge y=1$

A17: $o = 1 \leftrightarrow \square$

A18: $1 = o \leftrightarrow \square$

A19: $o = o \leftrightarrow \square$

A20: $1 = 1 \leftrightarrow \square$

A21: $.. + x + .. = .. + x + .. \leftrightarrow .. + .. = .. + ..$

This last item is the cancellation schema of addition, which can be stated as follows: "The sentence containing two occurrences of the same term each connected via any number of plus signs to opposite

sides of an outermost equality sign is equivalent to the sentence obtained by cancelling those two occurrences of that term". For example:

$$A + (B + C) = D + (E + B) \leftrightarrow A + C = D + E$$

Items for divides:

A22: $x|1 \leftrightarrow x = 1$ if the sub-expression being evaluated occurs in a goal.

A23: $o|x \leftrightarrow x = o$ if the sub-expression being evaluated occurs in a goal.

A24: $1|x \leftrightarrow$ ▨ if the sub-expression being evaluated occurs in a goal.

A25: $x|o \leftrightarrow$ ▨ if the sub-expression being evaluated occurs in a goal.

A26: $x|(..\cdot x\cdot..)\{\dot{+}\}..\{\dot{+}\}(..\cdot x\cdot..) \leftrightarrow$ ▨ if the sub-expression being evaluated occurs in an assertion.

A more explicit statement of A26 is:

$$x|\phi x \leftrightarrow$$ ▨

where ϕx is a term formed from any number of + and $\dot{-}$ signs and any number of products $\alpha_1..\alpha_n$ such that each α_i is of the form:

$$\beta_{i_1}\cdot...\cdot\beta_{i_n}$$

and for all i there exists a j such that β_{i_j} is X.

For example:

$$A|(A\cdot C)+((B\cdot A)\dot{-}A) \leftrightarrow$$ ▨

A27: $x|y \leftrightarrow \exists u\ x\cdot u = y$ if the sub-expression being evaluated occurs in an assertion.

Lemmas for plus:

A28: $x+o = x$

A29: $o+x = x$

A30: $(x+y)+z = x+(y+z)$

A31: $(x+y)+z = z+(x+y)$ if z is not a term beginning with a plus sign.

Lemmas for times:

A32: $x \cdot o = o$

A33: $o \cdot x = o$

A34: $x \cdot 1 = x$

A35: $1 \cdot x = x$

A36: $x \cdot (y+z) = (x \cdot y)+(x \cdot z)$

A37: $(x+y) \cdot z = (x \cdot z)+(y \cdot z)$

A38: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

A39: $(x \cdot y) \cdot z = z \cdot (x \cdot y)$ if z is not a term beginning with a times sign.

Items for minus:

A40: $o \dot{-} x = o$

A41: $x \dot{-} o = x$

A42: $(x \dot{-} y) \dot{-} z = x \dot{-} (y+z)$

A43: $(..+ x +..) \dot{-} (..+ x +..) = (..+..) \dot{-} (..+..)$

for example: $A + B \dot{-} C + A = B \dot{-} C$

Lemmas for case analysis:

A44: $\phi[x \dot{-} y] \leftrightarrow (\forall z.x = z+y \supset \phi z) \wedge (\exists u\ x+u = y. \supset .\phi o)$ if the sub-expression being evaluated occurs in a goal.

A45: $\phi[x \dot{-} y] \leftrightarrow (\exists z\ x = z+y \wedge \phi z) \vee (\exists u\ x+u = y \wedge \phi o)$ if the sub-expression being evaluated occurs in an assertion.

Factorization schema:

A46: $(..\cdot x\cdot..)_{\{\underline{\cdot}\}}^{+}..{}_{\{\underline{\cdot}\}}^{+}(..\cdot x\cdot..) = (..\cdot x\cdot..)_{\{\underline{\cdot}\}}^{+}..{}_{\{\underline{\cdot}\}}^{+}(..\cdot x\cdot..)$

$\leftrightarrow (..\cdot..)_{\{\underline{\cdot}\}}^{+}..{}_{\{\underline{\cdot}\}}^{+}(..\cdot..) = (..\cdot..)_{\{\underline{\cdot}\}}^{+}..{}_{\{\underline{\cdot}\}}^{+}(..\cdot..) \vee x = o$

This last item is simply the operation of factoring out x. An example is:

$$A\cdot B + C\cdot A = D\cdot A \dot{-} A \leftrightarrow B+C = D\dot{-}1 \vee A = 0$$

These lemmas are used only when all else fails.

Proof by contradiction:

A47: $x = y \leftrightarrow \sim\exists u\ x+1+u = y \wedge \sim\exists w\ y+1+w = x$

if the sub-expression being evaluated occurs in a goal and all else fails.

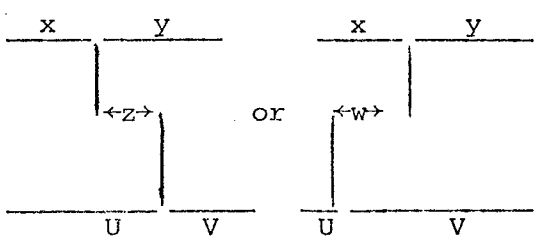
The proof by contradiction heuristic is essentially a method of allowing interaction between two goal equations. It does this by transforming one of the equations into assertion equations, thus allowing the extensionality schema = → to be used.

Magnitude heuristic:

A48: $x+y = u+v \leftrightarrow (\exists z\ x+z = u \wedge v+z = y) \vee (\exists w\ u+w = x \wedge y+w = v)$

if the sub-expression being evaluated occurs in an assertion and all else fails.

The justification that this last lemma is indeed a theorem can be seen by observing:



As in the case with expressions beginning with logical symbols, the interpreter tries to evaluate expressions beginning with arithmetic symbols in a call-by-need manner. That is, the interpreter tries to apply arithmetic items to arithmetic expressions, evaluating an argument only if it is necessary to effect the application. However, it will turn out that arguments which did not need to be evaluated in order to apply an item, will often be evaluated. Note that this would not be the case with the logical items, because most logical items never evaluate their arguments. The exception is the atom schema which does in fact save the partially evaluated expression for the use of the unify schema. The reason for this is as follows: Suppose the interpreter, while trying to apply an item to an expression $(f\ a_1\ ..\ a_n)$, evaluates some of the arguments say $a_1\ ..\ a_j$ to $a_1'\ ..\ a_j'$ getting $(f\ a_1'\ ..\ a_j'\ a_{j+1}\ ..a_n)$. Then suppose that the interpreter finds that it cannot apply that item. If the item is an arithmetic item, that is if f is an arithmetic symbol it indicates to the interpreter that $(fa_1'\ ..\ a_j'\ a_{j+1}\ ..a_n)$ not $(fa_1\ ..\ a_n)$ is now the sub-expression to be worked upon. All memory of the original expression $(fa_1\ ..\ a_n)$ is now erased, and the interpreter tries to apply the next arithmetic item in the order listed to the partially evaluated expression.

The advantage of always working on the partially evaluated expression is that the interpreter then avoids evaluating any arithmetic argument more than once. Because most of the arithmetic items do cause the evaluation of at least some arguments of an expression, this is significant. The system is able to always use the partially

evaluated expression rather than the original one: because what the arguments a_1 .. a_j evaluate to does not depend on f, but only on whether they are sub-expressions of a goal or an assertion. This property does not hold in more complicated mathematical domains such as set theory. Thus whereas the basic interpreter works in a call-by-need manner, evaluating arguments only when needed, on arithmetic expressions the interpreter often works in a call by value manner. That is it often evaluates all the arguments.

The items A47 and A48 are not always available to be used. These two items are used only as follows: If after evaluating every formulae in a particular sequent, $\square$ has not been derived, then if there is a goal equation, A47 is applied once, otherwise if there is an assertion equation, the interpreter tries to apply A48 once. If either of these items can be applied then the sequent is re-evaluated as previously described, otherwise the sequent itself is returned.

In section 2.1 we mentioned that the logical items could be viewed as being instructions in a database manipulation language. It turns out that this language is extensible in the sense that new instructions may be defined in terms of the primitive logical ones. For example A44, the $\phi[x \div y]$ goal instruction does not split the database, create and substitute new skolem constants for z and u, assert $x = z + y$ and add the goal ϕz to the first database, and assert $x + u = y$ and add the goal ϕo to the second database. Rather, $\phi[x \div y]$ simply returns the expression:

$$(\forall z.x = z + y \supset \phi z) \wedge (\exists u\ x + u = y \supset \phi o)$$

and lets the mathematical interpreter perform all these instructions

automatically using its logical items. ,

3. Examples

We give below a number of protocols produced by our theorem prover while proving various arithmetic theorems. We have listed on the right the name of the item which was applied to each sequent.

Example AE2: Associativity of Plus

$\rightarrow A + (B + C) = (A + B) + C$:A30

$\rightarrow A + (B + C) = A + (B + C)$:A43

$\rightarrow B + C = B + C$:A43

$\rightarrow C = C$:A43

$\rightarrow o = o$:A19

$\rightarrow$ ■ :$\rightarrow$ ■

time = 38 millisec.

Example AE13: an example of the importance of case analysis.

$\rightarrow (A + B) \dot{-} C = (A \dot{-} C) + B \dot{-} (C \dot{-} A)$:case analysis A44

↓

$\rightarrow [\forall x\ A = x + C \supset (A + B) \dot{-} C = x + (B \dot{-} (C \dot{-} A))] \wedge$

$[\exists x\ A + x = C \supset (A + B) \dot{-} C = o + (B \dot{-} (C \dot{-} A))]$:$\rightarrow \wedge$

"A"

$\rightarrow \forall x\ A = x + C \supset (A + B) \dot{-} C = x + (B \dot{-} (C \dot{-} A))$:$\rightarrow \forall$

$\rightarrow A = C_1 + C \supset (A + B) \dot{-} C = C_1 + (B \dot{-} (C \dot{-} A))$:$\rightarrow \supset$

$A = C_1 + C \rightarrow (A + B) \dot{-} C = C_1 + (B \dot{-} (C \dot{-} A))$:$\Rightarrow$

$\rightarrow ((C_1 + C) + B) \dot{-} C = C_1 + (B \dot{-} (C \dot{-} (C_1 + C)))$:A30

$\rightarrow C_1 + (C + B) \dot{-} C = C_1 + (B \dot{-} (C \dot{-} (C_1 + C)))$:A43

$\rightarrow (C_1 + B) \dot{-} o = C_1 + (B \dot{-} (C \dot{-} (C_1 + C)))$:A41

$\rightarrow C_1 + B = C_1 + (B \dot{-} (C \dot{-} (C_1 + C)))$:A43

↓

$\rightarrow C_1 + B = C_1 + (B \doteq (o \doteq C_1))$:A40

$\rightarrow C_1 + B = C_1 + (B \doteq o)$:A28

$\rightarrow C_1 + B = C_1 + B$:A21

$\rightarrow o = o$:A19

$\rightarrow$ ▣ :→▣

"A"

$\rightarrow \exists x\ A + x = C. \supset .\ (A + B) \doteq C = o + (B \doteq (C \doteq A))$:→⊃

$\exists x\ A + x = C \rightarrow (A + B) \doteq C = o + (B \doteq (C \doteq A))$:∃→

$A + C_2 = C \rightarrow (A + B) \doteq C = o + (B \doteq (C \doteq A))$:⇒→

$\rightarrow (A + B) \doteq (A + C_2) = o + (B \doteq ((A + C_2) \doteq A))$:A43

$\rightarrow B \doteq C_2 = o + (B \doteq ((A + C_2) \doteq A))$:A43

$\rightarrow B \doteq C_2 = o + (B \doteq (C_2 \doteq o))$:A41

$\rightarrow B \doteq C_2 = o + (B \doteq C_2)$:A29

$\rightarrow B \doteq C_2 = B \doteq C_2$:A21

$\rightarrow o = o$:A19

$\rightarrow$ ▣ :→▣

time = 151 millisec

Example AF10: The theorem prover discovers that the second hypothesis of this theorem 'B≤C' is not needed. In other words that:

$$A \leqslant C \supset A \doteq B = (C \doteq B) \doteq (C \doteq A)$$

is a theorem. A general rule for discovering irrelevant hypotheses is this:

If a descendent of a hypotheses occurs in every sequent in the proof and if no such descendent is ¤ and an assertion: and if no such descendent is ▣ and a goal, and if the Unify and Atom rules are not used then the hypothesis is irrelevant.

$\to A \leqslant C \wedge B \leqslant C \supset A \dot{-} B = (C \dot{-} B) \dot{-} (C \dot{-} A)$:$\to\supset$

$A \leqslant C \wedge B \leqslant C \to A \dot{-} B = (C \dot{-} B) \dot{-} (C \dot{-} A)$:$\wedge\to$

$A \leqslant C, B \leqslant C \to A \dot{-} B = (C \dot{-} B) \dot{-} (C \dot{-} A)$:A6

$\exists u\, A + u = C, B \leqslant C \to A \dot{-} B = (C \dot{-} B) \dot{-} (C \dot{-} A)$:$\exists\to$

$A + C_1 = C, B \leqslant C \to A \dot{-} B = (C \dot{-} B) \dot{-} (C \dot{-} A)$:$=\to$

$B \leqslant A + C_1 \to A \dot{-} B = ((A + C_1) \dot{-} B) \dot{-} ((A + C_1) \dot{-} A)$:A6

$\exists u\, B + u = A + C_1 \to A \dot{-} B = ((A + C_1) \dot{-} B) \dot{-} ((A + C_1) \dot{-} A)$:$\exists\to$

$B + C_2 = A + C_1 \to A \dot{-} B = ((A + C_1) \dot{-} B) \dot{-} ((A + C_1) \dot{-} A)$:A43

$B + C_2 = A + C_1 \to A \dot{-} B = ((A + C_1) \dot{-} B) \dot{-} (C_1 \dot{-} 0)$:A41

$B + C_2 = A + C_1 \to A \dot{-} B = ((A + C_1) \dot{-} B) \dot{-} C_1$:A42

$B + C_2 = A + C_1 \to A \dot{-} B = (A + C_1) \dot{-} (B + C_1)$:A43

$B + C_2 = A + C_1 \to A \dot{-} B = A \dot{-} B$:A21

$B + C_2 = A + C_1 \to 0 = 0$:A19

$B + C_2 = A + C_1 \to$ ▨ :$\to$▨

time = 76 millisec

Example AM7: The Law of De Morgan

$\to 1 \dot{-} (A + B) = (1 \dot{-} A) \cdot (1 \dot{-} B)$:case analysis A44

↓

$\to [\forall x\ 1 = x + A \supset 1 \dot{-} (A + B) = x \cdot (1 \dot{-} B)] \wedge$

$[\exists x\ 1 + x = A \supset 1 \dot{-} (A + B) = 0 \cdot (1 \dot{-} B)]$:$\to\wedge$

and

"C"

$\to \forall x\ 1 = x + A \supset 1 \dot{-} (A + B) = x \cdot (1 \dot{-} B)$:$\to\forall$

$\to 1 = C_1 + A \supset 1 \dot{-} (A + B) = C_1 \cdot (1 \dot{-} B)$:$\to\supset$

$1 = C_1 + A \to 1 \dot{-} (A + B) = C_1 \cdot (1 \dot{-} B)$:A14

$(C_1 = 0 \wedge A = 1) \vee (C_1 = 1 \wedge A = 0) \to 1 \dot{-} (A + B) = C_1 \cdot (1 \dot{-} B)$:$\vee\to$

"A" "B"

"A"

$C_1 = o \wedge A = 1 \to 1 \dot{-} (A + B) = C_1 \cdot (1 \dot{-} B)$:/\→

$C_1 = o,\ A = 1 \to 1 \dot{-} (A + B) = C_1 \cdot (1 \dot{-} B)$:⇒

$A = 1 \to 1 \dot{-} (A + B) = o\ (1 \dot{-} B)$:⇒

$\to 1 \dot{-} (1 + B) = o \cdot (1 \dot{-} B)$:A43

$\to o \dot{-} B = o \cdot (1 \dot{-} B)$:A40

$\to o = o \cdot (1 \dot{-} B)$:A33

$\to o = o$:A19

→ ▩ :→▩

"B"

$C_1 = 1 \wedge A = o \to 1 \dot{-} (A + B) = C_1 \cdot (1 \dot{-} B)$:/\→

$C_1 = 1,\ A = o \to 1 \dot{-} (A + B) = C_1 \cdot (1 \dot{-} B)$:⇒

$A = o \to 1 \dot{-} (A + B) = 1 \cdot (1 \dot{-} B)$:⇒

$\to 1 \dot{-} (o + B) = 1 \cdot (1 \dot{-} B)$:A29

$\to 1 \dot{-} B = 1 \cdot (1 \dot{-} B)$:A35

$\to 1 \dot{-} B = 1 \dot{-} B$:A21

$\to o = o$:A19

→ ▩ :→▩

"C"

$\to \exists x\ 1 + x = A \supset 1 \dot{-} (A + B) = o \cdot (1 \dot{-} B)$:→⊃

$\exists x\ 1 + x = A \to 1 \dot{-} (A + B) = o \cdot (1 \dot{-} B)$:∃→

$1 + C_2 = A \to 1 \dot{-} (A + B) = o \cdot (1 \dot{-} B)$:⇒

$\to 1 \dot{-} ((1 + C_2) + B) = o \cdot (1 \dot{-} B)$:A30

$\to 1 \dot{-} (1 + (C_2 + B) = o \cdot (1 \dot{-} B)$:A43

$\to o \dot{-} (C_2 + B) = o \cdot (1 \dot{-} B)$:A40

$\to o = o \cdot (1 \dot{-} B)$:A33

$\to o = o$:A19

→ ■ :→ ■

time = 189 millisec

Example AM32: No even number equals any odd number. We did not think that the theorem prover would be able to prove this theorem. In fact, we did not think that this theorem was implied by the items used by the theorem prover, since induction is not one of them.

→ ∿ 2•A = (2•B) + 1 :→∿

2•A = (2•B) + 1 → :A7

(S1)•A = (2•B) + 1 → :A1

(1+1)•A = (2•B) + 1 → :A37

(1•A)+(1•A) = (2•B) + 1 → :A35

A + (1•A) = (2•B) + 1 → :A34

A+A = (2•B) + 1 → :A7

A+A = ((S1)•B) + 1 → :A1

A+A = ((1+1)•B) + 1 → :A37

A+A = ((1•B)+(1•B)) + 1 → :A35

A+A = (B+(1•B)) + 1 → :A34

A+A = (B+B) + 1 → :A31

A+A = 1+(B+B) → :magnitude heuristic A48

[∃u A+u=1 ∧ (B+B)+u=A] ∨ [∃w 1+w=A ∧ A+w=B+B]→ :∨→

and "A"

∃u A+u = 1 ∧ (B+B) + u = A → :∃→

$A+C_1 = 1 \wedge (B+B) + C_1 = A \rightarrow$:∧→

$A+C_1 = 1, (B+B) + C_1 = A \rightarrow$:A13

$(A=o \wedge C_1=1) \vee (A=1 \wedge C_1=o),\ (B+B)+C_1 = A \rightarrow$:$\vee\rightarrow$

and

$A=o \wedge C_1=1,\ (B+B)+C_1=A \rightarrow$	:$\wedge\rightarrow$	$A=1 \wedge C_1=o,\ (B+B)+C_1=A \rightarrow$	:$\wedge\rightarrow$
$A=o,\ C_1=1,\ (B+B)+C_1=A \rightarrow$	:$\Rightarrow$	$A=1,\ C_1=o,\ (B+B)+C_1=A \rightarrow$	:$\Rightarrow$
$C_1=1,\ (B+B)+C_1=o \rightarrow$	:$\Rightarrow$	$C_1=o,\ (B+B)+C_1=1 \rightarrow$	:$\Rightarrow$
$(B+B)+1=o \rightarrow$	:A31	$(B+B)+o=1 \rightarrow$	:A31
$1+(B+B)=o \rightarrow$	:A9	$o+(B+B)=1 \rightarrow$	:A29
$1=o \wedge B+B=o \rightarrow$	:$\wedge\rightarrow$	$B+B=1 \rightarrow$	:A13
$1=o,\ B+B=o \rightarrow$	:A18	$(B=o \wedge B=1) \vee (B=1 \wedge B=o) \rightarrow$	:$\vee\rightarrow$
¤ , $B+B=o \rightarrow$	:¤ $\rightarrow$		

and

$B=o \wedge B=1 \rightarrow$	:$\wedge\rightarrow$	$B=1 \wedge B=o \rightarrow$	:$\wedge\rightarrow$
$B=o,\ B=1 \rightarrow$	:$\Rightarrow$	$B=1,\ B=o \rightarrow$	:$\Rightarrow$
$o=1 \rightarrow$	:A17	$1=o \rightarrow$	:A17
¤ $\rightarrow$	:¤ $\rightarrow$	¤ $\rightarrow$	:¤ $\rightarrow$

"A"

$\exists w\ 1+w = A \wedge A+w = B+B \rightarrow$	:$\exists\rightarrow$
$1+C_2 = A \wedge A+C_2 = B+B \rightarrow$	:$\wedge\rightarrow$
$1+C_2 = A,\ A+C_2 = B+B \rightarrow$	:$\Rightarrow$
$(1+C_2) + C_2 = B+B \rightarrow$	:A31
$C_2+(1+C_2) = B+B \rightarrow$	:magnitude heuristic A48

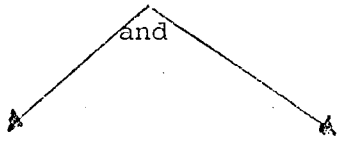
$(\exists u\ C_2+u=B \wedge B+u=1+C_2) \vee (\exists w\ B+w=C_2 \wedge (1+C_2)+w=B) \rightarrow$:$\vee\rightarrow$

$\exists u\ C_2+u=B \wedge B+u=1+C_2 \rightarrow$	:$\exists\rightarrow$	$\exists w\ B+w=C_2 \wedge (1+C_2)+w=B \rightarrow$	:$\exists\rightarrow$
$C_2+C_3=B \wedge B+C_3=1+C_2 \rightarrow$	:$\wedge\rightarrow$	$B+C_4=C_2 \wedge (1+C_2)+C_4=B \rightarrow$	:$\wedge\rightarrow$
$C_2+C_3=B,\ B+C_3=1+C_2 \rightarrow$	:$\Rightarrow$	$B+C_4=C_2,\ (1+C_2)+C_4=B \rightarrow$	:$\Rightarrow$
$(C_2+C_3)+C_3=1+C_2 \rightarrow$	:A31	$(1+(B+C_4))+C_4=B \rightarrow$	:A31

$C_3+(C_2+C_3)=1+C_2$ → :A21

$C_3+C_3=1$ → :A13

$(C_3=o \wedge C_3=1) \vee (C_3=1 \wedge C_3=o)$ → :∨→



$C_3=o \wedge C_3=1$ → :∧→ $C_3=1 \wedge C_3=o$ →:∧→

$C_3=o$, $C_3=1$ → :⇒→ $C_3=1$, $C_3=o$ →:⇒→

$o=1$ → :A17 $1=o$ → :A18

¤ → :¤ → ¤ → :¤ →

$C_4+(1+(B+C_4))=B$ → :A21

$C_4+(1+C_4)=o$ → :A9

$C_4=o \wedge 1+C_4=o$ → :∧→

$C_4=o$, $1+C_4=o$ → :⇒→

$1+o=o$ → :A28

$1=o$ → :A18

¤ → :¤ →

time = 432 millisec

Note that the magnitude heuristic A48 is applied each time to essentially the same formulae, but with a different parenthesization and commutation.

4. Results

In order to meet our objective we decided to make a detailed empirical study of the deductive ability of this theorem prover in comparison to the ability of an arithmetic theorem prover based on diagrams and forward chaining. The theorem prover we compared ours to, called SUMS by its author, is described in [20, 21]*. We ran all 87 theorems that SUMS had attempted to prove up to the time of IJCAI3. A summary of the results obtained by both theorem provers is given in Figure 2. Individual results for each of the 87 test theorems are given in Figure 3. All times in Figure 3 are given in milliseconds. If '(fail)' appears beside a time, it means that the theorem prover took that much time trying to prove the theorem before deciding to quit.

* A re-appraisal of SUMS, in view of this research, is given in [23].

Figure 2: Summary of Results

	Our System	SUMS Memo 61	SUMS IJCAI3	
	Oct. 1974	Feb.1973***	Aug.1973***	21 Sept.1973
Proven	86	20	64	70
Failed	1*	0	22	16
Untried	0	67	1	0
cpu time (sec)**	14 (DEC10,KA10)	-	-	442 (ICL4130)

* Due to an overflow on an internal stack of the LISP system while trying to prove 200+200 = 400. This was caused by the fact that the theorem prover represents numerals in unary notation.

** Our DEC10 is about 3 times faster than the ICL4130.

*** Date of publication of memo. Actual runs were made some time sooner.

Figure 3: Individual Results

Theorem		time (ms)	SUMS-time (ms)
AE1:	$A + B = B + A$	19	750
AE2:	$A + (B + C) = (A + B) + C$	38	3,625
AE3:	$P\,(A \dot{-} B) = PA \dot{-} B$	301	9,125
AE4:	$SA \dot{-} SB = A \dot{-} B$	38	8,563
AE5:	$(A + C) \dot{-} (B + C) = A \dot{-} C$	35	7,063
AE6:	$A \dot{-} (A + B) = o$	37	875
AE7:	$1 \dot{-} SA = o$	18	937
AE8:	$1 \dot{-} (1 \dot{-} SA) = 1$	37	1,250
AE9:	$A + (B \dot{-} A) = B + (A \dot{-} B)$	113	4,813
AE10:	$A \dot{-} (A \dot{-} B) = B \dot{-} (B \dot{-} A)$	111	4,125
AE11:	$A \dot{-} (B + C) = (A \dot{-} B) \dot{-} C$	38	12,500
AE12:	$(A \dot{-} B) \dot{-} C = (A \dot{-} C) \dot{-} B$	303	18,630
AE13:	$(A + B) \dot{-} C = (A \dot{-} C) + (B \dot{-} (C \dot{-} A))$	151	23,560
AE14:	$A \dot{-} B \leq A$	132	2,188
AE15:	$A \dot{-} (A \dot{-} B) \leq A$	147	2,750
AE16:	$A \dot{-} (A \dot{-} B) \leq B$	151	2,688
AE17:	$A \leq A + (B \dot{-} A)$	57	3,000
AE18:	$B \leq A + (B \dot{-} A)$	170	2,938
AE19:	$o \leq SA$	37	875
AE20:	$A \leq SA$	38	812
		(1,971)	(111,067)

		time(ms)	SUMS-time (ms)	
AF1:	$A \leqslant B \wedge B \leqslant A \supset A = B$	96	1,688	
AF2:	$A \leqslant B \supset B = A + (B \dot{-} A)$	38	1,313	
AF3:	$B = A + C \supset A \leqslant B$	56	1,000	
AF4:	$A \leqslant B \wedge B \leqslant C \supset A \leqslant C$	94	1,125	
AF5:	$SA \leqslant SB \supset A \leqslant B$	114	7,063	
AF6:	$A = B \supset SA = SB$	57	3,438	
AF7:	$A \leqslant B \wedge C \leqslant D \supset A + C \leqslant B + D$	94	9,063	(fail)
AF8:	$A \leqslant B \wedge C \leqslant D \supset A \dot{-} D \leqslant B \dot{-} C$	227	14,440	(fail)
AF9:	$C \leqslant B \supset A \dot{-} (B \dot{-} C) = (A + C) \dot{-} B$	75	8,750	
AF10:	$A \leqslant C \wedge B \leqslant C \supset A \dot{-} B = (C \dot{-} B) \dot{-} (C \dot{-} A)$	76	1,000	
AF11:	$A \leqslant B \supset A = A \dot{-} (A \dot{-} B)$	57	500	
AF12:	$B < A \vee A \leqslant B$	76	2,688	
AF13:	$A \leqslant B \leftrightarrow A < B \vee A = B$	999*GC	6,625	
AF14:	$A \leqslant B \leftrightarrow A \leqslant SB \wedge \sim A = SB$	284	2,438	
AF15:	$(\sim A = o) \supset PA < A \wedge A = SPA$	433	1,813	
AF16:	$A \leqslant B \wedge B < C \supset A < C$	113	2,188	
AF17:	$A < B \wedge B \leqslant C \supset A < C$	109	7,750	
AF18:	$A < B \leftrightarrow SA < SB$	150	4,875	
AF19:	$A < B \wedge C \leqslant D \supset A + C < B + D$	111	10,380	(fail)
AF20:	$A < B \wedge B \leqslant C \supset C \dot{-} B < C \dot{-} A$	132	4,875	
AF21:	$A + B < C \leftrightarrow A < C \dot{-} B$	226	10,500	
AF22:	$A < B \supset \sim B = o$	38	937	
AF23:	$(\sim A = o) \supset 1 \leqslant A$	114	625	
		(3,769)	(105,073)	

		time (ms)	SUMS-time (ms)	
AM1:	$o \cdot A = o$	36	312	
AM2:	$1 \cdot A = A$	37	375	
AM3:	$A \cdot B = B \cdot A$	35	1,063	
AM4:	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	38	3,688	
AM5:	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	38	3,625	
AM6:	$A \cdot (B \dot{-} C) = (A \cdot B) \dot{-} (A \cdot C)$	38	8,625	
AM7:	$1 \dot{-} (A + B) = (1 \dot{-} A) \cdot (1 \dot{-} C)$	189	35,500	
AM8:	$(1 \dot{-} (A \cdot B)) \cdot B = (1 \dot{-} A) \cdot B$	585	14,250	
AM9:	$A \leq B \wedge C \leq D \supset A \cdot C \leq B \cdot D$	132	6,188	(fail)
AM10:	$A < B \wedge C < D \supset A \cdot C < B \cdot D$	1,000*GC	10,750	(fail)
AM11:	$(\sim A = o) \wedge (\sim B = o) \supset \sim A \cdot B = o$	220	3,000	
AM12:	$(\sim B = o) \supset A \leq A \cdot B$	321	1,500	
AM13:	$(\sim C = o) \wedge A < B \supset A \cdot C < B \cdot C$	208	5,250	
AM14:	$A = B \cdot C \wedge (\sim C = o) \supset B \leq A$	359	3,500	(fail)
AM15:	$2 \leq A \wedge 2 \leq B \supset A + B \leq A \cdot B$	208	9,500	(fail)
AM16:	$B = A \cdot C \supset A \mid B$	38	1,063	
AM17:	$A \mid o$	19	125	
AM18:	$A \mid A \cdot B$	37	812	
AM19:	$1 \mid A$	38	125	
AM20:	$A \mid A$	18	125	
AM21:	$A \mid B \wedge B \mid C \supset A \mid C$	100	1,188	
AM22:	$A \mid B \wedge \sim B = o \supset A \leq B$	416	2,125	(fail)
AM23:	$o \mid A \supset A = o$	37	1,250	
AM24:	$A \mid 1 \supset A = 1$	57	687	
AM25:	$A \mid B \wedge B \mid A \supset A = B$	132	1,125	

		time (ms)	SUMS-time (ms)
AM26:	$A\|B \supset A\|B \cdot C$	53	1,438
AM27:	$A\|B \wedge A\|C \supset A\|B + C$	93	1,500 (fail)
AM28:	$A\|B \wedge A\|C \supset A\|B \dot{-} C$	94	3,813 (fail)
AM29:	$A\|B \wedge C\|D \supset (A \cdot C) \mid (B \cdot D)$	94	2,938 (fail)
AM30:	$A\|B \supset A \cdot C \mid B \cdot C$	75	2,375
AM31:	$A\|B \wedge A\|SB \supset A = 1$	2,145*GC	3,250 (fail)
AM32:	$\sim 2 \cdot A = (2 \cdot B) + 1$	432	10,250 (fail)
		(6,362)	(141,310)
AEV1:	$o + o = o$	23	1,000
AEV2:	$o \cdot o = o$	37	375
AEV3:	$o \cdot 1 = o$	19	875
AEV4:	$2 + 2 = 4$	75	3,436
AEV5:	$200 + 200 = 400$ [time lost due to failing]*GC		2,938
AEV7:	$(A + 2) \cdot (A + 3) = A \cdot A + 5 \cdot A + 6$	264	40,440 (fail)
AEV8:	$2 \leq A \wedge A < 4 \leftrightarrow A = 2 \vee A = 3$	496	20,130 (fail)
AEV9:	$A + A = 2 \cdot A$	37	2,438
AEV10:	$A + (A + A) = 3 \cdot A$	57	2,938 (fail)
AEV11:	$2 \leq 5$	96	1,938
AEV12:	$A \leq 2 \wedge 5 \leq B \supset A \leq B$	255	4,813
		(1,431)	(85,321)
	TOTALS:	13,533 ms	442,761 ms
		apx 14 sec	apx 442 sec
			or 7 minutes

*GC - means a LISP garbage collection occurred.

Notes for Figure 3

(1) AE17 may be generalized to $A \leq A + C$.

(2) AF10's second hypothesis is not needed.

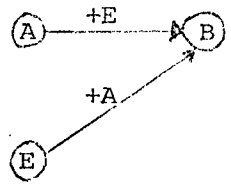
(3) No less than 17 of these theorems follow as logical instances of the items. AE1, 2, 11, AM1, 2, 3, 4, 5, 17, 18, 19, 20, 23, 24, AEV1, 2, 3. As many of these arithmetic items are more or less embodied in the code of the SUMS theorem prover [20, 21], one may wonder just in what sense our theorem prover and SUMS can be said to have proven these theorems, for normally one wishes to prove theorems, by assuming only simpler more perceivable truths as lemmas. This is a major reason for the doctrine of extensibility [24]; which implies that we must not implicitly assume a lemma while trying to prove it. This doctrine is described in the next chapter.

It will be apparent from Figures 2 and 3 that on these test examples, superior results were obtained by our theorem prover. These superior results were due to our theorem prover's use of list notation and truth value preserving transformations. To see why this is true it is first necessary to have some understanding as to how SUMS proves theorems.

The basic idea behind SUMS is that theorems should be proven as follows: First the hypotheses of the theorem should be represented as a semantic net, in such a manner that the skolem constants of the formulas become nodes, and the predicates become the arcs. For example, if the theorem were :

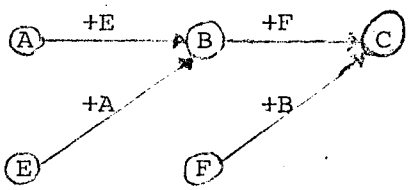
$$A \leqslant B \wedge B \leqslant C \rightarrow A \leqslant C$$

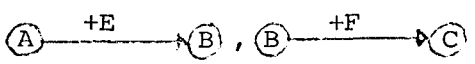
the hypothesis $A \leqslant B$ would first be represented as:



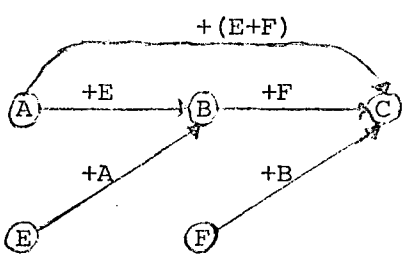
Note that $A \leqslant B$ iff there exists an E such that $A + E = B$. Thus SUMS actually uses the relation + E to state that $A \leqslant B$. E is just a new skolem constant.

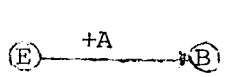
The hypothesis $B \leqslant C$ is next added to the net resulting in:

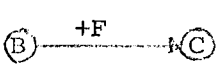


Having represented all the hypotheses, SUMS now begins a process of creating new relations by applying general arithmetic laws to the hypotheses. For example, the transitivity law is applied to the facts: 

, resulting in a new diagram:



Note that the transitivity law may also be applied to 

and 

.

After a while this process of forward chaining is stopped, and the system tries to access the conclusion of the theorem in the net.

For example, for the above theorem the system tries to access the fact that $A \leq C$, or rather the fact that there exists an X such that $A + X = C$. Since (A) —+(E+F)→ (C) is in the net the system succeeds in proving the theorem.

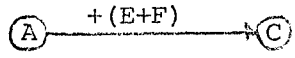
There are two points about the techniques used in SUMS which have been thought to be particularly important. The first is that SUMS uses what has been called "controlled" forward chaining; controlled in the sense that although new relations such as +(E+F) are created by forward chaining, no new terms (nodes) are ever created. This has been said to greatly limit the size of the space searched.

The second point thought to be important is the use of the semantic net data structure. The idea is that if most of the deduction carried out in the system is forward chaining, then a data structure in which one can go directly to the next property via a node will be quite useful. For example, the transitivity law after going from A to B collecting +E, is already at B and thus merely uses the +F arc to go to C , thus collecting +F and creating the new arc +(E+F).

Having now described the diagram and forward chaining method used by SUMS we now compare this to an arithmetic theorem prover based on list notation, and truthvalue preserving transformations.

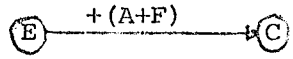
First the semantic net representation of SUMS is redundant in that every hypothesis is represented twice. In our above example $A \leq B$ is represented both as (A) —+E→ (B) and as (E) —+A→ (B). When forward chaining this redundancy means that there will be

alternate ways of deducing each new fact. In the above net, using the transitivity law we derive:



or rather $A + E + F = C$

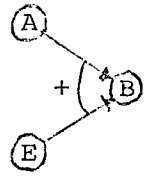
and



or rather $E + A + F = C$

which are simply two different ways of saying that $A \leq C$.

Granted then that the particular representation of formulas used by SUMS is redundant, and that maybe a net representation like:



should have been used, was there any point in using semantic nets as opposed to the traditional list notation in the first place? The simple fact is this: typical theorems of number theory do not involve lengthy property chains such as:

$$A_1 \leq A_2 \wedge A_2 \leq A_3 \wedge \ldots . \wedge A_{n-1} \leq A_n$$

Thus no significant time could be gained by having explicit pointers for properties. Furthermore, the use of a semantic network data structure makes many simple logical laws, very time-consuming to perform: For example, consider the formulae of the form:

$$\Gamma \rightarrow (A \supset B) \wedge (C \supset D)$$

where Γ is the current semantic net. We would like to split on the $\wedge$ sign getting:

$$\Gamma \rightarrow (A \supset B) \text{ and } \Gamma \rightarrow (C \supset D)$$

and then:

$\Gamma, A \rightarrow B$ and $\Gamma, C \rightarrow D$

But if Γ is a semantic net, then Γ must be duplicated on the split because later A must be put into one net while C must be put into the other net. If the traditional list representation had been used then all this would have been automatic because Γ would not have to be copied.

Beyond this redundancy in the way facts are represented in the net, there is an even worse redundancy caused by the arithmetic laws used to do the forward chaining. In our previous example, when we forward chain from: (A) —+E→ (B) —+F→ (C) to get the new fact: (A) —+(E+F)→ (C) the two previous facts are not deleted, but remain in the net. Since $A + E = B \wedge B + F = C$ is equivalent to $A + E + F = C$, this amounts to two ways of saying the same thing. Note that this redundancy caused by forward chaining means that the semantic net will grow continuously throughout the course of the proof.

This redundancy is eliminated in a system based on truthvalue preserving transformations, in that the previous expressions may be deleted, because we know that the resulting expression is equivalent to the previous one. For example, applying the item $= \rightarrow$ to the first equation of:

$$A + E = B, B + F = C \rightarrow A \leq C$$

gives:

$$(A + E) + F = C \rightarrow A \leq C$$

applying $= \rightarrow$ to the second equation gives:

$$A \leq A + E + F$$

which is true. Note that our system discards each equation after

using it once and only once, never redundantly. Thus it is quite typical for the number of such equations in our sequents to decrease during the course of proving a theorem, even to the extent that when ▣ is finally produced it is the only formula in the sequent.

Having now shown how redundant forward chaining is, let us consider the question of "controlled" forward chaining. We point out that the "control" which in this case is simply a ban on the creation of new terms (nodes) not already occurring in the theorem, is irrelevant because all it does is make an arbitrary (random if you like) pruning of the search space.

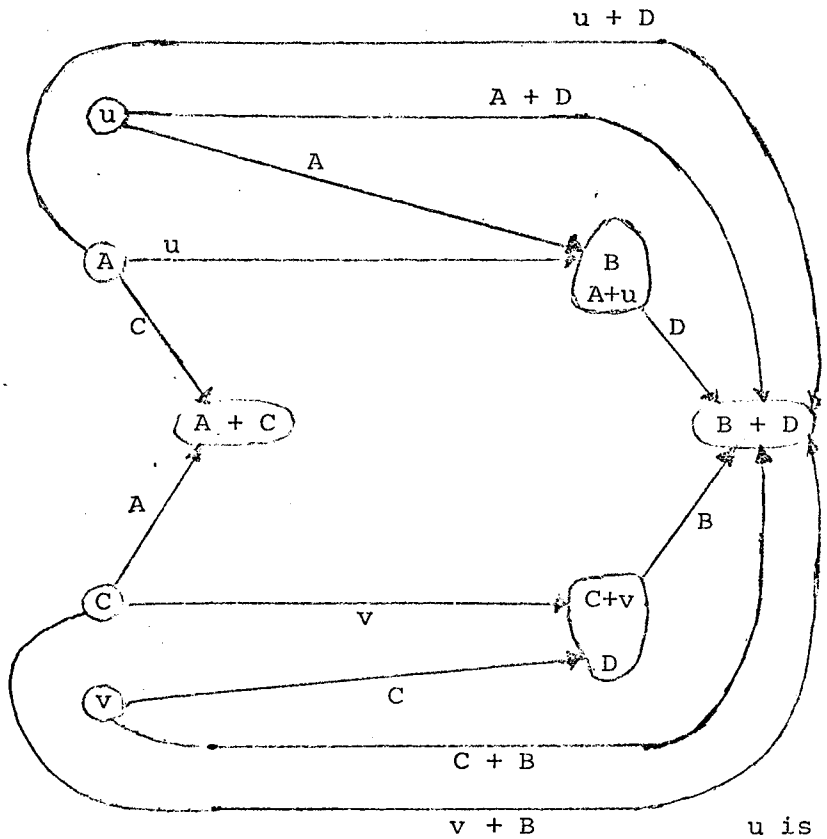
Just as an example of how irrelevant is this ban, consider the theorem AF7.

AF7: $A \leqslant B \wedge C \leqslant D \rightarrow A + C \leqslant B + D$

In Figure 4 we display the semantic net that SUMS would have produced after examining the hypotheses and the sub-expressions of the goal. Note that expressions of the form $A \leqslant B$ are treated as if they were $A + u = B$ where u is a new constant. The net representation of this last statement is (A) —u→ (B) and also (u) —A→ (B).

SUMS fails to prove this theorem because given its ban on creating new nodes, there is no way it can produce an arc from A + C to B + D. If, however, SUMS had a rule which could reverse node x and arc label y in (x) —y→ ( ) to get (y) —x→ ( ), thus possibly creating a new node y, a proof of this theorem could have been obtained. In Figure 5 such a net is shown where in step 1 (A) —u + D→ (B + D) has been reversed to create a new node (u + D).

Figure 4: Semantic net for theorem AF7

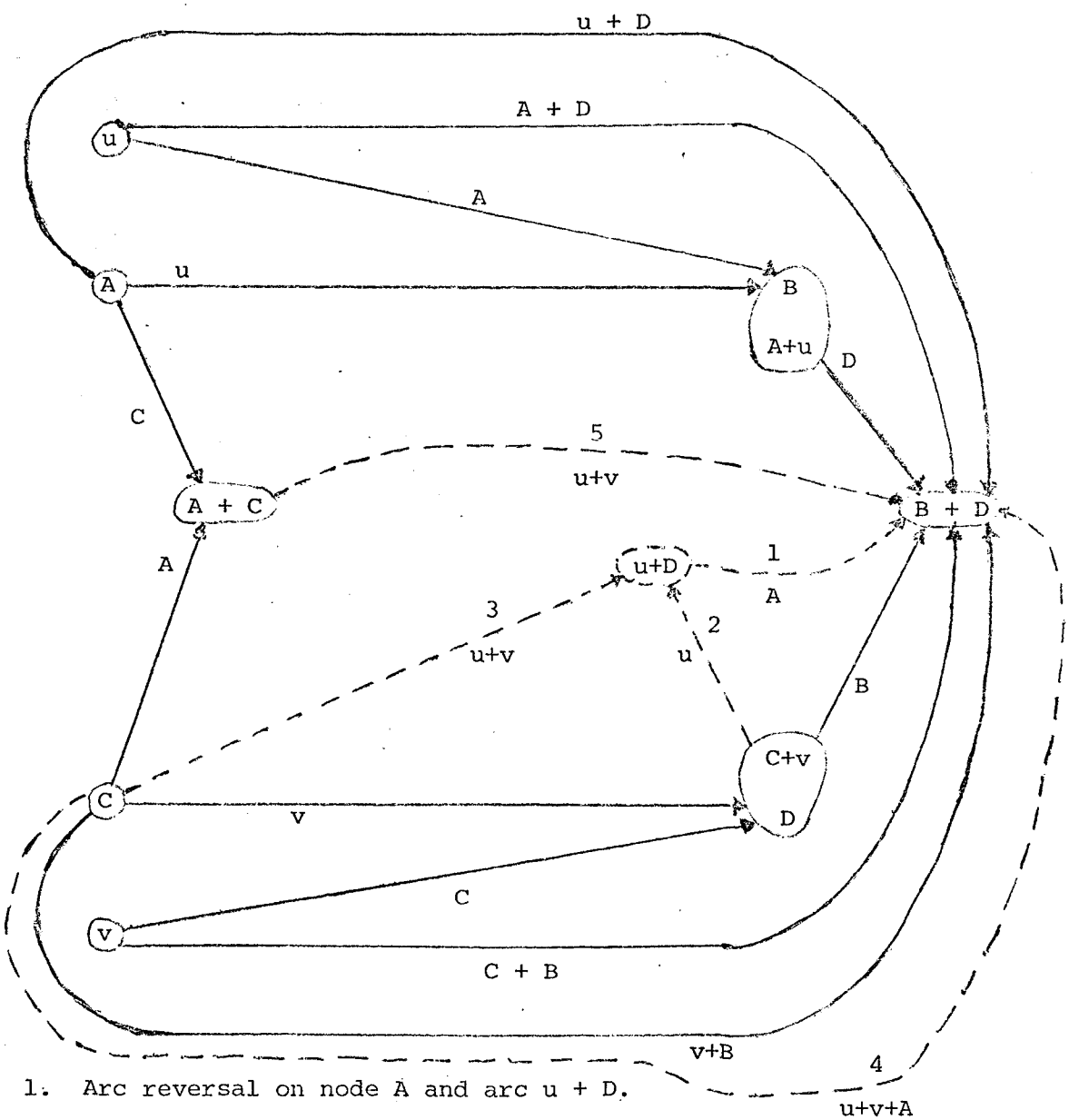


u is B ∸ A

where

v is D ∸ C

Figure 5: Semantic net for AF7 with arc reversal



1. Arc reversal on node A and arc u + D.
2. Node comparison on nodes u + D and D.
3. Chaining on arcs v and 2.
4. Chaining on arcs 1 and 3.
5. Chaining on arcs A and 4.

where { u is B ÷ A ; v is D ÷ C }

Then by comparing nodes and chaining an arc from (A + C) to (B + D) is finally produced. Note that if new nodes such as (u + d) could be automatically created only when needed, then a ban on the creation of just such nodes is obviously irrelevant.

The question, of course, remains as to how SUMS could choose just what new nodes to create, that is just what arcs to reverse. For example,in Figure 4 all of the arcs

u + D, A + D, C + B, v + B

would appear to be potential candidates. With this question in mind, let us examine the protocol produced by our theorem prover in Figure 6. The (1) sequent corresponds to the semantic net of Figure 4. The (2) sequent corresponds to the semantic net of Figure 5 just after the u + D has been created. Note that the term u + D has just been created in this sequent, and the production of ■ immediately follows.

Thus, we have answered our question - our theorem-prover gives us a method of creating the new nodes which are necessary in order to prove this theorem. It follows then, that a ban on creating just such nodes is completely irrelevant.

In summary we can say that not only does our theorem prover delete arcs and nodes (i.e. formulas and terms) when they are no longer necessary (For example, the hypotheses of sequents (1) and (2) in Figure 6 are deleted in the succeeding steps), but also it automatically creates the new nodes (i.e. terms) which are necessary in order to prove the theorem.

Figure 6: Proof of theorem AF7

$\rightarrow A \leqslant B \wedge C \leqslant D \supset A + C \leqslant B + D$:$\rightarrow\supset$

$A \leqslant B \wedge C \leqslant D \rightarrow A + C \leqslant B + D$:$\wedge\rightarrow$

$A \leqslant B, C \leqslant D \rightarrow A + C \leqslant B + D$:A6

$\exists u\ A+u = B, C \leqslant D \rightarrow A + C \leqslant B + D$:$\exists\rightarrow$

$A + u = B, C \leqslant D \rightarrow A + C \leqslant B + D$:A6

$A + u = B, \exists v\ C + v = D \rightarrow A + C \leqslant B + D$:$\exists\rightarrow$

(1) $A + u = B, C + v = D \rightarrow A + C \leqslant B + D$:$=\rightarrow$

$\downarrow$

$C + v = D \rightarrow A + C \leqslant (A + u) + D$:A21

(2) $C + v = D \rightarrow C \leqslant u + D$:$=\rightarrow$

$\downarrow$

$\rightarrow C \leqslant u + (C + v)$:A21

(3) $\rightarrow o \leqslant u + v$

.
.
.
$\downarrow$

(1) Corresponds to the semantic net of Figure 4.

(2) Corresponds to the semantic net of Figure 5, after adding the u + D node by arc node reversal.

(3) $o \leqslant u + v$ is now the only formulae in the sequent.

Note: This protocol is a slight permutation of the one actually obtained by our theorem prover.

5. Implementation

The short implementation time (17 days) is due to the use of our interpreter for mathematical symbols, which provided a uniform method of allowing our items of mathematical knowledge to interact with the sub-formulas of the theorem being proven. This allowed us to code small numbers of related items together as a module which was independent of any other items in the system, and independent of any particular sub-formulae of the theorem being proven.

Our interpreter for mathematical symbols is implemented in 342 lines of LISP [18] source code. The items of mathematical knowledge are encoded as either schemata of our mathematical language or as LISP functions. The logical items are implemented in 531 lines of LISP source code and the arithmetic items, with the exception of a few definitions encoded as schemata, are implemented in 336 lines of LISP source code.

In the case of items implemented as schemata, our mathematical interpreter, using its pattern matching facilities, works more or less as described in section 2. But in the case of an item encoded in a LISP function the interpreter passes the expression it is evaluating to that LISP function, and expects an equivalent, or equal, expression to be returned.

For example, in Figure 7 we display the LISP function NUMIN which encodes the four items: A40, A41, A42, A43, for the funny minus symbol: $\dot{-}$. Note that NUMIN returns either the result of applying one of these items, or the expression it was given if none of the items can be applied.

Figure 7: LISP function encoding items A40, A42, A43.

```
(DEFPROP NUMIN (LAMBDA (X) ( PROG (X1 X2 Y)
                          ! X has form X1 ÷ X2
(SETQ Y (PCANCEL (CDR X)))
(SETQ X1 (CAR Y))
(SETQ X2 (CADR Y))

(COND((EQUAL X1 @ (ZERO)) (RETURN X1))
                          ! A40: o ÷ a = o
((EQUAL X2 @ (ZERO)) (RETURN X1))
                        ! A41:  a ÷ o = a
((ATOM X1) (GO N4))
((EQ (CAR X1)@MINUS) (RETURN (LOGEVAL
(LIST @MINUS (CADR X1) (LIST @PLUS (CADDR X1) X2))  ))  )
)                                       ! A42:  (a ÷ b) ÷ c = a ÷ (b + c)
N4 (RETURN (COND ((EQ Y (CDR X)) X)
                                      ! return input X
(T (CONS @MINUS Y))  ))
)) EXPR)  ! A44
```

Just as an example of how modular the theorem prover is, notice that just before returning to the mathematical interpreter the result of applying the item A43; an explicit recursive call passing that result to the interpreter is made:

(LOGEVAL (LIST @ MINUS ...))

Since this result begins with the minus symbol a recursive call to NUMIN is also made. The LISP function NUMIN is not directly accessed, but only indirectly via the minus symbol. This is a general property of our theorem prover: All items are accessed via mathematical symbols, never through the LISP code. This makes our theorem prover extremely modular in that items or groups of items encoded in a LISP function may be added or deleted without ever affecting the rest of the theorem prover. For example, further items for the minus symbol could be added without ever changing the NUMIN function.

The run time structure of the theorem prover is as follows: (1K is 2^{10} words of core memory).

LISP interpreter:	9K
Mathematical Interpreter and Logical items:	6K
Arithmetic items:	2K
The 87 test theorems:	2K
Free cells:	11K
	30K

Most of the LISP functions are compiled.

6. Conclusion

We have compared an arithmetic theorem prover based on list notation and truth value preserving transformations with an arithmetic theorem prover based on semantic nets and forward chaining, and have found the first to be superior. We have also given an explanation to account for this result. This provides reasonable evidence that semantic nets and forward chaining are not very useful for proving theorems in this domain. It also supports the view that list-notation and truth value preserving transformations are useful for proving theorems in elementary number theory.

IV Extensible Deductive Knowledge System

1. Introduction

Contemporary research, such as that described in Chapters II and II, towards the automation of mathematical reasoning ignores certain fundamental problems. For although great improvements have been made in the construction of theorem provers for particular mathematical domains such as pure LISP, logic, arithmetic, set theory, topology, analysis and elementary algebra, research along these lines does not in itself explain the mathematical ability to create improved deductive systems. In particular, such research provides no explanation as to how domain dependent mathematical knowledge is acquired or even justified.

We believe that the ability of a mathematical system to improve its deductive capabilities by the acquisition of more sophisticated mathematical techniques is a prerequisite of realistic mathematical reasoning. In particular, we argue that significant improvement in deductive ability of automatic theorem provers presupposes the construction of deductive systems which make use of domain-dependent mathematical knowledge and which can justify the results of using such knowledge in terms of the axioms of the given domain.

In the remainder of this Chapter we will analyze these problems, and try to outline a research programme aimed at solving these problems.

2. Analysis of Research Objectives

We first describe two possible objectives for research on the

automation of deductive reasoning:

1. The first objective is to construct a program to determine if a sentence can be proven from the axioms of a theory.
2. The second objective is to construct a program to explain why, in terms of the given axioms of a theory, a sentence can be proven from those axioms.

Consider for example a theory consisting of the following axioms:

A1: $(x+y)+z = x+(y+z)$ associativity

A2: $x+y = y+x$ commutativity

A3: $z+y = x+z \leftrightarrow y=z$ cancellation

We could build a very efficient theorem prover for this theory, which instead of using those axioms to prove theorems, uses the following procedure:

P1: Cancel like variables on opposite sides of the equality sign.

with the strategic information that whenever P1 is used the old equation is to be erased.

Such a theorem prover would satisfy objective 1 because this procedure is essentially a derived rule of inference of the theory consisting of axioms A1, A2 and A3.

For example, a proof that $a+(b+c) = c+(b+a)$ is a theorem using this procedure could be obtained as follows:

```
a+(b+c) = c+(b+a)
        ↓              by P1 (cancel a)
   b+c  = c+b
        ↓              by P1 (cancel b)
     c  = c
        ↓              by logic
```

This theorem prover does not satisfy objective 2 because the inference steps in this proof are not justified in terms of the primitive axioms of the theory, but only in terms of P1. However, a theorem prover for this theory which also satisfies objective 2 could be obtained by substituting equals for equals using the axioms A1, A2, and A3. In this case a proof of a+(b+a) = c+(b+a) would be:

```
a+(b+c) = c+(b+a)
        ↓              by A1
a+(b+c) = (c+b)+a
        ↓              by A2
a+(b+c) = a+(c+b)
        ↓              by A3
   b+c  = c+b
        ↓              by A2
   b+c  = b+c
        ↓              by logic
```

Now it is easy to verify that a deductive system which embodies extra mathematical knowledge in terms of lemmas, procedures like P1

and the strategic knowledge as to how to use those lemmas and procedures, has the potential for being more efficient at proving theorems than systems which do not. For example, the first theorem prover we described which embodies the procedure P1, proves a+(b+c) = c+(b+a) without any search. On the other hand, the second theorem prover which uses only the axioms A1, A2, and A3 involves search due to superfulous applications of the axioms. For example, all of the following seven sentences, six of which are superfulous, are obtainable from the theorem a+(b+c) = c+(b+a) by a single application of one of the axioms:

(a+b)+c = c+(b+a)	by A1
a+(b+c) = (c+b)+a	by A1
(b+c)+a = c+(b+a)	by A2
a+(c+b) = c+(b+a)	by A2
a+(b+c) = (b+a)+c	by A2
a+(b+c) = c+(a+b)	by A2
x+(a+(b+c)) = x+(c+b+a))	by A3

Note that if the second theorem prover had some fairly sophisticated search strategic knowledge which restricted the use of the associativity and commutativity axioms so as to directly lead to the use of the cancellation axiom, then, there might very well have been no search. The problem with such strategies in a more complex theorem proving system, is that out of all the many possible strategies how does the theorem prover know which one to apply to which expression? Suppose for each strategy we had some very efficient method of testing whether it should be used on the given

expression, then it follows that our problem would be solved. The question remains as to just what are the efficient methods? The answer is that these methods are simply items of extra mathematical knowledge. For example in the case of the sophisticated search strategy that we gave for the second theorem prover, an efficient method to determine whether it should be used is simply the procedure P1.

In the remainder of this paper we shall continue to speak of items rather than complex strategies for what is important is not the mere existence of various strategies but rather the methods of determining when a particular strategy should be used. Since every search space including the one produced by adding various items, involves some strategic considerations, we shall reserve the word 'strategy' for this purpose. However, these strategies will not be very complex and will have simple, even obvious methods of determining when they should be used. An example of such a strategy was the erase strategy used in conjunction with P1.

Given then, that the most efficient deductive systems embody extra mathematical knowledge such as the procedure P1; and that while such systems trivially satisfy objective 1, a certain amount of extra effort must be made if such systems are also to satisfy objective 2; one may well wonder if there is any purpose at all in requiring objective 2 to be satisfied. That extra programming effort must be made in order for such systems to satisfy objective 2 can be seen from the fact that extra mathematical knowledge such as lemmas and procedures such as P1 would have to be justified in some sense from

the axioms of the theory. For example, lemmas could be justified by proving them using only the axioms of the theory, but what about procedures written in LISP or machine code? We leave this technical question until section 2 and now state why deductive systems should satisfy objective 2 in the first place.

The reason is this: any theorem prover not continually able to increase it's mathematical knowledge, such as lemmas, procedures, and strategies will quickly reach a plateau in the level of difficulty of theorems it can prove, after which a combinatorial explosion must occur in it's search space.

Consider, for example, some logical system with equality, such as resolution with paramodulation; with a two sorted theory consisting of Tarski's axioms of real numbers and Peano's axioms of number theory. A proof procedure based on this system would have little trouble proving:

$$(a+b)+c = c+(b+a)$$

but would find:

$$\exists x\, \exists y\, \exists z\, \exists u\, \exists v\, \forall n\, \left(\sum_{k=1}^{n} k^3 = x \cdot n^4 + y \cdot n^3 + z \cdot n^2 + u \cdot n + v\right)$$

very difficult indeed.

On the other hand, it is not difficult to implement a more knowledgeable algebra-number thoery theorem prover, embodying about 70 lemmas, strategies, and procedures which could easily prove the above theorem. In fact our algebra number-theory system [28] proves this theorem in about 24 seconds. Of course, since this theorem prover cannot itself further increase its mathematical knowledge, it too soon reaches a plateau in the level of difficulty of theorems it

can prove. For example, it is unable to prove the theorem:

$$\forall p\ \exists X\ \forall n\ \sum_{k=1}^{n} k^{p} = \sum_{j=0}^{p+1} (Xj)\cdot n^{j}$$

Having now exemplified the fact that a mathematical system must be able to increase its mathematical knowledge, it is easy to see why its deductive component must satisfy objective 2. For consider, if objective 2 were not required to be satisfied how would our program itself be able to tell if the mathematical knowledge it was adding to its deductive component, were correct or even consistent with the original axioms of the theory. For example, in the theory consisting of axioms A1, A2, and A3, what would there be to stop a procedure such as P2 from being acquired:

P2: Cancel unlike variables on opposite sides of the equality sign.

Note that whereas P1 was essentially a derived rule of inference of this theory, P2 is not. For whereas a+b = a+b is true in this fragment of arithmetic, applying P2 to it gives the sentence a = b (i.e. $\forall a \forall b\ a = b$) which is false in arithmetic.

In summary, the goal of our research programme, in respect to deductive reasoning, is to construct deductive systems which

(1) make use of extra mathematical knowledge (eventually to be created by other components of the mathematical system). (That is to make use of efficient methods of calculating which strategies should be used on which expressions.)

and

(2) satisfy objective 2.

For example, we would like to have a theorem prover as knowledgeable, and hence efficient, as our algebra-number theory program, but which could justify it's proofs in terms of the Peano and Tarski axioms.

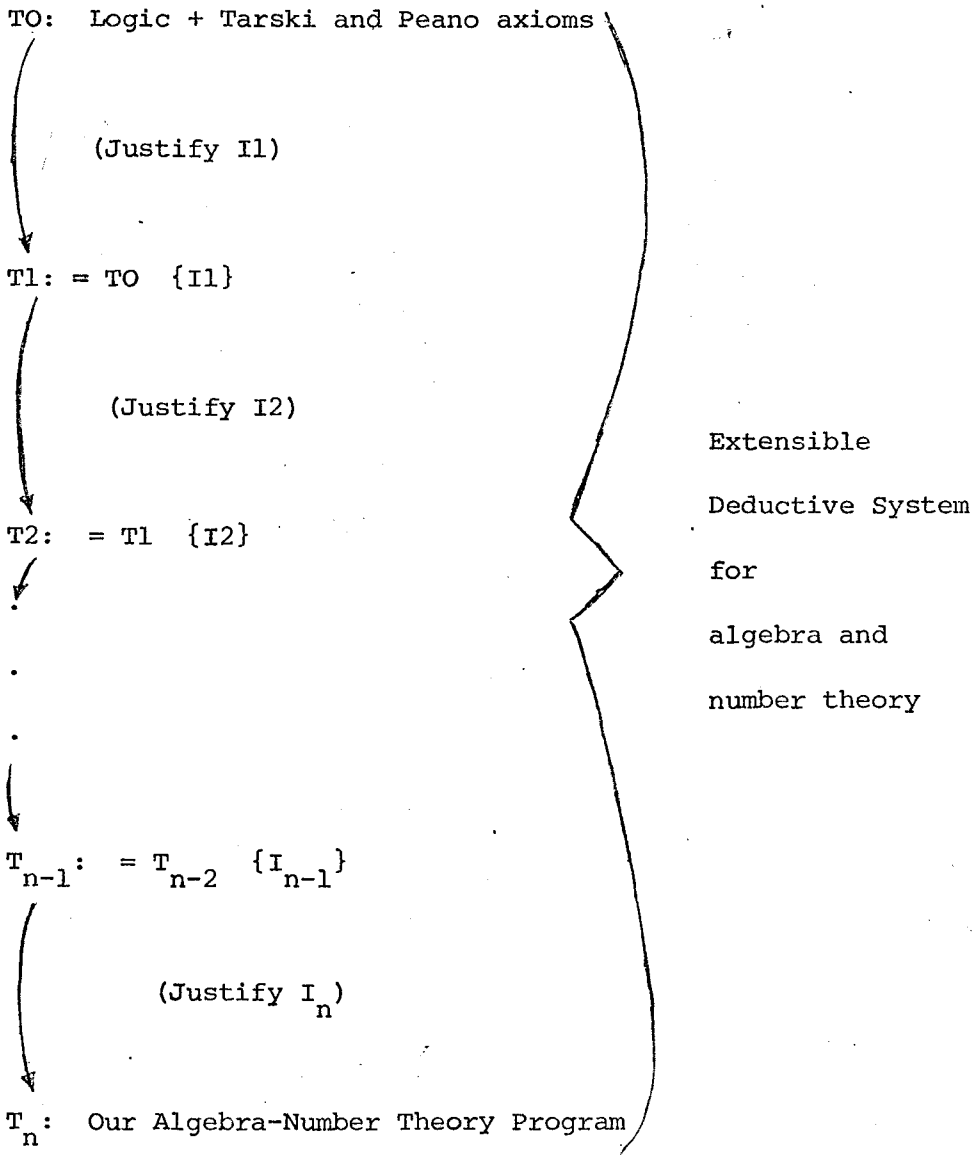
3. Definition of an Extensible Deductive System

How is it possible to construct deductive systems which use extra mathematical knowledge, and yet which satisfy objective 2? We intend to realize this goal by constructing extensible deductive systems:

An extensible deductive system is a theorem prover which starting from some system of axioms justifies more and more items of extra mathematical knowledge such as lemmas and procedures, one after another, creating ever more knowledgeable theorem provers. That is, after each such item is justified, it itself is organized into the deductive system, and is used in justifying further items.

For example, an extensible deductive system resulting in our algebra-number theory theorem prover could be obtained as follows: (see Fig. 1) let $I_1 \ldots I_n$ be all the items of extra mathematical knowledge in our algebra-number theory program. Each Ij will be representable in our mathematical language and hence by definition is a lemma, not a procedure. However, for the purposes of justification procedures are easily represented in our mathematical language. For example, the procedure P1 could be represented as being the lemma which is the conjunction of the axioms A1, A2, and A3. Let the theorem prover T_{j+1} be obtained from the theorem prover T_j by organizing into the system the item I_{j+1} after justifying I_{j+1} using T_j.

Figure 1: Example of Extensible Deductive System



It is important to understand that organizing an item into the system means a lot more than merely adding another sentence to some data base. In particular it means that knowledge about how and when the item is to be used is created and added to the system. For example, in the case of the lemma version of the procedure P1 (i.e. the conjunction of axioms A1, A2, and A3) what we want to create and add is an efficient LISP function to do the cancellation along with relevant information expressed in LISP as to when the system should attempt to use this function. Note that by allowing the full power of a language like LISP, it is not very difficult to represent strategic knowledge as to how to use items. Then by letting T_o be a theorem prover for logic including equality with the Tarski and Peano axioms, it follows that T_n will be our algebra-number theory program. Furthermore, the spectrum of theorem provers $<T_o \ldots T_n>$ will form an extensible deductive system resulting in T_n. That is any further theorem proven by T_n will be justifiable, via $T_1 \ldots T_{n-1}$, eventually in terms of the primitive axioms of T_o.

In order to experimentally test these ideas, last year we implemented a prototype extensible deductive system* for a small portion of the set theory given in Quine's book Set Theory and its Logic [1]. The initial theorem prover T_o of this extensible system consisted of a sequent calculus with equality, and about 30 axioms and definitions. A few of these definitions are listed below: (Note that $\{u:\phi u\}$ is the abstract of all things: u such that ϕu holds; $\{x\}$, $\{xy\}$, $<xy>$ are the unit-set,pairset, and ordered pair respectively; and $\{<xy>:\phi xy\}$ is the

* This system is described in detail in Chapter V.

abstract for all ordered pairs such that ϕxy holds.)

D1: $x \varepsilon\{u:\phi u\} \leftrightarrow \phi x$

D2: $\{x\} = \{z:z=x\}$

D3: $\{x,y\} = \{z:z=x \vee z=y\}$

D4: $\langle x,y\rangle = \{\{x\}\ \{xy\}\}$

D5: $\{\langle xy\rangle:\phi xy\} = \{z: \exists x \exists y\ \langle xy\rangle = z \wedge \phi xy\}$

D6: $x=y \leftrightarrow \forall z\ z\varepsilon x \leftrightarrow z\varepsilon y$

Succeeding theorem provers T_{j+1} were obtained by organizing into the system certain given lemmas I_{j+1} that had been proven using the preceding theorem prover T_j. Such items in the system were lemmas not procedures. Furthermore, the actual process of organizing was not automated. Instead the organized LISP representation of each lemma was initially given, but not used until automatically activated by a proof of that lemma. These lemmas are used in a manner similar to definitions by replacing what matches the left side of the equivalence by that instance of the right side. Furthermore, when several items might be applied to a sentence, the system prefers to use the least primitive item, usually the one justified last. Thus, definitions are used only if no other item could be used. A few of the lemmas in our set theory system are given below:

I1: $x = x \leftrightarrow$ ■

I2: $\{x\} = \{y\} \leftrightarrow x = y$

I3: $\{xy\} = \{z\} \leftrightarrow x = z \wedge y = z$

I4: $\{z\} = \{zy\} \leftrightarrow z = x \wedge z = y$

I5: $\{xy\} = \{uv\} \leftrightarrow (x = u \wedge y = v) \vee (x = v \wedge y = u)$

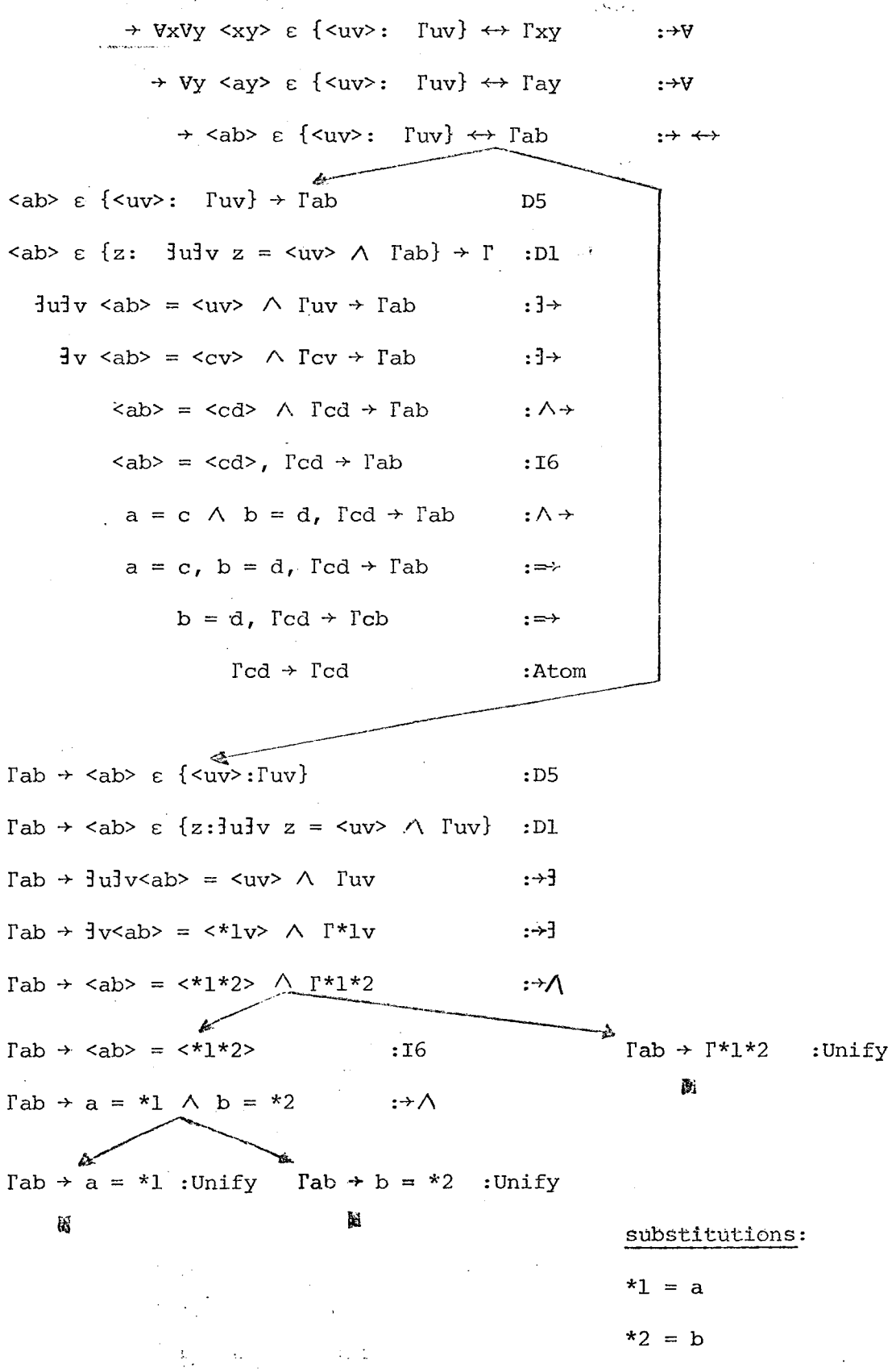
I6: $\langle xy\rangle = \langle uv\rangle \leftrightarrow x = u \wedge y = v$

I7: $\langle xy\rangle \in \{\langle uv\rangle : \phi uv\} \leftrightarrow \phi xy$

Consider now the proof of lemma I7 using the theorem prover T6. From Figure 2 we see that two of the inference steps in this proof were obtained by using the lemma I6. Thus this proof is not immediately in terms of the primitive axioms of this set theory. However, since I6 was proven from T5 using I2, I3, I4, and I5, and since each of these lemmas was proven using only T1, and since I1 was proven from the primitive axiom, namely T0, it follows that the lemma I7 has actually been justified in terms of the primitive axiom of this theory.

Note that items of extra mathematical knowledge such as lemmas I1, ..., I7 play an indispensable rôle in the deductive process not only because they cut down the sheer length of the proof of a theorem, but also because they cut down the possibilities of interaction between various subformulas of a theorem. For example, in the proof of I7 given in Figure 2, if the lemma I6 had not been used then the theorem prover would have had to show that various instances of I6 held; namely that $\langle ab\rangle = \langle cd\rangle$ was equivalent to $a = c \wedge b = d$ and that $\langle ab\rangle = \langle *_1 *_2\rangle$ was equivalent to $a = *_1 \wedge b = *_2$. Not only does this make the proof of I7 much longer but also the proofs of those instances of I6 must be carried out in the presence of sentences such as: Γab, Γcd, $\Gamma *_1 *_2$. If Γ were a large complex sentence, it is easy to see that there would be many possibilities of interaction between it and the intermediate formulas produced when proving the instances of I6. Continuing in this fashion, if the use of further lemmas is disallowed, say for example lemmas I2, I3, I4, and I5

Figure 2: Proof of lemma I7 using I6



substitutions:

*1 = a

*2 = b

which were used in proving I6 then the resulting proof must be even longer, and there become more and more possibilities of interaction, eventually creating a combinatorial explosion. On the other hand, by using items of extra mathematical knowledge such as the lemmas I1...I7, the proofs of further theorems in elementary set theory do not appear to be any longer,or any more difficult to obtain than the proofs of earlier lemmas.

For these reasons, it is plausible to suggest that by using items of extra mathematical knowledge in extensible deductive systems, that efficient theorem provers satisfying objective 2 can be constructed.

4. The Nature of Mathematical Knowledge

Given that items of extra mathematical knowledge are an essential component of extensible deductive systems, the question arises as to just what are these items? At any given stage of an extensible deductive system the items are simply the sentences or things which represent sentences of our formal language, which have previously been justified, organized as to be easily accessed when relevant to proving further sentences.

4.1 Accessing Items

Consider for example the applications of the items I6 in the proof of I7 given in Figure 2. Out of all the many items available, how is it possible for the theorem prover to know that I6 and no other item is to be used on the sequent below?

$$\langle ab\rangle = \langle cd\rangle, \Gamma cd \rightarrow \Gamma ab$$

How does the theorem prover know not to use the item I1 or the

item I2? Thus obtaining respectively:

$\forall x\ x = x,\ \langle ab\rangle = \langle cd\rangle,\ \Gamma cd \rightarrow \Gamma ab$

or $\forall x \forall y\ \{x\} = \{y\} \leftrightarrow x = y,\ \langle ab\rangle = \langle cd\rangle,\ \Gamma cd \rightarrow \Gamma ab$?

The answer is that item I6 and the sentence $\langle ab\rangle = \langle cd\rangle$ contain many of the same non variable symbols, namely equality and ordered pairs; and thus are about the same thing, namely the equality of ordered pairs. The reason that other items are not accessed, is because they do not share with $\langle ab\rangle = \langle cd\rangle$ as many symbols as does I6, and thus are probably not as relevant.

In our set theory program, the mechanics of accessing I6 were more or less as follows: First; $\langle ab\rangle = \langle cd\rangle$ and the left side of I6 were observed to have the equality sign as their outermost symbol, with ordered pairs in the argument positions. This caused the accessing of I6 producing the sequent:

$\forall x \forall y \forall u \forall v \langle xy\rangle = \langle uv\rangle \leftrightarrow (x = u \wedge y = v),\ \langle ab\rangle = \langle cd\rangle, \Gamma cd \rightarrow \Gamma ab$

However, since by instantiating the variables x, y, u, v, by a, b, c, d, respectively, and since $\langle ab\rangle = \langle cd\rangle$ is true in this sequent it follows that $a = c \wedge b = d$ is true. Note then that $\langle ab\rangle = \langle cd\rangle$ is equivalent to $a = c \wedge b = d$. Therefore we delete $\langle ab\rangle = \langle cd\rangle$ obtaining the new sequent:

$\forall x \forall y \forall u \forall v\ \langle xy\rangle = \langle uv\rangle \leftrightarrow x = u \wedge y = v,\ a = c \wedge b = d,\ \Gamma cd \rightarrow \Gamma ab$

And finally since the reason for accessing I6 no longer applies, and since it can always be re-accessed if needed again, it is eliminated from the sequent giving:

$a = c \wedge b = d,\ \Gamma cd \rightarrow \Gamma ab$

Thus, items are accessed by the non-variable symbols they have

in common with the sentence being proven.

However, this is not quite right, for consider: if when seeing <ab> = <cd> the theorem prover accesses all items with an equality sign, it would access all of I1,...,I6 because equality is a very common symbol. On the other hand if the theorem prover accessed all items with an ordered pair sign only I6 would be accessed. The point is that the ordered pair sign occurs less often in the items than does the equality sign, so that should be the sign it is accessed by. Thus in general we can say that items are accessed by the less common non-variable symbols that they share with the sentence being proven. For example, a simple LISP implementation would be to store items on the property list of their least most common non-variable symbol.

4.2 Organization of Equivalences

We note that all lemmas may be viewed as being equivalences, for if S is a lemma then $\square \leftrightarrow S$ is an equivalence.

In section 4.1 we remarked that the sentence <ab> = <cd> was deleted from the given sequent because a = c∧b = d was also in the sequent and <ab> = <cd> is equivalent to a = c∧b = d. This deletion is an instance of a powerful principle: If two equivalent sentences occur in a sequent then delete one of them but re-write the items which could be accessed by it, so as to be accessed by the other instead. We call the sentence which was not deleted the preferred form of the two. In the case of <ab> = <cd> there were no obvious items to be re-written, but consider the sequent:

(POS a), a>o → φa

with the items: $x>o \leftrightarrow (POS\ X)$

$x>o \rightarrow |x| = x$

If a>o is to be deleted by the first item then the second item had better be re-written as:

$(POS\ x) \rightarrow |x| = x$

Furthermore, if an item (e.g. I6) causes two equivalent sentences to occur in a sequent then decide which one is to be deleted and re-write the items which could be accessed by it so as to be accessed by the other instead. Note that any item of the form: $A \leftrightarrow B$ or $(A = B)$ which is accessed by an instance of A where $B\theta$ is the preferred form will invariably cause the equivalent sentences $A\theta$ and $B\theta$ ($\Gamma(A\theta)$ and $\Gamma(B\theta)$) to appear in some sequent, followed by the deletion of $A\theta$ leaving $B\theta$. This is the reason for the unidirectional nature, that is the replacement of $A\theta$ by the preferred form $B\theta$, of items such as: I1,...,I7.

Of course, this unidirectional nature, that is this deletion of $A\theta$ is only justified to the extent that all items including initial logical axioms, which would be accessed by $A\theta$ in proving the sequent are now accessible by $B\theta$. And research on this problem has, of course, been done; for example, the theory of definition and the more recent work of Lankford [16] and for simple algebraic theories without logical rules. However, the problem is probably too complex to solve for more difficult theories such as analysis or set theory. Our point is that we can structure our theories so that the $A\theta$'s need rarely be kept, and that we can implement theorem provers which can know, in a practical sense, when such $A\theta$'s should

be kept.

Consider for example a theory consisting of the axioms:

A1: $(x+y)+z = x+(y+z)$

A2: $x+y = y+x$

A3: $x+y = x+z \leftrightarrow y = z$

which are used left to right in a unidirectional manner (A2 being used only when all variables in the term bound to x alphabetically precede all variables in the term bound to y). Then $a+(b+c) = b+(a+c)$ will not be provable because none of the axioms can be applied. However, noting that A3 should have the effect of cancelling like variables on opposite sides of the equality sign:

$$(\ldots+ x + \ldots = \ldots+ x + \ldots) \leftrightarrow (\ldots+ \ldots = \ldots+ \ldots)$$

purely for the purpose of trying to apply A3 in a unidirectional manner we could allow A1 and A2 to be applied in a bi-directional manner, with the stipulation that when A3 is applied, then all the superfulous sentences produced would be immediately deleted:

$\rightarrow a+(b+c) = b+(a+c)$:A2

$\downarrow$

$\rightarrow a+(b+c) = (a+c)+b,\ a+(b+c) = b+(a+c)$:A1

$\downarrow$

$\rightarrow a+(b+c) = a+(c+b),\ a+(b+c) = (a+c)+b,\ a+(b+c) = b+(a+c)$:A3

$\downarrow$

$b+c = c+b$:A3

$\downarrow$

$b+c = b+c$:A2

$\downarrow$

■

As a second example consider the use of the logical lemmas:

L1: $(\forall x,\ x = t \supset \phi x) \leftrightarrow \phi t$

L2: $(\exists x,\ x = t \wedge \phi x) \leftrightarrow \phi t$

in the deductions shown in Figure 3.

Figure 3

(1) $\forall x\ x\varepsilon\{A\ B\} \rightarrow \phi x$:D2

$\downarrow$

$\forall x\ x\varepsilon\{u:\ u = A \vee u = B\} \rightarrow \phi x$:D1

$\downarrow$

$\forall x\ ((x = A \vee x = B) \rightarrow \phi x)$:Logic(CNF)

$\downarrow$

$\forall x[(x = A \rightarrow \phi x) \wedge (x = B \rightarrow \phi x)]$:Logic

$\downarrow$

$(\forall x\ x = A \rightarrow \phi x) \wedge (\forall x\ x = B \rightarrow \phi x)$:L1

$\downarrow$

$\phi A \wedge \phi B$

(2) $\exists x\ x\varepsilon\{A\ B\} \wedge \phi x$:D2

$\downarrow$

$\exists x\ x\varepsilon\{u:\ u = A \vee u = B\} \wedge \phi x$:D1

$\downarrow$

$\exists x((x = A \vee x = B) \wedge \phi x)$:Logic(DNF)

$\downarrow$

$\exists x\ (x = A \wedge \phi x) \vee (x = B \wedge \phi x)$:Logic

$\downarrow$

$(\exists x\ x = A \wedge \phi x) \vee (\exists x\ x = B \wedge \phi x)$:L2

$\downarrow$

$\phi A \vee \phi B$

In step three of the first proof the sentence is being put into conjunctive normal form by an application of a logical distributive rule, whereas in the third step of the second proof the sentence is being put into disjunctive normal form. Thus although conjunctive normal form may be regarded as the preferred form in which to put the sentence being proven (for example a sequent calculus essentially puts the sentence being proven into conjunctive normal form); we see from the second proof that in order to apply lemma L2 to some sentence beginning with an existential quantifier, that the sentence within the scope of that quantifier should be put into disjunctive normal form.

A third example is the use of axioms such as:

A1: $\phi \in V \leftrightarrow$ ▨

A2: $\forall x \forall y \{xy\} \in V \leftrightarrow$ ▨

in our set theory program. Although '▨' is the preferred form, such axioms are applied in the opposite directions, when the theorem prover fails to find a proof using its other techniques.

Thus from these examples, we conclude that items are applied in a unidirectional manner, that is deleting $A\Theta$, unless there is a specific reason (e.g. in order to apply another lemma or if no proof has been found using other techniques) not to do so.

4.3 Preferred Forms

Given that equivalent sentences should have a preferred form, the question arises as to which sentence is to be the preferred form. The answer is simple, that the preferred form should be the one which minimizes the accessing problems for the items of the theory.

For example, in our algebra-number theory system the preferred form of the equivalences:

E1: $x+o = x$

E2: $(\exists x\ x = t \wedge \phi x) \leftrightarrow \phi t$

E3: $(\forall n\ A \cdot n^2 + B \cdot n + C = D \cdot n^2 + E \cdot n + F) \leftrightarrow A = D \wedge B = E \wedge C = F$

E4: $\sum_{k=1}^{n+1} k = (\sum_{k=1}^{n} k) + n+1$

are respectively:

x

ϕt

$A = D \wedge B = E \wedge C = F$

n

$(\Sigma\ k) + n + 1;$

$k=1$

For the first three lemmas it is plausible that the accessing problems are minimized because (1) the preferred sentence is syntactically less complex than the other sentence, and (2) because it is not very difficult to recognize when sentences are in the non-preferred form, that is to tell when the items should be used. In the case of the last item, a recursive definition, it is plausible that the accessing problems are being minimized because it puts Σ expressions into a particular form, and again because it is not difficult to recognize the unpreferred form.

In our set theory systems for example the preferred form for lemmas such as I1,...,I7 and the contextual definition D1 is the sentence on the right side of each equivalence, for about the same reasons as in our algebra number theory program. However, the treatment of definitions such as D2,...,D6 is quite different. For although on the analogy of recursive definitions we might consider the definiens to be the preferred form, we also note the immense difference in syntactic complexity between the definiens and the definiendum. For this reason theorem provers whose lemmas were re-structured so as to apply to the definiens instead of to the definiendum would be very slow. For example, if the non-preferred sentence lemma I2 were re-structured in terms of definiens we would have:

$$[\forall u[\forall z.z\varepsilon u \leftrightarrow z\varepsilon x) \leftrightarrow (\forall w.w\varepsilon u \leftrightarrow w\varepsilon x)]]$$

which would be much harder to recognize than the definiendum form of the sentence

$$\{x\} = \{y\}$$

Of course, just as in recursive definitions we would not want the definiendum to be the preferred form because it is very difficult to recognize when something was the definiens.

For this reason, we believe that the handling of definitions should be done as was done in our set theory program as follows: Let the definiens be the preferred form, but try to apply all other lemmas which work off of that form before replacing any definiendum by its definiens. For example, when trying to prove the sentence

$$\langle ab\rangle = \langle cd\rangle \wedge \Gamma ab \rightarrow \Gamma cd$$

don't start applying definitions getting

$$\{\{a\}\{ab\}\} = \{\{c\}\{cd\}\} \wedge \Gamma ab \rightarrow \Gamma cd$$

but apply all lemmas which work off of ordered pairs first getting:

$$a = c \wedge b = d \wedge \Gamma ab \rightarrow \Gamma cd$$

Thus definitions form a sort of hierarchy. And only if no proof is found by using lemmas accessed by means of symbols at that level should a theorem prover start unravelling definitions trying to obtain a proof at a lower level.

In summary, we can say that basically the mathematical knowledge used in deductive reasoning consists of items which may be viewed as equivalences. That these equivalences have a preferred form which gives rise to a unidirectional use, except insofar as there are specific reasons for avoiding unidirectionality. And, that defined symbols are replaced by their definiens only when a proof cannot be found without doing so.

5. Formal Justification of Mathematical Knowledge

Up till now we have discussed the nature of mathematical knowledge used in deductive reasoning without mentioning how such knowledge is represented. In particular the question arises as to whether such items must always be represented in our mathematical language, or whether they might be represented as procedures in the programming language in which the deductive system was implemented.

At first glance, one might be tempted to argue that of course items must always be represented in our mathematical language, for if they were not how could they be proven from the axioms of the theory? However, this criticism is answered by the fact that such items (i.e.

procedures) do not need to be proven in order to justify their use in a proof. Consider for example the use of the procedure P1 given in section 1 in the following inference step:

$$\rightarrow \phi[a+(b+c) = c+(b+a)]$$

$$\downarrow \quad \text{by P1}$$

$$\rightarrow \phi[b+c = c+b]$$

We do not need to prove the procedure, or rather axiom scheme P1, but only that instance of the scheme which was actually used in this inference step, namely:

$$\phi[b+c = c+b] \rightarrow \phi[a+(b+c) = c+(b+a)]$$

Note that if ϕ is a complex sentence it is probably much easier to prove $\phi[b+c = c+b]$ and then $b+c = c+b \leftrightarrow a+(b+c) = c+(b+a)$ without using P1, than to prove $\phi[a+(b+c) = c+(b+a)]$ without using P1. Thus it is useful to use P1 even if the step must later be justified.

The fact that P1 does not need to be justified, can be generalized to the following principle: what must be justified is the actual proof that is produced, not the deductive system which produced the proof. For example, in the case of people this is quite clear. If a mathematician proves a theorem, and if we wish to check if he actually has proven that theorem, we inspect his proof; we do not look inside his head to see if his brain is consistent! Anyone who disagrees with this principle will probably also assert that all items should be represented in the mathematical language. We now give reasons as to why this restriction would lead to rather inefficient theorem provers.

Consider again the Procedure P1. There are two ways of viewing

it as being in our mathematical language. If we view P1 as being a schema of cancelling lemmas such as:

$$a + b = a + c \leftrightarrow b = c$$

$$a + (b + c) = (c + b) + a \leftrightarrow (b + c) = (c + b)$$

then each such lemma that the theorem prover might use would have to be explicitly stored. Besides taking up much storage it will obviously take much time to find the right scheme to use in any particular case.

If P1 is viewed as being the axioms A1, A2, and A3, given in Section 1, plus knowledge specifying how to use the axioms so as to achieve an effect similar to the procedure P1, then every time P1 is used in the entire search space and not just where P1 is used in the final proof that is found, the system will have to go through the laborious process of simulating P1 by applications of axioms A1, A2, and A3.

In neither case does this compare to the efficiency of P1 as a procedure which simply scans across the equation looking for occurrences of a variable connected by plus signs to opposite sides of the equality sign. Note that in comparison to the latter case, only those applications of P1 which actually appeared in the final proof that is found, would then need to be justified in terms of axioms A1, A2, and A3. All other applications of P1 on all other branches of the search space do not need to go through this laborious process of justification because they do not contribute to the proof.

In summary then we can say that there are symbolic operations

(e.g. P1) preformed in mathematics, whose results can be logically justified, but which themselves cannot reasonably be described in terms of more primitive operations which can be logically justified.

It is for this reason that items of mathematical knowledge generally have both a procedural form especially for their uni-directional use written in an implementation language such as LISP, and a mathematical form written in the mathematical language. Of course if it is not all that necessary that the use of a particular item be very efficient then the mathematical form may just as well serve as the procedural form for that item.

5.1 Procedural forms of a lemma

A simple example of these two forms involving a lemma is the identity axiom of addition. The mathematical form is of course:

$x + o = x$ or rather: (=(+ x (O)) x)

and a procedural form could be the LISP function

```
(LAMBDA (X) (OR (AND (EQ (CAR X) (QUOTE +))
                     (EQ(CAADDR X)(QUOTE O))
                     (CADR X))X))
```

or even that function compiled into machine code. In the following inference step note that $\phi(a)$ should be obtained from $\phi(a + o)$ by use of the procedural form, not the mathematical form, for this is more efficient.

$$\phi(a + c)$$

$$\text{LO} \qquad \downarrow \qquad \text{LO*: } x + O = x$$

$$\phi(a)$$

Note however that the justification given for this inference step

is the mathematical form x + o = x.

5.2 Procedural form of a schema

Items which are not lemmas are not represented in our formal mathematical language, and thus have only a procedural form. For this reason such items are called procedures. For example, a procedural form for P1 written in LISP rather than in English is given in Figure 4. Furthermore, because procedures are not represented, we will take as the justification for using them in an inference step, any particular group of lemmas which could have been used to obtain that inference. For example, from our remarks in section 1 we see that a group of axioms which could be used to justify the procedure P1 is {A1, A2, A3}. This leads to two different kinds of proof steps: the compressed and the uncompressed.

A compressed proof step using the procedure P1 is for example:

$$a + (b + c) = c + (b + a)$$

$$P1 \qquad\qquad \downarrow \qquad\qquad \{A1, A2, A3 \text{ once}\}$$

$$a + b = b + a$$

where each lemma in the group may need to be used zero or more times except where specified (e.g. A3) in justifying that:

$$a + b = b + a \text{ implies } a + (b + c) = c + (b + a).$$

Of course, in the traditional formal sense of 'proof', proofs are restricted to involving proof steps which are justified by use of a single application of one lemma. For this reason there is also the uncompressed form of a proof step involving a procedure. This form is obtained by filling in explicit proof steps, each involving a single lemma from the group of lemmas. For example, an

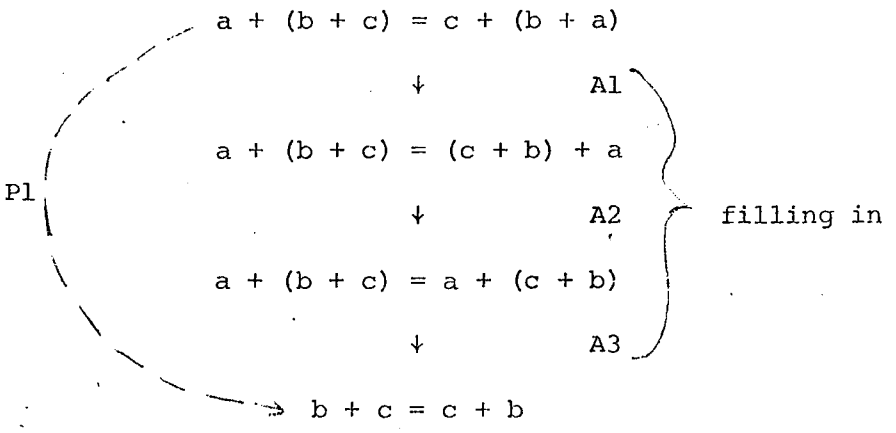
Figure 4

```
      (DEFPROP P1 (LAMBDA(X) (PROG(X1 X2 Y U)
      (SETQ Y(PCANCEL (CDR X)))
.
.
(other items)
.
.
.
N4 (RETURN(COND((EQ Y(CDR X))X) (T(CONS (QUOTE EQUAL)Y))
)) EXPR)

(DEFPROP CYL T SPECIAL)
(DEFPROP PCANCEL (LAMBDA(X) (PROG(X1 CYL X2 U)
    (SETQ U(CAR X))
P1  (COND((NOT(EQ U(SETQ X1(LOGEVAL U)))) (SETQ U X1) (GO P1)))
    (SETQ U(CADR X))
P2  (COND((NOT (EQ U(SETQ CYL(LOGEVAL U)))) (SETQ U CYL) (GO P2)))
    (SETQ X2(PC1 X1))
    (COND((AND(EQ X1(CAR X))(EQ CYL(CADR X))) (RETURN X)) )
    (RETURN(LIST X2 CYL))
)) EXPR)
(DEFPROP PC1 (LAMBDA(X) (PROG(U)
   (COND((OR(ATOM X) (NOT(EQ(CAR X)(QUOTE PLUS))))
            (RETURN(COND( (NOT(EQ CYL(SETQ U(PC2 X CYL))))
                                  (SETQ CYL U)(QUOTE (ZERO)))
                      (T X)   )))
        ((NOT(EQ CYL(SETQ U(PC2(CADR X)CYL))))
              (SETQ CYL U) (RETURN(PC1(CADDR X))))  )
   (SETQ U(PC1 (CADDR X)))
```

```
   (RETURN(COND((EQ U(CADDR X)) X)
                ((EQ(CAR U)(QUOTE ZERO)) (CADR X))
                (T (LIST(QUOTE PLUS)(CADR X)U))    ))
))  EXPR)
(DEFPROP PC2 (LAMBDA(X Y) (PROG(U)
   (COND((OR(ATOM Y)(NOT(EQ(CAR Y)(QUOTE PLUS))))
                (RETURN(COND((EQUAL X Y)(QUOTE (ZERO)))
                             (T Y) )))
         ((EQUAL X(CADR Y)) (RETURN(CADDR Y)))   )
   (SETQ U (PC2 X(CADDR Y)))
   (RETURN(COND((EQ U(CADDR Y)) Y)
                ((EQ(CAR U)(QUOTE ZERO)) (CADR Y))
                (T (LIST(QUOTE PLUS)(CADR Y)U))    ))
))  EXPR
```

uncompressed form of the above proof step is:



Note that b + c = c + b is produced by P1 before filling in even occurs. This filling in of the compressed form producing the uncompressed form may be done by either deducing b + c = c + b from a + (b + c) = c + (b + a) by using only the lemmas in the group, namely A1, A2, and A3, or by having an explicit prestored function for each procedure which fills in the missing steps.

Although, as we have explained, it is only the uncompressed form of proof steps which appear in traditional formal proofs, it is noted that the compressed forms are used quite often in mathematical texts, and have important implications for the abbreviation of proofs.

5.3 Theory of Expressions

Finally, any discussion of the nature of justification would not be complete without mentioning the idea of representing procedures (e.g. P1) in a mathematical language by use of a theory of expressions such as pure LISP. In such a case the group of axioms (e.g. {A1, A2, A3}) which are the justification of an inference step using the procedural form of P1, would be replaced by this mathematical representation of P1. For example, the procedure P1 could have the mathematical

form: (I have given in [29] a correct definition of the meaning function.)

```
P1*  (meaning (list '↔
                    (list '=
                          (cons '+ (Append x1 (cons z y1)))
                          (cons '+ (Append x2 (cons z y2))) )
                    (list '=
                          (cons '+ (Append s1 y1))
                          (cons '+ (Append x2 y2)) )
))
```

Schematically P1* is something like:

$$[(+ \underset{x1}{\ldots} \underset{y1}{z \ldots}) = (+ \underset{x2}{\ldots} \underset{y2}{z \ldots})] \leftrightarrow [+ \underset{x1}{\ldots} \underset{y1}{\ldots}) = (+ \underset{x2}{\ldots} \underset{y2}{\ldots})]$$

where the plus sign of arbitrary arity is defined in terms of a binary plus and zero:

$$(+x_1 \ldots x_n) = (+ x_1 \ (+ x_2 \ldots (+ x_{n-1} x_n) \ldots))$$

$$(+) = o$$

Then by use of induction over expressions P1* could be proven to be a meta theorem of any theory containing axioms A1, A2, and A3. Note that P1* is related to P1 in the same sense as LO* (i.e. 'x + o = x') is related to its procedural form LO. Namely that while P1* and LO* (x + o = x) are sentences and have mathematical meaning, P1 and LO do not.

As much as we like this idea, we point out that justifications using a theory of expressions do not seem to appear in mathematics as much as do the earlier 'group of lemmas' method. For example, if you asked someone why the following inference step was correct in

arithmetic:

$$(a + b) + c = c + (b + a)$$

$$\downarrow$$

$$b + c = c + b$$

he would probably answer: because of the associativity A1, commutativity A2, and cancellation laws A3. It is doubtful if the answer would be: because of P1*.

In summary let us note that we are not claiming that all mathematical results obtained by a deductive system are entirely justified in terms of the primitive axioms of that theory, but rather, that they are only potentially justifiable in the following sense: if the validity of any inference step in a proof is questioned, then the deductive system, will be able to describe how that step was produced (i.e. why the items justifying that step are valid) in terms of more primitive items.

6. Mathematical Reasoning Systems

Having now described what an extensible mathematical theory consists of, namely an extensible deductive system; we are left with the question as to how such systems can be automatically produced? In particular, assuming the extensible deductive system has available for its use a certain number of items, initially only the primitive axioms, how can further items be automatically created and added to the deductive system. How can such items be organized and how can their procedural forms be produced?

We will call any such system which can construct mathematical theories, a mathematical reasoning system. As a first approximation,

we suggest that a mathematical reasoning system consists of a number of highly interacting components, the first of which is an extensible deductive system which has already been described. The other components would be as follows:

(1) An inductive reasoning system which can create new sentences of our mathematical language, and propose that they are theorems.

(2) A notation introduction system, which can introduce new mathematical symbols (for example defined symbols) into our mathematical language, so as to facilitate the deductive processes by minimizing the accessing problems.

(3) A preference analysis system which can choose the preferred form for any given set of equivalent sentences.

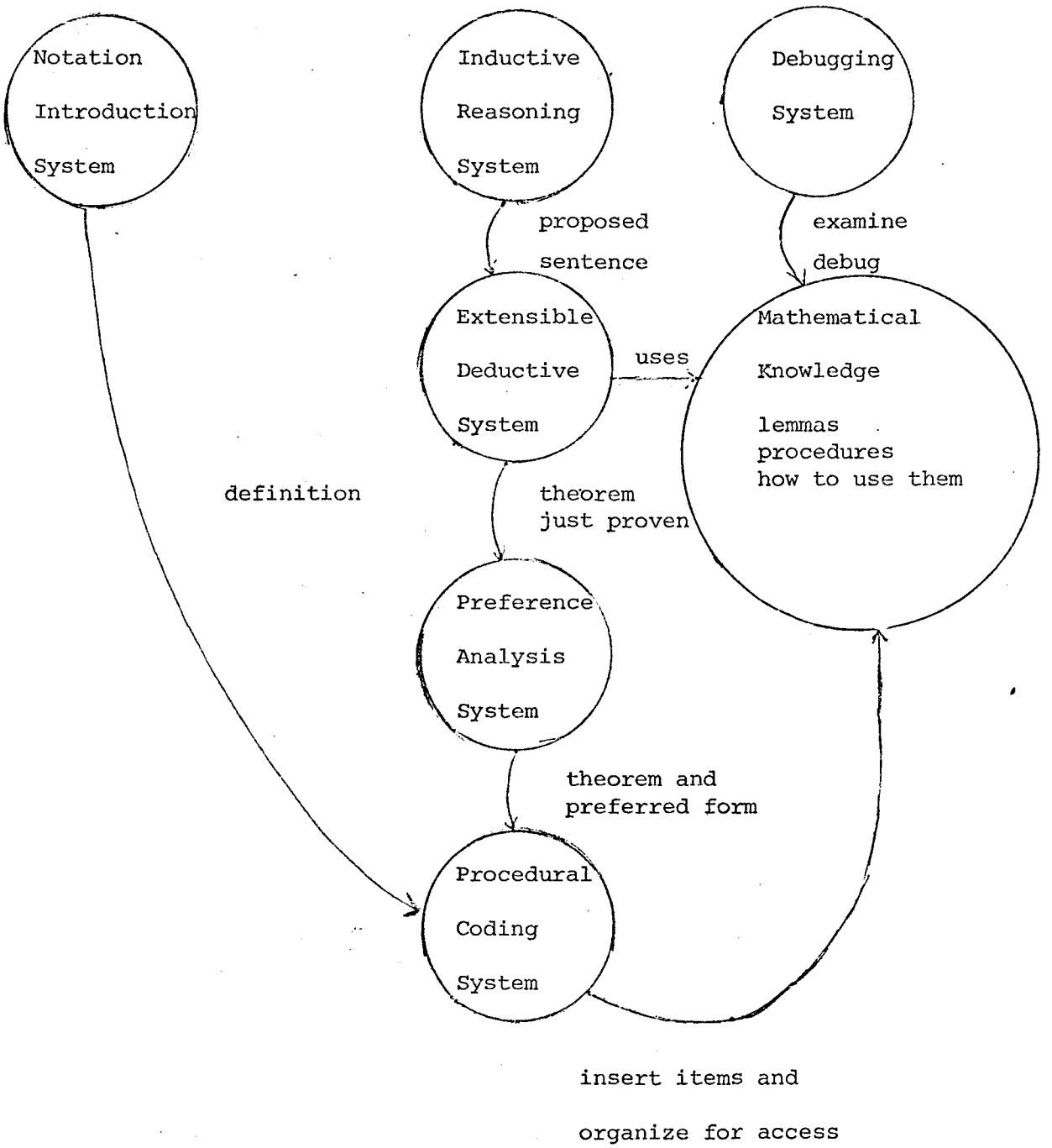
(4) A procedural coding system which can compile an axiom, or group of axioms, into their procedural forms.

(5) A debugging system which can relate failures to obtain justifications to possible errors in the procedural code.

As an example of how these components might interact, consider the problem of automatically constructing the theorem prover T7 from the theorem prover T6 in the extensible set theory system described in section 3. These interactions which are described below are displayed in Figure 5. Note however, that there are many more necessary interactions, than those here displayed or described.

First the inductive reasoning system must create the sentence I7 and propose that it is a theorem. Next the deductive system would then try to prove I7 by using T6. If successful then the preference

Figure 5: A Mathematical Reasoning System



analysis system would decide that Γxy was the preferred form of the two equivalent sub-sentences in I7. The procedural coding system would then construct a procedural form for I7 which would replace sentences of the unpreferred form:

$$\langle xy\rangle \in \{\langle uv\rangle : \Gamma uv\}$$

by corresponding sentences of the preferred form:

$$\Gamma xy$$

This procedural form would then be organized with other procedural forms in the deductive system resulting in the theorem prover T7.

When needed for proving further theorems, this lemma would be accessed from its less common constant symbols; for example, the relation symbol: $\{\langle ...,...\rangle : ...\}$. Now suppose that the definition D5 of this symbol had not yet been made and that the lemma I7 were instead:

$$\langle xy\rangle \in \{z: \exists u \exists v\ z = \langle uv\rangle \wedge \Gamma uv\} \leftrightarrow \Gamma xy$$

It would then be more difficult to access this lemma because the symbols occurring in it are quite common, and occur more often in set theory than does the relation symbol. Thus, in order to minimize the accessing problems it would be the job of the notation introduction system to immediately define the relation symbol.

Then using this defined symbol the inductive reasoning system would be able to create the earlier version of I7.

Finally, suppose that the theorem prover T6 were able to prove some theorem but that the theorem prover T7, using the lemma I7 were not. Then, assuming that nothing was wrong with T6, it would be reasonable to think that there was a bug in the procedural form of

I7. It would then be the job of the debugging system to isolate this bug and to correct the procedural form.

Some further work on the nature of inductive reasoning is given in [28]

7. Conclusions

We have argued that modern Deductive Knowledge systems are not in themselves the kind of theorem provers that are needed in order to develop general mathematical reasoning systems, and that what is needed are extensible Deductive Knowledge systems. Furthermore, we have discussed how an extensible deductive knowledge system would interact with a general mathematical reasoning system.

In the next chapter we describe a prototype extensible deductive system for elementary set theory, that we have implemented.

V Towards the Automation of Set Theory and its Logic

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 prototype extensible deductive system for the domain of elementary set theory.

Our goal in performing this research was to construct a theorem prover which was extensible, in the sense that new mathematical facts could be added and used by the system, while still allowing proofs involving those facts to be formally justified in terms of the original axioms of the theory. Our motivation for wanting to construct extensible systems is that we believe that such a system as opposed to an ordinary theorem prover is an essential component of a system which could construct mathematical theories.

In sections 2 and 3 we describe and then exemplify our basic deductive system. In section 4 we describe an application of an extensible version of this system, to the proving of a number of theorems in W.V.O. Quine's book: Set Theory and its Logic [1]. The implementation of our system is described in section 5, and some conclusions are drawn in section 6.

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 $(fa_1 \ldots a_n)$ is a sub-expression being evaluated, then the interpreter tries to apply its items of knowledge to that sub-expression before evaluating the arguments $a_1 \ldots a_n$. 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_i 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 $\{x\} = \{x\} \leftrightarrow$ ■ 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 $\{A\} = \{A\}$.

This interpreter has been used to prove theorems in several mathematical domains [2]. The items used by the interpreter in a particular domain are intended to be theorems of that domain. Thus for example, if Γ is a conjunction of axioms for a particular domain,

and if $\phi \leftrightarrow \psi$ is an item used in that domain, then $\Gamma \rightarrow (\phi \leftrightarrow \psi)$ should be logically valid.

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, in set theory, if we wish to prove theorems about functions (note that a function is a particular type of set) it would be quite useful to have available items which are only valid in the domain of functions, or more precisely when certain of the free variables occurring in the item are restricted to being functions: The representation of the item would be :

$$\text{Func } f \rightarrow (\phi_f \leftrightarrow \psi_f)$$

More generally then, 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:

$$\Pi x \rightarrow (\phi x \leftrightarrow \psi x) \quad (\text{or } \Pi x \rightarrow \phi x = \psi x)$$

Note that $(\Gamma \wedge \Pi x) \rightarrow (\phi x \leftrightarrow \psi x)$ should be logically valid.

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: Set Theory and its Logic [1] involves two domains. The larger domain, is the domain of abstracts, where an abstract of any propositional functional $f x$ with one free variable x is simply: $\{x: f x\}$, and the smaller domain is the domain of sets where a set is nothing more than an abstract α 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. Thus, for example, an item of the form

$$\phi x \leftrightarrow \psi x$$

is logically equivalent to one of the form:

$$\alpha \varepsilon V \rightarrow (\phi\alpha \leftrightarrow \psi\alpha)$$

where V is the universal abstract, that is the abstract which contains every set.

Just as a point about our notation, the reader will note that both Roman and Greek letters are also used to represent propositions. However, since the domain of propositions is distinct from the domain of abstracts, in most cases, there should be no difficulty in determining what is intended. For example, in $\alpha \wedge y$ (α and y) both α and y represent propositions whereas in $\alpha \cap y$ (α intersection y) α represents an abstract and y represents a set. Furthermore, even in

cases where disambiguation is not possible such as '$\phi\alpha$' (note that α represents either an abstract or a proposition depending on whether ϕ represents a predicate or logical connective) it does not really matter as Morse [3] has shown that we can let the two domains be identical.

An informed reader will recall that tables of rewrite rules (rather simple items represented in ones mathematical language), and other items were used in many of Bledsoe's theorem provers [4,5], and that Boyer and Moore [6,7] used a symbolic LISP interpreter to order the application of various recursive definitions, rewrite rules, and induction rules. More recently Aubin [8] has also used a symbolic interpreter for much the same purpose.

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
$\vee$	or
$\sim$	not
$\blacksquare$	true
$\square$	false
$\supset$	implies
$\leftrightarrow$	iff
$\exists$	there exists

$\forall$ for all

$=$ 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 $\phi 1$, ..., ψn called assertions which occur before the sequent arrow, and statements $\psi 1$, ..., ψn called goals which occur after the sequent arrow. The implicit conjunction of different sequents may be thought of as being a group of different databases.

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

Assertion schemata:

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

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

$\sim \rightarrow$: $(.. \sim x \ldots \rightarrow ..) \leftrightarrow (.. \ldots \rightarrow x ..)$

$\wedge \rightarrow$: $(.. x \wedge y .. \rightarrow ..) \leftrightarrow (.. x, y .. \rightarrow ..)$

$\vee \rightarrow$: $(.. x \vee y .. \rightarrow ..) \leftrightarrow (.. x .. \rightarrow ..)$ and $(.. y .. \rightarrow ..)$

$\supset \rightarrow$: $(.. x \supset y .. \rightarrow ..) \leftrightarrow (.. \ldots \rightarrow x ..)$ and $(.. y .. \rightarrow ..)$

$\leftrightarrow \rightarrow$: $(.. x \leftrightarrow y \ldots \rightarrow ..) \leftrightarrow (.. x, y .. \rightarrow ..)$ and $(.. \ldots \rightarrow x, y ..)$

$\exists \rightarrow$: $(.. \exists x \ \phi x .. \rightarrow ..) \leftrightarrow (.. \phi (f *_1 \ldots *_n) .. \rightarrow ..)$

where f is a new skolem function and $*_1 \ldots *_n$ are all the

unification variables which occur in ϕx.

$= \rightarrow$: $(\Pi a..\begin{Bmatrix} f*_1...*_n) = t \\ t = (f*_1...*_n) \end{Bmatrix}..\Gamma a \rightarrow \phi a..\psi a) \leftrightarrow (\Pi t..\Gamma t \rightarrow \phi t..\psi t)$

where f is a skolem function not occurring in t and a is $(f*_1...*_n)$

This is our version of the law of Leibnitz, for example:

$$(A = B \cup C \rightarrow C \subseteq A) \leftrightarrow (C \subseteq B \cup C)$$

Goal schemata:

$\rightarrow$ [redacted]: $(..\rightarrow..$ [redacted] $..) \leftrightarrow$ [redacted]

$\rightarrow \neg$: $(..\rightarrow..\ \neg\ ..) \leftrightarrow (..\rightarrow..\ ..)$

$\rightarrow \sim$: $(..\rightarrow..\sim x\ ..) \leftrightarrow (..x\rightarrow..\ ..)$

$\rightarrow \wedge$: $(..\rightarrow..x \wedge y..) \leftrightarrow (..\rightarrow..x..)$ and $(..\rightarrow..y..)$

$\rightarrow \vee$: $(..\rightarrow..x \vee y..) \leftrightarrow (..\rightarrow..x,y..)$

$\rightarrow \supset$: $(..\rightarrow..x \supset y..) \leftrightarrow (..x\rightarrow..y..)$

$\rightarrow \leftrightarrow$: $(..\rightarrow..x\leftrightarrow y\ ..) \leftrightarrow (..x\rightarrow..y..)$ and $(..y\rightarrow..x..)$

$\rightarrow \forall$: $(..\rightarrow..\forall x\ \phi x..) \leftrightarrow (..\rightarrow..\ \phi(f*_1\ ...\ *_n)\ ..)$

where f is a new skolem function and $*_1...*_n$ are all the unification variables which occur in ϕx.

Other logical schemata:

atom: $(..x..\rightarrow..x..) \leftrightarrow$ [redacted]

and : $(..$ and [redacted] and $..) \leftrightarrow (..$ and $..)$

Logical schemata used only at certain times:

Unify: $[(..x_1..\rightarrow..y_1..)$ and .. and .. $(..x_n..\rightarrow..y_n..)] \leftrightarrow$

$[(..x_i..\rightarrow..y_i..)$ and .. and .. $(..x_n..\rightarrow..y_n..)]\theta$

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

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. The rational behind this restriction, is similar to Bledsoe's [4,5] 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 ☐. An example instance of this item illustrating the forcing restriction is:

$$[(P^* \rightarrow Pa, Pb) \text{ and } (Q^* \rightarrow Qb) \text{ and } (R^* \rightarrow Ra)] \leftrightarrow (Rb \rightarrow Ra)$$

The unification variable * was instantiated to b not to a, because the substitution [* = b] makes tautologous the first two sequents, whereas the substitution [* = a] makes only the first sequent tautologous.

In this example, if the sequent $(R^* \rightarrow Ra)$ had occurred before the sequent $(Q^* \rightarrow Qb)$ then in fact * would have to be instantiated to a and the sequent $(Qa \rightarrow Qb)$ would have been the result. Thus we see that in the incomplete case where there is not a substitution θ which makes all the sequents tautologous, that the order of the sequents in the list of sequents to which the unify item is applied, can materially effect the results of applying the unify item. 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 $(..\forall x\phi x..\rightarrow..)$ then every sub-formulae of ϕt must occur in some sequent which must be made tautologous.* Thus for any sequent containing a sub-formulae ψ^* of ϕ^*, if t already occurs in the list associated with $(\forall x\phi x)$, then one of the following must hold:

(1) ψt is already in that sequent

(2) Some sub-formulae of ψt is already in that sequent and applying further logical items to ψt will eventually result in a sequent containing a second occurrence of that sub-formulae.

(3) ψt is a sub-formulae of some formulae already in that sequent, and applying further logical items to that formulae will eventually result in a sequent containing a second occurrence of ψt.

It follows that instantiating * to t, could only be redundant.

---

* Strictly speaking this remark must be modified so as to account for the $= \rightarrow$ item and any non logical items. It does, however, convey the basic idea.

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:

$$(Qt \rightarrow (\exists\ (x\ t\ *)\ (Px)),\ (Q*)) \leftrightarrow (Qt \rightarrow (\exists (x\ t\ *)\ (Px)),(Q*))$$

note that * cannot be instantiated to t because t already occurs in the list of the quantifier containing the unification variable *. We know that Q* is a sub-formulae of P* because the * in Q* occurs in the list of the quantifier binding Px.

A substitution list θ is applied to a sequent as follows: successively, for each substitution $[* = (t *_1 \ldots *_n)]$, every occurrence of * which occurs inside a skolem function is replaced by the segment $*_1 \ldots *_n$, and every other occurrence of * is replaced by the term $(t *_1 \ldots *_n)$. Thus for example if the sequent is:

$$\rightarrow * = (f*) \vee \{*\}$$

where f is a skolem function, then applying the substitution $[* = (*_1 \wedge *_2)]$ produces the new sequent:

$$\rightarrow *_1 \wedge *_2 = (f *_1\ *_2) \cup \{*_1 \wedge *_2\}$$

If the substitution is (* := A) then the new sequent is:

$$\rightarrow A = (f \cup A)$$

where f becomes a skolem constant.

The reason we are able to replace a variable * in a term $(f*)$ where f is a skolem function, by the sequence $*_1 \ldots *_n$ rather than by the term $(t *_1 \ldots *_n)$ is because every term beginning with f

must be identical. Thus if * is replaced by $(t\ *_1 \ldots *_n)$ in $(f*)$, every term beginning with f must then be identical to $(f(t\ *_1 \ldots *_n))$. But since every term beginning with f is followed by $(t\ *_1 \ldots *_n)$ and since f is merely a skolem function $(f(t\ *_1 \ldots *_n))$ is logically equivalent to $(g\ *_1 \ldots *_n)$ where g is a new skolem function. Replacing all occurrences of $(f(t\ *_1 \ldots *_n))$ by $(g\ *_1 \ldots *_n)$ we see that f no longer occurs in any expression. For this reason we can then rename g as f obtaining $(f\ *_1 \ldots *_n)$.

$\forall \rightarrow$: $(..\{\forall x\phi x / \forall(x...)\phi x\}..\rightarrow..) \leftrightarrow (..\rightarrow..\{\forall(x*)\phi x / \forall(x...*)\phi x\}, \phi*..)$

where * is a new unification variable and no more than one unification variable occurs in (x...).

$\rightarrow \exists$: $(..\rightarrow..\{\exists x\phi x / \exists(x...)\phi x\}..) \leftrightarrow (..\rightarrow..\{\exists(x*)\phi x / \exists(x...*)\phi x\}, \phi*..)$

where * is a new unification variable and no more than one unification variable occurs in (x...).

In the last two items we have seen formulae of the form $\forall(x...)\phi x$ and $\exists(x...)\phi x$ 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 found 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 applications of the $\forall \rightarrow$ or $\rightarrow \exists$ item. This pragmatic information is kept in order to be able to check the instantiation restriction, when using the Unify item, and in order to

check the restriction on the $\forall \rightarrow$ and $\rightarrow \exists$ 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 thoseunification variables, that could not be obtained by initially using only two instances and then after one of the unification variables is bound, and if necessary creating a third instance. An example of the use of this restriction is the fact that the $\rightarrow \exists$ 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 $\forall \rightarrow$, $\rightarrow \exists$, and unify items are used in a special way. Initially, the interpreter evaluates each sequent trying to apply the items in the following order:

(A)
(1) Non splitting assertion items:

$\Box \rightarrow$, $\neg \rightarrow$, $\sim \rightarrow$, $\wedge \rightarrow$, $\exists \rightarrow$, $= \rightarrow$

(2) Non splitting goal items:

$\rightarrow \Box$, $\rightarrow \neg$, $\rightarrow \sim$, $\rightarrow \vee$, $\rightarrow \supset$, $\rightarrow \forall$

(3) Non logical items

(4) The atom and "and" items

(5) Splitting assertion items:

$\vee \rightarrow$, $\supset \rightarrow$, $\leftrightarrow \rightarrow$

(6) Split ting goal items:

$\rightarrow \wedge$, $\rightarrow \leftrightarrow$

(B) The Unify item

(C) The $\forall \rightarrow$ and $\rightarrow \exists$ items once.

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) do not really reflect how the system is implemented, but merely 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 $= \rightarrow$ 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. For example, if we split on $(A \rightarrow B \wedge C)$ obtaining $(A \rightarrow B)$ and $(A \rightarrow C)$ we then have two copies of A to which to apply our items. However, we find that the time gained by not having to apply items a second time to A, 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 $\rightarrow \forall$, $\exists \rightarrow$, $\exists \rightarrow$, and $\forall \rightarrow$ 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.

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 re-ordering 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 theorem, 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. It would be detrimental because now that unification variable would already be bound to the wrong thing and could no longer be bound to what it should be bound to in order to solve the other sequents. Note then that if we do re-order according to our forcing strategy then the sequent containing that quantifier will probably not be made tautologous immediately, but later that quantifier will get its fair chance to contribute instances towards proving theorem. Note that the first application of the unify item will only result in that same conjunction of sequents because there would be no unification variables in the sequents until after an item in step (C) had been

applied at least once. An example of the effect of our forcing strategy is to re-order:

$$[(Qa \rightarrow \exists\ (x\ *_1\ C)\ Q^*_1)\ \text{and}\ (Qb \rightarrow \exists\ (x\ *_1)\ Q^*_1)]$$

as:

$$[(Qb \rightarrow \exists\ (x\ *_1\ C)\ Q^*_1)\ \text{and}\ (Qa \rightarrow \exists\ (x\ *_1)\ Q^*_1)]$$

thus forcing $*_1$ to be bound to b and not to a by the unify item.

(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 $\forall \rightarrow$ and $\rightarrow \exists$ 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. 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 $\forall \rightarrow$ item would be applied to the sequent:

$$((\forall x\ Px) \rightarrow (\exists\ (xa)\ Qx),\ (\exists\ (y^*)\ Qy))$$

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 initially causes one unification variable to be created in that sequent. Note, however, 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 variables as called for by the creation strategy. This allows us, for example, to prove theorems which need multiple variables from their quantifiers such as

examples 4 and 6 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.

The fact that only assertion equations are ever eliminated by = → schema implies a preference, for sequents containing only equations, not to have more than one goal equation. The reason for this preference is that there is no logical item which allows two goal equations to interact to produce □. Thus one of the goals would probably be irrelevant.

The application of every assertion or goal procedure to a database involves a particular logical symbol other than 'and' or '→'. For this reason we may view such procedures as being instructions of a language which manipulates databases. For example, the execution of the goal instruction → ∧ on A ∧ B creates two new databases (worlds, contexts) as follows:

(1) First A ∧ B is erased from the current database.

(2) A copy of the database is then created. (Actually in our system all this structure is shared).

(3) B is added as a new goal to the copy.

(4) A is added as a new goal to the current database.

A summary of the effect of goal and assertion instructions is given in Table 1. Note that the interpreter has complete freedom of choice both as to what database to work on first, and as to what instruction to execute first.

An informed reader will recall that these propositional rules

Table 1: A summary of logical instructions

Instructions*	effect as an assertion	effect as a goal
▩	no-op****	succeed
¤	succeed***	no-op
∿ A	add A as a goal	assert A
A ∧ B	assert A and B	split { add A as a goal / add B as a goal
A ∨ B	split** { assert A / assert B	add A and B as goals
A ⊃ B	split { add A as a goal / assert B	assert A, add B as a goal
A ↔ B	split { add A and B as goals / assert A and B	split { { assert A, add B as a goal / assert B, add A as a goal }
∀xϕx	assert ϕ* where * is a new unification variable.	add goal $\phi(f*_1 \ldots *_n)$ where f is a new skolem function and $*_1 \ldots *_n$ are all the unification variables which occur in ϕx.
∃xϕx	assert $\phi(f*_1 \ldots *_n)$ where f is a new skolem function and $*_1 \ldots *_n$ are all the	add goal ϕ* where * is a new unification variable.

$\left\{\begin{array}{l}(f^*_1 \ldots ^*_n) = t \\ \\ t = (f^*_1 \ldots ^*_n)\end{array}\right\}$ where f does not occur in t	unification variables which occur in ϕx. replace all occurrences of the skolem term $(f^*_1 \ldots ^*_n)$ by t.	———

* With the exception of the V assertion instruction and the] goal instruction, and executed instruction is always erased.

** Split means that a new data base is created.

*** Succeed means that the data base itself is erased.

**** A null operation.

are used in Wang's algorithm in the LISP 1.5 manual [9], and that Wang [10] 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 [11] 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 [12] 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 Unification variables and skolem constants. Bibel and Schreiber [13] have implemented a complete sequent logic, which disregarding certain inessential syntactic variations, is similar to our system except for the fact that it is complete and, always finds simplest proofs by a clever algorithm which, has the effect of always applying splitting rules before applying unification variable producing rules (i.e. → ∃ and ∀ →). For example, given the sequent

$$\rightarrow \exists x Ax \wedge Bx, \exists x Cx$$

their algorithm in effect simultaneously checks for the following three sequences of applying rules.

(1) $\rightarrow A^*_1 \wedge B^*_1, \exists x Cx$

$\rightarrow A^*_1, \exists x Cx; \rightarrow B^*_1 \exists x Cx$

$\rightarrow A^*_1, C^*_2 ; \rightarrow B^*_1 C^*_3$

(2) $\rightarrow A^*_1 \wedge B^*_1, \exists x Cx$

$\rightarrow A^*_1 \wedge B^*_1, C^*_2$

$\rightarrow A^*_1, C^*_2; \rightarrow B^*_1, C^*_2$

(3) $\rightarrow \exists x Ax \wedge Bx, C^*_1$

$\rightarrow A^*_2 \wedge B^*_2 C^*_1$

$\rightarrow A^*_2, C^*_1; \rightarrow B^*_2, C^*_1$

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 2. In that table, the name of each expression usually indicates where that expression may be found in Quine's book: Set Theory and its Logic [1]. For example, the definition of the symbol $\subset$ whose name is Q2P3 is expression number 3 in chapter 2 of that book. Note that a definition essentially defines the expression on the left side of the outermost $\leftrightarrow$ or = sign in terms of the expression on the right side.

There are four kinds of items:definitions,reduction lemmas,existence axioms, and existence lemmas. As previously mentioned, most of the definitions that were used are listed in Table 2. The reduction lemmas that were used are listed in Table 3. The existence axioms are given in Table 4, and the existence lemmas, are given in Table 5.

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.

Table 2: 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
Q2P3 :	$\alpha\subset\beta\leftrightarrow\alpha\subseteq\beta\wedge\sim\beta\subseteq\alpha$	is properly contained in
Q2P4 :	$\alpha\cup\beta=\{x:x\varepsilon\alpha\vee x\varepsilon\beta\}$	union
Q2P5 :	$\alpha\cap\beta=\{x:x\varepsilon\alpha\wedge x\varepsilon\beta\}$	intersection
Q2P6 :	$\alpha=\{x:\sim x\varepsilon\alpha\}$	complement
Q2P7 :	$\alpha=\beta\leftrightarrow\forall x\ x\varepsilon\alpha\leftrightarrow x\varepsilon\beta$	equals
Q2P8 :	$\emptyset=\{z:\ \square\}$	null set
Q2P9 :	$V=\{z:\ \blacksquare\}$	universe
Q5P5 :	$\{x:\Gamma x\}\varepsilon\beta\leftrightarrow\exists y\ y=\{x:\Gamma x\}\wedge y\varepsilon\beta$	the set $\{x:\Gamma x\}$ is in
Q7P1A :	$\{\alpha\}=\{z:\ z=\alpha\}$	unit set
Q7P1B :	$\{\alpha\beta\}=\{z:\ z=\alpha\vee z=\beta\}$	pair set
Q8P1 :	$\bigcup\alpha=\{x:\ \exists y\ x\varepsilon y\wedge y\varepsilon\alpha\}$	union of all subsets
Q8P9 :	$\bigcap\alpha=\{x:\ \forall y\ y\varepsilon\alpha\rightarrow x\varepsilon y\}$	intersection of all subsets
Q8P18 :	$\iota x\Gamma x=\ \{y:\ \forall x\ \Gamma x\leftrightarrow x=y\}$	the x such that Γx
Q9P1 :	$<\alpha\beta>=\{\{\alpha\}\{\alpha\beta\}\}$	ordered pair
Q9P4 :	$\{<xy>:\ \Gamma\ xy\}=\{u:\exists x\exists y\ u=<xy>\wedge\Gamma xy\}$	abstract of ordered pairs
Q9P6 :	$^{\circ}\alpha=\{<xy>:<xy>\varepsilon\alpha\}$	relational part
Q9P11 :	$\alpha\mathbf{x}\beta=\{<xy>:x\varepsilon\alpha\wedge y\varepsilon\beta\}$	cartesian product
Q9P12 :	$\breve{\alpha}=\{<xy>:<yx>\varepsilon\alpha\}$	converse
Q9P13 :	$\alpha\mid\beta=\{<xz>:\exists y<xy>\varepsilon\alpha\wedge<yz>\varepsilon\beta\}$	resultant
Q9P14 :	$\alpha"\beta=\{x:\exists y<xy>\varepsilon\alpha\wedge y\varepsilon\beta\}$	image
Q9P15 :	$I=\{<xy>:\ x=y\}$	identity relation
Q10P1 :	$\text{Func}\ \alpha\leftrightarrow(\forall x\forall y\forall z<xz>\varepsilon\alpha\wedge<yz>\varepsilon\alpha\rightarrow x=y)\wedge\alpha=^{\circ}\alpha$	is a function

Q1OP2 :	$\arg\ \alpha=\{x:\exists y \alpha"\{x\}=\{y\}\}$	the arguments of
Q1OP11:	$\alpha'\beta=\iota y<y\beta>\varepsilon\alpha$	apply
Q1OP21:	$\lambda x(\alpha x)=\{<yx>:y=(\alpha x)\}$	the function
D1 :	$P\alpha=\{u:\ u\subseteq\alpha\}$	powerset
Q11P1 :	$\alpha\leq\beta\leftrightarrow\exists f\ \mathrm{Func}f\wedge\alpha\subseteq f"\beta$	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 3: Reduction Lemmas

Q6P4 :	$\alpha=\alpha\leftrightarrow$ [illegible]
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 :	$\forall x\ \forall y\ <xy>=<uv>\leftrightarrow x=u\wedge y=v$
Q9P5 :	$\forall x\ \forall y\ <xy>\ \varepsilon\ \{<uv>:\phi uv\}\leftrightarrow\phi xy$
Q1OP24:	$\forall t\ \forall y\ \lambda x(\phi x)\ 'y=\phi y$
CRL1 :	$\mathrm{Func}\ f\rightarrow(<wy>\varepsilon f\leftrightarrow w=f'y)$

Table 4: Existence axioms and axiom of extentionality

Q7P1OA:	$\emptyset\varepsilon V$
Q7P1OB:	$\forall x\ \forall y\ \{xy\}\ \varepsilon V$
Q4P1 :	$\forall x\ \forall y\ \forall z\ (x=y\wedge x\varepsilon z.\supset y\varepsilon z)$

Table 5: Existence Lemmas

Q7P12: $\{\alpha\}\varepsilon V$

Q7P13: $\{\alpha\beta\}\varepsilon V$

E1 : $\exists x \Gamma x \varepsilon V$

E2 : $<\alpha\beta>\varepsilon V$

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 3.

and finally: All other definitions.

For example, since the definition Q9P1 is tried only after the the reduction lemma Q9P3, $\langle xy\rangle = \langle uv\rangle$ 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 reduced the complexity of its sequents, means that the resulting expressions are becoming less complex. This systematic reduction of complexity amounts to a searchless proof procedure

which tries to prove a theorem by developing one and only one path, no matter how devious, through the deductive search space. Since there are no infinitely descending chains of complexity, that is one cannot go on applying definitions forever, if the original expression was a theorem then this path is guaranteed to terminate with a proof.

Strictly speaking, this last remark is true only if the underlying logic is complete which ours is not. Nevertheless in most cases that occur in elementary set theory our logical system, while less explosive than a complete procedure for logic seems to be able to handle the logical difficulties involved in finding a proof. A complete version of our logic may be obtained by replacing our unify schema by Prawitz's unification rule [11], and by replacing our $= \rightarrow$ schema by explicit axioms.

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 Siekmann [14],

Plotkin [15], and Lankford [16].

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. One simple example comes to mind. Consider the contextual definition Q2P1 in Table 2:

$\forall y(y \varepsilon\{x:\Gamma x\} \leftrightarrow \Gamma y)$

Semantically, this definition means that for all y, y is an element of the abstract of all x such that Γ x iff Γy, but as an item to be used in evaluation, all this means is that any expression of the form: $y\varepsilon\{x:\Gamma x\}$ where y is a variable (or a skolem function or a unification variable in the framework of a sequent calculus) may be replaced by the corresponding expression Γy.

Let R be some abstract which can be proven to be equal to some variable y. Would it not then be useful to have a reduction lemma of the form?

$R \varepsilon \{x:\phi x\} \leftrightarrow \phi_R$

Clearly this lemma is indeed a theorem, for since R=y then it is equivalent to Q2P2.

Consider now Zermelo-Frankel set theory: From the axiom of foundation we learn that $x \notin x$, and hence that $\{x:x\varepsilon x\}$ is the null set which clearly is equal to some variable by axiom Q7P10A of Table 3. Letting R be: $\{x:x\varepsilon x\}$ and ϕx be $x\varepsilon x$ the above reduction lemma becomes:

$\{x:x\varepsilon x\} \varepsilon \{x:x\varepsilon x\} \leftrightarrow \{x:x\varepsilon x\} \varepsilon \{x:x\varepsilon x\}$

which clearly could not possibly reduce complexity. Never-the-less,

the failure to reduce complexity in such a pathological case, should not stop our interpreter from making use of this reduction lemma.

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. For example, if R_1 ε V and R_2 ε V are assumed then the new reduction lemmas corresponding to Q2P1 are:

$$R_1 \varepsilon \{x:\Gamma x\} \leftrightarrow \Gamma R_1$$

$$R_2 \varepsilon \{x:\Gamma x\} \leftrightarrow \Gamma R_2$$

and the new reduction lemmas corresponding to the reduction lemma Q7P7 are:

$$\{R_1\}=\{y\} \leftrightarrow R_1=y \qquad \{R_1\}=\{R_1\} \leftrightarrow R_1=R_1$$

$$\{x\}=\{R_1\} \leftrightarrow x=R_1 \qquad \{R_2\}=\{R_2\} \leftrightarrow R_2=R_2$$

$$\{R_2\}=\{y\} \leftrightarrow R_2=y \qquad \{R_1\}=\{R_2\} \leftrightarrow R_1=R_2$$

$$\{x\}=\{R_2\} \leftrightarrow x=R_2 \qquad \{R_2\}=\{R_1\} \leftrightarrow R_2=R_1$$

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.

Note then that the new lemmas corresponding to the contextual definition Q2P1 will be applicable only to formulas to which the contextual definition Q5P5 is applicable. If we are ever to use

these lemmas it is therefore necessary to require that we try to apply these lemmas related to Q2P1 before trying to apply Q5P5. Furthermore, in order to avoid an infinite circular oscillation caused by the interaction between the new lemmas related to $= \rightarrow$ and Q5P5 schemata on hypotheses it is necessary to restrict Q5P5 to being applied only it its first argument is not known to exist. An example of this oscillation is:

$$\phi \varepsilon A \rightarrow$$

$$\exists y\ y=\phi \wedge y \varepsilon A \rightarrow$$

$$a=\phi \wedge a \varepsilon A \rightarrow$$

$$\phi \varepsilon A \rightarrow$$

$$\vdots$$

There is a rationale behind this restriction: Let S be an abstract. Then Q5P5: $S\varepsilon\alpha \leftrightarrow \exists y\ y=S \wedge y\varepsilon\alpha$ means that S exists (because its equal to some variable) and is in α. However, if we already know that S exists, what would be the purpose in re-deducing this fact by applying Q5P5? An example of the use of this restriction is in the proof of theorem THS31 where Q5P5 is never applied to '<ab>$\varepsilon\delta$', because <ab> is known to exist.

There is one further way in which the existence axioms are used, and this has to do with the fact that they are axioms. Clearly if we wish to show that a theorem T is deducable from the axiom A then unless our reduction lemmas, being truth value preserving in A, implicitly introduce A, it will be necessary to assume A as a hypothesis. 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 T, it will in primitive notation add each axiom A as an additional hypothesis in that sequent.

Fortunately this rarely has to be done because usually reduction lemmas which are not logically valid, but only valid in our set theory are used, and the axioms are thus implicitly used as they are needed.

The point about adding the axiom in its primitive notation, that is with all the definitions eliminated is that, the defined notation, often by interaction with lemmas will simply be reduced to the true hypothesis "□". For example, Q7P10A if introduced at a stage where ∅ is known to be a set will reduce as follows:

$\emptyset \in V \rightarrow$:Q2P9

$\emptyset \in \{x: \square\} \rightarrow$:[Q2P1, Q7P10A]*

$\square \rightarrow$: $\square \rightarrow$

$\rightarrow$

On the other hand, if Q2P9 is introduced in primitive notation then the unwanted reduction does not occur:

$$\exists y\ \forall x(x \in y \leftrightarrow \square)$$

The point then, is that in some proofs it is necessary to have the actual code of the axiom, even after lemmas such as: [Q2P1, Q7P10A]* are available. The primitive notation of our axioms are given in Table 6.

Table 6: Primitive notations of existence axioms

Q7P10A: $\exists y\ \forall x \cup X \in Y$

Q7P10B: $\forall u \forall v \exists y\ \forall x\ x \in y \leftrightarrow x=u \vee x=v$

Q4P1 : $(\forall u\ u\varepsilon x \leftrightarrow u\varepsilon y) \supset (\forall x\ x\varepsilon z \supset 'y\varepsilon z)$

In section 2.1 we mentioned that the logical items could be viewed as being instructions in a database manipulation language. It turns out that this language may be extended in that new instructions may be defined in terms of the primitive logical ones. For example, definition "$\alpha \subset\!\!= \beta$" does not split the database, check to see if everything in α is in β in the first database and then find something which is in β but not in α in the second database. Rather $\alpha \subset\!\!= \beta$ simply returns its definition $\alpha \subseteq \beta \wedge \sim\beta \subseteq \alpha$ and lets the mathematical interpreter perform these instructions automatically using its previous definitions and logical items.

Again an informed reader will recall Bledsoe's set theory theorem prover [4].

3. Examples

We give below a number of protocols produced by our theorem prover while proving various theorems of set theory. We have listed on the right the name of the item which was applied to each sequent in the protocols. 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 proofs are 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 ($\rightarrow \wedge$, $\rightarrow \leftrightarrow$, $\vee \rightarrow$, $\supset \rightarrow$, $\leftrightarrow \rightarrow$) two sequents will be produced, and then they are indicated by drawn arrows.

Each 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 starting:

$:\theta_k$ starts here

where k refers to segment k. The system attempts to apply the Unify item once to a sequence of all end sequents 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.

The first theorem whose proof we give is the commutativity of intersection.

Example 1

THS6 The commutativity of intersection'

$\rightarrow \alpha \cap \beta = \beta \cap \alpha$:Q2P7

$\rightarrow \forall x\ x\varepsilon\alpha \cap \beta \leftrightarrow x\varepsilon\beta \cap \alpha$:$\rightarrow \forall$

$\rightarrow \quad c\varepsilon\alpha \cap \beta \leftrightarrow c\varepsilon\beta \cap \alpha$:$\rightarrow \leftrightarrow$

and

$c\varepsilon\alpha \cap \beta \rightarrow \quad c\varepsilon\beta \cap \alpha$:Q2P5

$c\varepsilon\{x: x\varepsilon\alpha \wedge x\varepsilon\beta\} \rightarrow c\varepsilon\beta \cap \alpha$:Q2P1

$c\varepsilon\alpha \wedge c\varepsilon\beta \rightarrow c\varepsilon\beta \cap \alpha$: $\wedge \rightarrow$

$c\varepsilon\alpha,\ c\varepsilon\beta \rightarrow c\varepsilon\beta \cap \alpha$:Q2P5

$c\varepsilon\alpha,\ c\varepsilon\beta \rightarrow c\varepsilon\{x: x\varepsilon\beta \wedge x\varepsilon\alpha\}$:Q2P1

$c\varepsilon\alpha,\ c\varepsilon\beta \rightarrow c\varepsilon\beta \wedge c\varepsilon\alpha$:$\rightarrow \wedge$

and

$c\varepsilon\alpha,\ c\varepsilon\beta \rightarrow c\varepsilon\beta$:atom $\quad c\varepsilon\alpha,\ c\varepsilon\beta \rightarrow c\varepsilon\alpha$:atom

$c\varepsilon\beta \cap \alpha \rightarrow c\varepsilon\alpha \cap \beta$:Q2P5

$c\varepsilon\{x: x\varepsilon\beta \wedge x\varepsilon\alpha\} \rightarrow c\varepsilon\alpha \cap \beta$:Q2P1

$c\varepsilon\beta \wedge c\varepsilon\alpha \rightarrow c\varepsilon\alpha \cap \beta$: $\wedge \rightarrow$

$c\varepsilon\beta,\ c\varepsilon\alpha \rightarrow c\varepsilon\alpha \cap \beta$:Q2P5

$c\varepsilon\beta,\ c\varepsilon\alpha \rightarrow c\varepsilon\{x: x\varepsilon\alpha \wedge x\varepsilon\beta\}$:Q2P1

$c\varepsilon\beta,\ c\varepsilon\alpha \rightarrow c\varepsilon\alpha \wedge c\varepsilon\beta$:$\rightarrow \wedge$

and

$c\varepsilon\beta,\ c\varepsilon\alpha \rightarrow c\varepsilon\alpha$:atom $\quad c\varepsilon\beta,\ c\varepsilon\alpha \rightarrow c\varepsilon\beta$:atom

time = 289 milliseconds

Of course, if the theorem prover had already known that intersection was commutative, that is if it had an item such as:

$$\alpha \cap \beta = \beta \cap \alpha \leftrightarrow \blacksquare$$

then the proof of THS6 would have been immediate. The point is that all the theorem prover knows about intersection is its definition, and it is this item which it is trying to prove.

In general then, the reader should bear in mind that our theorem prover knows nothing about set theory other than the definitions, axioms, and lemmas listed in Tables 2, 3, 4 and 5. Thus, the ability of this theorem prover at proving theorems lies not in having many special 'banana' lemmas* lying around which give immediate proof, but rather in the general deductive power of the mathematical interpreter and of the sequent logic.

An example of this deductive power can be seen in the next example.

---

* A phrase coined by Hayes [17] suggesting the reason as to why a proof of the Monkey and Banana problem took so many steps in his theory of action.

Example 2

THS101 The Cartesian product of two abstracts is contained in the powerset of the powerset of their union.

$$\rightarrow \alpha \times \beta \subseteq PP(\alpha \cup \beta) \quad :Q2P2$$

$$\rightarrow \forall x\ x \in \alpha \times \beta \supset x \in PP(\alpha \cup \beta) \quad :\rightarrow \forall$$

$$\rightarrow c \in \alpha \times \beta \supset c \in PP(\alpha \cup \beta) \quad :\rightarrow \supset$$

$$c \in \alpha \times \beta \rightarrow c \in PP(\alpha \cup \beta) \quad :Q9P11$$

$$c \in \{\langle xy \rangle : x \in \alpha \wedge y \in \beta\} \rightarrow c \in PP(\alpha \cup \beta) \quad :Q9P4$$

$$c \in \{u : \exists x \exists y\ u = \langle xy \rangle \wedge x \in \alpha \wedge y \in \beta\} \rightarrow c \in PP(\alpha \cup \beta) \quad :Q2P1$$

$$\exists x \exists y\ c = \langle xy \rangle \wedge x \in \alpha \wedge y \in \beta \rightarrow c \in PP(\alpha \cup \beta) \quad :\exists \rightarrow$$

$$\exists y\ c = \langle ay \rangle \wedge a \in \alpha \wedge y \in \beta \rightarrow c \in PP(\alpha \cup \beta) \quad :\exists \rightarrow$$

$$c = \langle ab \rangle \wedge a \in \alpha \wedge b \in \beta \rightarrow c \in PP(\alpha \cup \beta) \quad :\wedge \rightarrow$$

$$c = \langle ab \rangle,\ a \in \alpha \wedge b \in \beta \rightarrow c \in PP(\alpha \cup \beta) \quad :[\Rightarrow, E2]^*$$

$$a \in \alpha \wedge b \in \beta \rightarrow \langle ab \rangle \in PP(\alpha \cup \beta) \quad :\wedge \rightarrow$$

$$a \in \alpha,\ b \in \beta \rightarrow \langle ab \rangle \in PP(\alpha \cup \beta) \quad :D1$$

$$a \in \alpha,\ b \in \beta \rightarrow \langle ab \rangle \in \{u : u \subseteq P(\alpha \cup \beta) \quad :[Q2P1, E2]^*$$

$$a \in \alpha,\ b \in \beta \rightarrow \langle ab \rangle \subseteq P(\alpha \cup \beta) \quad :Q2P2$$

$$a \in \alpha,\ b \in \beta \rightarrow \forall x\ x \in \langle ab \rangle \supset x \in P(\alpha \cup \beta) \quad :\rightarrow \forall$$

$$a \in \alpha,\ b \in \beta \rightarrow d \in \langle ab \rangle \supset d \in P(\alpha \cup \beta) \quad :\rightarrow \supset$$

$$a \in \alpha,\ b \in \beta,\ d \in \langle ab \rangle \rightarrow d \in P(\alpha \cup \beta) \quad :Q9P1$$

$$a \in \alpha,\ b \in \beta,\ d \in \{\{a\}\{ab\}\} \rightarrow d \in P(\alpha \cup \beta) \quad :Q7P1B$$

$$a \in \alpha,\ b \in \beta,\ d \in \{u : u = \{a\} \vee u = \{ab\}\} \rightarrow d \in P(\alpha \cup \beta) \quad :Q2P1$$

$$a \in \alpha,\ b \in \beta,\ d = \{a\} \vee d = \{ab\} \rightarrow d \in P(\alpha \cup \beta) \quad :D1$$

↓

Continued.....

Example 2 continued

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\} \vee d=\{ab\} \rightarrow d\varepsilon\{u:\ u \subseteq \alpha \cup \beta\}$:Q2P1

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\} \vee d=\{ab\} \rightarrow d \subseteq \alpha \cup \beta$:Q2P2

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\} \vee d=\{ab\} \rightarrow \forall x\ x\varepsilon d \supset x\varepsilon\alpha \cup \beta$:→ ∀

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\} \vee d=\{ab\} \rightarrow e\varepsilon d \supset e\varepsilon\alpha \cup \beta$:→ ⊃

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\} \vee d=\{ab\},\ e\varepsilon d \rightarrow e\varepsilon\alpha \cup \beta$:Q2P4

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\} \vee d=\{ab\},\ e\varepsilon d \rightarrow e\varepsilon\{x:\ x\varepsilon\alpha \vee x\varepsilon\beta\}$:Q2P1

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\} \vee d=\{ab\},\ e\varepsilon d \rightarrow e\varepsilon\alpha \vee e\varepsilon\beta$:→ ∨

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\} \vee d=\{ab\},\ e\varepsilon d \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:∨ →

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{a\},\ e\varepsilon d \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:[=→,Q7P12]*

$a\varepsilon\alpha,\ b\varepsilon\beta,\ e\varepsilon\{a\} \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:Q7P1A

$a\varepsilon\alpha,\ b\varepsilon\beta,\ e\varepsilon\{x:\ x=a\} \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:Q2P1

$a\varepsilon\alpha,\ b\varepsilon\beta,\ e=a \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:=→

$a\varepsilon\alpha,\ b\varepsilon\beta \rightarrow a\varepsilon\alpha,\ a\varepsilon\beta$:atom

■

$a\varepsilon\alpha,\ b\varepsilon\beta,\ d=\{ab\},\ e\varepsilon d \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:[=→,Q7P13]*

$a\varepsilon\alpha,\ b\varepsilon\beta,\ e\varepsilon\{ab\} \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:Q7P1B

$a\varepsilon\alpha,\ b\varepsilon\beta,\ e\varepsilon\{x:\ x=a \vee x=b\} \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:Q2P1

$a\varepsilon\alpha,\ b\varepsilon\beta,\ e=a \vee e=b \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:∀ →

$a\varepsilon\alpha,\ b\varepsilon\beta,\ e=a \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:=→

$a\varepsilon\alpha,\ b\varepsilon\beta \rightarrow a\varepsilon\alpha,\ a\varepsilon\beta$:atom

■

$a\varepsilon\alpha,\ b\varepsilon\beta,\ e=b \rightarrow e\varepsilon\alpha,\ e\varepsilon\beta$:=→

$e\varepsilon\alpha,\ b\varepsilon\beta \rightarrow b\varepsilon\alpha,\ b\varepsilon\beta$:atom

■

time = 626 millisec

The four names in this proof which were starred represent reduction lemmas which were created from the knowledge that the abstracts $\{\alpha\}$, $\{\alpha\beta\}$, and $<\alpha\beta>$ are sets. Once these abstracts are known to be sets the proof of this theorem proceeds without any redundant steps. On the other hand if this knowledge were not already available it would have been quite difficult to prove this theorem: for essentially the theorem prover would have had to stop the proof where each of these items were used, create and prove one of the sentences $\{\alpha\}\varepsilon V$, $\{\alpha\beta\}\varepsilon V$, $<\alpha\beta>\varepsilon V$, then use this theorem to create new reduction lemmas and then proceed with the proof until it reached the place where the next one of these starred items was applied, and so forth.

As this phenomena appears when trying to prove many theorems of set theory, we make the following conclusion. It appears to be the case that many theorems cannot be easily proven until the theorem prover has acquired a certain level of knowledge (in this case in terms of reduction lemmas as existence lemmas) about the domain in which it is working.

Example 3

Q8P11: The infinite intersection of the union of two abstracts is the intersection of the infinite intersections of those two abstracts.

Merely for the purposes of presenting the protocol of this theorem in this document we make three abbreviations which have nothing to do with the theorem prover.

① =df (∀y yεα ∪ β ⊃ aεy)

② =df (∀y yεα ⊃ aεy)

③ =df (∀y yεβ ⊃ aεy)

→ ⋂(α ∪ β) = ⋂α ∩ ⋂β :Q2P7

→ ∀x xε ⋂(α ∪ β) ↔ xε ⋂α ∩ ⋂β :→ ∀

→ aε ⋂α ∪ β ↔ aε ⋂α ∩ ⋂β :→ ↔

"C"

aε ⋂α ∪ β → aε ⋂α ∩ ⋂β :Q9P9

aε{x:∀y yεα ∪ β ⊃ xεy} → aε ⋂α ∩ ⋂β :Q2P1

(∀y yεα ∪ β ⊃ aεy) → aε ⋂α ∩ ⋂β :Q2P5

① → aε{x: xε ⋂α ∧ xε ⋂β} :Q2P1

① → aε ⋂α ∧ aε ⋂β :→

①	→ aε ⋂α	:Q8P9	①	→ aε ⋂β	:Q8P9
①	→ aε{x: ∀y yεa ⊃ xεy}	:Q2P1	①	→ aε{x: ∀y yεβ ⊃ xεy}	:Q2P1
①	→ ∀y yεα ⊃ aεy	:→ ∀	①	→ ∀y yεβ ⊃ aεy	:→ ∀
①	→ bεα ⊃ aεb	:→ ⊃	①	→ cεβ ⊃ aεc	:→ ⊃
	↓ "A"			↓ "B"	

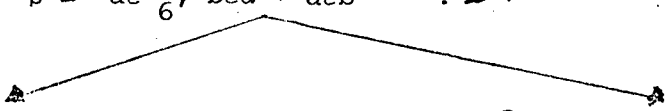
"A"

:θ4 starts here:

①, bεα → aεb :∀→

The two following steps marked unify are actually a single step producing [$*_6$=b]

①, $*_6$εα ∪ β ⊃ aε$*_6$, bεα → aεb :⊃→



①, bεα → $*_6$εα ∪ β, aεb :Q2P4 ①, aε$*_6$, bεα → aεb :Unify

①, bεα → $*_6$ε{x: xεα ∨ xεβ},aεb :Q2P1 ▣ [$*_6$=b]

①, bεα → $*_6$εα ∨ $*_6$εβ, aεβ :→∨

①, bεα → $*_6$εα, $*_6$εβ, aεβ :unify

▣ [$*_6$=b]

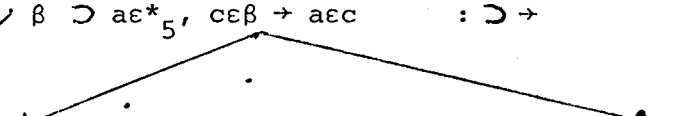
"B"

:θ3 starts here:

①, cεβ → aεc :∀ →

The two following steps marked Unify are actually a single step producing [$*_5$=c]

①, $*_5$εα ∪ β ⊃ aε$*_5$, cεβ → aεc : ⊃ →



①, cεβ → $*_5$εα ∪ β, aεc :Q2P4 ①, aε$*_5$, cεβ → aεc :Unify

①, cεβ → $*_5$ε{x: xεα ∨ xεβ},aεc :Q2P1 ▣ [$*_5$=c]

①, cεβ → $*_5$εα ∨ $*_5$εβ, aεc :→∨

①, cεβ → $*_5$εα, $*_5$εβ, aεc :Unify

▣ [$*_5$=c]

"C"

$a\varepsilon \bigcap \alpha \cap \bigcap \beta \rightarrow a\varepsilon \bigcap \alpha \cup \beta$:Q2P5

$a\varepsilon\{x: x\varepsilon \bigcap \alpha \wedge x\varepsilon \bigcap \beta\} \rightarrow a\varepsilon \bigcap \alpha \cup \beta$:Q2P1

$a\varepsilon \bigcap \alpha \wedge a\varepsilon \bigcap \beta \rightarrow a\varepsilon \bigcap \alpha \cup \beta$:$\wedge \rightarrow$

$a\varepsilon \bigcap \alpha, a\varepsilon \bigcap \beta \rightarrow a\varepsilon \bigcap \alpha \cup \beta$:Q9P9

$a\varepsilon\{x: \forall y\ y\varepsilon\alpha \supset x\varepsilon y\}, a\varepsilon \bigcap \beta \rightarrow a\varepsilon \bigcap \alpha \cup \beta$:Q2P1

$(\forall y\ y\varepsilon\alpha \supset a\varepsilon y), a\varepsilon \bigcap \beta \rightarrow a\varepsilon \bigcap \alpha \cup \beta$:Q9P9

②, $a\varepsilon\{x: \forall y\ y\varepsilon\beta \supset x\varepsilon y\} \rightarrow a\varepsilon \bigcap \alpha \cup \beta$:Q2P1

②, $(\forall y\ y\varepsilon\beta \supset a\varepsilon y) \rightarrow a\varepsilon \bigcap \alpha \cup \beta$:Q8P9

②, ③ $\rightarrow a\varepsilon\{x: \forall y\ y\varepsilon\alpha \cup \beta \supset x\varepsilon y\}$:Q2P1

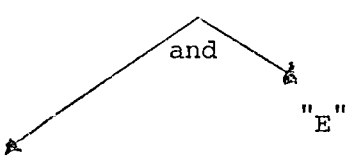
②, ③ $\rightarrow (\forall y\ x\varepsilon\alpha \cup \beta \supset a\varepsilon y)$:$\rightarrow \forall$

②, ③ $\rightarrow d\varepsilon\alpha \cup \beta \supset a\varepsilon d$:$\rightarrow \supset$

②, ③, $d\varepsilon\alpha \cup \beta \rightarrow a\varepsilon d$:Q2P4

②, ③, $d\varepsilon\{x: x\varepsilon\alpha \vee x\varepsilon\beta\} \rightarrow a\varepsilon d$:Q2P1

②, ③, $d\varepsilon\alpha \vee d\varepsilon\beta \rightarrow a\varepsilon d$:$\vee \rightarrow$



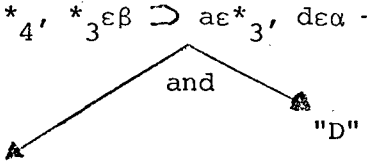
:Θ2 starts here:

②, ③, $d\varepsilon\alpha \rightarrow a\varepsilon d$:$\forall \rightarrow$

②, ③, $*_3\varepsilon\beta \supset a\varepsilon *_3, d\varepsilon\alpha \rightarrow a\varepsilon d$:$\forall \rightarrow$

The four following steps marked Unify are actually a single step producing [*4=d]

②, ③, $*_4\varepsilon\alpha \supset a\varepsilon *_4, *_3\varepsilon\beta \supset a\varepsilon *_3, d\varepsilon\alpha \rightarrow a\varepsilon d$:$\supset \rightarrow$



②, ③, $*_3\varepsilon\beta \supset a\varepsilon*_3$, $d\varepsilon\alpha \rightarrow a\varepsilon d$, $*_4\varepsilon\alpha$:⊃→

and

②, ③, $d\varepsilon\alpha \rightarrow a\varepsilon d$, $*_4\varepsilon\alpha$, $*_3\varepsilon\beta$:Unify ②, ③, $a\varepsilon*_3$, $d\varepsilon\alpha \rightarrow a\varepsilon d$, $*_4\varepsilon\alpha$:Unify

[$*_4$=d] [$*_4$=d] or [$*_3$=d]

"D"

②, ③, $a\varepsilon*_4$, $*_3\varepsilon\beta \supset a\varepsilon*_3$, $d\varepsilon\alpha \rightarrow a\varepsilon d$:⊃→

②, ③, $a\varepsilon*_4$, $d\varepsilon\alpha \rightarrow a\varepsilon d$, $*_3\varepsilon\beta$:Unify ②, ③, $a\varepsilon*_4$, $a\varepsilon*_3$, $d\varepsilon\alpha \rightarrow a\varepsilon d$:Unify

[$*_4$=d] [$*_4$=d] or [$*_3$=d]

"E"

:Θ starts here:

②, ③, $d\varepsilon\beta \rightarrow a\varepsilon d$:∀→

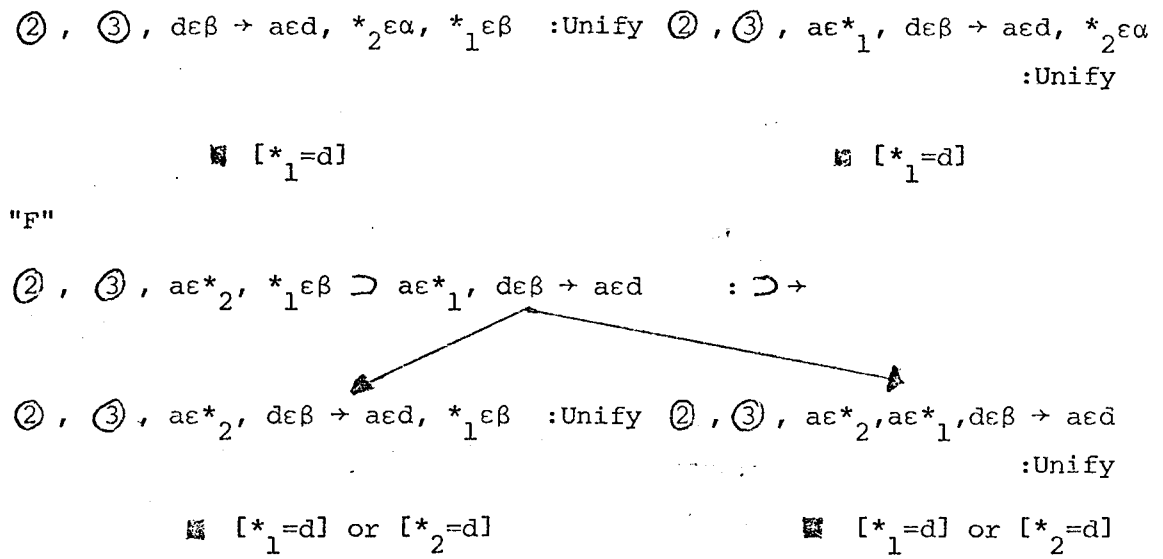
②, ③, $*_1\varepsilon\beta \supset a\varepsilon*_1$, $d\varepsilon\beta \rightarrow a\varepsilon d$:∀→

The four following steps marked Unify are actually a single step producing [$*_1$=d]

②, ③, $*_2\varepsilon\alpha \supset a\varepsilon*_2$, $*_1\varepsilon\beta \supset a\varepsilon*_1$, $d\varepsilon\beta \rightarrow a\varepsilon d$:⊃→

"F"

②, ③, $*_1\varepsilon\beta \supset a\varepsilon*_1$, $d\varepsilon\beta \rightarrow a\varepsilon d$, $*_2\varepsilon\alpha$:⊃→



time = 1,161 milliseconds

Note that the initial sequent where θO starts is broken down into four independent sequents which are solved in the parts of the proof known respectively as θ1, θ2, θ3 and θ4. Note also that the Unify rule was needed to solve these four subgoals, but that the ∀ → and → ∃ items were never applied more than once to any formulae in any sequent, that is no quantifier list e.g. ∃(x*1...) ever contains more than one unification variable.

In the next example we shall see that multiple instances of some formulae are sometimes needed in order to prove a theorem.

Example 4

Q6P10: Russell's paradox and Quine's more complicated version. We prove that these "paradoxical" abstracts are not sets.

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

① =df $(\forall x.x\varepsilon\alpha \leftrightarrow x\varepsilon\{x:\cup x\varepsilon x\})$

② =df $(\forall x.x\varepsilon b \leftrightarrow x\varepsilon\{x:\cup\exists y\ x\varepsilon y \wedge y\varepsilon x\})$

③ =df $(\exists y.*_1\varepsilon y \wedge y\varepsilon *_1)$

④ =df $(\exists y.\ c\ \varepsilon y \wedge y\varepsilon\ c\ )$

$\rightarrow \cup\{x:\cup x\varepsilon x\}\varepsilon V \wedge \cup\{x:\cup\exists y\ x\varepsilon y \wedge y\varepsilon x\}\varepsilon V$:$\rightarrow\wedge$

$\rightarrow \cup\{x:\cup x\varepsilon x\}\varepsilon V$:$\rightarrow \cup$ "C"

$\{x:\cup x\varepsilon x\}\varepsilon V\rightarrow$:Q5P5

$\exists y\ y=\{x:\cup x\varepsilon x\} \wedge y\varepsilon V\rightarrow$:$\exists\rightarrow$

$a=\{x:\cup x\varepsilon x\} \wedge a\varepsilon V\rightarrow$:$\wedge\rightarrow$

$a=\{x:\cup x\varepsilon x\},\ a\varepsilon V\rightarrow$:Q2P7

$\forall x\ x\varepsilon a \leftrightarrow x\varepsilon\{x:\cup x\varepsilon x\},\ a\varepsilon V\rightarrow$:Q2P9

① , $a\varepsilon\{x:$ [redacted]$\}\rightarrow$:Q2P1

① , [redacted] $\rightarrow$:[redacted]$\rightarrow$

:05 starts here $[*_5:=a]$

① $\rightarrow$:$\forall\rightarrow$

① , $*_5\varepsilon a \leftrightarrow *_5\varepsilon\{x:\cup x\varepsilon x\}\rightarrow$:$\leftrightarrow\rightarrow$

"A" "B"

"A"

① , $*_5 \varepsilon a$, $*_5 \varepsilon \{x: \sim x \varepsilon x\}$ → :Q2P1

① , $*_5 \varepsilon a$, $\sim *_5 \varepsilon *_5$ → :∼ →

① , $*_5 \varepsilon a$, → $*_5 \varepsilon *_5$:Unify

[$*_5$:=a]

"B"

① → $*_5 \varepsilon a$, $*_5 \varepsilon \{x: \sim x \varepsilon x\}$:Q2P1

① → $*_5 \varepsilon a$, $\sim *_5 \varepsilon *_5$:→ ∼

① , $*_5 \varepsilon *_5$ → $*_5 \varepsilon a$:Unify

[$*_5$:=a]

"C"

→ $\sim \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\} \varepsilon V$:→ ∼

$\{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\} \varepsilon V$ → :Q5P5

$\exists y\ y = \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\} \wedge y \varepsilon V$ → :∃ →

$b = \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\} \wedge b \varepsilon V$ → :∧→

$b = \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\}$, $b \varepsilon V$ → :Q2P7

$(\forall x\ x \varepsilon b \leftrightarrow x \varepsilon \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\})$, $b \varepsilon V$ → :Q2P9

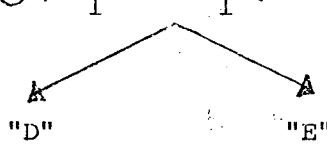
② , $b \varepsilon \{x:$ [redacted] $\}$ → :Q2P1

② , [redacted] → :¤ →

↓ :θ1 starts here:

② → :V →

② , $*_1 \varepsilon b \leftrightarrow *_1 \varepsilon \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\}$ → :↔ →



"D" "E"

"D"

② , $*_1 \varepsilon b$, $*_1 \varepsilon \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\} \rightarrow$:Q2P1

② , $*_1 \varepsilon b$, $\sim \exists y\ *_1 \varepsilon y \wedge y \varepsilon *_1 \rightarrow$:$\sim \rightarrow$

:θ2 starts here $[*_1 = b, *_2 = b]$

② $*_1 \varepsilon b \rightarrow \exists y\ *_1 \varepsilon y \wedge y \varepsilon *_1$:$\rightarrow \exists$

②, $*_1 \varepsilon b \rightarrow *_1 \varepsilon *_2 \wedge *_2 \varepsilon *_1$, ③ :$\rightarrow \wedge$

② , $*_1 \varepsilon b \rightarrow *_1 \varepsilon *_2$, ③ :Unify ▣ $[*_2 = b]$

② , $*_1 \varepsilon b \rightarrow *_2 \varepsilon *_1$, ③ :Unify ▣ $[*_1 = b, *_2 = *_1]$

"E"

② $\rightarrow *_1 \varepsilon b$, $*_1 \varepsilon \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\}$:Q2P1

② $\rightarrow *_1 \varepsilon b$, $\sim \exists y\ *_1 \varepsilon y \wedge y \varepsilon *_1$:$\rightarrow \sim$

② , $\exists y\ *_1 \varepsilon y \wedge y \varepsilon *_1 \rightarrow *_1 \varepsilon b$:$\exists \rightarrow$

② , $*_1 \varepsilon (c *_1) \wedge (c *_1) \varepsilon *_1 \rightarrow *_1 \varepsilon b$:$\wedge \rightarrow$

② , $*_1 \varepsilon (c *_1)$, $(c *_1) \varepsilon *_1 \rightarrow *_1 \varepsilon b$:propogation of θ1(θ2) unification

:θ3 starts here producing a second copy of ②

② , $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$:$\forall \rightarrow$

$*_3 \varepsilon b \leftrightarrow *_3 \varepsilon \{x: \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\}$, $b \varepsilon c, c \varepsilon b \rightarrow b \varepsilon b$:$\leftrightarrow \rightarrow$

"F" "G"

" "
F

② , $*_3 \varepsilon b$, $*_3 \varepsilon \{x : \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\}$, $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$:Q2Pl

② , $*_3 \varepsilon b$, $\sim \exists y\ *_3 \varepsilon y \wedge y \varepsilon *_3$, $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$:$\sim \rightarrow$

② , $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$, $(\exists y\ c \varepsilon y \wedge y \varepsilon c)$:Unify

:04 starts here $[*_4 := b]$

② , $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$, $\exists y\ c \varepsilon y \wedge y \varepsilon c$:$\rightarrow \exists$

② , $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$, $c \varepsilon *_4 \wedge *_4 \varepsilon c$, ④ :$\rightarrow \wedge$

② , $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$, $c \varepsilon *_4$, ④ :Unify

■ $[*_4 = b]$

② , $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$, $*_4 \varepsilon c$, ④ :Unify

■ $[*_4 = b]$

" "
G

② , $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$, $*_3 \varepsilon b$, $*_3 \varepsilon \{x : \sim \exists y\ x \varepsilon y \wedge y \varepsilon x\}$:Q2Pl

② , $b \varepsilon c$, $c \varepsilon b \rightarrow b \varepsilon b$, $*_3 \varepsilon b$, $\sim \exists y\ *_3 \varepsilon y \wedge y \varepsilon *_3$:$\rightarrow \sim$

② , $b \varepsilon c$, $c \varepsilon b$, $\exists y\ *_3 \varepsilon y \wedge y \varepsilon *_3 \rightarrow b \varepsilon b$, $*_3 \varepsilon b$:$\exists \rightarrow$

② , $b \varepsilon c$, $c \varepsilon b$, $*_3 \varepsilon d *_3 \wedge d *_3 \varepsilon *_3 \rightarrow b \varepsilon b$, $*_3 \varepsilon b$:$\wedge \rightarrow$

② , $b \varepsilon c$, $c \varepsilon b$, $*_3 \varepsilon d *_3$, $d *_3 \varepsilon *_3 \rightarrow b \varepsilon b$, $*_3 \varepsilon b$:Unify

■ $[*_3 = c]$

time = 733 milliseconds

Example 5

Q8P16 The intersection of all things in the universe is the nullset.

This protocol illustrates the addition of the axioms as extra hypotheses in a sequent.

The following abbreviations are used:

① = df(∀y yε$\mathbb{V}$ ⊃ aεy)

② = df(∀y ∀z(∀x xεy ⊃ xεz) ⊃ (∀w yεw ⊃ zεw))

③ = df ∀x∀y ∃w ∀u uεw ↔ u=x ∨ u=y

④ = df ∀u ∿ uεb

⑤ = df (∀y ∃w ∀u uεw ↔ u=$*_3$ ∨ u=y)

⑥ = df (∀z (∀x x$\varepsilon *_5$ ↔ xεz) ⊃ (∀w $*_5$εw ⊃ zεw))

→ $\bigcap\mathbb{V}$= ∅ :Q2P7

→ ∀x xε$\bigcap\mathbb{V}$ ↔ xε∅ :→ ∀

→ aε$\bigcap\mathbb{V}$ ↔ aε∅ :→ ↔

aε$\bigcap\mathbb{V}$ → aε∅ :Q8P9

aε{x:∀y yε$\mathbb{V}$ → xεy} ⊃ aε∅ :Q2P1

(∀y yε$\mathbb{V}$ ⊃ aεy) → aε∅ :Q2P8

① → aε{x: ¤} :Q2P1

① → ¤ :→ ¤

aε∅ → aε :Q2P8

aε{x:¤ } → aε$\bigcap\mathbb{V}$:Q2P1

¤ → aε$\bigcap\mathbb{V}$:¤ →

:01 starts here

① → :∀ →

①, $*_1$ε$\mathbb{V}$ ⊃ aε$*_1$ → :⊃ →

:02 starts here ①, → $*_1 \varepsilon V$:Q2P9

①, $a\varepsilon *_1$ → :∀ → ① → $*_1 \varepsilon \{x: \blacksquare\}$:Q2P1

①, $*_2 \varepsilon V \supset a\varepsilon *_2$, $a\varepsilon *_1$ → : ⊃ → ① → ■ :→ ■

■

:03 starts here ① , $a\varepsilon *_1$ → $*_2 \varepsilon V$:Q2P9

①, $a\varepsilon *_1$, $a\varepsilon *_2$:add axioms ① , $a\varepsilon *_1$ → $*_2 \varepsilon \{x: \blacksquare\}$:Q2P1

① , $a\varepsilon *_1$ → ■ :→ ■

■

①, $a\varepsilon *_1$, $a\varepsilon *_2$, $(\forall y\ \forall z\ (\forall x\ x\varepsilon y \leftrightarrow x\varepsilon z) \supset (\forall w\ y\varepsilon w \supset z\varepsilon w))$,

$(\exists w\ \forall u\ \sim u\varepsilon w)$, $(\forall x \forall y \exists w \forall u\ u\varepsilon w \leftrightarrow u=x \vee u=y)$:∃ →

:04 starts here

①, $a\varepsilon *_1$, $a\varepsilon *_2$, ②, ③, $\forall u \sim u\varepsilon b$ → :∀ →

①, ②, ③, ④, $a\varepsilon *_1$, $a\varepsilon *_2$, $(\forall y \exists w\ \forall u\ u\varepsilon w \leftrightarrow u=*_3 \vee u=y)$ →:∀ →

①, ②, ③, ④, ⑤, $a\varepsilon *_1$, $a\varepsilon *_2$, $\sim *_4 \varepsilon b$ → :∀ →

①, ②, ③, ④, ⑤, $a\varepsilon *_1$, $a\varepsilon *_2$, $\sim *_4 \varepsilon b$,

$(\forall z\ (\forall x\ x\varepsilon *_5 \leftrightarrow x\varepsilon z) \supset (\forall w *_5 \varepsilon w \supset z\varepsilon w))$ → :∼ →

①, ②, ③, ④, ⑤, ⑥, $a\varepsilon *_1$, $a\varepsilon *_2$ → $*_4 \varepsilon b$

■ $[*_1=b, *_4=a]$ or $[*_2=b, *_4=a]$

time = 426 milliseconds

Example 6

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

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

① = df $(\forall w\ \alpha \cap \{y:\cup y \varepsilon w'y\} \varepsilon V)$

② = df $(\forall x\ x \varepsilon P\alpha \rightarrow x \varepsilon f''\alpha)$

③ = df $(\forall x\ x \varepsilon (a^*_2) \leftrightarrow x \varepsilon \alpha \cap \{y:\cup y \varepsilon {}^*_2'y\})$

④ = df $(\forall x\ x \varepsilon (a^*_2) \leftrightarrow x \varepsilon f'(b^*_2)$

$\rightarrow (\forall w\ \alpha \cap \{y:\cup y \varepsilon w'y\} \varepsilon V) \supset \alpha < P\alpha$	:→⊃
$(\forall w\ \alpha \cap \{y:\cup y \varepsilon w'y\} \varepsilon V) \rightarrow \alpha < P\alpha$	:Q2OP3
① $\rightarrow \cup P\alpha \leq \alpha$	:→ ∪
①, $P\alpha \leq \alpha \rightarrow$	:Q11P1
①, $\exists x$ Func $x \wedge P\alpha \subseteq x''\alpha \rightarrow$	:∃ →
①, Func $f \wedge P\alpha \subseteq f''\alpha \rightarrow$	: ∧→
①, Func f, $P\alpha \subseteq f''\alpha \rightarrow$	:Q2P2
:θ1 starts here	
①, Func f, $(\forall x\ x \varepsilon P\alpha \supset x \varepsilon f''\alpha) \rightarrow$	:∀ →
①, Func f, ${}^*_1 \varepsilon P\alpha \supset {}^*_1 \varepsilon f''\alpha$, ② →	:∀ →
①, ②, Func f, $\alpha \cap \{y:\cup y \varepsilon {}^*_2'y\} \varepsilon V$, ${}^*_1 \varepsilon P\alpha \supset {}^*_1 \varepsilon f''\alpha \rightarrow$	:Q5P5
①, ②, Func f, $\exists x\ x = \alpha \cap \{y:\cup y \varepsilon {}^*_2'y\} \wedge x \varepsilon V$, ${}^*_1 \varepsilon P\alpha \supset {}^*_1 \varepsilon f''\alpha \rightarrow$	:∃ →

① , ② , Func f, $(a^*_2)=\alpha \cap \{y:\sim y\varepsilon *_2 'y\} \wedge (a^*_2)\varepsilon V$,

$*_1\varepsilon P\alpha \supset *_1\varepsilon f"\alpha \rightarrow$: $\wedge \rightarrow$

① , ② , Func f, $(a^*_2)=\alpha \cap \{y:\sim y\varepsilon *_2 'y\}$, $(a^*_2)\varepsilon V$,

$*_1\varepsilon P\alpha \supset *_1\varepsilon f"\alpha \rightarrow$:Q2P9

① , ② , Func f, $(a^*_2)=\alpha \cap \{y:\sim y\varepsilon *_2 'y\}$,

$(a^*_2)\varepsilon\{y:$ [illegible]$\}$, $*_1\varepsilon P\alpha \supset *_1\varepsilon f"\alpha \rightarrow$:Q2P1

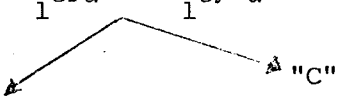
① , ② , Func f, $(a^*_2)=\alpha \cap \{y:\sim y\varepsilon *_2 'y\}$, [illegible] ,

$*_1\varepsilon P\alpha \supset *_1\varepsilon f"\alpha \rightarrow$:Q2P7

① , ② , Func f, $(\forall x\ x\varepsilon(a^*_2) \leftrightarrow x\varepsilon\alpha \cap \{y:\sim y\varepsilon *_2 'y\})$,

[illegible] , $*_1\varepsilon P\alpha \supset *_1\varepsilon f"\alpha \rightarrow$: [illegible] $\rightarrow$

① , ② , Func f, ③ , $*_1\varepsilon P\alpha \supset *_1\varepsilon f"\alpha \rightarrow$: $\supset \rightarrow$



① , ② , ③ , Func f, $*_1\varepsilon f"\alpha \rightarrow$:Q9P14

① , ② , ③ , Func f, $*_1\varepsilon\{x:\exists y<xy>\varepsilon f \wedge y\varepsilon\alpha\}$:Q2P1

① , ② , ③ , Func f, $\exists y\ <*_1 y>\varepsilon f \wedge y\varepsilon\alpha \rightarrow$: $\exists \rightarrow$

① , ② , ③ , Func f, $<*_1(b^*_1)> \varepsilon f \wedge (b^*_1)\varepsilon\alpha \rightarrow$: $\wedge \rightarrow$

① , ② , ③ , Func f, $<*_1(b^*_1)> \varepsilon f$, $(b^*_1)\varepsilon\alpha \rightarrow$:CRL1*

① , ② , ③ , Func f, $*_1=f'(b^*_1)$, $(b^*_1)\varepsilon\alpha \rightarrow$:Q2P7

① , ② , ③ , Func f, $(\forall x\ x\varepsilon *_1 \leftrightarrow x\varepsilon f'(b^*_1)), (b^*_1)\varepsilon\alpha \rightarrow$:propagation from Θ2

:Θ3 starts here

① , ② , ③ , Func f, $(\forall x\ x\varepsilon(a^*_2) \leftrightarrow x\varepsilon f'(b^*_2))$,

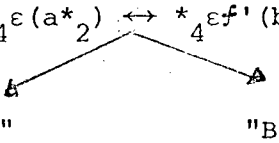
$(b^*_2)\varepsilon\alpha \rightarrow$: $\forall \rightarrow$

① , ② , ③ , Func f, ④ , $*_4\varepsilon(a^*_2) \leftrightarrow *_4\varepsilon f'(b^*_2)$,

$(b^*_2)\varepsilon\alpha \rightarrow$: $\forall \rightarrow$

① , ② , ③ , ④ , Func f, $*_5\varepsilon(a^*_2) \leftrightarrow *_5\varepsilon\alpha \cap$

$\{y:\sim y\varepsilon *_2 'y\}$, $*_4\varepsilon(a^*_2) \leftrightarrow *_4\varepsilon f'(b^*_2)$, $(b^*_2)\varepsilon\alpha \rightarrow$: $\leftrightarrow \rightarrow$



"A"

①, ②, ③, ④, Func f, $*_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha \cap \{y:\sim y\varepsilon *_2{}'y\}$,

$*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$, $(b*_2)\varepsilon\alpha \rightarrow$:Q2P5

↓

①, ②, ③, ④, Func f, $*_5\varepsilon(a*_2)$, $*_5\varepsilon\{x:x\varepsilon\alpha \wedge x\varepsilon\{y:\sim y\varepsilon *_2{}'y\}\}$,

$*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$, $(b*_2)\varepsilon\alpha \rightarrow$:Q2P1

①, ②, ③, ④, Func f, $*_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha \wedge *_5\varepsilon\{y:\sim y\varepsilon *_2{}'y\}$

$*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$, $(b*_2)\varepsilon\alpha \rightarrow$: $\wedge \rightarrow$

①, ②, ③, ④, Func f, $*_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha$, $*_5\varepsilon\{y:\sim y\varepsilon *_2{}'y\}$

$*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$, $(b*_2)\varepsilon\alpha \rightarrow$:Q2P1

①, ②, ③, ④, Func f, $*_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha$, $\sim *_5\varepsilon *_2{}'*_5$

$*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$, $(b*_2)\varepsilon\alpha \rightarrow$:$\sim \rightarrow$

①, ②, ③, ④, Func f, $*_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha$, $*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon *_2{}'*_5$:$\leftrightarrow \rightarrow$

①, ②, ③, ④, Func f, $*_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha$, $*_4\varepsilon(a*_2)$, $*_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon *_2{}'*_5$:Unify

$[*_2=f,\ *_4=*_5,\ *_5=(b*_2)]$

■

①, ②, ③, ④, Func f, $*_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha$, $(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon *_2{}'*_5$,

$*_4\varepsilon(a*_2)$, $*_4\varepsilon f'(b*_2)$:Unify

$[*_4=*_5]$

■

"B"

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha \frown \{y:\sim y\varepsilon *_2{}'y\}$:Q2P5

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon(a*_2)$, $*_5\varepsilon\{x:x\varepsilon\alpha \wedge x\varepsilon\{y:\sim y\varepsilon *_2{}'y\}\}$:Q2P1

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha \wedge *_5\varepsilon\{y:\sim y\varepsilon *_2{}'y\}$:→∧

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha$:↔ →

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2)$, $*_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha$:Unify

[$*_4=*_5$] or [$*_5=(b*_2)$]

①, ②, ③, ④, Func f, $(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon(a*_2)$, $*_5\varepsilon\alpha$,

$*_4\varepsilon(a*_2)$, $*_4\varepsilon f'(b*_2)$:Unify

[$*_5=(b*_2)$]

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon(a*_2)$, $*_5\varepsilon\{y:\sim y\varepsilon *_2{}'y\}$:Q2P1

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$,

$(b*_2)\varepsilon\alpha \rightarrow *_5\varepsilon(a*_2)$, $\sim *_5\varepsilon *_2{}'*_5$:→ ∼

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2) \leftrightarrow *_4\varepsilon f'(b*_2)$,
$(b*_2)\varepsilon\alpha$, $*_5\varepsilon*_2'*_5 \rightarrow *_5\varepsilon(a*_2)$:$\leftrightarrow \rightarrow$

①, ②, ③, ④, Func f, $*_4\varepsilon(a*_2)$, $*_4\varepsilon f'(b*_2)$, $(b*_2)\varepsilon\alpha$,
$*_5\varepsilon*_2'*_5 \rightarrow *_5\varepsilon(a*_2)$:Unify

$[*_4=*_5]$

①, ②, ③, ④, Func f, $(b*_2)\varepsilon\alpha$, $*_5\varepsilon*_2'*_5 \rightarrow *_5\varepsilon(a*_2)$,
$*_4\varepsilon(a*_2)$, $*_4\varepsilon f'(b*_2)$:Unify

$[*_2=f,\ *_4=*_5,\ *_5=(b*_2)]$

"C"

①, ②, ③, Func $f \rightarrow *_1 \varepsilon P\alpha$:D1

①, ②, ③, Func $f \rightarrow *_1\varepsilon\{x:x\subseteq\alpha\}$:Q2P1

①, ②, ③, Func $f \rightarrow *_1 \subseteq \alpha$:Q2P2

①, ②, ③, Func $f \rightarrow \forall x\ x\varepsilon*_1 \supset x\varepsilon\alpha$:$\rightarrow \forall$

①, ②, ③, Func $f \rightarrow (c*_1)\varepsilon*_1 \supset (c*_1)\varepsilon\alpha$:$\rightarrow \supset$

:θ2 starts here

①, ②, ③, Func f, $(c*_1)\varepsilon*_1 \rightarrow (c*_1)\varepsilon\alpha$:$\forall \rightarrow$

①, ②, ③, $*_3\varepsilon(a*_2) \leftrightarrow *_3\varepsilon\alpha \cap \{y:\sim y\varepsilon*_2'y\}$,
Func f, $(c*_1)\varepsilon*_1 \rightarrow (c*_1)\varepsilon\alpha$:$\leftrightarrow \rightarrow$

"D"

①, ②, ③, $*_3\varepsilon(a*_2)$, $*_3\varepsilon\alpha \cap \{y:\sim y\varepsilon*_2'y\}$,
Func f, $(c*_1)\varepsilon*_1 \rightarrow (c*_1)\varepsilon\alpha$:Q2P5

① , ② , ③ , $*_3\varepsilon(a*_2)$, $*_3\varepsilon\{x:x\varepsilon\alpha \wedge x\varepsilon\{y:\sim y\varepsilon *_2 'y\}\}$,

Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$:Q2P1

① , ② , ③ , $*_3\varepsilon(a*_2)$, $*_3\varepsilon\alpha \wedge *_3\varepsilon\{y:\sim y\varepsilon *_2 'y\}$,

Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$: $\wedge\rightarrow$

① , ② , ③ , $*_3\varepsilon(a*_2)$, $*_3\varepsilon\alpha$, $*_3\varepsilon\{y:\sim y\varepsilon *_2 'y\}$,

Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$:Q2P1

① , ② , ③ , $*_3\varepsilon(a*_2)$, $*_3\varepsilon\alpha$, $\sim *_3\varepsilon *_2 '*_3$,

Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$:$\sim\rightarrow$

① , ② , ③ , $*_3\varepsilon(a*_2)$, $*_3\varepsilon\alpha$, Func f, $(c*_1)\varepsilon *_1 \rightarrow *_3\varepsilon *_2 '*_3$,

$(c*_1)\varepsilon\alpha$:Unify

■ $[*_3=(c*_1)]$

"D"

① , ② , ③ , Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$, $*_3(a*_2)$,

$*_3\varepsilon\alpha \frown \{y:\sim y\varepsilon *_2 'y\}$:Q2P5

① , ② , ③ , Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$, $*_3\varepsilon(a*_2)$,

$*_3\varepsilon\{x:x\varepsilon\alpha \wedge x\varepsilon\{y:\sim y\varepsilon *_2 'y\}\}$:Q2P1

① , ② , ③ , Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$, $*_3\varepsilon(a*_2)$,

$*_3\varepsilon\alpha \wedge *_3\varepsilon\{y:\sim y\varepsilon *_2 'y\}$: $\rightarrow\wedge$

① , ② , ③ , Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$, $*_3\varepsilon(a*_2)$, $*_3\varepsilon\alpha$:Unify

■ $[*_1=(a*_2), *_3=(c*_1)]$

① , ② , ③ , Func f, $(c*_1)\varepsilon *_1 \rightarrow (c*_1)\varepsilon\alpha$, $*_3\varepsilon(a*_2)$,

$*_3\varepsilon\{y:\sim y\varepsilon *_2 'y\}$:Q2P1

① , ② , ③ , Func f, $(c^*_1)\varepsilon^*_1 \rightarrow (c^*_1)\varepsilon\alpha$, $^*_3\varepsilon(a^*_2)$,

$\sim^*_3\varepsilon^*_2\,'^*_3$:→ ∼

① , ② , ③ , Func f, $(c^*_1)\varepsilon^*_1$, $^*_3\varepsilon^*_2\,'^*_3 \rightarrow (c^*_1)\varepsilon\alpha$,

$^*_3\varepsilon(a^*_2)$:Unify

■ $[^*_1=(a^*_2),\ ^*_3=(c^*_1)]$

time = 1,494 milliseconds or approximately 1½ seconds

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

Variable	Formulae	Instantiation	Branch of Proof
$*_1$	②	a	$\theta_1(02)$
$*_2$	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 its 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 01: and is to decide which of the following two formulas to work on first.

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

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

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

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

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

---

* 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.

working on the first formulae. However, this first formulae in a few steps reduces to ③. If 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).

The trickyness of these logical operations is, we feel, one reason why Cantor's theorem is so surprising.

In set theories such as ZF (Zermelo Fraenkel) Cantor's theorem is usually thought of as the sentence: that each set have smaller cardinality than its powerset:

$$\forall x \; x < Px$$

and indeed this sentence is easily derived from Quine's version of Cantor's theorem which we have just proven.

$$\forall w \; \alpha \frown \{y : \sim y \varepsilon w' y\} \; \varepsilon V \supset \alpha < P\alpha$$

For if α is a set x:

$$\forall w \; x \frown \{y : \sim y \varepsilon w' y\} \; \varepsilon V \supset x < Px$$

the hypothesis of this implication is an instance of the Assonderrung axiom of ZF.

$$\forall x \; x \frown \beta \; \varepsilon \; V$$

and hence is true in ZF. Thus, by modus pones it follows that $x<Px$ is also true in ZF.

4. Results

Our main purpose in implementing this theorem prover was to see how well it performed as an extensible deductive system. That is, as a system which starting from the axioms of a theory proceeds by successively proving a theorem and then assimulating what it has just proven thus creating an extension of the previous system from which further theorems may be proven.

To be clear we point out that this work is not primarily concerned with the process of assimulating a theorem and creating an extension* and in fact these processes of theory construction are not carried out by this theorem prover, but were merely preset by the author. Rather, our work is primarily concerned with whether the deductive aspects of such a process can in fact be carried out.

For this reason, what we did was to take Quine's book Set Theory and its Logic [1], and try using our theorem prover to automatically prove the theorems in that book in the same order in which they were proven in that book not assuming as extra knowledge any theorems of set theory which had not already been proven at that point in the book.

For any theorem in Quine's book the reader will be able to determine exactly which lemmas listed in Table 3 and Table 5 of Section 2 were used to prove it, by simply noting whether the name of the lemma precedes the name of that theorem. For example,

---

* e.g. the choice of theorems to assimulate, how a theorem is assimulated.

the name of the theorem Q6P10 is "Q6P10" which indicates that it is the tenth numbered formulae in Chapter 6 of Quine's book. Likewise the lemma Q6P4 is the fourth numbered formulae of Chapter 6 and hence precedes Q6P10. Thus it is assumed when the system tries to prove Q6P10. On the other hand the lemma Q7P7 occurring in Chapter 7 succeeds Q6P10 and hence is not assumed when trying to prove Q6P10. There are also a number of miscellaneous theorems either unnumbered or not occurring in Quine's book. The names of these theorems begin with the letters "TH", and all the lemmas in Tables 3 and 5 were assumed when trying to prove these theorems.

Our results were as follows: Skipping over the first four chapters of Quine's book which contained no numbered formulae, we were able to prove in an extensible fashion fifty-six out of sixty theorems in Chapters 5, 6, 7, 8 and 9. A summary of results for each chapter is given in Table 7 and individual results for each theorem which was proven are given in Table 8. The six theorems in these five chapters which the theorem prover failed to prove are given in Table 9.

We believe that failure to prove these six theorems was not due to the lack of relevant set theoretic lemmas, but rather due to the incompleteness of the underlying sequent logic caused by the forcing restriction. Finally, in Table 10 we list individual results of a few other theorems the system has proven.

Table 7: Summary of results of Chapters 5, 6, 7, 8 and 9 in Quine's book: Set Theory and its Logic. (Note: there were no numbered theorems in Chapters 1, 2, 3, and 4.)

Chapter	No. of theorems	Proven	Failed
Q5	4	4	0
Q6	16	16	0
Q7	12	9	3
Q8	21	19	2
Q9	7	6	1

Table 8: Theorems Proven in book order

Name	Theorem	Conses	time(ms)	Sequents	Unify	GC	Axioms
Q5P1	$\alpha=\{x:\Gamma x\}\leftrightarrow(\forall x\ x\varepsilon\alpha\leftrightarrow\Gamma x)$	1,059	302	43	4	0	-
Q5P2	$\alpha=\{x:x\varepsilon\alpha\}$	192	48	9	0	0	-
Q5P3	$\forall y\sim\forall x(x\varepsilon y\leftrightarrow\sim x\varepsilon x)$	283	85	10	1	0	-
Q5P4	$\forall y\ \forall x(x\varepsilon y\leftrightarrow\sim(\exists y\ x\varepsilon y\wedge y\varepsilon x))$	1,827	539	29	4	0	-
Q6P1	$\forall y\ \Gamma y\leftrightarrow(\exists x\ x=y\wedge\Gamma x)$	477	130	18	1	1	-
Q6P2	$\forall y\ \Gamma y\leftrightarrow(\forall x\ x=y\supset\Gamma x)$	481	138	18	1	0	-
Q6P3	$\alpha\varepsilon\beta\leftrightarrow(\exists x\ x=\alpha\wedge x\varepsilon\beta)$	10,034	5,105	153	6	0	-
Q6P4*	$\alpha=\alpha$	145	57	7	0	0	-
Q6P5	$\alpha=\beta\leftrightarrow\beta=\alpha$	1,258	366	39	4	0	-
Q6P6	$(\alpha=\beta\wedge\phi\alpha)\supset\phi\beta$	65	25	5	0	0	-
Q6P7	$\alpha=\beta\supset(\phi\alpha\leftrightarrow\phi\beta)$	86	51	7	0	0	-
Q6P8	$\forall x\ x\varepsilon V$	99	44	3	0	0	-
Q6P9	$\alpha\varepsilon V\leftrightarrow\exists x\ x=\alpha$	552	185	19	1	0	-

Name	Theorem	Conses	time(ms)	Sequents	Unify	GC	Axioms
Q6P10	$\sim\{x:\sim x\varepsilon x\}\varepsilon V \wedge \sim\{x:\sim\exists y\ x\varepsilon y\wedge y\varepsilon x\}\varepsilon V$	2,743	773	56	5	0	-
Q6P11	$((\alpha\varepsilon V\wedge\forall x\ \phi x)\supset\phi\alpha)\wedge((\alpha\varepsilon V\wedge\phi\alpha)\supset\exists x\phi x)$	551	177	23	2	0	A
Q6P12	$\alpha\varepsilon\beta\supset\alpha\varepsilon V$	200	64	8	0	0	-
Q6P13	$\alpha\varepsilon\{x:\phi x\}\leftrightarrow\alpha\varepsilon V\wedge\phi\alpha$	418	167	21	0	0	-
Q6P14	$\sim\alpha\varepsilon\emptyset$	167	72	7	0	0	-
Q6P15	$\forall x.\ x\varepsilon\alpha\leftrightarrow\alpha=V$	850	284	34	2	0	-
Q6P16	$\forall x\ \Gamma x\leftrightarrow\{x:\Gamma x\}=V$	1,271	411	56	4	0	-
Q7P2	$\sim\alpha\varepsilon V\leftrightarrow\{\alpha\}=\emptyset$	1,494	456	47	2	0	-
Q7P3	$(\alpha\varepsilon\beta\wedge\forall x\phi x)\supset\phi\alpha$	257	271	10	1	0	-
Q7P4	$\forall x\forall y\{x\}\subseteq y\leftrightarrow x\varepsilon y$	849	271	23	1	0	-
Q7P5	$\forall x\forall y\forall z\{x\ y\}\subseteq z\leftrightarrow x\varepsilon z\wedge y\varepsilon z$	2,260	742	66	2	0	-
Q7P6	$\forall x\forall y\ x\varepsilon\{x\}\wedge x\varepsilon\{x\ y\}\wedge y\varepsilon\{x\ y\}$	558	182	17	0	0	-
Q7P7*	$\forall x\forall y\{x\}=\{y\}\leftrightarrow x=y$	2,407	945	67	2	0	-
Q7P8*	$\forall x\forall y\forall z\{x\ y\}=\{z\}\leftrightarrow x=z\wedge y=z$	8,722	3,717	252	4	0	-
Q7P9*	$\forall x\forall y\forall z\{x\ y\}=\{x\ w\}\leftrightarrow y=w$	21,167	14,132	620	6	1	-
Q7P11	$\forall x\exists y\ x\varepsilon y$	80,370	58,331	72	6	9	A
Q8P2	$\forall x\ \bigcup\{x\}=x$	1,028	335	24	1	0	-
Q8P3	$\forall x\forall y\ \bigcup(x\cup y)=\bigcup x\cup\bigcup y$	3,303	1,038	63	4	0	-
Q8P4	$\forall x\forall y\ \bigcup\{xy\}=x\cup y$	2,607	873	69	2	0	-
Q8P5	$\bigcup\alpha\subseteq\beta\leftrightarrow(\forall x\ x\varepsilon\alpha\supset x\subseteq\beta)$	2,137	667	35	4	0	-
Q8P6	$\forall x\ x\varepsilon\alpha\supset x\subseteq\bigcup\alpha$	522	171	12	1	0	-
Q8P7	$\bigcup\emptyset=\emptyset$	434	167	11	0	0	-
Q8P8	$\bigcup V=V$	81,093	63,456	91	6	9	A
Q8P10	$\forall x\ \bigcap\{x\}=x$	1,047	366	24	1	0	-

Name	Theorem	Conses	time(ms)	Sequents	Unify	GC	Axioms
Q8P11	$\bigcap\alpha\cup\beta=\bigcap\alpha\cap\bigcap\beta$	3,203	1,161	61	4	0	-
Q8P12	$\forall x\forall y\bigcap\{xy\}=x\cap y$	2,667	1,026	69	2	0	-
Q8P13	$\beta\subseteq\bigcap\alpha\leftrightarrow(\forall x\ x\varepsilon\alpha\supset\beta\subseteq x)$	2,166	2,218	35	4	0	-
Q8P14	$\forall x\ x\varepsilon\alpha\supset\bigcap\alpha\subseteq x$	523	189	12	1	0	-
Q8P15	$\bigcap\emptyset=V$	608	252	14	0	0	-
Q8P16	$\bigcap V=\emptyset$	1,234	426	25	4	0	A
Q8P17	$\forall y.\ (\forall x\ \Gamma x\leftrightarrow x=y)\supset(\bigcup\{x:\Gamma x\}=y$ $\wedge\ \bigcap\{x:\Gamma x\}=y)$	3,548	1,341	99	4	0	-
Q8P19*	$\forall y\ (\forall x\ \Gamma x\leftrightarrow x=y)\supset\eta x\Gamma x=y$	12,133	7,113	231	4	1	-
Q8P20	$\forall y\ \forall z\ (\forall x\ \Gamma x\leftrightarrow x=y)\supset$ $(\Gamma z\leftrightarrow z=\eta z\Gamma z)$	114,892	78,229	1,044	22	13	-
Q8P21	$\forall y\ y=\eta x(x=y)$	2,279	801	57	2	0	-
Q8P22*	$(\sim\exists y\forall x\ \Gamma x\leftrightarrow x=y)\supset\eta x\Gamma x=\emptyset$	7,059	3,318	148	3	0	-
Q9P2	$\forall x\forall y\forall u\forall v<xy>=<uv>\supset x=u\wedge y=v$	1,225	408	42	0	0	-
Q9P3*	$\forall x\forall y\forall u\forall v<xy>=<uv>\leftrightarrow x=u\wedge y=v$	1,601	432	47	0	0	-
Q9P5*	$\forall z\forall w<zw>\varepsilon\{<xy>:\Gamma xy\}\leftrightarrow\Gamma zw$	1,564	457	35	2	0	-
Q9P7	$^{\circ}\alpha=\alpha\frown{}^{\circ}V$	1,905	595	27	0	0	-
Q9P8	$^{\circ}\alpha\subseteq\beta\leftrightarrow(\forall x\forall y<xy>\varepsilon\alpha\supset<xy>\varepsilon\beta)$	3,096	968	46	5	0	-
Q9P10	$^{\circ}\{<xy>:\Gamma xy\}=\{<xy>:\Gamma xy\}$	1,851	607	31	0	0	-

*These theorems are needed in order to justify lemmas in Tables 3 and 5 which were used in proving later theorems.

Table 9: The theorems of Chapters 5, 6, 7, 8 and 9 which the system failed to prove.

Q7P12 $\{\alpha\}\varepsilon V$

Q7P13 $\{\alpha\beta\}\varepsilon V$

Q7P14 $\alpha=z \leftrightarrow \forall x\ z\varepsilon x \supset \alpha\varepsilon x$

Q8P23 $\eta\, x\phi x\varepsilon V$

Q8P24 $\forall y((\forall x\ \phi x \leftrightarrow x=y) \supset \phi(\eta\, x\phi x))$

Q9P9 ${}^{o}\alpha={}^{o}\beta \leftrightarrow \forall x\ \forall y\ \langle xy\rangle\varepsilon\alpha \leftrightarrow \langle xy\rangle\varepsilon\beta$

Table 10: Some miscellaneous theorems which have been proven.

Name	Theorem	Conses	time(ms)	Sequents	Unify
Q10P3	Func $\emptyset$	1,091	367	21	0
Q10P4	Func I $\wedge$(arg I) = V	3,702	1,203	69	1
Q10P5	$\forall x\ \forall y$ Func $\{\langle xy\rangle\}$	2,482	1,352	43	0
Q10P9	arg $\alpha \subseteq \alpha$ " V	1,466	478	29	1
Q10P19	$\forall x$ I 'x = x	3,066	1,095	61	2
Q10P20	$\emptyset$ '$\alpha = \emptyset$	1,261	435	25	1
Q28P17	$\forall w\ \alpha \wedge \{y:\sim y\varepsilon w\ y\}\varepsilon V \supset \alpha<P\alpha$	4,621	1,494	76	3
THS1	$(\alpha \cup \beta) \cup \delta = \alpha \cup (\beta \cup \delta)$	1,440	472	42	0
THS2	$\alpha \cup \beta = \beta \cup \alpha$	805	280	25	0
THS3	$\alpha \cup \alpha = \alpha$	482	169	15	0
THS4	$\alpha \cup \emptyset = \alpha$	520	178	16	0
THS5	$(\alpha \wedge \beta) \wedge \delta = \alpha \wedge (\beta \wedge \delta)$	1,489	495	42	0
THS6	$\alpha \wedge \beta = \beta \wedge \alpha$	832	289	25	0
THS7	$\alpha \wedge \alpha = \alpha$	502	147	16	0
THS8	$\alpha \wedge \emptyset = \emptyset$	434	148	13	0

Name	Theorem	Conses	time(ms)	Sequents	Unify
THS22	$\forall x\ \forall y\ \{x\} \cup \{y\} = \{x\ y\}$	2,176	753	67	0
THS31	$\delta"\ (\alpha \cup \beta) = (\delta\ "\alpha) \cup (\delta"\ \beta)$	3,534	1,183	61	4
THS32	$\cup \cup \alpha = \alpha$	1,840	612	21	0
THS101	$\alpha \times \beta \leq P\ P(\alpha \cup \beta)$	2,184	626	41	0

5. Implementation

Our theorem prover is implemented in three modules of LISP [9] code. The first module SYEV5 consists of the symbolic interpreter for mathematical symbols described in Section 2, and the sequent logic described in Section 2.1. It also includes another experimental logic system which,however, has not been used in proving any of the results given in this paper. The second module SET1 consists of the set theory axioms and lemmas described in Section 2.2, and also a few top level user routines related specifically to set theory. The third module contains all the definitions and theorems listed in Sections 2 and 4. The size of each of these three modules is listed in Table 11 along with their contribution to the run time structure of the theorem prover.

Our theorem prover was initially developed several years ago to run in about a 35k Stanford AI LISP 1.6 system [18]. We subsequently modified our theorem prover to run, as of September 1976, in a 45k UCI LISP system [19]. Our theorem prover can of course be run in much less space than this at the expense of extra garbage collections but it runs very well in this much space.

Table 11: Implementation of the theorem prover

Module	Source Blocks	Compile time	LAP Blocks	BPS Words	Free Words	Full Words
SYEV5	74	136sec	243	5,631	2,322	447
SET1	14	21sec	48	1,043	540	100
SETDEF	25	-	-	0	4,267	197
Workspace in 45k system (with 16,000 octal BPS allocated)				-	8,543	690
Total:				7,168	15,662	1,434

Total space used by theorem prover:	24,264
The LISP interpreter, stacks and remaining BPS	20,736
Grand Total:	45,000

Notes:

1. One block is 512 words of disc storage.
2. One k is 1024 words of core memory.
3. The Edinburgh implementation of UCI LISP does not contain all the functions as specified by the UCI LISP manual [18]. This perhaps explains why the size of our AI LISP interpreter is about 3k short of the 24k size (14k high segment, 8k low segment) specified by that manual.

The times given in the last section (Section 4) were obtained during September 1976 on a DEC10-KA10 with SYEV5 and SET1 compiled, with NOUUO=nil, and without storing the proof onto the disc for later examination. The compilation times given in Table 11 however were obtained using a DEC10-KI10. (Our DEC10 was upgraded in October 1976).

6. Conclusions

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. We have also seen that it is fairly successful at proving a limited number of successive theorems in a book, in an extensible fashion. In particular, we have seen that it was able to prove in an extensible manner, most of the theorems in Chapters 5, 6, 7, 8 and 9 of Quine's book Set Theory and its Logic.

VI Conclusions

We have argued that the improved performance of modern theorem provers is due basically to the use of domain dependent deductive knowledge possessing certain simple properties. We have exemplified this thesis by discussing various theorem provers for the propositional logic and elementary arithmetic. In particular we have implemented a deductive knowledge system for the domain of elementary arithmetic which is based on the use of rules possessing these properties.

We then have argued that domain dependent deductive knowledge systems, such as our arithmetic theorem prover, are not the kind of deductive system that is needed as a component of a general mathematical reasoning system. The reason for this being that such systems are not extensible in that they cannot assimulate and use new deductive knowledge created by a mathematical reasoning system. Finally, we have described a prototype deductive knowledge system for the domain of elementary set theory which is extensible in this sense.

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. I thank Peggy Avison for typing this thesis, and finally, I thank my family, and especially my broker for looking after my affairs in Texas.

References

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

2. F. Malloy Brown. "Doing Arithmetic without Diagrams". DAI Research Report No. 16a. 1976 and Journal of Artificial Intelligence, Vol. 8, 1977.

3. Anthony P. Morse. A Theory of Sets. Academic Press, London 1965.

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

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

6. Robert S. Boyer and J. Strother Moore. "Proving Theorems about LISP functions". IJCAI3 Proceedings, 1973.

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

8. R. Aubin. Mechanizing Structural Induction. Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh, 1976.

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

10. Hao Wang. "Toward Mechanical Mathematics", IBM Journal of Research and Development, Vol. 4, 1960, pp.2-22.

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

12. J. A. Robinson. "A Machine-Oriented Logic based on the Resolution Principle". JACM, 12, January, 1965, pp 23-41.

13. W. Bibel and J. Schreiber. "Proof Search in Gentzen-like System of First Order Logic". International Computing Symposium, 1975, North Holland Publishing Company.

14. J. Siekmann. "A Modification of the Unification Algorithm in Automatic Theorem Proving". Masters Thesis, University of Essex, 1972.

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

16. Dallas S. Lankford. "Canonical Algebraic Simplification". Maths Department Memo ATP-25, University of Texas at Austin, May 1975.

17. Pat Hayes. "A Theory of Action". Machine Intelligence 6, 1971.

18. Lynn Quam. "Stanford AI LISP 1.6 Manual". Stanford Artificial Intelligence Laboratory Operating Note 28.6.

19. R. Bobrow et al. "UCI LISP Manual". Technical Report 21, Department of Information and Computer Science, University of California, Irvine.

20. Alan Bundy. "Doing Arithmetic with Diagrams". DCL Report No. 61, 1973.

21. Alan Bundy. "Doing Arithmetic with Diagrams". In Adv. Papers - 3rd International Joint Conference on Artificial Intelligence, 1973, pp. 130-138.

22. Michael Ballantyne and William Bennett. "Graphing Methods for Topological Proofs". University of Texas at Austin, Math. Dept. Memo ATP 7, 1973.

23. Alan Bundy. Obituary Column in A.I.S.B. European Newsletter, No. 18, November, 1974.

24. F. Malloy Brown. "The Role of Extensible Deductive Systems in Mathematical Reasoning". Proceedings of the A.I.S.B. Conference on Artificial Intelligence, Edinburgh, July 1976.

25. A. Newell, J.C. Shaw and H. A. Simon. "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics". in Computers and Thought, ed. E. A. Feigenbaum & J.Feldman, 1963.

26. L. Siklossy, A. Rich. "The Logic Theorist Revisited, or a Defence of the British Museum Algorithm". Unpublished paper, Dept. of Computer Science, University of Texas.

27. M. E. Szabo (ed.). The Collected Papers of Gerhard Gentzen, North Holland Publishing Company, Amsterdam, 1969.

28. F. Malloy Brown, Sten-Ake Tarnlund. "Inductive Reasoning in Mathematics". DAI Research Report 30, 1977. Also to appear in the Proceedings of the 5th International Joint Conference on Artificial Intelligence, 1977.

29. F. Malloy Brown. "The Theory of Meaning", DAI Research Report 35, 1977.

Appendix 1: Consistency of our Sequent Logic

We will prove that the following rules of our sequent logic are consistent: $\rightarrow\wedge$, $\rightarrow\forall$, $\rightarrow\sim$, $\rightarrow\forall$, $\rightarrow\exists$, $\rightarrow$Unify. Proofs of the consistency of the remaining rules may be obtained analogously.

These proofs will be carried out in the second order logic that Frege used in Begriftschrift. In particular the skolem functions: f used in our logic will be represented as universally quantified function faviables $\forall f$. In these proofs:

(1) $\forall \underline{f}$ represents a sequence of universal quantifiers: $\forall f_1 \ldots \forall f_n$.

(2) $\exists^* \underline{}$ represents a sequence of existential quantifiers: $\exists^*_1 \ldots \exists^*_n$.

(3) S represents a conjunction of sequents: $S_1 \wedge \ldots \wedge S_n$.

(4) ϕ when it occurs within a sequent represents a sequence of formulas $\phi_1, \ldots, \phi_n$ otherwise it represents a conjunction of formulas.

(5) ψ when it occurs within a sequent represents a sequence of formulas, otherwise it represents a disjunction of formulas.

Recall that a sequent $(\phi \rightarrow \psi)$ or rather: $(\phi_1, \ldots, \phi_n \rightarrow \psi_1 \ldots, \psi_n)$ is interpreted as:

$(\phi \supset \psi)$ or rather: $(\phi_1 \wedge \ldots \wedge \phi_n \rightarrow \psi_1 \vee \ldots \vee \psi_n)$

We will now prove the consistency of each of the above six rules in succession:

$\rightarrow\wedge$ $\forall \underline{f}\ \exists^* \underline{}(S \wedge (\phi \rightarrow \psi, (A \wedge B)))$: def of sequent

$\forall \underline{f}\ \exists^* \underline{}(S \wedge (\phi \supset \psi \vee (A \wedge B)))$: logic

$\forall \underline{f}\ \exists^* \underline{}(S \wedge \phi \supset (\psi \vee (A \wedge B)))$: logic

$\forall \underline{f}\ \exists^* \underline{}(S \wedge \phi \supset \psi \vee (\phi \supset (\psi \vee B)))$: def of sequent

$\forall \underline{f}\ \exists^* \underline{}(S \wedge (\phi \rightarrow \psi, B))$

$\rightarrow\vee$ $\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \rightarrow \psi, (A \vee B)))$: def of sequent

$\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \supset \psi \vee A \vee B))$: def of sequent

$\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \rightarrow \Psi, A, B))$

$\rightarrow\sim$ $\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \rightarrow \psi, \sim A))$: def of sequent

$\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \supset (\psi \vee \sim A)))$: logic

$\forall \underline{f}\ \exists^*(\underline{S} \wedge ((\phi \wedge A) \supset \psi))$: def of sequent

$\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi, A \rightarrow \psi))$

$\rightarrow\forall$ $\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \rightarrow \psi, \forall x Px))$: def of sequent

$\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \supset (\psi \vee (VxPx))))$: renaming x by a gensymed variable a

$\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \supset (\psi \vee (VaPa))))$: since a does not occur in ψ

$\forall \underline{f}\ \exists^*(\underline{S} \wedge (\phi \supset \forall a(\psi \vee Pa)))$: since a does not occur in ϕ

$\forall \underline{f}\ \exists^*(\underline{S} \wedge \forall a(\phi \supset (\psi \vee Pa)))$: since a does not occur in $\underline{S}$

$\forall \underline{f}\ \exists^*\ \forall a(\underline{S} \wedge (\phi \supset (\psi \vee Pa))$: def of sequent

$\forall \underline{f}\ \exists^*\ \forall a(\underline{S} \wedge (\phi \rightarrow \psi, Pa))$: def of $\exists^*$

$\forall \underline{f}\ \exists^*_1 \ldots \exists^*_n\ \forall a(\underline{S} \wedge (\phi \rightarrow \psi, Pa))$: second order logic

$\forall \underline{f}\ \exists^*_1 \ldots \exists^*_{n-1}\ \forall a\ \exists^*_n(\underline{S} \wedge (\phi \rightarrow \psi, P(a^*_1)))$: ⋮

$\forall \underline{f}\ \exists^*_1\ \forall a\ \exists^*_2 \ldots \exists^*_n(\underline{S} \wedge (\phi \rightarrow \psi\ P(a^*_1 \ldots {}^*_{n-1})))$: second order logic

$\forall \underline{f}\ \forall a\ \exists^*_1 \ldots \exists^*_n(\underline{S} \wedge (\phi \rightarrow \psi, P(a^*_1 \ldots {}^*_n)))$: def of $\exists^*$

$\forall \underline{f}\ \forall a\ \exists^*(\underline{S} \wedge (\phi \rightarrow \psi, P(a^*_1 \ldots {}^*_n)))$

The law of second order logic that is used n times in this proof is:

$$(\exists^*\ \forall a(\Pi\ a\ *)) \leftrightarrow \forall a\ \exists^*(\Pi(a\ *)*)$$

Note that $\forall \underline{f}\ \forall a$ is a sequence of universally quantified variables, and hence is essentially of the same form as $\forall \underline{f}$.

$\rightarrow \exists$	$\forall \underline{f}\ \exists^{*}(\underline{S} \wedge (\phi \rightarrow \psi, (\exists x Px)))$	: def of sequent
	$\forall \underline{f}\ \exists^{*}(\underline{S} \wedge (\phi \supset (\psi \vee (\exists x Px))))$	: logic
	$\forall \underline{f}\ \exists^{*}(\underline{S} \wedge (\phi \supset (\psi \vee (\exists x Px) \vee (\exists x Px))))$	: renaming the second x to a gensyned variable x
	$\forall \underline{f}\ \exists^{*}(\underline{S} \wedge (\phi \supset (\psi \vee (\exists x Px) \vee (\exists \alpha P\alpha))))$	: since α does not occur in Px
	$\forall \underline{f}\ \exists^{*}(\underline{S} \wedge (\phi \supset (\psi \vee \exists \alpha((\exists x Px) \vee P\alpha))))$	: since α does not occur in ψ
	$\forall \underline{f}\ \exists^{*}(\underline{S} \wedge (\phi \supset \exists \alpha(\psi \vee (\exists x Px) \vee P\alpha)))$	: since α does not occur in ϕ
	$\forall \underline{f}\ \exists^{*}(\underline{S} \wedge \exists \alpha(\phi \supset \psi \vee \exists x\ Px \vee P\alpha))$	: since α does not occur in $\underline{S}$
	$\forall \underline{f}\ \exists^{*}\ \exists \alpha(\underline{S} \wedge (\phi \supset (\psi \vee \exists x\ Px \vee P\alpha)))$	: def of sequent
	$\forall \underline{f}\ \exists^{*}\ \exists \alpha(\underline{S} \wedge (\phi \rightarrow \psi, \exists x Px, P\alpha)))$	

Note that $\exists^{*}\ \exists \alpha$ is a sequence of existentially quantified variables and hence essentially of the form of $\exists^{*}$.

Unify We let $\exists^{*'}$ be the sequence of existential quantifiers whose variables are instantiated by the substitution $\theta = [*' \leftarrow \underline{f}]$ of the unification rule. We let $\exists^{*}$ be the quantifiers for any other unification variables.

We let $\underline{T}$ be the sequents which are made tautologous by the substitution θ and let $\underline{S}$ be any other sequents.

$\forall \underline{f}\ \exists^{*}\ \exists^{*'} \underline{S} \wedge \underline{T}$	: is implied by:
$\forall \underline{f}\ \exists^{*}(\underline{S} \wedge \underline{T})\ [*' \leftarrow \underline{f}]$	: substitution
$\forall \underline{f}\ \exists^{*}(\underline{S}[*' \leftarrow \underline{f}]) \wedge (\underline{T}[*' \leftarrow \underline{f}])$	: $\underline{T}$ is made tautologous by θ
$\forall \underline{f}\ \exists^{*}(\underline{S}[*' \leftarrow \underline{f}]) \wedge$ ■	
$\forall \underline{f}\ \exists^{*}\ \underline{S}[*' \leftarrow \underline{f}]$	

One final point is that we define the substitution t^{*}_{1} for $*_{o}$ into a skolem function f^{*}_{o} as f^{*}_{1}. This is clearly consistent since $\forall f\ \forall t(\phi(f(t^{*}_{1}))(t^{*}_{1}))$ is implied by $\forall f\ \forall t(\phi(f^{*}_{o})(t^{*}_{o}))$.

Note that every occurence of f in $\forall f \forall t(\phi(f(t^{*}_{1}))(t^{*}_{o}))$ if required to contain a term equal to t^{*}_{o} as its argument.